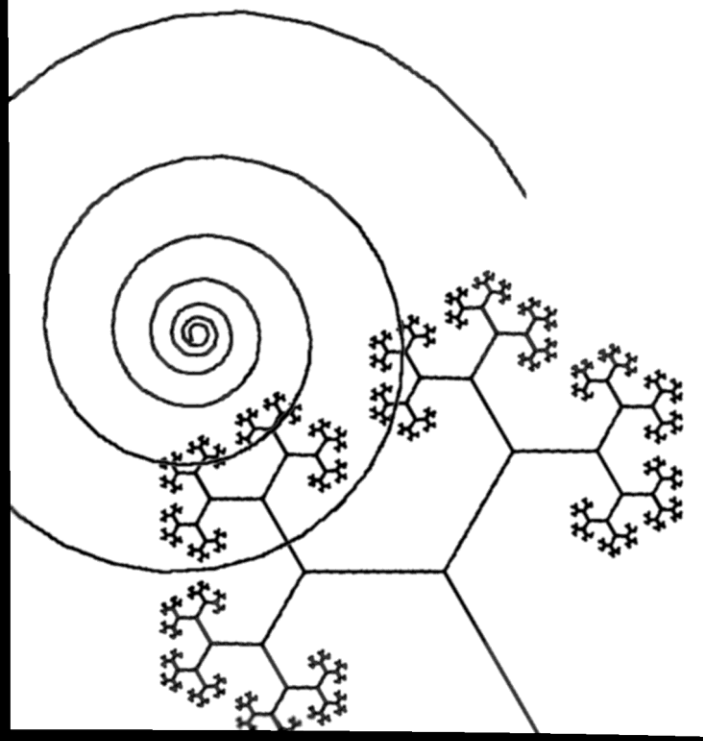
Java



Atli Harðarson

Kennslubók í forritun fyrir framhaldsskóla

2. útgáfa 2001

Atli Harðarson

Java
Kennslubók í forritun
fyrir framhaldsskóla
2. útgáfa

Iðnú 2001

Java - Kennslubók í forritun fyrir framhaldsskóla, 2. útgáfa

Iðnmennt/Iðnú

Efnisyfirlit

Inngangur

Þessi bók er til þess gerð að styðja byrjendur í forritun fyrstu skrefin. Hún hentar bæði til sjálfsnáms og sem kennsluefni fyrir forritunaráfangana TÖL 103 og TÖL 203 eins og þeim er lýst í nýlegri námskrá fyrir framhaldsskóla (Menntamálaráðuneytið 2000).

Bókinni er ekki ætlað að gera forritunarmálinu Java tæmandi skil. Hún hefur þann eina tilgang að þjálfa byrjendur í undirstöðuatriðum forritunar.

Kaflar 1 til 9 eru kappnóg efni fyrir TÖL 103. Vinnist ekki tími til að fara yfir allt þetta efni er hægt að sleppa köflum 6.d til 7.b og öllum 8. kafla eða fara mjög lauslega yfir þá án þess að það komi niður á framhaldinu. Kaflar A til 12 henta til kennslu í TÖL 203. Sé ekki tími til að fara yfir allt efnið er hægt að sleppa köflum B og C án þess að það komi niður á framhaldinu. Einnig er hægt að fara lauslega yfir kafla E og sleppa F.b til F.d og öftustu tveim köflunum.

Öll forrit í bókinni liggja frammi á heimasíðu minni (http://www.ismennt.is-/not/atli/). Nokkur þessara forrita eru lengri og viðameiri en algengt er í kennslubók-um fyrir byrjendur. Mjög mikilvægt er að nemendur skoði þessi forrit vel, prófi að keyra þau og vinna með þau jafnóðum og þeir lesa textann.

Í hverjum kafla eru nokkur verkefni. Sum eru merkt með *. Þau ættu allir nemend-ur að leysa. Á eftir hverjum kafla eru spurningar og á eftir sumum þeirra eru líka efni til umhugsunar. Mikilvægt er að nemendur lesi kaflana nógu vel til að þeir geti svarað spurningunum. Umhugsunarefnin gefa vonandi tilefni til að ræða ýmis undirstöðu-atriði tölvu- og hugbúnaðarfræða.

Við samningu efnisins hef ég haldið mig við Java 1.0. Þetta kemur lítt að sök fyrir þá sem vilja fremur nota Java 1.1 eða enn nýrri útgáfur því lítil áhersla er lögð á atriði þar sem mismunur á nýrri gerðunum og þeirri eldri skiptir máli. Um muninn á þessum útgáfum er fjallað í viðauka IV.

Öll forrit í heftinu eru unnin og kembd með *Visual J++* 1.1 frá Microsoft. Applet hafa verið prufukeyrð í *Microsoft Internet Explorer* 4.0 og *Appletviewer* sem fylgir *JDK* (*Java Development Kit*) 1.1.6 frá Sun Microsystems.

Heppilegt er að nemendur noti gagnvirkt forritunarumhverfi fremur en *JDK* frá Sun Microsystems eitt og sér. Strax í fyrsta kafla er byrjað að smíða forrit sem keyra innan í vefsjá (Applet) án þess það sé skýrt hvernig skuli þýða Java-kóða og búa til vefsíður sem ræsa forritin. Þetta hentar nemendum vel ef þeir nota gagnvirkt forrit-unarumhverfi sem annast þessa hluti.

Þau verkfæri sem ég hef reynslu af að nota og get mælt með eru þessi helst:

- *Visual J++* frá Microsoft. Það má reyndar efast um að svo viðamikið forritunar-umhverfi henti algerum byrjendum. Einnig er rétt að vara við því að *Visual J++* miðar fyrst og fremst við Windows. Ýmsir kostir þess nýtast ekki ef það á að keyra forritin í öðru stýrikerfi. Segja má að þetta umhverfi sé andstætt þeirri stefnu sem forritarar hjá Sun Microsystems mörkuðu, þegar þeir bjuggu til forritunarmálið Java, að forrit sem skrifuð eru í því skyldu jafnvíg á ólík stýrikerfi. Upplýsingar um *Visual J++* er að finna á vefsíðum Microsoft (http://www.microsoft.com/).
- *Kawa*. Þetta umhverfi er raunar skel yfir *JDK* (*Java Development Kit*) frá Sun Microsystems og fylgir því fullkomlega þeim stöðlum og viðmiðum sem höfundar Java hafa ákveðið. Þetta er einfalt umhverfi sem hentar byrjendum. Vísað er í vefsíður um Kawa af heimsíðu minni (http://www.ismennt.is/not/atli).

Með nýjustu útgáfum af *Visual J++* fylgja tæki sem skrifa kóða fyrir notendaskil með sjálfvirkum hætti svo forritari þarf aðeins að draga skjámyndirnar upp með músinni.

Ýmis önnur tæki fyrir Java forritara, eins og *JBuilder* frá Borland, *Visual Age* frá IBM og *Visual Café* frá Symantek, bjóða upp á sjálfvirka framleiðslu á kóða fyrir notendaskil svipaða þeirri sem notendur *Visual BASIC* og *Delphi* þekkja. Þessir möguleikar henta þeim sem kunna Java og vilja nýta sjálfvirknina til að flýta smíði forrita. Að mínu viti eru þeir ekki heppilegir fyrir byrjendur í forritun. Meðan nemendur eru að stíga fyrstu sporin ættu þeir að skrifa allan kóða sjálfir, jafnvel þótt það þýði að forritin verði ekki eins falleg á skjánum og skjámyndirnar sem verða til með sjálfvirkum hætti þegar fyrrgreind verkfæri eru notuð. Tilvera þessara verkfæra réð þó miklu um það að lítil áhersla er lögð á forritun notendaskila í þessu kveri. Í sýnidæmum og verkefnum eru þau höfð eins einföld og mest getur verið.

Leiðbeiningar um notkun hugbúnaðar á borð við *Visual J++* og *Kawa* er ekki að finna í þessu kveri enda er textanum ætlað að nýtast hvort sem menn nota þessi verkfæri eða einhver önnur. Í viðauka III. er örstutt kynning á *JDK*.

*

Fyrstu drög þessarar bókar voru skrifuð sem glósur handa nemendum við Fjölbrautaskóla Vesturlands á Akranesi. Þegar ég afréð að auka við glósurnar og gera úr þeim bók hlaut ég til þess styrk frá menntamálaráðuneytinu. Fyrir það ber að þakka. Einnig þakka ég þeim mörgu sem sendu mér ráð, ábendingar og hvatningarorð meðan textinn var í smíðum og drög að honum lágu frammi á heimasíðu minni.

Í fyrstu útgáfu þessarar bókar voru allmargar villur. Í þessari útgáfu hafa þær verið leiðréttar og orðalag fært til betri vegar á stöku stað. Að öðru leyti hafa litlar breytingar verið gerðar á textanum.

Október 2001
Atli Harðarson
atli@ismennt.is
http://www.ismennt.is/not/atli/

Til nemanda

Öll forrit í þessari bók eru í einni pakkaðri skrá (zip-skrá) á heimasíðu minni (http://www.ismennt.is/not/atli/) á bak við krækju sem merkt er *Java kennslubók*. Þessi skrá heitir javasafn.zip. Þú skalt sækja hana og pakka henni upp áður en þú byrjar á bókinni. Það er nær engin leið að læra efni hennar án þess að keyra forritsdæmin í textanum jafnóðum og hann er lesinn. Þegar javasafn.zip er pakkað upp verður til ein efnisskrá fyrir hvert forrit.

Í textanum ægir saman íslensku og glefsum á Java. Úr verður hálfgert hrognamál á köflum. Til að koma í veg fyrir rugl og misskilning eru orð úr Java rituð með `courier` letri.

1. kafli: Fyrstu skrefin

1.a. Fyrsta forritið

Líttu á þetta forrit. Númerin lengst til vinstri eru ekki hluti af forritinu heldur aðeins sett þarna svo auðveldara sé að vísa á einstakar línur í því.

```
1.    import java.applet.Applet;
2.    import java.awt.*;
3.    //
4.    // forrit_01_01
5.    // FyrstaTilraun
6.    //
7.    //
8.    public class FyrstaTilraun extends Applet
9.    {
10.      TextField t;
11.
12.      public void init()
13.      {
14.         t = new TextField(20);
15.         this.add(t);
16.         t.setText("Hæ, hér er ég.");
17.      }
18.
19.   }
```

Verkefni 1.1 *

1. Sláðu forritið hér að ofan inn, láttu Java þýðanda þýða það og keyrðu það svo. Aðferðirnar til að gera þetta eru ólíkar eftir því hvaða verkfæri eru notuð. Aflaðu þér upplýsinga um hvernig á að gera þetta í því umhverfi sem þú ætlar að nota.

Skoðum forritið nú línu fyrir línu.

Fyrstu tvær línurnar „flytja inn“ þá klasa sem notaðir verða. Sú fyrsta „flytur inn“ klasann `Applet` sem er í pakkanum `java.applet`, sú næsta „flytur inn“ alla klasa í pakkanum `java.awt`. (* merkir allt.)

Línur númer 3 til 7 eru ekki hluti af forritinu. Allt sem er fyrir aftan // eru athugasemdir sem þýðandinn hleypur yfir þegar hann þýðir forritið. Línur númer 11 og 18 hafa heldur engin áhrif. Þær eru hafðar til þess eins að forritið sé læsilegra.

Lína númer 8 tilkynnir að á eftir komi skilgreining á klasa (`class`) sem heitir `FyrstaTilraun` og erfir frá (`extends`) klasanum `Applet`. Línan byrjar á orðinu `public` sem þýðir að klasinn sé allra gagn, allir aðrir klasar megi nota hann.

Það sem er milli slaufusviganna, þ.e. milli línu 9 og 19, segir hvernig klasinn `FyrstaTilraun` hagar sér. Öll Javaforrit eru mynduð úr klösum sem eru byggðar svona: Með titillínu sem segir hvað klasinn heitir, hvaðan hann erfir og hvort hann er `public`. Skilgreiningin á því hvað klasinn gerir kemur svo fyrir neðan innan slaufusviga.

`Klasinn FyrstaTilraun` erfir `Applet`. Þetta þýðir að hann kann allt sem `Applet` kann. Við segjum að hann erfi alla eiginleika og aðferðir `Applet`. Klasar sem erfa frá `Applet` kallast einu nafni `Applet`. Þeir eru tegundir af `Applet`-um á svipaðan hátt og

kettir, hundar og hestar eru tegundir af spendýrum eða bílar, mótorhjól og hestvagnar eru tegundir af farartækjum. Við tölum því um að forritið `FyrstaTilraun` sé `Applet` alveg eins og talað er um að hundar séu spendýr eða mótorhjól séu farartæki.

Klasi í Java er í rauninni skilgreining á tegund eða flokki hluta. (Um þetta verður fjallað nánar í 5. kafla.) Héðan af verða orðin „klasi“ og „tegund“ notuð jöfnum höndum yfir það sem kallast „class“ í Java. Forrit sem samin eru á Java eru mynduð úr klösum sem sumir fylgja með málinu (eins og t.d. TextField) og sumir eru búnir til frá grunni og sumir er byggðir ofan á klasa sem fylgja málinu (eins og klasinn FyrstaTilraun er byggður á klasanum Applet). Öll forrit í fyrstu fjórum köflum þessarar bókar eru gerð með því að skrifa einn klasa. Í 5. kafla verður fjallað um stærri forrit sem eru mynduð úr mörgum klösum.

Við sjáum ekki hvernig klasinn `Applet` er skilgreindur. Okkur dugar að vita að `Applet` kann að keyra innan í vefsjá og þennan hæfileika erfir `FyrstaTilraun`. Þegar vefsjáin framkvæmir forritið býr hún til einn hlut af tegundinni `FyrstaTilraun` og lætur hann byrja á að framkvæma aðferðina `init`. Hér er þessi aðferð samsett úr þrem skipunum sem eru í línum 14 til 16.

Lína 10 skilgreinir breytuna `t` og segir að hún skuli innihalda hlut af tegundinni `TextField`. Þessi tegund/klasi er í pakkanum `java.awt` og var því „flutt inn“ í 2. línu.

Línur númer 12 til 17 eru skilgreiningar á aðferðinni `init` en aðferð með því nafni er ævinlega framkvæmd fyrst þegar Java forrit er keyrt innan í vefsjá.

12. lína er haus aðferðarinnar. Þar segir að hún sé allra gagn (`public`) og `void` (þ.e. skili ekki neinni útkomu). Sumar aðferðir eins og t.d. reikniaðferðir á borð við kvaðratrót skila útkomu en `init` gerir það ekki. Aftan við nöfn aðferða eru ævinlega svigar og af þessu má þekkja þau frá nöfnum á breytum og klösum. Innan slaufusviganna, þ.e. á milli lína 13 og 17, eru svo þrjár skipanir sem segja hvað `init` á að gera.

Lína 14 segir að í breytuna `t` skuli setja nýjan hlut og hann skuli vera `TextField(20)` þ.e. textareitur af breiddinni 20. Hlutir eru ævinlega búnir til með því að skrifa `new` og heiti klasa þar fyrir aftan.[1]

Lína 15 segir að `this` (þessi hlutur sem er af tegundinni sem verið er að skilgreina þ.e. tegundinni `FyrstaTilraun`) skuli framkvæma á sér aðferðina `add` og senda henni hlutinn `t`. Þetta lætur hann bæta `t` á skjámynd sína.

16. línan lætur hlutinn `t` framkvæma á sér aðferðina `setText` og senda henni strenginn (þ.e. stafarununa) "Hæ, hér er ég."

Taktu eftir hvernig hlutir framkvæma aðferðirnar sem þeir kunna. Hlutur af tegundinni `textField` kann margar aðferðir. Ein þeirra er `setText` og hann framkvæmir hana með því að setja nafn hennar á eftir sínu með punkti á milli.

Taktu líka eftir því að sérhver skipun endar á semíkommu. Í Java gegnir semíkomma svipuðu hlutverki og punktur í íslensku. Hún táknar lok málsgreinar.

Áttaðu þig á hvernig klasinn `TextField` er notaður. Fyrst er hann „fluttur inn“ með

[1] Þetta er strangt tekið ekki alveg rétt. Fyrir aftan `new` kemur heiti aðferðar til að skapa nýjan hlut af einhverri tegund en slík aðferð kallast smiður og heitir alltaf sama nafni og klasinn en með svigum fyrir aftan og stundum eitthvað á milli þeirra. Um þetta verður fjallað nánar í 5. kafla.

```
import java.awt.*;
```

Svo er skilgreind ein breyta af tegundinni `TextField` með

```
TextField t;
```

Inni í aðferðinni `init` er búinn til nýr hlutur af tegundinni `TextField` og hann settur í breytuna `t`. Síðan er þessum hlut bætt á skjámyndina með aðferðinni `add` og hann látinn setja í sig texta með aðferðinni `setText`.

Það væri hægt að nota `TextField` þótt `import` skipuninni væri sleppt, en þá þyrfti að sækja klasann „alla leið“ í hvert sinn sem hann er nefndur og skrifa:

```
java.awt.TextField t
```

og

```
t = new java.awt.TextField(20);
```

1.b. Takkar, merki o.fl.

Í pakkanum `java.awt` er ýmislegt fleira en `TextField`. Þar eru líka takkar, merki, valmyndir og ýmislegt fleira sem birtist á skjánum þegar forrit eru keyrð.

Eftirfarandi forrit inniheldur tvo textareiti (`TextField`), einn takka (`Button`) og einn merkimiða (`Label`). Allar þessir klasar eru í pakkanum `java.awt` og því „fluttir inn“ með

```
import java.awt.*;
```

Það mætti „flytja inn“ einn og einn í einu með því að skrifa

```
import java.awt.TextField;
import java.awt.Button;
import java.awt.Label;
```

En hér er það ekki gert.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_01_02
// OnnurTilraun
//
public class OnnurTilraun extends Applet
{
   TextField t1, t2;
   Button b1;
   Label a1;

   public void init()
   {
      t1 = new TextField(20);
      t2 = new TextField(20);
      b1 = new Button("Heilsa");
      a1 = new Label("Nafn:");
      this.add(a1);
      this.add(t1);
      this.add(b1);
      this.add(t2);
   }
}
```

Þegar þetta er framkvæmt birtast tveir reitir fyrir texta, einn merkimiði og einn takki. En það gerist ekkert þegar smellt er á takkann.

Java klasar sem erfa frá `Applet` fylgjast með atburðum sem verða meðan þeir eru keyrðir og framkvæma aðferðir til að bregðast við þeim. Atburður er til dæmis þegar ýtt er á hnapp á lyklaborði, smellt á takka með músinni eða músin dregin til. `Applet` bregst við þegar smellt er með músinni á hnapp (`Button`) með því að framkvæma aðferð sem hetir `action`.

Taktu eftir að aðferðin `action` er ekki `void` heldur skilar hún útkomu af tegundinni `boolean` (sem þýðir að út úr henni kemur annað hvort gildið `true` eða `false`. Um það lærir þú meira seinna.)

Skipunin

```
if (e.target == b1)
{
   t2.setText("Góðan dag " + t1.getText());
   return true;
}
```

þýðir að ef hluturinn sem fyrir atburðinum varð er `b1` þá skuli framkvæma skipanirnar innan slaufusviganna. Fyrri skipunin lætur textann í `t2` verða „Góðan dag" plús textinn sem er í `t1`. Aðferðin `getText` skilar útkomu sem er textinn sem hluturinn inniheldur.

„Sama og" eða „sama sem" er táknað með tveim jafnaðarmerkjum í röð, eitt jafnaðarmerki er lesið „skal verða" og er notað til að gefa breytu gildi eins og í

```
t1 = new TextField(20);
```

Hér kemur forritið fullgert með `action`-aðferð til viðbótar við aðferðina `init`. Sú síðarnefnda er framkvæmd um leið og forritið fer af stað, sú fyrrnefnda í hvert sinn sem smellt er á hnapp með músinni.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_01_02
// OnnurTilraun
//
public class OnnurTilraun extends Applet
{
   TextField t1, t2;
   Button b1;
   Label a1;

   public void init()
   {
      t1 = new TextField(20);
      t2 = new TextField(20);
      b1 = new Button("Heilsa");
      a1 = new Label("Nafn:");

      this.add(a1);
      this.add(t1);
      this.add(b1);
      this.add(t2);
   }

   public boolean action(Event e, Object o)
   {
```

```
      if (e.target == b1)
      {
         t2.setText("Góðan dag " + t1.getText());
         return true;
      }
      return false;
   }
}
```

Verkefni 1.2 *
Búðu til forrit sem er eins og forritið hér að ofan nema með tveim tökkum, einum til að heilsa karli og einum til að heilsa konu. Sé skrifað „Siggi" í fyrri textareitinn og slegið á annan takkann á forritið að skrifa „Sæll Siggi" í seinni reitinn en „Sæl Siggi" ef slegið er á hinn takann.

Verkefni 1.3 *
Búðu til forrit sem hefur þrjá textareiti og einn takka og hagar sér þannig að sé slegið á takkann þá sé innihald fyrstu tveggja reitanna skrifað í þann þriðja með „og" á milli. Sé t.d. „Jón" í fyrsta reitnum og „Gunna" í öðrum þá á „Jón og Gunna" að birtast í þeim þriðja þegar slegið er á takkann.

1.c. `init`, `action`, `paint` og teikning

Um leið og `Applet` (þ.e. Java forrit sem keyrt er innan í vefsjá) fer af stað framkvæmir það aðferð sem heitir `init`. Í hvert sinn sem smellt er á hnapp (`Button`) framkvæmir það aðferð sem heitir `action`. Þessar tvær aðferðir voru kynntar í forrit_01_02. Nú verður þriðja aðferðin kynnt til sögunnar. Hún heitir `paint`. Skipanirnar í henni eru framkvæmdar um leið og `Applet` er teiknað á skjáinn (þ.e. næstum strax og lokið er við `init`-aðferðina).

Aðferðin `action` fær senda tvo hluti. Í forrit_01_02 heitir sá fyrrnefndi `e` og er af tegundinni `Event`. Sá seinni heitir `o` og er af tegundinni `Object`. Þessir hlutir geyma upplýsingar um atburð (eins og t.d. hvaða hlut var smellt á með músinni). Það er ekki nein skylda að láta þessa hluti heita `e` og `o`, en þeir verða að vera af tegundunum `Event` og `Object`. Það væri t.d. í lagi að hafa aðferðina svona og láta hlutina heita `x` og `y`

```
public boolean action(Event x, Object y)
{
   if (x.target == b1)
   {
      t2.setText("Góðan dag " + t1.getText());
      return true;
   }
   return false;
}
```

Rétt eins og `action`-aðferðin fær sendan atburð fær `paint`-aðferðin sendan myndflöt, þ.e. hlut af tegundinni `Graphics`, sem hægt er að teikna á. Þessi hlutur er bakgrunnurinn sem sést á skjánum og tökkum og textareitum er raðað á.

Textareitir kunna aðferðir til að birta texta (`setText`) og sækja texta sem skrifaður hefur verið í þá (`getText`). Myndfletir (`Graphics`) kunna aðferðir til að teikna á sig

línur, ferhyrninga og fleiri form. Forrit_01_03 teiknar tvær línur og þrjá ferhyrninga, þar af tvo fyllta (sem eru teiknaðir með `fillRect`-aðferðinni).

Síðasta skipunin lætur myndflötinn skrifa á sig „Glæsilegt einbýlishús“. Tölurnar 50 og 180 þýða að upphaf textans eigi að hafa xhnit 50 og yhnit 180.

Myndflöturinn sem teiknað er á er eins og hnitakerfi þar sem punkturinn (0, 0) er efst til vinstri og punkturinn (60, 70) er 60 punktum til hægri frá vinstri jaðri og 70 punktum neðan við efri brún.

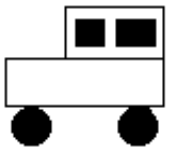
Aðferðirnar `drawRect` og `fillRect` taka við fjórum tölum. Þær fyrstu tvær segja hvar hornið efst til vinstri á að vera. Sú þriðja segir hvað ferhyrningurinn á að vera breiður og sú síðasta hvað hann á að vera hár.

`drawLine` tekur líka við fjórum tölum. Þær fyrstu tvær eru x og y hnit annars enda striksins og þær seinni tvær eru x og y hnit hins endans.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_01_03
// TridjaTilraun
//
public class TridjaTilraun extends Applet
{
   public void paint(Graphics g)
   {
      g.drawRect(50, 100, 90, 60);
      g.drawLine(50, 100, 95, 40);
      g.drawLine(95, 40, 140, 100);
      g.fillRect(70, 120, 20, 20);
      g.fillRect(105, 120, 20, 40);
      g.drawString("Glæsilegt einbýlishús", 50, 180);
   }
}
```

Verkefni 1.4 *
Búðu til forrit sem teiknar þríhyrning úr þrem strikum.

Verkefni 1.5
Búðu til forrit sem teiknar kassabíl eins og hér er til hægri.



Til að leysa þetta verkefni þarftu að teikna fyllta hringi (hjólin). Til þess er hægt að nota aðferðina `fillArc` sem tilheyrir tegundinni `Graphics`. Notaðu hjálpartextann sem fylgir Java-umhverfinu sem þú notar til að komast að því hvernig `fillArc`-aðferðin er notuð.

1.d. Nöfn

Taktu eftir að heiti klasanna `FyrstaTilraun`, `OnnurTilraun` og `TridjaTilraun` eru rituð með stórum upphafsstaf. Í Java er til siðs að rita heiti klasa með stórum upphafsstaf en láta heiti aðferða (eins og `init` og `action`) og nöfn á breytum (eins og `t1`, `t2` og `b1`) byrja á litlum staf.

Gerður er strangur greinarmunur á stórum og litlum staf þannig að `OnnurTilraun` og `onnurTilraun` skoðast sem tvö ólík nöfn.

Nöfn á klösum (eins og `OnnurTilraun`, `TextField` eða `Button`), aðferðum (eins og `action` og `init`) og breytum (eins og `t1`, `t2`, `b1`, `a1`) skulu mynduð úr enskum bókstöfum (A, B, C, D, E … X, Y, Z og a, b, c, d, e … x, y, z), tölustöfum (0, 1, 2, … 9) dollaramerki og striki ($, _). Önnur tákn má ekki nota og fyrsti stafurinn í nafni má ekki vera tölustafur.

Nokkur dæmi um leyfileg nöfn á breytur, aðferðir og klasa:

```
d1
Nr4
nr_4
refurinn_sem_veiddi_andarungann
refurinnSemVeiddiAndarungann
```

Nokkur dæmi um óleyfileg nöfn:

`1d`	Byrjar á tölustaf
`Númer4`	ú er ekki í enska stafrófinu
`nr.4`	Inniheldur punkt
`refurinn-sem-veiddi-andarungann`	Inniheldur mínusmerki
`refurinnSemVeiddiÖndina`	Ö er ekki í enska stafrófinu

1.x Spurningar og umhugsunarefni

1. Hvað merkir línan hér fyrir neðan?

   ```
   import java.awt.*;
   ```

2. Hvað merkja orðin `class` og `extends` í línunni hér fyrir neðan?

   ```
   public class FyrstaTilraun extends Applet
   ```

3. Hvaða mikilvæga eiginleika hafa klasa sem erfa frá `Applet`?
4. Hvað heitir aðferðin sem er byrjað á að framkvæma þegar Java forrit er keyrt í vefsjá?
5. Hvað merkja þessar skipanir?

   ```
   TextField t;
   t = new TextField(20);
   this.add(t);
   ```

6. Hvað merkir orðið `void` í línunni hér fyrir neðan?

   ```
   public void init()
   ```

7. Hvaða hlutverk hefur táknið **;** (semíkomma)?
8. Hvað eru `TextField`, `Button` og `Label` og hvaða pakka tilheyra þessar tegundir?
9. Hvað merkja táknin `=` og `==` ?
10. Hvenær framkvæma `Applet` aðferðirnar `init`, `paint` og `action`?
11. Hvaða tegund getur beitt aðferðunum `drawLine` og `drawRect`?
12. Hvaða hnit hefur punktur sem er 100 punktum til hægri við vinstri brún myndflatar og 60 punktum neðan við efri jaðar hans?
13. Hver þessara orða er leyfilegt að nota fyrir nöfn á breytur, aðferðir og tegundir?

    ```
    uxi   öxarViðÁna   Akrafjall   x   1.mars   1_mars   mars_1   x2y
    ```

14. Hverju er vani að gefa nafn sem byrjar á stórum staf og hvað er vani að nefna nöfnum sem byrja á litlum staf?

2. kafli: Tölur og einfaldar tegundir

2.a. `double` og `Double`

Eftirfarandi forrit hefur þrjá reiti fyrir texta og einn takka sem er merktur +. Séu settar tölur í tvo fyrstu reitina og ýtt á takkann þá leggur forritið tölurnar saman og setur útkomuna í þriðja reitinn.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_02_01
// Samlagning
//
public class Samlagning extends Applet
{
   TextField t1, t2, tsvar;
   Button bplus;
   Label a1, a2;
   Double D1, D2;
   double d1, d2, dsvar;

   public void init()
   {
      t1 = new TextField(3);
      t2 = new TextField(3);
      tsvar = new TextField(15);
      bplus = new Button("+");
      a1 = new Label("Tala 1");
      a2 = new Label("Tala 2");

      this.add(a1); this.add(t1);
      this.add(a2); this.add(t2);
      this.add(bplus); this.add(tsvar);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bplus)
      {
         D1 = new Double(t1.getText());
         d1 = D1.doubleValue();
         D2 = new Double(t2.getText());
         d2 = D2.doubleValue();
         dsvar = d1 + d2;
         tsvar.setText("Summan er: " + dsvar);
         return true;
      }
      return false;
   }
}
```

Líttu á línurnar

```
   Double D1, D2;
   double d1, d2, dsvar;
```

Sú fyrri skilgreinir tvær breytur (`D1` og `D2`) af tegundinni `Double`. Sú seinni skilgreinir þrjár breytur af tegundinni `double`. Breytur af báðum þessum tegundum geta geymt

tölur eins og til dæmis -5.0, 0.75, 1567.3 eða 4.09e+15. Svona tölur eru kallaðar kommutölur, því þær þurfa ekki endilega að vera heilar. Taktu eftir því að í stað kommu er ritaður punktur eins og gert er í ensku. (4.09e+15 þýðir 4.09×10^{15}.)

Þótt tegundirnar `Double` og `double` geymi báðar samskonar gögn, nefnilega kommutölur eru þær þó ólíkar.

Breyta af tegundinni `Double` geymir hlut sem kann aðferðir og þarf að búa til með skipuninni `new`. Á breytur af þessari gerð er ekki hægt að beita reikniaðgerðum eins og `+, -, *` og `/`.

Breyta af tegundinni `double` er einfaldlega hólf (pláss í minni tölvunnar) sem geymir eina kommutölu og kann engar aðferðir. Á breytur af þessari gerð er hins vegar hægt að beita reikniaðgerðum eins og `+, -, *` og `/`.

Í Java hafa allar tegundir af hlutum nöfn sem byrja á stórum staf en allar einfaldar tegundir (sem ekki kunna neinar aðferðir) hafa nöfn sem byrja á litlum staf.

Skoðaðu nú línurnar

```
D1 = new Double(t1.getText());
d1 = D1.doubleValue();
D2 = new Double(t2.getText());
d2 = D2.doubleValue();
dsvar = d1 + d2;
tsvar.setText("Summan er: " + dsvar);
```

Sú fyrsta býr til nýjan hlut af tegundinni `Double` og setur hann í breytuna `D1`. Hluturinn fær gildi úr `t1.getText()`, þ.e. úr innihaldi reitsins `t1`. Það sem er í þessum reit er ekki tala heldur runa (strengur) úr tölustöfum. Aðferðin `Double` (sem er til að mynda hlut af tegundinni `Double`) hefur vit á að breyta strengnum í tölu af tegundinni `Double`. Það er hins vegar engin aðferð til að breyta streng í `double`. Það er semsagt engin aðferð til að breyta streng eins og "-5.0" í tölu sem hægt er að beita á reikniaðgerðunum `+, -, *` og `/`. En eftir að búið er að mynda hlut af tegundinni `Double` úr slíkum streng er hægt að breyta honum í `double` með því að láta hann beita á sig aðferðinni `doubleValue`. Þetta gerir skipunin sem hér er í annarri línu.

Næst síðasta línan beitir aðgerðinni `+` á `d1` og `d2` og setur útkomuna í `dsvar`.

Verkefni 2.1 *
Búðu til forrit sem tekur við upplýsingum um lengd og breidd ferhyrnings og skrifar flatarmál hans.

Verkefni 2.2 *
Búðu til klasa sem er eins og `Samlagning` nema með fjórum tökkum, einum fyrir samlagningu, einum fyrir frádrátt, einum fyrir margföldun og einum fyrir deilingu.

2.b. Reikningur

Auk aðgerðanna `+`, `-`, `*` og `/` er hægt að reikna kvaðratrót, hornaföll, veldi og margt fleira. Þessar aðferðir eru hluti af klasa sem heitir `Math`. Eigi til dæmis að reikna 5 í 3 veldi og setja útkomuna í breytuna x þarf að skipa

```
x = Math.pow(5, 3);
```

og til að reikna kvaðratrót af 7 má nota

```
x = Math.sqrt(7);
```

Verkefni 2.3 *
Notaðu hjálpina sem fylgir með Java umhverfinu sem þú notar til að finna upplýsingar um allar aðferðir sem tilheyra klasanum `Math`. Finndu líka allar aðferðir sem breytur af tegundunum `Double` og `TextField` geta framkvæmt.

Aðferðin `sin` (sínus) er skilgreind svona:

```
public static double sin(double a)
```

Hún er `public` (allar tegundir mega nota hana), útkoman úr henni er af tegundinni `double` og hún tekur við einu gildi af tegundinni `double`. Þassi aðferð er `static`. Hér er ekki tímabært að skýra hvað það þýðir en þú þarft að vita að sé aðferð static þarf að setja nafn klasans sem hún tilheyrir framan við hana í hvert sinn sem hún er notuð. Aðferðin `sin` tilheyrir klasanum `Math` og til að reikna sínus af 7 dugar ekki að skrifa `sin(7)`, það þarf að skrifa `Math.sin(7)`.

Auk reikniaðferða inniheldur `Math` tvo fasta. Annar er kunnuglegur. hann heitir `PI`. Til að reikna flatarmál hrings með radíus 7 getum við skrifað

```
f = 7*7*Math.PI;
```

Eftirfarandi forrit tekur við upplýsingum um skammhliðar í rétthyrndum þríhyrningi og notar pýþagórasarreglu til að reikna langhliðina.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_02_02
// Pythagoras
//
public class Pythagoras extends Applet
{
   TextField t1, t2, tsvar;
   Button bplus;
   Label a1, a2;
   Double D1, D2;
   double d1, d2, dsvar;

   public void init()
   {
      t1 = new TextField(3);
      t2 = new TextField(3);
      tsvar = new TextField(15);
      bplus = new Button("Reikna langhlið");
      a1 = new Label("Skammhlið 1");
      a2 = new Label("Skammhlið 2");

      this.add(a1); this.add(t1);
```

```
      this.add(a2); this.add(t2);
      this.add(bplus); this.add(tsvar);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bplus)
      {
         D1 = new Double(t1.getText());
         d1 = D1.doubleValue();
         D2 = new Double(t2.getText());
         d2 = D2.doubleValue();
         dsvar = Math.sqrt(Math.pow(d1, 2) + Math.pow(d2, 2));
         tsvar.setText("Langhliðin er: " + dsvar);
         return true;
      }
      return false;
   }
}
```

Hér er klasinn `Math` notaður án þess að hann sé „fluttur inn“ í upphafi forrits. Þetta getur gengið því `Math` er í pakkanum `java.lang` og allt innihald hans er flutt inn óumbeðið. Við getum litið svo á að þýðandinn bæti sjálfkrafa framan við hvern klasa (`class`) sem við skilgreinum skipuninni

```
import java.lang.*
```

Verkefni 2.4 *
Búðu til forrit sem tekur við upplýsingum um radíus kúlu og reiknar yfirborðsflatarmál hennar. (Yfirborð kúlu er $4\pi r^2$)

Verkefni 2.5
Búðu til forrit sem tekur við upplýsingum um lengd tveggja hliða í þríhyrningi og stærð hornsins á milli þeirra í gráðum og reiknar flatarmál þríhyrningsins. Ef hliðarnar heita a og b og hornið C þá er flatarmálið 0.5×a×b×sin(C). Ath. `Math.sin` tekur við horni í radíönum svo ef það er slegið inn í gráðum þarf að breyta því í radíana. Innihaldi C horn í gráðum má breyta því í radíana með

```
C = C * Math.PI / 180.
```

2.c. `int`, `Integer` og `%`

Tegundirnar `double` og `Double` eru til að geyma kommutölur. `int` og `Integer` eru svipaðar en geta aðeins geymt heilar tölur. Á heiltölur af tegundinni `int` er hægt að beita aðgerðunum `+`, `-`, `*` og `/`. Í venjulegum reikningi getur útkoma úr deilingu tveggja heiltalna verið brot en ef `c` er af tegundinni `int` og gefin er skipunin

```
c = 8 / 3
```

þá fær `c` gildið 2 en ekki 2.66666 (því sem er fyrir aftan kommu er einfaldlega hent).

Til eru fleiri reikniaðgerðir en `+`, `-`, `*` og `/`. Ein þeirra er `%` til að finna afgang. Sé gefin skipunin

```
c = 7 % 3
```

fær `c` gildið 1 því ef 3 er deilt í 7 með heiltöludeilingu þá verður einn afgangs. (Ef 7 eplum er skipt milli 3 barna fær hvert barn 2 epli og 1 gengur af.)

Eftirfarandi forrit notar heiltölur og reikniaðgerðina `%` til að segja hvað svo og svo margar sekúndur eru margar mínútur og sekúndur. Sé talan 130 sett í reitinn sem er merktur sekúndur skrifar forritið „130 sek. er 2 mín. og 10 sek.” í hinn reitinn.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_02_03
// Klukka
//
public class Klukka extends Applet
{
   TextField t1, tSvar;
   Button bReikna;
   Label a1;
   Integer I;
   int i, sek, min;

   public void init()
   {
      t1 = new TextField(5);
      tSvar = new TextField(25);
      bReikna = new Button("Reikna mínútur");
      a1 = new Label("Sekúndur");

      this.add(a1);
      this.add(t1);
      this.add(bReikna);
      this.add(tSvar);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bReikna)
      {
         I = new Integer(t1.getText());
         i = I.intValue();
         min = i / 60;
         sek = i % 60;
         tSvar.setText(i + " sek. er " + min + " mín. og " +
                       sek + " sek.");
         return true;
      }
      return false;
   }
}
```

Verkefni 2.6 *
Búðu til forrit sem tekur við upplýsingum um hvað maður er margir sentimetrar á hæð og segir hvað hann er í metrum og sentimetrum. Sé t.d. slegin inn talan 210 á forritið að skrifa „2 m og 10 cm”.

Verkefni 2.7
Breyttu klasanum Klukka þannig að klst. séu með svo ef slegin er inn talan 3800 skrifi forritið „1 klst. 3 mín. og 20 sek."

2.d. Yfirlit yfir einfaldar tegundir og dálítið um breytur

Til eru nokkrar einfaldar tegundir til viðbótar við `int` og `double`. Þær eru allar taldar upp í töflunni hér fyrir neðan.

Heiti tegundar	**möguleg gildi**	**sjálfgefið gildi**
`boolean`	true, false	false
`char`	Allir stafir í Unicode stafatöflunni	\u0000
`byte`	Heilar tölur frá -128 til 127	0
`short`	Heilar tölur frá -32768 til 32767	0
`int`	Heilar tölur frá -2147483648 til 2147483647	0
`long`	Heilar tölur frá -9223372036854775808 til 9223372036854775807	0
`float`	Kommutölur á bilinu ±3.40282347e+38 til ±1.40239846e-45	0.0
`double`	Kommutölur á bilinu ±1.7976931348e+308 til ±4.9406564584e-324	0.0

Java hefur 4 tegundir af heiltölum og tvær af kommutölum. Þetta kann að virðast óþarfi. Hvers vegna dugar ekki að hafa bara `long` og `double`? Í breytum af þessum gerðum er hægt að geyma allar tölur sem hinar gerðirnar ráða við. Svarið er að því stærri tölur sem breyta getur geymt því meira rúm tekur hún í minni tölvunnar og því seinlegra er að vinna með hana. Það er því sóun á tíma og rúmi að búa til margar breytur af tegundinni `long` ef aðeins á að vinna með svo lágar tölur að `byte` dugi.

Allar talnabreytur hafa gildið 0 ef þeim hefur ekki verið gefið neitt annað gildi. Allar breytur sem innihalda hluti hafa hins vegar gildið `null` ef þeim hefur ekki verið gefið neitt gildi. `null` þýðir einfaldlega að breyta vísi ekki á neinn hlut.

Breytur af einföldum tegundum haga sér nokkuð öðru vísi en breytur sem vísa á hluti. Allar breytur eru í raun og veru pláss eða hólf í minni tölvunnar. Ef breyta af einfaldri tegund, t.d. `int`, heitir `x` og inniheldur gildið 7 þá er talan 7 skrifuð í það pláss í minninu sem gengur undir nafninu `x`. Ef breyta sem vísar á hlut, t.d. `TextField` eða `Double`, heitir `x` þá er hluturinn ekki geymdur í þeim hluta minnisins sem kallast `x`. Hann er einhvers staðar annars staðar og `x` geymir vistfang hlutarins, þ.e.a.s. tölu sem segir hvar hann er geymdur.

Ef `x` vísar á `TextField` sem er geymt í hólfi númer 10.000 í minninu þá inniheldur `x` töluna 10.000 en ekki textareit. Ekkert er því til fyrirstöðu að tvær breytur vísi á sama hlutinn. Sé hlutnum sem önnur vísar á breytt þá breytist líka hluturinn sem hin vísar á því þetta eru ekki tveir hlutir heldur einn. Tökum dæmi:

```
int x, y;               // Nú innihalda x og y báðar gildið 0.
x = 7;                  // x fær gildið 7.
y = x;                  // y látið fá sama gildi og x hefur.
x = x + 1;              // Hér fær x gildið 8 en y hefur enn
                        // gildið 7 því það hefur engin áhrif
                        // á y þótt innihaldi x sé breytt.

TextField a, b;         // Nú innihalda a og b báðar gildið null.
```

```
a = new TextField(25);  // Nú inniheldur a staðsetningu (vist-
                        // fang) textareits en b inniheldur
                        // ennþá null.
a.setText("voff");      // Texti settur í TextField-ið sem
                        // breytan a vísar á.
b = a;                  // Nú vísa a og b á sama TextField
                        // (þær geyma sama vistfang) svo
                        // það stendur "voff" í reitnum sem b
                        // vísar á.
a.setText("mjá");       // Nú vísa bæði a og b á reit sem
                        // inniheldur "mjá".
```

Í þessum texta og mörgum öðrum er talað um breytur eins og þær innihaldi gildið sem þær vísa á. Þegar um er að ræða breytur sem vísa á hluti er þetta strangt tekið ekki rétt. Þær benda eða vísa á hlutina með því að innihalda vistfang þeirra. Einfaldar breytur innihalda hins vegar í bókstaflegum skilningi þau gildi sem þeim eru gefin.

Eins og nefnt var í kafla 1.a. er klasi skilgreining á tegund eða flokki af hlutum. Hann inniheldur breytur sem geymt geta upplýsingar um eiginleika hlutanna og aðferðir sem segja hvað hlutirnir geta gert. Tegundir sem fylgja málinu eins og t.d. TextField og Button eru skilgreindar af klösum sem fylgja með Java og eru „fluttir inn“ í forrit með `import` skipunum. Einfaldar tegundir eru hins vegar innbyggðar í málið sjálft en ekki skilgreindar af forritum (klösum) sem fylgja með því. Þær kunna engar aðferðir og geta ekkert gert annað en að geyma eitt gildi, t.d. eina heiltölu.

2.e. `char`, `(char)` o.fl.

Breyta af tegundinni `char` getur geymt einn staf. Ef `s` er breyta af þessari tegund er hægt að gefa henni gildi með skipuninni

```
s = 'R'
```

Eftir þetta geymir `s` bókstafinn R. Taktu eftir að stafurinn er hafður innan einfaldra gæsalappa. (Strengir, þ.e. runur af stöfum eru hins vegar ritaðir innan tvöfaldra gæsalappa.)

Stafirnir í Unicode stafatöflunni, sem Java forrit nota, eru númeraðir. Númerin eru yfirleitt skrifuð sem 4 stafa hexadesímaltala með `\u` fyrir framan. Í stað

```
s = 'R'
```

má því skrifa

```
s = '\u0052'
```

því stafurinn 'R' er númer 82 og talan sem er rituð 82 í tugakerfi er rituð 52 í hexadesímalkerfi. Allir stafir í okkar stafrófi hafa númer undir 16^2 og eru því táknaðir með tölu sem byrjar á 00. Stafir með hærri númer tilheyra flestir framandi málum eins og t.d. rússnesku (0400 til 04FF), hebresku (0590 til 05FF) eða bengali (0980 til 09FF).

Með því að nota númerin er hægt að tákna stafi sem erfitt er að skrifa eins og t.d. vendi (return) og dálk (tab). Þá algengustu er líka hægt að tákna svona:

dálkur	`\t`
línuskipti	`\n`
vendi	`\r`
tvöfaldar gæsalappir	`\"`
einfaldar gæsalappir	`\'`
skástrik	`\\`

Hægt er að gefa breytum gildi um leið og þær eru skilgreindar. Dæmi:

```
double x = 0.7;
char s = '\t';
char stafur = 'R'
Button b = new Button("Reikna");
```

Hægt er að breyta `double` í `int`, `int` í `char` o.s.frv. Ef `i` er af tegundinni `int` og `s` af tegundinni `char` þá verða skipanirnar

```
i = 65;
s = (char)i;
```

til þess að í `s` fer stafur númer 65 sem er 'A'.

Skipanirnar

```
double d = 6.7;
int i = (int)d;
```

verða til þess að `i` fær gildið 6. Athugið að þegar `double` er breytt í `int` er alltaf rúnnað að næstu heiltölu fyrir neðan, því sem er fyrir aftan kommu er einfaldlega hent svo 6.999 verður 6 en ekki 7.

Í eftirfarandi forriti er breyting úr einni tegund í aðra notuð til þess að breyta hexadesímaltölum í samsvarandi stafi.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_02_04
// Stafir
//
public class Stafir extends Applet
{
   TextField tTala, tStafur;
   Button bFinnaStaf;
   Label a1;
   char stafur;
   Integer I;
   int i;

   public void init()
   {
      tTala = new TextField(5);
      tStafur = new TextField(25);
      bFinnaStaf = new Button("Finna staf");
      a1 = new Label("Hexadesímaltala");

      this.add(a1); this.add(tTala);
      this.add(bFinnaStaf); this.add(tStafur);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bFinnaStaf)
      {
         I = new Integer(0);
         I = Integer.valueOf(tTala.getText(), 16);
         i = I.intValue();
         stafur = (char)i;
         tStafur.setText("Stafur númer " + tTala.getText()
                         + " er " + stafur);
```

```
            return true;
        }
        return false;
    }
}
```

Taktu eftir skipununum

```
I = new Integer(0);
I = Integer.valueOf(tTala.getText(), 16);
```

Fyrst er búinn til nýr hlutur af tegundinni `Integer` og honum gefið gildið 0. Síðan er honum gefið gildið sem fæst með því að túlka `tTala.getText()` sem tölu með grunntölunni 16, þ.e. sem hexadesímaltölu. Ef ekki ætti að skrifa númerin sem hexadesímaltölur heldur á venjulegan hátt í tugakerfi mætti í stað þessara tveggja skipana setja

```
I = new Integer(tTala.getText());
```

2.f. `TextArea`

Hlutur af tegundinni `TextField` getur aðeins sýnt eina línu af texta. Þurfi að skrifa margar línur er hægt að nota `TextArea`. Ef breytan `t` er af tegundinni `TextArea` er hægt að búa til reit sem er 5 línur á hæð og 25 stafir á breidd með

```
t = new TextArea(5, 25);
```

Til að bæta línu af texta í reitinn og láta standa í henni "hani, krummi, hundur, svín" er hægt að skipa

```
t.appendText("hani, krummi, hundur, svín" + '\n');
```

Stafurinn `'\n'` veldur því að hoppað er niður í næstu línu.

Verkefni 2.8
Breyttu klasanum `Stafir` þannig að svörin birtist í `TextArea` og safnist þar fyrir. Ef fyrst er beðið um staf númer 52 og svo númer 53 á að standa í neðri reitnum

Stafur númer 52 er R
Stafur númer 53 er S

Verkefni 2.9
Breyttu forritinu í verkefni 8 þannig að þar sem svörin birtast séu númerin sýnd bæði sem hexadesímaltölur og sem tölur í tugakerfi. Sé til dæmis slegin inn talan 52 á línan

Stafur númer 52 (=82 í tugak.) er R

að birtast í neðri reitnum (þ.e. í `TextArea`).

2.x. Spurningar og umhugsunarefni

1. Hvaða munur er á tegundunum `double` og `Double`?
2. Hvað gera eftirfarandi skipanir ef `t` er af tegundinni `TextField`?

```
Double D;
double d;
```

```
D = new Double(t.getText());
d = D.doubleValue();
```

3. Hvaða upplýsingar um aðferðina til að reikna sínus má lesa út úr titillínunni

```
public static double sin(double a)
```

4. Hvers vegna er hægt að nota klasann `Math` og aðferðir sem tilheyra honum án þess að hann sé „fluttur inn“ með `import` skipun í upphafi forrits?

5. Hvaða gildi fá breyturnar `c` og `d` ef þessar skipanir eru gefnar?

```
int c, d;
c = 10 / 3;
d = 10 % 3;
```

6. Hvaða einfaldar tegundir geyma heilar tölur?

7. Hvaða einfaldar tegundir geyma kommutölur?

8. Hvaða einfaldar tegundir geyma hvorki heilar tölur né kommutölur?

9. Hvers vegna eru 4 gerðir af einföldum breytum til að geyma heilar tölur?

10. Hvaða gildi hafa breyturnar `x` og `t` upphaflega ef þær eru skilgreindar svona?

```
int x;
TextField t;
```

11. Hvaða munur er á einföldum breytum og breytum sem vísa á hluti?

12. Hvað merkja `'\n'`, `'\r'`, `'\\'` og `'\t'` ?

13. Hvaða gildi fá `d` og `i` ef eftirfarandi skipanir eru gefnar?

```
double d = 1.7
int i = (int)d
```

14. Hvaða munur er á tegundunum `TextArea` og `TextField`?

3. kafli: Skilyrði og `boolean` breytur

3.a. `if else`

Forrit þurfa oft að velja milli tveggja eða fleiri möguleika. Í Java er þetta gert með skipununum `if` og `else`.

Gerðu ráð fyrir að `i` sé af tegundinni `int` og `t` af tegundinni `TextField`. Skipunin sem hér fer á eftir skrifar þá „Stór tala“ í `t` ef `i` er meira en 100.

```
if (i > 100)
{
   t.setText("Stór tala");
}
```

Ef `i` er ekki meira en 100 þá gerir þessi skipun ekki neitt. Ef við viljum að skrifað sé „Lítil tala“ ef `i` er ekki meira en 100 þá getum við notað

```
if (i > 100)
{
   t.setText("Stór tala");
}
else
{
   t.setText("Lítil tala");
}
```

Þessi setning velur milli tveggja möguleika. Einnig er hægt að nota `if` og `else` til að velja milli þriggja eða fleiri kosta. Eftirfarandi skrifar „Stór tala“ ef `i` er meira en 100, „Meðalstór tala“ ef `i` er milli 11 og 100 og „Lítil tala“ ef `i` er 10 eða minna.

```
if (i > 100)
{
   t.setText("Stór tala");
}
else if (i > 10)
{
   t.setText("Meðalstór tala");
}
else
{
   t.setText("Lítil tala");
}
```

Aftan við `if` kemur setning innan sviga og hún hefur annað hvort gildið `true` (satt) eða `false` (ósatt). Hér eru nokkur dæmi um setningar sem eru sannar eða ósannar. Gert er ráð fyrir að `i` og `j` séu einhver gerð heiltalna.

`i == 7`	i er jafnt og 7.
`i < 100`	i er minna en 100
`i >= 200`	i er meira en eða jafnt og 200
`i != j`	i er ekki jafnt og j
`(i % 10) == 1`	þegar 10 er deilt í i gengur 1 af
`(i == 7) || (i == 11)`	i er 7 eða i er 11
`(j > 10) && (j <= i)`	j er meira en 10 og j er minna en eða jafnt og i

Í þessum dæmum koma fyrir ýmsar samanburðar- og rökaðgerðir. Þær helstu eru:

==	Sama og	&&	bæði og
!=	ekki sama og	||	a.m.k. annað hvort
>	meira en	!	ekki
<	minna en		
>=	meira en eða sama		
<=	minna en eða sama		

Verkefni 3.1 *
Breyttu lausninni á verkefni 2.1 þannig að ásamt flatamálinu sé skrifað „Stór ferhyrningur“ ef það er meira en 100 og „Lítill ferhyrnningur“ ef það er ekki meira en 100.

Verkefni 3.2 *
Breyttu lausninni á verkefni 3.1 þannig að skrifað sé:
„Stór ferhyrningur“ ef flatarmálið er meira en 100;
„Venjulegur ferhyrningur“ ef flatarmálið er milli 10 og 100 og
„Lítill ferhyrningur“ ef flatarmálið er minna en 10.

3.b. ==, strengir og `equals`

Hægt er að nota samanburðaraðgerðina `==` til að finna hvort tvær einfaldar breytur innihalda sama gildi. Málin verða flóknari þegar finna þarf hvort tveir hlutir (t.d. af tegundinni `Integer` eða `String`) innihalda sama gildi. Ef `I` og `J` eru af tegundinni `Integer` þá er `(I == J)` satt ef `I` og `J` eru sami hlutur, en ef `I` og `J` eru tveir hlutir sem innihalda sama gildi (t.d. töluna 7) þá er `(I == J)` ósatt.

Hlutur af tegundinni `String` getur innihaldið streng, þ.e. runu af bókstöfum. Við getum búið til strengjabreytu og gefið henni gildi svona:

```
String nafn;
nafn = new String("Jónatan"); // Nafn er nýr strengur sem
                              // inniheldur stafarununa
                              // "Jónatan"
```

Þar sem tegundin `String` er í pakkanum `java.lang` þarf ekki að flytja hana inn til að hægt sé að nota hana.

Ef við viljum nú komast að því hvort breytan `nafn` hafi sama gildi og önnur breyta sem við skulum kalla `x` getum við ekki notað `(nafn == x)` því þetta er þá aðeins satt að `nafn` og `x` innihaldi sama vistfang og vísi þar með á sama hlut. Það dugar ekki að breyturnar vísi á tvo eins hluti. En sem betur fer kunna hlutir af tegundinni `String` aðferð til að finna út hvort þeir eru eins og annar strengur. Við getum gáð hvort `x` og `nafn` vísa á sömu stafrunu með skipuninni

```
nafn.equals(x)
```

Eftirfarandi forrit sýnir dæmi um þetta.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_03_01
// Skilyrdi
//
public class Skilyrdi extends Applet
{
   TextField t1, t2;
   Button b1;
   Label a1;
   String nafn;

   public void init()
   {
      t1 = new TextField(20);
      t2 = new TextField(20);
      b1 = new Button("Heilsa");
      a1 = new Label("Nafn:");
      this.add(a1); this.add(t1);
      this.add(b1); this.add(t2);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == b1)
      {
         nafn = new String();
         nafn = t1.getText();
         if (nafn.equals("Jónatan"))
         {
            t2.setText("Jæja svo þú heitir Jónatan");
         }
         else
         {
            t2.setText("Góðan dag " + t1.getText());
         }
         return true;
      }
      return false;
   }
}
```

Verkefni 3.3 *
Búðu til forrit sem er eins og forritið hér að ofan nema hvað það segir:
„Jæja svo þú heitir Jónatan“ ef nafnið er Jónatan;
„Jæja svo þú heitir Pálína“ ef nafnið er Pálína og
„Ég heiti líka Jósafat“ ef nafnið er Jósafat.

Forrit_02_03 sagði hvað svo og svo margar sekúndur eru margar mínútur og sekúndur. Það notaði skammstafanir, sagði t.d. „82 sek. er 1 mín. og 22 sek.“. Með því að skammstafa orðin „mínúta“ og „sekúnda“ hliðraði forritið sér hjá að taka afstöðu til þess hvenær á að segja „mínúta“ og hvenær á að segja „mínútur“. Reglan er sú að nota eintölu ef talan er 1, 21, 31, 41 o.s.frv. þ.e.a.s. ef talan heitir `i` þá er notuð eintala ef

```
((i % 10) == 1) && ((i % 100) != 11)
```

Eftirfarandi forrit skrifar „mínúta” og „sekúnda” þar sem við á og „mínútur” og „sekúndur” þar sem við á.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_03_02
// Klukka
//
public class Klukka extends Applet
{
   TextField t1, tSvar;
   Button bReikna;
   Label a1;
   Integer I;
   int i, sek, min;
   String svar;

   public void init()
   {
      t1 = new TextField(5);
      tSvar = new TextField(25);
      bReikna = new Button("Reikna mínútur");
      a1 = new Label("Sekúndur");
      this.add(a1); this.add(t1);
      this.add(bReikna); this.add(tSvar);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bReikna)
      {
         I = new Integer(t1.getText());
         i = I.intValue();
         min = i / 60; sek = i % 60;
         svar = ""; // Tryggir að strengur sé tómur í upphafi.

         if (((min % 10) == 1) && ((min % 100) != 11))
         {
            svar = min + " mínúta";
         }
         else
         {
            svar = min + " mínútur";
         }

         if (((sek % 10) == 1) && (sek != 11))
         {
            svar = svar + " og " + sek + " sekúnda.";
         }
         else
         {
            svar = svar + " og " + sek + " sekúndur.";
         }
         tSvar.setText(svar);
         return true;
      }
      return false;
   }
}
```

Verkefni 3.4 *
Bættu við forritið hér að ofan þannig að það taki klukkustundir með og segi t.d.
„2 klukkustundir 1 mínúta og 3 sekúndur" ef talan 7263 er slegin inn og
„1 klukkustund 2 mínútur og 1 sekúnda" ef talan er 3721.

Verkefni 3.5
Bættu við lausnina á verkefni 3.4 þannig að forritið skrifi
„7263 sekúndur eru 2 klukkustundir 1 mínúta og 3 sekúndur" og
„3721 sekúnda er 1 klukkustund 2 mínútur og 1 sekúnda" ef tölurnar 7263 og 3721 eru slegnar inn.

Verkefni 3.6
Bættu við lausnina á verkefni 3.5 þannig að forritið skrifi „0 sekúndur eru enginn tími" ef talan 0 er slegin inn.

Í `action`-aðferðinni í forrit_03_02 er strengjabreytunni `svar` gefið gildi án þess fyrst sé smíðaður strengur með skipuninni `new`.

Strengir eru undantekning frá þeirri reglu að allir hlutir skuli búnir til með skipuninni `new`. Ef við ætlum að láta strengjabreytu heita `n` og geyma í henni strenginn "Jónatan" þá getum við fylgt almennu reglunni og notað `new` svona:

```
String n;
n = new String("Jónatan");
```

eða svona:

```
String n = new String("Jónatan");
```

En við getum líka sleppt því að nota `new` og sagt bara

```
String n;
n = "Jónatan";
```

eða

```
String n = "Jónatan";
```

Það er hægt að búa streng til með því einu að setja stafarunu innan gæsalappa.

3.c. `boolean` breytur

Breyta af tegundinni `boolean` getur tekið tvö mismunandi gildi: `true` og `false`. Ef eftirfarandi skipanir eru gefnar fá `b1` og `b4` gildið `false` og `b2` og `b3` fá gildið `true`.

```
int i = 5;                  i fær gildið 5
b1 = i > 10;                i > 10 er ósatt
b2 = !(b1);                 (ekki b1) er satt
b3 = (i > 12) || (i < 6);   (i > 12) eða (i < 6) er satt því (i < 6)
b4 = (12 % i) == 0;         að 0 gangi af ef 5 er deilt í 12 er ósatt
```

Eftirfarandi forrit notar `boolean` breytu til að geyma upplýsingar um hvort ár er hlaupár eða ekki. (Meginreglan er að það er hlaupár ef 4 ganga upp í ártalið og ekki

eru aldamót. Þannig var t.d. hlaupár 1896 en ekki 1900. Þó er hlaupár fjórðu hver aldamót, þ.e. þegar 400 ganga upp í ártalið. Það er því hlaupár árið 2000.)

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_03_03
// Hlaupar
//
public class Hlaupar extends Applet
{
   TextField tArtal, tUtkoma;
   Button bReikna;
   Label a1;
   Integer I;
   int i;
   boolean h;

   public void init()
   {
      tArtal = new TextField(5);
      tUtkoma = new TextField(15);
      a1 = new Label("Ártal");
      bReikna = new Button("Reikna");
      this.add(a1); this.add(tArtal);
      this.add(bReikna);
      this.add(tUtkoma);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bReikna)
      {
         I = new Integer(tArtal.getText());
         i = I.intValue();

         if ((i % 400) == 0)
         {
            h = true;
         }
         else
         {
            h = ((i % 4) == 0) && ((i % 100) != 0);
         }

         if (h)
         {
            tUtkoma.setText(i + " er hlaupár.");
         }
         else
         {
            tUtkoma.setText(i + " er ekki hlaupár.");
         }
         return true;
      }
      return false;
   }
}
```

Verkefni 3.7
Búðu til forrit sem tekur við ártali og númeri mánaðar og segir hve margir dagar eru í mánuðinum.

Dæmi:
Séu slegnar inn tölurnar 1996 og 11 á forritið að svara „30 dagar";
séu slegnar inn tölurnar 1996 og 2 á forritið að svara „29 dagar" og
séu slegnar inn tölurnar 1997 og 2 á það að „svara 28" dagar.

3.d. `switch`

Stundum þarf að láta forrit velja milli margra möguleika. Þá getur verið hentugt að nota `switch` fremur en `if else`. Hugsum okkur til dæmis að `int`-breytan `x` geymi númer vikudags og það eigi að setja nafn hans í textareitinn `t`. Þetta er hægt að gera með skipuninni

```
if (x == 1) { t.setText("sunnudagur"); }
else if (x == 2) { t.setText("mánudagur"); }
else if (x == 3) { t.setText("þriðjuudagur"); }
else if (x == 4) { t.setText("miðvikudagur"); }
else if (x == 5) { t.setText("fimmtudagur"); }
else if (x == 6) { t.setText("föstudagur"); }
else if (x == 7) { t.setText("laugardagur"); }
else { t.setText("enginn dagur hefur þetta númer"); }
```

Það er hægt að gera nákvæmlega það sama með

```
switch (x)
{
   case 1 : t.setText("sunnudagur"); break;
   case 2 : t.setText("mánudagur"); break;
   case 3 : t.setText("þriðjuudagur"); break;
   case 4 : t.setText("miðvikudagur"); break;
   case 5 : t.setText("fimmtudagur"); break;
   case 6 : t.setText("föstudagur"); break;
   case 7 : t.setText("laugardagur"); break;
  default : t.setText("enginn dagur hefur þetta númer");
}
```

Til að hægt sé að nota `switch` þarf gildið sem stjórnar því hvað valið er (í þessu tilviki breytan `x`) að vera af einhverri tegundanna `int`, `byte`, `short` eða `char`.

Allt sem hægt er að gera með `switch` er líka hægt að gera með `if else` en í sumum tilvikum eru forrit læsilegri ef `switch` er notað.

Verkefni 3.8
Búðu til forrit sem tekur við ártali og númeri mánaðar og segir hve margir dagar eru í mánuðinum og hvað hann heitir. Notaðu `switch` til að velja mánuðinum nafn.

3.x. Spurningar og umhugsunarefni

1. Hvað merkja táknin `&&`, `||` og `!` ?
2. Hvernig eru eftirtaldar fullyrðingar skrifaðar á Java?

   i er 7
   i er ekki 7
   i er minna en eða jafnt og 7
   i er meira en 7 eða i er minna en 3
   i er meira en 3 og i er minna en 7
   i er milli 3 og 7 (hvorki 3 né 7 meðtalið)
   i er milli 3 og 7 að báðum tölum meðtöldum

3. Hvaða gildi fær `x` ef þessar skipanir eru gefnar?

```
int x;
String n1, n2;
n1 = new String("voff");
n2 = new String("voff");
if (n1 == n2)
{
   x = 1;
}
else
{
   x = 0;
}
```

4. Hvaða gildi fá breyturnar `p`, `q`, `r` og `s` ef þessar skipanir eru gefnar og af hvaða tegund ætli þær séu?

```
int i = 5;
int j = 7;
p = ((j % i) == 2);
q = ((j / i) == 2);
r = !p || q;
s = p && !q;
```

5. Hvaða tilgangi þjónar skipunin `switch` og af hvaða tegundum getur `n` verið ef eftirfarandi skipun er gefin?

```
switch(n)
{
   case 0 : d=0;
   case 1 : d=5;
   case 2 : d=25;
  default : d=27;
}
```

4. kafli: Endurtekning

4.a. `while`, `++` og `--`

Skilyrðissetningar eru önnur af mikilvægustu byggingareiningum forrita. Hin er endurtekning. Í síðasta kafla var farið í skilyrðissetningar sem eru myndaðar með orðunum `if` og `else`. Nú er komið að endurtekningu. Í Java er hægt að nota skipunina `while` til þess að endurtaka sömu skipanarununa aftur og aftur.

```
TextArea t = new TextArea(5, 25);
int i = 1;
while (i <= 5)
{
   t.appendText("Mér finnst rigningin góð" + '\n');
   i++;
}
```

Forritsbúturinn hér að ofan skrifar fimm línur í textareitinn `t` og lætur standa það sama í þeim öllum.

```
while (i <= 5)
{

}
```

þýðir að meðan `i` er minna en eða jafnt og 5 skuli framkvæma skipanirnar milli slaufusviganna. Ef `i` er meira en 5 í upphafi eru skipanirnar milli slaufusviganna aldrei framkvæmdar. Ef `i` verður aldrei meira en 5 eru þær framkvæmdar aftur og aftur endalaust. Til að framkvæma þær í 1 eða fleiri skipti en þó ekki endalaust þarf `i` að vera minna en eða jafnt og 5 í byrjun og hækka þegar skipanirnar milli slaufusviganna eru framkvæmdar. Hér sér skipunin `i++` um að `i` hækki alltaf um 1.

```
i++;
```

þýðir nákvæmlega það sama og

```
i = i + 1;
```

Eins og aðgerðin `++` hækkar innihald breytu um einn lækkar `--` innihald breytu um 1.

```
i--;
```

Þýðir nákvæmlega það sama og

```
i = i - 1;
```

Hér kemur svo dæmi um heilt forrit sem notar while til að endurtaka sömu skipanirnar aftur og aftur.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_04_01
// Endurtekning
//
public class Endurtekning extends Applet
{
   TextField tTala;
   TextArea tFloskur;
   Button bByrja;
   Label a1;
```

```
   Integer I;
   int i, n;

   public void init()
   {
      tTala = new TextField(5);
      tFloskur = new TextArea(5,25);
      bByrja = new Button("Byrja");
      a1 = new Label("Tala");

      this.add(a1); this.add(tTala);
      this.add(bByrja); this.add(tFloskur);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bByrja)
      {
         I = new Integer(tTala.getText());
         i = I.intValue();
         n = 1;
         while (n <= i)
         {
            tFloskur.appendText(n +
               " grænar flöskur hangandi uppi á vegg." + '\n');
            n++;
         }
         return true;
      }
      return false;
   }
}
```

Verkefni 4.1 *
Búðu til forrit sem skrifar 25 sinnum „Ég er stór og duglegur”. (Þú mátt láta það skrifa „Ég er stór og dugleg” ef þú vilt það heldur.)

Verkefni 4.2 *
Breyttu forritinu hér að ofan þannig að það skrifi „græn flaska” en ekki „grænar flöskur” þegar talan er 1, 21, 31 o.s.frv.

Ef breytan `tTafla` er af tegundinni `TextArea` og `i` og `u` eru af tegundinni `int` þá er hægt að nota eftirfarandi til að skrifa margföldunartöflu.

```
   n = 1;
   while (n <= i)
   {
      tTafla.appendText(n + " * " + i + " = " + n*i + '\n');
      n++;
   }
```

Verkefni 4.3 *
Búðu til forrit sem tekur við einni tölu (í `TextField`) og skrifar samsvarandi margföldunartöflu (í `TextArea`) þegar ýtt er á takka. Sé til dæmis slegin inn talan 4 á forritið að skrifa

1 * 4 = 4
2 * 4 = 8
3 * 4 = 12
4 * 4 = 16

Verkefni 4.4 *
Búðu til forrit sem tekur við 2 tölum og skrifar töflu yfir fallið x^2 frá fyrri tölunni til þeirrar seinni. Séu tölurnar t.d. 5 og 8 á forritið að skrifa

5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64

4.b. Almenn brot, aðferð Evklíðs og strengir

Margir útreikningar verða ekki framkvæmdir án þess að beita endurtekningu. Dæmi um þetta er stytting á almennum brotum. Til að stytta brot eins og t.d. 42/105 þarf að finna stærstu tölu sem gengur bæði upp í teljarann og nefnarann. Í þessu tilviki er það auðvelt. 21 gengur bæði upp í 42 og 105 ($2 \times 21 = 42$ og $5 \times 21 = 105$) svo fullstytt er brotið skrifað 2/5.

Algengasta aðferðin til að finna stærsta sameiginlegan þátt tveggja talna er kölluð aðferð Evklíðs og kennd við gríska stærðfræðinginn Evklíð sem uppi var um 300 f. Kr.

t	n	afgangur
42	105	42
↙	↙	
105	42	21
↙	↙	
42	21	0
↙	↙	
21	0	

Til að finna stærsta sameiginlegan þátt tveggja talna sem við skulum kalla t og n (fyrir teljara og nefnara) byrjum við á að reikna hvað gengur af ef n er deilt í t. Síðan færum við n í sæti t og afganginn í sæti n og endurtökum (sjá rammann til hliðar) aftur og aftur þar til 0 er í sæti n, þá er stærsti sameiginlegi þáttur í sæti t.

Eftirfarandi forrit notar aðferð Evklíðs til að stytta brot.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_04_02
// Evklid
//
public class Evklid extends Applet
{
   TextField tTeljari, tNefnari, tFullstytt;
   Button bStytta;
   Label lTeljari, lNefnari, lFullstytt;
   Integer Teljari, Nefnari;
   int teljari, nefnari, t, n, afg;
```

```
   public void init()
   {
      lTeljari = new Label("Teljari");
      tTeljari = new TextField(2);
      lNefnari = new Label("Nefnari");
      tNefnari = new TextField(2);
      lFullstytt = new Label("Fullstytt");
      tFullstytt = new TextField(4);
      bStytta = new Button("Stytta");
      this.add(lTeljari); this.add(tTeljari);
      this.add(lNefnari); this.add(tNefnari);
      this.add(bStytta); this.add(lFullstytt);
      this.add(tFullstytt);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bStytta)
      {
         Teljari = new Integer(tTeljari.getText());
         teljari = Teljari.intValue();
         Nefnari = new Integer(tNefnari.getText());
         nefnari = Nefnari.intValue();

         t = teljari;
         n = nefnari;

         // Aðferð Evklíðs notuð til að finna stærsta
         // sameiginlegan þátt talnanna t og n.
         while (n > 0)
         {
            afg = t % n;
            t = n;
            n = afg;
         }

         // Nú er t stærsta tala sem gengur
         // bæði upp í teljara og nefnara
         // og við styttum brotið.
         teljari = teljari / t;
         nefnari = nefnari / t;

         tFullstytt.setText(teljari + "/" + nefnari);
         return true;
      }
      return false;
   }
}
```

Á þessu forriti er sá galli að ekki er hægt að skrifa brotið í einn reit heldur verður að skrifa teljarann og nefnarann hvorn í sinn reitinn. Úr þessu er hægt að bæta með því að taka við brotinu sem einum streng (t.d. "42/105") og klippa svo úr honum tvo búta (í þessu tilviki bútana "42" og "105" og breyta hvorum bút fyrir sig í tölu. Þetta er hægt að gera með strengjaaðferðunum `indexOf`, `length` og `substring`.

`indexOf(String s)` finnur hvar einn strengur kemur fyrst fyrir innan í öðrum streng. Séu t.d. gefnar skipanirnar

```
String s = "Hundur";
int x = s.indexOf("du");
int y = s.length();
String r = s.substring(1, 4);
```

þá fær `x` gildið 3 því í strengnum „Hundur” er „H” í sæti númer 0, „u” í sæti númer 1 o.s.frv. svo „du” byrjar í sæti 3.

Aðferðin `length` finnur út hvað strengur er margir stafir á lengd. Hér að ofan fær y gildið 6 því orðið „Hundur” er 6 stafir.

Skipunin `s.substring(1, 4)` skilar streng sem er myndaður úr táknum frá og með númer 1 að númer 4 (númer 4 þó ekki talið með) í strengnum `s`. Í dæminu hér að ofan fær `r` því gildið "und".

Ef strengur sem táknar almennt brot er látinn heita `sBrot` og breyturnar `sTeljari` og `sNefnari` eru líka af tegundinni `String` þá er hægt að koma teljara og nefnara brotsins fyrir í `int` breytum svona. (`tBrot` er að sjálfsögðu af tegundinni `TextField`. `Teljari` og `Nefnari` eru `Integer`. `teljari` og `nefnari` eru `int`.)

```
sBrot = tBrot.getText();
int i = sBrot.indexOf("/");          //Finnur hvar / er.
int j = sBrot.length();

sTeljari = sBrot.substring(0, i);    //Teljari er tákn frá og
Teljari = new Integer(sTeljari);     //með númer 0 til númer i
teljari = Teljari.intValue();

sNefnari = sBrot.substring(i+1, j); // Nefnari er tákn frá
Nefnari = new Integer(sNefnari);     // númer i+1 til enda.
nefnari = Nefnari.intValue();
```

Verkefni 4.5 *
Breyttu forritinu sem styttir brot (forrit_04_02) þannig að það taki við brotinu sem einum streng.

Verkefni 4.6 *
Búðu til forrit sem tekur við tveim almennum brotum, leggur þau saman og skilar útkomunni fullstyttri.

4.c. `for`, `do-while` og `*=`

Skipunin `while` er ekki sú eina sem hægt er að nota til að endurtaka sömu romsuna aftur og aftur. Það er líka hægt að nota `for` og `do-while`. Romsur sem eru endurteknar með `while`, `for` eða `do-while` kallast einu nafni slaufur.

Gerum ráð fyrir að `i` og `n` séu af tegundinni `int` og `tTafla` af tegundinni `TextArea`. Þá merkir

```
for (n=1; n <= i; n++)
{
   tTafla.appendText("Sísí sá sól" + '\n');
}
```

að `n` skuli í upphafi vera 1 og svo lengi sem `n` er minna en eða jafnt og `i` skuli framkvæma allt sem er innan slaufusviganna og hækka `n` um 1 (þ.e. framkvæma `n++`)

Skipunin `for er` strangt tekið óþörf. Í staðinn fyrir

```
for (n=1; n <= i; n++)
{
   tTafla.appendText("Sísí sá sól" + '\n');
}
```

er hægt að nota

```
n = 1;
while (n <= i)
{
   tTafla.appendText("Sísí sá sól" + '\n');
   n++;
}
```

Sé breytan `n` skilgreind inni í sviganum aftan við `for` svona

```
for (int n=1; n <= i; n++)
{
   tTafla.appendText("Sísí sá sól" + '\n');
}
```

þá er hún aðeins til innan `for`-slaufunnar.

Þriðja gerðin af endurtekingu í Java er `do-while`. Skipunin

```
do
{
   tTafla.appendText("Sísí sá sól" + '\n');
   n++;
} while (n < i)
```

framkvæmir það sem er á milli slaufusviganna aftur og aftur á meðan `n` er minna en `i`. Munurinn á `do-while`-slaufu og venjulegri `while`-slaufu er sá að í `do-while` slaufunni er skilyrðið sem þarf að uppfylla til að romsan innan slaufusviga sé endurtekin sett neðst. Það er því ekki gáð hvort það eigi að endurtaka slaufuna fyrr en búið er að framkvæma hana einu sinni. `do-while` slaufa er því alltaf framkvæmd að minnsta kosti einu sinni en sé skilyrðið ósatt í upphafi er `while` slaufa aldrei framkvæmd.

Verkefni 4.7 *
Breyttu lausninni á verkefni 4.3 með því að nota `for`-slaufu í stað `while`-slaufu.

Verkefni 4.8
Breyttu lausninni á verkefni 4.3 með því að nota `do-while`-slaufu í stað `while`-slaufu.

Eftirfarandi forritsbútur reiknar i![1] Gert er ráð fyrir að breyturnar `i` og `n` séu af tegundinni `int` en `utkoma` af tegundinni `long` því jafnvel þótt `i` sé fremur lág tala verður útkoman ansi há.

[1] i! er lesið i hrópmerkt. 7! er 1*2*3*4*5*6*7.

```
utkoma = 1;
for (n = 2; n <= i; n++)
{
   utkoma *= n;
}
```

Reikniaðgerðin `*=` margafaldar breytuna vinstra megin með breytunni hægra megin og setur útkomuna í breytuna vinstra megin.

`utkoma *= n;` merkir það sama og: `utkoma = utkoma * n;`

Til eru samsvarandi skipanir fyrir samlagningu, frádrátt og deilingu:

`utkoma += n;` merkir það sama og: `utkoma = utkoma + n;`
`utkoma -= n;` merkir það sama og: `utkoma = utkoma - n;`
`utkoma /= n;` merkir það sama og: `utkoma = utkoma / n;`

Verkefni 4.9 *
Búðu til forrit sem tekur við einni heiltölu x og reiknar reiknar x!.

Verkefni 4.10
Búðu til forrit sem tekur við einni tölu n og leggur saman $1^3 + 2^3 + ... + n^3$.

4.d. Tvöfaldar slaufur

Stundum er ein slaufa sett innan í aðra eins og í þessu forriti sem skrifar allar margföldunartöflur upp í 10 sinnum töfluna.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_04_03
// Margfoldunartoflur
//
public class Margfoldunartoflur extends Applet
{
   TextArea tTafla;
   Button bByrja;
   int i, j;

   public void init()
   {
      tTafla = new TextArea(10,25);
      bByrja = new Button("Byrja");
      this.add(bByrja);
      this.add(tTafla);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bByrja)
      {
         for (i=1; i <= 10; i++)
         {
            for (j=1; j <= i; j++)
            {
               tTafla.appendText(i + " * " + j + " = "
                           + i*j + '\n');
```

```
            }
            tTafla.appendText("\n");
         }
         return true;
      }
      return false;
   }
}
```

Taktu eftir skipuninni

```
tTafla.appendText("\n");
```

Hér er ekki hægt að nota

```
tTafla.appendText('\n');
```

því `appendText` aðferðin tekur við streng. `"\n"` er strengur með einu tákni (þ.e. af tegundinni `String`). `'\n'` er hins vegar af tegundinni `char`. Ástæðan fyrir því að hægt er að nota `'\n'` í

```
tTafla.appendText(i + " * " + j + " = " + i*j + '\n');
```

er að aðgerðin + skilar streng ef henni er beitt á streng og eitthvað annað. T.d. fá `s` og `t` gildin "dvergarnir 7" og "3 kettlingar" ef gefnar eru skipanirnar

```
s = "dvergarnir " + 7;
t = '3' + " kettlingar";
```

Þótt `'3'` sé af tegundinni `char` og 7 sé heiltala þá verður útkoman strengur því "dvergarnir " og " kettlingar" eru strengir.

Verkefni 4.11
Breyttu forritinu hér að framan (forrit_04_03) þannig að það skrifi hverja töflu upp í 10 sinnum.

Það er hægt að nota tvöfalda `while` slaufu til að þátta tölu (sem hér kallast `i`) í prímþætti svona:

```
n = 2;                              // Fyrsta talan sem
while (i > 1)                       // er prófuð er 2
{
   while ((i % n) == 0)             // meðan n gengur
   {                                // upp í i
      tTaettir.appendText(n+"\n");  // er n skrifað
      i /= n;                       // og deilt í i með n.
   }                                // Eftir það
   n++;                             // er n hækkað um 1.
}
```

Hér er gert ráð fyrir að `tTaettir` sé af tegundinni `TextArea` og `i` og `n` séu int.

Verkefni 4.12
Gerðu ráð fyrir að `i` sé talan 60. Rektu á blaði eða í huganum hvaða gildi `i` og `n` hafa og hvað er í `tTaettir` þegar búið er að fara í fyrsta sinn, annað sinn, þriðja sinn og fjórða sinn gegnum ytri `while`-slaufuna hér að framan. (Innri slaufan er

```
while ((i % n) == 0)
{
   tTaettir.appendText(n+"\n");
   i /= n;
}
```

og sú ytri endurtekur hana og skipunina `n++` meðan `i > 1.`)

Verkefni 4.13
Búðu til forrit sem tekur við einni tölu og skrifar alla prímþætti hennar.

Verkefni 4.14
Búðu til forrit sem tekur við einni tölu og segir hvort hún er prímtala.

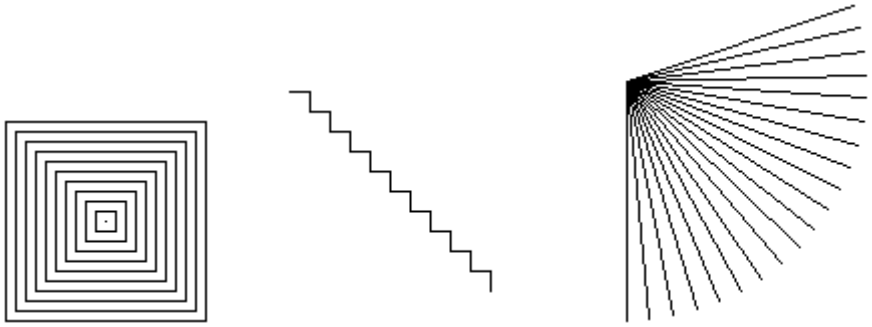
4.e. Dálítil teikning

Með því að nota endurtekningu er hægt að teikna alls konar mynstur og reglulegar myndir. Forrit_04_04 teiknar til dæmis 12 mislöng strik með dálitlu bili á milli.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_04_04
// Mynd
// Teiknar 12 strik
//
public class Mynd extends Applet
{
   int xa, ya, xb, yb;

   public void paint(Graphics g)
   {
      xa = 20;
      ya = 20;
      xb = 30;
      yb = 180;
      for (int i=0; i<12; i++)
      {
         g.drawLine(xa+10*i, ya+4*i, xb+12*i, yb-8*i);
      }
   }
}
```

Verkefni 4.15
Prófaðu að keyra forrit_04_04. Breyttu svo `paint` aðferðinni í því þannig að það teikni öðru vísi mynstur t.d. einhver þeirra sem hér eru fyrir neðan.



4.x. Spurningar og umhugsunarefni

1. Hvaða gildi fá breyturnar `i` og `j` ef þessar skipanir eru gefnar?

```
int i = 5;
int j = 7;
i++;
j--;
```

2. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 1;
while (i < 10)
{
   i = i * 2;
}
```

3. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 1;
for (int n = 0; n < 5; n++)
{
   i = i * 2;
}
```

4. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 1;
do
{
   i = i * 2;
} while (i < 10)
```

5. Hvað er aðferð Evklíðs?

6. Hvaða gildi fá breyturnar `a`, `b`, og `c` ef þessar skipanir eru gefnar?

```
String a = new String("Skagamenn skoruðu öll mörkin.");
int b = a.indexOf("skoruðu");
String c = a.substring(b, b+7);
```

7. Hvaða gildi fá breyturnar `a`, `b`, og `c` ef þessar skipanir eru gefnar?

```
int a, b, c;
a = 1; b = 2; c = 3;
a *= b;
b += c;
c -= 1;
```

8. Af hvaða tegund er `x` ef hægt er að gefa skipunina

```
x = "5" + 7;
```

9. Gerðu ráð fyrir að `t` sé af tegundinni `TextArea`. Af hverju er í lagi að gefa skipunina

```
t.appendText("Skagamenn skoruðu öll mörkin." + '\n');
```

   en ekki í lagi að gefa skipunina

```
t.appendText('\n');
```

10. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 0;
for (int a=0; a<10; a++)
{
   i++;
}
```

11. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 0;
for (int a=0; a<10; a++)
{
   for (int b=0; b<10; b++)
   {
      i++;
   }
}
```

12. Hvaða gildi fær breytan `i` ef þessar skipanir eru gefnar?

```
int i = 0;
for (int a=0; a<10; a++)
{
   for (int b=0; b<10; b++)
   {
      for (int c=0; c<10; c++)
      {
         i++;
      }
   }
}
```

Til umhugsunar

Sumar slaufur eru endalausar eins og t.d.

```
int a = 2;
int i = 4;
while(i < 5)
{
   a *= -1;
}
```

Þessi slaufa er augljóslega endalaus. Ekkert sem gerist inni í henni getur orðið þess valdandi að `i` hækki upp í 5. En stundum er það engan veginn augljóst hvort slaufa er endalaus eða ekki.

Til er fræg setning í talnafræði sem er kölluð *síðasta setning Fermat* eftir franska stærðfræðingnum Pierre Fermat sem var uppi á 17. öld og sló henni fram og kvaðst geta sannað hana. En Fermat dó án þess að láta neitt uppi um hvernig hann sannaði þessa setningu. Síðan liðu rúmalega þrjár aldir án þess nokkrum tækist að skera úr um hvort setningin er sönn eða ósönn. Þó reyndu margir af færustu stærðfræðingum heims sitt ítrasta til að fá úr þessu skorið. Það tókst loks árið 1994 þegar Andrew Wiles stærðfræðingur við Princeton háskóla sannaði setninguna.

Síðasta setning Fermat segir að ef a, b, c, og n eru náttúrulegar tölur (þ.e. heilar tölur stærri en 0) og $n > 2$ þá sé ekki til lausn á jöfnunni $a^n + b^n = c^n$.

Nú er fremur vandalítið að búa til slaufu sem endar ef síðasta setning Fermat er ósönn en er endalaus ef hún er sönn. Spurningin um hvort sú slaufa er endalaus eða ekki er þá jafn erfið og spurningin um hvort síðasta setning Fermat er sönn. Það getur semsagt þurft nokkuð djúpar stærðfræðilegar pælingar til að komast að því hvort slaufa er endalaus eða ekki.

Dæmi um slaufu sem endaði því aðeins að síðasta setning Fermat væri ósönn fyrir n=3 gæti t.d. verið á þessa leið.

```
boolean lausnFundin = false;
int a = 1;
while (!lausnFundin)
{
   for(int b=1; b<a; b++)
   {
      for (int c=1; c<a+b; c++)
      {
         lausnFundin = ((a*a*a)+(b*b*b) == (c*c*c));
      }
   }
   a++;
}
```

(Hér er gert ráð fyrir því að c hljóti að vera minna en a+b því $(a+b)^3 > a^3 + b^3$.

Getur þú fundið út hvort eftirfarandi slaufa en endalaus?

```
int r = 3;
while (r > 0)
{
   r = (7 * r + 11) % 13
}
```

5. kafli: Klasar, hlutir, aðferðir og atburðir

5.a. Ættartré tegundanna

Forritin í köflum 1 til 4 eru öll gerð úr einum klasa sem hefur nokkrar breytur og eina eða tvær aðferðir. Í þeim flestum eru aðferðirnar

```
public void init()
```

og

```
public boolean action(Event e, Object o)
```

Í þessum forritum raðar `init`-aðferðin upp sýnilegum hlutum og `action`-aðferðin bregst við og gerir eitthvað þegar smellt er á hnappa.

Þegar forritin eru framkvæmd gerist heilmargt bak við tjöldin sem er ekki er beinlínis tekið fram í forritskóðanum að eigi að gerast. T.d. er búinn til hlutur af tegundinni sem myndar forritið og hann látinn framkvæma á sér aðferðina `init`. Ef forritið er klasi með titillínuna

```
public class Reikna extends Applet
```

Þá er eitthvað á borð við skipanirnar

```
Reikna x;
x = new Reikna();
x.init();
```

framkvæmt þegar forritið fer af stað.

Stór forrit eru iðulega mynduð úr mörgum klösum sem hver um sig kann margar aðferðir.

Allir klasar erfa beint eða óbeint frá `Object`. Klasarnir sem eru skilgreindir í köflum 1 til 4 erfa t.d. frá `Applet`. `Applet` erfir `Panel`, `Panel` erfir `Container`, `Container` erfir `Component` og `Component` erfir `Object`. Klasinn `Button` erfir frá `Component` sem erfir frá `Object`. Tegundirnar sem Java forrit eru mynduð úr raðast semsagt í eitt samhangandi ættartré.

Sagt er að tegund eða klasi sem önnur erfir frá sé yfirtegund hennar eða yfirklasi og erfinginn sé undirtegund eða undirklasi. Þannig er `Panel` yfirklasi `Applet` og `Applet` undirklasi `Panel`.

Hver klasi hefur aðferð til að búa til hluti. Slík aðferð kallast smiður (constructor á ensku) og heitir sama nafni og klasinn. Sé enginn smiður skilgreindur býr Java-þýðandinn til smið sem gerir ekkert annað en kalla á smið yfirklasans.

5.b. Brot, smiðir og `toString`

Hér fer á eftir dæmi um forrit sem er myndað úr tveim klösum. Fyrri klasinn (`Brot`) skilgreinir almennt brot. Í titillínu hans er ekkert tekið fram um að hann erfi neinn annan klasa. Þetta þýðir raunar ekki að hann sé arflaus heldur að hann erfi frá `Object` sem er sameiginlegur forfaðir allra klasa í Java. Seinni klasinn erfir frá `Applet` og notar tegundina `Brot` til að leggja saman tvö almenn brot.

```
// forrit_05_01
// Brot
```

```
//
public class Brot
{
   // Brot hefur aðeins tvo eiginleika (tvær klasabreytur)
   // sem eru teljari og nefnari. Þeir eru báðir public sem þýðir
   // að hlutir af öðrum tegundum geta haft aðgang að þeim.

   public int teljari, nefnari;

   // Hér eru skilgreindir tveir smiðir fyrir klasann Brot. Sá fyrri
   // tekur við tveim tölum sem eru teljari og nefnari. Sá seinni
   // tekur við einum streng, t.d. "5/7". Smiðirnir gera ekkert annað
   // en að gefa breytunum teljari og nefnari gildi.

   public Brot(int t, int n)
   {
      teljari = t;
      nefnari = n;
   }

   public Brot(String b)
   {
      int i, j;
      String sT, sN;
      Integer T, N;

      i = b.indexOf("/");
      j = b.length();

      sT = b.substring(0, i);
      T = new Integer(sT);
      teljari = T.intValue();

      sN = b.substring(i+1, j);
      N = new Integer(sN);
      nefnari = N.intValue();
   }

   // Hér fara á eftir þrjár aðferðir sem Brot eiga að kunna. Sú
   // fyrsta gerir broti kleift að stytta sig, sú önnur gerir því
   // mögulegt að bæta sér við annað brot. Þriðja aðferðin breytir
   // broti í streng.

   public void stytta()
   {
      int t, n, afg;
      t = teljari;
      n = nefnari;

      // Aðf. Evklíðs notuð til að finna stærsta  sameiginlegan þátt
      // talnanna t og n.

      while (n > 0)
      {
         afg = t % n;
         t = n;
         n = afg;
      }

      // Nú er t stærsta tala sem gengur bæði
      // upp í teljara og nefnara
      // og við styttum brotið.

      teljari = teljari / t;
      nefnari = nefnari / t;
   }
```

```
   public Brot plus(Brot b)
   {
      int t, n;
      Brot utkoma;
      t = teljari*b.nefnari + b.teljari*nefnari;
      n = nefnari*b.nefnari;
      utkoma = new Brot(t, n);
      utkoma.stytta();        // Aðferðin skilar útkomu sem
      return utkoma;          // fullstyttu broti.
   }

   public String toString()   // Yfirskyggir toString
   {                          // í teg. Object
      return teljari + "/" + nefnari;
   }
}  // Hér endar skilgr. á teg. Brot.
```

Taktu eftir því að klasinn `Brot` hefur tvo smiði. Þeir gera það sama, þ.e. gefa breytunum `teljari` og `nefnari` gildi. En þeir eru ólíkir að því leyti að þeir taka ekki við sams konar gildum. Annar tekur við tveim tölum, hinn tekur við einum streng. Í Java er allt í lagi að láta tvær aðferðir heita það sama ef þær taka ekki við sams konar gildum.

Áður en smiður framkvæmir skipanir sínar kallar hann á smið yfirklasa án þess að senda honum nokkur gildi. Þar sem `Brot` erfir beint frá `Object` byrja aðferðirnar

```
public Brot(int t, int n)
```

og

```
public Brot(String b)
```

á að framkvæma smið klasans `Object`.

Ef yfirklasi hefur engan smið sem hægt er að framkvæma án þess að senda honum gildi kvartar þýðandinn þegar reynt er að láta hann þýða forritið. Þetta þýðir að ef við ætlum að láta annan klasa erfa frá `Brot` þá ættum við að búa til smið með titillínunni

```
public Brot()
```

Hann gæti t.d. verið svona og smíðað brot með gildið 1/1.

```
public Brot()
{
   teljari = 1;
   nefnari = 1;
}
```

Hafi yfirklasi engan smið sem hægt er að framkvæma án þess að senda honum gildi getur smiður undirklasa komið í veg fyrir villu með því að hafa skipunina `super` efst.

Hver klasi kallar sjálfan sig `this` og yfirklasa sinn `super`. Ef smiður klasa sem erfir frá `Brot` byrjar t.d. á skipuninni

```
super(1, 1)
```

þá er ekki sjálfkrafa kallað á

```
Brot()
```

eins og ella væri gert heldur á

```
Brot(1, 1)
```

Hugsum okkur að við viljum búa til klasa sem heitir `RaedTala`, erfir frá `Brot` og geymir ræðar tölur á forminu 1 2/3, þ.e. sem heila tölu og brot. (Klasinn `Brot` verður að meðhöndla þessa tölu sem 5/3). Ef smiður fyrir `RaedTala` á að taka við þrem heiltölum þar sem sú fyrsta er heila talan, sú önnur teljarinn og sú þriðja nefnarinn þá getum við bætt við `Brot` smið sem ekki tekur við neinum gildum og haft smið `RaedTala` svona

```
public RaedTala(int h, int t, int n)
{
   teljari = t + h * n;
   nefnari = n;
}
```

Við getum líka haft klasann `Brot` óbreyttan og smiðinn fyrir `RaedTala` svona

```
public RaedTala(int h, int t, int n)
{
   super(t + h * n, n);
}
```

Sé `super`-aðferðin notuð á þennan hátt verður hún að vera fyrsta skipunin í smiðnum.

*

Klasinn `Object` hefur aðferð sem heitir `toString`. `Brot` inniheldur aðferð með sama nafni sem yfirskyggir `toString` aðferð klasans `Object`. Þetta þýðir að ef hlutur af tegundinni `Brot` framkvæmir `toString` þá er `toString` sem er skilgreint í `Brot` notað en ekki `toString`-aðferð klasans `Object`. Aðrar aðferðir sem tilheyra `Object` geta hlutir af tegundinni `Brot` framkvæmt rétt eins og þær væru skilgreindar í klasanum `Brot`.[1]

Almennt gildir að ef klasi, sem heitir t.d. `KlasiY`, hefur aðferð, sem heitir t.d. `buu`, og undirklasi hans, sem heitir t.d `KlasiU`, hefur aðferð með sama nafni þá yfirskyggir aðferð undirklasans aðferð yfirklasans. Þetta þýðir að ef gefnar eru skipanirnar

```
KlasiU u;
u = new KlasiU();
u.buu();
```

Þá lætur síðasta skipunin `u` framkvæma aðferðina `buu` í klasanum `KlasiU` en ekki samnefnda aðferð í `KlasiY`.

Flestir klasar innihalda sína eigin `toString` aðferð sem yfirskyggir samnefnda aðferð í klasanum `Object`. Aðferð með þessu nafni er iðulega látin skila streng sem segir eitthvað um innihald hlutarins. Þó hlutur hafi ekki sína eigin `toString` aðferð er samt hægt að láta hann framkvæma `toString` því hann getur notað aðferðina sem hann erfir frá `Object`. `toString`-aðferðin í `Object` skilar streng sem inniheldur heiti tegundarinnar (klasans) sem hlutur tilheyrir.

Java notar `toString`-aðferð hlutar ævinlega þegar þarf að breyta hlut í streng, t.d. þegar honum er bætt við annan streng með aðgerðinni +. Til dæmis eru skipanirnar

```
Brot b = new Brot(5, 7);
String s = "Brotið er: " + b;
```

jafngildar skipununum

```
Brot b = new Brot(5, 7);
String s = "Brotið er: " + b.toString();
```

[1] Þetta gildir ekki alveg án undantekninga. Sjá um `clone()` í kafla E.d.

*

Líttu á titillínu aðferðarinnar `stytta`. Hún er

```
public void stytta()
```

`public` þýðir að aðferðin sé allra gagn og `void` þýðir að hún skili engri útkomu. Innan sviganna er ekkert svo hún tekur ekki við neinu gildi.

Lítum nú á titillínu aðferðarinnar `plus`. Hún er svona:

```
public Brot plus(Brot b)
```

Útkoman úr þessari aðferð er af tegundinni `Brot` og hún tekur við einu gildi sem er líka af tegundinni `Brot`. Aðferðin endar á skipun um að skila hlutnum `utkoma` sem er af tegundinni `Brot`. Þessi skipun er

```
return utkoma;
```

`return`-skipun lætur aðferð skila útkomu og hætta. Skipanir sem eru fyrir aftan eða neðan `return`-skipun eru ekki framkvæmar ef `return`-skipunin er framkvæmd. Í `action`-aðferðum er yfirleitt `return` skipun inni í hverri skilyrðissetningu. Þetta þýðir að séu skipanirnar inni í skilyrðissetningunni framkvæmdar þá er öllu þar fyrir neðan sleppt.

Það er sjálfgefið að smiðir skila útkomu af tegundinni sem þeir smíða. Þess vegna er ekki skrifað

```
public Brot Brot(int t, int n)
```

og

```
public Brot Brot(String b)
```

heldur bara:

```
public Brot(int t, int n)
```

og

```
public Brot(String b)
```

Snúum okkur nú að því hvernig hægt er að nota tegundina `Brot`. Klasinn `Brotareikningur` gerir það. Ef hann er í sömu efnisskrá og `Brot` getur hann notað klasann `Brot` án þess að flytja hann inn með `import` skipun. (Eftirleiðis verður ævinlega gert ráð fyrir að tveir eða fleiri klasar sem mynda saman eitt forrit séu í sömu efnisskrá og þurfi því ekki að flytja hver annan inn með `import`.)

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_05_01
// Brotareikningur
//
public class Brotareikningur extends Applet
{
   TextField tBrot1, tBrot2, tSumma;
   Button bLeggjaSaman;
   Label lBrot1, lBrot2, lSumma;
   Brot brot1, brot2, summa;

   public void init()
   {
      lBrot1 = new Label("Brot 1");
      lBrot2 = new Label("Brot 2");
      lSumma = new Label("Summa");
      tBrot1 = new TextField(4);
      tBrot2 = new TextField(4);
      tSumma = new TextField(4);
      bLeggjaSaman = new Button("Leggja saman");

      this.add(lBrot1); this.add(tBrot1);
      this.add(lBrot2); this.add(tBrot2);
      this.add(bLeggjaSaman);
      this.add(lSumma); this.add(tSumma);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bLeggjaSaman)
      {
         brot1 = new Brot(tBrot1.getText());
         brot2 = new Brot(tBrot2.getText());
         summa = brot1.plus(brot2);
         tSumma.setText(summa.toString());
         return true;
      }
      return false;
   }
}
```

Taktu eftir línunni

```
summa = brot1.plus(brot2);
```

Þar sem `plus`-aðferðin skilar gildi af tegundinni `Brot` þarf að láta útkomuna úr `brot1.plus(brot2)` lenda í breytu af þeirri tegund. Þess vegna setjum við

```
summa =
```

framan við

```
brot1.plus(brot2)
```

Eigi hins vegar bara að stytta brotið `brot1` þá gefum við skipnina

```
brot1.stytta();
```

Við getum ekki sagt

```
summa = brot1.stytta();
```

því `stytta` er `void` og skilar engri útkomu sem hægt er að setja í breytuna summa.

Þetta forrit er samsett úr tveim klösum. Þegar það er skrifað eru búnar til tvær skrár sem heita það sama og klasarnir að fornafni en hafa eftirnafnið java. Það er hægt að setja báða klasana í sömu skrá. Hún verður þá að heita sama nafni og klasinn sem framkvæma skal (þ.e. `Brotareikningur`) og hinn klasinn (þ.e. `Brot`) má þá ekki hafa `public` í titillínu. Hver skrá má aðeins innihalda einn `public` klasa.

Þegar kóðinn er þýddur verða til skrár sem heita sömu nöfnum og klasarnir að viðbættu eftirnafninu class. Það má einu gilda hvort klasarnir eru upphaflega skrifaðir í sömu skrá eða sinn í hvora, þegar forritið er þýtt verður til ein skrá (með eftirnafnið class) fyrir hvern klasa.

Verkefni 5.1 *
Bættu við klasann `Brot` aðferðum til að margfalda brot með öðru broti, draga brot frá broti og deila broti í brot. Bættu svo við klasann `Brotareikningur` þannig að hann hafi fjóra takka, einn fyrir samlagningu, einn fyrir margföldun, einn fyrir frádrátt og einn fyrir deilingu og framkvæmi viðeigandi reikniaðgerð þegar smellt er á einn þessara takka.

5.c. Punktar, strik, litir og `Graphics`

Nú skulum við líta á forrit sem er samsett úr fjórum klösum. Þeir eru `Myndhluti`, `Punktur`, `Strik` og `Mynd`. `Punktur` og `Strik` erfa `Myndhluta` og `Mynd` erfir `Applet` en notar hluti af tegundunum `Punktur` og `Strik` til að teikna mynd á skjáinn.

Klasinn `Myndhluti` hefur tvo eiginleika sem eru `bakgrlitur` (sá litur sem á að teikna hlut í til að hann renni saman við bakgrunninn og verði þar með ósýnilegur) og litur (sem er sá litur sem hlutur á að hafa ef hann er sýnilegur).

Klasinn `Mynd` notar enga hluti af tegundinni `Myndhluti`, en þar sem bæði `Punktur` og `Strik` erfa þennan klasa hafa hlutir af þeim tegundum þá tvo eiginleika sem hér um ræðir. Í hvert sinn sem smiðir klasanna `Punktur` og `Strik` eru framkvæmdir byrja þeir á að framkvæma smið yfirklasans (`Myndhluti`) og hann gefur litabreytunum gildi. `Punktur` og `Strik` geta svo breytt um lit með því að nota aðferðirnar

```
public void litur(int R, int G, int B)
```

og

```
public void bakgrlitur(int R, int G, int B)
```

Breyturnar `bakgrlitur` og `litur` eru af tegundinni `Color` og þær eru `protected` (en ekki `public`). Ef breyta eða aðferð er `protected` þá geta undirklasar og klasar í sama pakka notað hana en aðrir ekki.

Smiðurinn fyrir `Color` sem hér er notaður tekur við þrem tölum á bilinu 0 til 255. Sú fyrsta segir hvað á að vera mikið af rauðu, sú næsta hvað á að vera mikið af grænu og sú þriðja hvað á að vera mikið af bláu. Sé til dæmis gefin skipunin

```
litur = new Color(150, 0, 150);
```

verður litur blanda úr 150 hlutum af rauðu, engu grænu og 150 hlutum af bláu sem er einhvern veginn fjólublátt. Með því að setja 0 í öll sætin fæst svartur litur (ekkert ljós af neinum lit). Sé 255 í öllum sætum er liturinn hvítur og séu allar tölur jafnar og á milli 0 og 255 fæst einhver grátónn. Hér er hvítt bakgrunnslitur og allt sem er teiknað

í honum verður ósýnilegt. Ef notuð er vefsjá sem gefur `Applet`-um annan bakgrunnslit en hvítan þarf að breyta stillingunni á bakgrunnslit í smiðnum `Myndhluti`.[1]

Tegundin `Color` er í pakkanum `java.awt`. Þess vegna er

```
import java.awt.*;
```

haft efst.

```
import java.awt.*;
//
// forrit_05_02
// Myndhluti
//
public class Myndhluti
{
   protected Color bakgrlitur;
   protected Color litur;

   public Myndhluti()  // er framkvæmt af smið erfingja
   {
      bakgrlitur = new Color(255, 255, 255);
      litur = new Color(0, 0, 0);
   }

   public void litur(int R, int G, int B)
   {
      litur = new Color(R, G, B);
   }

   public void bakgrlitur(int R, int G, int B)
   {
      bakgrlitur = new Color(R, G, B);
   }
}
```

Klasinn `Punktur` hefur tvo eiginleika til viðbótar við þá sem hann erfir frá `Myndhluti`. Þetta eru x og y-hnit sem ráða staðsetningu punktsins. Hnitin x=0 og y=0 eru efst til vinstri. x-hnitin hækka svo til hægri og y-hnitin hækka niður á við. Hnitin x=100 og y=50 eru því 100 punktum hæga megin við og 50 punktum neðan við hornið efst til vinstri.

Taktu eftir aðferðunum

```
public void syna(Graphics g)
```

og

```
public void fela(Graphics g)
```

Þær taka við einum hlut af tegundinni `Graphics`. Þetta er hluturinn sem teiknað er á (myndflöturinn sem við sjáum á skjánum). `Graphics` er í pakkanum `java.awt`, þess vegna er

```
import java.awt.*;
```

haft efst.

[1] Sumar vefsjár hafa bakgrunnslitinn `Color(192, 192, 192)` sem er ljósgrátt.

Hlutir af tegundinni `Graphics` kunna margar aðferðir, þ.á.m.

```
public abstract[1] void setColor(Color c)
```

sem stillir litinn sem teiknað er með á myndflötinn;

```
public abstract void drawLine(int x1, int y1, int x2, int y2)
```

sem teiknar strik frá punktinum (x1, y1) í punktinn (x2, y2) og

```
public abstract void drawArc(int x, int y,
                 int width, int height,
                 int startAngle, int arcAngle)
```

sem teiknar boga (hring, ellipsu eða hluta af hring eða ellipsu). Boginn er látinn passa inn í ferhyrning sem hefur vinstra topphorn í (x, y), breiddina `width` og hæðina `height`, byrjar í horninu `startAngle` og dekkar `arcAngle` gráður. Skipunin

```
g.drawArc((int)x-2, (int)y-2, 4, 4, 0, 360);
```

teiknar boga á myndflötinn `g` og lætur hann falla inn í ferhyrning sem hefur vinstra topphorn 2 punktum ofan við og 2 punktum vinstra megin við (x, y), er 4 punktar á breidd og 4 á hæð, byrjar í stefnunni 0 og nær allan hringinn (þ.e. 360 gráður). Með öðrum orðum, skipunin teiknar hring með miðju í (x,y) og radíus 2.

Þar sem `x` og `y` eru af tegundinni `double` þarf að breyta þeim í `int` með `(int)x` og `(int)y` því `drawArc`-aðferðin tekur við gildum af tegundinni `int` en ekki `double`.

```
import java.awt.*;
//
// forrit_05_02
// Punktur
//
public class Punktur extends Myndhluti
{
   protected double x;
   protected double y;

   // Punktur hefur tvo smiði. Annar tekur ekki við neinum
   // gildum og býr til punkt með hnitin 0,0. Hinn tekur við tveim
   // kommutölum sem eru x og y hnit punktsins.

   public Punktur()
   {
      x = 0;
      y = 0;
   }

   public Punktur(double xhnit, double yhnit)
   {
      x = xhnit;
      y = yhnit;
   }

   // Punktur kann fjórar aðferðir. Þær fystu tvær breyta
   // staðsetningu hans. syna-aðferðin teiknar lítinn hring
   // þar sem punkturinn er og fela-aðferðin teiknar ofan í
   // hringinn í bakgrunnslit þannig að hann verður ósýnilegur.

   public void faera(double dx, double dy)
   {
```

[1] Aðferð sem er `abstract` er ekki skilgreind í klasanum sem hún tilheyrir heldur í undirklösum hans. Í kafla A verður útskýrt betur hvað `abstract` þýðir.

```
      x += dx;
      y += dy;
   }

   public void faeraTil(double xhnit, double yhnit)
   {
      x = xhnit;
      y = yhnit;
   }

   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawArc((int)x-2, (int)y-2, 4, 4, 0, 360);
   }

   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawArc((int)x-2, (int)y-2, 4, 4, 0, 360);
   }
}
```

Ef þú hefur skilið hvernig tegundin `Punktur` er hugsuð þá ætti `Strik` ekki að vefjast fyrir þér. `Strik` hefur aðeins tvo eiginleika sem báðir eru af tegundinni `Punktur`. Þetta eru upphafs- og endapunktur striksins.

```
import java.awt.*;
//
// forrit_05_02
// Strik
//
public class Strik extends Myndhluti
{
   protected Punktur p1, p2;

   public Strik(Punktur punktur1, Punktur punktur2)
   {
      p1 = punktur1;
      p2 = punktur2;
   }

   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
   }

   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
   }
}
```

Loksins kemur klasi sem gerir eitthvað við punkta og strik. Hann heitir `Mynd` og býr til mynd úr tveim punktum og einu striki.

Þessi klasi notar `paint`-aðferðina sem kynnt var í kafla 1.c. Titillína hans er

```
   public void paint(Graphics g)
```

Áður hefur verið minnst á `init`-aðferðina sem hlutir af tegundinni `Applet` (og undirtegundum hennar) framkvæma strax og þeir verða til. Hlutir af þessum tegundum (þ.e. Applet) framkvæma 4 aðrar aðferðir sjálfkrafa. Þær eru

```
public void start()
public void stop()
public void destroy()
```

og

```
public void paint(Graphics g)
```

Hér hugsum við ekkert um þær þrjár fyrsttöldu. Sú síðastnefnda er framkvæmd í hvert sinn sem `Applet` sýnir eða uppfærir mynd sína á skjánum. Aðferðin tekur við einum hlut sem er myndflötur (flöturinn sem sést á skjánum).

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_05_02
// Mynd
//
public class Mynd extends Applet
{
   Punktur p1, p2;
   Strik s1;

   public void init()
   {
      p1 = new Punktur(25, 25);
      p2 = new Punktur(125, 125);
      s1 = new Strik(p1, p2);
      p1.litur(255, 0, 0); // Rauður
      p2.litur(0, 255, 0); // Grænn
      s1.litur(0, 0, 255); // Blár
   }

   public void paint(Graphics g)
   {
      p1.syna(g); p2.syna(g); s1.syna(g);
   }
}
```

Verkefni 5.2 *
Búðu til forrit sem notar hluti af tegundunum `Punktur` og `Strik` til að teikna fernING.

Verkefni 5.3 *
Bættu klasanum `Hringur` við forritið hér að framan (forrit_05_02). Láttu hann hafa tvo eiginleika (auk þeirra sem hann erfir frá `Myndhluti`)

```
protected Punktur midpunktur;
protected double radius;
```

einn smið

```
public Hringur(Punktur p, double r)
```

og tvær aðferðir:

```
public void syna(Graphics g)
public void fela(Graphics g)
```

Breyttu svo klasanum `Mynd` þannig að hann noti `Hringur` til að teikna hring á skjáinn.

Verkefni 5.4
Bættu við klasann `Hringur` aðferð til að stilla radíus hrings. Titillínan getur verið:

```
public void stillaRadius(double nyrRadius)
```

Breyttu svo klasanum `Mynd` þannig að hann noti endurtekningu (t.d. með `for`) til að teikna 10 sammiðja hringi eins og myndast þegar steini er hent í vatn.

5.d. Aðgangur að hlutum, breytum og aðferðum

Klasar sem við höfum skoðað til þessa hafa verið `public`. Sé orðið `public` framan við heiti klasa í titillínu hans þá mega allir hlutir af öllum tegundum nota hann. Sé þessu orði sleppt mega aðeins klasar sem eru hluti af sama pakka nota hann. Forritin í þessum kafla eru úr nokkrum klösum en ef þeir eru allir hluti af sama pakka hefur það engin áhrif þótt `public` sé sleppt úr titillínu annarra tegunda en þeirra sem vefsjáin á að keyra (þ.e. þeirra sem erfa frá `Applet`).

Aðgangur að klasa getur aðeins verið á tvo vegu. Hann getur verið `public` eða ekki `public`. Séu margir klasar saman í skrá þegar forrit er skrifað má aðeins einn þeirra vera `public` og hann verður að heita sama nafni og skráin (að frátöldu eftirnafninu java). Klasar sem eru saman í skrá tilheyra sama pakka og geta því notað hver annan hvort sem þeir eru `public` eða ekki.[1]

`public` er ekki eina orðið sem hægt er að setja framan við heiti klasa í titillínu hans. Þar geta líka komið

`abstract`	Þýðir að klasinnn innihaldi aðferðir sem eru skilgreindar í undirtegundum. Ekki er hægt að búa til hluti af tegund sem er `abstract`.
`final`	Þýðir að ekki sé hægt að búa til undirklasa af þessari tegund.

Um `abstract` og `final`-klasa verður fjallað í A. kafla.

Aðgengi að eiginleikum (breytum) og aðferðum getur verið á fjóra vegu.

`public`	Allir hlutir af öllum tegundum geta látið hlut framkvæma aðferðina og haft aðgang að eiginleikanum.
`protected`	Allir hlutir sem tilheyra sömu tegund eða undirtegund eða tegund í sama pakka geta látið hlut framkvæma aðferðina og haft aðgang að eiginleikanum. Sé ekkert tiltekið um aðgang er hann bundinn við tegundir í sama pakka. Aðferðir og breytur sem hafa þetta aðgengi eru kallaðar „vinsamlegar" (á ensku: „friendly").
`private`	Aðgangur einskorðaður við hluti af sömu tegund.

Breyturnar `teljari` og `nefnari` í klasanum `Brot` eru `public` svo hlutir af öðrum tegundum hafa aðgang að þeim. Þetta þýðir að ef klasi hefur breytu sem heitir `x` og er

[1] Um pakka er fjallað í viðauka II.

af tegundinni `Brot` þá getur hann innihaldið skipun sem nær í gildi `x.teljari` eins og t.d.

```
int t = x.teljari
```

og líka breytt gildi `x.teljari` með skipun eins og

```
x.teljari = t + 5;
```

Þessar skipanir væri ekki hægt að gefa ef `teljari` væri `private`.

Auk orðanna `public`, `protected` og `private` sem stjórna aðgengi að eiginleikum og hlutum er hægt að setja ýmis önnur fyrir framan heiti hluta og eiginleika þegar þeir eru skilgreindir. Þau helstu eru:

`abstract`	Getur aðeins átt við aðferðir, ekki eiginleika. Þýðir að klasinn innihaldi aðeins titillínu aðferðarinnar, skilgreining á hvað hún geri sé í undirklasa. Klasi sem inniheldur `abstract` aðferð verður sjálfur að vera `abstract`.
`final`	Ef breyta er final er ekki hægt að breyta innihaldi hennar. Ef aðferð er `final` getur aðferð í undirklasa ekki yfirskyggt hana.
`static`	Ef breyta er `static` þá er ekki búið til eitt eintak fyrir hvern hlut af tegundinni heldur eru allir hlutir af þessari tegund saman um eitt eintak af breytunni. Breyti einn hlutur innihaldi hennar þá breytist það fyrir alla hlutina. Ef aðferð er `static` þá geta einstakir hlutir ekki framkvæmt hana heldur aðeins klasinn sem heild. Aðferð sem er `static` er framkvæmd með því að setja heiti klasa framan við hana en ekki heiti einstaks hlutar.

5.e. Klasabreytur, staðværar breytur og færibreytur

Lítum á aðferðina `plus` í klasanum `Brot`. Klasinn hefur tvær breytur (`teljari` og `nefnari`) sem allar aðferðir hans geta notað og þar sem þær eru public geta hlutir af öðrum tegundum líka gefið þessum breytum gildi og sótt innihald þeirra. En inni í aðferðinni `plus` (sem er sýnd hér að neðan) eru þrjár breytur sem heita `t`, `n` og `utkoma`. Þessar breytur eru lokaðar inni í aðferðinni. Aðrar aðferðir geta ekki notað þær. Þær eru staðværar. Breyturnar `teljari` og `nefnari` eru hins vegar klasabreytur og tilheyra ekki einstakri aðferð heldur öllum klasanum.

```
public Brot plus(Brot b)
{
   int t, n;
   Brot utkoma;
   t = teljari*b.nefnari + b.teljari*nefnari;
   n = nefnari*b.nefnari;
   utkoma = new Brot(t, n);
   utkoma.stytta();
   return utkoma;
}
```

Auk staðværu breytanna `t`, `n`, og `utkoma` hefur aðferðin `plus` færibreytu sem heitir `b` og er af tegundinni `Brot`. Færibreytur eru skilgreindar innan sviga aftan við heiti aðferðar og þeim er gefið gildi í hvert sinn sem aðferð er framkvæmd með því að setja það innan sviganna aftan við nafnið. Tökum sem dæmi að við leggjum saman brotin 2/3 og 3/5 með skipununum.

```
Brot x, y, svar;     // Breytur skilgreindar.
x = new Brot(2, 3);  // Brotið 2/3 búið til.
y = new Brot(3, 5);  // Brotið 3/5 búið til.
```

```
svar = x.plus(y);    // Brotin lögð saman.
```

Síðasta skipunin leggur brotin saman. Brotið `x` framkvæmir aðferðina `plus` og færibreytunni `b` í `plus`-aðferðinni er gefið gildið sem `y` hefur, nefnilega 3/5, því `y` er sett innan sviga fyrir aftan heiti aðferðarinnar.

Ef aðferðin foo er skilgreind svona:

```
public int foo(int a, int b, int c)
{
   int d;
   d = a*b-c;
   return d*d;
}
```

og gefnar skipanirnar

```
int d = 5;
int a = 2
int x;
x = foo(a+1, d, 7);
```

þá fá færibreyturnar `a`, `b` og `c` í `foo`-aðferðinni gildin 3, 5 og 7. Staðværa breytan `d` fær gildið 8 og útkoman sem lendir í `x` verður 64. Eftir að síðasta skipunin hefur verið framkvæmd hefur breytan `d` sem skilgreind var með

```
int d = 5;
```

ennþá gildið 5 því staðværa breytan `d` í aðferðinni `foo` hefur engin áhrif á hana.

Breyturnar `d` og `a` sem koma fyrir í skipununum hér að ofan hafa engin tengsl við samnefndar breytur í aðferðinni `foo`. Þótt staðværar breytur og færibreytur heiti ef til vill sömu nöfnum og einhverjar klasabreytur er þetta tvennt ólíkt.

Sé færibreyta ekki af einfaldri tegund heldur tegund sem vísar á hlut þá er dálítið annað uppi á teningnum. Hugsum okkur að aðferðin goo sé svona:

```
public int goo(Object o)
{
   // Hér koma skipanir sem breyta ástandi o.
}
```

og gefnar skipanirnar

```
Object x;
x = new Object();
goo(x);
```

þá fær færibreytan `o` sama gildi og breytan `x`. En breytan `x` inniheldur ekki `Object` heldur tölu (vistfang) sem segir hvar `Object`-ið er staðsett í minni tölvunnar. Meðan aðferðin `goo` er framkvæmd geyma færibreytan `o` og breytan `x` sama vistfang og vísa því á sama hlutinn. Ef `goo` breytir þessum hlut þá er það sem `x` vísar á breytt eftir að aðferðin hefur verið framkvæmd.

Þar sem klasabreytur eru skilgreindar er hægt að taka fram hvort þær skuli vera `public`, `protected` eða `private` eða hafa þær vinsamlegar með því að taka ekkert fram um aðgengi að þeim. Þetta gildir ekki um staðværar breytur og færibreytur. Orðin `public`, `private` og `protected` eiga ekki erindi framan við skilgreiningar á þeim. Staðværar breytur og færibreytur eru alltaf lokaðar inni í sinni aðferð og óaðgengilegar öllum öðrum aðferðum.

Það þykir heldur slæmur siður að hafa klasabreytur `public`. Oftast er betra að aðgangur að þeim sé takmarkaður. Um það verður fjallað nánar í næsta kafla.

5.f. Atburðir og Mús

Þau forrit sem fjallað hefur verið um til þessa hafa aðeins brugðist við einni tegund atburða sem eru `action`-atburðir sem verða t.d. þegar smellt er með músinni á takka (`Button`).

Í Java eru atburðir hlutir af tegundinni `Event` og sú tegund er í pakkanum `java.awt`. Tegundin `Event` hefur ýmsa eiginleika (klasabreytur). Þeir mikilvægustu eru `id` og `target`. Sá fyrrnefndi geymir gildi sem segir hvers konar atburð er um að ræða og sá síðarnefndi vísar á hlutinn sem fyrir atburðinum varð. Hlutir eins og takkar (`Button`) eða `Applet` sem verða fyrir atburðum framkvæma aðferðina

```
public boolean handleEvent(Event evt)
```

og senda henni atburðinn sem þeir urðu fyrir.

Þessi aðferð tilheyrir sameiginlegum forföður `Button`, `Applet` og margra fleiri tegunda. Þessi sameiginlegi forfaðir er tegundin `Component`. `handleEvent`-aðferðin framkvæmir svo `action`-aðferðina ef

```
evt.id == ACTION_EVENT
```

Hafi hluturinn sem fyrir atburðinum varð sína eigin `action`-aðferð þá er hún framkvæmd (enda yfirskyggir hún `action`-aðferð yfirklasa) annars er `action`-aðferð yfirklasa framkvæmd.

Til eru margar aðrar gerðir atburða en `ACTION_EVENT`, t.d. `WINDOW_MOVE` og `KEY_PRESS` og `MOUSE_DOWN`. Atburðir af þessum gerðum verða til þegar gluggi er hreyfður á skjánum, ýtt er á lykil á lyklaborði og músartakka ýtt niður.

Í hvert sinn sem músin er hreyfð eða ýtt er á hnappa hennar verða til atburðir. Músaratburðir eru af sex gerðum.

`MOUSE_DOWN`	Gerist þegar ýtt er á takka á músinni. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseDown(Event evt, int x, int y)`
`MOUSE_UP`	Gerist þegar takka á músinni er sleppt. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseUp(Event evt, int x, int y)`
`MOUSE_DRAG`	Gerist þegar mús er dregin með takka niðri. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseDrag(Event evt, int x, int y)`
`MOUSE_ENTER`	Gerist þegar músarbendill fer inn á Applet. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseEnter(Event evt, int x, int y)`

`MOUSE_EXIT`	Gerist þegar músarbendill yfirgefur Applet. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseExit(Event evt, int x, int y)`
`MOUSE_MOVE`	Gerist þegar mús er hreyfð án þess að takki sé niðri. Ef `evt.id` hefur þetta gildi kallar `handleEvent`-aðferðin á `public boolean mouseMove(Event evt, int x, int y)`

Breyturnar `x` og `y` fá tölur sem eru `x` og `y` hnit músarbendilsins.

Lyklaborðsatburðir eru af nokkrum gerðum. Hér skal aðeins getið einnar

`KEY_PRESS`	Gerist þegar ýtt er á hnapp á lyklaborðinu. Ef `evt.id` hefur þetta gildi kallar `handleEvent` aðferðin á `public boolean keyDown(Event evt, int key)`

Til að klasi bregðist við atburðum þarf hann að innihalda aðferðir sem taka við atburðunum. Eigi klasi til dæmis að bregðast við þegar ýtt er á músartakka þarf hann að hafa `mouseDown`-aðferð. Aðferðirnar sem taka við atburðunum skila gildi af tegundinni `boolean`. Eðlilegast er að útkoman úr þeim sé `true` ef þær bregðast á viðhlítandi hátt við atburði, `false` ef þær gera það ekki.

Þú getur prófað að nota `mouseDrag`- og `keyDown`-aðferðirnar með því að setja eftirfarandi í stað tegundarinnar `Mynd` í forrit_05_02. `keyDown`-aðferðin sér til þess að strikið litast rautt, grænt eða blátt þegar slegið er á `r`, `g` eða `b` á lyklaborðinu.[1]

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_05_02
// Mynd2
//
public class Mynd2 extends Applet
{
   Punktur p1, p2;
   Strik s1;
   Graphics g;

   public void init()
   {
      p1 = new Punktur(25, 25);
      p2 = new Punktur(125, 125);
      s1 = new Strik(p1, p2);
      p1.litur(255, 0, 0);          // Rauður
      p2.litur(0, 255, 0);          // Grænn
      s1.litur(0, 0, 255);          // Blár
      g = this.getGraphics();       // getGraphics() skilar
   }                                // myndfleti hlutar.

   public void paint(Graphics g)
   {
      p1.syna(g); p2.syna(g); s1.syna(g);
   }
```

[1] Sumar útgáfur forritsins *Appletviewer* (sem fylgir *JDK* frá *Sun Microsystems*) eru óþægar við að grípa áslátt á lyklaborð og geta því ekki keyrt klasann `Mynd2` í forrit_05_02 með viðhlítandi hætti. Þeir sem nota *Appletviewer* geta því þurft að keyra þennan eina klasa í *Internet Explorer* eða *Netscape Navigator*.

```
   public boolean keyDown(Event e, int key)
   {
      if ((char)key == 'r')
      {
         s1.litur(255, 0, 0);
         s1.syna(g);
         return true;
      }
      else if ((char)key == 'g')
      {
         s1.litur(0, 255, 0);
         s1.syna(g);
         return true;
      }
      else if ((char)key == 'b')
      {
         s1.litur(0, 0, 255);
         s1.syna(g);
         return true;
      }
      return false;
   }

   public boolean mouseDrag(Event e, int x, int y)
   {
      p2.fela(g); s1.fela(g);
      p2.faeraTil(x, y);
      p2.syna(g); s1.syna(g);
      return true;
   }
}
```

Ef eftirfarandi er sett í staðinn fyrir `Mynd` í forrit_05_02 er komið heilt teikniforrit, hvorki meira né minna.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_05_02
// Mynd3
//
public class Mynd3 extends Applet
{
   Punktur p1, p2;
   Strik s1;
   Graphics g;

   public void init()
   {
      g = this.getGraphics();
   }

   public boolean mouseDown(Event e, int x, int y)
   {
      p1 = new Punktur(x, y);
      p1.syna(g);
      return true;
   }
```

```
   public boolean mouseUp(Event e, int x, int y)
   {
      p2 = new Punktur(x, y);
      p2.syna(g);
      s1 = new Strik(p1, p2);
      s1.syna(g);
      return true;
   }
}
```

Verkefni 5.5 *
Keyrðu klasann `Mynd2` úr forrit_05_02 og áttaðu þig á því hvernig hann virkar. Bættu svo við `Mynd2` möguleika á að lita strik fjólublátt með því að ýta á f.

Verkefni 5.6 *
Keyrðu klasann `Mynd3` úr forrit_05_02 og áttaðu þig á því hvernig hann virkar. Breyttu honum svo þannig að strikin á myndinni verði rauð.

Verkefni 5.7 *
Settu eftirfarandi aðferðir í stað `mouseDown` og `mouseUp` í `Mynd3` og kannaðu hvaða áhrif það hefur.

```
   public boolean mouseDown(Event e, int x, int y)
   {
      p1 = new Punktur(x, y);
      p2 = new Punktur(x, y);
      s1 = new Strik(p1, p2);
      s1.syna(g);
      return true;
   }

   public boolean mouseDrag(Event e, int x, int y)
   {
      s1.fela(g);
      p2.faeraTil(x, y);
      s1.syna(g);
      return true;
   }
```

Verkefni 5.8
Breyttu lausninni á verkefni 5.7 þannig að strik séu í daufum lit, t.d gráum, meðan þau eru að teiknast en verði svört um leið og músinni er sleppt.

Verkefni 5.9
Búðu til forrit sem byrjar á að teikna einn hring og færir hann svo í hvert sinn sem smellt er með músinni þannig að hann lendi á þeim stað sem músin bendir á. Forritið á að nota tegundirnar `Myndhluti`, `Punktur` og `Hringur` (sem var lausn á verkefnum 5.3 og 5.4) og `mouseDown`-aðferðina.

Verkefni 5.10
Bættu við teikniforritið hér að ofan (þ.e. klasann `Mynd3`) þannig að það geti bæði teiknað strik og hringi. (Ábending: Hafðu tvo takka sem heita `bStrik` og `bHringur`. Láttu teiknast strik ef síðast var ýtt á `bStrik` og hring ef síðast var ýtt á `bHringur`.) Hafðu alla hringi jafnstóra, t.d með radíus 25.

Verkefni 5.11
Bættu við lausnina á verkefni 5.10 þannig að hægt sé að stjórna því hvað hringir hafa stóran radíus. (Ábending: Láttu einn punkt verða til þar sem músartakka er ýtt niður og annan þar sem honum er sleppt. Hafðu fyrri punktinn fyrir miðpunkt hringsins og bilið milli punktanna fyrir radíus hans.)

5.x. Spurningar og umhugsunarefni

1. Hvað heitir klasinn sem allir aðrir Java klasar erfa?
2. Ef klasi heitir `Brot` hvað heitir þá smiður hans?
3. Hvaða hlutverki þjóna aðferðir sem heita `toString`?
4. Hvað merkja orðin `this` og `super`?
5. Gerðu ráð fyrir að aðferðin `flatarmal` sé skilgreind svona:

```
public double flatarmal(double a, double b)
{
   return a * b;
}
```

   Hvaða gildi fær `x` ef þessi skipun er gefin?

```
double x = flatarmal(5, 7)
```

6. Hvað merkja orðin `return` og `void`?
7. Hvað merkja orðin `private`, `protected` og `public`?
8. Hvað merkja orðin `final` og `static`?
9. Hvaða litir eru geymdir í breytunum `a`, `b` og `c` eftir að þessar skipanir hafa verið gefnar?

```
Color a = new Color(0, 0, 0);
Color b = new Color(0, 128, 128);
Color c = new Color(255, 255, 255);
```

10. Hvað heita yfirklasar klasanna `Hattur` og `Fattur` ef titillínur þeirra eru svona?

```
public class Hattur
public class Fattur extends Hattur
```

11. Hvaða munur er á staðværri breytu og klasabreytu?
12. Gerðu ráð fyrir að aðferðin `boo` sé skilgreind svona:

```
public int boo(int a)
{
   int b;
   a += 1
   b = a * a;
   return b;
}
```

   Hvaða gildi fá `a`, `b`, og `c` ef þessar skipanir eru gefnar?

```
int a = 5;
int b = 7;
int c = boo(a);
```

13. Undir hvaða kringumstæðum verður til hlutur af tegundinni `Event`?
14. Hvaða hlutverki þjónar aðferðin `handleEvent` í klasanum `Component`?
15. Til hvers eru aðferðirnar `action`, `mouseDown`, `mouseUp`, `mouseDrag` og `keyDown`?

6. kafli: Fylki

6.a. Fylki skilgreind

Fylki er breyta sem inniheldur marga sams konar hluti, t.d. margar kommutölur, marga strengi eða marga takka.

Breytu sem heitir `x` og inniheldur eina kommutölu er hægt að skilgreina með

```
double x;
```

Eigi breytan hins vegar að innihalda fylki af kommutölum er hún skilgreind með

```
double[] x;
```

eða

```
double x[];
```

Ekki skiptir máli hvor aðferðin er notuð.

Fylki þarf að búa til með skipuninni `new`. Að þessu leyti eru þau eins og hlutir en ólík einföldum tegundum á borð við `double`, `char` og `int`. Eigi fylkið `x` t.d. að rúma 10 kommutölur er það búið til með skipuninni

```
x = new double[10];
```

Eftir þetta hefur `x` 10 hólf sem eru númeruð frá 0 til 9 og geta hvert um sig geymt eina kommutölu. Til að setja kommutöluna 3,7 í hólf númer 6 er hægt að gefa skipunina

```
x[6] = 3.7;
```

Til að setja tvöfalt stærri tölu í hólf númer 7 má nota

```
x[7] = 2 * x[6];
```

Ef fylki heitir x þá heita hólfin í því `x[0]`, `x[1]`, `x[2]` o.s.frv. Séu aðeins 10 hólf í fylkinu þá er síðasta hólfið númer 9 og það má ekki vísa í hólf með hærra númeri. Í þessu tilviki má t.d. ekki gefa skipun á borð við

```
x[14] = 5;
```

Hægt er að búa til fylki af hvaða tegund sem er t.d. fylki af tegundinni `Punktur` sem skilgreind var í síðasta kafla. Eftirfarandi skipanir búa til fylki af 25 punktum og setja punktinn (25, 75) í 8. hólf þess (þ.e. hólf nr. 7).

```
Punktur[] p;
p = new Punktur[25];
p[7] = new Punktur(25, 75);
```

Hægt er að gefa fylki gildi um leið og það er skilgreint með því að telja upp innan slaufusviga þau gildi sem eiga að fara í hólf fylkisins svona:

```
int[] n = {5, 3+4, 9}
```

Hér fær fylkið stærðina 3. Hólf númer 0 fær gildið 5, hólf númer 1 gildið 7 og þriðja hólfið sem er númer 2 fær gildið 9.

Eftirfarandi forrit hefur tvö fylki. Annað rúmar 5 kommutölur. Hitt hefur pláss fyrir 3 strengi.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_06_01
// Fylki
//
public class Fylki extends Applet
{
   TextArea t;
   double[] talnafylki;
   String[] strengjafylki;
   int i;

   public void init()
   {
      t = new TextArea(5,25);
      this.add(t);

      // Hér er búið til fylki með 5 tölum.
      talnafylki = new double[5];

      // Hér er búið til fylki af 3 strengjum
      strengjafylki = new String[3];

      talnafylki[0] = 2;      //Fyrsta sætið er nr. 0
      for (i=1; i < 5; i++)
      {
         talnafylki[i] = 2*talnafylki[i-1];
      }
      for(i=0; i < 5; i++)
      {
         t.appendText("Tala nr. " + i + " er: " +
                       talnafylki[i] + "\n");
      }

      t.appendText("\n");

      strengjafylki[0] = new String("Hani");
      strengjafylki[1] = new String("Svín");
      strengjafylki[2] = new String(strengjafylki[1] +
                                    " og " + strengjafylki[0]);
      for(i = 2; i >= 0; i--)
      {
         t.appendText("Strengur nr. " + i + " er: " +
                        strengjafylki[i] + "\n");
      }
   }
}
```

Verkefni 6.1 *
Búðu til forrit sem setur tölurnar 5, 10, 15, 20, ... , 95, 100 í fylki og skrifar svo allt innihald fylkisins í textaglugga (`TextArea`).

Verkefni 6.2 *
Búðu til forrit sem hefur eitt `TextField`, eitt `TextArea` og tvo takka sem eru merktir „Bæta við nafni" og „Skrifa nöfn". Sé skrifað nafn í `TextField` og ýtt á fyrri takkann á forritið að bæta nafninu í fylki af strengjum. Sé ýtt á seinni takkann á forritið að skrifa öll nöfnin í fylkinu í `TextArea`.

Forritin í þessum kafla nota eingöngu einvíð fylki. Rétt er þó að geta þess að í Java er hægt að skilgreina fylki með tvær eða fleiri víddir. Eftirfarandi skipanir skilgreina og búa til tvívítt fylki af kommutölum með 3 sinnum 7 hólf og setja töluna 0,5 í hólfið sem er í línu númer 1 og dálki númer 4.

```
double[][] d;
d = new double[3][7];
d[1][4] = 0.5;
```

Eðlilegast er að hugsa sé einvítt fylki sem runu af hólfum, svona:

				0,5		

Einvítt fylki með 7 hólfum (sem númeruð eru frá 0 til 6). Talan 0,5 er í hólfi númer 4.

Tvívítt fylki má hugsa sér sem töflu (margar runur sem staflað er lóðrétt) svona:

				0,5		

Tvívítt fylki með 3 sinnum 7 hólfum og töluna 0,5 í hólfi númer [1][4].

Hægt er að gera sér mynd af þrívíðu fylki sem runu af tvívíðum fylkjum sem staflað er hverju aftan við annað. Erfiðara er að gera sér mynd af fjórvíðu fylki en það er samt hægt að skilgreina slíkt fylki og gefa einstökum hólfum þess gildi, t.d. svona:

```
double[][][][] d;
d = new double[3][7][11][2];
d[1][4][2][1] = 0.5;
```

6.b. Breytan length

Í Java eru fylki hlutir. Þau eru búin til með skipuninni `new` og þau erfa aðferðir og eiginleika klasans `Object`. Fylki hafa eina klasabreytu sem heitir `length` og inniheldur heiltölu sem segir hvað er mikið pláss (mörg hólf) í fylkinu. Fylki eru frábrugðin öðrum tegundum hluta m.a. að því leyti að þau hafa engan smið heldur eru búin til með því að nefna tegundina sem þau eiga að innihalda og setja þar fyrir aftan tölu innan hornklofa sem segir hvað á að vera rúm fyrir marga hluti í fylkinu.

Til að gera eitthvað við hvert hólf í fylki er yfirleitt notuð `for`-slaufa og staðnæmst þegar kemur að hólfi númer `length`. Ef `p` er t.d. fylki af tegundinni `Punktur` sem skilgreind var í 5. kafla þá setur þessi slaufa punkt í sérhvert hólf þess og séu punktarnir teiknaðir á skjáinn mynda þeir sínusferil.

```
for (int i = 0; i < p.length; i++)
{
   p[i] = new Punktur(i*2, 100+50*Math.sin((double)i/5));
}
```

Það er svo hægt að sýna alla punktana t.d. með því að hafa þessa slaufu innan í `paint`-aðferðinni

```
for (int i = 0; i < p.length; i++)
{
   p[i].syna(g);
}
```

Breytan `length` geymir tölu sem segir hve mörg pláss eru í fylki, ekki hve mörg þeirra eru notuð. Stundum eru aðeins hlutir í sumum hólfum, hin innihalda þá `null`. Eigi að sýna alla punkta í fylki, `p`, þar sem punktar eru í sumum hólfum en sum eru tóm er hægt að nota

```
for (int i = 0; i < p.length; i++)
{
   if (p[i] != null)
   {
      p[i].syna(g);
   }
}
```

Verkefni 6.3 *
Búðu til forrit sem notar fylki af 100 punktum til að teikna ferilinn y = sin(x). (Láttu x taka gildi frá $-\pi$ til π.)

Verkefni 6.4
Breyttu lausninni á verkefni 6.3 þannig að forritið teikni ferilinn $y = x^2$. (Láttu x taka gildi frá -2 til 2.)

6.c. Reikningur með talnafylki, staðværar breytur og `StringTokenizer`

Hér á eftir er forrit sem reiknar meðaltal af tölum sem eru skrifaðar í `TextArea`. Til að setja tölurnar í fylkið er innihald textaglugga fyrst sett í streng og strengnum svo breytt í hlut af tegundinni `StringTokenizer`. Skipanirnar sem sjá um þetta eru

```
TextArea tTalnasafn;
String sTalnasafn;
StringTokenizer st;
sTalnasafn = tTalnasafn.getText();
st = new StringTokenizer(sTalnasafn, ", \n\r\t");
```

Breytan `tTalnasafn` er klasabreyta því hún er notuð af meira en einni aðferð (nefnilega bæði af `init`-aðferðinni og `action`-aðferðinni). Hinar breyturnar (`sTalnasafn` og `st`) eru staðværar inni í `action`-aðferðinni enda eru þær ekki notaðar utan þeirrar aðferðar. Breytur sem aðeins eru notaðar af einni aðferð er best að skilgreina inni í þeirri aðferð. Þær eru þá lokaðar inni í aðferðinni og það gerir ekkert til þó aðrar aðferðir hafi breytur með sömu nöfnum.

Skipunina sem breytir `tTalnasafn` í streng þarf ekki að skýra. En síðasta línan er ekki eins auðskilin. Hún breytir `sTalnasafn` í hlut af tegundinni `StringTokenizer` sem er raunar runa af aðskildum bútum úr strengnum. Smiðurinn sem notaður er tekur við tveim gildum. Það fyrra er strengurinn `sTalnasafn`. Það seinna er strengurinn ", \n\r\t", þ.e. strengur sem inniheldur kommu, bil og tákn fyrir línuskipti, vendi og

dálk. Þessi seinni strengur segir hvernig á að klippa fyrri strenginn (`sTalansafn`) í búta. Hann er bútaður niður með því að klippa öll tákn úr seinni strengnum úr honum.

Ef gefnar eru skipanirnar

```
String s, partur1, partur2;
StringTokenizer st;
s = "Hani krummi hundur svín";
st = new StringTokenizer(s, "nu");
partur1 = st.nextToken();
partur2 = st.nextToken();
```

þá verður `st` runa af eftirtöldum strengjum

"Ha" "i kr" "mmi h" "d" "r svi"

Bútarnir verða til með því að táknin n og u eru klippt úr strengnum. Breytan `partur1` fær gildið "Ha" og `partur2` fær gildið "i kr" því aðferðin `nextToken` tekur bút framan af `st` og skilar honum.

Hér kemur svo forritið sem reiknar meðaltal.

```
//
import java.applet.Applet;
import java.awt.*;
import java.util.StringTokenizer;
//
// forrit_06_02
// Medaltal
//
public class Medaltal extends Applet
{
   TextField tMedaltal;
   TextArea tTalnasafn;
   Button bReikna;
   double[] talnasafn;     //Fylki af tölum
   int fjoldiTalna;
   double summa, medaltal;

   public void init()
   {
      tMedaltal = new TextField(25);
      tTalnasafn = new TextArea(5,25);
      bReikna = new Button("Reikna meðaltal");

      //Hér er búið til 100 talna fylki með
      //sæti nr. 0 til 99.
      talnasafn = new double[100];

      this.add(tTalnasafn);
      this.add(bReikna);
      this.add(tMedaltal);
   }

   public boolean action(Event e, Object o)
   {
      int i;
      String sTalnasafn;
      StringTokenizer st;
      String sTala;
      Double Tala;

      if (e.target == bReikna)
      {
         sTalnasafn = tTalnasafn.getText();
```

```
            // Nú er búið að setja innihald tTalnasafn
            // í einn streng. Næst þarf að tína tölurnar úr
            // strengnum og setja í fylkið talnasafn.
            // Þetta er gert með hlut af teg. StringTokenizer.
            // Hann klippir strenginn í búta með því að
            // skipta honum þar sem eitthvert af táknunum
            // í strengnum ", \n\r\t" kemur fyrir. Öllum
            // þessum táknum er hent um leið og strengnum
            // er skipt í búta.

            st = new StringTokenizer(sTalnasafn, ", \n\r\t");

            // Í slaufunni notar st aðferðirnar hasMoreTokens,
            // sem skilar true ef eitthvað er eftir í st, og
            // nextToken sem klípur fremsta bútinn af st og
            // skilar á formi strengs.

            fjoldiTalna = 0;
            while (st.hasMoreTokens())
            {
               sTala = new String(st.nextToken());
               Tala = new Double(sTala);
               talnasafn[fjoldiTalna] = Tala.doubleValue();
               fjoldiTalna++; //Hækkar um 1 í hvert sinn
            }                 //sem tala bætist í fylkið.

            summa = 0;

            // Nú eru allar tölurnar í fylkinu lagðar saman.
            for(i = 0; i < fjoldiTalna; i ++)
            {
               summa += talnasafn[i];
            }

            medaltal = summa/fjoldiTalna;
            tMedaltal.setText("Meðaltalið er " + medaltal);
            return true;
         }
         return false;
      }
}
```

Taktu eftir `while`-slaufunni sem tínir alla bútana úr `st`, breytir hverjum bút í tölu og setur töluna í fylkið `talnasafn`. Aðferðin `hasMoreTokens` skilar gildinu `true` meðan eitthvað er enn eftir í `StringTokenizer`.

Verkefni 6.5 *
Notaðu klasann `Medaltal` (forrit_06_02) sem fyrirmynd og búðu til forrit sem tekur við tölum í `TextArea`, setur þær allar í fylki og reiknar summuna af öðru veldi þeirra.

Verkefni 6.6
Búðu til forrit sem tekur við mörgum tölum og reiknar meðalfrávik (þ.e. meðalfjarlægð þeirra frá meðaltali $= \frac{\sum |x - \bar{x}|}{N}$)
Til að gera þetta þarf fyrst að reikna meðaltal allra talnanna. Til að finna fjarlægðina frá meðaltalinu þarf að reikna tölugildið af mismuni hverrar tölu og meðaltalsins. Hægt er að nota `Math.abs`-aðferðina til að reikna tölugildi.

Verkefni 6.7

Búðu til forrit sem tekur við mörgum tölum og reiknar fervik þeirra (þ.e. $\frac{\sum (x - \bar{x})^2}{N}$)

6.d. Tegundin `Talnasafn`, `private` eiginleikar og `this`

Það sem eftir er af kafla 6, sem og stór hluti af köflum 7 og 8, snýst að miklu leyti um eitt verkefni, að búa til forrit sem vinnur tölfræðilega útreikninga. Forritið í heild er í 8. kafla.

Til að smíða slíkt forrit er best að byrja á að búa til klasa sem innheldur safn af tölum og kann að beita á sig tölfræðilegum aðferðum eins og að finna fjölda talna, reikna summu, meðaltal, meðalfrávik, fervik, staðalfrávik, miðgildi, spönn og ef til vill fleira.

Klasinn sem hér fer á eftir inniheldur aðeins þrjár fyrsttöldu aðferðirnar en þar að auki hefur hann tvo smiði og eina aðferð til að taka margar tölur úr streng og setja í safnið.

```
import java.util.StringTokenizer;
//
// forrit_06_03
// Talnasafn
//
public class Talnasafn
{
   private double[] t;
   private int fjoldiTalna;
   private int hamarksfjoldiTalna = 100;

   public Talnasafn()         // Býr til talnasafn með pláss fyrir
   {                          // hamarksfjoldiTalna, sem er 100.
      t = new double[hamarksfjoldiTalna];
      fjoldiTalna = 0;
   }

   public Talnasafn(int h)    // Býr til talnasafn með pláss
   {                          // fyrir h tölur.
      hamarksfjoldiTalna = h;
      t = new double[hamarksfjoldiTalna];
      fjoldiTalna = 0;
   }

   public void setjaTolurISafn(String s, String d)
   {                          // Breytir streng
      StringTokenizer st;     // í fylki af
      String sTala;           // kommutölum.
      Double Tala;

      fjoldiTalna = 0;
      st = new StringTokenizer(s, d);
      while ((st.hasMoreTokens()) &&
            (fjoldiTalna < hamarksfjoldiTalna))
      {
         sTala = new String(st.nextToken());
         Tala = new Double(sTala);
         t[fjoldiTalna] = Tala.doubleValue();
         fjoldiTalna++;
```

```
        }
    }

    public int fjoldi()
    {
        return fjoldiTalna;
    }

    public double summa()
    {
        int i;
        double summa = 0;
        for(i=0; i<fjoldiTalna; i++)
        {
            summa += t[i];
        }
        return summa;
    }

    public double medaltal()
    {
        return summa()/fjoldiTalna;
    }
}
```

Taktu eftir klasabreytunum. Þær eru `private` sem þýðir að hlutir af öðrum tegundum geta hvorki notað þær né breytt innihaldi þeirra.

```
    private double[] t;
    private int fjoldiTalna;
    private int hamarksfjoldiTalna = 100;
```

Aðferðirnar sem tilheyra klasanum `Talnasafn` nota þessar breytur og geta breytt gildi þeirra. Þetta er eins og það á að vera því ef klasann sem notar `Talnasafn` getur t.d. breytt innihaldi breytunnar `fjoldiTalna` þá er ekki hægt að tryggja að aðferðirnar í `Talnasafn` eins og t.d. `medaltal` skili réttri útkomu. Með því að hafa aðferðir sínar `private` getur `Talnasafn` sagt við aðra klasa: „Þið getið ekki fiktað í mínu dóti og þess vegna get ég lofað ykkur að vinna rétt þau verk sem þið felið mér.“

Líttu á aðferðina

```
    public double medaltal()
    {
        return summa()/fjoldiTalna;
    }
```

Hún er ósköp einföld og stutt því hún notar aðferðina `summa`. Með skipuninni `summa` reiknar `Talnasafn` summuna af sjálfu sér. Það er álitamál hvort eðlilegra er að skrifa summa eða `this.summa`. Væri seinni kosturinn valinn væri aðferðin `medaltal` svona:

```
    public double medaltal()
    {
        return this.summa()/fjoldiTalna;
    }
```

`this.summa()` þýðir að hluturinn sjálfur (þ.e. talnasafnið) beiti aðferðinni summa á sjálfan sig. Ef `this` er sleppt er gert ráð fyrir að aðferðinni sé beitt á hlutinn sjálfan. Oftast nær hefur það engin áhrif til eða frá þótt `this` sé sleppt því ef ekki er tekið fram hvaða hlutur á að framkvæma aðferð er sjálfkrafa gert ráð fyrir að átt sé við hlutinn `this`.

Hlutur af tegundinni `Talnasafn` gerir ekkert einn og sér. En hér kemur athafnasamur klasi sem notar `Talnasafn`.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_06_03
// Reiknivjel
//
public class Reiknivjel extends Applet
{
   Talnasafn tsafn;
   String millitalna = new String(" ,\n\r\t");
   TextField tUtkoma;
   TextArea tTsafn;
   Button bSumma, bMedaltal, bFjoldi;
   double utkoma;

   public void init()
   {
      tsafn = new Talnasafn();
      tUtkoma = new TextField(25);
      tTsafn = new TextArea(5,25);
      bSumma = new Button("Summa");
      bMedaltal = new Button("Meðaltal");
      bFjoldi = new Button("Fjöldi");

      this.add(tUtkoma);   // Hér má sleppa this og skrifa bara
      this.add(bSumma);    // add(tUtkoma); add(bSumma); o.s.frv.
      this.add(bMedaltal);
      this.add(bFjoldi);
      this.add(tTsafn);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bSumma)
      {
         tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
         tUtkoma.setText("Summan er " + tsafn.summa());
         return true;
      }

      if (e.target == bMedaltal)
      {
         tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
         tUtkoma.setText("Meðaltalið er " + tsafn.medaltal());
         return true;
      }
```

```
      if (e.target == bFjoldi)
      {
         tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
         tUtkoma.setText("Fjöldi talna er " + tsafn.fjoldi());
         return true;
      }
      return false;
   }
}
```

Verkefni 6.8
Bættu við klasann `Talnasafn` aðferðum til að reikna meðalfrávik, fervik, og staðalfrávik. (Staðalfrávik er kvaðratrót af ferviki).

Verkefni 6.9
Bættu við klasann `Reiknivjel` tökkum fyrir meðalfrávik, fervik og staðalfrávik.

6.x. Spurningar og umhugsunarefni

1. Hvaða númer hafa fyrsta og síðasta hólf fylkis sem er búið til með eftirfarandi skipunum?

```
double[] x;
x = new double[5];
```

2. Hvaða gildi fær breytan x ef þessar skipanir eru gefnar?

```
int x;
int[] n;
n = new int[5];
x = n.length;
```

3. Hvaða gildi fá `p1` og `p2` ef þessar skipanir eru gefnar?

```
String s, p1, p2;
StringTokenizer st;
s = new String("5, sjö og IX");
st = new StringTokenizer(s, ", ");
p1 = st.nextToken();
p2 = st.nextToken();
```

4. Hvaða gildi fær s ef þessar skipanir eru gefnar?

```
int[] n = {2, 4, 6, 8};
int s = 0;
for (int i=0; i<n.length; i++)
{
   s += n[i];
}
```

5. Hvers vegna eru klasabreyturnar í klasanum `Talnasafn` hafðar `private`?

Til umhugsunar

Með því að nota fylki er hægt að búa til forrit sem vinna útreikninga með mörgum tölum, jafnvel mörg þúsund. Það er líka hægt að nota fylki til að geyma mjög stórar tölur því fylki af 1000 tölum milli 0 og 9 er t.d. hægt að túlka sem eina 1000 stafa tölu.

En ætli það sé mögulegt að vinna útreikninga með hvað margar eða hvað stórar tölur sem er? Ætli það sé til dæmis hægt að búa til forrit sem margfaldar saman tvær trilljón stafa tölur?

Á árunum milli 1930 og 1940 setti enski stærðfræðingurinn Alan Turing fram þá kenningu að hægt sé að byggja allar stærðfræðilegar aðferðir úr fáeinum einföldum aðgerðum. Á þessum árum voru ekki til neinar tölvur. Þær urðu ekki til fyrr en milli 1945 og 1950 og raunar átti Alan Turing drjúgan þátt í hönnun og smíði þeirra fyrstu.

Til að gera grein fyrir því hvernig byggja mætti flóknar aðferðir úr einföldum vélrænum aðgerðum lýsti Turing ímyndaðri vél sem nefnd er eftir honum og kölluð Turing vél. Slík vél hefur endalausan pappírsrenning sem hún getur fært fram og aftur og skrifað á merki af einni gerð. Hún getur líka lesið merki af renningnum og brugðist við þeim á örfáa einfalda vegu. Með því að skrifa merki á renninginn áður en vélin er sett í gang má forrita hana til að vinna alls konar útreikninga.

Turing gerði nákvæma grein fyrir þeim einföldu aðgerðum sem vélin þyrfti að ráða við til að hægt væri að forrita hana og setti fram þá tilgátu að þessar aðgerðir dugi til að vinna öll verk sem hægt er að vinna eftir stærðfræðilegri aðferð.

Hægt er að nota hvaða forritunarmál sem er til að skipa tölvu að framkvæma allar þessar einföldu aðgerðir. (Endurtekning, skilyrði og samlagning, frádráttur og samanburður heilla talna duga.) Einnig hefur verið sýnt fram á að aðgerðirnar sem Turing lýsti duga til að vinna alla útreikninga sem tölvur geta unnið. Í mikilvægum skilningi eru tölva og Turing vél því jafngildar. Það er hægt að forrita þær til að vinna sömu útreikninga.

Turing gat ekki sannað kenningu sína um að vélarnar sem hann lýsti gætu unnið öll verk sem hægt er að vinna eftir stærðfræðilegri aðferð. En þar sem kenningin er enn óhrakin eftir meira en 60 ár og kemur ágætlega heim við allt sem vitað er um stærðfræðilegar aðferðir álíta flestir að hún hljóti að vera rétt.

Sé kenning Turing sannleikanum samkvæm er hægt að nota þann hluta Java málsins sem þegar hefur verið fjallað um í köflum 1 til 6 til þess að orða hvaða reikniaðferð sem er, eða allar aðferðir sem hægt er að setja fram af stærðfræðilegri nákvæmni.

Það er lykilatriði í rökfærslu Turings að pappírsræman sem vélin hefur til að skrifa á hafi ótakmarkaða lengd. Og rökin fyrir því að tölva sé jafngild Turing vél gera ráð fyrir því að tölvan geti unnið með tölur af hvaða stærð sem er. En engin tölva hefur ótakmarkað minnispláss.

Ætli þetta þýði að tölvur geti í raun og veru ekki unnið eftir hvaða reikniaðferð sem er?

Skyldi vera hægt að sigrast á þessum takmörkunum með því að láta tölvu skrifa á disk það sem ekki kemst í minnið?

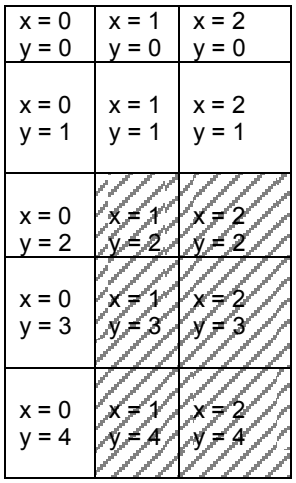
En hvað ef diskurinn fyllist? Ætli það sé mögulegt að láta tölvu bæta í sig nýjum diskum þegar þeir sem fyrir eru fyllast?

7. kafli: Stjórn útlits

7.a. GridBagLayout

Klasinn `GridBagLayout` er einn af fimm útlitsstjórum í pakkanum `java.awt`. Hann er sá flóknasti og jafnframt sá fullkomnasti. Með því að nota útlitsstjóra (Layout manager) er hægt að stjórna því hvernig takkar, textareitir og fleiri sýnilegir hlutir raðast á skjáinn.

`GridBagLayout` skiptir skjámyndinni í ferhyrnda reiti. Reitirnir eru ekki endilega jafnstórir, þeir teygjast til að hlutirnir sem í þá eru settir passi sem best. Hver reitur er auðkenndur með x og y hnitum eins og myndin sýnir.



Eigi hlutur að lenda á skyggða svæðinu þá eru hnit hans stillt á x = 1 og y = 2, breidd á 2 og hæð á 3. Stillingarnar eru gerðar með því að búa til hlut af tegundinni `GridBagConstraints` og gefa breytum hans gildi.

Ef nota á `GridBagLayout` er byrjað á að skilgreina breytu af þeirri gerð og búa hlutinn til og stilla `this` (þ.e. `Applet`-ið sem er verið að forrita) á að nota hann. Eigi breytan að heita `utlit` eru notaðar skipanirnar

```
GridBagLayout utlit = new GridBagLayout();
this.setLayout(utlit);
```

Sé svo ætlunin að staðsetja hlut sem heitir `t` og er af tegundinni `TextArea` á skyggða svæðinu þarf að búa til stillingar fyrir hlutinn `t`. Til þess er búið til `GridBagConstraints` (sem hér er látið fá nafnið `tStadur` því það stjórnar staðsetningu `t`). Þetta er gert með skipuninni

```
GridBagConstraints tStadur =  new GridBagConstraints();
```

Þá er að gefa klasabreytum `tStadur` gildi. Það er gert með

```
tStadur.gridx = 1;
tStadur.gridy = 2;
tStadur.gridwidth = 2;
tStadur.gridheight= 3;
```

Að síðustu þarf svo að stilla `t` á að nota þessa staðsetningu og bæta hlutnum á skjámyndina. Eftirfarandi skipanir sjá um það. Sú fyrri lætur `utlit` stilla `t` á að vera á þeim stað sem tilgreindur er í breytum hlutarins `tStadur`.

```
utlit.setConstraints(t, tStadur);
this.add(t);
```

Breyturnar `gridx`, `gridy`, `gridwidth` og `gridheight` stjórna því í hvaða reiti hlutur er settur. `GridBagConstraints` hefur líka breytur sem stjórna því hvernig hlutur er staðsettur í reitunum sem hann fær til afnota. Þær mikilvægustu eru `fill` og `insets`.

Gildi breytunnar `fill` ræður því hvort hlutur sem er minni en svæðið sem hann hefur til umráða er teygður til að fylla út í það. Hún getur haft 4 mismunandi gildi. Þau eru:

`GridBagConstraints.NONE`	Ekki teygt á neinn veg.
`GridBagConstraints.HORIZONTAL`	Hlutur teygður svo hann fylli út í breiddina.
`GridBagConstraints.VERTICAL`	Hlutur teygður svo hann fylli út í hæðina.
`GridBagConstraints.BOTH`	Hlutur teygður svo hann fylli bæði út í hæð og breidd svæðisins.

Breytan `insets` stjórnar því hvað er höfð breið „spássía" á svæðinu (þ.e. breið rönd sem skilin er eftir auð). Hún er af tegundinni `Insets` og eigi t.d. að vera 2 punkta spássía efst, 3 punktar til vinstri, 4 neðst og 5 til hægri í kringum hlutinn t þá þurfum við að bæta inn skipuninni

```
tStadur.insets = new Insets(2,3,4,5);
```

Hér fer á eftir endurbætt gerð af reiknivélarklasanum þar sem `GridBagLayout` er notað til að raða hlutunum á skjáinn.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_07_01
// Reiknivjel
//
public class Reiknivjel extends Applet
{
   Talnasafn tsafn;
   String millitalna = new String(" ,\n\r\t");

   TextField tUtkoma;
   TextArea tTsafn;
   Button bSumma;
   Button bMedaltal;
   Button bFjoldi;

   public void init()
   {
      tsafn = new Talnasafn();

      GridBagLayout utlit = new GridBagLayout();
      this.setLayout(utlit);

      // tUtkoma sett efst og látið vera
      // 1 reitur á hæð og 6 á breidd.
      GridBagConstraints tUtkomaStadur =
                                 new GridBagConstraints();
      tUtkomaStadur.gridx = 0;       // Hornið efst til vinstri
      tUtkomaStadur.gridy = 0;       // hefur hnitin x=0 og y=0.
      tUtkomaStadur.gridwidth = 6;  // Breiddin er 6 reitir og
      tUtkomaStadur.gridheight= 1;  // hæðin er 1 reitur.
      // Stillt á að hafa 3 punkta breiða rönd hringinn.
      tUtkomaStadur.insets = new Insets(3,3,3,3);
      // Stillt á að fylla út í alla breiddina.
      tUtkomaStadur.fill = GridBagConstraints.HORIZONTAL;
      tUtkoma = new TextField(24);
      utlit.setConstraints(tUtkoma, tUtkomaStadur);
      this.add(tUtkoma);

      // Næst er staðsetning tTsafn stillt
      GridBagConstraints tTsafnStadur =
                              new GridBagConstraints();
      tTsafnStadur.gridx = 0;
      tTsafnStadur.gridy = 2;
      tTsafnStadur.gridwidth = 2;
      tTsafnStadur.gridheight= 8;
      tTsafnStadur.insets = new Insets(0,3,0,3);
      // Stillt á að fylla bæði út í breiddina og hæðina.
      tTsafnStadur.fill = GridBagConstraints.BOTH;
      tTsafn = new TextArea(5,8);
      utlit.setConstraints(tTsafn, tTsafnStadur);
      this.add(tTsafn);
```

```
   // Næst er staðsetning bSumma stillt
   GridBagConstraints bSummaStadur =
                           new GridBagConstraints();
   bSummaStadur.gridx = 2;
   bSummaStadur.gridy = 2;
   bSummaStadur.gridwidth = 2;
   bSummaStadur.gridheight= 1;
   bSummaStadur.insets = new Insets(0,0,3,3);
   // Stillt á að fylla út í breiddina.
   bSummaStadur.fill = GridBagConstraints.HORIZONTAL;
   bSumma = new Button("Summa");
   utlit.setConstraints(bSumma, bSummaStadur);
   this.add(bSumma);

   // Þá er komið að takkanum  bMedaltal
   GridBagConstraints bMedaltalStadur =
                           new GridBagConstraints();
   bMedaltalStadur = (GridBagConstraints)bSummaStadur.clone();
   bMedaltalStadur.gridx = 4; // bMedaltalStadur er
                              // eins og bSummaStdur
                              // nema hvað gridx er
                              // 4 en ekki 2.
   bMedaltal = new Button("Meðaltal");
   utlit.setConstraints(bMedaltal, bMedaltalStadur);
   this.add(bMedaltal);

   // Hér kemur svo takkinn bFjoldi
   GridBagConstraints bFjoldiStadur =
                           new GridBagConstraints();
   bFjoldiStadur = (GridBagConstraints)bSummaStadur.clone();
   bFjoldiStadur.gridy = 3;   // bFjoldiStadur er
                              // eins og bSummaStdur
                              // nema hvað gridy er
                              // 3 en ekki 2.
   bFjoldi = new Button("Fjöldi");
   utlit.setConstraints(bFjoldi, bFjoldiStadur);
   this.add(bFjoldi);
}


public boolean action(Event e, Object o)  // Eins og í
{                                         // forrit_06_03
   if (e.target == bSumma)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Summan er " + tsafn.summa());
      return true;
   }

   if (e.target == bMedaltal)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Meðaltalið er " + tsafn.medaltal());
      return true;
   }
```

```
      if (e.target == bFjoldi)
      {
         tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
         tUtkoma.setText("Fjöldi talna er " + tsafn.fjoldi());
         return true;
      }
      return false;
   }
}
```

7.b. `clone` og =

Taktu eftir því hvernig staðsetning takkans `bMedaltal` er stillt. Það er tekið afrit af stillingum takkans `bSumma` með skipuninni

```
bMedaltalStadur = (GridBagConstraints)bSummaStadur.clone();
```

Síðan er því eina atriði sem er ekki eins í stillingum þessara tveggja takka breytt með skipuninni

```
bMedaltalStadur.gridx = 4;
```

Til að afrita hlut og láta annan hlut verða eins og hann er aðferðin `clone` notuð. Hún tilheyrir klasanum `GridBagConstraints` og yfirskyggir samnefnda aðferð í klasanum `Object`. Þar sem `clone` skilar hlut af tegundinni `Object` þarf að breyta útkomunni í `GridBagConstraints`. Þetta er gert alveg eins og t.d. `int` er breytt í `char` eða `double` í `int` með því að setja tegundarheitið innan sviga fyrir framan.

Marga hluti er hægt að klóna með `clone` í klasanum `Object` en þó ekki alla. Um þetta verður fjallað nánar í E. kafla.

Þegar unnið er með einfaldar tegundir eins og `int`, `double` og `char` er hægt að afrita gildi einnar breytu í aðra breytu með =. Dæmi:

```
int i, j;
i = 5;
j = i;
i += 3;
```

Eftir að þessar skipanir hafa verið gefnar hefur `j` gildið 5 og `i` gildið 8. Þriðja skipunin afritar innihald `i` og setur í `j` en `i` og `j` eru samt sjálfstæðar breytur þannig að þó innihald `i` breytist eftir að skipunin

```
j = i;
```

er gefin þá hefur það engin áhrif á `j`. Þetta á ekki við um hluti eins og `Button`, `Talnasafn` eða `GridBagConstraint`. Skipunin

```
bMedaltalStadur = bSummaStadur;
```

mundi ekki afrita hlutinn sem `bSummaStadur` vísar á og setja afritið í breytuna `bMedaltalStadur` heldur láta `bMedaltalStadur` og `bSummaStadur` báðar vísa á sama hlut. Væri síðan gefin skipunin

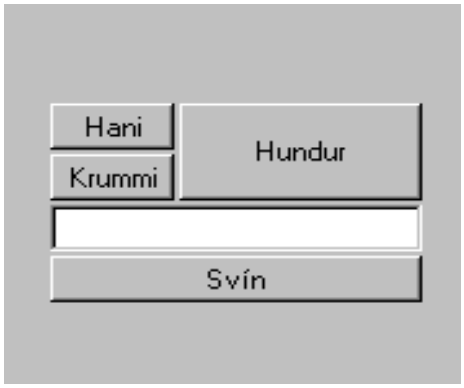
```
bMedaltalStadur.gridx = 4;
```

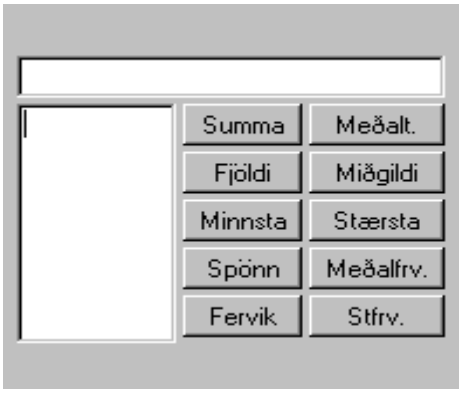
þá mundi `bSummaStadur.gridx` líka fá gildið 4. Ástæðan fyrir þessari ólíku hegðun einfaldra tegunda og hluta var skýrð í kafla 2.d. Þar var gerð grein fyrir því að breyta af einfaldri tegund er nafn á því plássi í minni tölvunnar sem geymir gildi breytunnar en breyta sem geymir hlut nefnir ekki beint plássið sem hluturinn fyllir heldur stað í minni sem geymir vistfang hans (þ.e. tölu sem segir hvar í minni tölvunnar hluturinn

er). Ef `I` og `J` eru tvær breytur af tegundinni `Integer` þá nefnir `I` stað í minni sem geymir vistfang eins hlutar af tegundinni `Integer` og `J` nefnir annan stað sem einnig geymir vistfang eins hlutar af tegundinni `Integer`. Sé gefin skipunin

```
J = I;
```

þá fær `J` sama innihald og `I` svo innihald beggja vísar á sama vistfang. Þótt `I` og `J` séu tvær breytur vísa þær því á einn og sama hlutinn. (Tveir kassar sem báðir innihalda t.d. bíl geta ekki innihaldið sama bílinn en tveir kassar sem báðir innihalda miða með bílnúmeri geta haft innihald sem vísar á sama bílinn.)





Verkefni 7.1
Búðu til forrit sem lítur út eins og myndin til vinstri. Forritið á að skrifa „Galar" í textareitinn ef ýtt er á takkann sem merktur er Hani, „Krunkar" ef ýtt er á Krummi, „Geltir" ef ýtt er á Hundur og „Hrín" ef ýtt er á Svín.

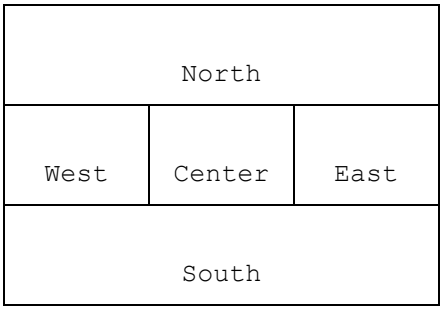
Verkefni 7.2
Bættu við `init`-aðferðina í `reiknivjel` þannig að forritið líti út eins og myndin til hægri. Láttu alla takkana virka eðlilega nema Miðgildi, Minnsta, Stærsta og Spönn. Þeim aðgerðum verður bætt við klasann `Talnasafn` í næsta kafla.

7.c. Útlitsstjórarnir fimm

`GridBagLayout` er einn af 5 útlitsstjórum í pakkanum `java.awt`. Hinir heita `FlowLayout`, `BorderLayout`, `CardLayout` og `GridLayout`.

`Applet` hafa sjálfkrafa `FlowLayout` ef annað er ekki tekið fram. Meðan það er í gildi raðast hlutirnir einfaldlega frá vinstri til hægri þar til ekki kemst meira á breiddina, þá er næsti hlutur settur vinstra megin í næstu línu fyrir neðan. `FlowLayout` býður ekki upp á neina aðferð til að stjórna því hvað hver hlutur fær mikið pláss og eina leiðin til að stýra því hvar hlutirnir lenda er að ákveða í hvaða röð þeim er bætt á myndflötinn.

`BorderLayout` skiptir fletinum í 5 reiti sem heita `North`, `West`, `East`, `South` og `Center`. Þeim er raðað svona.



Eigi forrit að nota `BorderLayout` og setja takka sem á stendur OK á svæðið `East` þá hægt að nota skipanirnar

```
BorderLayout bl = new BorderLayout();
this.setLayout(bl);
Button b = new Button("OK");
this.add("East", b);
```

Sé `BorderLayout` notað er hægt að stjórna staðsetningu hluta með því að senda `add`-aðferðinni nafn reits innan gæsalappa.

Í stuttu máli sagt er `FlowLayout` einfaldasti útlitsstjórinn en hann er takmarkaður að því leyti að hann hefur engar aðferðir til að stjórna stærð og staðsetningu hluta. Næst einfaldasti útliststjórinn er `BorderLayout`. Hann skiptir fleti í 5 svæði og leyfir mönnum að stjórna því í hverju þeirra hver hlutur lendir. Flóknasti og fullkomnasti útlitsstjórinn er `GridBagLayout`. Hann gefur forritara fullt vald yfir stærð og staðsetningu hluta.

Hér verður ekki fjallað um `GridLayout` og `CardLayout`.

7.d. `Panel`

Þegar `BorderLayout` er notað er aðeins hægt að setja einn hlut á hvert svæði. En það er samt hægt að hafa marga hluti á hverju svæði með því að setja hlutina fyrst á `Panel` og `Panel`-inn svo á svæðið. Svæðið inniheldur þá einn `Panel` sem inniheldur marga hluti.

Myndin til hægri sýnir `BoderLayout` þar sem er eitt merki (`Label`) á norðursvæði, einn textareitur (`TextArea`) á miðsvæði, og einn takki (`Button`) á austur- og vestursvæðum. Á suðursvæði er svo `Panel` með þrem tökkum.

Klasabreyturnar og `init`-aðferðin sem notuð voru til að setja upp þessa mynd koma hér á eftir.

```
BorderLayout bl;
FlowLayout fl;
Panel pSudur;
Label merkiN;
Button bSudur1, bSudur2, bSudur3;
Button bVestur, bAustur;
TextArea t;

public void init()
{
   // Hlutir búnir til;
   bl = new BorderLayout();
   fl = new FlowLayout();
   pSudur = new Panel();
   merkiN = new Label("Margir takkar á suðursvæði");
   bSudur1 = new Button("Takki 1");
   bSudur2 = new Button("Takki 2");
```

```
        bSudur3 = new Button("Takki 3");
        bAustur = new Button("Hæ");
        bVestur = new Button("Hó");
        t = new TextArea(10, 15);

        // Applet-ið sem heild (this) látið hafa BorderLayout.
        this.setLayout(bl);

        // Flöturinn (panel-ið) sem á að fara á suðursvæðið látinn
        // hafa FlowLayout og 3 takkar settir á hann.
        pSudur.setLayout(fl);
        pSudur.add(bSudur1);
        pSudur.add(bSudur2);
        pSudur.add(bSudur3);

        // Panel-inn settur á suðursvæði myndflatar

        this.add("South", pSudur);

        // Aðrir hlutir settir á myndflötinn.
        this.add("North", merkiN);
        this.add("East", bAustur);
        this.add("West", bVestur);
        this.add("Center", t);
    }
```

Með því að nota `Panel` er hægt að búa til alls konar skjámyndir. Það er til dæmis hægt að láta einn `Panel` hafa `BorderLayout` og annan hafa `FlowLayout` og setja þann síðarnefnda svo á miðsvæði þess fyrrnefnda og hann svo á suðursvæði á `Applet` með `BorderLayout`. Þeir möguleikar sem opnast með því að nota `Panel` gefa þó ekki eins fullkomið vald yfir útliti skjámynda og `GridBagLayout`.

Verkefni 7.3

Búðu til forrit sem birtir textasvæði (`TextArea`) lengst til hægri á myndfletinum og tvo takka (`Button`) vinstra megin við það.

7.x. Spurningar og umhugsunarefni

1. Til hvers eru útlitsstjórar?
2. Hvaða hlutverki þjóna eftirfarandi skipanir?

```
GridBagLayout u = new GridBagLayout();
this.setLayout(u);
```

3. Hvaða áhrif hafa þessar skipanir?

```
GridBagLayout u = new GridBagLayout();
this.setLayout(u);
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridx = 3;
gbc.gridy = 1;
gbc.gridwidth = 4;
gbc.gridheight= 1;
gbc.insets = new Insets(3,3,3,3);
gbc.fill = GridBagConstraints.HORIZONTAL;
t = new TextField(14);
```

```
u.setConstraints(t, gbc);
this.add(t);
```

4. Hvaða munur er á `FlowLayout`, `Borderlayout` og `GridBagLayout`?
5. Hvers konar hlutur er `Panel`?
6. Hvernig er hægt að setja marga hluti á eitt svæði á `BorderLayout`?

Til umhugsunar

Sá háttur sem forrit hefur á samskiptum við notendur kallast notendaskil þess. Forritin í þessu hefti hafa afskaplega einföld notendaskil. Þau eru að mestu leyti gerð úr fjórum tegundum sýnilegra hluta (`Label`, `Button`, `TextField` og `TextArea`) og aðferðum sem bregðast við fáeinum gerðum atburða (innslætti á lyklaborð og hreyfingum músarinnar).

Hvaða fleiri sýnilegar tegundir en `Label`, `Button`, `TextField` og `TextArea` eru til?

Eru einhver verkefni sem er ómögulegt að vinna með því að nota aðeins þessar fjórar gerðir?

8. kafli: Fylkjum raðað

8.a. Fylki af tölum raðað

Enn á eftir að bæta við `Talnasafn` aðferð til að reikna miðgildi. Til að reikna miðgildi talnasafns þarf að raða öllum tölunum í röð eftir stærð og finna svo hvaða tala eða tölur lenda í miðri röðinni. Forrit_08_01 raðar fylki af kommutölum.

```
import java.applet.Applet;
import java.awt.*;
import java.util.StringTokenizer;
//
// forrit_08_01
// Rodun
//
public class Rodun extends Applet
{
double[] talnalisti;
Button bRada;
TextArea tTalnalisti;
int fjoldiTalna;

   public void init()
   {
      talnalisti = new double[20];
      bRada = new Button("Raða");
      tTalnalisti = new TextArea(10, 20);
      this.add(bRada);
      this.add(tTalnalisti);
   }

   public boolean action(Event e, Object o)
   {
      String sTalnalisti, sTala;
      Double Tala;
      StringTokenizer st;
      int i;

      if (e.target == bRada)
      {
         // Tölurnar í textareitnum settar í fylkið,
         // ein tala í hvert hólf í fylkinu.

         sTalnalisti = new String(tTalnalisti.getText());
         st = new StringTokenizer(sTalnalisti, ", \n\r");

         fjoldiTalna = 0;

         while (st.hasMoreTokens())
         {
            sTala = new String(st.nextToken());
            Tala = new Double(sTala);
            talnalisti[fjoldiTalna] = Tala.doubleValue();
            fjoldiTalna++;
         }

         // Fylkinu raðað
         rada();

         tTalnalisti.setText(""); //Reiturinn fyrst tæmdur.

         // Raðaða fylkið svo sett í textareitinn.
         for(i=0; i<fjoldiTalna; i++)
```

```
            {
               tTalnalisti.appendText(talnalisti[i] + "\n");
            }
            return true;
         }
         return false;
      }

      private int fremstaTala(int upph, int endi)
      {
         int tilgata, i;
         tilgata = upph;

         for(i=upph+1; i<=endi; i++)
         {
            if (talnalisti[i] < talnalisti[tilgata])
            {
               tilgata = i;
            }
         }
         return tilgata;
      }

      private void rada()
      {
         int fremsta, i;
         double x;

         for(i=0; i<fjoldiTalna-1; i++)
         {
            // Fundið nr. fremstu (þ.e. minnstu) tölu á bilinu frá
            // nr. i til enda fylkis.
            fremsta = fremstaTala(i, fjoldiTalna-1);
            // Skipt á tölu nr. i og minnstu tölu á bilinu i til enda.
            x = talnalisti[i];
            talnalisti[i] = talnalisti[fremsta];
            talnalisti[fremsta] = x;
         }
      }
}
```

Röðunin er framkvæmd af tveim aðferðum sem eru

```
   private int fremstaTala(int upph, int endi)
```

og

```
   private void rada()
```

Sú fyrrnefnda skilar tölu sem er númer hólfsins sem inniheldur lægstu töluna á bilinu frá `upph` til `endi`. Ef fylki inniheldur tölurnar

3 5 0 7 4 11 14 2 9 8

þá fær x gildið 7 ef gefin er skipunin

```
x = fremstaTala(3, 9)
```

því lægsta talan á svæðinu frá og með tölu númer 3 (sem er 4. talan því fyrsta hólfið í fylkinu er númer 0) til og með tölu númer 9 er í hólfi númer 7.

Aðferðin `rada` notar `fremstaTala`-aðferðina til að finna fyrst lægstu tölu á bilinu frá númer 0 til enda í fylkinu og skipta á henni og þeirri fremstu. Ef verið væri að raða tölunum hér að ofan hefði þetta þær afleiðingar að skipt væri á 3 og 0 og innihald fylkisins yrði

0 5 3 7 4 11 14 2 9 8

Næst finnur `rada` fremstu tölu á bilinu frá hólfi 1 til enda og skiptir á henni og tölu númer 1. Fylkið verður þá

0 2 3 7 4 11 14 5 9 8

Næst er sama endurtekið með tölurnar frá númer 2 til enda. Við það gerist ekkert því lægsta talan er fremst.

Næst er fundin lægsta tala á bilinu frá númer 3 til enda og skipt á henni og tölu númer 3. Fylkið verður þá

0 2 3 4 7 11 14 5 9 8

Þannig er haldið áfram aftur og aftur þar til að síðustu er fundin lægsta tala á blinu frá þeirri næstsíðustu til enda og skipt á henni og þeirri næstsíðustu.

Forriturum þykir þessi röðunaraðferð ekki merkileg. Hún er afar seinvirk en hefur þó þann kost að vera fremur einföld og skiljanleg. Til eru mun fljótvirkari aðferðir en þær eru líka talsvert flóknari.

Taktu eftir því hvernig aðferðin `fremstaTala(int upph, int endi)` fer að. Hún byrjar á að gera ráð fyrir að tala númer `upph` sé minnst og fer svo gegnum fylkið og endurskoðar tilgátuna í hvert sinn sem lægri tala finnst.

Taktu líka eftir því hvernig `rada` skiptir á tölu númer `i` og þeirri tölu sem á að vera fremst. Þetta er gert með skipununum

```
x = talnalisti[i];
talnalisti[i] = talnalisti[fremsta];
talnalisti[fremsta] = x;
```

Sú fyrsta geymir innihald hólfs númer `i` í breytunni `x`, þá er minnsta talan sem á að vera fremst (þ.e. talan í hólfi nr. fremsta) sett í hólfið sem er fremst á bilinu (þ.e. hólf nr. `i`) og að síðustu er tala sem var í hólfi númer `i` og geymd var í `x` sett í hólfið þar sem minnsta talan var.

Verkefni 8.1 *
Búðu til forrit sem sem tekur við allt að 100 tölum og finnur minnstu og stærstu töluna.

Verkefni 8.2 *
Breyttu forrit_08_01 þannig að það raði tölum í öfuga röð.

Verkefni 8.3
Bættu við forrit_08_01 þannig að það reikni miðgildi talnanna. Ef fjöldi talnanna er f og f er oddatala þá er miðgildið tala númer f/2. Sé fjöldinn slétt tala þá er miðgildið meðaltalið af tölu númer f/2-1 og tölu númer f/2.

8.b. Reiknivélin fullgerð

Nú þegar við höfum aðferð til að raða tölum og reikna miðgildi og hæsta og lægsta gildi er ekki mikið mál að klára reiknivélarforritið. Hér á eftir fer það fullgert.

Taktu eftir aðferðunum

```
private int fremstaTala(int upph, int endi)
```

og

```
private void rada()
```

Þær eru `private`, því þær eru aðeins notaðar af aðferðum í klasanum `Talnasafn` en aðrir klasar hafa ekkert við þær að gera.

```
import java.util.StringTokenizer;
//
// forrit_08_02
// Talnasafn
//
public class Talnasafn
{
   private double[] t;
   private int fjoldiTalna;
   private int hamarksfjoldiTalna = 100;

   // ************ SMIÐIR ****************

   public Talnasafn()         // Býr til talnasafn með pláss fyrir
   {                          // hamarksfjoldiTalna sem er 100.
      t = new double[hamarksfjoldiTalna];
      fjoldiTalna = 0;
   }

   public Talnasafn(int h)    // Býr til talnasafn með pláss
   {                          // fyrir h tölur.
      hamarksfjoldiTalna = h;
      t = new double[hamarksfjoldiTalna];
      fjoldiTalna = 0;
   }

   // ************** AÐFERÐ TIL AÐ SETJA TÖLUR Í SAFNIÐ **********
   public void setjaTolurISafn(String s, String d)
   {
      StringTokenizer st;
      String sTala;
      Double Tala;
      fjoldiTalna = 0;
      st = new StringTokenizer(s, d);
      while ((st.hasMoreTokens()) &&
            (fjoldiTalna < hamarksfjoldiTalna))
      {
         sTala = new String(st.nextToken());
         Tala = new Double(sTala);
         t[fjoldiTalna] = Tala.doubleValue();
         fjoldiTalna++;
      }
   }
```

```
// ************* REIKNIAÐFERÐIR *************

public int fjoldi()
{
   return fjoldiTalna;
}

public double summa()
{
   int i;
   double summa = 0;
   for(i=0; i<fjoldiTalna; i++)
   {
      summa += t[i];
   }
   return summa;
}

public double medaltal()
{
   return summa()/fjoldiTalna;
}

// Það sem er hér fyrir neðan er viðbætur við
// klasann Talnasafn í forrit_06_03.

public double midgildi()
{
   rada();
   if ((fjoldiTalna % 2) == 1)   // Ef fjöldinn er oddat. þá á
   {                             // að skila miðtölunni (þar
      return t[fjoldiTalna/2];   // sem fyrsta t. er nr. 0 er
   }                             // miðt. nr. fjoldiTalna/2)
                                 // annars á að skila meðalt.
   else                          // talnanna sitt hvoru megin
   {                             // við miðju.
      return (t[fjoldiTalna/2 - 1] + t[fjoldiTalna/2])/2;
   }
}
public double minnsta()
{
   return t[fremstaTala(0, fjoldiTalna-1)];
}

public double staersta()
{
   return t[aftastaTala(0, fjoldiTalna-1)];
}

public double sponn()   // Finnur muninn á hæstu og
{                       // lægstu tölu í talnasafni.
   return (staersta() - minnsta());
}

public double medalfv() // Reiknar meðalfrávik
{
   int i;
   double s=0;
   double m = medaltal();
   for(i=0; i<fjoldiTalna; i++)
   {
      s += Math.abs(t[i] - m);
   }
   return s/fjoldiTalna;
}
```

```
public double fervik()
{
   int i;
   double s = 0;
   double m = medaltal();
   for(i=0; i<fjoldiTalna; i++)
   {
      s += Math.pow((t[i]-m), 2);
   }
   return s/fjoldiTalna;
}

public double stdev()   // Reiknar staðalfrávik
{
   return Math.sqrt(fervik());
}


// ***************** PRIVATE AÐFERÐIR *********************
// Hér koma aðferðir sem raða talanasafni.
// og finna minnstu og stærstu tölu.

private int aftastaTala(int upph, int endi)
{
   int tilgata, i;
   tilgata = upph;

   for(i=upph+1; i<=endi; i++)
   {
      if (t[i] > t[tilgata])
      {
         tilgata = i;
      }
   }
   return tilgata;
}


private int fremstaTala(int upph, int endi)
{
   int tilgata, i;
   tilgata = upph;

   for(i=upph+1; i<=endi; i++)
   {
      if (t[i] < t[tilgata])
      {
         tilgata = i;
      }
   }
   return tilgata;
}
```

```
   private void rada()
   {
      int fremsta, i;
      double x;
      for(i=0; i<fjoldiTalna-1; i++)
      {
         fremsta = fremstaTala(i, fjoldiTalna-1);
         x = t[i];
         t[i] = t[fremsta];
         t[fremsta] = x;
      }
   }
   // Hér lýkur aðferðum til að raða talansafni.
   // og finna minnstu og stærstu tölu.
}



import java.applet.Applet;
import java.awt.*;
//
// forrit_08_02
// Reiknivjel
//
public class Reiknivjel extends Applet
{
   Talnasafn tsafn;
   String millitalna = new String(" ,\n\r\t");

   TextField tUtkoma;
   TextArea tTsafn;
   Button bSumma;
   Button bMedaltal;
   Button bFjoldi;
   Button bMidgildi;
   Button bMinnsta;
   Button bStaersta;
   Button bSponn;
   Button bMedalfv;
   Button bFervik;
   Button bStdev;

   public void init()
   {
      tsafn = new Talnasafn();

      GridBagLayout utlit = new GridBagLayout();
      this.setLayout(utlit);

      // tUtkoma sett efst og látið vera
      // 1 reitur á hæð og 6 á breidd.
      GridBagConstraints tUtkomaStadur = new GridBagConstraints();
      tUtkomaStadur.gridx = 0;       // Hornið efst til vinstri
      tUtkomaStadur.gridy = 0;       // hefur hnitin x=0 og y=0.
      tUtkomaStadur.gridwidth = 6;   // Breiddin er 6 reitir og
      tUtkomaStadur.gridheight= 1;   // hæðin er einn reitur.
      // Stillt á að hafa 3 punkta breiða rönd hringinn.
      tUtkomaStadur.insets = new Insets(3,3,3,3);
      // Stillt á að fylla út í alla breiddina.
      tUtkomaStadur.fill = GridBagConstraints.HORIZONTAL;
      tUtkoma = new TextField(24);
      utlit.setConstraints(tUtkoma, tUtkomaStadur);
      this.add(tUtkoma);

      // Næst er staðsetning tTsafn
      // stillt og hlutnum bætt á this.
```

```
GridBagConstraints tTsafnStadur = new GridBagConstraints();
tTsafnStadur.gridx = 0;
tTsafnStadur.gridy = 2;
tTsafnStadur.gridwidth = 2;
tTsafnStadur.gridheight= 8;
tTsafnStadur.insets = new Insets(0,3,0,3);
// Stillt á að fylla bæði út í breiddina og hæðina.
tTsafnStadur.fill = GridBagConstraints.BOTH;
tTsafn = new TextArea(5,8);
utlit.setConstraints(tTsafn, tTsafnStadur);
this.add(tTsafn);

// Næst er staðsetning bSumma
// stillt og takkanum bætt á this.
GridBagConstraints bSummaStadur = new GridBagConstraints();
bSummaStadur.gridx = 2;
bSummaStadur.gridy = 2;
bSummaStadur.gridwidth = 2;
bSummaStadur.gridheight= 1;
bSummaStadur.insets = new Insets(0,0,3,3);
// Stillt á að fylla út í breiddina, þá
// verða allir takkar jafnbreiðir.
bSummaStadur.fill = GridBagConstraints.HORIZONTAL;
bSumma = new Button("Summa");
utlit.setConstraints(bSumma, bSummaStadur);
this.add(bSumma);

// Þá er komið að takkanum bMedaltal
GridBagConstraints bMedaltalStadur = new GridBagConstraints();
bMedaltalStadur = (GridBagConstraints)bSummaStadur.clone();
bMedaltalStadur.gridx = 4;          // bMedaltalStadur er
bMedaltalStadur.gridy = 2;          // eins og bSummaStdur
                                    // nema hvað gridx er
bMedaltal = new Button("Meðalt.");  // 4 en ekki 2.
utlit.setConstraints(bMedaltal, bMedaltalStadur);
this.add(bMedaltal);

// Hér kemur takkinn bFjoldi
GridBagConstraints bFjoldiStadur = new GridBagConstraints();
bFjoldiStadur = (GridBagConstraints)bSummaStadur.clone();
bFjoldiStadur.gridx = 2;         // bFjoldiStadur er
bFjoldiStadur.gridy = 3;         // eins og bSummaStdur
                                 // nema hvað gridy er
bFjoldi = new Button("Fjöldi");  // 3 en ekki 2.
utlit.setConstraints(bFjoldi, bFjoldiStadur);
this.add(bFjoldi);

// Hér kemur takkinn bMidgildi
GridBagConstraints bMidgildiStadur = new GridBagConstraints();
bMidgildiStadur = (GridBagConstraints)bSummaStadur.clone();
bMidgildiStadur.gridx = 4;
bMidgildiStadur.gridy = 3;
bMidgildi = new Button("Miðgildi");
utlit.setConstraints(bMidgildi, bMidgildiStadur);
this.add(bMidgildi);

// Hér kemur takkinn bMinnsta
GridBagConstraints bMinnstaStadur = new GridBagConstraints();
bMinnstaStadur = (GridBagConstraints)bSummaStadur.clone();
bMinnstaStadur.gridx = 2;
bMinnstaStadur.gridy = 4;
bMinnsta = new Button("Minnsta");
utlit.setConstraints(bMinnsta, bMinnstaStadur);
this.add(bMinnsta);

// Hér kemur takkinn bStaersta
```

```
    GridBagConstraints bStaerstaStadur = new GridBagConstraints();
    bStaerstaStadur = (GridBagConstraints)bSummaStadur.clone();
    bStaerstaStadur.gridx = 4;
    bStaerstaStadur.gridy = 4;
    bStaersta = new Button("Stærsta");
    utlit.setConstraints(bStaersta, bStaerstaStadur);
    this.add(bStaersta);

    // Hér kemur takkinn bSponn
    GridBagConstraints bSponnStadur = new GridBagConstraints();
    bSponnStadur = (GridBagConstraints)bSummaStadur.clone();
    bSponnStadur.gridx = 2;
    bSponnStadur.gridy = 5;
    bSponn = new Button("Spönn");
    utlit.setConstraints(bSponn, bSponnStadur);
    this.add(bSponn);

    // Hér kemur takkinn bMedalfv
    GridBagConstraints bMedalfvStadur = new GridBagConstraints();
    bMedalfvStadur = (GridBagConstraints)bSummaStadur.clone();
    bMedalfvStadur.gridx = 4;
    bMedalfvStadur.gridy = 5;
    bMedalfv = new Button("Meðalfrv.");
    utlit.setConstraints(bMedalfv, bMedalfvStadur);
    this.add(bMedalfv);

    // Hér kemur takkinn bFervik
    GridBagConstraints bFervikStadur = new GridBagConstraints();
    bFervikStadur = (GridBagConstraints)bSummaStadur.clone();
    bFervikStadur.gridx = 2;
    bFervikStadur.gridy = 6;
    bFervik = new Button("Fervik");
    utlit.setConstraints(bFervik, bFervikStadur);
    this.add(bFervik);

    // Hér kemur takkinn bStdev
    GridBagConstraints bStdevStadur = new GridBagConstraints();
    bStdevStadur = (GridBagConstraints)bSummaStadur.clone();
    bStdevStadur.gridx = 4;
    bStdevStadur.gridy = 6;
    bStdev = new Button("Stfrv.");
    utlit.setConstraints(bStdev, bStdevStadur);
    this.add(bStdev);
} // Hér endar init
```

```
public boolean action(Event e, Object o)
{
   if (e.target == bSumma)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Summan er " + tsafn.summa());
      return true;
   }

   if (e.target == bMedaltal)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Meðaltalið er " + tsafn.medaltal());
      return true;
   }

   if (e.target == bFjoldi)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Fjöldi talna er " + tsafn.fjoldi());
      return true;
   }

   if (e.target == bMidgildi)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Miðgildið er " + tsafn.midgildi());
      return true;
   }

   if (e.target == bMinnsta)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Minnsta talan er " + tsafn.minnsta());
      return true;
   }

   if (e.target == bStaersta)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Stærsta talan er " + tsafn.staersta());
      return true;
   }

   if (e.target == bSponn)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Spönnin er " + tsafn.sponn());
      return true;
   }

   if (e.target == bMedalfv)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Meðalfrávikið er " + tsafn.medalfv());
      return true;
   }

   if (e.target == bFervik)
   {
      tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
      tUtkoma.setText("Fervikið er " + tsafn.fervik());
      return true;
   }

   if (e.target == bStdev)
```

```
      {
         tsafn.setjaTolurISafn(tTsafn.getText(), millitalna);
         tUtkoma.setText("Staðalfrávikið er " + tsafn.stdev());
         return true;
      }
      return false;
   } // Hér endar action
}
```

8.c. Að raða öðru en tölum, íslensk stafrófsröð

Það er auðvelt að raða tölum því hægt er að finna hvor tveggja talna er á undan með samanburðaraðgerðinni `<`. Það er ekki eins auðvelt að finna hvor tveggja strengja er á undan ef það á að raða þeim t.d. í stafrófsröð. Til að raða strengjum eða almennum brotum eða öðrum tegundum þarf aðferð sem finnur út hvor tveggja hluta er á undan. Hér kemur dæmi um slíka aðferð. Hún tekur við tveim strengjum og skilar true ef sá fyrri er á undan hinum í stafrófsröð miðað við að stöfunum sé raðað eins og í Unicode stafatöflunni. Aðferðin notar `compareTo`-aðferðina sem tilheyrir tegundinni `String`.

```
private boolean fremri(String s1, String s2)
{
   if (s1.compareTo(s2) < 0) { return true; }
   else { return false; }
}
```

Með því að nota þessa aðferð í stað samanburðaraðgerðarinnar `<` er hægt að búa til aðferð til að finna hvaða orð í fylki er fremst í stafrófsröðinni. Aðferðin getur verið næstum alveg eins og `fremstaTala` í forrit_08_01.

Verkefni 8.4
Búðu til aðferð sem tekur við tveim almennum brotum og skilar true ef það fyrra er lægra en það síðara. Notaðu klasann Brot úr 5. kafla. Búðu svo til forrit sem raðar fylki af brotum.

Verkefni 8.5
Notaðu aðferðina hér að ofan til að búa til forrit sem raðar strengjum.

Aðferðin `fremri` finnur hvor tveggja strengja er framar í stafrófsröð miðað við Unicode stafatöfluna. Í henni eru allir ensku stafirnir í réttri röð frá a til z en séríslensku stafirnir (á, ð, é, í, ó, ú, ý, þ, æ, ö) sem og þeir dönsku og þýsku (å, ø, ä, ß, ü) og aðrir sérþjóðlegir stafir aftan við z. Sé þessi aðferð notuð til að raða orðum í stafrófsröð lendir „Daði“ því aftan við „Davíð“ og „Ása“ aftan við nöfn sem byrja á Z.

Til eru ýmsar leiðir til að bera saman orð og finna hvort er á undan í íslenskri stafrófsröð. Hér kemur ein. Skýringar eru settar inn í kóðann.

```
// Aðferðin numer tekur við staf og skilar númeri þannig
// að ef einn stafur er á undan öðrum í íslenska stafrófinu
// þá fær hann lægra númer.
//   Allir stafir sem tilheyra enska stafrófinu fá númer sem
// er tvöfalt hærra en númerið sem þeir hafa í Unicode
```

```
// stafatöflunni. Allir ensku stafirnir fá því sléttar tölur
// á fær númer sem er einum hærra en númer a, ð fær númer
// sem er einum hærra en d fær o.s.frv. Íslensku stafirnir
// fá þannig oddatölur sem lenda á milli a og b, d og e, i
// og j o.s.frv.
//
public int numer(char c)
{
   if (c == 'á') { return 1+2*(int)'a'; }
   else if (c == 'á') { return 1+2*(int)'a'; }
   else if (c == 'Á') { return 1+2*(int)'A'; }
   else if (c == 'ð') { return 1+2*(int)'d'; }
   else if (c == 'Ð') { return 1+2*(int)'D'; }
   else if (c == 'é') { return 1+2*(int)'e'; }
   else if (c == 'É') { return 1+2*(int)'E'; }
   else if (c == 'í') { return 1+2*(int)'i'; }
   else if (c == 'Í') { return 1+2*(int)'I'; }
   else if (c == 'ó') { return 1+2*(int)'o'; }
   else if (c == 'Ó') { return 1+2*(int)'O'; }
   else if (c == 'ú') { return 1+2*(int)'u'; }
   else if (c == 'Ú') { return 1+2*(int)'U'; }
   else if (c == 'ý') { return 1+2*(int)'y'; }
   else if (c == 'Ý') { return 1+2*(int)'Y'; }
   else if (c == 'þ') { return 1+2*(int)'z'; }
   else if (c == 'Þ') { return 1+2*(int)'Z'; }
   else if (c == 'æ') { return 2+2*(int)'z'; }
   else if (c == 'Æ') { return 2+2*(int)'Z'; }
   else if (c == 'ö') { return 3+2*(int)'z'; }
   else if (c == 'Ö') { return 3+2*(int)'Z'; }
   else { return 2*(int)c; }
}

// Aðferðin fremri tekur við tveim strengjum og skilar true ef sá
// fyrri er framar en sá seinni í íslenska stafr. en false annars.
public boolean fremri(String s1, String s2)
{
   char[] c1;
   char[] c2;
   c1 = s1.toCharArray();           // Strengjunum breytt í
   c2 = s2.toCharArray();           // fylki af char.

   int max;                         // max fær gildi sem er
   if (c1.length > c2.length)       // lengd lengra
   { max = c1.length; }             // fylkisins.
   else
   { max = c2.length; }

   int[] i1, i2;                    // Búin til tvö talna-
   i1 = new int[max];               // fylki af lengdinni
   i2 = new int[max];               // max.

   for(int n=0; n<c1.length; n++)   // Í talnafylkin eru sett
   {                                // númer fyrir hvern staf í
      i1[n] = numer(c1[n]);         // stafafylkjunum þannig að
   }                                // stafur sem er framar en
                                    // annar í íslenska staf-
   for(int n=0; n<c2.length; n++)   // rófinu fær lægra númer.
   {
      i2[n] = numer(c2[n]);
   }

   int n = 0;                       // Fundið númer fyrsta sætis
   while ((n<max)&&(i1[n]==i2[n]))  // sem er ekki eins í báðum
   {                                // talnafylkjunum.
      n++;
   }
```

```
      if (n == max)                    // Ef öll sætin í fylkjunum
      {                                // eru eins þá er útkoman
         return false;                 // false því þá er s1 ekki
      }                                // framar en s2 heldur á
      else                             // sama stað, annars er
      {                                // útkoman fengin með
         return (i1[n] < i2[n]);       // því að bera saman fyrsta
      }                                // sætið sem er ekki eins.
   }
```

Verkefni 8.6
Búðu til forrit sem raðar orðum í rétta íslenska stafrófsröð.

Verkefni 8.7
Búðu til forrit sem raðar bæði íslenskum og dönskum orðum rétt í stafrófsröð.

8.x. Spurningar og umhugsunarefni

1. Hugsaðu þér að breytan `t` geymi fylki af 10 kommutölum. Hvaða skipanir þarf að gefa til að skipta á tölunum í sætum númer 3 og 4 (þannig að tala sem er í sæti nr. 3 lendi í sæti nr. 4 og öfugt)?
2. Hvers vegna er ekki hægt að nota þessa aðferð

```
private boolean fremri(String s1, String s2)
{
   if (s1.compareTo(s2) < 0) { return true; }
   else { return false; }
}
```

   til að finna hvort af tveim íslenskum orðum er framar í stafrófsröðinni?

Til umhugsunar

Ein grein tölvufræðinnar fjallar um greiningu aðferða. Slík greining leiðir í ljós hve langan tíma tekur að vinna eftir þeim og hve mikið rúm þær nota í minni tölvunar.

Skoðaðu aðferðirnar sem raða fylkinu `t` í klasanum `Talnasafn`.

```
private int fremstaTala(int upph, int endi)
{
   int tilgata, i;
   tilgata = upph;

   for(i=upph+1; i<=endi; i++)
   {
      if (t[i] < t[tilgata])
      {
         tilgata = i;
      }
   }
   return tilgata;
}

private void rada()
{
   int fremsta, i;
   double x;
```

```
      for(i=0; i<fjoldiTalna-1; i++)
      {
         fremsta = fremstaTala(i, fjoldiTalna-1);
         x = t[i];
         t[i] = t[fremsta];
         t[fremsta] = x;
      }
   }
```

Gerðu ráð fyrir að í `t` séu 10 tölur. Hvað ætli eftirfarandi skipanirnar sem eru inni í `for`-slaufunni í `rada`-aðferðinni séu framkvæmdar oft?

```
fremsta = fremstaTala(i, fjoldiTalna-1);
x = t[i];
t[i] = t[fremsta];
t[fremsta] = x;
```

En hvað ætli eftirfarandi skipanir, sem eru inni í `for`-slaufunni í aðferðinni `fremstaTala`, séu framkvæmdar oft?

```
if (t[i] < t[tilgata])
{
   tilgata = i;
}
```

Gerðu ráð fyrir að það taki k_1 tímaeiningar (t.d. millisekúndur) að framkvæma skipanirnar inni í `for`-slaufunni í aðferðinni `rada` og k_2 tímaeiningar að framkvæma skipanirnar í `for`-slaufunni í `fremstaTala`.

Hvað tekur langan tíma að raða 10 tölum ef $k_1=1$ og $k_2=1$? En 100 tölum? En n tölum?

Hvernig er hægt að tákna tímann sem röðunin tekur sem fall af k_1, k_2 og n?

Gildin á k_1 og k_2 fara eftir því hve hraðvirk tölva er. Hvernig er hægt að komast að því hvaða gildi k_1 og k_2 hafa fyrir einhverja tiltekna tölvu?

Hraðvirkasta aðferð, til að raða fylki, sem þekkt er, heitir Quicksort. Tíminn sem tekur að raða n tölum ef hún er notuð er um það bil $k \times n \times \ln(n)$ þar sem k er (eins og k_1 og k_2 í dæminu hér að ofan) háð því hve hraðvirk tölvan er.

Gerðu ráð fyrir að í Quicksort aðferðinni sé k=20 og í aðferðinni hér að ofan sé $k_1=1$ og $k_2=1$.

Hvor aðferðin er þá fljótari að raða 10 tölum?

En hvor er fljótari að raða 10000 tölum og hvað munar miklu?

9. kafli: Applet og sjálfstæð forrit

9.a. `Applet`

Hingað til hafa öll forritin í þessu hefti verið `Applet`, þ.e. forrit sem erfa frá klasanum `Applet` og vefsjár eins og Netscape Navigator og Internet Explorer geta keyrt á vefsíðum. Java er þó ekki eingöngu til þess að búa til `Applet`. Það er líka hægt að nota það til að búa til sjálfstæð forrit. Raunar er Java fullkomið alhliða forritunarmál sem hægt er að nota til að búa til allar mögulegar gerðir tölvuforrita.

`Applet` eru ýmsum takmörkum háð því vefsjár leyfa forritum sem keyrð eru á vefsíðum ekki fullkominn aðgang að tölvunni. Ef hægt væri að búa til `Applet` sem t.d. eyðir öllu af harða diskinum í tölvunni þá gætu hrekkjalómar sett slík forrit á vefsíður og þannig orðið þeim sem skoða síðurnar til ama og tjóns. Til að koma í veg fyrir hættu á að forrit á vefsíðum valdi skaða banna vefsjár þeim hvers konar fikt í skrám og stýrikerfi tölvunnar. Þetta þýðir meðal annars að:

a) `Applet` geta ekki lesið eða skrifað á diska og disklinga í tölvunni. Þau hafa alls engan aðgang að skrám.

b) Þau geta ekki náð sambandi gegnum Internetið við neina aðra tölvu en þá sem þau sjálf voru sótt af.

c) Þau hafa ekki aðgang að stýrikerfisaðgerðum eins og að bæta á klemmuspjald (clipboard), sækja af því eða setja útprentun í gang.

Sumar vefsjár slaka nokkuð á þessum öryggiskröfum ef `Applet` er sótt af diski í vélinni sem keyrir það (en ekki af annarri vél gegnum Internetið). Það er því mögulegt að `Applet` komist upp með að gera ýmislegt meðan það er keyrt af diski sem vefsjáin mundi ekki leyfa því að gera inni á vefsíðu sem er sótt af annarri tölvu.

Sjálfstæð forrit sem gerð eru í Java eru ekki háð takmörkunum af því tagi sem hér hefur verið sagt frá.

9.b. Bytecode

Java-þýðandi þýðir forrit af Java á mál sem heitir Bytecode. Java forritið er ævinlega geymt í skrá með eftirnafnið java. Þýðingin lendir í skrá með sama fornafni en eftirnafnið class. Ef klasi heitir t.d. Reiknivjel þá er hann skrifaður í skrá sem heitir Reiknivjel.java og þegar sú skrá er þýdd verður til Bytecode-skrá með nafnið Reiknivjel.class.

Flestir þýðendur þýða af forritunarmáli á vélamál einhverrar tiltekinnar tölvugerðar. Þannig þýðir C++ þýðandi fyrir PC tölvu til dæmis af C++ á vélamál Pentium örgjörvans. Java-þýðandi þýðir ekki á vélamál neinnar tölvutegundar heldur á mál sem heitir Bytecode. Bytecode málið er sérstakt að því leyti að fyrir öll venjuleg vélamál er hægt að búa til hraðvirkan Bytecode-túlk.

Bytecode-túlkur lætur tölvu herma eftir vél sem hefur Bytecode fyrir vélamál. Tölva sem keyrir slíkan túlk er stundum kölluð JVM. (Þetta er skammstöfun á „Java Virtual Machine“ sem mætti þýða með „Java-sýndarvél“.) Þótt tölvur séu ekki smíðaðar til að keyra Java forrit, heldur vélamál gjörva eins og Pentium eða Alpha eða PowerPC, er

hægt að láta hvaða tölvu sem er keyra túlk sem breytir henni í JVM og lætur hana haga sér eins og hún hafi Bytecode fyrir vélamál.

Nýlegar vefsjár innihalda Bytecode-túlk og geta því keyrt forritin sem til verða þegar Java-kóði er þýddur á Bytecode. Eins og fyrr er getið leyfa túlkar sem byggðir eru inn í vefsjár forritum ekki að gera hvað sem er, heldur takmarka aðgang þeirra að skrám og stýrikerfi til að koma í veg fyrir að ráp um vefinn spilli skrám eða uppsetningu hugbúnaðar á tölvunni.

9.c. HTML

Til að vefsjá keyri Java forrit þarf að búa til vefsíðu sem kallar á forritið. Vefsíður eru skrifaðar á skipanamáli sem heitir HTML (HyperText Markup Language). Eigi vefsíða að innihalda Java forrit sem heitir reiknivjel og ætla því pláss sem er 250 punktar á breidd og 300 á hæð þarf HTML-kóðinn að innihalda

```
<APPLET code="Reiknivjel.class" width=250 height=300>
</APPLET>
```

Á milli þessara tveggja lína í HTML-skjalinu geta komið línur sem gefa breytum í `Applet`-inu gildi. Það er semsagt hægt að senda forritinu upplýsingar úr HTML-skránni.

Til að `Applet` lesi þessar upplýsingar þarf það að nota aðferðina.

```
public String getParameter(String name)
```

sem er í klasanum `Applet`. Gerum ráð fyrir að HTML-kóði innihaldi

```
<APPLET code="reiknivjel.class" width=250 height=300>
   <PARAM name="xhnit" value="25">
   <PARAM name="yhnit" value="75">
</APPLET>
```

þá getur `Applet` sótt gildin "25" og "75" inn í breyturnar `X` og `Y` með skipununum

```
String X = this.getParameter("xhnit");
String Y = this.getParameter("yhnit");
```

Síðan er svo hægt að breyta strengjunum `X` og `Y` í tölur t.d. af tegundinni `int` eða `double`.

9.d. Sjálfstæð forrit og `Frame`

Bytecode-túlkar þurfa ekki endilega að vera hluti af vefsjá. Þeir eru líka til sem stök forrit. Slíkir túlkar geta keyrt sjálfstæð Java forrit. Öll sjálfstæð Java forrit byrja á að framkvæma aðferð með titillínunni

```
public static void main(String args[])
```

Við getum breytt hvaða `Applet`-i sem er í sjálfstætt forrit með því að búa til klasa sem inniheldur ekkert nema þessa einu aðferð. Eina hlutverk þessa klasa er að koma forritinu í gang. Það er því ekki úr vegi að láta hann heita `Starta`.

Hugsum okkur til dæmis að við höfum búið til `Applet` sem heitir `A1` og við viljum láta það keyra sem sjálfstætt forrit í glugga sem er 250 sinnum 150 punktar á stærð með vinstra topphorn í punktinum (50, 100) á skjánum og með fyrirsögninni "Sjálfstætt forrit". Til að koma þessu í kring búum við til svona klasa.

```
public class Starta
{
   public static void main(String args[])
   {
      A1 s = new A1("Sjálfstætt forrit");
      s.init();
      s.resize(250, 150);
      s.move(50, 100);
      s.show();
   }
}
```

Fyrsta skipunin í `main`-aðferðinni býr til nýjan hlut af tegundinni `A1` og lætur hann heita `s`. Næsta skipun lætur hlutinn framkvæma `init`-aðferð sína, svo koma tvær sem stjórna stærð hans og staðsetningu. Síðasta skipunin, `s.show()`, gerir hlutinn svo sýnilegan.

Ekki er þetta þó alveg svo einfalt að hægt sé að keyra `Applet` á þennan hátt sem sjálfstætt forrit án þess að breyta þeim neitt. Til að þetta virki þarf m.a. að bæta smið við forritið, láta það erfa frá `Frame` en ekki `Applet` og bæta við það aðferð til að hætta. `Applet` hættir sjálfkrafa þegar vefsjáin yfirgefur vefsíðuna sem það er á. Sjálfstætt forrit sem erfir frá `Frame` þarf sjálft að sjá um að hætta með því að framkvæma skipanirnar

```
this.dispose();
System.exit(0)
```

Sú fyrri lokar glugganum (`Frame`) og sú síðari lætur forritið hætta keyrslu.

Forritið hér á eftir er búið til með því að breyta forrit_01_02 í sjálfstætt forrit. `Applet` hefur sjálfkrafa `FlowLayout` ef ekki er beðið um annað. `Frame` er sjálfkrafa með `BorderLayout`. Til að forritið hafi sams konar útlit og `Applet`-ið sem það er byggt á eru skipanirnar

```
FlowLayout fl = new FlowLayout();
this.setLayout(fl);
```

settar efst í `init`-aðferðina.

```
import java.awt.*; //Frame er partur af java.awt.*
//
// forrit_09_01
// Sjalfstaett_forrit
//
// Byggt á forrit_01_02. Allar breytingar á því eru merktar með því
// að skrifa Breyting eða  Viðbót aftan við línurnar sem breytast eða
// er bætt við.
//
public class Sjalfstaett_forrit extends Frame   // Breyting
// Ath. í titillínu kemur Frame í stað Applet.
{
   TextField t1, t2;
   Button b1;
   Label l1;

   // Applet hættir sjálfkrafa þegar vefsjáin yfirgefur vefsíðuna
   // en sjálfstætt forrit þarf sjálft að sjá um að hætta. Þetta
   // forrit. Hættir þegar ýtt er á takkann bHætta
```

```
   Button bHaetta;                              // Viðbót

   // Þessum smið er bætt við forrit_01_02
   public Sjalfstaett_forrit(String titill)  // Viðbót
   {                                            // Viðbót
      super(titill); // super-aðferðin             Viðbót
   }                 // kallar á smið             Viðbót
                     // yfirklasa.

   public void init()
   {
      // Næstu tveim línu er bætt við forrit_01_02.  Applet er
      // sjálfkrafa með Flowlayout en Frame er sjálfkrafa með
      // BorderLayout. Til að hafa betri stjórn á útliti er hægt að
      // nota GridBagLayout alveg eins og í Applet.
      FlowLayout fl = new FlowLayout();   // Viðbót
      this.setLayout(fl);                 // Viðbót

      t1 = new TextField(20);
      t2 = new TextField(20);
      b1 = new Button("Heilsa");
      l1 = new Label("Nafn:");
      bHaetta = new Button("Hætta");      // Viðbót

      this.add(l1); this.add(t1);
      this.add(b1); this.add(t2);
      this.add(bHaetta);                  // Viðbót
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == b1)
      {
         t2.setText("Góðan dag " + t1.getText());
         return true;
      }

      if (e.target == bHaetta)            // Viðbót
      {                                   // Viðbót
         this.dispose(); // Ramma eytt       Viðbót
         System.exit(0); // Keyrslu hætt    Viðbót
         return true;                     // Viðbót
      }                                   // Viðbót
      return false;
   }
}


// forrit_09_01
// Starta
//
public class Starta
{
   // Öll sjálfstæð forrit byrja á að keyra main-aðferð með titillínu
   // eins og þessa

   public static void main(String args[])
   {
      // Næsta lína býr til hlut af tegundinni sjalfstaett_forrit.
      Sjalfstaett_forrit s =
            new Sjalfstaett_forrit("Sjálfstætt forrit");

      s.init();               // Klasinn sjalfstaett_forrit
      s.resize(250, 150);     // erfir resize-, move- og
      s.move(50, 100);        // show-aðferðirnar frá Frame.
      s.show();               // resize(250, 150) lætur gluggann
```

```
   }                              // verða 250 punkta breiðan og 150 á
                                  // hæð. move(50, 50) staðsetur hornið
}                                 // efst til vinstri og show() gerir
                                  // rammann sýnilegan.
```

Tegundin `Frame` er að ýmsu leyti lík `Applet`. Það er hægt að setja á hana textareiti, takka og fleira. En hún er þó fjölhæfari en `Applet`, getur t.d. haft valmyndir sem `Applet` getur ekki.

Forfeður `Applet` eru

```
Object -> Component -> Container -> Panel
```

en `Frame` erfir

```
Object -> Component -> Container -> Window
```

Þessar tegundir eru því náskyldar. Báðar erfa alla hæfileika `Component` og `Container` (og svo auðvitað `Object` sem allir klasar erfa).

Ekkert er því til fyrirstöðu að `Applet` búi til nokkra hluti af tegundinni `Frame` og hafi slatta af gluggum undir sinni stjórn. `Frame` getur líka innihaldið hlut af tegundinni `Applet`. Þetta síðarnefnda er hægt að nota til að smíða forrit sem bæði er hægt að keyra sem `Applet` og sem sjálfstætt forrit.

Hugsum okkur að búið sé að forrita `Applet` sem heitir `App1`. Til að hægt sé að keyra það bæði sem `Applet` og sem sjálfstætt forrit dugar að bæta við það aðferðinni

```
public static void main(String args[])
{
   App1 a = new App1();
   a.init();
   a.start();
   Frame f = new Frame();
   f.add("Center", a);
   f.resize(200, 200); // Hér má að sjálfsögðu setja aðrar
   f.move(100, 100);   // tölur en 200 og 100.
   f.show();
}
```

Aðferðin býr til nýjan hlut, `a`, af tegundinni `App1`, framkvæmir `init`- og `start`-aðferðir hans, býr svo til nýjan ramma (`Frame`) með skipuninni

```
Frame f = new Frame();
```

og setur `a` á miðsvæðið í rammanum með skipuninni

```
f.add("Center", a);
```

Eigi ramminn að hafa fyrirsögn er hægt að senda smiðnum hana og setja eitthvað á borð við

```
Frame f = new Frame("Skagamenn skoruðu öll mörkin.");
```

í staðinn fyrir

```
Frame f = new Frame();
```

Þótt `App1` hafi e.t.v. enga `start`-aðferð er óráðlegt að sleppa skipuninni `a.start()` því `App1` erfir start aðferð `Applet` og `start`-aðferðin gegnir hlutverki við að koma `Applet`-i í gang.

Þegar vefsjá keyrir `Applet` framkvæmir hún sjálfkrafa svipaðar skipanir og main-aðferðin hér að framan. En vefsjá framkvæmir ekki `main` aðferð svo þótt `Applet`

innifeli slíka aðferð hefur hún engin áhrif ef það er keyrt í vefsjá. Að bæta svona `main`-aðferð við `Applet` spillir því ekkert möguleikum þess á að koma fram sem `Applet` en bætir við möguleikanum á að keyra sem sjálfstætt forrit.

Sé `Applet`-i breytt með þessum hætti í forrit sem getur keyrt bæði sem `Applet` og sem sjálfstætt forrit þá hefur það enga skipun til að hætta keyrslu þegar það er sjálfstætt. Það ætti því að bæta við `Applet`-ið því sem til þarf svo það geti hætt.

Verkefni 9.1 *
Breyttu forrit_02_04 í sjálfstætt forrit.

Verkefni 9.2 *
Búðu til `Applet` sem les þrjár tölur, x, y og r, úr HTML-skjalinu sem það tilheyrir og teiknar hring með miðju í x,y og radíus r.

Verkefni 9.3
Breyttu reiknivélinni (forrit_08_02) þannig að hún geti bæði keyrt sem `Applet` og sem sjálfstætt forrit.

9.x. Spurningar og umhugsunarefni

1. Hvað af því sem sjálfstæð forrit geta er ekki hægt að láta `Applet` gera?
2. Hvað er Bytecode?
2. Hvað er JVM?
4. Hvað merkja nafnaukarnir (eftirnöfnin) java og class?
5. Hvað er HTML?
6. Til hvers er aðferðin `getParameter` í klasanum `Applet`?
7. Í hverju felst sérstaða aðferðar sem er skilgreind með titillínunni:
   ```
   public static void main(String args[])
   ```
8. Hverju þarf að bæta við `Applet` til að breyta því í sjálfstætt forrit?
9. Hverju þarf að bæta við `Applet` til að bæði sé hægt að keyra það sem `Applet` og sem sjálfstætt forrit?
10. Hvaða áhrif hafa eftirtaldar skipanir?
    ```
    this.dispose();
    System.exit(0);
    ```

Til umhugsunar

Í 9.a var talið upp ýmislegt sem `Applet` mega ekki gera.

Bytecode-túlkurinn sem er innbyggður í Internet Explorer, Netscape og aðrar vefsjár sér um að `Applet` virði þessi boð og bönn. Hann framkvæmir einfaldlega ekki skipanir sem brjóta gegn þeim.

Túlkur sem keyrir sjálfstæð Java forrit leyfir þeim flest sem forrit á vélamáli komast upp með að gera. Slík forrit mega láta tölvu gera flest það sem tölvur yfirleitt geta gert. Þó leyfist þeim ekki hvað sem er. Til dæmis fá þau ekki aðgang að þeim hlutum minnisins sem önnur forrit hafa til ráðstöfunar. Túlkurinn sem keyrir Bytecode forritin sér um að þau trufli ekki önnur verk sem eru í gangi á tölvunni.

Þegar forrit á vélamáli eru keyrð verður stýrikerfið í tölvunni að sjá um að setja þeim reglur og tryggja að þær séu virtar. Meðal mikilvægustu hlutverka stýrikerfis er að úthluta forritum plássi í minni tölvunnar og aðgangi að jaðartækjum (skjá, prentara, lyklaborði, mús, diskum o.fl.) og skipta tíma gjörvans milli þeirra forrita sem eru í gangi.

Stýrikerfið á að tryggja að ekki komi til árekstra milli forrita sem keyrð eru samtímis. Slíkir árekstrar geta t.d. orðið ef eitt forrit skrifar í þann hluta minnisins sem annað forrit notar eða breytir skrá sem annað forrit er að lesa.

Í vissum skilningi stendur stýrikerfið á milli vélabúnaðar og annarra forrita. Það skammtar forritunum aðgang að vélabúnaðinum. Þegar forrit eru keyrð af túlki bætist við einn milliliður í viðbót. Þetta gerir keyrsluna hægari en á móti kemur að auðveldara er að framfylgja ítrustu kröfum um öryggi því túlkur getur með fremur hægu móti séð um að forrit geri ekki annað en það má gera. Hann getur „skoðað“ hverja skipun í forritinu og „ákveðið“ hvort hún skuli framkvæmd eða ekki. Það er öllu flóknara mál að láta stýrikerfi stjórna keyrslu vélamálsforrita. Skipanir í slíkum forritum eru keyrðar af vélbúnaðinum án þess að stýrikerfið yfirfari hverja og eina.

Aðferðin sem flest stýrikerfi hafa til að koma í veg fyrir að forrit skrifi hvert í annars pláss í minninu byggist á því að einoka þær vélamálsskipanir sem nota þarf til að hafa óheftan aðgang að öllu minninu og leyfa forritum aðeins að vísa í minnishólf með númer á tilteknu bili.

Þörfin á að banna `Applet`-um að skrifa á diska og eyða skrám er nokkuð augljós. En hvers vegna þarf að banna þeim að lesa skrár af diski?

Meðal þess sem öllum forritum, öðrum er stýrikerfi, er bannað að gera er að skrifa í þá hluta minnisins sem önnur forrit hafa til umráða. Hvað fleira er ástæða til að banna sjálfstæðum forritum að gera?

A. kafli: Abstract aðferðir, tátugrafík og undirforrit

A.a. `Padda`, `Bjalla`, `Ormur` og hlutbundin forritun

Java er hlutbundið forritunarmál sem þýðir að forrit eru mynduð úr hlutum sem framkvæma aðferðir. Sérhver hlutur tilheyrir einhverjum klasa sem erfir einn yfirklasa og getur átt sér marga undirklasa.

Í þessum kafla verða smíðaðar þrjár gerðir af dýrum: `Padda`, `Bjalla` og `Ormur`. Dýr af öllum þessum tegundum eiga það sameiginlegt að geta ferðast um tölvuskjáinn. Aðferðirnar sem þau hafa til þess heita `fram`, `haegri` og `vinstri`. Sé búið að smíða hlut, `p`, af tegundinni `Padda` á að vera mögulegt að láta hann fara 10 skref áfram, beygja svo um 45 gráður til hægri og labba 20 skref áfram með skipununum

```
p.fram(10);
p.haegri(45);
p.fram(20);
```

Allar dýrategundirnar þrjár eiga það sameiginlegt að hafa stefnu og staðsetningu og kunna aðferðirnar `fram`, `haegri` og `vinstri`. Því er eðlilegt að búa til einn yfirklasa fyrir þær allar. Hér heitir hann `Kvikindi`. Klasarnir `Padda`, `Ormur` og `Bjalla` erfa frá `Kvikindi` aðferðirnar `fram`, `haegri` og `vinstri` og klasabreyturnar `xhnit`, `yhnit` og `stefna`.

Hér kemur aðferðin `fram`. Hún reiknar hvaða hnit kvikindi fær eftir að það hefur farið fram um `x` skref, felur kvikindið svo, gefur klasabreytunum `xhnit` og `yhnit` svo ný gildi og gerir kvikindið aftur sýnilegt. Aðferðin er í stuttu máli sú að stroka kvikindið út og teikna það á annan stað á skjánum.

```
public void fram(double x)
{
   double ny_xhnit, ny_yhnit;
   ny_xhnit = xhnit + x * Math.cos(stefna*Math.PI/180);
   ny_yhnit = yhnit - x * Math.sin(stefna*Math.PI/180);
   fela();
   xhnit = ny_xhnit;
   yhnit = ny_yhnit;
   syna();
}
```

Í forritinu sem hér fer á eftir (forrit_0A_01) er aðferðin `fram` ögn flóknari en þetta því kvikindi geta dregið slóð á skjáinn. Klasabreytan `dregurStrik` sem er af tegundinni `boolean` stjórnar því hvort þau draga slóð eða ganga án þess að ferill þeirra sjáist á skjánum. Ef `dregurStrik` hefur gildið `true` þá er teiknuð lína úr gömlu hnitunum í þau nýju um leið og kvikindið er fært.

```
public void fram(double x)
{
   double ny_xhnit, ny_yhnit;
   ny_xhnit = xhnit + x * Math.cos(stefna*Math.PI/180);
   ny_yhnit = yhnit - x * Math.sin(stefna*Math.PI/180);
   fela();
   if (dregurStrik)
   {
      myndflotur.setPaintMode();
      myndflotur.setColor(liturStriks);
      myndflotur.drawLine((int)xhnit, (int)yhnit,
                     (int)ny_xhnit, (int)ny_yhnit);
```

```
        }
        xhnit = ny_xhnit;
        yhnit = ny_yhnit;
        syna();
    }
```

Verkefni A.1 *
Keyrðu `Dyr1` í forrit_0A_01 og áttaðu þig á hvernig klasinn virkar. Forritið lætur eitt kvikindi af tegundinni `Padda` ferðast um. Breyttu því þannig að kvikindið sé ekki `Padda` heldur `Bjalla`. Breyttu líka lit kvikindisins og lit striksins sem það dregur. (Ath. þú þarft ekki að breyta neinum öðrum klösum en `Dyr1`.)

A.b. `abstract` aðferðir og klasinn `Kvikindi`

Skoðum aðferðina `fram` nú dálítið betur. Hún inniheldur skipanirnar `syna()` og `fela()`. Til að hægt sé að framkvæma þær þarf tegundin `Kvikindi` að kunna aðferðir sem heita `syna` og `fela`. En þótt sama aðferð sé notuð til að færa öll kvikindi hvort sem þau eru `Padda`, `Bjalla` eða `Ormur` er ekki hægt að sýna þau öll með sömu aðferð. Dýrategundirnar hafa ólíkt útlit og því þarf hver þeirra að hafa sínar eigin aðferðir til að sýna og fela (enda sjá þessar aðferðir um að teikna dýrin á skjáinn og stroka þau út).

Við stöndum frammi fyrir því vandamáli að yfirklasinn `Kvikindi` þarf að innihalda aðferðina `fram` sem er sameiginleg fyrir pöddur, orma og bjöllur en til að skrifa fram þarf að nota aðferðir (til að sýna og fela) sem geta ekki verið sameiginlegar. Lausnin á þessum vanda er að nota `abstract` aðferðir.

`Kvikindi` hefur aðferðir sem heita `syna` og `fela` en þær eru `abstract` sem þýðir að:

a) Klasinn `Kvikindi` inniheldur aðeins titillínur þeirra en ekkert innihald eða skipanir sem segja hvað á að gera til að framkvæma þessar aðferðir.

b) Hver einasti undirklasi klasans `Kvikindi` inniheldur aðferðir sem heita `syna` og `fela`.

c) Í hvert sinn sem beðið er um að framkvæma `syna` og `fela` aðferðir klasans `Kvikindi` er gáð hvaða undirklasa hluturinn sem á að framkvæma aðferðina tilheyrir og aðferðir þess undirklasa framkvæmdar.

d) Enginn hlutur getur verið af tegundinni `Kvikindi` nema með því að tilheyra einhverri undirtegund hennar.

`abstract` aðferðir eru hálfgert plat. Þær eru aldrei framkvæmdar. Tegundir (klasar) sem innihalda `abstract` aðferðir eru líka hálfgert plat. Það er ekki hægt að búa til hlut af slíkri tegund. Ef klasi inniheldur `abstract` aðferð þá verður hann sjálfur að vera `abstract` og það er ekki hægt að beita `new` til að smíða hlut af `abstract` gerð.

Í 5. kafla var tegundin `Graphics` notuð. Hún er `abstract` og ýmsar aðferðir sem tilheyra þessari tegund eins og `drawLine` og `drawArc` eru skilgreindar sem `abstract` aðferðir. Þetta þýðir að myndfletirnir sem teiknað er á tilheyra undirklasa `Graphics` og það er ekki hægt að búa til myndflöt, með smið `Graphics`, svona:

```
Graphics m;
```

```
m = new Graphics();
```

Eigi breytan m að vísa á myndflötinn í `Applet`-i þá er henni gefið gildi svona:

```
Graphics m;
m = this.getGraphics();
```

`getGraphics` skilar hlut sem tilheyrir undirklasa `Graphics` og útfærir þær aðferðir sem eru skilgreindar sem abstract í `Graphics`. Það er misjafnt eftir stýrikerfum hvernig þessi undirklasi `Graphics` er skilgreindur. (Hann er t.d. ekki eins fyrir Windows og Macintosh.) Þar sem Java forrit eiga að keyra í ólíkum stýrikerfum nefna þau aðeins `Graphics` en aldrei undirklasana.

Í forritinu sem hér fer á eftir er klasinn `Kvikindi abstract`. Af þessu leiðir að það er ekki hægt að smíða `Kvikindi` svona:

```
Kvikindi k;
k = new Kvikindi(g);
```

Hins vegar er vandræðalaust að búa til `Kvikindi` svona:

```
Padda p;
p = new Padda(g);
```

eða svona:

```
Kvikindi k;
k = new Padda(g);
```

Þar sem allar pöddur eru kvikindi er hægt að búa til kvikindi með því að búa til pöddu (eða björlu eða orm) en það er ekki hægt að búa til neitt sem er bara kvikindi. Þar sem `Padda` er undirklasi `Kvikindi` er ekkert því til fyrirstöðu að geyma hlut sem er af tegundinni `Padda` í breytu af tegundinni `Kvikindi` en sé það gert getur paddan aðeins notað þær aðferðir sem skilgreindar eru í `Kvikindi`.

Það er hægt að geyma hvaða hlut sem er í breytu af yfirtegund hans og það er hægt að geyma alla mögulega hluti í breytum af tegundinni `Object`. En ef hlutur af tegundinni `TextField` eða `Padda` er geymdur í breytu af tegundinni `Object` getur hann aðeins nýtt þær aðferðir og klasabreytur sem tilheyra tegundinni `Object` og þar af leiðandi ekki gert mjög margt.

Með því að nota `abstract` aðferðir er hægt að smíða aðferðir sem eru sameiginlegar mörgum tegundum jafnvel þótt einhverjir partar af þeim þurfi að taka tillit til sérkenna einstakra tegunda. Í því dæmi sem hér er til umfjöllunar er aðferðin `fram` til dæmis höfð sameiginleg öllum undirklösum `Kvikindi` jafnvel þótt tvær skipanir í henni (`syna` og `fela`) séu ólíkar frá einum undirklasa til annars. Ef `Kvikindi` hefði ekki þessar tvær `abstract` aðferðir þá væri alls ekki hægt að hafa skipanirnar `syna()` og `fela()` í aðferðinni `fram`.

Þar sem aðferðirnar `syna` og `fela` eru `abstract` er hægt að láta `Kvikindi` sýna sig og fela án þess að vita hvers konar kvikindi er um að ræða (hvort það er t.d. `Padda` eða `Bjalla`). Tökum sem dæmi að breytan `k` sé skilgreind sem `Kvikindi` og `t` sé `TextField`, `g` sé `Graphics` og gefnar séu skipanirnar

```
String s = t.getText();
if (s.equals("Padda"))
{
   k = new Padda(g);
}
else if (s.equals("Bjalla"))
{
```

```
      k = new Bjalla(g);
   }
   k.syna();
```

Þegar forritið er samið er engin leið að vita hvort `k` muni innihalda pöddu eða bjöllu. Samt er hægt að láta `k` sýna sig því þar sem klasinn `Kvikindi` hefur abstract aðferðina `syna` sér hann um að framkvæma `syna`-aðferð undiklasans `Padda` ef hluturinn sem sýna skal er af þeirri tegund og `syna` aðferð undirklasans `Bjalla` ef hann er af tegundinni `Bjalla`.

Hér kemur klasinn `Kvikindi` í heild. Smiðurinn tekur við myndfleti af tegundinni `Graphics` sem kvikindin skulu teiknuð á.

```
import java.awt.*;
//
// forrit_0A_01
// Kvikindi
//
public abstract class Kvikindi         // Kvikindi geta
{                                      // ferðast um skjáinn
   // með því að fara fram og beygja til hægri eða vinstri.
   // Hægt er að stjórna því hvort þau skilja eftir sig slóð á
   // skjánum eða ekki. Klasinn inniheldur bara þær aðferðir
   // sem eru sameiginlegar öllum kvikindum. Hann er abstract
   // því aðferðirnar syna og fela eru abstract, enda eru
   // þær ólíkar eftir því hvernig kvikindi eru í laginu.

   public double xhnit, yhnit, stefna;    // Allir hlutir sem
   public Color litur;                    // nota klasann kvik-
   public Color liturStriks;              // indi geta stjórnað
   public boolean dregurStrik;            // staðsetningu og
                                          // stefnu kvikinda,
   // lit þeirra og því hvort þau skilja eftir sig slóð og
   // hvernig hún er á litinn. Þess vegna eru allar þessar
   // breytur hafðar public.

   protected Graphics myndflotur;

   public Kvikindi(Graphics g)               // Þegar kvikindi
   {                                         // er búið til fær
      xhnit = 0;                             // smiðurinn sendan
      yhnit = 0;                             // myndflötinn (þ.e.
      stefna = 0;                            // hlut af teg.
      myndflotur = g;                        // Graphics sem
      liturStriks = new Color(0, 0, 0);      // kvikindið skal
      litur = new Color(0, 0, 0);            // teiknað á).
      dregurStrik = false;
   }

   // Aðferðirnar fram, haegri og vinstri nota syna og fela
   // sem eru skilgreindar í undirklösum. Til að þetta sé
   // hægt verða þær að vera til í Kvikindi. Aðferð sem
   // er til í klasa en útfærð í undirklösum kallast
   // abstract og er skilgreind eins og syna og fela hér
   // að neðan.
```

```
   public void fram(double x)
   {
      double ny_xhnit, ny_yhnit;
      ny_xhnit = xhnit + x * Math.cos(stefna*Math.PI/180);
      ny_yhnit = yhnit - x * Math.sin(stefna*Math.PI/180);
      fela();
      if (dregurStrik)
      {
         myndflotur.setPaintMode();
         myndflotur.setColor(liturStriks);
         myndflotur.drawLine((int)xhnit, (int)yhnit,
                        (int)ny_xhnit, (int)ny_yhnit);
      }
      xhnit = ny_xhnit;
      yhnit = ny_yhnit;
      syna();
   }

   public void haegri(double gradur)
   {
      fela();
      stefna = (stefna - gradur) % 360;
      syna();
   }

   public void vinstri(double gradur)
   {
      fela();
      stefna = (stefna + gradur) % 360;
      syna();
   }

   public abstract void syna();      // Hægt er að sýna
                                     // og fela öll kvik-
   public abstract void fela();      // indi en aðferðirnar
                                     // til þess eru mis-
  // munandi eftir lögun kvikinda. Klasar eins og Padda eða
  // Ormur sem erfa frá Kvikindi skilgreina sínar eigin syna
  // og fela aðferðir.
} // Hér endar klasinn Kvikindi
```

Lestu klasann `Kvikindi` vel. Skipunin

```
   myndflotur.setPaintMode();
```

í aðferðinni `fram` verður útskýrð í kafla A.d. Annað ætti að vera þokkalega skiljanlegt. Aðferðirnar `haegri` og `vinstri` nota `abstract` aðferðirnar `syna` og `fela` alveg eins og `fram` gerir. Þar sem `syna` og `fela` eru útfærðar í undirklösum getur hlutur sem er bara `Kvikindi` (en tilheyrir engum undirklasa) hvorki farið fram né tekið beygju en það kemur aldrei að sök því slíkan hlut er ekki hægt að smíða.

A.c. `final` klasar og `Padda`

Nú er klasinn `Kvikindi` tilbúinn. Þá er komið að því að búa til einstakar dýrategundir. Þar sem `Padda` og `Bjalla` verða teiknaðar úr hringjum (`Padda` verður með hringlaga frambúk, hringlaga afturbúk og hringlaga haus og `Bjalla` með hringlaga búk og hringlaga haus) er heppilegt að byrja á að smíða tegund sem teiknar hring með gefinni miðju og radíus.

Þar sem `Hringur` mun ekki hafa neinar undirtegundir er klasinn `final`. Klasar sem eru `final` geta ekki haft undirklasa en á móti kemur að forritið verður hraðvirkara.

Til að forrit verði sem hraðvirkust er rétt að skilgreina klasa sem munu örugglega ekki hafa neina undirklasa sem `final`. Ef klasi er `final` þá eru allar aðferðir hans sjálfkrafa `final` (sem þýðir að ekki er hægt að yfirskyggja þær) enda er ekki um það að ræða að aðferðir séu yfirskyggðar nema klasinn sem þær tilheyra hafi undirklasa.

Eins og fram kom í 5. kafla geta breytur verið `final` og þá er ekki hægt að breyta innihaldi þeirra.

```
import java.awt.*;
//
// forrit_0A_01
// Hringur
//
public final class Hringur // Klasinn Hringur er final sem þýðir að
{                          // ekki er hægt að búa til undirklasa.
   public double xHnitMidju, yHnitMidju;
   public double radius;

   public Hringur(double r, double x, double y)
   {
      radius = r; xHnitMidju = x; yHnitMidju = y;
   }

   public void syna(Graphics g)
   {
      g.fillArc((int)(xHnitMidju - radius),
                (int)(yHnitMidju - radius),
                (int)(2 * radius), (int)(2 * radius), 0, 360);
   }
} // Hér endar klasinn Hringur
```

Hér kemur svo fyrsta dýrategundin.

```
import java.awt.*;
//
// forrit_0A_01
// Padda
//
public class Padda extends Kvikindi
{
   private final int ab = 5;           // Radíus afturbúks
   private final int fb = 4;           // Radíus frambúks
   private final int h = 2;            // Radíus hauss
   private final int fb_h = fb+h;      // Fjarlægð frá miðjum frambúk
   private final int fb_ab = fb+ab-1;  // í miðjan haus og frá miðjum
                                       // frambúk í miðjan afturbúk.

   private Hringur haus, fremriBukur, aftariBukur;

   public Padda(Graphics g)
   {
      super(g);  // Sendir smið yfirklasans (þ.e. Kvikindi) g.
      // xhnit og yhnit sem Padda erfir frá Kvikindi eru miðjan á
      // fremri búknum og út frá henni og stefnu dýrsins er reiknað
      // út hvar miðjan á hausnum og aftari búknum eiga að vera.
      float hausXhnit = Math.round(xhnit +
                           fb_h * Math.cos(stefna*Math.PI/180));
      float hausYhnit = Math.round(yhnit -
                           fb_h * Math.sin(stefna*Math.PI/180));
      float afturbuksXhnit = Math.round(xhnit -
                           fb_ab * Math.cos(stefna*Math.PI/180));
```

```
        float afturbuksYhnit = Math.round(yhnit +
                               fb_ab * Math.sin(stefna*Math.PI/180));
        haus = new Hringur(h, hausXhnit, hausYhnit);
        fremriBukur = new Hringur(fb, xhnit, yhnit);
        aftariBukur = new Hringur(ab, afturbuksXhnit, afturbuksYhnit);
    }

    public void syna()
    {
        fremriBukur.xHnitMidju = xhnit;
        fremriBukur.yHnitMidju = yhnit;
        haus.xHnitMidju = Math.round(xhnit +
                               fb_h * Math.cos(stefna*Math.PI/180));
        haus.yHnitMidju = Math.round(yhnit -
                               fb_h * Math.sin(stefna*Math.PI/180));
        aftariBukur.xHnitMidju = Math.round(xhnit -
                               fb_ab * Math.cos(stefna*Math.PI/180));
        aftariBukur.yHnitMidju = Math.round(yhnit +
                               fb_ab * Math.sin(stefna*Math.PI/180));
        myndflotur.setColor(litur);
        myndflotur.setXORMode(new Color(255,255,255));
        aftariBukur.syna(myndflotur);
        fremriBukur.syna(myndflotur);
        haus.syna(myndflotur);
    }

    public void fela()
    {
        syna();
    }
}
```

`Padda` hefur bara þrjár aðferðir auk þeirra sem hún erfir frá `Kvikindi`. Þær eru smiðurinn `Padda` og aðferðirnar `syna` og `fela`. Tegundin hefur 5 klasabreytur af tegundinni `int` sem geyma upplýsingar um stærð líkamshluta og bil milli þeirra og 3 breytur af tegundinni `Hringur` sem geyma líkamshluta pöddunnar.

Þar sem yfirklasinn `Kvikindi` hefur engan smið sem hægt er að kalla á án þess að senda honum gildi verður smiðurinn `Padda` að byrja á að kalla á smiðinn `Kvikindi` með skipuninni

```
super(g);
```

Eins og fjallað var um í kafla 5.b byrjar hver smiður á að kalla á smið yfirklasa. Sé skipunin `super` ekki notuð er sjálfkrafa kallað á smið yfirklasa án þess að senda honum neitt gildi.

Í smiðnum þarf varla að skýra fleira. Það sem er torskildast við aðferðirnar `syna` og `fela` er hvernig `fela`-aðferðin virkar. Hún gerir ekkert nema kalla á `syna`. Til að fela pöddu dugar semsagt að sýna hana upp á nýtt. Skýringin á þessu er skipunin

```
myndflotur.setXORMode(new Color(255,255,255));
```

A.d. Aðgerðin XOR og `setXORMode`

Breytan myndflötur er af tegundinni `Graphics` og hlutir af þeirri gerð geta verið í tvenns konar ham: `PaintMode` og `XORMode`. Þegar hlutur af tegundinni `Graphics` er fyrst búinn til er hann í `PaintMode` sem þýðir að litirnir sem teiknað er með leggjast

yfir það sem fyrir er. Sé hins vegar stillt á `XORMode` þá haga litirnir sé öðru vísi eins og þú getur séð með því að leysa verkefni A.2.

Verkefni A.2 *
Búðu til `Applet` sem hefur aðeins þessa einu aðferð:

```
public void paint(Graphics g)
{
   g.setXORMode(new Color(255, 255, 255));
   g.setColor(new Color(255, 0, 0));
   g.fillRect(50, 50, 100, 100);
   g.setColor(new Color(0, 255, 0));
   g.fillRect(25, 75, 150, 50);
   g.setColor(new Color(255, 0, 0));
   g.fillRect(100, 25, 25, 75);
}
```

Prófaðu að keyra `Applet`-ið og taktu eftir því hvernig litli ferningurinn þar sem teiknað er með rauðu ofan í rautt verður á litinn. Sjáðu svo hvernig hegðun þess breytist ef:

a) fyrstu skipuninni er breytt í:
```
   g.setXORMode(new Color(192, 192, 192));
```
b) stillt er á `PaintMode` í stað `XORMode` með því að sleppa fyrstu skipuninni.

Í Java er til aðgerð sem heitir XOR og er táknuð með `^`. Sé gefin skipunin

```
int x = 65 ^ 44
```

```
1 xor 1 = 0
1 xor 0 = 1
0 xor 1 = 1
0 xor 0 = 0
```

þá fær x gildið 109 vegna þess að í tvíundakerfi er talan 65 rituð 01000001 (þ.e. 64+1) og talan 44 rituð 00101100 (þ.e. 32+8+4) og útkoman úr því að beita XOR á þessar tvær tvíundakerfistölur er 01101101 sem er ritað 109 í tugakerfi (þ.e. 64+32+8+4+1).

```
      01000001
xor   00101100
      --------
      01101101
```

Aðgerðin XOR setur 0 þar sem bitarnir í báðum tölunum eru eins en 1 þar sem þeir eru ekki eins. Orðið XOR er skammstöfun á „exclusive or“ sem þýðir annar og aðeins annar.

Litir eru táknaðir með þrem tölum milli 0 og 255, þ.e. þrem tölum sem hver um sig er rituð með 8 stöfum í tvíundakerfi. Dæmi:

```
Hvítt er 255 255 255 eða í tvíundakerfi 11111111 11111111 11111111
Rautt er 255   0   0 eða í tvíundakerfi 11111111 00000000 00000000
Grátt er 192 192 192 eða í tvíundakerfi 11000000 11000000 11000000
```

Gerum ráð fyrir að breytan `h` geymi hvítan lit, breytan `r` rauðan, bakgrunnurinn sem teiknað er á sé grár, `g`, myndflöturinn heiti `m` og gefnar séu skipanirnar

```
m.setColor(r);
m.setXORMode(h);
```

og síðan teiknað. Liturinn sem kemur á skjáinn er fenginn með (h XOR g) XOR r. Stærðin í sviganum (h XOR g) er reiknuð svona:

```
        11111111 11111111 11111111
xor     11000000 11000000 11000000
        -------------------------
        00111111 00111111 00111111
```

og (h XOR g) XOR r er

```
        00111111 00111111 00111111
xor     11111111 00000000 00000000
        -------------------------
        11000000 00111111 00111111 = 192 63 63 (dumbrautt)
```

Það merkilegasta við `XORMode` er að sé aftur teiknað með sama lit ofan í mynd þá hverfur hún og flöturinn sem teiknað er á fær aftur sinn upprunalega lit. Sé til dæmis reiknað með rauðu ofan í dumbrauða litinn í þessu dæmi verður útkoman grá eins og upprunalegi bakgrunnurinn því (h XOR dumbrautt) er

```
        11111111 11111111 11111111
xor     11000000 00111111 00111111
        -------------------------
        00111111 11000000 11000000
```

og ((h XOR dumbrautt) XOR r) er

```
        00111111 11000000 11000000
xor     11111111 00000000 00000000
        -------------------------
        11000000 11000000 11000000 = 192 192 192 (grátt)
```

Þetta þýðir að sé stillt á `XORMode` og mynd teiknuð og svo aftur teiknuð alveg eins mynd ofan í hana þá hverfur hún og eftir verður upprunalegur litur eins og var á skjánum áður en teiknað var í fyrra skiptið. Þetta er skýringin á því að hægt er að fela pöddu með því einfaldlega að teikna hana upp á nýtt og aðferðin `fela` þarf ekkert annað að gera en að framkvæma `syna`.

A.e. `Padda` teiknar nokkur strik og fleiri dýr búin til

Nú þegar búið er að skilgreina pöddu er mál til komið að búa til forrit sem lætur slíkt kvikindi gera eitthvað. Ef þú hefur leyst verkefni A.1 kannastu við þetta forrit.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0A_01
// Dyr1
//
public class Dyr1 extends Applet
{
   Padda p;
   Button bFram, bHaegri, bVinstri;
   Graphics g;
```

```
   public void init()
   {
      bFram = new Button("Fram");
      bHaegri = new Button("Hægri");
      bVinstri = new Button("Vinstri");
      this.add(bFram);
      this.add(bHaegri);
      this.add(bVinstri);
      g = this.getGraphics();
      p = new Padda(g);
      p.xhnit = 100;
      p.yhnit = 100;
      p.litur = new Color(0, 0, 255);
      p.liturStriks = new Color(255, 0, 0);
      p.dregurStrik = true;
   }

   public void paint(Graphics g)
   {
      p.syna();
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bFram) {p.fram(10); return true;}
      if (e.target == bHaegri) {p.haegri(10);  return true;}
      if (e.target == bVinstri) {p.vinstri(10); return true;}
      return false;
   }
}
```

Verkefni A.3 *
Notaðu klasann `Dyr1` sem fyrirmynd og búðu til forrit sem lætur tvær pöddur, eina græna og eina gula, þvælast um. Forritið á að hafa þrjá takka fyrir hvora pöddu (einn fyrir áfram, einn fyrir vinstri beygju og einn fyrir hægri beygju).

Verkefni A.4
Breyttu lausninni á A.3 þannig að pöddurnar séu þrjár og takkarnir aðeins 3 (einn fyrir áfram, einn fyrir vinstri beygju og einn fyrir hægri beygju) og hægt sé að stjórna því hvaða padda hreyfist með því að skrifa 1, eða 2 eða 3 í textareit.

Verkefni A.5
Búðu til nýja tegund kvikinda og láttu hana heita `Maur` og vera eins og `Padda` nema helmingi minni. Breyttu svo lausninni á A.3 þannig að í staðinn fyrir tvær pöddur sé ein padda og einn maur.

Í því sem eftir er af þessum kafla og í næstu tveim verða klasarnir `Kvikindi`, `Hringur` og `Padda` sem hér hafa verið skilgreindir notaðir og tveir til viðbótar. Þeir koma hér og heita `Bjalla` og `Ormur`.

```
import java.awt.*;
//
// forrit_0A_01
// Bjalla
//
public class Bjalla extends Kvikindi
{
   private final int b = 5;      // Radíus búks
   private final int h = 2;      // Radíus hauss
   private final int s = b+h;    // Fjarlægð frá miðjum
                                 // búk í miðjan haus.
   private Hringur haus, bukur;

   public Bjalla(Graphics g)
   {
      super(g);
      float hausXhnit = Math.round(xhnit +
                           s * Math.cos(stefna*Math.PI/180));
      float hausYhnit = Math.round(yhnit -
                           s * Math.sin(stefna*Math.PI/180));
      haus = new Hringur(h, hausXhnit, hausYhnit);
      bukur = new Hringur(b, xhnit, yhnit);
   }

   public void syna()
   {
      bukur.xHnitMidju = xhnit;
      bukur.yHnitMidju = yhnit;
      haus.xHnitMidju = Math.round(xhnit +
                           s * Math.cos(stefna*Math.PI/180));
      haus.yHnitMidju = Math.round(yhnit -
                           s * Math.sin(stefna*Math.PI/180));
      myndflotur.setColor(litur);
      myndflotur.setXORMode(new Color(255,255,255));
      bukur.syna(myndflotur);
      haus.syna(myndflotur);
   }

   public void fela()
   {
      syna();
   }
} // Hér endar klasinn Bjalla



import java.awt.*;
//
// forrit_0A_01
// Ormur
//
public class Ormur extends Kvikindi
{
   private final int s = 5;   // s er fjórðungur af lengd ormsins

   public Ormur(Graphics g)
   {
      super(g);
   }
```

```
   public void syna()
   {
      double dx = s*Math.cos(stefna*Math.PI/180);
      double dy = s*Math.sin(stefna*Math.PI/180);
      int midja1x = (int)(xhnit - dx);
      int midja1y = (int)(yhnit + dy);
      int midja2x = (int)(xhnit + dx);
      int midja2y = (int)(yhnit - dy);
      myndflotur.setColor(litur);
      myndflotur.setXORMode(new Color(255,255,255));
      myndflotur.drawArc(midja1x-s, midja1y-s, 2*s, 2*s,
                         (int)stefna, 180);
      myndflotur.drawArc(midja2x-s, midja2y-s, 2*s, 2*s,
                         (int)stefna+180, 180);
   }

   public void fela()
   {
      syna();
   }
} // Hér endar klasinn Ormur
```

A.f. Tátugrafík

Þegar teiknað er á tölvuskjá er stundum notað hnitkerfi eins og kynnt var í 5. kafla. Til að teikna t.d. rétthyrndan jafnarma þríhyrning á myndflötinn, `g`, eru þá notaðar skipanir eins og

```
g.drawLine(50, 50, 100, 50);
g.drawLine(100, 50, 100, 100);
g.drawLine(100, 100, 50, 50);
```

En stundum eru myndir teiknaðar með skipunum á borð við `fram`, `haegri` og `vinstri` sem eru innbyggðar í kvikindin sem hér hafa verið til umfjöllunar. Hægt er að nota þessar skipanir til að láta pöddu, `p`, draga rétthyrndan jafnarma þríhyrning svona:

```
p.fram(50);
p.haegri(90);
p.fram(50);
p.haegri(135);
p.fram(Math.sqrt(50*50 + 50*50)); // Lengd á langhlið reiknuð með
p.haegri(135);                    // reglu Pýþagórasar.
```

Tölvugrafík sem byggist fyrst og fremst á skipunum eins og `fram`, `haegri` og `vinstri` (þ.e. tilfærslu fremur en hnitum) er kölluð tátugrafík. Hægt er að nota klasana `Padda`, `Bjalla` og `Ormur` til að teikna myndir með tátugrafík.

Verkefni A.6 *
Búðu til forrit sem lætur pöddu draga ferning á skjáinn með 50 skrefa löngum hliðum.

Verkefni A.7 *
Láttu pöddu draga reglulegan fimmhyrning á skjáinn með 50 skrefa löngum hliðum.

Verkefni A.8 *
Láttu pöddu teikna hring með radíus 50. (Ath. ekki er hægt að búa til fullkominn hring með skipununum `fram`, `haegri` og `vinstri` en það er hægt að búa til feril sem lítur út eins og hringur með því að teikna reglulegan marghyrning með mjög mörgum hornum. Á venjulegum tölvuskjá sést t.d. lítill munur á 45-hyrningi og hring).

Hér kemur dæmi um forrit sem lætur pöddu teikna ögn flóknari mynd.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0A_01
// Dyr2
//
public class Dyr2 extends Applet
{
   Padda p;
   Graphics g;

   public void init()
   {
      g = this.getGraphics();
      p = new Padda(g);
      p.xhnit = 100;
      p.yhnit = 100;
      p.dregurStrik = true;
   }

   public void paint(Graphics g)
   {
      p.syna();
      for (int j=5; j<500; j+=5)
      {
         p.fram(j*0.01);
         p.vinstri(5);
      }
   }
}
```

Verkefni A.9
Prófaðu að keyra `Dyr2` og reyndu að skilja hvernig forritið virkar. Breyttu því svo þannig að spírallinn fari í 3 hringi.

A.g. Fylki af kvikindum

Eins og minnst hefur verið á er vandræðalaust að geyma hlut af tegundinni `Padda` í breytu af tegundinni `Kvikindi` (því `Padda` er `Kvikindi`). Ef þetta er gert er aðeins hægt að láta pödduna framkvæma þær aðferðir sem eru skilgreindar í klasanum `Kvikindi` en það kemur ekki að sök því í klasanum `Padda` eru engar aðferðir til viðbótar við þær sem `Kvikindi` býr yfir. Þar eru aðeins útfærðar aðferðirnar `syna` og `fela` sem eru skilgreindar sem abstract aðferðir í `Kvikindi`. `Padda` sem er geymd í breytu af tegundinni `Kvikindi` getur því gert allt sem pöddur yfirleitt geta.

Eftirfarandi forrit geymir þrjú kvikindi (eina pöddu, eina bjöllu og einn orm) í fylki. Taktu eftir því hversu auðvelt er að láta öll kvikindin í fylkinu ferðast um eins og hér er gert í `paint`-aðferðinni.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0A_01
// Dyr3
//
public class Dyr3 extends Applet
{
   Kvikindi[] k;
   Graphics g;

   public void init()
   {
      k = new Kvikindi[3];
      g = this.getGraphics();
      k[0] = new Padda(g);     // Það er hægt að setja hlut af
      k[0].xhnit = 50;         // tegundinni Padda (eða Ormur eða
      k[0].yhnit = 150;        // Bjalla) í breytu af tegundinni
                               // Kvikindi. Allt sem er af tegundinni
      k[1] = new Ormur(g);     // Padda er líka af tegundinni Kvikindi
      k[1].xhnit = 100;        // þar sem Padda er undirklasi (eða
      k[1].yhnit = 150;        // undirtegund) Kvikindi. Fylkið K
                               // hefur rúm fyrir 3 Kvikindi og í það
      k[2] = new Bjalla(g);    // er hér sett eitt Kvikindi af hverri
      k[2].xhnit = 150;        // tegundanna: Padda, Bjalla og Ormur.
      k[2].yhnit = 150;        // Þar sem k er skilgreint sem fylki
                               // af Kvikindum geta hlutirnir í k
      // aðeins framkvæmt þær aðferðir sem tilheyra tegundinni
      // Kvikindi. Þar á meðal eru abstract aðferðirnar syna og fela
      // sem eru útfærðar  í undirklösunum.

      for (int i=0; i<k.length; i++)
      {
         k[i].dregurStrik = true;
      }
   }


   public void paint(Graphics g)
   {
      for (int i=0; i<k.length; i++)
      {
         k[i].syna();
      }

      for (int j=5; j<500; j+=5)
      {
         for (int i=0; i<k.length; i++)
         {
            k[i].fram(j*0.01);
            k[i].vinstri(5);
         }
      }
   }
}
```

Verkefni A.10
Notaðu `Dyr3` sem fyrirmynd og búðu til forrit sem lætur fylki af 5 kvikindum teikna hvert sinn hringinn þannig að útkoman verði eins og merki Ólympíuleikanna.

A.h. Undirforrit og færibreytur

Þær aðferðir sem hafa verið byggðar til þessa hafa flestar verið einfaldar. Flóknari aðferðir eru yfirleitt samsettar úr mörgum einföldum. Þegar einfaldar aðferðir eru notaðar sem byggingareiningar í flóknari aðferðir er þær kallaðar undirforrit. Undirforrit er sjálfstæð eining sem sinnir einum verkþætti eða hluta af samsettu verki. Hér fer á eftir klasi sem inniheldur meðal annars einfalda aðferð til að smíða ferning. Þessa aðferð er hægt að nota til að byggja eitthvað flóknara.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0A_01
// Dyr4
//
public class Dyr4 extends Applet
{
   Bjalla b;
   Graphics g;

   public void init()
   {
      g = this.getGraphics();
      b = new Bjalla(g);
      b.xhnit = 100;  b.yhnit = 100;
      b.dregurStrik = true;
   }

   public void ferningur(Kvikindi k, double lengdHlida)
   {
      for (int i=1; i<=4; i++)    // Ath. breyta af tegundinni
      {                           // Kvikindi getur tekið við
         k.fram(lengdHlida);      // hlut af tegundinni Bjalla
         k.haegri(90);            // því Bjalla er undirklasi
      }                           // tegundarinnar Kvikindi.
   }

   public void paint(Graphics g)
   {
      b.syna();
      ferningur(b, 60);
   }
}
```

Hér er aðferðinni `ferningur` sent eitt kvikindi (þ.e. dýrið sem á að teikna ferninginn) og ein tala (sem segir hvað hann á að vera stór). Það kann að virðast óþarfi að senda aðferðinni hlut af tegundinni `Kvikindi`. Væri ekki hægt að láta hana nota klasabreytuna `b` og hafa hana svona:

```
public void ferningur(double lengdHlida)
{
   for (int i=1; i<=4; i++)
   {
      b.fram(lengdHlida);
      b.haegri(90);
   }
}
```

og láta `paint`-aðferðina framkvæma hana með því að segja bara

```
ferningur(60);
```

Þetta væri vissulega hægt en það hefur samt ókosti í för með sér að láta aðferðir nota klasabreytur. Aðferð sem hefur engar klasabreytur heldur aðeins staðværar breytur og færibreytur getur staðið óbreytt þótt hún sé flutt í aðra klasa eða nöfnum á klasabreytum sé breytt.

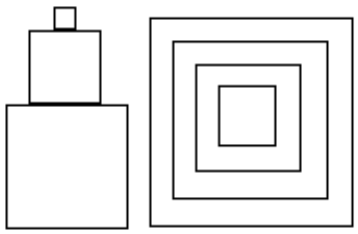
Það er hægt að nota aðferðina ferningur í `Dyr4` algerlega óbreytta þó klasanum sé breytt þannig að kvikindið sem teiknar sé ekki lengur `Bjalla` sem heitir `b` heldur t.d. `Padda` sem heitir `p`.

Verkefni A.11*
Prófaðu að breyta `paint`-aðferðinni í `Dyr4` þannig að hún verði svona:

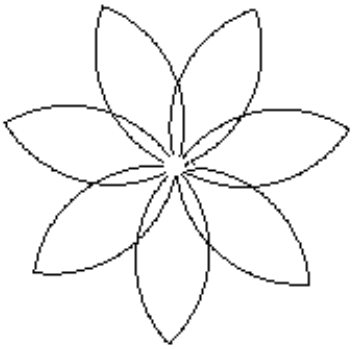
```
public void paint(Graphics g)
{
   b.syna();
   for (int i=1; i<=10; i++)
   {
      ferningur(b, 60);
      b.haegri(36);
   }
}
```

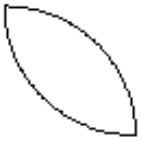
Verkefni A.12
Breyttu `paint`-aðferðinni í `Dyr4` þannig að hún noti aðferðina `ferningur` til að teikna mynd eins og er hér að neðan til vinstri.

Verkefni A.13
Breyttu `paint`-aðferðinni í `Dyr4` þannig að hún noti aðferðina `ferningur` til að teikna mynd eins og er hér að neðan til hægri. Til að leysa þetta verkefni þarf að færa kvikindi til án þess að það dragi strik. Þetta er hægt að gera með því að gefa klasabreytunni `dregurStrik` gildið `false`.



Hér til hægri er dæmi um nokkuð flókna mynd af blómi. Það er erfitt að búa til aðferð til að teikna svona flókinn hlut. En ef búið er að búa til aðferð til að teikna lauf eins og er hér til vinstri þá er vandalaust að teikna blóm með því að gera lauf aftur og aftur og beygja á milli.





Hugsum okkur að titillína aðferðarinnar til að teikna lauf sé svona:

```
public void lauf(Kvikindi k, double radius)
```

og breyturnar `k` og `radius` hafi fengið gildi, þá er hægt að teikna blóm með skipununum:

```
for (int i=1; i<=fjoldiKronublada; i++)
{
   lauf(k, radius);
   k.haegri(360/fjoldiKronublada);
}
```

Skynsamlegasta leiðin til að búa til forrit sem teiknar blóm er að byrja á að ímynda sér að búið sé að búa til aðferð til að teikna lauf. Það er svo hægt að velta því fyrir sér seinna hvernig á að gera hana.

Það er ekki auðvelt að teikna lauf en ef búið er að smíða aðferð til að teikna boga eins og er hér að ofan til hægri þá er vandalaust að búa til lauf.

Hugsum okkur að aðferðin til að teikna boga hafi titillínuna:

```
public void hbogi(Kvikindi k, double radius, int gradur)
```

þá er hægt að búa til aðferð til að teikna lauf úr tveim bogum svona:

```
public void lauf(Kvikindi k, double radius)
{
   hbogi(k, radius, 90);
   k.haegri(90);
   hbogi(k, radius, 90);
   k.haegri(90);
}
```

Við höfum nú einfaldað vandamálið að teikna blóm. Það eina sem á eftir að gera er að búa til aðferð til að teikna boga.

Hér fer á eftir klasi sem notar þessar aðferðir til að láta pöddu draga upp mynd af blómi. Taktu eftir því að aðferðirnar nota engar klasabreytur heldur eingöngu færibreytur og staðværar breytur.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0A_01
// Dyr5
//
public class Dyr5 extends Applet
{
   Bjalla b;
   Graphics g;
```

```
   public void init()
   {
      g = this.getGraphics();
      b = new Bjalla(g);
      b.xhnit = 100;
      b.yhnit = 100;
      b.dregurStrik = true;
   }

   public void hbogi(Kvikindi k, double radius, int gradur)
   {
      double skref = 2*Math.PI*radius/360;
      for (int i=1; i<=gradur; i++)
      {
         k.fram(skref);
         k.haegri(1);
      }
   }

   public void lauf(Kvikindi k, double radius)
   {
      hbogi(k, radius, 90);
      k.haegri(90);
      hbogi(k, radius, 90);
      k.haegri(90);
   }

   public void blom(Kvikindi k, double radius, int fjoldiKronublada)
   {
      for (int i=1; i<=fjoldiKronublada; i++)
      {
         lauf(k, radius);
         k.haegri(360/fjoldiKronublada);
      }
   }

   public void paint(Graphics g)
   {
      b.syna();
      b.liturStriks = new Color(255, 0, 0);
      blom(b, 60, 7);
   }
}
```

Verkefni A.14
Notaðu aðferðina `hbogi` úr `Dyr5` til að smíða aðferðir til að teikna hjarta og spaða eins og eru á spilum.

Verkefni A.15
Búðu til forrit sem teiknar mynd eins og er hér fyrir neðan. (Ath. þú getur notað `hbogi` úr `Dyr5` til að gera öldutoppana en til að gera öldudalina er best að nota svipaða aðferð með vinstri-beygju. Hún gæti t.d. heitið `vbogi`.)

A.x. Spurningar og umhugsunarefni

1. Að hvaða leyti er skilgreining á `abstract` aðferð ólík skilgreiningum á öðrum aðferðum?
2. Hvað er hægt að gera með `abstract` aðferðum sem er ekki hægt að gera án þeirra?
3. Hvað er ekki hægt að gera við klasa sem inniheldur `abstract` aðferðir?
4. Af hverju er ekki hægt að nota eftirfarandi skipanir til að búa til hlut af tegundinni `Graphics`?

```
Graphics x;
x = new Graphics();
```

5. Hvað er ekki hægt að gera við klasa sem eru `final` og hvaða kosti hefur það að skilgreina klasa sem `final`?
6. Undir hvaða kringumstæðum er smiður `abstract` tegundar (klasa) framkvæmdur fyrst ekki er hægt að búa til einstaka hluti af henni?
7. Hvaða gildi fær `x` ef þessi skipun er gefin?

```
int x = 10^12;
```

8. Hvaða munur er á `paintMode` og `XORMode`?
9. Hvað er tátugrafík?
10. Hvaða aðferðir getur hlutur af tegundinni `Padda` framkvæmt ef hann er geymdur í breytu sem er skilgreind sem `Object`?
11. Hugsaðu þér að klasinn `Lifvera` sé skilgreindur svona:

```
public abstract class Lifvera
{
   public abstract void eta();
   public abstract void vaxa();
   public abstract void deyja();
}
```

    Hvaða aðferðir verða undirklasar `Lifvera` að innihalda?

12. Hvað er undirforrit?
13. Hvaða kostir fylgja því að láta aðferðir ekki nota neinar klasabreytur heldur eingöngu staðværar breytur og færibreytur?

Til umhugsunar

Undir lok þessa kafla var fjallað um hvernig hægt er að byggja flóknar aðferðir úr öðrum einfaldari. Eitt mikilvægasta viðfangsefni tölvufræðinnar er að móta aðferðir til að henda reiður á flóknum verkefnum og skipta þeim í einfaldari verkþætti.

Sú aðferð til að takast á við flókin verk sem hér var kynnt er kölluð ofansækin (á ensku „top-down“) því það er byrjað á toppnum, ef svo má segja, og undirstöðurnar (einföldustu verkþættirnir) smíðaðar síðast. Þessari aðferð má beita á alls konar viðfangsefni. Við getum t.d. skrifað aðferð til að baka súkkulaðiköku svona:

```
void bakaSúkkulaðiköku()
{
   búaTilKökubotn();
   BúaTilSúkkulaðikrem();
   setjaSúkkulaðikremiðÁKökubotninn();
}
```

Nú er búið að skipta verkinu í þrjá einfaldari þætti. Þá er að ráðast á þann fyrsta þeirra:

```
void búaTilKökubotn()
{
   hræraDeig();
   hitaOfn();
   setjaDeigÍMót();
   setjaMótÍOfn();
   bíða();
   takaMótÚtÚrOfni();
}
```

Hér er búið að skipta þessum þætti verksins niður í sex einfaldari þætti. Sumum þeirra má svo skipta í enn einfaldari. Ef súkkulaðikökuforritið á að framkvæmast af mennskum manni er eðlilegast að halda áfram að skipta verkþáttum niður í einfaldari undirþætti þar til við komum að skipunum sem ætla má að hver maður geti unnið eftir, án frekari útskýringa. Séum við hins vegar að skrifa forrit fyrir tölvu þá höldum við áfram að skipta viðfangsefninu niður í sífellt einfaldari verkþætti þar til kemur að verkþáttum sem eru nógu einfaldir til að auðvelt sé að lýsa þeim með orðaforðanum sem er innbyggður í forritunarmálið og þá klasa sem fylgja með því.

Hvernig er eðlilegast að skipta þessum verkum niður í einfaldari verkþætti?

a) Skipta um dekk á bíl.
b) Þvo þvott í þvottavél og hengja hann til þerris.
c) Sortera spilastokk þannig að öll rauðu spilin lendi í einum bunka og öll þau svörtu í öðrum.

Sum þessara verka eru þess eðlis að til að vinna þau þarf að endurtaka sömu aðgerðir nokkrum sinnum eða gá hvort einhver skilyrði eru uppfyllt. Sama má reyndar segja um aðferðina við að baka súkkulaðiköku. Einn undirþáttur aðferðarinnar við að hræra deig er kannski í því fólginn að brjóta nokkur egg. Þann undirþátt mætti ef til vill útfæra svona:

```
void brjótaEgg(int n)
{
   for (int i=0; i<n; i++)
   {
      brjótaEittEgg();
   }
}
```

Hver af verkunum sem talin voru upp undir liðum a, b, og c hér að ofan eru þess eðlis að það þurfi að nota endurtekningu (slaufu) eða skilyrði (if-skipun) til að útfæra þau?

B. kafli: Þræðir

B.a. Þræðir og kvikmyndir

Einn af klösunum sem fylgja með Java í pakkanum `java.lang` heitir `Thread`. Þessi klasi og undirklasar hans kallast þræðir. `Thread` hefur m.a. aðferðir sem heita `run`, `start`, `stop`, `suspend` og `resume`. Undirklasar `Thread` þurfa að hafa sína eigin `run`-aðferð sem yfirskyggir `run`-aðferð `Thread`.

Með því að nota þræði er hægt að láta forrit vinna mörg verk samtímis. Til að setja þráð í gang er aðferðin `start` framkvæmd. Hún setur `run`-aðferð þráðarins í gang.

Hugsum okkur að `t` sé af einhverri tegund og gefnar séu skipanirnar

```
skipunA();
t.adferd1();
skipunB();
skipunC();
```

þá er byrjað á að framkvæma `skipunA` og þegar því er lokið er `t` látinn framkvæma `adferd1`. `skipunB` er svo ekki framkvæmd fyrr en `t` hefur lokið við að framkvæma `adferd1` og þegar `skipunB` er lokið er tekið til við `skipunC`.

Hugsum okkur nú að `t` sé þráður og gefnar séu skipanirnar

```
skipunA();
t.start();
skipunB();
skipunC();
```

þá er byrjað á að framkvæma `skipunA`, síðan er `t` látinn framkvæma `start`-aðferðina sem setur `run`-aðferð `t` í gang en það er ekki beðið með að framkvæma `skipunB` þar til `run`-aðferinni lýkur heldur eru `skipunB` og svo `skipunC` framkvæmdar meðan `run`-aðferð þráðarins `t` er í gangi.

Keyrslu þráðar lýkur (hann deyr sem kallað er) ef `run`-aðferðin lýkur keyrslu eða aðferðin `stop` er framkvæmd. Oft inniheldur `run` endalausa slaufu og þá heldur þráðurinn áfram þar til `stop`-aðferðin er framkvæmd.

Hægt er að stöðva keyrslu þráðar tímabundið með `suspend` og setja hana aftur í gang með `resume`.

Java forrit sem ekki notar `Thread` eða undriklasa hans er í raun og veru einn þráður. Samhliða keyrslu hans fer fram ruslatínsla því túlkurinn sem keyrir klasana sér sjálfkrafa um að henda úr minni tölvunnar hlutum sem horfnir eru úr notkun. Ruslatínslan er annar þráður. Við verðum ekkert vör við hann. En ef hann væri ekki keyrður er hætt við að minni tölvunnar fylltist við langvarandi keyrslu stórra forrita.

Forrit_0B_01 sýnir hvernig þræðir eru búnir til og hvernig aðferðirnar `start`, `stop`, `suspend` og `resume` eru notaðar. Forritið er myndað úr tveim klösum sem heita `Kvikmyndir` og `Traedir`. Þeir eru báðir í sömu skrá enda er allt í lagi að hafa marga klasa í sömu skránni ef aðeins einn þeirra er `public`. (Skráin verður þá að heita sama nafni og `public` klasinn.)

Klasinn `Thradur` er þráður. Hann erfir `start`, `stop`, `suspend` og `resume` aðferðir `Thread` en hefur sína eigin `run` aðferð eins og allir þræðir. Smiðurinn `Thradur` tekur við einum hlut af tegundinni `Kvikmyndir` og þrem heiltölum. Fyrstu tvær tölurnar

segja hvar á að setja myndirnar og sú síðasta hvað hver mynd á að vera margar millisekúndur á skjánum.

Smiðurinn les 5 myndir inn í fylki af myndum. Til þess notar hann aðferðina `getImage` sem tilheyrir klasanum `Applet`. Með þessari aðferð er hægt að lesa myndir á gif og jpg formi. Þar sem aðferðin tilheyrir `Applet` þarf smiðurinn hlut af þeirri tegund til að framkvæma aðferðina. `Kvikmyndir` er undirklasi `Applet` og færibreytan `k` er af þeirri tegund.

Aðferðin `getImage` tekur við tveim strengjum sem eru staðsetning (þ.e. efnisskrá) og heiti skráar. Hér er aðferðin `getCodeBase` látin sjá um að skila staðsetningu myndar. Þessi aðferð tilheyrir klasanum `Applet` og skilar heiti efnisskrárinnar sem `Applet`-ið var sótt úr. Til að hægt sé að sækja mynd í fyrsta hólf fylkisins `myndir` með skipuninni

```
myndir[0] = k.getImage(k.getCodeBase(), "oli1.gif");
```

þarf hún að vera í skrá sem heitir oli1.gif og er staðsett í sömu efnisskrá og `Applet`-ið `k`. Til að forritið virki rétt þurfa skrárnar oli1.gif, oli2.gif, … oli5.gif að vera í sömu efnisskrá og forritið.

Aðferðin `run` inniheldur endalausa slaufu sem sýnir myndirnar fimm í röð aftur og aftur. Aðferðin `bida` lætur þráðinn bíða, eða „sofa“ sem kallað er. Hún tekur við tölu sem segir hvað á að „sofa“ í margar millisekúndur. Þessi aðferð verður útskýrð í kafla D.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0B_01
// Kvikmyndir
//
class Tradur extends Thread
{
   Graphics g;
   Image[] myndir;
   Kvikmyndir km;
   int xHnit, yHnit, bidtimi;

   public Tradur(Kvikmyndir k, int x, int y, int bid)
   {
      g = k.getGraphics();
      km = k;
      bidtimi = bid;
      xHnit = x; yHnit = y;
      myndir = new Image[5];
      myndir[0] = k.getImage(k.getCodeBase(), "oli1.gif");
      myndir[1] = k.getImage(k.getCodeBase(), "oli2.gif");
      myndir[2] = k.getImage(k.getCodeBase(), "oli3.gif");
      myndir[3] = k.getImage(k.getCodeBase(), "oli4.gif");
      myndir[4] = k.getImage(k.getCodeBase(), "oli5.gif");
   }

   void bida(long t)
   {
      try {this.sleep(t);}
      catch (InterruptedException e) {}
   }
```

```
   public void run()
   {
      while (true) // Slaufan fer endalaust hring eftir hring.
      {
         for (int i = 0; i < 5; i++)
         {
            g.drawImage(myndir[i], xHnit, yHnit, km);
            bida(bidtimi);
         }
      }
   }
}


public class Kvikmyndir extends Applet
{
   Tradur t1, t2;
   Button t1AfStad, t2AfStad, t1Bida, t2Bida;

   public void init()
   {
      t1AfStad = new Button("Nr. 1 í gang");
      t1Bida   = new Button(" Nr. 1 bíði ");
      t2AfStad = new Button("Nr. 2 í gang");
      t2Bida   = new Button(" Nr. 2 bíði ");
      this.add(t1AfStad); this.add(t2AfStad);
      this.add(t1Bida); this.add(t2Bida);
      // Hlutur af tegundinni Kvikmyndir sendir smiðnum Tradur this,
      // þ.e. sjálfan sig.
      t1 = new Tradur(this,  45, 100, 800);
      t2 = new Tradur(this, 105, 100, 200);
      t1.start();    // Setur run aðferð t1 af stað.
      t2.start();    // Setur run aðferð t2 af stað.
   }


   public void destroy()   // Applet framkvæmir destroy aðferðina
   {                       // þegar það hættir keyrslu.
      t1.stop();           // Stöðvar keyrslu t1.
      t2.stop();           // Stöðvar keyrslu t2.
   }


   public boolean action(Event e, Object o)
   {
      if (e.target == t1AfStad)
      {
         t1.resume();   // Setur þráðinn t1 í gang
         return true;   // ef hann er óvirkur.
      }
      if (e.target == t1Bida)
      {
         t1.suspend();  // Gerir þráðinn t1 óvirkan
         return true;   // þar til hann framkvæmir
      }                 // resume.
      if (e.target == t2AfStad)
      {
         t2.resume();
         return true;
      }
```

```
      if (e.target == t2Bida)
      {
         t2.suspend();
         return true;
      }
      return false;
   }
}
```

`init`-aðferð klasans `Kvikmyndir` býr til tvo þræði af tegundinni `Thradur` og setur þá báða í gang. Aðferðin `action` sér um að stöðva þræði og setja aftur af stað þegar ýtt er á viðeigandi takka.

Þegar `Applet` hættir keyrslu framkvæmir það aðferð sem heitir `destroy`. Hér inniheldur hún skipanir sem stöðva (drepa) báða þræðina.

Hlutir af tegundinni `Thradur` þurfa að nota aðferðirnar `getGraphics` (til að ná í myndflöt sem hægt er að setja myndirnar á), `getImage` (til að lesa myndir af diski) og `getCodeBase` (til að finna í hvaða efnisskrá forritið er). Klasinn `Thread` inniheldur ekki þessar aðferðir en þær tilheyra allar klasanum `Applet` (og þar með `Kvikmyndir` sem erfir frá `Applet`). Tvær síðasttöldu aðferðirnar eru skilgreindar í `Applet` en `getGraphics`-aðferðin er arfur frá `Panel` sem erfir hana frá `Container` sem erfir hana frá `Component`. Til þess að `Thradur` geti notað þessar aðferðir sem tilheyra `Applet` þarf að senda smiðnum hlut af þeirri gerð. Einfaldasta aðferðin til þess er að láta hlutinn af tegundinni `Kvikmyndir` senda sjálfan sig. Hver hlutur kallar sjálfan sig `this` og þræðirnir `t1` og `t2` eru smíðaðir með

```
t1 = new Tradur(this,  45, 100, 800);
t2 = new Tradur(this, 105, 100, 200);
```

Verkefni B.1 *
Búðu til forrit sem spilar tvær ólíkar kvikmyndir. Til að gera þetta þarftu að byrja á að teikna tvær myndaseríur og vista myndirnar sem gif eða jpg. `Applet`-ið sem setur þræðina í gang getur verið alveg eins og `Kvikmyndir` nema hvað `t1` og `t2` þurfa að vera hvor af sinni gerð.

B.b. Lifandi kvikindi

Í síðasta kafla voru búnar til þrjár tegundir kvikinda. Hægt er að gæða þau lífi með því að breyta yfirklasanum `Kvikindi` úr forrit_0A_01 í þráð. Til að gera þetta dugar að breyta titillínu hans úr

```
public abstract class Kvikindi
```

í

```
public abstract class Kvikindi extends Thread
```

Til að þráður geri eitthvað verður hann að hafa `run`-aðferð. Ekki er þó nauðsynlegt að bæta `run`-aðferð við Kvikindi þar sem sú tegund er abstract og aldrei búnir til hlutir af henni. Nóg er að bæta `run`-aðferð við klasana `Bjalla`, `Padda` og `Ormur`.

Hér kemur `Bjalla` með `run`-aðferð. Klasinn `Hringur` sem `Bjalla` notar er óbreyttur frá forrit_0A_01 og í klasanum `Kvikindi` er aðeins titillínunni breytt.

```
import java.awt.*;
import java.util.Random;
//
// forrit_0B_02
// Bjalla
//
// Eins og Bjalla í forrit_0A_01 nema hér hefur run aðferð verið bætt
// við sem og aðferðunum rekstAVegg og bida. run notar
// java.util.Random svo sá klasi er fluttur inn hér fyrir ofan.
//
public class Bjalla extends Kvikindi
{
   private final int b = 5;          // Radíus búks
   private final int h = 2;          // Radíus hauss
   private final int s = b+h;        // Fjarlægð frá miðjum
                                     // búk í miðjan haus.
   private Hringur haus, bukur;

   public Bjalla(Graphics g)
   {
      super(g);
      float hausXhnit = Math.round(xhnit +
            s * Math.cos(stefna*Math.PI/180));
      float hausYhnit = Math.round(yhnit -
            s * Math.sin(stefna*Math.PI/180));
      haus = new Hringur(h, hausXhnit, hausYhnit);
      bukur = new Hringur(b, xhnit, yhnit);
   }

   public void syna()
   {
      bukur.xHnitMidju = xhnit;
      bukur.yHnitMidju = yhnit;
      haus.xHnitMidju = Math.round(xhnit +
                                   s * Math.cos(stefna*Math.PI/180));
      haus.yHnitMidju = Math.round(yhnit -
                                   s * Math.sin(stefna*Math.PI/180));
      myndflotur.setColor(litur);
      myndflotur.setXORMode(new Color(255,255,255));
      bukur.syna(myndflotur);
      haus.syna(myndflotur);
   }

   public void fela()
   {
      syna();
   }


   // Það sem er hér fyrir neðan er viðbót við Bjalla í forrit_0A_01.

   void bida(long t)
   {                                      // Í kafla D verður skýrt
      try {this.sleep(t);}                // hvað try og catch gera.
      catch (InterruptedException e) {}
   }
```

```
   public boolean rekstAVegg()
   {
      return   (xhnit < myndflotur.getClipRect().x + s) ||
               (yhnit < myndflotur.getClipRect().y + s) ||
               (xhnit > myndflotur.getClipRect().width - s) ||
               (yhnit > myndflotur.getClipRect().height - s);
   }

   public void run()
   {
      double skreflengd;
      Random r = new Random();
      while (true)
      {
         skreflengd = 2+r.nextGaussian();
         fram(skreflengd);
         if (rekstAVegg())
         {
            fram(-skreflengd);
            vinstri(180*r.nextGaussian());
         }
         vinstri(15*r.nextGaussian());
         bida(30);
      }
   }
}
```

Aðferðin `run` notar hlut af tegundinni `Random` sem er í pakkanum `java.util`. `Random` er slembitalnagjafi, þ.e. hlutur sem skilar tölum völdum af handahófi. Klasinn `Random` inniheldur m.a. aðferðina `nextGaussian`. Sú aðferð skilar gildi af tegundinni `double` og hagar sér þannig að sé henni beitt oft eru tölurnar sem út koma normaldreifðar með meðaltalið 0 og staðalfrávikið 1. (Þetta þýðir að oftast eða í um 68,25% tilvika kemur tala milli -1 og 1, í um það bil 95,5% tilvika er talan milli -2 og 2 og í 99,75% tilvika kemur tala milli -3 og 3.)

Klasinn `Random` inniheldur svokallað slembifall, þ.e. fall sem velur tölur af handahófi. Að vísu er ekki um eiginlega tilviljun að ræða heldur aðeins reikniformúlu sem hagar sér nógu óreglulega til þess að sé henni beitt aftur og aftur þá virðast útkomurnar vera handahófskenndar. Um slembiföll verður fjallað nánar í næsta kafla.

Aðferðin `run` lætur bjöllu þvælast um með því að fara aftur og aftur fram um `2+r.nextGaussian()` og beygja um `15*r.nextGaussian()`. Þetta þýður að í um að bil 99,75% tilvika fer hún fram um -1 til 5 skref og beygjan er oftast innan við 15 gráður til hægri eða vinstri og örsjaldan meira en 45 gráður.

Aðferðin `bida` er eins og í klasanum `Kvikmyndir` í forrit_0B_01. Hún verður útskýrð betur í kafla D.

Aðferðin `rekstAVegg` skilar `true` ef `Bjalla` er komin að jaðri myndflatar. Hún notar `getCliprect` aðferðina í klasanum `Garphics`. Sú aðferð skilar hlut af tegundinni `Rectangle` (ferhyrningur) með klasabreyturnar `x`, `y`, `width` og `height`, allar af tegundinni `int`. Þær tvær fyrsttöldu geyma x og y hnit hornsins efst til vinstri. Ferhyrningurinn sem `getClipRect` skilar er það svæði sem hægt er að teikna á. Hægt er að stilla stærð og staðsetningu þessa svæðis með aðferðinni `clipRect`. Dæmi um notkun hennar má sjá í klasanum `Dyralif1` sem fer hér á eftir. Hann ræsir tvær bjöllur og lætur þær þvælast um.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0B_02
// Dyralif1
//
public class Dyralif1 extends Applet
{
   Bjalla b1, b2;
   Graphics g;

   public void init()
   {
      g = this.getGraphics();
      g.clipRect(0,0,this.size().width, this.size().height);

      b1 = new Bjalla(g);
      b1.xhnit = 50;
      b1.yhnit = 100;
      // b1.litur = new Color(128, 64, 0);  // Gengur ekki

      b2 = new Bjalla(g);
      b2.xhnit = 150;
      b2.yhnit = 100;
      // b2.litur = new Color(64, 128, 0);  // Gengur ekki
   }

   public void paint(Graphics g)
   {
      b1.syna();
      b2.syna();
      b1.start();
      b2.start();
   }

   public void destroy()
   {
      b1.stop();
      b2.stop();
   }
}
```

Verkefni B.2
Keyrðu forrit_0B_02. Það samanstendur af klösunum `Kvikindi`, `Hringur`, `Bjalla` og `Dyralif1`. Bættu svo við klasann `Dyralif1` tökkum til að stöðva bjöllurnar (`suspend`) og setja þær aftur af stað (`resume`). Það þarf tvo takka fyrir hvora bjöllu.

Verkefni B.3
Bættu `bida`-, `rekstAVegg`- og `run`-aðferðum við `Orm` og `Padda` eins og gert var við `Bjalla` í forrit_0B_02 og búðu svo til forrit sem lætur eitt kvikindi af hverri tegund þvælast um skjáinn. Láttu pöddur fara tvöfalt hraðar en bjöllur og orma tvöfalt hægar. Hafðu leiðina sem ormarnir fara hlykkjóttari en leiðirnar sem bjöllur og pöddur fara.

Verkefni B.4
Í `init`-aðferð `Dyralif1` eru línurnar:

```
// b1.litur = new Color(128, 64, 0);  // Gengur ekki
```

og

```
// b2.litur = new Color(64, 128, 0);  // Gengur ekki
```

Þær innihalda skipanir sem gefa björnunum tveim ólíka liti en eru gerðar óvirkar með því að hafa // fremst. Gerðu þessar línur virkar með því að eyða // fremst úr þeim og prófaðu svo að keyra forritið. Reyndu að átta þig á hvers vegna það gengur ekki að hafa björnurnar í ólíkum litum.

B.c. `synchronized`

Ef þú hefur leyst verkefni B.4 hefur þú séð að þegar tvö dýr í ólíkum lit þvælast um myndflötinn þá mistekst stundum að stroka þau út og það verða eftir slettur á myndfletinum sem sumar eru eins og haus í laginu, sumar eins og búkur og sumar eins og heilt kvikindi. Ástæðan fyrir þessu er sú að tveir þræðir eru að teikna á myndflötinn samtímis. Til að teikna björn eru notaðar skipanirnar

```
myndflotur.setColor(litur);
   myndflotur.setXORMode(new Color(255,255,255));
   bukur.syna(myndflotur);
   haus.syna(myndflotur);
```

(Allar þessar skipanir eru í aðferðinni syna.) Það sem bjalla gerir til að sýna sig er semsagt tvennt. Hún stillir lit og teikniham myndflatar (með fyrri tveim skipununum) og teiknar haus og búk (með þeim seinni tveim). Ef tvær björn eru á ferð samtímis og það á að teikna báðar á myndlötinn þá er æskilegt að atburðir gerist í þessari röð:

bjalla 1 stillir lit;
bjalla 1 teiknar búk;
bjalla 1 teiknar haus;
bjalla 2 stillir lit;
bjalla 2 teiknar búk;
bjalla 2 teiknar haus.

En röð atburða getur alveg eins orðið þessi:

bjalla 1 stillir lit;
bjalla 1 teiknar búk;
bjalla 2 stillir lit;
bjalla 1 teiknar haus;
bjalla 2 teiknar búk;
bjalla 2 teiknar haus.

og þá verður hausinn á björn 1 í þeim lit sem tilheyrir björn 2. Þegar haus björn 1 er svo strokaður út með því að teikna ofan í hann þá hverfur hann ekki því hann er alls ekki í þeim lit sem notaður er til að teikna ofan í. (Þar sem teiknað er í `XORMode` hverfur mynd ef teiknað er ofan í hana með sama lit en ef teiknað er ofan í hana með öðrum lit verður útkoman sá litur sem fæst með því að beita XOR eins og skýrt var í kafla A.d.)

Vandinn kemur til af því að tveir þræðir nota sama myndflöt og spilla hvor annars stillingum á honum. Til að leysa vandann þarf að tryggja þráðum tímabundinn einka-

rétt á notkun myndflatarins. Meðan bjalla númer 1 er að færa sig þarf að koma í veg fyrir að bjalla númer 2 framkvæmi aðgerðir á myndfletinum. Þetta er hægt að gera með skipuninni `synchronized`. Hún er notuð svona:

```
synchronized(x)
{
   // Meðan verið er að framkvæma skipanirnar hér innan
   // slaufusviganna er x læstur
}
```

Skipunin læsir hlut sem kallað er. Þetta þýðir að á meðan verið er að framkvæma skipanirnar í `synchronized`-blokkinni (þ.e. innan slaufusviganna) getur enginn annar þráður læst sama hlut. Ef annar þráður „ætlar“ að beita `synchronized`-skipun á sama hlut verður hann að bíða meðan þessi hlutur klárar `synchronized`-blokkina. Af þessu leiðir að ef allar aðferðir sem breyta ástandi hlutarins `x` eru innan `synchronized` blokka sem læsa `x` þá er útilokað að margir þræðir beiti þeim samtímis.

Hægt er að skilgreina heilar aðferðir sem `synchronized` t.d. svona:

```
public synchronized void foo()
{
   // Aðferðin er synchronized svo meðan hlutur beitir henni
   // getur sami hlutur ekki beitt neinni annarri synchronized
   // aðferð. Þetta útilokar að tveir þræðir láti hlut beita
   // synchronized aðferð samtímis.
}
```

Í forrit_0B_03 er skipunin `synchronized` notuð til að tryggja að kvikindi spilli ekki stillingum myndflatar hvort fyrir öðru. Skipunin er sett inn í aðferðirnar `fram`, `haegri` og `vinstri` í klasanum `Kvikindi`.

Inn í `Kvikindi` hefur líka verið bætt aðferðunum `bida` og `rekstAVegg` sem voru partar af klasanum `Bjalla` í forrit_0B_02. Þessar aðferðir eru þar með sameiginlegar öllum kvikindum hvort sem þau eru pöddur, bjöllur eða ormar.

Hér fara á eftir klasarnir `Kvikindi` og `Dyralif2` úr forrit_0B_03. Aðrir klasar eru ekki prentaðir. (Klasinn `Hringur` er óbreyttur frá forrit_0A_01. `Bjalla` er eins og í forrit_0B_02 nema hvað aðferðinar bida og `rekstAVegg` hafa verið teknar burt. Klasarnir `Padda` og `Ormur` eru eins og í forrit_0A_01 nema hvað `run`-aðferð hefur verið bætt við þá.)

```
import java.awt.*;
//
// forrit_0B_03
// Kvikindi
//
// Eins og í forrit_0B_02 nema aðferðunum bida og rekstAVegg hefur
// verið bætt við og allar aðgerðir á myndfleti gerðar synchronized.
//
public abstract class Kvikindi extends Thread
{
   public double xhnit, yhnit, stefna;
   public Color litur;
   public Color liturStriks;
   public boolean dregurStrik;

   protected Graphics myndflotur;
```

```
public Kvikindi(Graphics g)
{
   xhnit = 0;
   yhnit = 0;
   stefna = 0;
   myndflotur = g;
   liturStriks = new Color(0, 0, 0);
   litur = new Color(0, 0, 0);
   dregurStrik = false;
}


public void fram(double x)
{
   double ny_xhnit, ny_yhnit;
   synchronized (myndflotur)     // Kemur í veg fyrir að aðrir
   {                             // þræðir hafi áhrif á mynd-
                                 // flöt meðan skipanablokk
                                 // er framkvæmd.
      ny_xhnit = xhnit + x * Math.cos(stefna*Math.PI/180);
      ny_yhnit = yhnit - x * Math.sin(stefna*Math.PI/180);
      fela();
      if (dregurStrik)
      {
         myndflotur.setPaintMode();
         myndflotur.setColor(liturStriks);
         myndflotur.drawLine((int)xhnit, (int)yhnit,
                     (int)ny_xhnit, (int)ny_yhnit);
      }
      xhnit = ny_xhnit;
      yhnit = ny_yhnit;
      syna();
   }
}

public void haegri(double gradur)
{
   synchronized (myndflotur)
   {
      fela();
      stefna = (stefna - gradur) % 360;
      syna();
   }
}

public void vinstri(double gradur)
{
   synchronized (myndflotur)
   {
      fela();
      stefna = (stefna + gradur) % 360;
      syna();
   }
}

public abstract void syna();
public abstract void fela();
```

```
   void bida(long t)
   {
      try {this.sleep(t);}
      catch (InterruptedException e) {}
   }

   public boolean rekstAVegg()
   {
      return   (xhnit < myndflotur.getClipRect().x + 5) ||
         (yhnit < myndflotur.getClipRect().y + 5) ||
         (xhnit > myndflotur.getClipRect().width - 5) ||
         (yhnit > myndflotur.getClipRect().height - 5);
   }
} // Hér endar skilgreining á Kvikindi



import java.applet.Applet;
import java.awt.*;
//
// forrit_0B_03
// Dyralif2
//
public class Dyralif2 extends Applet
{
   Bjalla b;
   Padda p;
   Ormur o;
   Graphics g;

   public void init()
   {
      g = this.getGraphics();
      g.clipRect(0,0,this.size().width, this.size().height);

      b = new Bjalla(g);
      b.xhnit = 50;
      b.yhnit = 100;
      b.litur = new Color(128, 64, 0);
      b.dregurStrik = true;
      b.liturStriks = new Color(255, 255, 0);

      o = new Ormur(g);
      o.xhnit = 100;
      o.yhnit = 150;
      o.litur = new Color(128, 0, 64);
      o.dregurStrik = true;
      o.liturStriks = new Color(255, 0, 255);

      p = new Padda(g);
      p.xhnit = 150;
      p.yhnit = 100;
      p.litur = new Color(64, 128, 0);
      p.dregurStrik = true;
      p.liturStriks = new Color(0, 255, 255);
   }
```

```
   public void paint(Graphics g)
   {
      b.syna();
      o.syna();
      p.syna();
      b.start();
      o.start();
      p.start();
   }

   public void destroy()
   {
      b.stop();
      o.stop();
      p.stop();
   }
}  // Hér endar Dyralif2
```

Verkefni B.5
Breyttu klasanum `Dyralif2` í forrit_0B_03 þannig að þrjú kvikindi af hverri gerð þvælist um og dragi slóð. Notaðu fylki af kvikindum svona:

```
Kvikindi[] k;
k = new Kvikindi[9];
```

og settu bjöllur í þrjú fyrstu hólfin, orma í þrjú þau næstu og pöddur í þrjú þau síðustu. Áður skaltu keyra forrit_0B_03 óbreytt og átta þig á hvernig það er byggt.

Verkefni B.6
Búðu til þráð sem hermir eftir götuvita með því að sýna til skiptis rauðan, gulan og grænan hring. Láttu smið klasans taka við upplýsingum um hvað hvert ljós á að loga lengi og hvar götuvitinn á að vera staðsettur. Búðu svo til forrit sem setur 3 götuvita í gang.

B.x. Spurningar og umhugsunarefni

1. Í klasanum `Thread` eru aðferðir sem heita `start`, `stop`, `suspend` og `resume`. Til hvers eru þær?
2. Hvað er ruslatínsla og hvaða tilgangi þjónar hún?
3. Hvaða tegundir af myndum geta `Applet` sótt með aðferðinni `getImage`?
4. Hvaða aðferð þurfa undirklasar `Thread` að hafa?
5. Hvaða hlutverk hefur tegundin `Random`?
6. Hvers konar vandamál er hægt að leysa með því að nota skipunina `synchronized`?

Til umhugsunar

Með skipuninni `synchronized` getur einn þráður læst hlut fyrir öðrum þráðum meðan hann framkvæmir einhverjar aðferðir. Aðrir þræðir sem læsa sama hlut geta þá þurft að bíða eftir að lásinn sé tekinn af.

Hugsum okkur að tveir þræðir, $þ_1$ og $þ_2$, séu í gangi og þeir læsi hlutunum h_1 og h_2.

Ætli það geti gerst að $þ_1$ bíði með að taka lásinn af h_1 þar til h_2 losnar og $þ_2$ bíði með að taka lásinn af h_2 þar til h_1 losnar?

Hvernig þarf forrit að vera til að þessi staða komi upp?

Hvað ætli gerist ef þessi staða kemur upp?

Er hægt að tryggja að þessi staða komi ekki upp?

Í kafla 4.x var minnst á endalausar slaufur. Hvað ætli gerist ef það er endalaus slaufa inni í `synchronized`-blokk? Þá er lásinn aldrei tekinn af hlutnum sem `synchronized` skipunin læsir. Þetta getur valdið því að aðrir þræðir þurfi að bíða endalaust og verði því í reynd óvirkir.

Stundum er talað um að forrit „frjósi“. Ástæður slíkra uppákoma geta verið margvíslegar, allt frá alvarlegri vélarbilun til lítils háttar villu í forriti. Stundum virðast forrit líka stopp vegna þess að þræðir bíða endalaust eftir að lás sé tekinn af eða vegna þess að aðferð er föst í endalaustri slaufu.

Ætli eitthvað fleira en þetta tvennt geti valdið því að forrit virðist „frosið“?

C. kafli: Slembiföll og hermilíkön

C.a. Slembitalnagjafi

Í síðasta kafla var tegundin `Random` notuð til að velja tölur af handahófi. Í forritum þarf mjög oft að líkja eftir tilviljun. Þetta er t.d. gert þegar tölvur eru látnar draga í happadrætti eða líkja eftir einhverri atburðarás í veruleikanum sem er að hluta til handahófskennd. Nú er ekki um það að ræða að hegðun forrits sé í raun og veru tilviljanakennd. Það mætti þó hugsa sér að tölva væri tengd við tæki eins og vélina sem dregur í lóttóinu á laugardagskvöldum eða næman hitamæli. (Ef hitastig er mælt upp á lítið brot úr gráðu þá sveiflast síðasti aukastafurinn með tilviljanakenndum hætti.) Þetta er þó yfirleitt ekki gert þegar velja þarf tölur af handahófi heldur eru notuð föll sem haga sér nógu óreglulega til þess að engin regla virðist á útkomunum fremur en á útkomunum í lottóinu. Slík föll kallast slembiföll.

Slemiföll eru oft á forminu

$$r_{n+1} = (a \times r_n + c) \bmod m$$

þar sem $m>r_0$, $m>a$ og $m> c$. r_{n+1} er tala númer n+1 og hún er reiknuð út frá tölu númer n. Hugsum okkur til dæmis að tala númer 0 (fyrsta talan) sé 3, í stað a, c og m komi tölurnar 7, 11 og 13. Þá verður tala númer 1

$$(7 \times 3 + 11) \bmod 13 = 32 \bmod 13 = 6.$$

Tala númer 2 verður svo

$$(7 \times 6 + 11) \bmod 13 = 53 \bmod 13 = 1$$

og tala númer 3 verður

$$(7 \times 1 + 11) \bmod 13 = 18 \bmod 13 = 5.$$

Fyrstu 4 tölurnar sem þetta fall skilar eru semsagt 3, 6, 1, 5. Næst koma svo 7, 8, 2, 12, 4, 0, 11 og 10 og síðan endurtekur romsan sig aftur og aftur: 3, 6, 1, 5, 7, 8, 2, 12, 4, 0, 11, 10 …

Með því að velja heppilegar tölur fyrir a, m, c og r_0 er hægt að búa til talnaromsur sem virðast nær algerlega handahófskenndar. Í klasanum `Slembitalnagjafi` sem hér fer á eftir er a=10001, c=3 og m=17417. Klukka tölvunnar er látin sjá um að finna gildi á r_0. Smiður klasans les af klukku tölvunnar og gefur r_0 (sem er geymt í breytunni `x`) gildi sem er fjöldi millisekúndna frá upphafi árs 1970. Þar sem r_0 fær ekki sama gildi ef aðferðin er keyrð aftur og aftur verður til ný og ný talnaruna en ekki alltaf sú sama.

```
import java.util.Date;
//
// forrit_0C_01
// Slembitalnagjafi
//
public class Slembitalnagjafi
{
   long x;

   public Slembitalnagjafi()  // Smiðurinn gefur x
   {                          // upphafsgildi.
      Date d = new Date();    // getTime skilar fjölda millisekúndna
      x = d.getTime();        // sem liðnar eru frá byrjun árs 1970.
   }
```

```
   public double slembitala() // Skilar tölu milli 0 og 1.
   {
      x = (x*10001+3)%17417;
      return (double)x/17417;
   }
} // Hér endar Slembitalnagjafi
```

Hér fer á eftir klasi sem notar slembitalnagjafann til að velja nokkrar tölur af handahófi og teiknar súlurit sem sýnir stærð talnanna.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0C_01
// Handahof
//
//
public class Handahof extends Applet
{
   Slembitalnagjafi s;
   double d;

   public void init()
   {
      s = new Slembitalnagjafi();
   }

   void teiknaSulu(Graphics g, int xhnit, int haed)
   {
      g.setColor(new Color(255, 0, 0));
      g.fillRect(xhnit, 180-haed, 3, haed);
      g.setColor(new Color(0, 0, 0));
      g.drawRect(xhnit, 180-haed, 3, haed);
   }

   public void paint(Graphics g)
   {
      g.drawString("Slembitölur", 2, 25);
      g.drawLine(5, 180, 195, 180);
      g.drawLine(10, 185, 10, 50);
      g.drawLine(9, 80, 11, 80);
      g.drawString("1", 2, 82);
      for(int i=10; i<190; i+=3)
      {
         d = s.slembitala();
         teiknaSulu(g, i, (int)(100*d));
      }
   }
} // Hér endar Handahof
```

Verkefni C.1
Prófaðu að keyra forrit_0C_01 nokkrum sinnum og reyndu að meta hversu handahófskenndar útkomurnar eru.

Verkefni C.2
Búðu til forrit sem notar klasann `Slembitalnagjafi` til að velja 10.000 slembitölur og telur jafnóðum hve margar eru á milli 0 og 0,0999..., hve margar milli 0,1 og 0,1999... o.s.frv. og geymir niðurstöðurnar í fylki og teiknar súlurit yfir þær.

Verkefni C.3
Búðu til forrit sem teiknar hringi aftur og aftur endalaust og velur stað, radíus og lit af handahófi.

C.b. Hermilíkön

Hugsum okkur að tveir líffræðingar rannsaki bjöllutegund sem étur orma. Bjöllurnar eru duglegar að finna orma og annar líffræðingurinn heldur að þær hljóti að hafa nokkuð þroskuð skynfæri og rökstyður skoðun sína með því að segja að annars gætu bjöllurnar ekki fundið alla orma sem þær koma nálægt og gengið nánast beint að þeim og étið þá.

Hinn líffræðingurinn telur að skynfæri bjallanna séu mjög frumstæð. Hann bendir á máli sínu til stuðnings að miðtaugakerfi þeirra sé of einfalt til þess að um þroskað sjónskyn geti verið að ræða. Hins vegar virðist þær nema lykt og bregðast við henni með því að breyta um stefnu. Tilgáta hans er sú að bjöllurnar ráfi um nánast handahófskennt en beygi um 30 til 45 gráður til hægri þegar lykt af ormum dofnar en haldi nánast beint áfram ef lyktin dofnar ekki.

Hvernig ætli sé mögulegt að prófa þessa tilgátu og komast að því hvort það eitt að beygja af leið í hvert sinn sem lyktin af bráðinni dofnar dugar til að ná árangri við veiðar?

Ef til vill getur stærðfræðileg greining á viðfangsefninu leitt í ljós einhverjar aðferðir til að reikna það út. Sennilega er þó engin fljótleg aðferð til að reikna þetta út önnur en sú að herma skref fyrir skref eftir atburðarásinni þegar bjalla þvælist um meðal orma.

Ein leið væri ef til vill að smíða vélbjöllur og orma (eða leikfangabíla sem hægt er að nota í staðinn) sem aka um handahófskennt og láta „bjöllurnar“ taka beygju í hvert sinn sem merki (t.d. radíómerki) frá „ormum“ dofnar. Svo væri hægt að setja nokkur svona tæki af stað á gólfinu og gá hvort „bjöllurnar“ rekast fljótlega á „ormana“. Tilraun af þessu tagi væri vissulega framkvæmanleg en æði tímafrek og líklega nokkuð kostnaðarsöm. Mun einfaldara og fljótlegra er að smíða forrit sem hermir eftir þeirri hegðun sem tilgátan eignar bjöllunum.

Forrit sem hermir eftir hegðun eða atburðarás sem menn ætla að eigi sér stað í veruleikanum kallast hermilíkan. Í vísindum nútímans gegna hermilíkön mikilvægu hlutverki við að prófa tilgátur og rannsaka hvað af þeim leiðir. Stundum koma þau í staðinn fyrir dýrar tilraunir eða flókna útreikninga. Stundum verður tilraunum eða útreikningum illa eða ekki við komið og þá kunna hermilíkön að vera eina leiðin til að prófa tilgátur og kenningar.

Forritið sem hér fer á eftir (forrit_0C_02) á að kanna tilgátu líffræðingsins um veiðiaðferð bjöllunnar. Það notar klasana úr forrit_0B_03 óbreytta og slembitalnagjafann úr forrit_0C_01. Við þetta er bætt tveim klösum, rándýrinu

`BjallaSemEturOrma` sem er undirklasi `Bjalla` úr forrit_0B_03 og `Dyralif3` sem er `Applet`.

`init`-aðferðin í `Dyralif3` býr til fylki af tegundinni `Kvikindi` og setur orma í sæti númer 1 til 7 en `BjallaSemEturOrma` (þ.e. björllu sem étur orma) í sæti númer 0. `paint`-aðferðin setur svo alla þræðina (þ.e. öll kvikindin) í gang.

Ormarnir eru alveg eins og í forrit_0B_03 og þeir gera ekkert annað en að skríða um. `BjallaSemEturOrma` hefur hins vegar ýmislegt til viðbótar við `Bjalla` úr forrit_0B_03. Skýringar á þessum viðbótum eru skrifaðar inn í kóðann sem athugasemdir.

```
import java.awt.*;
//
// forrit_0C_02
// BjallaSemEturOrma
//
// Finnur orma með því að taka  beygju um 30 til 45 gráður til hægri
// í hvert sinn sem lykt af Ormum dofnar.
//
public class BjallaSemEturOrma extends Bjalla
{
   private int hradi;
   private double styrkurLyktar;
   Slembitalnagjafi s;
   Kvikindi[] dyrASvaedinu;   // Fylki sem inniheldur
                              // öll kvikindi á svæðinu.

   // Smiðurinn tekur við fylki af öllum dýrum á svæðinu.
   public BjallaSemEturOrma(Graphics g, Kvikindi[] k)
   {
      super(g);  // Kallað á smið yfirtegundarinnar Ormur.
      double f2;
      s = new Slembitalnagjafi();
      hradi = 3;
      dyrASvaedinu = k;
      styrkurLyktar = 0;

      // Farið gegnum fylkið af Kvikindum á svæðinu og reiknað hve
      // sterka lykt af ormum bjallan finnur.
      for (int i=0; i<dyrASvaedinu.length; i++)
      {
         // Ef dýr númer i er af tegundinni ormur
         // þá er fjarlægð þess í 2. veldi fundin
         // enda er styrkur lyktar í öfugu hlutfalli
         // við fjarlægð í 2. veldi.
         if (dyrASvaedinu[i] instanceof Ormur)
         {
            f2 = fjarlaegdI2Veldi(dyrASvaedinu[i]);
            if (f2 < 10000)      // Lykt finnst ekki
            {                    // ef fjarl. > 100.
               styrkurLyktar += 1/f2;
            }
         }
      }
   }
```

```
// Run aðferðin lætur bjöllu sem étur orma valsa um
// og beygja í hvert sinn sem lykt af ormum dofnar.
public void run()
{
   double skreflengd;
   int nrOrms;
   while (true)
   {
      skreflengd = 2*s.slembitala();
      fram(skreflengd);
      if (rekstAVegg())
      {
         fram(-skreflengd);
         vinstri(180-360*s.slembitala());
      }
      // Rásar tilviljanakennt um 0 til 15
      // gráður í hverju skrefi.
      vinstri(30*(0.5-s.slembitala()));

      // Beygir markvisst um 30 til 40 gráður ef
      // lykt af ormum dofnar.
      if (lyktAfOrmumDofnar())
      {
         haegri(30+15*s.slembitala());
      }
      // Ef bjalla hefur rekist á orm þá skilar ormurFundinn tölu
      // sem segir hvar í fylkinu dyrASvaedinu ormurinn er. Ef
      // bjallan hefur ekki rekist á orm er útkoman úr
      // ormurFundinn -1.
      nrOrms = ormurFundinn();
      if (nrOrms != -1) { etaOrm(nrOrms); }
      bida(100/hradi);
   }
} // run-aðferðin endar

// Aðferðin etaOrm lætur orm einfaldlega hverfa, stöðvar þráðinn
// og setur gildið null (sem merkir ekkert) í sæti hans í fylkinu.
private void etaOrm(int nr)
{
   synchronized (myndflotur)
   {
      dyrASvaedinu[nr].fela();
      dyrASvaedinu[nr].stop();
      dyrASvaedinu[nr] = null;
   }
   styrkurLyktar = 0;
}

// Ef bjalla hefur rekist á orm þá skilar ormurFundinn tölu sem
// segir hvar í fylkinu dyrASvaedinu ormurinn er. Ef bjallan hefur
// ekki rekist á orm er útkoman úr ormurFundinn -1. Bjalla telst
// hafa rekist á orm ef fjarlægð hennar frá orminum í 2. veldi er
// minna en 50, þ.e. ef fjarlægðin er 7 punktar eða minna.
private int ormurFundinn()
{
   int nr = -1;
   for (int i=0; i<dyrASvaedinu.length; i++)
   {
      if (dyrASvaedinu[i] instanceof Ormur)
      {
         if (50>fjarlaegdI2Veldi(dyrASvaedinu[i]))
         nr = i;
      }
   }
   return nr;
}
```

```
   // Reiknar út fjarl. í 2. veldi milli björnunnar og
   // kvikindisins k.
   private double fjarlaegdI2Veldi(Kvikindi k)
   {
      return (xhnit-k.xhnit)*(xhnit-k.xhnit) +
             (yhnit-k.yhnit)*(yhnit-k.yhnit);
   }

   // Reiknar styrk lyktar og setur útkomuna í breytuna styrkurLyktar
   // og skilar true ef hann er minni en gildið sem fyrir var í
   // breytunni.
   private boolean lyktAfOrmumDofnar()
   {
      double f2;
      double fyrriStyrkur = styrkurLyktar;
      styrkurLyktar = 0;
      for (int i=0; i<dyrASvaedinu.length; i++)
      {
         if (dyrASvaedinu[i] instanceof Ormur)
         {
            f2 = fjarlaegdI2Veldi(dyrASvaedinu[i]);
            if (f2 < 1000)
            {
               styrkurLyktar += 1/f2;
            }
         }
      }
      return styrkurLyktar < fyrriStyrkur;
   }
} // Hér endar klasinn BjallaSemEturOrma.
```

Taktu eftir því hvernig fastinn `null` er notaður í aðferðinni `etaOrm`. Þegar breyta af tegund sem vísar á hlut (þ.e. breyta sem er ekki af einfaldri tegund) er fyrst búin til inniheldur hún gildið null sem þýðir einfaldlega að hún vísi ekki á neitt. Sé breytu sem vísar á hlut gefið gildið `null` hættir hún að vísa á hlutinn og vísi engar aðrar breytur á hlutinn þá er honum einfaldlega eytt því Bytecode-túlkurinn annast sjálfvirka ruslatínslu.

Taktu líka eftir því hvernig `instanceof` er notað t.d. í aðferðinni `lyktAfOrmumDofnar`. Þessi aðgerð er notuð til að finna hvort hlutur er af tiltekinni tegund.

```
x instanceof Padda
```

skilar gildinu `true` ef hluturinn sem `x` vísar á er af tegundinni `Padda`. Breytan `x` þarf samt ekki að vera skilgreind sem `Padda`. Hún getur verið skilgreind sem yfirtegund `Padda`, þ.e. sem `Kvikindi` eða `Object`.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0C_02
//
public class Dyralif3 extends Applet
{
   Kvikindi[] dyr;
   Graphics g;
   Slembitalnagjafi s;
   boolean ekki_byrjad = true;

   public void init()
   {
      g = this.getGraphics();
      g.clipRect(0,0,this.size().width, this.size().height);
      s = new Slembitalnagjafi();
      dyr = new Kvikindi[8];

      for (int i=1; i<dyr.length; i++)
      {
         dyr[i] = new Ormur(g);
         dyr[i].xhnit = 200*s.slembitala();
         dyr[i].yhnit = 200*s.slembitala();
         dyr[i].stefna = 360*s.slembitala();
         dyr[i].litur = new Color(128, 0, 64);
         dyr[i].dregurStrik = false;
      }

      dyr[0] = new BjallaSemEturOrma(g, dyr);
      dyr[0].xhnit = 200*s.slembitala();
      dyr[0].yhnit = 200*s.slembitala();
      dyr[0].litur = new Color(128, 64, 0);
      dyr[0].dregurStrik = false;
   }

   public void paint(Graphics g)
   {
      if (ekki_byrjad)
      {
         for (int i=0; i<dyr.length; i++)
         {
            dyr[i].start();
            dyr[i].syna();
         }
         ekki_byrjad = false;
      }
   }

   public void destroy()
   {
      for (int i=0; i<dyr.length; i++)
      {
         dyr[i].stop();
      }
   }
}
```

Verkefni C.4
Keyrðu forrit_0C_02 og reyndu að meta hvort það staðfestir eða hrekur þá kenningu að hægt sé að veiða orma með því að skynja aðeins hvort lykt minnkar eða ekki og beygja til hægri um 30 til 45 gráður ef hún minnkar.

Verkefni C.5
Breyttu forrit_0C_02 þannig að hægt sé að nota það til að kanna hvort bjalla geti veitt orma með því einu að valsa um handahófskennt og auka hraðann þegar lyktin dofnar en minnka hann þegar hún eykst.

Verkefni C.6 (erfitt)
Búðu til forrit sem kannar hvort bjalla geti veitt orma með því einu að valsa um handahófskennt og beygja til hægri þegar lyktin af þeim dofnar ef ormarnir flýja með því að auka hraða sinn þegar bjalla nálgast og hægja aftur á sér þegar hún fjarlægist. (Ef ormur eykur hraðann þegar bjalla nálgast þá gæti hann að vísu hlaupið beint í fangið á henni ef hann er svo óheppinn að stefna í átt að henni en líkurnar á því hljóta að vera heldur litlar.) Hvað þurfa ormarnir að auka hraðann mikið til að veiðiaðferð bjöllunnar hætti að bera árangur?

C.c. Endurnýting og undirklasar

Það hefði mátt hugsa sér að smíða bjöllu sem veiðir orma með því að breyta klasanum `Bjalla`. Sú aðferð hefði þó verið óheppileg. Klasinn `Bjalla` stendur fyrir sínu eins og hann er og með því að búa til ólíka undirklasa er hægt að búa til bjöllutegund sem veiðir orma og svo aðra sem étur gulrætur. Klasinn `Kvikindi` hefur til að bera það sem er sameiginlegt öllum kvikindum og `Bjalla` það sem er sameiginlegt öllum bjöllum. Eigi að búa til sérhæfða bjöllutegund er það gert með því að smíða undirklasa en ekki með því að breyta klasanum `Bjalla`.

Í Java og öðrum hlutbundnum forritunarmálum er auðvelt að nota tegundir sem er búið að búa til og gæða hlutina nýjum hæfileikum með því að smíða undirtegundir. Sé hin leiðin valin að breyta klösum sem búið er að fullvinna er hætt við að tegundir sem búið er að þrautprófa og losa við allar villur séu skemmdar, breytingarnar innihaldi villur eða galla af einhverju tagi. Þá er verið að skemma það sem búið er að leggja vinnu í að fullkomna.

Einn helsti kostur hlutbundinna forritunarmála er hve auðvelt er að endurnýta kóða. Klasi sem var búinn til þegar unnið var að einu forriti í fyrra getur nýst óbreyttur við gerð allt öðru vísi forrits á næsta ári.

C.x. Spurningar og umhugsunarefni

1. Hverjar eru 3 fyrstu útkomurnar úr fallinu $r_{n+1} = (a \times r_n + c) \bmod m$ ef $r_0=5$, $a=17$, $c=13$ og $m=11$?
2. Hvað er hermilíkan?
3. Hvaða gildi fá breyturnar `a`, `b`, `c`, `d`, `e` og `f` ef eftirfarandi skipanir eru gefnar? (Gerðu er ráð fyrir að `Bjalla` sé undirklasi `Kvikindi`.)

```
boolean a, b, c, d;
Kvikindi n, o;
n = new Bjalla(g);
o = new Object();
a = (n instanceof Bjalla);
b = (n instanceof Kvikindi);
c = (n instanceof Object);
d = (o instanceof Bjalla);
e = (o instanceof Kvikindi);
f = (o instanceof Object);
```

Til umhugsunar

Í þessum kafla hefur verið fjallað lítillega um hermilíkön. Slík forrit gegna mikilvægu hlutverki í mörgum vísindagreinum þar sem þau eru notuð til að prófa tilgátur og spá fyrir um framvindu af einhverju tagi.

Hermilíkan sem byggir á réttri kenningu um hegðun einhvers fyrirbæris getur í mörgum tilvikum reiknað út hvernig það muni haga sér. Til dæmis eru hermilíkön af veðurkerfum notuð til að búa til veðurspár. Að vísu er veðrið of flókið til að hægt sé að forrita tölvur til að herma eftir hegðun þess af nákvæmni. Þess vegna er engin leið að búa til nákvæmar veðurspár langt fram í tímann. Bestu líkön af veðrinu sem smíðuð hafa verið duga aðeins til að spá með sæmilegum líkum hver þróun þess verður nokkra daga fram í tímann.

Til að hægt sé að búa til hermilíkan af einhverri atburðarás og nota það til að spá fyrir um framhald hennar þarf hún að fylgja einhverri reglu sem er þekkt og hægt er að láta tölvu herma eftir.

Hugsum okkur nú að við höfum fyrir framan okkur tölvu sem er að tefla skák. Allt sem gerist í rafrásum tölvunnar fylgir þekktum lögmálum sem hægt er að forrita aðra tölvu til að herma eftir. Ætli þetta þýði að hermilíkan af skáktölvunni sem keyrt er á annarri tölvu geti spáð fyrir um hvaða leik hún muni leika næst?

Ef tölvan sem keyrir hermilíkanið er miklu hraðvirkari en tölvan sem teflir getur hún kannski spáð fyrir um næsta leik áður en honum er leikið. En hvað ef vélarnar eru jafn hraðvirkar?

Fyrir um það bil 200 árum setti franski stærðfræðingurinn og heimspekingurinn Pierre Simon de Laplace fram þá kenningu að ef ástand hverrar einustu efnisagnar í heiminum væri þekkt þá væri hægt að reikna út hvernig veröldin muni verða eftir ár og jafnvel aldir.

Það er ekki hægt að reikna út hvað tölva gerir, og fá svar áður en það sem segja skal fyrir um er liðið, án þess að nota til þess hraðvirkari tölvu. Ætli mögulegt sé að búa til nógu hraðvirka tölvu, forrita hana og mata á nægum upplýsingum til að hún geti

reiknað út framvindu flóknari kerfa, eins og til dæmis veðurs á jörðinni, og lokið þeim útreikningum áður en veðrið er komið og farið?

Ef engin takmörk eru fyrir því hvað hægt er að smíða afkastamikla tölvu og mæla ástand lofthjúpsins af mikilli nákvæmni þá má ætla að svarið sé jákvætt.

En ætli það séu ekki einhver takmörk fyrir því hvað hægt er að smíða afkastamiklar tölvur?

Og ætli að yrði nokkuð veður ef menn settu upp nóg af mælitækjum til að mæla ástand hverrar einustu sameindar í lofthjúpi jarðar?

D. kafli: Villur og frávik

D.a. `Throwable`, `Error` og `Exception`

Einn af klösunum sem fylgir með Java í pakkanum `java.lang` heitir `Throwable`. Í hvert sinn sem Javaforrit lendir í vandræðum eða einhvers konar villa verður í keyrslu þess býr það til hlut af einhverri undirtegund `Throwable` og „kastar“ honum. Eðlilegast er að líta á þessa hluti sem eins konar neyðarskeyti sem forrit sendir frá sér þegar það lendir í vandræðum.

`Throwable` hefur tvo undirklasa. Þeir heita `Error` og `Exception`. Hvor þeirra hefur svo marga aðra undirklasa. Vandræði sem upp koma við keyrslu forrits skiptast semsagt í tvo megin flokka: Villur (`Error`) og frávik (`Exception`). Villur greinast í nokkra undirflokka. Frávik eru af tveim megingerðum: Annars vegar er klasinn `RuntimeException` og undirklasar hans. Hins vegar allir aðrir undirklasar `Exception`.

Hlutur af tegundinni `Error` er búinn til ef það er beinlínis villa í forritinu, ef til dæmis er reynt að beita aðferð sem er ekki til eða geyma gögn í breytu af vitlausri tegund. Java-þýðandinn á að tryggja að forrit séu villulaus og yfirleitt neitar hann að þýða forrit sem innihalda villur. En Java-þýðendur eru ekki fullkomnir frekar en önnur mannanna verk svo stundum fara forrit í gang þótt þau innihaldi villur.

Frávik (`Exception`) er búið til ef vandræði koma upp í keyrslu forrits án þess að það sé endilega villa í forritinu. Þetta gerist til dæmis ef reynt er að sækja úr hólfi númer 10 í fylki sem hefur aðeins 5 hólf, breyta strengnum "1oo1" í tölu (það á að nota tölustafinn núll en ekki sérhljóðann o), deila í heiltölu með núlli eða lesa skrá af disklingi þegar ekki er neinn disklingur í drifinu. Allt þetta getur gerst við keyrslu forrits þó forritið sé rétt samið og villulaust.

D.b. Að grípa frávik

Fjölmargar aðferðir sem fylgja með Java eru þannig byggðar að þær kasta fráviki (þ.e. skeyti af gerðinni `Exception`) ef þeim tekst ekki að vinna það verk sem þeim er ætlað. Hér má nefna sem dæmi smiðina

```
public Double(String  s) throws NumberFormatException
```

og

```
public Integer(String s) throws NumberFormatException
```

sem báðir taka við streng og smíða úr honum tölu.

Með skipununum

```
String s = new String("26");
Integer I = new Integer(s);
int i = I.intValue();
```

er strengnum "26" breytt í `int` gildið (þ.e. heiltöluna) 26. En ef gefnar eru skipanirnar

```
String s = new String("XXVI");
Integer I = new Integer(s);
int i = I.intValue();
```

þá getur smiðurinn `Integer` ekki breytt strengnum í heiltölu því hann kann ekkert á rómverskar tölur svo hann kastar frá sér skeyti af tegundinni `NumberFormatException` sem er undirklasi `RuntimeException`.

Forrit getur brugðist við skeytasendingu með því að grípa skeytið. Þetta er gert með því að setja aðferðir sem gætu sent frá sér skeyti inn í `try`-blokk og hafa `catch`-blokk þar fyrir neðan svona:

```
try
{
   // Hér koma skipanir sem gætu kastað skeyti.
}
catch (NumberFormatException n)
{
   // Hér kemur það sem á að gera ef fráviki af tegundinni
   // NumberFormatException er kastað úr try-blokkinni.
}
```

Ef einhver skipun í `try`-blokkinni kastar skeyti er stokkið niður fyrir hana og skipunum þar fyrir neðan sleppt. Ef `catch`-blokkinn grípur skeyti af þeirri gerð sem kastað var þá eru skipanirnar í henni framkvæmdar.

Stundum eru margar skipanir inni í `try`-blokk og stundum er möguleiki að þær kasti skeytum af ýmsum gerðum. Þá er hægt að hafa margar `catch`-blokkir, eina fyrir hverja gerð, svona:

```
try
{
   // Hér koma skipanir sem gætu kastað skeyti.
}
catch (FravikAfTegundA n)
{
   // Hér kemur það sem á að gera ef fráviki af tegundinni
   // FravikAfTegundA er kastað úr try-blokkinni.
}
catch (FravikAfTegundB n)
{
   // Hér kemur það sem á að gera ef fráviki af tegundinni
   // FravikAfTegundB er kastað úr try-blokkinni.
}
catch (FravikAfTegundC n)
{
   // Hér kemur það sem á að gera ef fráviki af tegundinni
   // FravikAfTegundC er kastað úr try-blokkinni.
}
```

Oft er neðsta `catch`-blokkin látin grípa öll möguleg frávik og stundum er þar fyrir neðan sett `finally`-blokk, en skipanirnar í henni eru framkvæmdar síðast á hverju sem gengur. Ef frávik verða í `try`-blokkinni er þá stokkið í viðeigandi `catch`-blokk og síðan í `finally`-blokkina. Verði engin frávik er `try`-blokkin kláruð og svo stokkið yfir `catch`-blokkirnar og skipanirnar í `finally`-blokkinni framkvæmdar.

```
   try
   {
      // Hér koma skipanir sem gætu kastað skeyti.
   }
   catch (FravikAfTegundA n)
   {
      // Hér kemur það sem á að gera ef fráviki af tegundinni
      // FravikAfTegundA er kastað úr try-blokkinni.
   }
   catch (FravikAfTegundB n)
   {
      // Hér kemur það sem á að gera ef fráviki af tegundinni
      // FravikAfTegundB er kastað úr try-blokkinni.
   }
   catch (FravikAfTegundC n)
   {
      // Hér kemur það sem á að gera ef fráviki af tegundinni
      // FravikAfTegundC er kastað úr try-blokkinni.
   }
   catch (Exception n)
   {
      // Hér kemur það sem á að gera ef fráviki af einhverri
      // annarri gerð en FravikAfTegundA, FravikAfTegundB eða
      // FravikAfTegundC er kastað úr try-blokkinni.
   }
   finally
   {
      // Hér koma skipanir sem verður undir öllum kringumstæðum
      // að framkvæma síðast.
   }
```

Hér fer á eftir forrit sem er svipað forrit_02_01. Það tekur við tveim heilum tölum og reiknar kvóta og leif. Við keyrslu þessa forrits geta að minnsta kosti komið upp tvenns konar frávik. Það getur gerst að smiðurinn `Integer` kasti `NumberFormatException` ef ekki eru skrifaðar heilar tölur í textareitina. Það getur líka gerst að aðgerðirnar `/` og `%` kasti `ArithmeticException` ef talan í seinni reitnum (sem deilt er með) er núll.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0D_01
// Deiling1
//
public class Deiling1 extends Applet
{
   TextField t1, t2;
   TextArea tsvar;
   Button bdeila;
   Label a1, a2;
   Integer I1, I2;
   int i1, i2, ikvoti, ileif;

   public void init()
   {
      t1 = new TextField(3);
      t2 = new TextField(3);
      tsvar = new TextArea(5, 25);
      bdeila = new Button("Deila");
      a1 = new Label("Tala 1");
      a2 = new Label("Tala 2");

      this.add(a1); this.add(t1);
      this.add(a2); this.add(t2);
      this.add(bdeila); this.add(tsvar);
```

```
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bdeila)
      {
         I1 = new Integer(t1.getText());
         i1 = I1.intValue();
         I2 = new Integer(t2.getText());
         i2 = I2.intValue();
         ikvoti = i1 / i2;
         ileif = i1 % i2;
         tsvar.setText("Kvóti: " + ikvoti + " Leif: " + ileif);
         return true;
      }
      return false;
   }
}
```

Verkefni D.1 *
Prófaðu að keyra klasann `Deiling1` og láta keyrsluna klikka með því að slá inn annað en heilar tölur eða einhverja heila tölu í fyrri reitinn og núll í þann seinni.

Ef þú keyrir forritið í vefsjá (en ekki *Appletviewer*) þá sérð þú e.t.v. engin skilaboð um að frávikum hafi verið kastað. Þá skaltu prófa að keyra `Deiling1` sem sjálfstætt forrit með því að nota klasann `Starta` í forrit_0D_01.

Um aðferðina `main` í klasanum `Starta` var fjallað í 9. kafla. Klasinn er skilgreindur svona:

```
import java.awt.*;
//
// forrit_0D_01
// Starta
//
public class Starta           // Ræsir hlut af tegundinni Deiling1
{                             // sem sjálfstætt forrit.
   public static void main(String args[])
   {
      Deiling1 a = new Deiling1();
      a.init();
      a.start();
      Frame f = new Frame();
      f.add("Center", a);
      f.resize(200, 200);
      f.move(100, 100);
      f.show();
   }
}
```

Skoðaðu skilaboðin sem túlkurinn skrifar þegar reynt er að deila með núlli eða breyta streng sem inniheldur t.d. bókstafi í tölu.

Í eftirfarandi klasa eru skipanirnar sem gætu klikkað komnar inn í `try`-blokk og þar fyrir neðan eru komnar `catch`-blokkir sem grípa skeyti frá smiðnum `Integer` og aðgerðunum `/` og `%`. Fyrir utan `action`-aðferðina er klasinn eins og `Deiling1`.

```
import java.applet.Applet;
import java.awt.*;
//
```

```
// forrit_0D_01
// Deiling2
//
//
public class Deiling2 extends Applet
{
   TextField t1, t2;
   TextArea tsvar;
   Button bdeila;
   Label a1, a2;
   Integer I1, I2;
   int i1, i2, ikvoti, ileif;

   public void init()
   {
      t1 = new TextField(3);
      t2 = new TextField(3);
      tsvar = new TextArea(5, 25);
      bdeila = new Button("Deila");
      a1 = new Label("Tala 1");
      a2 = new Label("Tala 2");
      this.add(a1); this.add(t1);
      this.add(a2); this.add(t2);
      this.add(bdeila); this.add(tsvar);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bdeila)
      {
         tsvar.setText("");  // Textasvæði tæmt.

         // Allt sem gæti klikkað (og meira til) sett inni í
         // try-blokk.
         try
         {
            I1 = new Integer(t1.getText());
            i1 = I1.intValue();
            I2 = new Integer(t2.getText());
            i2 = I2.intValue();

            ikvoti = i1 / i2;
            ileif = i1 % i2;
            tsvar.appendText("Kvóti: " + ikvoti + " Leif: " + ileif);
         }
         catch (NumberFormatException fravik)
         {
            tsvar.appendText
                  ("Þú átt að slá inn heilar tölur" + "\n");
         }
         catch (ArithmeticException fravik)
         {
            tsvar.appendText("Get ekki reiknað þetta" + "\n");
         }
         catch (Exception fravik)
         {
            tsvar.appendText("Hver þremillinn er á seyði" + "\n");
         }
         finally
         {
            tsvar.appendText
                  ("\n" + "Nú máttu reikna nýtt dæmi" + "\n");
         }

         return true;
      }
```

```
        return false;
    }
}
```

Verkefni D.2 *
Prófaðu að keyra klasann `Deiling2` og láta hann grípa frávik með því að slá inn annað en heilar tölur eða einhverja heila tölu í fyrri reitinn og núll í þann seinni.

Verkefni D.3 *
Breyttu forrit_06_02, sem reiknar meðaltal af mörgum tölum, þannig að það bregðist skynsamlega við ef slegið er inn annað en tölur eða slegnar eru inn of margar tölur. (Ath. ef reynt er að setja fleiri stök í fylki en rúm er fyrir er kastað fráviki af tegundinni `ArrayIndexOutOfBoundsException`.)

D.c. `RuntimeException` og önnur frávik

Ef þú hefur leyst verkefni D.1 og D.2 hefur þú væntanlega séð að ef forrit grípur ekki frávik sem kastað er þá gerir túlkurinn sem keyrir þau athugasemdir. Segja má að skeytið endi hjá honum. Grípi forrit hins vegar frávik þá fer það ekki lengra og túlkurinn gerir engar athugasemdir. Séu frávik ekki gripin heldur látin enda hjá túlkinum getur keyrsla forrits stöðvast.

Það er að öllum jafnaði skynsamlegt að setja skipanir sem geta valdið frávikum innan í `try`-blokk og hafa `catch`-blokkir þar fyrir neðan sem grípa öll frávik sem kastað er. Þegar um önnur frávik en `RuntimeException` og undirklasa þess klasa er að ræða er raunar óhjákvæmilegt að nota `try-catch`. Ef skipun sem getur valdið öðrum frávikum en `RuntimeException` er ekki sett innan í `try`-blokk og höfð `catch`-blokk á eftir sem grípur frávikin þá telst það villa og Java-þýðandinn neitar að þýða forritið. Frávik af gerðinni `RuntimeException` eru sérstök að því leyti að forrit geta sleppt því að grípa þau og látið þau gossa alla leið til túlksins.

Í forrit_0B_01 var aðferðin `bida` skilgreind svona:

```
void bida(long t)
{
   try {this.sleep(t);}
   catch (InterruptedException e) {}
}
```

Þessi aðferð notar `sleep`-aðferðina í klasanum `Thread`. Titillína hennar er svona:

```
public static void sleep(long millis) throws InterruptedException
```

og klasinn `InterruptedException` er undriklasi `Exception`, ekki `RuntimeException`. Það er því ekki hægt að nota aðferðina `sleep` nema setja hana inn í `try`-blokk með `catch`-blokk á eftir. En hér er ekkert gert til að bregðast við frávikum sem `sleep` kastar því `catch`-blokkin er tóm. Þetta þýðir að ef `sleep` tekst ekki að láta þráð blunda í stundarkorn þá er einfaldlega stokkið út úr `try`-blokkinni inn í `catch`-blokkina. Þar sem hún er tóm er ekkert gert í málinu heldur bara lokið við aðferðina `bida`. Stundum er fullkomlega eðlilegt að bregðast við fráviki á þennan hátt, þ.e. með því að gera ekki neitt. Í forrit_0B_01 gerir lítið til þó það mistakist við og við að láta þráð blunda (eða stöðvast tímabundið). Þetta verður til þess eins að hann fer stundum hraðar en til var ætlast.

D.d. Frávik skilgreind, `throw` og `throws`

Hægt er að skilgreina nýja undirklasa `Exception` og `RuntimeException`. forrit_0D_02 leggur saman tvö almenn brot. Það er myndað úr þrem klösum. Þeir eru `Brot` og `Brotareikningur`, sem eru að mestu byggðir á samnefndum klösum í forrit_05_01, og frávikið `NefnariNullException`, sem smiðir klasans `Brot` eiga að kasta ef reynt er að búa til brot með núlli í nefnara.

Eigi aðferð að kasta fráviki verður að taka það fram í titillínu hennar. Í klasanum `Brot` er þetta gert í titillínum smiðanna með því að setja `throws` fyrir aftan svigana og nefna svo hvers konar fráviki sé kastað, svona:

```
public Brot(int t, int n) throws NefnariNullException
```

og svona

```
public Brot(String b) throws NefnariNullException
```

Hægt er að nota fráviksklasa sem fylgja með Java eða heimatilbúna klasa sem erfa frá `Exception` eða undirklösum hans. Hér er það síðarnefnda gert og búin til sérstakur fráviksklasi. Skilgreiningin á henni er afskaplega einföld. Hún er svona:

```
import java.lang.Exception;
//
// forrit_0D_02
// NefnariNullException
//
public class NefnariNullException extends RuntimeException
{
}
```

Klasinn er tómur, inniheldur hvorki aðferðir né klasabreytur enda eiga hlutir af tegundinni `NefnariNullException` aldrei að gera neitt nema láta kasta sér í klærnar á `catch`-skipunum. Til hvers er að kunna aðferðir ef maður á aldrei að gera neitt?

Smiðirnir í `Brot` eru svo látnir kasta fráviki ef nefnari er núll með skipuninni

```
if (nefnari == 0)
{
   NefnariNullException fravik = new NefnariNullException();
   throw fravik;
}
```

Ef nefnari er núll er semsagt búinn til nýr hlutur af tegundinni `NefnariNullException` og honum kastað með skipuninni `throw`. Hægt er að gera það sama í ögn styttra máli með

```
if (nefnari == 0)
{
   throw new NefnariNullException();
}
```

Hér kemur svo klasinn `Brot`. Allt nema smiðir er eins og í samnefndum klasa í forrit_05_01.

```
// forrit_0D_02
// Brot
//
// Að mestu byggt á Brot í forrit_05_01
//
public class Brot
{
   public int teljari, nefnari;

   public Brot(int t, int n) throws NefnariNullException
   {
      teljari = t;
      nefnari = n;
      // Ef nefnari er 0 þá er búinn til nýr hlutur af tegundinni
      // NefnariNullException og honum kastað.
      if (nefnari == 0)
      {
         NefnariNullException fravik = new NefnariNullException();
         throw fravik;
      }
   }

   public Brot(String b) throws NefnariNullException
   {
      int i, j;
      String sT, sN;
      Integer T, N;

      i = b.indexOf("/");
      j = b.length();

      sT = new String();
      sT = b.substring(0, i);
      T = new Integer(sT);
      teljari = T.intValue();

      sN = new String();
      sN = b.substring(i+1, j);
      N = new Integer(sN);
      nefnari = N.intValue();

      // Ef nefnari er 0 þá er búinn til nýr hlutur af tegundinni
      // NefnariNullException og honum kastað.
      if (nefnari == 0)
      {
         NefnariNullException fravik = new NefnariNullException();
         throw fravik;
      }
   }

   // Hér fara á eftir þrjár aðferðir sem
   // Brot eiga að kunna. Sú fyrsta gerir broti
```

```
   // kleift að stytta sig, sú önnur gerir því
   // mögulegt að bæta sér við annað brot.
   // Þriðja aðferðin breytir broti í streng.

   public void stytta()
   {
      int t, n, afg;
      t = teljari;
      n = nefnari;

      // Aðf. Evklíðs notuð til að finna stærsta
      // sameiginlegan þátt talnanna t og n.

      while (n > 0)
      {
         afg = t % n;
         t = n;
         n = afg;
      }

      // nú er t stærsta tala sem gengur bæði
      // upp í teljara og nefnara
      // og við styttum brotið.

      teljari = teljari / t;
      nefnari = nefnari / t;
   }

   public Brot plus(Brot b)
   {
      int t, n;
      Brot utkoma;
      t = teljari*b.nefnari + b.teljari*nefnari;
      n = nefnari*b.nefnari;
      utkoma = new Brot(t, n);
      utkoma.stytta();
      return utkoma;
   }

   public String toString()
   {
      return teljari + "/" + nefnari;
   }
}  // Hér endar skilgreining á tegundinni Brot.
```

Taktu eftir því að inni í aðferðinni `plus` er smiðurinn `Brot` notaður án þess að hann sé hafður inni í `try`-blokk. Þetta er leyfilegt því frávikið sem hann kastar er undirklasi `RuntimeException`. Væri `NefnariNullException` skilgreindur svona

```
   public class NefnariNullException extends Exception
   {
   }
```

þá yrði að hafa `plus`-aðferðina í `Brot` eitthvað á þessa leið:

```
   public Brot plus(Brot b)
   {
      int t, n;
      Brot utkoma;
      t = teljari*b.nefnari + b.teljari*nefnari;
      n = nefnari*b.nefnari;
      try
      {
         utkoma = new Brot(t, n);
         utkoma.stytta();
         return utkoma;
      }
      catch (Exception fravik)
      {
         // Ef brotinu tekst ekki að bæta b við sig þá
         // skilar það sjálfu sér.
         return this;
      }
   }
```

Hér er svo klasinn `Brotareikningur` sem notar `Brot` til að leggja saman tvö almenn `Brot`. Hann er eins og samnefndur klasi í forrit_05_01 nema hvað búið er að setja `try`- og `catch` blokkir í `action`-aðferðina.

Þar sem skipanirnar í `try`-blokkinni geta kastað fleiri gerðum frávika en `NefnariNullException` eru hafðar þrjár `catch` blokkir. Ein fyrir `NefnariNullException`, ein fyrir annað líklegt frávik, sem er `NumberFormatException`, og að síðustu ein sem grípur öll frávik sem ekki hafa þegar verið gripin og er sett til vonar og vara ef eitthvað óvænt skyldi gerast.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0D_02
// Brotareikningur
//
// Að mestu byggt á Brotareikningur í forrit_05_01
//
public class Brotareikningur extends Applet
{
   TextField tBrot1, tBrot2, tSumma;
   Button bLeggjaSaman;
   Label lBrot1, lBrot2, lSumma;
   Brot brot1, brot2, summa;
   public void init()
   {
      lBrot1 = new Label("Brot 1");
      lBrot2 = new Label("Brot 2");
      lSumma = new Label("Summa");
      tBrot1 = new TextField(4);
      tBrot2 = new TextField(4);
      tSumma = new TextField(14);
      bLeggjaSaman = new Button("Leggja saman");

      this.add(lBrot1); this.add(tBrot1);
      this.add(lBrot2); this.add(tBrot2);
      this.add(lSumma); this.add(tSumma);
      this.add(bLeggjaSaman);
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bLeggjaSaman)
```

```
      {
         try
         {
            brot1 = new Brot(tBrot1.getText());
            brot2 = new Brot(tBrot2.getText());
            summa = brot1.plus(brot2);
            tSumma.setText(summa.toString());
         }
         catch (NefnariNullException fravik)
         {
            tSumma.setText("Núll í nefnara");
         }
         catch (NumberFormatException fravik)
         {
            tSumma.setText("Ekki tala.");
         }
         catch (Exception fravik)
         {
            tSumma.setText("Hvað er að gerast?");
         }
         return true;
      }
      return false;
   }
}
```

Í forrit_0D_02 er klasinn `NefnariNullException` tómur. Oft eru frávíksklasar hafðir tómir en þeir eru líka oft hafðir þannig að þeir geti borið skilaboð. Eigi að láta `NefnariNullException` bera skilaboð er t.d. hægt að hafa hann svona:

```
   public class NefnariNullException extends RuntimeException
   {
      String skilabod;

      public NefnariNullException()
      {
         skilabod = "0 í nefnara";
      }

      public NefnariNullException(String s)
      {
         skilabod = s;
      }

      public String toString()
      {
         return skilabod;
      }
   }
```

Væri frávíksklasinn skilgreindur svona þá væri hægt að hafa `catch`-blokk sem grípur `NefnariNullException` svona:

```
catch (NefnariNullException fravik)
{
   tSumma.setText(fravik.toString());
}
```

Skipunin í smiðnum `Brot` sem kastar frávikinu gæti þá notað hvorn smiðinn sem er og verið hvort heldur sem er svona:

```
if (nefnari == 0)
{
   NefnariNullException fravik = new NefnariNullException();
   throw fravik;
}
```

eða einhvern veginn svona:

```
if (nefnari == 0)
{
   NefnariNullException fravik =
      new NefnariNullException("Núll í nefnara, fuss og svei.");
   throw fravik;
}
```

Verkefni D.4 *
Bættu við klasann `Brot` aðferð til að deila broti í brot og láttu hana kasta frávikinu `NefnariNullException` ef útkoman er með 0 í nefnara. Prófaðu bæði að hafa frávikið tómt eins og í forrit_0D_02 og þannig að það geti borið skilaboð.

Verkefni D.5
Breyttu reiknivélinni sem var kláruð í 8. kafla þannig að hún skrifi skilaboð ef eitthvað annað en tölur er skrifað í textareitinn eða ef reynt er að setja fleiri tölur en komast í fylkið.

Verkefni D.6
Búðu til frávik sem heitir `EngarTolurIFylkiException` og bættu svo við lausnina á verkefni D.5 því sem þarf til að reikniaðferðirnar í klasanum `Talnasafn` kasti þessu fráviki ef reynt er að beita þeim á tómt fylki. Láttu reiknivélina skrifa kvörtun í reitinn fyrir útkomu ef reynt er að beita reikniaðgerðum meðan reiturinn fyrir tölurnar er tómur.

D.x. Spurningar og umhugsunarefni

1. Hvaða munur er á villu og fráviki (`Error` og `Exception`)?
2. Hvaða munur er á frávikum af gerðinni `RuntimeException` og öðrum gerðum frávika?
3. Hvaða hlutverk hafa orðin `try`, `catch`, `finally`, `throw` og `throws`?
4. Klasinn `NefnariNullException` í forrit_0D_02 er tómur. Hvaða vit er í að búa til klasa sem inniheldur hvorki eiginleika né aðferðir?

Til umhugsunar

Í kafla B.x var rætt um þann möguleika að forrit „stöðvaðist“ vegna þess að það lenti í endalausri slaufu eða þræðir biðu endalaust hver eftir öðrum. Þegar þetta gerist er hvorki um að ræða villu né frávik í þeim skilningi sem var til umfjöllunar í þessum kafla. Það getur semsagt ýmislegt farið úrskeiðis í keyrslu forrits án þess að villur og frávik séu á ferðinni.

Meðal þess sem getur valdið vandræðum er vélarbilanir, villur í stýrikerfi og hugsunar- og hönnunarvillur í forriti. Vandræði af síðastnefndu gerðinni verða þegar forrit er rangt eða óskynsamlega hugsað án þess að um villu eða frávik sé að ræða. Sem dæmi má taka: að forrit sem á að raða í stafrófsröð setji alla íslensku stafina aftan við z; eða forrit sem á að reikna bil milli tveggja dagsetninga á forminu dd.mm.áá líti svo á að 01.01.00 sé heilli öld á undan 31.12.99; eða forrit sem á að tefla skák færi peðin við og við aftur á bak.

Í ljósi þessa getum við skipt því sem fer úrskeiðis í keyrslu forrits gróflega í fjóra flokka:

i. Hegðunarvandamál sem má rekja til galla í vélbúnaði eða stýrikerfi.

ii. Villur í forriti. (Ef þýðandi vinnur sitt verk fullkomlega eins og til er ætlast ættu vandmál af þessu tagi ekki að geta komið upp.)

iii. Frávik sem ekki eru gripin.

iv. Hugsunarvillur og gallar í hönnun.

Gerðu ráð fyrir að `Applet` keyri í vefsjá og sýni myndir til skiptis eins og forrit_0B_01. Hvers konar vandamál gætu verið á ferðinni ef:

a) Forritið á að sýna 10 myndir til skiptis en sýnir í raun aðeins 9 og fyrsta myndin sést aldrei?

b) Forritið virkar eðlilega nema hvað það fer alls ekki af stað ef tvær vefsjár eru í gangi samtímis í tölvunni?

c) Forritið gengur eðlilega í 2 til 3 mínútur og stöðvast svo?

d) Forritið gengur yfirleitt eðlilega en þegar tölvan hefur verið lengi í gangi á vefsjáin það til að frjósa?

e) Forritið sýnir 3 fyrstu myndirnar í réttri röð og stöðvast svo?

f) Myndirnar birtast en stoppa svo stutt á skjánum að þær sjást tæplega nema sem ógreinilegt flökt?

E. kafli: Viðmót

E.a. Viðmót skilgreind

Í Java getur hver klasi aðeins haft einn yfirklasa. Eigi að búa til klasa sem samsvara tegundum spendýra væri hægt að byrja á að búa til einn yfirklasa og láta hann heita `Spendyr`. Svo væri hægt að búa til undirklasa eins og `Kattardyr`, `Ranadyr` og `Hundar`. Undirklasar `Kattardyr` yrðu svo `Ljon`, `Tigrisdyr` o.s.frv. Undirklasar `Ranadyr` gætu verið `IndverskurFill` og `Afrikufill` og undirklasar `Hundar` yrðu `Ulfur`, `Hyena` o.s.frv.

Svona getum við látið klasana líkja eftir dýraríkinu ef við höfum áhuga á skyldleika dýrategundanna. En ef við höfum meiri áhuga á hvaðan þær eru upprunnar þá búum við til klasa eins og `Afrikudyr`, `Asiudyr`, `Evropudyr` og `Amerikudyr`. `Afrikudyr` getur svo haft undirklasana `Ljon`, `Afrikufill` og `Hyena`. `Asiudyr` hefur þá undirklasa eins og `IndverskurFill`, `Tigrisdyr` og `Hyena`.

Kattardýr	
Ljón	Heimkynni: Afríka
Tígrisdýr	Heimkynni: Asía
Ranadýr	
Afríkufíll	Heimkynni: Afríka
Indverskur fíll	Heimkynni: Asía
Hundaætt	
Úlfur	Heimkynni: Evrópa, Ameríka, Asía
Hýena	Heimkynni: Afríka, Asía

Með þessu móti getum við engan veginn flokkað tegundir bæði eftir heimkynnum og eftir skyldleika. Við getum látið ljónið vera í senn spendýr, afríkudýr og kattardýr með því að hafa titillínur klasanna

```
class Kattardyr extends Spendyr
class Afrikudyr extends Kattardyr
class Ljon extends Afrikudyr
```

En það er engin leið að búa til ættartré þannig að `Ljon` og `Tigrisdyr` erfi `Kattardyr`, `Ljon` og `Afrikufíll` erfi `Afríkudýr` og `Afrikufill` erfi `Ranadyr` án þess að gera ljónið að ranadýri eða fílinn að kattardýri.

Það er hægt að leysa bæði þessi vandamál með því að nota viðmót (`interface`) því þótt hver klasi geti aðeins haft einn yfirklasa getur hann erft mörg viðmót. Ef við búum til klasa sem heita `Kattardyr`, `Ranadyr`, `Hundar`, `Tigrisdyr`, `Ljon`, `Afrikufill`, `InverskurFill`, `Ulfur` og `Hyena` og viðmót sem heita `Afrikudyr`, `Asiudyr` og `Amerikudyr` þá getum við skilgreint allar dýrategundirnar með því að hafa titillínur þeirra svona:

```
class Ljon extends Kattardyr implements Afrikudyr
class Tigrisdyr extends Kattardyr implements Asiudyr
class Afrikufill extends Ranadyr implements Afrikudyr
class IndverskurFill extends Ranadyr implements Asiudyr
class Ulfur extends Hundar implements Evropudyr, Amerikudyr, Asiudyr
class Hyena extends Hundar implements Afrikudyr, Asiudyr
```

Það mætti líka hugsa sér að skilgreina `Kattardyr`, `Ranadyr` og `Hundar` sem viðmót fremur en `Klasa` og hafa titillínur dýrategundanna svona:

```
class Ljon extends Spendyr implements Kattardyr, Afrikudyr
```

```
class Tigrisdyr extends Spendyr implements Kattardyr, Asiudyr
...
class Hyena extends Spendyr implements Hundar, Afrikudyr, Asiudyr
```

Viðmót eru eins og klasar með eintómum `abstract` aðfeðrum. Aðferðirnar í viðmótum eru sjálfkrafa `public` og `abstract` þótt það sé ekki tekið fram. Klasabreytur sem skilgreindar eru í viðmóti eru alltaf `static` og `final`, þ.e. þær hafa alltaf fast óbreytanlegt innihald. Viðmót eru sem sagt ýmsum takmörkum háð.

Viðmót sem heitir `Hundar` og inniheldur aðferðirnar `spangola`, `gelta` og `urra` er skilgreint svona (ef gert er ráð fyrir að engin aðferðanna taki við gildi eða skili útkomu):

```
public interface Hundar
{
   void spangola();
   void gelta();
   void urra();
}
```

Viðmótið inniheldur aðeins titillínur aðferðanna þar sem þær eru `abstract` og klasi sem setur upp (`implements`) þetta viðmót verður að útfæra þær. Aðferðirnar eru `public` og `abstract` þótt það sé ekki tekið fram. Það mætti svo sem hafa viðmótið svona:

```
public interface Hundar
{
   public abstract void spangola();
   public abstract void gelta();
   public abstract void urra();
}
```

En orðin `public` og `abstract` eru óþörf og hafa engin áhrif.

E.b. Dæmi um viðmót

forrit_0E_01 er byggt á forrit_05_02. Það inniheldur klasann `Myndhluti` sem hefur undirklasana `Punktur`, `Strik`, `Hringur` og `Trihyrningur`. `Myndhluti` er svipaður samnefndum klasa í forrit_05_02 en hefur tvær `abstract` aðferðir til viðbótar sem eru `syna` og `fela`. Það er því hægt að sýna og fela hringi og strik þótt þessir hlutir séu geymdir í breytum sem eru skilgreindar sem `Myndhluti`.

Hringir, strik og þríhyrningar eiga það sameiginlegt að vera ferlar og hafa ferillengd. Ferillengd hrings og þríhyrnings er einfaldlega ummálið og ferillengd striksins er lengdin á því. Hringir og þríhyrningar eiga það sameiginlegt að vera fletir og hafa flatarmál. forrit_0E_01 inniheldur tvö viðmót sem heita `Flotur` og `Ferill`. Titillínur klasa og viðmóta í forritinu eru.

```
public abstract class Myndhluti
interface Flotur
interface Ferill
public class Punktur extends Myndhluti
public class Strik extends Myndhluti implements Ferill
public class Hringur extends Myndhluti implements Flotur, Ferill
public class Trihyrningur extends Myndhluti implements Flotur, Ferill
```

Viðmótið `Flotur` hefur eina aðferð sem er `flatarmal` og `Ferill` hefur eina aðferð sem er `ferillengd`. Viðmótin eru svona:

```
// Ferill
```

```
// forrit_0E_01
//
// Allir klasar sem hafa viðmótið Ferill verða
// að útfæra aðferðina ferillengd
//
interface Ferill
{
   double ferillengd();
} // Hér endar Ferill

//
// Flotur
// forrit_0E_01
//
// Allir klasar sem hafa viðmótið Flotur verða
// að útfæra aðferðina flatarmal
//
interface Flotur
{
   double flatarmal();
} // Hér endar Flotur
```

`Myndhluti`, sem er yfirklasi klasanna `Punktur`, `Strik`, `Hringur` og `Trihyrningur`, er svona:

```
import java.awt.*;
//
// forrit_0E_01
// Myndhluti
//
public abstract class Myndhluti
{
   protected Color bakgrlitur;
   protected Color litur;

   public Myndhluti()  // er framkvæmt af smið erfingja
   {
      bakgrlitur = new Color(192, 192, 192);
      litur = new Color(0, 0, 0);
   }

   public void litur(int R, int G, int B)
   {
      litur = new Color(R, G, B);
   }

   public void bakgrlitur(int R, int G, int B)
   {
      bakgrlitur = new Color(R, G, B);
   }

   public abstract void syna(Graphics g);

   public abstract void fela(Graphics g);

} // Hér endar Myndhluti
```

`Punktur` er eins og samnefndur klasi í forrit_05_02.

```
import java.awt.*;
//
// forrit_0E_01
// Punktur
```

```
//
public class Punktur extends Myndhluti
{
   protected double x;
   protected double y;

   public Punktur()
   {
      x = 0; y = 0;
   }

   public Punktur(double xhnit, double yhnit)
   {
      x = xhnit; y = yhnit;
   }

   public void faera(double dx, double dy)
   {
      x += dx; y += dy;
   }

   public void faeraTil(double xhnit, double yhnit)
   {
      x = xhnit; y = yhnit;
   }

   // Aðferðir sem tilheyra Myndhluti

   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawArc((int)x-2, (int)y-2, 4, 4, 0, 360);
   }

   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawArc((int)x-2, (int)y-2, 4, 4, 0, 360);
   }
} // Hér endar Punktur
```

`Strik` er eins og samnefndur klasi í forrit_05_02 nema hvað

```
   implements Ferill
```

hefur verið bætt við titillínuna og aðferðinni `ferillengd` hefur verið bætt við enda verður klasinn að hafa hana til að útfæra viðmótið `Ferill`.

```
import java.awt.*;
//
// forrit_0E_01
// Strik
//
public class Strik extends Myndhluti implements Ferill
{
   protected Punktur p1, p2;

   public Strik(Punktur punktur1, Punktur punktur2)
   {
      p1 = punktur1;
      p2 = punktur2;
   }

   // Aðferðir sem tilheyra Myndhluti

   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
   }

   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
   }

   // Aðferð sem tilheyrir Ferill

   public double ferillengd()
   {
      return Math.sqrt ((p1.x - p2.x)*(p1.x - p2.x) +
                        (p1.y - p2.y)*(p1.y - p2.y));
   }
} // Hér endar Strik
```

Klasinn `Hringur` útfærir bæði viðmótin `Flotur` og `Ferill` og hefur bæði aðferðir til að reikna ferillengd og flatarmál.

```
import java.awt.*;
//
// forrit_0E_01
// Hringur
//
//
public class Hringur extends Myndhluti implements Flotur, Ferill
{
   protected Punktur midja;
   protected double radius;

   public Hringur(Punktur p, double r)
   {
      midja = p;
      radius = r;
   }
```

```
   // Aðferðir sem tilheyra Myndhluti
   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawArc((int)(midja.x-radius), (int)(midja.y-radius),
                  (int)(2*radius), (int)(2*radius), 0, 360);
   }

   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawArc((int)(midja.x-radius), (int)(midja.y-radius),
                  (int)(2*radius), (int)(2*radius), 0, 360);
   }

   // Aðferð sem tilheyrir Flotur
   public double flatarmal()
   {
      return radius*radius*Math.PI;
   }

   // Aðferð sem tilheyrir Ferill
   public double ferillengd()
   {
      return radius*2*Math.PI;
   }
} // Hér endar Hringur
```

Klasinn `Trihyrningur` útfærir bæði viðmótin `Flotur` og `Ferill` og hefur bæði aðferðir til að reikna ferillengd og flatarmál. Aðferðin `flatarmal` notar vel þekkta formúlu úr línulegri algebru sem ekki verður hirt um að skýra hér. Aðferðin `ferillengd` byggir ekki á neinu flóknara en pýþagórasarreglu.

```
import java.awt.*;
//
// forrit_0E_01
// Trihyrningur
//
public class Trihyrningur extends Myndhluti implements Flotur, Ferill
{
   protected Punktur p1, p2, p3;

   public Trihyrningur(Punktur a, Punktur b, Punktur c)
   {
      p1=a; p2=b; p3=c;
   }

   // Aðferðir sem tilheyra Myndhluti

   public void syna(Graphics g)
   {
      g.setColor(litur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
      g.drawLine((int)p2.x, (int)p2.y, (int)p3.x, (int)p3.y);
      g.drawLine((int)p3.x, (int)p3.y, (int)p1.x, (int)p1.y);
   }
```

```
   public void fela(Graphics g)
   {
      g.setColor(bakgrlitur);
      g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
      g.drawLine((int)p2.x, (int)p2.y, (int)p3.x, (int)p3.y);
      g.drawLine((int)p3.x, (int)p3.y, (int)p1.x, (int)p1.y);
   }

   // Aðferð sem tilheyrir Flotur
   public double flatarmal()
   {
      return 0.5*Math.abs(p1.x*p2.y + p2.x*p3.y + p3.x*p1.y
                          -p1.x*p3.y - p2.x*p1.y - p3.x*p2.y);
   }

   // Aðferð sem tilheyrir Ferill
   // skilar samanlegðri lengd þriggja strika.
   public double ferillengd()
   {
      return Math.sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                       (p1.y - p2.y)*(p1.y - p2.y))
           + Math.sqrt((p2.x - p3.x)*(p2.x - p3.x) +
                       (p2.y - p3.y)*(p2.y - p3.y))
           + Math.sqrt((p3.x - p1.x)*(p3.x - p1.x) +
                       (p3.y - p1.y)*(p3.y - p1.y));
   }
} // Hér endar Trihyrningur
```

Lítum nú á hvernig hægt er að nota þessar tegundir og viðmót. Það er gert í klasanum `Mynd1`.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0E_01
// Mynd1
//
//
public class Mynd1 extends Applet
{
   final int MAX_FJOLDI = 25;
   Myndhluti[] m;
   Button bReikna;
   TextField tUtkoma;

   public void init()
   {
      bReikna = new Button("Reikna");
      this.add(bReikna);
      tUtkoma = new TextField(15);
      this.add(tUtkoma);
      m = new Myndhluti[MAX_FJOLDI];
      m[ 0] = new Punktur(50, 50);
      m[ 1] = new Punktur(100, 50);
      m[ 2] = new Punktur(75, 75);
      m[ 3] = new Trihyrningur((Punktur)m[0],
                               (Punktur)m[1], (Punktur)m[2]);
      m[ 4] = new Punktur(100, 125);
      m[ 5] = new Hringur((Punktur)m[4], 30);
      m[ 6] = new Punktur(25, 175);
      m[ 7] = new Punktur(150, 150);
      m[ 8] = new Strik((Punktur)m[6], (Punktur)m[7]);
      m[ 9] = new Punktur(175, 175);
      m[10] = new Punktur(175, 75);
```

```
      m[11] = new Trihyrningur((Punktur)m[7],
                                  (Punktur)m[9], (Punktur)m[10]);
      m[12] = new Hringur((Punktur)m[10], 15);
      m[13] = new Strik((Punktur)m[4], (Punktur)m[6]);
   }

   public void paint(Graphics g)
   {
      g.drawString("Flatarmál og lengd ferla", 10, 40);
      for(int i=0; i<m.length; i++)
      {
         if (m[i] instanceof Punktur)
         {
            m[i].litur(255,0,0);
         }
         if (m[i] != null)
         {
            m[i].syna(g);
         }
      }
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bReikna)
      {
         double samanlagtFlatarmal = 0;
         for(int i=0; i<m.length; i++)
         {
            if (m[i] instanceof Flotur)
            {
               samanlagtFlatarmal += ((Flotur)m[i]).flatarmal();
            }
         }
         double samanlogdLengd = 0;
         for(int i=0; i<m.length; i++)
         {
            if (m[i] instanceof Ferill)
            {
               samanlogdLengd += ((Ferill)m[i]).ferillengd();
            }
         }
         tUtkoma.setText("F= " + (int)samanlagtFlatarmal +
                         " U=" + (int)samanlogdLengd);
         return true;
      }
      return false;
   }
}
```

`Applet`-ið `Mynd1` hefur `init`-aðferð sem býr til 14 myndhluta (8 punkta, 2 þríhyrninga, 2 hringa og 2 strik) og setur þá alla í fylki.

Taktu eftir hvernig hringur er búinn til úr punktinum í hólfi 4 og tölunni 30 og settur í hólf númer 5 í fylkinu.

```
   m[ 5] = new Hringur((Punktur)m[4], 30);
```

Þar sem smiðurinn `Hringur` tekur við gildi af tegundinni `Punktur` en ekki `Myndhluti` þarf að breyta `m[4]` í punkt með `(Punktur)m[4]`. Þetta er vandræðalaust að gera því `m[4]` er `Punktur`. Sama aðferð er notuð til að breyta breytum af tegundinni `Myndhluti` í tegundina `Punktur` þegar þríhyrningarnir eru smíðaðir.

Aðferðin `paint` litar alla punktana rauða og sýnir alla myndhlutana í fylkinu. Til að tryggja að ekki sé reynt að sýna það sem auð hólf í fylkinu vísa á eru skipanir um að framkvæma `syna` settar inn í `if`-blokk og aðeins framvæmdar ef hólf inniheldur ekki `null`. (Í fylki af hlutum hafa þau sæti sem ekkert hefur verið sett í gildið `null` rétt eins og allar hlutabreytur sem ekki hefur verið gefið neitt gildi.)

```
if (m[i] != null)
{
   // Það sem er hér er því aðeins framkvæmt að m[i] vísi á hlut.
}
```

Sé reynt að meðhöndla breytu sem geymir `null` eins og hlutur væri í henni er kastað fráviki af tegundinni `NullPointerException`. Það væri því sem best hægt að hafa `paint`-aðferðina svona:

```
public void paint(Graphics g)
{
   g.drawString("Flatarmál og lengd ferla", 10, 40);
   for(int i=0; i<m.length; i++)
   {
      if (m[i] instanceof Punktur)
      {
         m[i].litur(255,0,0);
      }
      try
      {
         m[i].syna(g);
      }

      catch (NullPointerException fravik)
      {
         // Ekkert er gert þó NullPointerException sé kastað
      }
   }
}
```

Í `action`-aðferðinni er reiknað samanlagt flatarmál og ferillengd allra flata og ferla. Skipanirnar sem sjá um að reikna flatarmálið eru

```
double samanlagtFlatarmal = 0;
for(int i=0; i<m.length; i++)
{
   if (m[i] instanceof Flotur)
   {
      samanlagtFlatarmal += ((Flotur)m[i]).flatarmal();
   }
}
```

Hér þarf ekki að hafa áhyggjur af `NullPointerException` því ekkert er gert við hlut nema hann sé af tegundinni `Flotur`, enda kemur gildið `false` út úr

```
(m[i] instanceof Flotur)
```

ef `m[i]` inniheldur `null`.

Verkefni E.1 *
Klasinn `Mynd3` í forrit_05_02 er frumstætt teikniforrit sem hægt er að nota til að teikna bein strik. Notaðu viðmótið `Ferill` til að bæta við hann því sem þarf til að samanlögð lengd allra strikanna sem búið er að teikna sjáist í textareit.

Verkefni E.2
Bættu við viðmótið `Flotur` aðferðinni lita og útfærðu hana í klösunum `Hringur` og `Trihyrningur`. Vandalaust er að útfæra aðferðina fyrir hringa með því að nota `fillArc` sem tilheyrir tegundinni `Graphics`. Nokkru snúnara er að lita þríhyrning. Líklega er best að nota aðferðina

```
fillPolygon(int xPoints[], int yPoints[], int nPoints);
```

sem tilheyrir líka `Graphics`.

Bættu svo einum takka við klasann `Mynd1` og láttu alla fleti litast gula þegar ýtt er á hann.

Verkefni E.3
Hafir þú leyst verkefni E.2. getur þú nú litað hringa og þríhyrninga (þ.e. fyllt fleti af lit). Það má líka hugsa sér að lita punkta. Skilgreindu viðmótið `Litanlegur` og láttu það hafa eina aðferð sem heitir `lita`. Breyttu svo forrit_0E_01 þannig að tegundirnar `Punktur`, `Hringur` og `Trihyrningur` útfæri viðmótið `Litanlegur`. Bættu svo einum takka við klasann `Mynd1` og láttu alla litanlega hluti litast græna þegar ýtt er á hann.

E.c. Að breyta úr einni tegund í aðra

Allir hlutir af tegundinni `Hringur` eru líka af tegundunum `Myndhluti`, `Ferill`, `Flotur` og `Object`. Sé hringur geymdur í breytu af tegundinni `Hringur` er hægt að færa hann í breytur af tegundunum `Myndhluti`, `Flotur`, `Ferill` eða `Object`. Ef breytan `h` er skilgreind og henni gefið gildi svona:

```
Hringur h;
h = new Hringur(new Punktur(120, 120), 30);
```

þá er allt í lagi að færa innihald hennar í einhverja yfirtegund með skipunum á borð við:

```
Object o;
o = (Object)h;
```

eða

```
Ferill f;
f = (Ferill)h;
```

Hins vegar kann að vera varasamara að breyta hlut í undirtegund sína. Þetta er þó í lagi ef hann er í raun og veru af þeirri gerð sem breytt er í. Til dæmis er allt í lagi að gefa skipanirnar

```
Myndhluti m;
Hringur h;
Flotur f;
m = new Hringur(new Punktur(120, 120), 30);
f = (Flotur)m;
h = (Hringur)m;
```

því `m` vísar á hlut sem er í raun og veru bæði `Flotur` og `Hringur`. Það er sem sagt allt í lagi að setja hlut sem er skilgreindur sem `Hringur` í breytu af tegundinni `Myndflotur` því allir hringir eru myndfletir. Hins vegar er óráðlegt að setja hlut sem

er skilgreindur sem `Myndflotur` í breytu af tegundinni `Hringur` nema í þeim tilvikum sem myndflöturinn er hringur.

Í `action`- og `paint`-aðferðunum í klasanum `Mynd2` sjást nokkur dæmi um hvernig hægt er að breyta milli tegunda.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_0E_01
// Mynd2
//
//
public class Mynd2 extends Applet
{
   Flotur[] f;
   Button bStaekka, bMinnka;
   TextField tUtkoma;
   Graphics g;

   public void init()
   {
      bStaekka = new Button("Stækka");
      bMinnka = new Button("Minnka");
      this.add(bStaekka);
      this.add(bMinnka);
      tUtkoma = new TextField(28);
      this.add(tUtkoma);

      g = this.getGraphics();

      f = new Flotur[4];

      f[0] = new Trihyrningur(new Punktur(10, 180),
                              new Punktur(40, 180),
                              new Punktur(25, 120));
      f[1] = new Hringur(new Punktur(120, 120), 30);
      f[2] = new Trihyrningur(new Punktur(60, 180),
                              new Punktur(100, 180),
                              new Punktur(80, 100));
      f[3] = new Trihyrningur(new Punktur(120, 180),
                              new Punktur(190, 180),
                              new Punktur(170, 110));
      tUtkoma.setText("Samanlagt flatarmál er " + reiknaFlatarmal());
   }

   public void paint(Graphics g)    // Sýnir alla hluti í fylkinu f
   {
      for(int i=0; i<f.length; i++)
      {
         if (f[i] != null)
         {
            ((Myndhluti)f[i]).syna(g);
         }
      }
   }

   public double reiknaFlatarmal()  // Reiknar samanlagt flatarmál
   {                                // allra hluta í fylkinu f.
      double samanlagtFlatarmal = 0;
      for(int i=0; i<f.length; i++)
      {
         samanlagtFlatarmal += f[i].flatarmal();
      }
```

```
      return samanlagtFlatarmal;
   }

   public void minnka(Hringur h)
   {
      h.radius -= 5;
   }

   public void staekka(Hringur h)
   {
      h.radius += 5;
   }

   public void minnka(Trihyrningur t)
   {
      t.p3.y += 5;
   }

   public void staekka(Trihyrningur t)
   {
      t.p3.y -= 5;
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bStaekka)
      {
         for(int i=0; i<f.length; i++)
         {
            ((Myndhluti)f[i]).fela(g);
            if (f[i] instanceof Hringur)
            {
               staekka((Hringur)f[i]);
            }
            if (f[i] instanceof Trihyrningur)
            {
               staekka((Trihyrningur)f[i]);
            }
            ((Myndhluti)f[i]).syna(g);
         }
         tUtkoma.setText("Flatarmál er alls " + reiknaFlatarmal());
         return true;
      }
```

```
      if (e.target == bMinnka)
      {
         for(int i=0; i<f.length; i++)
         {
            ((Myndhluti)f[i]).fela(g);
            if (f[i] instanceof Hringur)
            {
               minnka((Hringur)f[i]);
            }
            if (f[i] instanceof Trihyrningur)
            {
               minnka((Trihyrningur)f[i]);
            }
            ((Myndhluti)f[i]).syna(g);
         }
         tUtkoma.setText("Samanlagt flatarmál er " +
                                          reiknaFlatarmal());
         return true;
      }
      return false;
   }
}
```

Taktu eftir því að þótt `Flotur` sé ekki klasi heldur viðmót (`interface`) þá er hægt að skilgreina breytur af tegundinni `Flotur`. Hún er eins og hver annar `abstract` klasi.

Taktu líka eftir því að `Mynd2` hefur tvær aðferðir sem heita `minnka` og tvær sem heita `staekka`. Það er ekkert því til fyrirstöðu að tvær aðferðir heiti sama nafni svo framarlega sem þær taka ekki við sams konar gildum. Gildið sem sent er stjórnar því þá hvor aðferðin er notuð. Ef `x` er af tegundinni `Hringur` og skipunin

```
minnka(x)
```

er gefin þá er aðferðin

```
public void minnka(Hringur h)
{
   h.radius -= 5;
}
```

framkvæmd. Sé `x` hins vegar af tegundinni `Trihyrningur` þá er aðferðin

```
public void minnka(Trihyrningur t)
{
   t.p3.y += 5;
}
```

framkvæmd. Nú er fylkið `f` hvorki af gerðinni `Hringur` né `Trihyrningur` en í því eru samt hlutir af þeim gerðum og það er hægt að athuga hvort hólf í fylkinu inniheldur hlut af tegundinni `Hringur` og minnka hann ef svo er með skipununum:

```
if (f[i] instanceof Hringur)
{
   minnka((Hringur)f[i]);
}
```

Með þessu móti er `minnka` aðferðinni sendur hlutur af tegundinni `Hringur`.

Viðmótið `Flotur` hefur enga aðferð sem heitir `syna` en það samt hægt að sýna hlut af tegundinni `Flotur` með því að breyta honum í tegund sem hefur `syna`-aðferð. Þar sem allar tegundir sem erfa viðmótið `Flotur` eru undirtegundir `Myndhluti` er vandræðalaust að breyta hvaða fleti sem er í myndhluta og sýna hann svona

```
((Myndhluti)f[i]).syna(g);
```

Taktu eftir svigunum. Ef aðeins stæði

```
(Myndhluti)f[i].syna(g);
```

væri verið að breyta útkomunni úr `f[i].syna(g)` í tegundina `Myndhluti` og það er ekki hægt, enda skilar `syna`-aðferðinn alls engri útkomu og þar með ekki neinu sem hægt er að breyta í tegundina `Myndhluti`.

Verkefni E.4
Breyttu lausninni á verkefni 5.11 þannig að allir teiknaðir hlutir séu geymdir í fylki og hægt sé að láta forritið reikna samanlagða lengd allra ferla og samanlagða stærð allra flata.

E.d. `Cloneable`

Í pakkanum `java.lang` er viðmót sem heitir `Cloneable`. Um það var fjallað lítillega í 7. kafla. Þetta viðmót er tómt, inniheldur engar aðferðir og er einfaldlega skilgreint svona.

```
public interface Cloneable
{
}
```

Hlutverk þessa viðmóts er einungis að merkja þá klasa sem mega nota aðferðina `clone` í klasanum `Object`. Eigi að vera leyfilegt að afrita hlut með `clone`-aðferðinni verður klasinn sem hann tilheyrir að erfa `Cloneable` viðmótið. Sé reynt að beita clone á hluti sem ekki erfa `Cloneable` er kastað fráviki af gerðinni `CloneNotSupportedException`.

Ef klasinn `Myndhluti` hefur titillínuna

```
public abstract class Myndhluti implements Cloneable
```

og breytan `x` er af tegundinni `Myndhluti` þá er hægt að búa til annan hlut sem er nákvæmlega eins og `x` og setja hann í breytuna `y` með skipununum

```
Myndhluti y;
try {y = (Myndhluti)x.clone();}
catch (CloneNotSupportedException fravik) {}
```

Útkoman úr `clone`-aðferðinni er af tegundinni `Object` og það þarf því að breyta henni í tegundina `Myndhluti`.

E.x. Spurningar og umhugsunarefni

1. Hvaða munur er á viðmóti og klasa?
2. Hvaða hlutverki gegna orðin `extends` og `implements`?
3. Hugsaðu þér að klasinn `Randyr` hafi yfirklasa sem heitir `Spendyr` og undirklasa sem heitir `Tigrisdyr` og breyturnar `r`, `s`, `t` séu skilgreindar svona:

   ```
   Randyr r;
   Spendyr s;
   Tilgrisdyr t;
   ```

   Undir hvaða kringumstæðum er óhætt að gefa eftirfarandi skipun?

   ```
   t = (Tigrisdyr)s;
   ```

4. Undir hvaða kringumstæðum er óhætt að gefa skipunina

   ```
   s = (Spendyr)t;
   ```

   ef `r`, `s` og `t` eru skilgreindar eins og í spurningu 3?

5. Hvaða munur er á merkingu þessara tveggja skipana?

   ```
   s = ((Object)x).toString();
   s = (Object)x.toString();
   ```

6. Hvaða hlutverk hefur viðmótið `Cloneable`?

Til umhugsunar

Í sumum forritunarmálum, eins og t.d. C++, er hægt að láta einn klasa erfa marga aðra. Í Java getur hver klasi aðeins erft einn klasa (og þá um leið yfirklasa hans). En það eru engin takmörk fyrir því hvað sami klasi getur erft mörg viðmót.

Að hvaða leyti væri forritun í Java erfiðari en hún er ef klasar gætu ekki erft neinn klasa, fyrir utan Object, heldur aðeins viðmót?

Eru einhver verkefni sem væri beinlínis ómögulegt að leysa ef klasar gætu aðeins erft Object og viðmót?

Hvaða vandamál gætu fylgt því að leyfa einum klasa að erfa marga aðra?

F. kafli: Straumar, skrár og URL

F.a. Straumar, inntak og úttak

Í pakkanum `java.io` eru klasar sem hægt er að nota til að lesa úr skrám og skrifa í þær. Einn þessara klasa heitir `InputStream`. Hann er `abstract` en hefur undirklasa sem geta opnað skrár og lesið úr þeim. Annar `abstract` klasi í `java.io` er `OutputStream`. Sumir undirklasar hans hafa aðferðir til að opna skrár og skrifa í þær.

`InputStream`, `OutputStream` og undirklasar þeirra kallast straumar. Með þessari nafngift er klösunum líkt við eitthvað sem rennur eins og vatn rennur eftir röri eða farvegi. Við getum hugsað okkur að gögn renni inn í tölvuna af diski eftir inntaksstraumi (`InputStream`) og þau renni úr tölvunni út á disk eftir úttaksstraumi (`OutputStream`).

Gögn sem skrifuð eru í skrá eru í raun og veru runur af átta bita tvíundakerfistölum, eða heilum tölum milli 0 og 255. Þessar talnarunur má túlka á ýmsa vegu. Með því að líta á þær sem númerskóða fyrir bókstafi má til dæmis skoða þær sem texta, og með því að slá saman fjórum og fjórum bætum, í 32 bita romsur, má túlka þær sem heiltölur af gerðinni `int`.

`InputStream` og `OutputStream` hafa marga undirklasa. Hér verður aðeins fjallað um tvo inntaksstrauma og þrjá úttaksstrauma. Þeir eru:

`FileInputStream`	Undirklasi `InputStream`. Þessi klasi hefur aðferðir til að opna skrá á diski til lestrar og loka henni aftur. Hann getur einnig sótt gögn úr skrá en aðeins á formi heiltalna af gerðinni `byte`.
`DataInputStream`	Undirklasi `InputStream`. `DataInputStream` getur ekki opnað skrá en hann getur lesið gögn úr öðrum inntaksstraumi, t.d. `FileInputStream`. `DataInputStream` hefur aðferðir til að túlka gögn sem einfaldar tegundir (`int`, `double`, `char` o. s. frv.)
`FileOutputStream`	Undirklasi `OutputStream`. Getur opnað skrá til útskriftar og lokað henni aftur en aðeins skrifað í hana gögn á formi heiltalna af tegundinni `int`.
`PrintStream`	Undirklasi `OutputStream`. `PrintStream` getur ekki opnað skrá en hann getur skrifað strengi í annan úttaksstraum, t.d. `FileOutputStream`.
`DataOutputStream`	Undirklasi `OutputStream`. Getur ekki opnað skrá en getur skrifað allar einfaldar tegundir (`int`, `double`, `char` o.s.frv.) í annan úttaksstraum t.d. `FileOutputStream`.

Eigi að opna skrá sem heitir a:\data.txt og lesa úr henni streng sem endar á línuskilum (þ.e. eina línu af texta) er hægt að gera það með skipununum

```
FileInputStream f;
DataInputStream d;
String einLina;
try
{
   // Straumurinn f búinn til og látinn opna skrána a:\data.txt
   f = new FileInputStream("a:\data.txt");
   // d búinn til og hann stilltur á að taka við úr strauminum f.
   d = new DataInputStream(f);
   einLina = d.readLine();
   f.close();  // Skrúfað fyrir strauminn f
   d.close();  // Skrúfað fyrir strauminn d
}
```

```
   // Smiðurinn FileInputStream og aðferðin readline kasta
   // fráviki af gerðinni IOException ef eitthvað fer úrskeiðis.
   catch (IOException fravik)
   {
      // Hér kemur það sem á að gera ef fráviki af gerðinni
      // IOException  er kastað.
   }
```

Eigi að búa til skrá með nafninu a:\data.txt og skrifa í hana strenginn "Sísí sá sól." er hægt að nota skipanirnar

```
   FileOutputStream f;
   PrintStream p;
   try
   {
      // Straumurinn f búinn til og hann stilltur á
      // að skrifa í skrána a:\data.txt
      f = new FileOutputStream("a:\data.txt");
      // p búinn til og hann stilltur á að skrifa í strauminn f.
      p = new PrintStream(f);
      p.print("Sísí sá sól.");
      // Skrúfað fyrir strauma.
      f.close();
      p.close();
   }
   // Smiðurinn FileOutputStream kastar fráviki af
   // tegundinni IOException ef eitthvað fer úrskeiðis.
   catch (IOException fravik)
   {
      // Hér kemur það sem á að gera ef fráviki af gerðinni
      // IOException  er kastað.
   }
```

Hér fer á eftir heilt forrit sem getur sótt og vistað texta í skrá. Það er sjálfstætt forrit en ekki `Applet` enda er `Applet`-um bannað að nota gagnageymslur (diska og disklinga) eins og fjallað var um í kafla 9.a.

Klasinn `Starta` gerir ekkert annað en að búa til einn hlut af gerðinni `Textaskrar1`, framkvæma `init`-aðferð hans, koma honum fyrir á skjánum og sýna hann.

```
//
// forrit_0F_01
// Starta
// Byggt á samnefndum klasa í forrit_09_01
//
public class Starta
{
   public static void main(String args[])
   {
      Textaskrar1 s = new Textaskrar1("Textaskrár");
      s.init();
      s.resize(250, 350);
      s.move(50, 100);
      s.show();
   }
}
```

Klasinn `Textaskrar1` sem hér fer á eftir er undirklasi `Frame` (en ekki `Applet`). Smiðurinn gerir ekkert annað en að senda smið yfirtegundarinnar streng með heiti sem skrifað skal efst í rammann.

`Applet` byrja ævinlega á að framkvæma aðferðina `init`. Hér er aðferðnni sem byrjað er á gefið sama nafn þótt það sé engin nauðsyn. Hún mætti eins heita `byrja`, `starta` eða hvaða öðru nafni sem er.

Eins og fjallað var um í kafla 9.d hefur `Frame` sjálfkrafa `BorderLayout`. Hér er rammanum gefið `FlowLayout` (eins og það sem `Applet` hafa sjálfkrafa) með skipununum

```
FlowLayout fl = new FlowLayout();
this.setLayout(fl);
```

Aðferðirnar til að lesa úr skrá og skrifa í skrá eru þær sömu og lýst var hér að ofan. Til að lesa allt innihald skráar er notuð slaufan

```
while (d.available() != 0)
{
   einLina = d.readLine();
   allurTexti = allurTexti + einLina + "\n";
}
```

Aðferðin `available`, sem `DataInputStream` erfir, skilar tölu sem segir hvað er mikið eftir ólesið í straumi. Slaufan er framkvæmd aftur og aftur meðan þessi tala er ekki núll.

```
import java.awt.*;
import java.io.*;
//
// forrit_0F_01
// Textaskrar1
//
public class Textaskrar1 extends Frame
{
   TextField tSkraarnafn;
   TextArea tTexti;
   Label lSkraarnafn, lTexti;
   Button bVista, bOpna, bHaetta;

   // Smiðurinn gerir ekkert annað en að kalla á smið
   // yfirklasans Frame.
   public Textaskrar1(String titill)
   {
      super(titill);
   }

   public void init()
   {
      FlowLayout fl = new FlowLayout();
      this.setLayout(fl);

      lSkraarnafn = new Label("Nafn skráar ");
      tSkraarnafn = new TextField(25);
      lTexti = new Label("Texti ");
      tTexti = new TextArea(10, 25);
      bVista = new Button("Vista");
      bOpna = new Button("Opna");
      bHaetta = new Button("Hætta");

      this.add(lSkraarnafn); this.add(tSkraarnafn);
      this.add(lTexti); this.add(tTexti);
      this.add(bVista); this.add(bOpna);
      this.add(bHaetta);
   }

   public String lesaTextaskra(String skrNafn)
```

```
{
   // FileInputStream er tegund sem tekur við
   // úr skrá. DataInputStream tekur við gögnum
   // úr öðrum straumi, hér FileInputStream.
   FileInputStream f;
   DataInputStream d;
   String allurTexti = "";
   String einLina;
   try
   {
      // FileInputStream búinn til og hann stilltur á
      // að taka við úr skránni sem nefnd er í skrNafn.
      f = new FileInputStream(skrNafn);
      // DataInputStream búinn til og hann stilltur á
      // að taka við úr strauminum f.
      d = new DataInputStream(f);

      // Aðferðin available skilar tölu sem segir hvað
      // mikið er eftir ólesið í straumnum.
      while (d.available() != 0)
      {
         einLina = d.readLine();
         allurTexti = allurTexti + einLina + "\n";
      }

      // Skrúfað fyrir strauma.
      f.close();
      d.close();
   }
   // Smiðirnir FileInputStream og DataInputStream
   // og aðferðin readLine kasta fráviki af tegundinni
   // IOException ef eitthvað fer úrskeiðis.
   catch (IOException fravik)
   {
      tSkraarnafn.setText("Get ekki lesið " + skrNafn);
   }

   // Aðferðin skilar streng sem er samsettur
   // úr öllu sem lesið var úr skránni.
   return allurTexti;
}

public void skrifaTextaskra(String skrNafn, String texti)
{
   FileOutputStream f;
   PrintStream p;
   try
   {
      // FileOutputStream búinn til og hann stilltur á
      // að skrifa í skrána sem nefnd er í skrNafn.
      f = new FileOutputStream(skrNafn);
      // PrintStream búinn til og hann stilltur á
      // að skrifa í strauminn f.
      p = new PrintStream(f);

      // Innihald breytunnar texti skrifað í strauminn p.
      p.print(texti);
      // Skrúfað fyrir strauma.
      f.close();
      p.close();
   }
   // Smiðurinn FileOutputStream kastar fráviki af
   // tegundinni IOException ef eitthvað fer úrskeiðis.
   catch (IOException fravik)
   {
      tSkraarnafn.setText("Get ekki skrifað " + skrNafn);
```

```
        }
    }

    public boolean action(Event e, Object o)
    {
        if (e.target == bVista)
        {
            skrifaTextaskra(tSkraarnafn.getText(), tTexti.getText());
            return true;
        }

        if (e.target == bOpna)
        {
            tTexti.setText(lesaTextaskra(tSkraarnafn.getText()));
            return true;
        }

        if (e.target == bHaetta)
        {
            this.dispose();
            System.exit(0);
            return true;
        }
        return false;
    }
}
```

Skipanirnar sem eru framkvæmdar þegar ýtt er á `bHaetta` eru skýrðar í kafla 9.d.

Verkefni F.1 *
Búðu til forrit sem skrifar textaskrá sem í stendur ein setning (t.d. „Sísí sá gula sól.“).

Verkefni F.2 *
Búðu til textaskrá sem inniheldur tölur með bili eða kommu á milli. (Þú getur t.d. notað Notepad ritilinn sem fylgir með Windows). Búðu svo til forrit sem les allar tölurnar og reiknar meðaltal af þeim. (Forritið verður að breyta strengjum sem tákna heilar tölur í tölur af gerðinni `int` eða `long`.)

F.b. `FileDialog` og `Frame`

Þegar skrár eru sóttar og vistaðar eru yfirleitt notuð eyðublöð (dialogs) til að velja efnisskrá og skráarnafn. Java forrit geta birt slík eyðublöð til að velja nafn og staðsetningu á skrá. Þau eru af tegundinni `FileDialog`. Forrit_0F_02 sýnir dæmi um notkun klasans `FileDialog`. Það opnar líka ramma (`Frame`) með villumeldingu ef tilraunir til að vista eða sækja skrá kasta fráviki.

Forritið er byggt úr þrem klösum: `Starta`, `Villumelding` og `Textaskrar2`.

`Starta` er næstum eins og samnefndur klasi í forrit_0F_01 og er því ekki birtur hér. `Villumelding` sér um að opna ramma með villuboðum. Klasinn `Textaskrar2` gerir það sama og `Textaskrar1` í forrit_0F_01 en notar `Villumelding` til að bregðast við villum og `FileDialog` til að velja skrár.

Taktu eftir smið klasans `Villumelding`. Hann byrjar á að kalla á smið yfirklasans, `Frame`, velur villumeldingunni svo stærð og stað, setur merki (`Label`) í hana miðja og gerir hana sýnilega.

Til að hægt sé að losna við villumeldingu af skjánum með því að smella á × efst til hægri í rammanum þarf hún að bregðast við atburðum af gerðinni `Event.WINDOW_DESTROY`. Aðferðin `handleEvent` sér um það.

```
import java.awt.*;
//
// forrit_0F_02
// Villumelding
//
public class Villumelding extends Frame
{
   // Rammi sem birtir skilaboð með því að skrifa þau sem Label í sig
   // miðjan.
   public Villumelding(String s)
   {
      super("Villa");
      this.resize(333, 50);
      this.move(100, 100);
      this.add("Center", new Label(s));
      this.show();
   }

   // Þessi aðferð yfirskyggir handleEvent aðferð teg. Component (en
   // Frame erfir frá henni). Aðferðin bregst aðeins við einni gerð
   // atburða sem verður ef smellt er á x-ið efst til hægri í
   // glugganum.
   public boolean handleEvent(Event evt)
   {
      if (evt.id == Event.WINDOW_DESTROY)
      {
         this.hide();
         return true;
      }
      return false;
   }
}
```

Taktu eftir því hvernig smiðurinn `FileDialog` er notaður í klasanum `Textaskrar2`.

```
   fdOpna  = new FileDialog(this, "Veldu skrá til að opna.",
                                       FileDialog.LOAD);
   fdVista = new FileDialog(this, "Veldu skránni stað og heiti.",
                                       FileDialog.SAVE);
```

Hann tekur við þrem gildum, það fyrsta er af tegundinni `Frame`, svo kemur strengur með titillínu eyðublaðsins og þá fasti sem segir hvort á að sækja skrá (`LOAD`) eða vista (`SAVE`).

Aðferðin `action` inniheldur eftirfarandi skipanir til að bregðast við þegar smellt er á hnappinn `bVista`.

```
   if (e.target == bVista)
   {
      fdVista.show();
      if (fdVista.getFile() != null)
      {
         String skrNafn = fdVista.getDirectory() + dVista.getFile();
         skrifaTextaskra(skrNafn, tTexti.getText());
         return true;
      }
      else
      {
         return false;
```

```
      }
   }
```

Fyrst er hluturinn `fdVista`, sem er af gerðinni `FileDialog`, látinn framkvæma aðferðina `show` til að sýna sig. Hann felur sig svo sjálfkrafa aftur um leið og smellt er á OK eða Cancel hnapp. Aðferðirnar `getFile` og `getDirectory` skila strengjum með nafni á skrá og efnisskrá. Hafi engin skrá verið valin er útkoman úr `getFile` `null`. Hér er `action` aðferðin látin skila gildinu `false` ef engin skrá er valin (þ.e. ef `fdVista.getFile()` skilar gildinu `null`.)

Taktu eftir því hvernig klasinn `Villumelding` er notaður. Hlutir af þessari gerð eru búnir til inni í `catch`-blokkunum.

```
import java.awt.*;
import java.io.*;
//
// forrit_0F_02
// Textaskrar2
//
public class Textaskrar2 extends Frame
{
   TextArea tTexti;
   Label  lTexti;
   Button bVista;
   Button bOpna;
   Button bHaetta;

   FileDialog fdOpna, fdVista;

   public Textaskrar2(String titill)
   {
      super(titill);
   }

   public void init()
   {
      FlowLayout fl = new FlowLayout();
      this.setLayout(fl);

      lTexti = new Label("Texti ");
      tTexti = new TextArea(10, 25);
      bVista = new Button("Vista");
      bOpna = new Button("Opna");
      bHaetta = new Button("Hætta");

      this.add(lTexti); this.add(tTexti);
      this.add(bVista); this.add(bOpna);
      this.add(bHaetta);

      // Þessar tegundir eru gluggar til að velja efniskrá og skrá.
      fdOpna  = new FileDialog(this, "Veldu skrá til að opna.",
                                     FileDialog.LOAD);
      fdVista = new FileDialog(this, "Veldu skránni stað og heiti.",
                                     FileDialog.SAVE);
   }

   public String lesaTextaskra(String skrNafn)
   {
      FileInputStream f;
      DataInputStream d;
      String allurTexti = "";
      String einLina;
      try
```

```
      {
         f = new FileInputStream(skrNafn);
         d = new DataInputStream(f);

         // Aðferðin available skilar tölu sem segir hvað
         // mikið er eftir ólesið í straumnum.
         while (d.available() != 0)
         {
            einLina = d.readLine();
            allurTexti = allurTexti + einLina + "\n";
         }

         f.close();
         d.close();
      }
      catch (IOException fravik)
      {
         // Búinn til nýr hlutur af tegundinni Villumelding.
         // Smiður hlutarins sér um að sýna hann og hann sér
         // sjálfur um að fela sig þegar smellt er á x-ið í
         // horninu efst til hægri.
         Villumelding v =
            new Villumelding("Get ekki lesið " + skrNafn);
      }
      return allurTexti;
   }

   public void skrifaTextaskra(String skrNafn, String texti)
   {
      FileOutputStream f;
      PrintStream p;
      try
      {
         f = new FileOutputStream(skrNafn);
         p = new PrintStream(f);
         p.print(texti);
         f.close(); p.close();
      }
```

```
      catch (IOException fravik)
      {
         Villumelding v =
                  new Villumelding("Get ekki skrifað " + skrNafn);
      }
   }


   public boolean action(Event e, Object o)
   {
      if (e.target == bVista)
      {
         // Gluggi til að velja skrá sýndur. Hann sér sjálfur um að
         // fela sig þegar smellt er á OK eða Cancel.
         fdVista.show();
         // Ef smellt var á Cancel þá skilar getFile gildinu null
         // sem merkir að engin skrá hafi verið valin.
         if (fdVista.getFile() != null)
         {
            String skrNafn = fdVista.getDirectory() +
                             fdVista.getFile();
            skrifaTextaskra(skrNafn, tTexti.getText());
            return true;
         }
         else
         {
            return false;
         }
      }

      if (e.target == bOpna)
      {
         // Gluggi til að velja skrá sýndur. Hann sér sjálfur um að
         // fela sig þegar smellt er á OK eða Cancel.
         fdOpna.show();
         // Ef smellt er á Cancel þá skilar getFile gildinu null
         // sem merkir að engin skrá hafi verið valin.
         if (fdOpna.getFile() != null)
         {
            String skrNafn = fdOpna.getDirectory() +
                             fdOpna.getFile();
            tTexti.setText(lesaTextaskra(skrNafn));
            return true;
         }
         else
         {
            return false;
         }

      }

      if (e.target == bHaetta)
      {
         this.dispose();
         System.exit(0);
         return true;
      }
      return false;
   }
}
```

Verkefni F.3 *
Keyrðu forrit_0F_02 og reyndu að láta það gera eitthvað ómögulegt eins og að sækja skrá sem er ekki til eða skrifa á ritvarinn diskling. Taktu eftir villuboðunum sem forritið birtir og reyndu að skilja hvernig klasinn `Villumelding` er notaður.

Verkefni F.4 *
Breyttu lausnunum á verkefnum F.1 og F.2 þannig að forritin noti klasana `FileDialog` og `Villumelding` á svipaðan hátt og gert er í forrit_0F_02.

Verkefni F.5
Bættu við lausnina á verkefnum D.5 og D.6 þannig að reiknivélin geti sótt og vistað talnasafn sem geymt er í skrá.

Forrit_0F_01 og forrit_0F_02 lesa texta línu fyrir línu (heilar stafarunur sem enda á línuskiptum, þ.e. tákninu '\n') og skrifa einn langan streng (sem getur innihaldið mörg línuskipti). Stundum er betra að lesa og skrifa texta bæti fyrir bæti (þ.e. einn staf í senn). Það er gert í forrit_0F_03 sem les texta og breytir honum í dulmál með því að setja b í stað a, c í stað b, d í stað c o.s.frv.

Skráin er lesin sem tölur en ekki sem stafir og hver tala hækkuð um einn ef verið er að breyta skiljanlegri skrá í dulmál og lækkuð um 1 ef verið er að breyta dulmálsskrá í skiljanlega skrá.

Klasarnir `Villumelding` og `Starta` eru eins og í forrit_0F_02.

Forritið geymir upplýsingar um hvaða efnisskrá var síðast sótt úr eða skrifað í og opnar þá sömu efnisskrá næst þegar `FileDialog` er sýndur. Það gefur skrám líka sjálfkrafa viðeigandi nafnauka. Dulmálsskrár fá nafnaukann .dul og skiljanlegar skrár fá nafnaukann .ski.

```
import java.awt.*;
import java.io.*;
//
// forrit_0F_03
// Textaskrar3
//
public class Textaskrar3 extends Frame
{
   TextArea tTexti;
   Label lTexti;
   Button bADulmal;
   Button bAfDulmali;
   Button bHaetta;

   String efnisskra;    // Geymir heiti þeirrar efnisskrár sem skrá
                        // var sótt úr síðast. Er upphaflega null.
   FileDialog fdADulmalOpna, fdAfDulmaliOpna;

   // Smiður
   public Textaskrar3(String titill)
   {
      super(titill);
   }
```

```
public void init()
{
   FlowLayout fl = new FlowLayout();
   this.setLayout(fl);

   lTexti = new Label("Texti");
   tTexti = new TextArea(12, 30);
   bADulmal = new Button("Snara á dulmál");
   bAfDulmali = new Button("Snara af dulmáli");
   bHaetta = new Button("Hætta");

   this.add(lTexti); this.add(tTexti);
   this.add(bADulmal); this.add(bAfDulmali);
   this.add(bHaetta);

   fdADulmalOpna  = new FileDialog(this,
               "Hvaða skrá á að þýða á dulmál?", FileDialog.LOAD);
   fdAfDulmaliOpna  = new FileDialog(this,
               "Hvaða dulmálsskrá á að þýða?", FileDialog.LOAD);
}

public void snaraADulmal(String dir, String skr)
{
   FileOutputStream fSkrif;   // Skrá sem skrifuð er.
   DataOutputStream dSkrif;   // Gaganstraumur í fSkrif.
   FileInputStream fLes;      // Skrá sem lesið er úr.
   DataInputStream dLes;      // Gagnastraumur úr fLes.
   int eittByte;

   String skrNafn, nyttSkrNafn;

   // Valið nafn á skrána sem skrifa skal í.
   // Hún fær sama fornafn og skráin sem lesið er
   // úr en nafnaukann .dul.
   skrNafn = dir + skr;
   int i = skr.indexOf(".");
   if (i == -1)   // Ef skrá sem lesið er úr hefur engan
   {              // nafnauka er dul einfaldlega bætt við
      nyttSkrNafn = dir + skr + ".dul"; // heiti hennar.
   }
   else
   {
      nyttSkrNafn = dir + skr.substring(0, i) + ".dul";
   }

   try
   {
      fLes = new FileInputStream(skrNafn);
      dLes = new DataInputStream(fLes);
      fSkrif = new FileOutputStream(nyttSkrNafn);
      dSkrif = new DataOutputStream(fSkrif);

      // Meðan eitthvað er enn ólesið úr dLes
      // (gagnastraumnum sem rennur í úr fLes)
      // er lesið eitt bæti og skrifað í dSkrif.
      while (dLes.available() > 0)
      {
         // Hér er lesið úr dLes
         eittByte = dLes.readUnsignedByte();
```

```
            // Hér er bætinu sem lesið var breytt
            // með því að hækka gildið um 1.
            if (eittByte == 255)
            {
               eittByte = 0;
            }
            else
            {
               eittByte += 1;
            }

            // Hér er bætið skrifað í dSkrif.
            dSkrif.writeByte(eittByte);

            // Hér er bætinu breytt í eins stafs streng og
            // honum bætt í textareit. (Ath. aðferðin
            // valueOf í klasanum String er static.)
            tTexti.appendText(String.valueOf((char)eittByte));
         }

         // Skrúfað fyrir strauma.
         fLes.close();
         dLes.close();
         fSkrif.close();
         dSkrif.close();
      }

      catch (IOException fravik)
      {
         Villumelding v =
            new Villumelding("Get ekki snarað " + skrNafn);
      }
   }

   // Næsta aðferð er næstum því eins og snaraADulmal.
   public void snaraAfDulmali(String dir, String skr)
   {
      FileOutputStream fSkrif;
      DataOutputStream dSkrif;
      FileInputStream fLes;
      DataInputStream dLes;
      int eittByte;
      String skrNafn, nyttSkrNafn;

      skrNafn = dir + skr;
      int i = skr.indexOf(".");  // Skiljanlg skrá
      if (i == -1)               // fær nafnaukann
      {                          // .ski
         nyttSkrNafn = dir + skr + ".ski";
      }
      else
      {
         nyttSkrNafn = dir + skr.substring(0, i) + ".ski";
      }

      try
      {
         fLes = new FileInputStream(skrNafn);
         dLes = new DataInputStream(fLes);
         fSkrif = new FileOutputStream(nyttSkrNafn);
         dSkrif = new DataOutputStream(fSkrif);

         while (dLes.available() > 0)
         {
            eittByte = dLes.readUnsignedByte();
            if (eittByte == 0)
```

```
            {
               eittByte = 255;
            }
            else
            {
               eittByte -= 1;
            }
            dSkrif.writeByte(eittByte);
            tTexti.appendText(String.valueOf((char)eittByte));
         }

         fLes.close();
         dLes.close();
         fSkrif.close();
         dSkrif.close();
      }

      catch (IOException fravik)
      {
         Villumelding v =
            new Villumelding("Get ekki snarað " + skrNafn);
      }
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bADulmal)
      {
         if (efnisskra != null)  // Ef breytan efnisskra hefur
         {                       // fengið gildi.
            fdADulmalOpna.setDirectory(efnisskra);
         }
         fdADulmalOpna.show();
         if (fdADulmalOpna.getFile() != null)   // Ef einhver
         {                                      // skrá var valin
            tTexti.setText("");
            efnisskra = fdADulmalOpna.getDirectory();
            snaraADulmal(fdADulmalOpna.getDirectory(),
                         fdADulmalOpna.getFile());
            return true;
         }
         else
         {
            return false;
         }
      }

      if (e.target == bAfDulmali)
      {
         if (efnisskra != null)
         {
            fdAfDulmaliOpna.setDirectory(efnisskra);
         }
         fdAfDulmaliOpna.show();
```

```
            if (fdAfDulmaliOpna.getFile() != null)
            {
               tTexti.setText("");
               efnisskra = fdAfDulmaliOpna.getDirectory();
               snaraAfDulmali(fdAfDulmaliOpna.getDirectory(),
                                    fdAfDulmaliOpna.getFile());
               return true;
            }
            else
            {
               return false;
            }
         }

         if (e.target == bHaetta)
         {
            this.dispose();
            System.exit(0);
            return true;
         }
         return false;
      }
}
```

Verkefni F.6
Dulmálið sem forrit_0F_03 skrifar er lélegt og það er enginn vandi að ráða það. Hægt er að búa til ögn fullkomnara dulmál með því að velja tölu fyrir dulmálslykil og beita aðgerðinni XOR sem rituð er `^` á hana og hvert bæti. Sé hvert bæti t.d. lesið í breytu sem heitir `x` og talan 99 notuð fyrir dulmálslykil þá er dulmálskóðinn fyrir x einfaldlega `x ^ 99`. Til að þýða af dulmáli er sama aðgerð endurtekin (sbr. það sem segir um aðgerðina XOR í kafla A.d). Breyttu forrit_0F_03 þannig að það hafi textareit sem tekur við dulmálslykli og noti XOR eins og hér var lýst til að þýða á dulmál og af því.

Verkefni F.7
Það er ekki erfitt að ráða dulmál af því tagi sem notað er í verkefni F.6. Raunar dugar að nýta tölfræðilegar upplýsingar um tíðni stafa til að finna út hvaða tákn í dulmálinu samsvarar hverjum staf í venjulegum texta. Reyndu að endurbæta forrit_0F_03 þannig að ekki sé hægt að ráða dulmálið með svona einfaldri aðferð.

Verkefni F.8
Búðu til forrit sem les texta á íslensku og þýðir hann á ísl-ensku með því að setja heiti stafa í stað séríslensku stafanna. Þau nöfn sem oftast eru notuð eru í töflunni hér fyrir neðan.

á	á	Á	Á	ú	ú	Ú	Ú
ð	ð	Ð	Ð	ý	ý	Ý	Ý
é	é	É	É	þ	þ	Þ	Þ
í	í	Í	Í	æ	æ	Æ	Æ
ó	ó	Ó	Ó	ö	ö	Ö	Ö

F.c. Texti og annars konar gögn

Eins og fram kemur í forrit_0F_03 er hægt að lesa skrá sem tölur (`int`) þótt hún innihaldi texta á íslensku. En skrár geta innihaldið fleira en texta, t.d. kommutölur af gerðinni `double` eða heiltölur af gerðinni `int`.

Í Java taka tölur af gerðinni `int` 4 bæti svo ef talan 357892 er skrifuð sem `int` tala í skrá þá tekur hún 4 bæta pláss á diskinum. En ef strengurinn "357892" er skrifaður þá tekur hann 6 bæta pláss, eitt bæti fyrir hvern staf, ef hann er skrifaður með `print`-aðferð klasans `PrintStream`.

Svona líta talan 357892 og strengurinn "357892" út á formi tvíundakerfis.

Talan 357892	00000000 00000101 01110110 00000100
Strengurinn "357892"	00110011 00110101 00110111 00111000 00111001 00110010

Eigi að geyma margar heilar tölur í skrá sparast rúm með með að geyma þær sem `int` fremur en strengi og forrit er líka mun fljótara að vinna með þær (t.d. að leggja þær saman) ef hægt er að lesa þær beint inn í `int` breytur í stað þess að lesa þær fyrst sem strengi og breyta þeim svo í `int`.

Forrit_0F_04 breytir skrá með texta sem inniheldur nöfn heiltalna í skrá af tölum af gerðinni `int` og öfugt. Forritið er byggt úr þrem klösum: `Starta`, `Villumelding` og `TextaOgTalnaskrar1`. Þeir tveir fyrstnefndu eru eins og í forrit_0F_02 og eru því ekki birtir hér.

Líkt og `Textaskrar3` í forrit_0F_03 velur klasinn `TextaOgTalnaskrar1` sjálfkrafa eftirnöfn á skrár. En hér er komin sérstök aðferð til þess arna sem heitir `finnaNyttSkrNafn`. Skoðaðu hana vel og áttaðu þig á hvernig hún virkar.

Til að skrifa tölur á `int` formi er `writeInt` aðferð klasans `DataOutputStream` notuð.

```
import java.awt.*;
import java.io.*;
import java.util.StringTokenizer;
//
// forrit_0F_04
// TextaOgTalnaskrar1
//
// Byggt á klasanum Textaskrar3 í forrit_0F_03
//
public class TextaOgTalnaskrar1 extends Frame
{
   Button bText2Int;
   Button bInt2Text;
   Button bHaetta;
   String efnisskra;    // Geymir heiti þeirrar efnisskrár sem skrá
                        // var sótt úr síðast. Er upphaflega  null.
   FileDialog fdText2IntOpna, fdInt2TextOpna;

   public TextaOgTalnaskrar1(String titill)
   {
      super(titill);
   }
```

```
public void init()
{
   FlowLayout fl = new FlowLayout();
   this.setLayout(fl);

   bText2Int = new Button("Breyta texta í tölur.");
   bInt2Text = new Button("Breyta tölum í texta.");
   bHaetta = new Button("Hætta");
   this.add(bText2Int); this.add(bInt2Text); this.add(bHaetta);

   fdText2IntOpna  = new FileDialog(this,
      "Hvaða textaskrá á að breyta í talnaskrá?",
      FileDialog.LOAD);
   fdInt2TextOpna  = new FileDialog(this,
      "Hvaða talnaskrá á að breyta í textaskrá?",
      FileDialog.LOAD);
}

public String finnaNyttSkrNafn(String dir,
                               String skr, String nyrNafnauki)
{
   String nyttSkrNafn;
   int i = skr.indexOf(".");
   if (i == -1)    // Ef skrá sem lesið er úr hefur engan
   {               // nafnauka er nýjum Nafnauka einfald-
                   // lega bætt við heiti hennar.
      nyttSkrNafn = dir + skr + nyrNafnauki;
   }
   else
   {
      nyttSkrNafn = dir + skr.substring(0, i) + nyrNafnauki;
   }
   return nyttSkrNafn;
}


// Þessi aðferð les textaskrá sem inniheldur nöfn á heilum
// tölum og breytir henni í skrá af int-tölum.
public void text2int(String dir, String skr)
{
   FileOutputStream fSkrif;      // Skrá sem skrifuð er.
   DataOutputStream dSkrif;      // Gaganstraumur í fSkrif.
   FileInputStream fLes;         // Skrá sem lesið er úr.
   DataInputStream dLes;         // Gagnastraumur úr fLes.

   StringTokenizer st;
   String biltakn = " ,\t\n\r";  // Tákn sem geta aðgreint
   String sTala;                 // tvær tölur í textaskránni
   Integer Tala;                 // eru bil, komma, \t, \n, og \r.
   String einLina;
   String skrNafn, nyttSkrNafn;
   skrNafn = dir + skr;
   nyttSkrNafn = finnaNyttSkrNafn(dir, skr, ".int");

   try
   {
      fLes = new FileInputStream(skrNafn);
      dLes = new DataInputStream(fLes);
      fSkrif = new FileOutputStream(nyttSkrNafn);
      dSkrif = new DataOutputStream(fSkrif);

      // Meðan eitthvað er enn ólesið úr dLes
      // (gagnastraumnum sem rennur í úr fLes)
      // er lesin ein lína, stengnum breytt í
      // heiltölur og skrifað í dSkrif.
      while (dLes.available() > 0)
```

```
      {
         // Hér er ein lína lesin úr dLes
         einLina = dLes.readLine();
         st = new StringTokenizer(einLina, biltakn);

         // Hér er línunni breytt í heiltölur
         // af gerðinni int og þær skrifaðar í dSkrif.
         while (st.hasMoreTokens())
         {
            sTala = new String(st.nextToken());
            try
            {
               Tala = new Integer(sTala);
               dSkrif.writeInt(Tala.intValue());
            }
            catch (NumberFormatException nfFravik)
            {
               Villumelding v =
                  new Villumelding(sTala + " er ekki tala.");
            }
         }  // while ((st.hasMoreTokens()) endar
      } // while (dLes.available() > 0) endar

      // Skrúfað fyrir strauma.
      fLes.close();
      dLes.close();
      fSkrif.close();
      dSkrif.close();
   } // try blokk endar

   catch (IOException fravik)
   {
      Villumelding v =
         new Villumelding("Get ekki snarað " + skrNafn);
   }
}


// Þessi aðferð les skrá með heiltölum af gerðinni int
// og breytir henni í textaskrá þar sem tölurnar eru
// ritaðar með kommu og bili á milli.
public void int2text(String dir, String skr)
{
   FileOutputStream fSkrif;
   PrintStream pSkrif;        // PrintStream notað til að
   FileInputStream fLes;      // skrifa textaskrá.
   DataInputStream dLes;
   int einTala;
   String skrNafn, nyttSkrNafn;

   skrNafn = dir + skr;
   nyttSkrNafn = finnaNyttSkrNafn(dir, skr, ".txt");
```

```
      try
      {
         fLes = new FileInputStream(skrNafn);
         dLes = new DataInputStream(fLes);
         fSkrif = new FileOutputStream(nyttSkrNafn);
         pSkrif = new PrintStream(fSkrif);

         while (dLes.available() > 0)
         {
            einTala = dLes.readInt();
            pSkrif.print(einTala + ", ");
         }

         fLes.close();
         dLes.close();
         fSkrif.close();
         pSkrif.close();
      }

      catch (IOException fravik)
      {
         Villumelding v =
            new Villumelding("Get ekki snarað " + skrNafn);
      }
   }


   public boolean action(Event e, Object o)
   {
      if (e.target == bText2Int)
      {
         if (efnisskra != null)  // Ef breytan efnisskra hefur
         {                       // fengið gildi.
            fdText2IntOpna.setDirectory(efnisskra);
         }

         fdText2IntOpna.show();

         if (fdText2IntOpna.getFile() != null)  // Ef einhver
         {                                      // skrá var valin
            efnisskra = fdText2IntOpna.getDirectory();
            text2int(fdText2IntOpna.getDirectory(),
                        fdText2IntOpna.getFile());
            return true;
         }
         else
         {
            return false;
         }
      }

      if (e.target == bInt2Text)
      {
         if (efnisskra != null)
         {
            fdInt2TextOpna.setDirectory(efnisskra);
         }

         fdInt2TextOpna.show();
```

```
         if (fdInt2TextOpna.getFile() != null)
         {
            efnisskra = fdInt2TextOpna.getDirectory();
            int2text(fdInt2TextOpna.getDirectory(),
                        fdInt2TextOpna.getFile());
            return true;
         }
         else
         {
            return false;
         }
      }

      if (e.target == bHaetta)
      {
         this.dispose();
         System.exit(0);
         return true;
      }
      return false;
   } // Hér endar action aðferðin.
}
```

Verkefni F.9

Búðu til forrit sem les skrá af heiltölum, túlkar hvert talnapar sem hnit eins punkts og dregur strik á milli þeirra. Prófaðu svo að láta forritið lesa skrána skrimsli.int sem til verður þegar forrit_0F_04 er notað til að breyta skránni skrimsli.txt sem fylgir því.

Til að hægt sé að túlka skrá með heiltölum sem skrá af punktum þarf fjöldi talna í henni að vera slétt tala. Láttu forritið bregðast við með skynsamlegri athugasemd ef fjöldi talna í skránni er oddatala.

Skipanirnar sem lesa tölurnar og draga strikin geta verið eitthvað á þessa leið:

```
      int x1, y1, x2, y2;
      Graphics g = this.getGraphics();
      try
      {
         fLes = new FileInputStream(skrNafn);
         dLes = new DataInputStream(fLes);
         x1 = dLes.readInt();
         y1 = dLes.readInt();
         while (dLes.available() > 0)
         {
            x2 = dLes.readInt();
            y2 = dLes.readInt();
            g.drawLine(x1, y1, x2, y2);
            x1 = x2;
            y1 = y2;
         }
         fLes.close();
         dLes.close();
      }

      // Hér komi catch setningar
```

F.d. Internet og URL

Úttaksstraumur þarf ekki að lenda á diski. Hann getur allt eins farið til prentara eða út um nettengi. Inntaksstraumur þarf heldur ekki að eiga upptök á diski. Hann getur til dæmis komið frá lyklaborði eða annarri tölvu sem samband er við gegnum Internetið.

Forrit_0F_05 les skrá með tölum af Internetinu. Það er byggt úr klösunum `Starta`, `Villumelding` og `TolurAfNeti`. Þeir tveir fyrstnefndu er óbreyttir frá forrit_0F_02.

Klasinn `TolurAfNeti` notar klasann `URL` úr pakkanum `java.net`. Smiður klasans `URL` tekur við streng með veffangi skráar (t.d. "http://www.ismennt.is/not/atli/data/tolur.int"). `URL` getur svo beitt aðferðinni `openStream` til að opna straum úr skránni sem það vísar á. Þetta virkar að sjálfsögðu ekki nema tölvan sé í sambandi við Internetið.

Eigi til dæmis að láta hlut af gerðinni `DataInputStream` lesa eina tölu úr skránni http://www.ismennt.is/not/atli/data/tolur.int er hægt að nota skipanirnar

```
URL veffang;
DataInputStream d;
int x;
try
{
   veffang =
      new URL("http://www.ismennt.is/not/atli/data/tolur.int");
}
catch (MalformedURLException fravik)
{
   // Það sem gera skal ef ekki tekst að mynda url úr strengnum.
}
try
{
   // Aðferðin openStream opnar inntaksstraum frá veffangi
   // d er stillt á að taka við úr honum.
   d = new DataInputStream(veffang.openStream());
   x = d.readInt();
   d.close();
}
catch (IOException fravik)
{
   // Það sem gera skal ef lesturinn mistekst.
}
```

Hér kemur svo klasinn `TolurAfNeti`.

```
import java.awt.*;
import java.io.*;
import java.net.*;
//
// forrit_0F_05
// TolurAfNeti
//
public class TolurAfNeti extends Frame
{
   TextField tURL;
   TextArea tTolur;
   URL veffang;
   Button bSaekja;
   Button bHaetta;

   public TolurAfNeti(String titill)
   {
      super(titill);
   }
```

```
public void init()
{
   FlowLayout f = new FlowLayout();
   this.setLayout(f);
   bSaekja = new Button("Sækja tölur");
   bHaetta = new Button("Hætta");
   tURL = new TextField(28);
   tTolur = new TextArea(15, 10);

   this.add(tURL);
   this.add(bSaekja);
   this.add(bHaetta);
   this.add(tTolur);
}

// Þessi aðferð les skrá með heiltölum af gerðinni int
// úr skrá sem sótt er yfir Internetið.
public void lesaTolurAfNeti(URL u)
{
   DataInputStream dLes;
   try
   {
      // Aðferðin openStream í klasanum URL býr til
      // straum úr veffangi.
      dLes = new DataInputStream(u.openStream());

      while (dLes.available() > 0)
      {
         tTolur.appendText(dLes.readInt() + "\n");
      }
      dLes.close();
   }

   catch (IOException fravik)
   {
      Villumelding v = new Villumelding(fravik.toString());
   }
   catch (Exception fravik)
   {
      Villumelding v = new Villumelding(fravik.toString());
   }
}

public boolean action(Event e, Object o)
{
   if (e.target == bSaekja)
   {
      try
      {
         veffang = new URL(tURL.getText());
         lesaTolurAfNeti(veffang);
      }
      catch (MalformedURLException fravik)
      {
         Villumelding v = new Villumelding("Rangt myndað URL");
      }
      catch (Exception fravik)
      {
         Villumelding v = new Villumelding("Óþekkt villa!");
      }
   }

   if (e.target == bHaetta)
   {
      this.dispose();
```

```
            System.exit(0);
            return true;
        }

        return false;
    } // Hér endar action aðferðin.
}
```

Verkefni F.10 *
Keyrðu forrit_0F_05 og láttu það sækja tölur úr skránni:
http://www.ismennt.is/not/atli/data/tolur.int Hún inniheldur tölurnar: 25, 25, 175, 25, 175, 175, 25, 175, 25, 25, 175, 175. Komdu svo skrá með tölum fyrir einhvers staðar á Internetinu og láttu forritið lesa hana. (Þú getur notað forrit_0F_04 til að búa til talnaskrá úr textaskrá sem inniheldur nöfn á tölum.)

Verkefni F.11 *
Búðu til forrit sem les texta úr skrá af Internetinu. Prófaðu svo að láta það sækja skrána http://www.ismennt.is/not/atli/data/visa.txt

Verkefni F.12
Breyttu forrit_0F_05 þannig að það skrifi tölurnar tvær og tvær saman innan sviga með kommu á milli eins og vani er að skrifa hnit í tvívíðu hnitakerfi, svona:

(25, 25) (175, 25) (175, 175) o.s.frv.

Verkefni F.13
Breyttu lausninni á verkefni F.9 þannig að forritið geti sótt tölur til að teikna eftir af Internetinu. Prófaðu svo að láta forritið sækja http://www.ismennt.is/not/atli/data/tolur.int. Myndin sem birtist ætti að vera eins og ferningur með hornalínu. Þú getur líka sótt tölur úr skránni http://www.ismennt.is/not/atli/data/skrimsli.int.

Verkefni F.14
Bættu við lausnina á verkefni F.5 þannig að reiknivélin geti sótt textakrá sem inniheldur tölur af Internetinu.

F.x. Spurningar og umhugsunarefni

1. Hverjir af þeim klösum sem taldir eru upp hér fyrir neðan geta opnað skrá til að lesa úr, hverjir geta opnað skrá til að skrifa í og hverjir geta hvorugt?

   `DataInputStream`, `DataOutputStream`, `FileInputStream`, `FileOutputStream`, `PrintStream`.

2. Af hverju eru forritin í þessum kafla sjálfstæð forrit en ekki `Applet`?
3. Hvaða munur er á klösunum `PrintStream` og `DataOutputStream`?
4. Til hvers er klasinn `FileDialog` og hvað gera aðferðirnar `getDirectory` og `setDirectory` sem tilheyra honum?
5. Hlutir af gerðinni `DataInputStream` geta beitt aðferð sem heitir `available`. Hvað gerir hún?
6. Hvaða munur er á skrá sem geymir heiltöluna 100 og skrá sem geymir strenginn "100"?
7. Hvaða munur er á skrá sem geymir 20 tölur og skrá sem geymir hnit 10 punkta í tvívíðu hnitakerfi?
8. Hvort tekur meira rúm á diski 100 heiltölur milli 0 og 99, sem geymdar eru sem `int`, eða strengur sem inniheldur 100 tveggja stafa tölur og eitt orðabil á milli hverra tveggja samliggjandi talna?
9. Hvort tekur meira rúm á diski 100 heiltölur milli -10^9 og 10^9, sem geymdar eru sem `int`, eða strengur sem inniheldur 100 níu stafa tölur og eitt orðabil á milli hverra tveggja samliggjandi talna?
10. Hvaða hlutverk hefur klasinn `URL` og hvaða pakka tilheyrir hann?
11. Inntaksstraumur þarf ekki endilega að eiga upptök sín í skrá á gagnageymslu tölvunnar sem forritið er keyrt á. Nefndu tvö önnur möguleg upptök.

Til umhugsunar

Ef þú hefur leyst verkefni F.9 þá hefur þú búið til forrit sem túlkar runur af heiltölum sem teikningu úr beinum strikum.

Í vissum skilningi eru öll gögn sem tölvur vinna með runur af heilum tölum sem skrifaðar eru í tvíundakerfi. Forritin túlka þessar talnarunur svo á mismunandi vegu.

Hvernig ætli sé hægt að láta forrit túlka runu af heilum tölum sem:

a) Mynd í mörgum litum og með alls konar form (t.d. ljósmynd af manni)?
b) Texta (t.d. sögu eða ljóð)?
c) Hljóð (t.d. lag eða upplestur)?
d) Kvikmynd?

Ætli það sé mögulegt að búa til talnarunu sem eitt forrit túlkar sem lag og annað forrit sem mynd?

Ætli það sé mögulegt að búa til talnarunu sem eitt forrit túlkar sem texta (runu af bókstöfum) og annað forrit sem upplestur (hljóð) á sama texta?

10. kafli: Safnklasar og gagnagrindur

10.a. Strengir, `StringBuffer` og fylki

Safnklasi er klasi sem getur geymt mikið magn upplýsinga. Gagnagrind er breyta eða kerfi af breytum, sem hægt er að geyma í mikið magn af upplýsingum. Sumar gagnagrindur, eins og t.d. straumar, strengir og `StringTokenizer` eru safnklasar. En sumar gagnagrindur, eins og fylki, er álitamál hvort rétt er að kalla klasa.

Hægt er að geyma sömu gögn á marga ólíka vegu. Runu af bókstöfum er t.d. hægt að geyma í fylki af `char`, í streng, `StringTokenizer` eða straumi (t.d. `PrintStream`). Hvaða geymsluaðferð er heppilegust veltur á því hvað á að gera við stafarununa. Eigi að skrifa hana í skrá er heppilegt að nota `PrintStream`. Þurfi að klippa hana sundur í stök orð er gott að nota `StringTokenizer`. Eigi að breyta henni, t.d. þannig að alls staðar þar sem 'z' kemur fyrir sé sett 's' í staðinn, hentar að nota fylki af `char`.

Hvað hægt er, með sæmilega auðveldu móti, að gera við gögn veltur á því á hvaða formi þau eru geymd.

Forrit_10_01 hefur þrjár aðferðir til að breyta stafarunu. Þær heita `lagaStafsetningu1`, `lagaStafsetningu2` og `lagaStafsetningu3`. Þær taka allar við streng og skila honum breyttum. Sú fyrsta setur 'z' í stað 's' og 'y' í stað 'i' þannig að sé henni sendur strengurinn "Siggi og Pétur" skilar hún strengnum "Zyggy og Pétur". Hinar tvær bæta stafsetninguna enn meira með því að setja "je" í stað 'é'. Ef þær fá sendan strenginn "Siggi og Pétur" skila þær strengnum "Zyggy og Pjetur".

Í Java eru strengir óbreytanlegir. Það er að vísu hægt að gefa strengjabreytu nýtt gildi en þá er ekki verið að breyta strengnum sem fyrir er í henni heldur henda honum og setja nýjan í staðinn. Séu til dæmis gefnar skipanirnar

```
String s = "Siggi og Pétur";
s = "Z" + s.substring(1,5);
```

þá fær `s` að vísu fyrst gildið "Siggi og Pétur" og síðan gildið "Ziggi" en strengnum "Siggi og Pétur" er ekki breytt heldur er búinn til nýr strengur með því að tengja saman "Z" og "iggi" og þessi nýi strengur settur í `s` og því sem fyrir var (nefnilega strengnum "Siggi og Pétur") hent.

Eigi að breyta streng er heppilegast að færa stafarununa í honum fyrst á annað form. Aðferðin `lagaStafsetningu1` breytir strengnum í fylki af `char`. Hinar tvær breyta honum í hlut af tegundinni `StringBuffer`. Sú tegund er mjög svipuð tegundinni `String` en hefur það fram yfir að hægt er að breyta stafarununum sem hún geymir.

Skoðaðu aðferðina `lagaStafsetningu1`. Hún breytir streng í fylki og fer svo gegnum fylkið og setur 'y' alls staðar það sem er 'i' og 'Y' alls staðar þar sem er 'I' o.s.frv. Þessi aðferð er ágæt ef aðeins þarf að breyta einstökum stöfum í einstaka stafi en hún hentar ekki til að breyta 'é' í "je" því til þess þarf að setja tvo stafi í stað eins. Ætti að gera þetta við fylki þyrfti að láta það hafa nokkur auð hólf aftast til að byrja með og ýta restinni af stöfunum um eitt hólf til hægri eftir fylkinu í hvert sinn sem 'é' kemur fyrir.

`StringBuffer` hefur það fram yfir fylki af `char` að auðvelt er að bæta stöfum inn í stafarunu eins og gera þarf til að breyta "Pétur" í "Pjetur". Þetta er notað í aðferðinni `lagaStafsetningu2`.

Skipunin

```
StringBuffer sb = new StringBuffer(s);
```

breytir strengnum `s` í `StringBuffer` með sama innihaldi. Þegar stafirnir eru komnir í `StringBuffer` er hægt að breyta einstökum stöfum og bæta nýjum stöfum inn í. Skipanirnar

```
a = "Siggi og Pétur";
StringBuffer sb = new StringBuffer(a);
sb.setCharAt(1, 'y');
sb.insert(2, 'w');
a = new String(sb);
```

verða til þess að a fær gildið "Sywggi og Pétur".

Í stað skipunarinnar

```
a = new String(sb);
```

hefði mátt nota

```
a = sb.toString();
```

Þriðja aðferðin til að laga stafsetningu notar aðferð sem heitir `skipta` og sú aðferð breytir streng fyrst í `StringBuffer` og skiptir svo út einni stafarunu fyrir aðra. Aðferðin skipta er gott dæmi um verk sem er þokkalega auðvelt að vinna með því að geyma stafrunu í `StringBuffer` en erfitt ef hún er geymd á öðru formi, t.d. sem `String` eða fylki af `char`.

```
import java.applet.Applet;
import java.awt.*;
//
// Forrit_10_01
// Strengjavinnsla
//
public class Strengjavinnsla extends Applet
{
      Button bStafsetn;
      TextField t1;

      public void init()
      {
         t1 = new TextField(25);
         bStafsetn = new Button("Laga stafsetningu");
         this.add(t1);
         this.add(bStafsetn);
      }

      public String lagaStafsetningu1(String s)
      {
         char[] c;
         // Stafirnir í strengnum c settir í fylki  af char
         c = s.toCharArray();

         // Farið í gegnum fylkið og 'i' breytt í 'y' og 's' í 'z'.
         for (int i = 0; i < c.length; i++)
         {
            if (c[i] == 'i') { c[i] = 'y'; }
            else if (c[i] == 'I') { c[i] = 'Y'; }
            else if (c[i] == 's') { c[i] = 'z'; }
            else if (c[i] == 'S') { c[i] = 'Z'; }
         }
         // Fylkinu breytt í streng og honum skilað.
```

```
        return new String(c);
    }

    public String lagaStafsetningu2(String s)
    {
        StringBuffer sb = new StringBuffer(s);

        // Farið í gegnum sb og 'é' breytt
        // í 'je', 'i' í 'y' og 's' í 'z'.
        for(int i = 0; i < sb.length(); i++)
        {
            if (sb.charAt(i) == 'i') { sb.setCharAt(i, 'y'); }
            else if (sb.charAt(i) == 'I') { sb.setCharAt(i, 'Y'); }
            else if (sb.charAt(i) == 's') { sb.setCharAt(i, 'z'); }
            else if (sb.charAt(i) == 'S') { sb.setCharAt(i, 'Z'); }
            else if (sb.charAt(i) == 'é')
            {
                  sb.setCharAt(i, 'j');
                  sb.insert(i+1, 'é');
            }
            else if (sb.charAt(i) == 'É')
            {
                  sb.setCharAt(i, 'J');
                  sb.insert(i+1, 'e');
            }
        }
        return sb.toString();
    }

    public String lagaStafsetningu3(String s)
    {
        s = skipta(s, "i", "y");
        s = skipta(s, "I", "Y");
        s = skipta(s, "s", "z");
        s = skipta(s, "S", "Z");
        s = skipta(s, "é", "je");
        s = skipta(s, "É", "Je");
        return s;
    }

    public String skipta(String s, String ut, String inn)
    {
        // ut er stafarunan sem á að hverfa og inn er runan
        // sem á að koma í staðinn.

        // b vísar á upphaf strengins s, þ.e. staf númer 0.
        int b = 0;

        // e vísar á fyrsta staðinn í s þar sem ut kemur fyrir.
        int e = s.indexOf(ut, 0);

        // hér er búinn til tómur StringBuffer.
        StringBuffer st = new StringBuffer();

        // Ef ut kemur ekki fyrir oftar fær e gildið -1. Slaufan er
        // endurtekin meðan ut kemur fyrir í s.
```

```
            while (e != -1)
            {
               // Við st er bætt öllum stöfum í s frá númer b að
               // númer e og síðan strengum inn.
               st.append(s.substring(b, e));
               st.append(inn);

               // b látið vísa á næsta staf aftan við síðasta
               // tilvik inn.
               b = e + (ut.length());
               // e látið vísa á fyrsta staf í næsta tilviki af inn.
               e = s.indexOf(ut, b);
            }
            st.append(s.substring(b, s.length()));
            return st.toString();
         }

         public boolean action(Event e, Object o)
         {
            // Hér er aðferðin lagaStafsetningu3 notuð. Eigi að  nota
            // lagaStafsetningu1 eða lagaStafsetningu2 verður að breyta:
            // t1.setText(lagaStafsetningu3(t1.getText()));
            // í
            // t1.setText(lagaStafsetningu1(t1.getText()));
            // eða
            // t1.setText(lagaStafsetningu2(t1.getText()));
            //
            if (e.target == bStafsetn)
            {
               t1.setText(lagaStafsetningu3(t1.getText()));
               return true;
            }
            return false;
         }
}
```

Verkefni 10.1*
Búðu til forrit sem tekur við streng og breytir honum þannig að alls staðar þar sem "kr." kemur fyrir komi "IKR" í staðinn og alls staðar þar sem "$" kemur fyrir komi "USD" í staðinn.

Verkefni 10.2*
Búðu til forrit sem tekur við einu mannsnafni og skammstafar millinöfn ef nafnið er alls meira en 30 stafir á lengd. Sé t.d. slegið inni nafnið "Bergmundur Katarínus Engilráðsson" (alls 33 stafir með orðabilum) á forritið að skrifa "Bergmundur K. Engilráðsson". Láttu sérstaka aðferð sjá um skammstafanirnar. Hún á að taka við streng með nafni og skila sama streng ef hann er minna en 30 stafir eða inniheldur færri en 3 orð en streng með skammstöfuðu nafni ef hann er meira en 30 stafir og inniheldur 3 orð eða meira.

Verkefni 10.3
Bættu við lausnina á verkefni F.8 þannig að forritið geti líka unnið í hina áttina og breytt ísl-ensku í íslensku.

10.b. Vector

Í 10.a. var fjallað um ólíkar aðferðir til að geyma runu af stöfum. Með því að nota fylki er hægt að geyma fleira en stafi. Raunar hafa fylki sérstöðu meðal gagnagrinda sem standa til boða í Java því það er hægt að nota þau til að geyma hvað sem er, bæði hluti af öllum gerðum og einfaldar tegundir eins og `char`, `double`, `int` eða `boolean`.

Yfirleitt hentar vel að nota fylki þegar vitað er fyrirfram hvað þarf að geyma marga hluti en síður þegar fjöldinn er óviss. Það er auðvelt að nálgast einstaka hluti í fylki en hins vegar fremur erfitt að bæta hlutum inn í miðja röðina.

Klasinn `Vector` sem fylgir Java í pakkanum `java.util` er eins og fylki að því leyti að hann getur geymt marga hluti af hvaða tegund sem er, en ekki þó einfaldar tegundir. `Vector` hefur það fram yfir fylki að ekki þarf að ákveða fyrirfram hvað hann á að rúma mikið, plássið vex eftir þörfum og það er hægt að bæta hlutum hvort sem er aftan við rununa eða inn í miðju. Eftirfarandi skipanir búa til þrjá strengi og einn `Vector` og setja strengina í hann.

```
String s1 = "hani";
String s2 = "krummi";
String s3 = "svín";
Vector v;
v = new Vector();
v.addElement(s1);
v.addElement(s2);
v.addElement(s3);
```

Hægt er að bæta nýjum hlutum hvar sem er í vektorinn. Eigi t.d. að setja strenginn "hundur" á milli "krummi" og "svín" er hægt að gera það með skipuninni

```
String s4 = "hundur";
v.insertElementAt(s4, 2);
```

`Vector` er eins og fylki að því leyti að fyrsta sætið er númer 0. Áður en síðustu tvær skipanir voru gefnar var "hani" í sæti 0, "krummi" í sæti 1 og "svín" í sæti 2 en nú er "hundur" kominn í sæti 2 og "svín" er í sæti númer 3.

Auk aðferðanna `addElement` og `insertElementAt` til að setja hluti í `Vector` eru til aðferðir til að sækja hluti úr `Vector`, eyða hlut og skrifa yfir hlut. Ef eftirfarandi skipanir eru gefnar í framhaldi af skipununum hér fyrir ofan verður `v` með "hani" í sæti númer 0, "köttur" í sæti númer 1 og "svín" í sæti númer 2. Breytan `s6` fær gildið "svín".

```
v.removeElementAt(1);   // s2 sem inniheldur "krummi" og er í
                        // sæti númer 1 fjarlægður.
String s5 = "köttur";
v.setElementAt(s5, 1);  // s5 fer ofan í "hundur" sem nú er í
                        // sæti númer 1 eftir að "krummi" sem
                        // var þar hefur verið fjarlægður.
String s6 = (String)v.elementAt(2);
```

`Vector` getur geymt ótiltekinn fjölda hluta og bætt nýjum hlutum hvort sem er aftast eða fremst í röðina. En þessi gerð safnklasa getur aðeins geymt hluti af tegundinni `Object`. Allar aðferðir til að sækja hluti úr `Vector` skila gildi af tegundinni `Object` og allar aðferðir til að setja eitthvað í `Vector` taka við gildi af þessari sömu tegund.

Þar sem allir klasar eru undirklasar `Object` þýðir þetta að `Vector` getur geymt hluti af hvaða tegund sem er en ekki einfaldar tegundir en þegar hlutur er sóttur úr `Vector`

verður yfirleitt að breyta honum í rétta tegund. Síðasta skipunin hér að ofan sækir t.d. streng í `Vector`. Ef hún væri svona

```
String s6 = v.elementAt(2);
```

þá birti Java þýðandinn villumeldingu þess efnis að ekki sé hægt að geyma hlut af tegundinni `Object` í breytu sem er skilgreind sem `String`. Þótt innihald sætis númer 2 í vektornum `v` sé, í þessu tilviki, strengur er hann geymdur í breytu sem er skilgreind sem `Object` og til að nota hann sem streng verður því að breyta um tegund með því að setja (`String`) framan við `v.elementAt(2)`.

Forrit_10_02 geymir hluti af tegundinni `Hringur` í `Vector`. Í því kemur vel fram hvernig þarf að breyta því sem sótt er í `Vector` úr `Object` í aðrar tegundir.

Klasinn `Hringur` er svo til eins og í forrit_0A_01 og er því ekki prentaður hér. Klasinn `NotkunVektora` setur 4 takka á myndflöt. Ef smellt er á einn er nýjum hring bætt í vektorinn `v`. Ef smellt er á annan eru y-hnit miðpunkts allra hringja lækkuð um 5. Sá þriðji hækkar y-hnitin um 5 og sá fjórði eyðir öllu út vektornum `v`.

```
import java.awt.*;
import java.applet.Applet;
import java.util.Random;
import java.util.Vector;
//
// Forrit_10_02
// NotkunVektora
//
public class NotkunVektora extends Applet
{
   Vector v;
   Button bBuaTilHring;
   Button bUpp;
   Button bNidur;
   Button bHreinsa;
   Random r;

   public void init()
   {
      v = new Vector();
      r = new Random();
      bBuaTilHring = new Button("Búa til hring");
      bUpp = new Button("Upp");
      bNidur = new Button("Niður");
      bHreinsa = new Button("Hreinsa");
      this.add(bBuaTilHring); this.add(bUpp);
      this.add(bNidur); this.add(bHreinsa);
   }

   // Aðferðin paint sýnir alla hringi sem geymdir eru í vektornum v.
   public void paint(Graphics g)
   {
      for(int i=0; i < v.size(); i++)
      {
         // Ath. til að framkvæma aðferðina syna á hring sem
         // vistaður er í Vector þarf að breyta honum úr Object
         // í Hringur.
         ((Hringur)v.elementAt(i)).syna(g);
      }
   }
```

```
   public boolean action(Event e, Object o)
   {
      if (e.target == bBuaTilHring)
      {
         // Hringur búinn til. x-hnit, y-hnit og
         // radíus eru fengin úr slembitalnagjafanum r.
         double radius = 5+20*r.nextDouble();
         double xhnit = 25+150*r.nextDouble();
         double yhnit = 25+150*r.nextDouble();

         // Hring bætt í vektor.
         v.addElement(new Hringur(radius, xhnit, yhnit));

         // Aðferðin repaint sér um að paint-aðferðin verði
         // framkvæmd við fyrstu hentugleika.
         this.repaint();
         return true;
      }

      if (e.target == bUpp)
      {
         for(int i=0; i < v.size(); i++)
         {
            ((Hringur)v.elementAt(i)).yHnitMidju -= 5;
         }
         this.repaint();
         return true;
      }

      if (e.target == bNidur)
      {
         for(int i=0; i < v.size(); i++)
         {
            ((Hringur)v.elementAt(i)).yHnitMidju += 5;
         }
         this.repaint();
         return true;
      }

      if (e.target == bHreinsa)
      {
         v.removeAllElements();
         this.repaint();
         return true;
      }

      return false;
   }
}
```

Eins og ljóst er af klasanum `NotkunVektora` flækir það málin töluvert mikið að þurfa sífellt að breyta því sem sótt er í `Vector` úr `Object` í einhverja aðra tegund til að geta framkvæmt á hlutunum aðrar aðferðir en þær sem þeir erfa frá `Object`.

Það er hægt að nota klasann `Vector` sem fylgir Java til að búa til sérhæfða vektora sem geta t.d. aðeins geymt strengi eða hringi. Í forrit_10_03 er þetta gert. Klasinn `HringaVektor` er raunar ekkert annað en umbúðir utan um `Vector`-klasann í `java.util` pakkanum. Aðferðir eins og `addElement` og `size` gera ekkert annað en að framkvæma samnefndar aðferðir í `Vector`. En aðferðin `elementAt`, sem sækir hluti úr vektornum, breytir því sem sótt er úr `Object` í tegundina `Hringur` áður en því er skilað.

Klasinn `Hringur` er eins og í forrit_0A_01 og forrit_10_02 og er því ekki prentaður hér. Klasinn `notkunVektora` gerir nákvæmlega það sama og samnefndur klasi í

forrit_10_02 en notar `HringaVektor` í stað `Vector` með þeim afleiðingum að í stað skipana á borð við

```
((Hringur)v.elementAt(i)).syna(g);
```

dugar að nota skipanir eins og

```
v.elementAt(i).syna(g);
```

Hér koma klasarnir `HringaVektor` og `NotkunVektora`.

```
import java.util.Vector;
//
// Forrit_10_03
// HringaVektor
//
class HringaVector
{
   Vector v;

   public HringaVector()
   {
      v = new Vector();
   }

   public void addElement(Hringur h)
   {
      v.addElement(h);
   }

   public int size()
   {
      return v.size();
   }

   public Hringur elementAt(int i)
   {
      return (Hringur)(v.elementAt(i));
   }

   public void removeAllElements()
   {
      v.removeAllElements();
   }
} // HringaVektor endar.
```

```
import java.awt.*;
import java.applet.Applet;
import java.util.Random;
//
// Forrit_10_03
// NotkunHringaVektora
//
public class NotkunHringaVektora extends Applet
{
   HringaVector v;
   Button bBuaTilHring;
   Button bUpp;
   Button bNidur;
   Button bHreinsa;
   Random r;

   public void init()
   {
      v = new HringaVector();
      r = new Random();
      bBuaTilHring = new Button("Búa til hring");
      bUpp = new Button("Upp");
      bNidur = new Button("Niður");
      bHreinsa = new Button("Hreinsa");
      this.add(bBuaTilHring);
      this.add(bUpp);
      this.add(bNidur);
      this.add(bHreinsa);
   }

   public void paint(Graphics g)
   {
      for(int i=0; i < v.size(); i++)
      {
         v.elementAt(i).syna(g);
      }
   }

   public boolean action(Event e, Object o)
   {
      if (e.target == bBuaTilHring)
      {
         double radius = 5+20*r.nextDouble();
         double xhnit = 25+150*r.nextDouble();
         double yhnit = 25+150*r.nextDouble();
         v.addElement(new Hringur(radius, xhnit, yhnit));
         this.repaint();
         return true;
      }

      if (e.target == bUpp)
      {
         for(int i=0; i < v.size(); i++)
         {
            v.elementAt(i).yHnitMidju -= 5;
         }
         this.repaint();
         return true;
      }
```

```
      if (e.target == bNidur)
      {
         for(int i=0; i < v.size(); i++)
         {
            v.elementAt(i).yHnitMidju += 5;
         }
         this.repaint();
         return true;
      }

      if (e.target == bHreinsa)
      {
         v.removeAllElements();
         this.repaint();
         return true;
      }

      return false;
   }
}
```

Verkefni 10.4*
Bættu við klasann `NotkunVektora` í forrit_10_02 því sem þarf til að ekki sé bara hægt að færa hringina upp og niður heldur líka til hægri og vinstri.

Verkefni 10.5*
Notaðu klasann `HringaVektor` í forrit_10_03 sem fyrirmynd og búðu til `StrengjaVektor`, þ.e. klasa sem geymir strengi í `Vector`. Búðu svo til klasa sem notar `StrengjaVektor` til að geyma strengi. Láttu hann hafa einn textareit (`Textfield`) til að skrifa strengi í og textasvæði (`TextArea`) til að sýna allt innihald vektorsins. Forritið þarf tvo takka. Þegar ýtt er á annan á strengurinn í textareitnum að bætast við vektorinn og þegar ýtt er á hinn á allt innihald vektorsins að skrifast í textasvæðið.

Verkefni 10.6
Bættu við lausnina á verkefni 10.5 þannig að með því að ýta á þriðja takkann sé öllum strengjum í vektornum breytt á sama hátt og í lausninni á verkefni 10.1.

Verkefni 10.7
Breyttu lausninni á verkefni F.9 þannig að punktarnir séu geymdir í vektor jafnóðum og þeir eru sóttir og myndin ekki teiknuð fyrr en allir punktarnir eru komnir inn í vektorinn.

Verkefni 10.8
Bættu við lausnina á verkefni 10.7 þannig að hægt sé að láta myndina færast upp og niður.

Verkefni 10.9 (Erfitt og aðeins fyrir þá sem hafa yndi af stærðfræði)
Bættu við lausnina á 10.7 þannig að hægt sé að tilgreina punkt og horn og láta myndina snúast um punktinn jafnmargar gráður og hornið er.

10.c. Algengar gagnagrindur

Þegar þörf er að geyma eða vinna með marga hluti af sömu tegund hentar oftast nær best að nota fylki eða vektora. En til eru miklu fleiri möguleikar því auk þess sem Java hefur innbyggða safnklasa til viðbótar við þá sem hér hafa verið taldir geta forritarar búið til sína eigin.

Meðal þeirra safnklasa sem innbyggðir eru í Java og ekki hefur verið fjallað um hér má merkasta telja stafla (`Stack` í pakkanum `java.util`) og tætitöflur (`HashTable` í pakkanum `java.util`).

Staflar gegna mikilvægu hlutverki í ýmsum forritum, t.d. stýrikerfum. Þeir eru eins og `Vector` nema hvað aðeins er hægt að bæta hlutum við endann (toppinn) á þeim og aðeins er hægt að sækja hluti frá enda. Þetta þýðir að sá hlutur sem síðast er settur í stafla er fyrstur í röðinni þegar sótt er úr honum. Þessu má líkja við stafla af diskum í mötuneyti. Sá sem vaskar upp raðar diskunum í stafla og matargestir taka diska úr staflanum og fyrsti matargesturinn tekur þann disk sem síðast var þveginn.

Tætitöflur gegna mikilvægu hlutverki í gagnasöfnum af ýmsu tagi. Þeirra helsti kostur er að mjög fljótlegt er að leita að gögnum (eða fletta upp) í þeim, miklu fljótlegra heldur en að leita í gegnum fylki eða vektor.

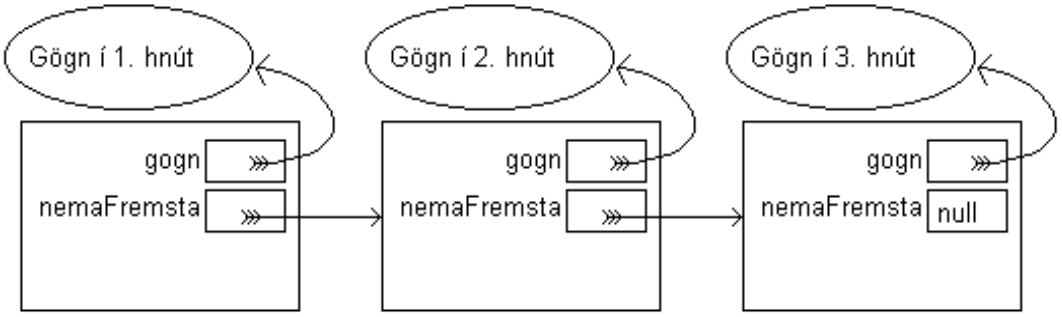
Sem dæmi um aðra safnklasa sem gegna mikilvægu hlutverki í ýmislegum hugbúnaði má nefna lista, biðraðir og tré. Forrit_10_04 sýnir hvernig listi er búin til og notaður. Um tré verður fjallað í næsta kafla. Biðraðir eru svipaðar stöflum að því leyti að þær eru vektorar (eða listar) þar sem fylgt er strangri reglu um í hvaða röð hlutir eru sóttir. Það sem fyrst er sett í biðröð er líka fyrst út úr henni, öfugt við stafla þar sem hluturinn sem fyrst er settur í staflann er tekinn úr honum síðast.

10.d. Listi

Klasinn `Listi` í forrit_10_04 sýnir hvernig hægt er að nota Java til að smíða gagnagrind af þessari tilteknu gerð. `Listi` býður upp á svipaða möguleika og `Vector` því hann hefur ekki fyrirframákveðna stærð heldur stækkar jafnóðum og bætt er í hann. Öll verkefni sem hægt er að leysa með því að nota lista (eða stafla eða biðröð) er hægt að leysa jafnvel með því að nota klasann `Vector` sem er innbyggður í Java. Það má því segja að óþarft sé að búa til klasa eins og `Listi`. En það sama verður ekki sagt um tré sem eru til umfjöllunar í næsta kafla og til að skilja hvernig tré eru búin til þarftu fyrst að átta þig á því hvernig listi er smíðaður svo það er ekki til einskis að skoða forrit_10_04.

Hlutur af tegundinni `Listi` er í rauninni bara eitt stak á listanum. Eitt slíkt stak eða atriði kallast hnútur. Hver hnútur geymir einn hlut af tegundinni `Object` í klasabreytunni `gogn` og nýjan lista í klasabreytunni `nemaFremsta`. Þessar tvær klasabreytur innihalda í raun vistfang hluta. `gogn` bendir á eitt striði á listanum og `nemaFremsta` bendir á næsta hnút á listanum.

Myndin sýnir lista með þrem hnútum. Í hverjum hnút bendir klasabreytan `nemaFremsta` á næsta hnút nema í þeim síðasta, þar hefur hún gildið `null` og vísar því ekki á neitt. Ef breytan `x` er af tegundinni `Listi` og vísar á fyrsta hnút listans. Þá vísar `x.nemaFremsta` á lista sem hefst á hnút númer 2, þ.e. á upphaf lista sem inniheldur allt nema fremsta hnútinn.



Sú tækni sem hér er notuð, að láta klasa hafa klasabreytu af eigin tegund, gegnir lykilhlutverki í smiði safnklasa. Hlutur inniheldur þá gögn og tilvísun í annan hlut af sömu tegund, sem inniheldur samskonar gögn og tilvísun í þriðja hlutinn o. s. frv.

```
//
// Forrit_10_04
// Listi
//
public class Listi                  // Listi geymir einn hlut af
{                                   // tegundinni Object og vísar
   Object gogn;                     // á annan lista sem kallast
   Listi nemaFremsta;               // nemaFremsta því hann
                                    // inniheldur allt nema
                                    // fremsta hnútinn á listanum.
   public Listi(Object o)           //
   {                                // Þegar listi er fyrst búinn
      gogn = o;                     // til hefur hann aðeins einn
      nemaFremsta = null;           // hnút og nemaFremsta er null.
   }

   public Object fremsta()          // Skilar gögnunum úr
   {                                // fremsta hnút
      return gogn;                  // listans.
   }

   public Listi nemaFremsta()       // Skilar listanum
   {                                // sem er fyrir aftan
      return nemaFremsta;           // fremsta hnútinn.
   }

   // Býr til nýjan hnút með sömu gögnum og fremsti hnútur og setur
   // hann aftan við fremsta hnútinn. Lætur svo gögnin í fremsta
   // hnútnum verða o.
   public void baetaFramanAListaObject o)
   {
      Listi hnutur = new Listi(gogn);
      hnutur.nemaFremsta = nemaFremsta;
      nemaFremsta = hnutur;
      gogn = o;
   }
```

```
   // Býr til nýjan hnút sem hefur o fyrir gögn. Finnur svo aftasta
   // hnútinn á listanum og bætir nýja hnútnum aftan við hann.
   public void baetaAftanAListA(Object o)
   {
      Listi hnutur = new Listi(o);
      Listi x = this;
      while (x.nemaFremsta != null)
      {
         x = x.nemaFremsta;
      }
      x.nemaFremsta = hnutur;
   }
}
```

Klasinn `ListaNotkun` í forrit_10_04 sýnir hvernig nota má lista. Hann gerir nákvæmlega það sama og `NotkunVektora` í forrit_10_02 og forrit_10_03. Klasinn `Hringur` er eins og í forrit_0A_01 og forrit_10_02 og forrit_10_03 og er því ekki prentaður hér.

Eins og listi er skilgreindur hér geymir hann aðeins `Object`. Þegar gögn eru sótt þarf því að bera sig eins að og þegar sótt er úr `Vector` og breyta gögnunum úr `Object` í aðrar tegundir.

Taktu eftir því að þegar fyrsti hlutur er settur á lista er smiðurinn `Listi` notaður en eftir það er aðferðin `baetaFramanAListA` notuð til að bæta við hann nýjum hnútum.

```
import java.awt.*;
import java.applet.Applet;
import java.util.Random;
//
// Forrit_10_04
// ListaNotkun
//
public class ListaNotkun extends Applet
{
   Listi listi;               // Breytan listi er notuð
   Button bBuaTilHring;       // til að vísa á fremsta
   Button bUpp;               // hnút á listanum.
   Button bNidur;
   Button bHreinsa;
   Random r;

   public void init()
   {
      r = new Random();
      bBuaTilHring = new Button("Búa til hring");
      bUpp = new Button("Upp");
      bNidur = new Button("Niður");
      bHreinsa = new Button("Hreinsa");
      this.add(bBuaTilHring);
      this.add(bUpp);
      this.add(bNidur);
      this.add(bHreinsa);
   }
```

```
public void paint(Graphics g)
{
   // Upphaflega vísar x á fremsta hnút lista, þ.e. það
   // sama og listi vísar á.
   Listi x = listi;
   // x færist eftir listanum þar til komið er að hnút
   // sem ekki vísar á annan hnút heldur á null.
   while(x != null)
   {
      ((Hringur)x.fremsta()).syna(g);
      x = x.nemaFremsta();
   }
}

public boolean action(Event e, Object o)
{
   if (e.target == bBuaTilHring)
   {
      double radius = 5+20*r.nextDouble();
      double xhnit = 25+150*r.nextDouble();
      double yhnit = 25+150*r.nextDouble();
      if (listi == null)
      {
       listi = new Listi(new Hringur(radius, xhnit, yhnit));
      }
      else
      {
       listi.baetaFramanAListi(new Hringur(radius, xhnit, yhnit));
      }
      this.repaint();
      return true;
   }

   if (e.target == bUpp)
   {
      Listi x = listi;
      while(x != null)
      {
         ((Hringur)x.fremsta()).yHnitMidju -= 5;
         x = x.nemaFremsta();
      }
      this.repaint();
      return true;
   }

   if (e.target == bNidur)
   {
      Listi x = listi;
      while(x != null)
      {
         ((Hringur)x.fremsta()).yHnitMidju += 5;
         x = x.nemaFremsta();
      }
      this.repaint();
      return true;
   }
```

```
        if (e.target == bHreinsa)
        {
           listi = null;
           this.repaint();
           return true;
        }

        return false;
    }
}
```

Verkefni 10.10
Búðu til klasa sem notar lista til að geyma strengi. Láttu hann hafa einn textareit (`TextField`) til að skrifa strengi í og textasvæði (`TextArea`) til að sýna allt innihald listans. Forritið þarf tvo takka. Þegar ýtt er á annan á strengurinn í textareitnum að bætast við listann og þegar ýtt er á hinn á allt innihald listans að skrifast í textasvæðið. (Þetta verkefni er svipað verkefni 10.5.)

Verkefni 10.11
Bættu við lausnina á verkefni 10.10 þannig að með því að ýta á þriðja takkann sé öllum strengjum á listanum breytt á sama hátt og í lausninni á verkefni 10.1. (Þetta verkefni er svipað verkefni 10.6.)

Verkefni 10.12
Bættu við klasann `Listi` aðferðum sem samsvara aðferðunum `elementAt`, `setElementAt`, `insertElementAt` og `removeElementAt` í klasanum `Vector`.

10.x. Spurningar og umhugsunarefni

1. Hvað merkja orðin „safnklasi“ og „gagnagrind“?
2. Hvaða munur er á `String` og `StringBuffer`?
3. Hvers konar gögn er hægt að geyma í fylki en ekki í `Vector`?
4. Hvaða munur er á fylki og `Vector` (hvað getur `Vector` sem fylki getur ekki)?
5. Gerðu ráð fyrir að `v` sé `Vector` sem inniheldur nokkra strengi. Hvers vegna er ekki hægt að gefa skipunina
   ```
   String s = v.elementAt(1);
   ```
6. Til hvers eru aðferðirnar: `addElemnt`, `elementAt`, `setElementAt`, `insertElementAt` og `removeElementAt`?
7. Hvað eru stafli og biðröð og hvaða munur er á þessu tvennu?
8. Hvað er hnútur?
9. Hvers vegna þarf klasinn `Listi` að innihalda klasabreytu af eigin tegund?
10. Á hvað vísar klasabreytan sem er af tegundinni `Listi` í hnút sem er aftast á lista?

Til umhugsunar

Klasinn `Listi` inniheldur eina klasabreytu af eigin tegund. Ekkert er því til fyrirstöðu að klasi innihaldi margar klasabreytur af eigin tegund. T.d. er hægt að smíða svokallaða tvítengda lista með því að láta hvern hnút vísa bæði á næsta hnút fyrir framan sig og næsta hnút fyrir aftan sig. Hver hnútur, nema fremsti og aftasti, hefur því tengingu frá tveim öðrum.

Getur þú teiknað mynd af tvítengdum lista, hliðstæða myndinni fremst í kafla 10.d?

Ætli listi geti verið hringtengdur þannig að síðasti hnútur bendi ekki á `null` heldur á þann fyrsta? Hvaða kosti/ókosti hefur slíkur listi?

Hægt er að búa til flóknari gagnagrindur en lista með því að láta hvern hnút vísa á tvo eða jafnvel fleiri aðra. Það er jafnvel ekkert því til fyrirstöðu að hver hnútur innihaldi klasabreytu með fylki, `Vector` eða lista af hlutum eigin gerðar?

Getur þú teiknað mynd af gagnagrind þar sem hver hnútur vísar á tvo aðra og engir tveir vísa á þann sama?

Getur þú ímyndað þér einhver not fyrir gagnagrind þar sem hver hnútur inniheldur marg-ar klasabreytur af eigin tegund?

11. kafli: Endurkoma og tré

11.a. Endurkoma

Í kafla 10.d. var kynntur klasi sem inniheldur breytu af eigin tegund. Það er eins og hlutir af þessari gerð innihaldi annan eins sem svo aftur inniheldur annan eins. Þetta verður þó ekki endalaust því að lokum komum við að hlut sem vísar ekki á annan eins heldur á null.

Í ljósi þessa getum við skilgreint lista svona:

> Listi er null eða
> hnútur með lista fyrir aftan.

Þessi skilgreining er að því leyti dálítið skrítin að hugtakið sem skilgreina á (nefnilega listi) kemur fyrir í skilgreiningunni. Slík skilgreining kallast endurkvæm. Í stærðfræði og tölvufræði eru endurkvæmar skilgreiningar algengar. Til dæmis er hægt að skilgreina heiltöluveldi þar sem veldisvísirinn er 0 eða meira svona.

> Ef $n = 0$ þá er $a^n = 1$ og
> ef $n > 0$ þá er $a^n = a \times a^{n-1}$.

Hér er a^{n-1} notað til að skilgreina a^n sem er allt í lagi því runan verður ekki endalaus, hún endar þegar kemur að a^0. Samkvæmt þessari skilgreiningu er

$$a^3 = a \times a^2 = a \times a \times a^1 = a \times a \times a \times a^0 = a \times a \times a \times 1.$$

Þegar skilgreiningin er rakin alla leið kemur í ljós að hún styðst eingöngu við margföldun.

Á svipaðan hátt er hægt að skilgreina a! svona:

> Ef $a = 0$ þá er $a! = 1$ og
> ef $a > 0$ þá er $a! = a \times (a - 1)!$

Það er ekki eingöngu í stærðfræði og tölvufræði sem endurkvæmar skilgreiningar koma við sögu. Það er líka hægt að nota endurkomu til að skilgreina ýmis hversdagsleg hugtök, eins og t.d. hugtakið afkomandi.

> Afkomendur manns eru börn hans og afkomendur þeirra.

Í flestum forritunarmálum er hægt að nota endurkomu til að skilgreina aðferðir og klasa. Aðferð notar endurkomu ef hún kallar á sjálfa sig. Klasi notar endurkomu ef hann inniheldur klasabreytu af eigin tegund.

Forrit_11_01 inniheldur nokkrar endurkvæmar aðferðir. Þær heita `endurkoma1`, `endurkoma2`, `endurkoma3`, `hropmerkt` og `fibonacci`. Fyrir hverja þessara aðferða er einn takki og aðferðin er framkvæmd þegar smellt er á hann.

Verkefni 11.1*
Keyrðu klasann `Endurkoma` í forrit_11_01 og prófaðu allar aðferðirnar. Áttaðu þig á muninum á `endurkoma1`, `endurkoma2` og `endurkoma3`.

```
import java.applet.Applet;
import java.awt.*;
//
// Forrit_11_01
// Endurkoma
//
public class Endurkoma extends Applet
{
      Button bEndurkoma1, bEndurkoma2, bEndurkoma3,
             bEndurkoma4, bEndurkoma5;
      TextArea t1;

      public void init()
      {
         t1 = new TextArea(6, 20);
         bEndurkoma1 = new Button("Endurk. 1");
         bEndurkoma2 = new Button("Endurk. 2");
         bEndurkoma3 = new Button("Endurk. 3");
         bEndurkoma4 = new Button("Hropmerkt");
         bEndurkoma5 = new Button("Fibonacci");
         this.add(t1); this.add(bEndurkoma1);
         this.add(bEndurkoma2); this.add(bEndurkoma3);
         this.add(bEndurkoma4); this.add(bEndurkoma5);
      }

      void endurkoma1(int n)
      {
         if (n > 0)
         {
            t1.appendText("n = " + n + "\n");
            endurkoma1(n - 1);
         }
      }

      void endurkoma2(int n)
      {
         if (n > 0)
         {
            endurkoma2(n - 1);
            t1.appendText("n = " + n + "\n");
         }
      }

      void endurkoma3(int n)
      {
         if (n > 0)
         {
            t1.appendText("n = " + n + "\n");
            endurkoma3(n - 1);
            t1.appendText("n = " + n + "\n");
         }
      }

      int hropmerkt(int n)
      {
         if (n == 0) { return 1; }
         else { return n * hropmerkt(n - 1); }
      }
```

```
      int fibonacci(int n)           // Finnur fibonacci tölu
      {                              // númer n
         if ((n == 1) || (n == 2))
         {
            return 1;
         }
         else
         {
            return fibonacci(n - 1) + fibonacci(n - 2);
         }
      }

      public boolean action(Event e, Object o)
      {
         if (e.target == bEndurkoma1)
         {
            t1.setText("");
            endurkoma1(3);
            return true;
         }
         if (e.target == bEndurkoma2)
         {
            t1.setText("");
            endurkoma2(3);
            return true;
         }
         if (e.target == bEndurkoma3)
         {
            t1.setText("");
            endurkoma3(3);
            return true;
         }
         if (e.target == bEndurkoma4)
         {
            t1.setText("10! er " + hropmerkt(10));
            return true;
         }
         if (e.target == bEndurkoma5)
         {
            t1.setText("Fibonacci tala \nnúmer 10 er "
                                              + fibonacci(10));
            return true;
         }
         return false;
      }
}
```

Skoðum nú aðferðina `endurkoma1`

```
      void endurkoma1(int n)
      {
         if (n > 0)
         {
            t1.appendText("n = " + n + "\n");
            endurkoma1(n - 1);
         }
      }
```

Ef hún fær senda töluna 3 þá byrjar hún á að bæta strengnum "n = 3" í textareitinn `t1` og setur svo í gang nýtt eintak af sjálfri sér og sendir því töluna 2 (þ.e. 3-1). Þetta nýja eintak af aðferðinni bætir "n = 2" í textareitinn og setur svo í gang þriðja eintakið og sendir því töluna 1. Þetta þriðja eintak bætir "n = 1" í textareitinn og ræsir fjórða eintakið af aðferðinni og sendir því töluna 0. Þar sem aðferðin gerir ekkert ef færibreytan

n hefur gildið 0 stöðvast runan hér og fjórða eintakið lýkur störfum án þess að ræsa hið fimmta. Þar með hefur þriðja eintakið lokið við að keyra hið fjórða og getur kvatt þennan heim og svo koll af kolli. Ef aðferðin setti nýtt eintak af stað án þess að lækka gildi n, ef hún væri t.d. svona

```
void endurkoma1(int n)
{
   if (n > 0)
   {
      t1.appendText("n = " + n + "\n");
      endurkoma1(n);
   }
}
```

þá tæki endurkoman aldrei enda. Það færi af stað nýtt og nýtt eintak af aðferðinni og aldrei kæmi að eintaki sem fengi sent 0 og gæti því hætt án þess að ræsa enn eitt. Þar sem hvert eintak af endurkvæmri aðferð tekur rúm í minni tölvunnar verður endalaus endurkoma til þess að minnið sem forritið hefur til umráða fyllist.

Aðferðin `endurkoma2` er svipuð `endurkoma1` en hún kallar fyrst á nýtt eintak af sjálfri sér og bætir svo streng í textareitinn. Þriðja aðferðin, `endurkoma3`, gerir það sama og hinar báðar. Hún skrifar fyrst gildi `n` í textareitinn, kallar svo á nýtt eintak af sjálfri sér og skrifar að síðustu gildið á `n` aftur í textareitinn.

Taflan hér að neðan sýnir í hvaða röð skipanirnar eru framkvæmdar ef aðferðin `endurkoma3` er upphaflega ræst með því að senda henni töluna 2. Til að spara pláss er því sem á að vera innan sviga í `t1.appendText("n = " + n + "\n")` sleppt og settir þrír punktar í staðinn. Í stað færibreytunnar `n` er skrifað gildið sem hún fær. Aðferðin `appendText` í þriðja eintaki aðferðarinnar er aldrei framkvæmd því þetta síðasta eintak fær senda töluna 0 og það sem er innan í `if`-blokkinni er þá aðeins framkvæmt að `n > 0`.

Fyrsta eintak	Annað eintak	Þriðja eintak
`void endurkoma3(2)`	`void endurkoma3(1)`	`void endurkoma3(0)`
`{`	`{`	`{`
`if (2 > 0)` **1**	`if (1 > 0)` **4**	`if (0 > 0)` **7**
`{`	`{`	`{`
`t1.appendText(...);` **2**	`t1.appendText(...);` **5**	`t1.appendText(...);`
`endurkoma3(2 - 1);` **3**	`endurkoma3(1 - 1);` **6**	`endurkoma3(0 - 1);`
`t1.appendText(...);` **9**	`t1.appendText(...);` **8**	`t1.appendText(...);`
`}`	`}`	`}`
`}`	`}`	`}`

Aðferðirnar `hropmerkt` og `fibonacci` nota endurkvæmar skilgreiningar á aðgerðinni hrópmerkt og fibonacci talnarununni til að reikna n! og fibonacci tölu númer n. Fibonacci talnarunan er skilgreind svona:

$f_1 = 1.$
$f_2 = 1.$
Ef $n > 2$ þá er $f_n = f_{n-1} + f_{n-2}$

Fyrstu 10 fibonacci tölurnar eru:

1 1 2 3 5 8 13 21 34 55.

Það er ekki heppilegt að nota endurkomu til að reikna n! eða finna fibonacci tölur. Aðferð sem notar venjulega endurtekningu er bæði hraðvirkari og sparneytnari á minnispláss. Hvert eintak af aðferð sem verður til við endurkoma þarf pláss í minni fyrir allar sínar staðværu breytur og færibreytur svo djúp endurkoma getur verið ansi minnisfrek.

Í fjórða kafla var fjallað um aðferð Evklíðs til að finna stærsta sameiginlegan þátt tveggja talna. Þetta er dæmi um aðferð þar sem endurkoma á vel við. Hún er bæði einfaldari og skiljanlegri ef hún er forrituð með endurkomu heldur en ef hún er forrituð með venjulegri endurtekningu. Aðferðina má skilgreina svona:

t og n eru heilar tölur.
Ef n = 0 þá er útkoman t
annars er útkoman stærsti sameiginlegi þáttur talanna n og (t mod n).

Forrit_11_02 er eins og forrit_05_01 nema hvað aðferð Evklíðs er skrifuð með endurkomu svona:

```
int evklid(int t, int n)
{
   if (n == 0)
   {
      return t;
   }
   else
   {
      return evklid(n, t % n);
   }
}
```

Þar sem forrit_11_02 er að öllu öðru leyti eins og forrit_05_01 er það ekki prentað hér.

Verkefni 11.2*
Notaðu skilgreininguna á veldi hér að ofan til að búa til endurkvæma aðferð til að reikna a^n þar sem n er jákvæð heiltala.

Verkefni 11.3*
Fallið f(x) er skilgreint svona:

Ef x = 0 þá er f(x) = 1 annars er
$f(x) = 2^x + f(x - 1)$

Búðu til aðferð sem notar endurkomu til að reikna f(x).

Verkefni 11.4 (Erfitt)
Í kafla 8.x var minnst á röðunaraðferðina Quicksort. Hún er oft forrituð með endurkomu. Quicksort aðferð til að raða fylki af heilum tölum má lýsa svona (á samblandi af Java og íslensku). Gert er ráð fyrir að fylkið f sé klasabreyta og qs aðferðin sé ræst með því að senda henni númer fyrsta og síðasta hólfsins í því.

```
public void qs(int upphaf, int endi)
{
   ef (upphaf < endi)
   {
      Velja eitt stak, s, sem er með númer milli upphaf og endi.
      Setja öll stök í fylkinu sem eiga að vera framan við s fremst
      og þar fyrir aftan öll stök sem ekki eiga að vera framan við s.
      int midja = sætið_sem_s_er_í.
      qs(upphaf, midja-1);
      qs(midja + 1, endi);
   }
}
```

Búðu til forrit sem raðar fylki af heilum tölum með quicksort aðferðinni.

11.b. Teikning með endurkomu

Með því að nota endurkomu er hægt að teikna ýmis mynstur sem erfitt er að forrita með öðru móti. Klasinn `TeikningMedEndurkomu` í forrit_11_03 hefur endurkvæmar aðferðir til að teikna spíral, snjókorn og tré með tátugrafík eins og kynnt var í kafla A.f. Aðrir klasar í forritinu (`Kvikindi`, `Bjalla` og `Hringur`) eru gamalkunnir og verða ekki prentaðir hér.

```
import java.applet.Applet;
import java.awt.*;
//
// forrit_11_03
// TeikningMedEndurkomu
//
public class TeikningMedEndurkomu extends Applet
{
   Bjalla b;
   Graphics g;
   Button bEndurkoma1, bEndurkoma2, bEndurkoma3;

   public void init()
   {
      bEndurkoma1 = new Button("Spírall");
      bEndurkoma2 = new Button("Snjókorn");
      bEndurkoma3 = new Button("Tré");
      this.add(bEndurkoma1);
      this.add(bEndurkoma2);
      this.add(bEndurkoma3);
      g = this.getGraphics();
      b = new Bjalla(g);
      b.dregurStrik = true;
   }

   void spirall(double lengd, double horn)
   {
      if (lengd < 15)
      {
         b.fram(lengd);
         b.vinstri(horn);
         spirall(lengd*1.02, horn);
      }
   }

   void korn(double lengd)
```

```
{
   if (lengd < 5)
   {
      b.fram(lengd);
   }
   else
   {
      korn(lengd/3);
      b.vinstri(60);
      korn(lengd/3);
      b.haegri(120);
      korn(lengd/3);
      b.vinstri(60);
      korn(lengd/3);
   }
}

void tre (double lengd)
{
   if (lengd > 1)
   {
      b.fram(lengd);
      b.vinstri(60);
      tre(lengd*0.6);
      b.haegri(120);
      tre(lengd*0.6);
      b.vinstri(60);
      b.fram(-lengd);
   }
}

public boolean action(Event e, Object o)
{
   if (e.target == bEndurkoma1)
   {
      g.clearRect(0, 0, this.size().width, this.size().height);
      b.xhnit = 100;
      b.yhnit = 120;
      b.syna();
      spirall(1, 12);
      return true;
   }

   if (e.target == bEndurkoma2)
   {
      g.clearRect(0, 0, this.size().width, this.size().height);
      b.xhnit = 25;
      b.yhnit = 180;
      b.stefna = 60;
      b.syna();
      for (int i=0; i < 3; i++)
      {
         korn(150);
         b.haegri(120);
      }
      return true;
   }
```

```
      if (e.target == bEndurkoma3)
      {
         g.clearRect(0, 0, this.size().width, this.size().height);
         b.xhnit = 100;
         b.yhnit = 190;
         b.stefna = 90; // Snýr nefinu á b upp.
         b.syna();
         tre(60);
         return true;
      }
      return false;
   }
}
```

Verkefni 11.5
Keyrðu klasann `TeikningMedEndurkomu` og prófaðu allar þrjár aðferðirnar. Athugaðu svo hvernig myndin af trénu verður ef aðferðinni `tre` er breytt þannig að í stað 1 í

```
if (lengd > 1)
```

komi aðrar tölur, t.d. 10, 20 eða 30.
Athugaðu svo líka hvernig myndin af snjókorninu verður ef aðferðinni `korn` er breytt þannig að í stað 5 í

```
if (lengd > 5)
```

komi aðrar tölur, t.d. 15, 50 eða 100. Reyndu að finna hvernig einfaldasta mynd sem aðferðin `korn` getur teiknað lítur út. Þessi einfaldasta mynd er grunnmynstrið. Flóknari myndir (með dýpri endurkomu) eru gerðar með því að setja það í stað hvers striks.

Skoðum nú hvernig aðferðin `tre` virkar.

```
   void tre (double lengd)
   {
      if (lengd > 1)
      {
         b.fram(lengd);
         b.vinstri(60);
         tre(lengd*0.6);
         b.haegri(120);
         tre(lengd*0.6);
         b.vinstri(60);
         b.fram(-lengd);
      }
   }
```

Ef lengd er 1 eða minna er ekkert gert. Ef lengd er meira en 1 er fyrst dregið strik og svo tekin vinstri beygja og teiknað tré af stærðinni lengd × 0,6. Svo er snúið til hægri og teiknað annað tré af stærðinni lengd × 0,6. Að síðustu er aftur snúið í upphaflega stefnu og bakkað þannig að endað sé í sömu sporum og byrjað var í.

Í hvert sinn sem aðferðin kallar á sjálfa sig upp á nýtt sendir hún sér minni og minni tölu með því að margfalda lengd með 0,6. Á endanum er kallað á aðferðina með minni tölu en 1.

Aðferðin `tre` kallar tvisvar á sjálfa sig. Ef hún fær upphaflega sent gildið 5 setur hún af stað tvo eintök af sjálfri sér, fyrst eitt til að teikna grein til vinstri og þegar það hefur lokið störfum og búið er að beygja til hægri þá fer hitt af stað og teiknar grein til hægri. Þessi tvö eintök fá bæði sent gildið 3 (= 5 × 0,6). Hvort þeirra setur í gang tvö til og sendir þeim gildið 1,8 (= 3 × 0,6). Það verða semsagt til fjórar greinar af

lengdinni 1,8. Hver þessara fjögurra setur aðferðina tvívegis í gang og sendir gildið 1,08 (= 1,8 × 0,6) svo það verða til átta greinar af lengdinni 1,08. Hver þeirra ræsir aðferðina tvisvar og sendir gildið 0,648 (= 1,08 × 0,6) svo aðferðin fer sextán sinnum af stað með gildi undir 1 og þessi eintök eiga það sammerkt að gera ekki neitt.

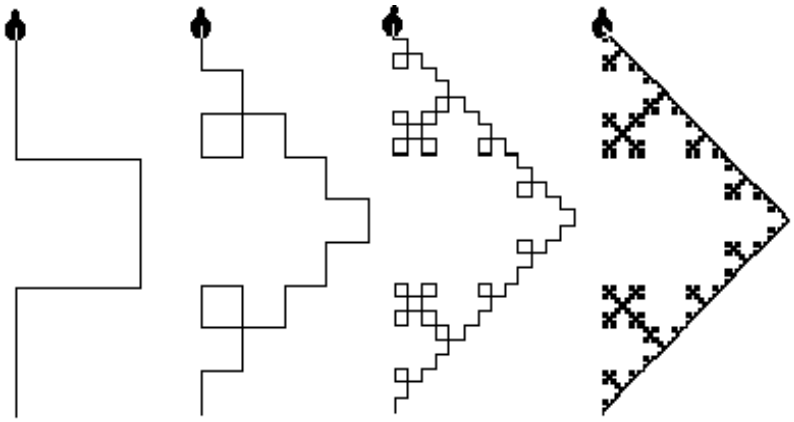
Verkefni 11.6

Myndirnar fjórar hér fyrir neðan eru allar búnar til með sömu endurkvæmu aðferðinni sem við getur kallað munstur.

Aðferðin getur t.d. verið eitthvað á þessa leið:

```
void munstur(double lengd)
{
   if (lengd < 1)
   {
      b.fram(lengd);
   }
   else
   {
      // Hér komi skipanir sem láta
      // aðferðina ræsa sjálfa sig
      // nokkrum sinnum og beygja á milli.
   }
}
```

Ef lengd er undir einhverju tilteknu marki, m (sem hér að ofan er haft 1), þá á aðferðin að teikna strik, annars á hún að kalla á sjálfa sig 5 sinnum og senda sér lengd/3 og beygja um 90 gráður á milli (ýmist til hægri eða vinstri).



Þegar þessar myndir voru gerðar var aðferðin upphaflega ræst með gildinu 180. Fyrst var m haft rúmlega 60 og þá varð fyrsta myndin til, svo var m = 30 og þá varð önnur myndin til. Þriðja myndin kom þegar m var rúmlega 11 og að síðustu var m = 1 og þá varð myndin lengst til hægri til.

Fyrsta myndin er í rauninni grunnmynstrið. Sú næsta fæst með því að teikna svona grunnmynstur (þrefalt minna) í stað hvers striks o.s.frv.

Ljúktu við aðferðina munstur. Fyrsta myndin segir allt sem segja þarf um hvenær á að beygja til hægri og hvenær til vinstri.

11.c. Tré

Í upphafi þessa kafla var listi skilgreindur svona:

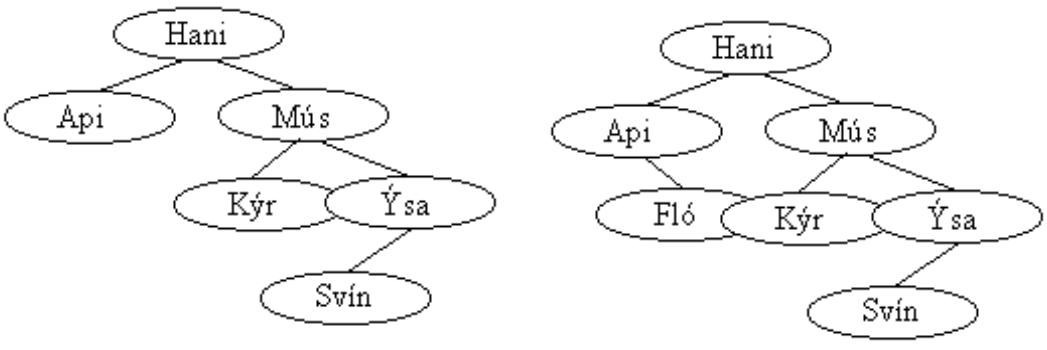
> Listi er null eða
> hnútur með lista fyrir aftan.

Tré er gagnagrind sem hægt er að skilgreina endurkvæmt svona:

> Tré er null eða
> hnútur með tré fyrir hægri grein og annað tré fyrir vinstri grein.

Gagnagrindur af þessu tagi eru merkilegar fyrir þá sök að auðvelt er að geyma röðuð gögn í þeim og fljótlegt er að leita í tré sem geymir röðuð gögn. Það er til dæmis auðvelt að láta tré geyma orð í stafrófsröð og bæta í það nýjum orðum án þess að röðin ruglist og án þess að þörf sé að færa orðin sem fyrir eru. (Eigi hins vegar að geyma orð sem standa í stafrófsröð í fylki er ekki hægt að bæta nýju orði í miðja röðina án þess að færa öll orð þar fyrir aftan um eitt sæti.)

Aðferðin sem notuð er til að bæta gögnum í raðað tré þannig að þau fari á réttan stað í röðina byggist á því að láta það sem er framar í röðinni fara til vinstri og það sem er ekki framar í röðinni fara til hægri. Hugsum okkur að tré geymi orðin „Hani“, „Mús“, „Api“, „Ýsa“, „Kýr“, „Svín“ og þau hafi verið sett í tréð í þessari röð. Þar sem orðið „Hani“ var sett fyrst er það í rót (upphafshnút) trésins. „Mús“ kom næst og þar sem það orð er ekki framan við „Hani“ í stafrófinu var það sett hægra megin við rótina. Þá kom „Api“ sem er framan við „Hani“ og lendir því vinstra megin við hann. Næst er „Ýsa“. Það orð lendir hægra megin við „Hani“. Þar er „Mús“ fyrir og þar sem „Ýsa“ er aftan við „Mús“ lendir orðið hægra megin við hnútinn sem geymir „Mús“. „Kýr fer svo hægra megin við „Hani“ og vinstra megin við „Mús“ og „Svín“ að síðustu hægra megin við „Hani“, hægra megin við „Mús“ og vinstra megin við „Ýsa“. Tréð er þá eins og myndin til vinstri sýnir. Sé orðinu „Fló“ bætt í það verður það eins og myndin til hægri.



Ef hver hnútur kann aðferð til að skrifa innihald sitt, sem lætur hann gera hlutina í þessari röð

> Ef til er hnútur til vinstri láta hann þá skrifa innihald sitt.
> Skrifa eigið innihald
> Ef til er hnútur til hægri láta hann þá skrifa innihald sitt.

þá skrifast orðin út í stafrófsröð.

Í trjánum á myndunum inniheldur fyrsti hnúturinn (rótin) orðið „Hani“. Þessi hnútur er í efstu línu trésins. Í annarri línu eru 2 hnútar. Í þriðju línu er rúm fyrir 4 hnúta þótt

á myndunum séu aðeins tveir og þrír. Í 4 línu geta verið allt að 8 hnútar o.s.frv. Í línu númer n geta því verið allt að 2^{n-1} hnútar og tré sem hefur n línur getur því haft allt að

$$1 + 2 + 4 + ... + 2^{n-1} = 2^0 + 2^1 + 2^2 + ... + 2^{n-1} = 2^n - 1 \approx 2^n \text{ hnúta.}$$

Tré sem geymir 1.000.000 atriði þarf því ekki að vera mjög djúpt. Það dugar að það sé 20 línur því $2^{20} > 1.000.000$. Sé slíku tré raðað með þeim hætti sem hér hefur verið gerð grein fyrir þá þarf ekki að skoða meira en 20 hnúta til að finna eitt einstakt atriði. Sé atriðið sem leitað er að framan við rót er farið til vinstri. Sé það aftan við gögnin í þeim hnút er næst beygt til hægri og svo koll af kolli þar til annað hvort gerist að það sem leitað er að finnst eða komið er niður á botn.

Forrit_11_04 er sýnir hvernig tré eru skilgreind og notuð. Forritið er myndað úr tveim klösum. Þeir heita `Trje` og `NotkunTrjes`. Klasinn `Trje` er að því leyti líkur klasanum `Listi` í forrit_10_04 að hann hefur klasabreytur af eigin tegund. En þar sem `Listi` hefur aðeins eina klasabreytu af tegundinni `Listi` hefur `Trje` tvær klasabreytur af tegundinni `Trje`. `Listi` í forrit_10_04 hafði eina klasabreytu af tegundinni `Object` til að geyma gögn. Klasinn `Trje` sem hér fer á eftir er gerður til að geyma strengi og klasabreytan gögn er því af tegundinni `String`.

Klasinn `NotkunTrejs` setur fjóra takka, einn textareit (`TextField`) og eitt textasvæði (`TextArea`) á myndflötinn. Í textareitinn er hægt að skrifa orð og þegar ýtt er á takkann sem merktur er „Bæta í tré“ er orðinu bætt í tréð. Þegar ýtt er á takkann sem merktur er „Prenta úr tré“ eru öll orðin sem geymd eru í trénu skrifuð í textasvæðið. Þegar ýtt er á takkann sem merktur er „Leita í tré“ er leitað í trénu að streng sem byrjar eins og strengurinn í textareitnum. Sé smellt á „Hreinsa“ er tréð tæmt.

Hér kemur klasinn `NotkunTrjes`. Hann skýrir sig að mestu sjálfur.

```
import java.awt.*;
import java.applet.Applet;
//
// Forrit_11_04
// NotkunTrjes
//
public class NotkunTrjes extends Applet
{
   Trje trje;
   TextField tf;
   TextArea ta;
   Button bBaetaITrje;
   Button bPrentaTrje;
   Button bLeitaITrje;
   Button bHreinsa;

   public void init()
   {
      tf = new TextField(20);
      ta = new TextArea(6, 20);
      bBaetaITrje = new Button("Bæta í tré");
      bPrentaTrje = new Button("Prenta úr tré");
      bLeitaITrje = new Button("Leita í tré");
      bHreinsa = new Button("Hreinsa");
      this.add(tf);
      this.add(bBaetaITrje);
      this.add(bPrentaTrje);
      this.add(bLeitaITrje);
      this.add(bHreinsa);
      this.add(ta);
   }
```

```
   public boolean action(Event e, Object o)
   {
      if (e.target == bBaetaITrje)
      {
         if (trje == null)
         {
            trje = new Trje(tf.getText());
         }
         else
         {
            trje.baetaITrje(tf.getText());
         }
         tf.setText("");
         return true;
      }
      if (e.target == bPrentaTrje)
      {
         StringBuffer sb = new StringBuffer();
         try  // Kastar NullPointerException ef trje == null.
         {
            trje.skrifaIStringBuffer(sb);
            ta.setText(sb.toString());
         }
         catch (NullPointerException fravik)
         {
            ta.setText("Það er ekkert tré til að prenta.")  ;
         }
         return true;
      }
      if (e.target == bLeitaITrje)
      {
         try  // Kastar NullPointerException ef trje == null.
         {
            String s = trje.leitaITrje(tf.getText());
            if (s != null)
            {
               ta.setText(s);
            }
            else
            {
               ta.setText("*** FINNST EKKI ***");
            }
         }
         catch (NullPointerException fravik)
         {
            ta.setText("Það er ekkert tré til að leita í.") ;
         }
         return true;
      }
      if (e.target == bHreinsa)
      {
         trje = null;
         ta.setText("");
         tf.setText("");
         return true;
      }
      return false;
   }
} // NotkunTrjes endar
```

Hér á eftir fer svo klasinn `Trje`.

```
//
// Forrit_11_04
// Trje
//
public class Trje       // Tré sem geymir strengi.
{                       // Hver hnútur vísar á
   String gogn;         // tvo aðra sem heita
   Trje vinstri;        // vinstri og haegri
   Trje haegri;

   public Trje(String s)
   {
      gogn = s;
      vinstri = null;
      haegri = null;
   }

   // Trje kann þrjár aðferðir sem allar byggjast á endurkomu.
   public void baetaITrje(String s)       // Bætir einum streng, s,
   {                                      // í tréð. Ef s er framar
      if (s.compareTo(gogn) < 0)          // en gogn í stafrófinu fer
      {                                   // það til vinstri í trénu.
         if (vinstri == null)             // Ef ekki er nein vinstri
         {                                // grein er hún búin
               vinstri = new Trje(s);     // til og s sett á hana
         }                                // annars er s bætt á
         else { vinstri.baetaITrje(s); }  // vinstri greinina.
      }
      else                                // Ef s er ekki framar en
      {                                   // gogn í stafrófinu fer
         if (haegri == null)              // það til hægri í trénu.
         {
            haegri = new Trje(s);
         }
         else { haegri.baetaITrje(s); }
      }
   }

   // Þessi aðferð setur alla strengi í trénu í einn
   // StringBuffer og setur "\n" á milli þeirra.
   // Til að strengirnir lendi í stafrófsröð
   // er byrjað á vinstri helmingi trésins.
   public void skrifaIStringBuffer(StringBuffer sb)
   {
      if (vinstri != null) { vinstri.skrifaIStringBuffer(sb); }
      sb.append(gogn + "\n");
      if (haegri  != null) { haegri.skrifaIStringBuffer(sb); }
   }

   // Þessi aðferð leitar í streng sem byrjar á s
   // og skilar honum ef hann finnst. Ef hann finnst
   // ekki skilar hún null.
   public String leitaITrje(String s)
   {
      if (gogn.startsWith(s))
      {
         return gogn;
      }
```

```
      else if (s.compareTo(gogn) < 0)
      {
         if (vinstri != null)
         {
            return vinstri.leitaITrje(s);
         }
      }
      else
      {
         if (haegri != null)
         {
            return haegri.leitaITrje(s);
         }
      }
      return null;
   }
}
```

Verkefni 11.7*
Keyrðu klasann `NotkunTrjes` og prófaðu að vista nokkur orð, leita að einhverju þeirra og skrifa þau út.

Verkefni 11.8*
Búðu til tré sem geymir ekki strengi heldur tölur og búðu svo til klasa sem prófar notkun þess með sama hætti og `NotkunTrjes` í forrit_11_04 prófar notkun trés sem geymir strengi.

Verkefni 11.9
Notaðu lausnina á verkefni 11.8 til að búa til forrit sem les skrá af heiltölum, setur þær í raðað tré og skrifar sömu tölur í aðra skrá þar sem þeim er raðað eftir stærð.

Verkefni 11.10
Breyttu klasanum `Trje` þannig að hann geymi strengi í réttri íslenskri stafrófsröð. (Ábending: notaðu aðferðirnar úr kafla 8.c.)

11.x. Spurningar og umhugsunarefni

1. Af hverju er eftirfarandi skilgreining á lista kölluð „endurkvæm skilgreining“?

   Listi er null eða
   hnútur með lista fyrir aftan.

2. Hvernig er hægt aðskilgreina a^n (þar sem n er heil tala og $n \geq 0$) með endurkvæmri skilgreiningu?
3. Hvernig er hægt að skilgreina forföður með endurkvæmri skilgreiningu?
4. Hvað er átt við þegar sagt er að aðferð noti endurkomu?

5. Gerðu ráð fyrir að aðferðin `foo` sé skilgreind svona og `t1` og `t2` séu af tegundinni `TextArea`.

```
public void foo(int n)
{
   t1.appendText("n = " + n);
   if (n > 0)
   {
      foo(n - 2);
   }
   t2.appendText("n = " + n);
}
```

   Hvað birtist í `t1` og `t2` ef gefin er skipunin `foo(6)`?

6. Af hverju nota aðferðir sem byggja á endurkomu mikið minnispláss?
7. Hvaða tvo mikilvæga kosti hafa gagnagrindur af gerðinni tré?
8. Hvaða hnútur í tré er kallaður „rót“?
9. Hvað þarf tré að vera djúpt (margar línur) til að geta geymt 100 hluti?
10. Í upphafi 11.c var rakið hvernig raðað tré lítur út ef eftirtalin orð eru sett í það í þessari röð

    Hani Mús Api Ýsa Kýr Svín.

    Hvernig yrði tréð ef orðin væru sett í það í þessari röð

    Ýsa Kýr Svín Hani Mús Api.

    En ef þau væru sett í þessari röð

    Api Hani Kýr Mús Svín Ýsa.

Til umhugsunar

Ef þú hefur svarað spurningu 10 hér að ofan hefur þú væntanlega áttað þig á að ef hlutir eru settir í tré í þeirri röð sem þeir eiga að koma úr því, þá verður tréð ekkert eiginlegt tré, heldur listi.

Sagt er að tré sé í jafnvægi ef hver hnútur hefur hægri grein og vinstri grein sem næst jafn stórar. Tré sem hefur eingöngu hægri greinar eða eingöngu vinstri greinar er í rauninni listi og meira ójafnvægi getur ekki verið á einu tré.

Getur þú sýnt fram á að ef tré er í jafnvægi þá sé dýpt þess um það bil $\log_2(n)$ þar sem n er fjöldi hnúta?

Hver verður dýpt trés sem hefur n hnúta ef gögnin eru sett í það í þeirri röð sem þau eiga að koma út úr því (t.d. í stafrófsröð ef tréð geymir orð og setur þau til vinstri eða hægri við hnút eftir því hvort þau eru framar en hann í stafrófinu)?

Tíminn sem tekur að leita að hlut í röðuðu tré er í hlutfalli við dýpt þess. Hvað ætli taki mörgum sinnum lengri tíma að leita í lista (eða tré sem er í mesta hugsanlegu ójafnvægi) heldur en tré sem er í fulkomnu jafnvægi?

Hvernig er hægt að tryggja að tré sé í jafnvægi?

Ætli það sé hægt að búa til aðferð sem breytir tré sem ekki er í jafnvægi í tré sem er í jafnvægi?

12. kafli: Samskipti tölva á neti

12.a. Fróðleikur um Internetið: TCP/IP, umdæmisheiti, IP tölur o.f.l.

Í pakkanum `java.net` eru klasar til að senda gögn milli tölva gegnum Internetið eða önnur net sem nota TCP/IP samskiptastaðalinn. Til að nota þessa klasa þarf dálitla þekkingu á tölvunetum. Hér verður gerð stutt grein fyrir helstu hugtökum sem notuð eru í umfjöllun um net og nokkrum mikilvægum fróðleiksatriðum.

Net, Internet, innra net

Net er margar tölvur sem eru tengdar saman og geta sent gögn sín á milli.

Staðarnet er tölvur sem eru tengdar saman um skamman veg (yfirleitt innan sömu byggingar). Víðnet er margar tölvur sem eru tengdar saman um langan veg. Lottóvélar Íslenskrar getspár mynda víðnet. **Internetið** er víðnet sem nær um allan heim. Sérstaða Internetsins er einkum fólgin í því að:

Það tengir saman ólíkar vélar (PC tölvur, Macintosh, UNIX vinnustöðvar o.fl.).
Það leyfir ólíkar tengingar (símasamband, fastar línur af ýmsu tagi).
Allar vélar á netinu eru jafnréttháar, það er engin miðstöð og engin vél er ómissandi.
Allir geta tengst því.
Enginn á það eða ræður yfir því.

Hægt er að nota internetsamband milli tölva á ótal vegu. Algengustu þjónustuflokkar eru tölvupóstur, vefur, ráðstefnukerfi, spjallrásir (IRC) og FTP.

Innra net er staðarnet sem býður upp á sams konar þjónustu og Internetið. Víða eru staðarnet sett upp sem innri net með fast samband við Internetið. Þá er innra netið tengt við beini (router) eða gátt (gateway) sem er í sambandi við Internetið og miðlar gögnum milli innra netsins og Internetsins.

TCP/IP

Til að tvær eða fleiri tölvur geti skipst á boðum þurfa þær að fylgja sömu reglum um framsetningu (á hvern hátt eru boðin skrifuð), merkingar (hvernig er gefið til kynna hver sendi þau og hver á að fá þau) o. fl. Slíkar reglur kallast **samskiptastaðlar**. Mörg staðarnet þar sem netþjónar keyra Novell stýrikerfið nota t.d. samskiptastaðal sem kallast IPX. Net Macintosh tölva nota oft samskiptastaðal sem heitir AppleTalk. Internetið notar tvo staðla sem kallast **IP** (Internet Protocol) og **TCP** (Transmission Control Protocol). Net sem nota þessa staðla (hvort sem þau eru innri net eða hluti Internetsins) eru stundum kölluð **TCP/IP** net.

IP staðallinn segir hvernig á að senda gögn og merkja þau réttum viðtakanda. Aðferðin er í stuttu máli sú að skipta sendingu (t.d. texta eða mynd) í pakka sem innihalda runu af táknum og bæta framan við hvern pakka nokkrum táknum sem segja m.a. hver á að fá hann, hver sendi hann og hvað eru mörg tákn (bæti) í honum. **TCP** staðallinn kveður m.a. á um hvernig vél sem tekur við gögnum eigi að senda boð til baka um að móttaka hafi heppnast og hvernig eigi að fara fram á að gögn séu endursend ef þau hafa ekki komist óbrengluð til skila

IP tölur og umdæmisheiti

Til að tölvur geti skipst á boðum eftir TCP/IP staðli verða þær hver og ein að hafa einkennisnúmer sem kallast **IP tala**. Á sama neti mega engar tvær tölvur hafa sömu IP tölu.

IP tölur eru yfirleitt ritaðar með 4 tölum á bilinu 0 til 255 og hafður punktur á milli. Til dæmis hefur ein af vélum Íslenska menntanetsins IP töluna 193.4.232.109

Þótt engar tvær tölvur á Internetinu geti haft sömu IP tölu er ekkert því til fyrirstöðu að tvær tölvur sem tilheyra hvor sínu innra neti hafi sömu IP tölu. Það er til siðs að láta tölvur á innrinetum hafa IP tölur sem eru ekki notaðar á Internetinu. IP tölur sem byrja á 10 og 192 eru ekki notaðar á Internetinu. Innri net nota því yfirleitt IP tölur sem byrja á 10 eða 192. IP tölur sem byrja á 127 eru hvorki notaðar á Internetinu né á innri netum. Tölva sem tengd er á TCP/IP net notar tölu sem byrjar á 127 (oftast 127.0.0.1) til að vísa á sig sjálfa. Það er semsagt sama hvaða IP tölu vél hefur, ef hún er beðin að senda boð til vélar númer 127.0.0.1 þá sendir hún sjálfri sér þau. Þetta er stundum notað til að láta tvö forrit sem eru í gangi á sömu vél skiptast á boðum.

Hægt er að rita 4 bæti (þ.e. fjórar tölur milli 0 og 255) á $256^4 = 4.294.967.296$ vegu. Á Internetinu er því pláss fyrir rúmlega 4 milljarða véla. Þetta rúm nýtist ekki nema að hluta til því IP tölum er úthlutað í blokkum. Stofnun fær kannski til ráðstöfunar allar IP tölur sem byrja á 167.1. Þetta eru $256^2 = 65.536$ tölur. Ef stofnunin rekur 536 tölvur á Internetinu eru 65.000 af þessum tölum ónotaðar. Til að tryggja rúm fyrir nógu margar vélar er í ráði að taka upp nýjan staðal fyrir IP tölur og rita þær með 16 bætum (16 tölum milli 0 og 255). Með því móti verður rúm fyrir um það bil $3.4 * 10^{38}$ vélar. Forrit sem nota IP tölur ættu að virka jafn vel hvort sem tölurnar eru myndaðar úr 4 bætum eða 16, annars verða þau ónothæf þegar nýr staðall fyrir IP tölur tekur gildi.

Það er erfitt fyrir fólk að muna IP tölur. Þess vegna er flestum vélum, sem hafa fast samband við Internetið, gefið nafn. Slíkt nafn kallast lén eða **umdæmisheiti** (á ensku: „domain name“). Vélar sem ekki hafa fast samband við Internetið heldur tengjast aðeins stund og stund, t.d. gegnum mótald, hafa sumar ekki neitt nafn heldur aðeins IP tölu. Stundum hafa þær fasta IP tölu (þ.e. alltaf þá sömu) en algengast er að heimilistölvur sem tengjast með því að hringja í vél með fast samband fái IP tölu úthlutað um leið og þær ná sambandi og þá ekki alltaf sömu IP tölu. Þetta er allt í lagi ef aðeins á að nota sambandið til að sækja póst, gramsa á vef eða tengjast spjallrás en eigi tölva að geyma gögn, t.d. vefsíður, og veita öðrum aðgang að þeim þarf hún að hafa fasta IP tölu, vera stöðugt í sambandi og helst að hafa umdæmisheiti.

Sama vél getur haft mörg nöfn (alveg eins og einn sími með eitt símanúmer getur verið skráð á mörg nöfn í símaskránni). Hér eru nokkur dæmi um umdæmisheiti og samsvarandi IP tölur.

www.isholf.is	194.105.226.1
www.simnet.is	194.105.226.1
pop.ismennt.is	212.30.217.11
www.ismennt.is	193.4.232.109
ftp.ismennt.is	193.4.1.153.
www.fva.is	193.4.7.194
www.microsoft.com	207.46.130.14
www.apple.com	17.254.0.91

Sé tölva á Internetinu beðin um samband við 193.4.232.109 þá nær hún sambandi við eina af vélum Íslenska menntanetsins en sé hún beðin um samband við www.ismennt.is þarf hún fyrst að komast að því hvaða númer samsvarar þessu nafni.

Til þess að finna hvaða IP tala samsvarar umdæmisheiti eins og rvik.ismennt.is eða www.apple.com hefur tölva samband við **umdæmisheitaþjónustu** (á ensku: „domain name server“, skst. „**DNS**“). **Umdæmisheitaþjónusta** sér um að fletta upp IP tölum fyrir aðrar tölvur. Til að tölva (t.d. heimilistölva með mótald) geti tengst Internetinu og náð sambandi við vélar sem nefndar eru nöfnum, en ekki bara IP tölum, þarf hún að "vita" IP tölu a.m.k. einnar tölvu sem keyrir **umdæmisheitaþjónustu** eða **DNS**.

Íslensk umdæmisheiti enda á is, sem er skammstöfun landsins. Síðasti liður umdæmisheitis er tveggja stafa skammstöfun sem segir til um í hvaða landi netfangið er skráð nema ef það er í Bandaríkjunum þá er hann þriggja stafa skammstöfun sem segir til um hvaða flokki stofnunin sem á tölvuna tilheyrir. Dæmi:

dk	Danmörk
is	Ísland
uk	Bretland
edu	Menntastofnun (skráð í Bandaríkjunum)
mil	Stofnun á vegum bandarískra hersins
gov	Stofnun á vegum bandarískra ríkisins
com	Ýmis fyrirtæki

Oftast er næstsíðasti liður umdæmisheitis nafn (eða skammstöfun á nafni) fyrirtækis eða stofnunar. Þannig hafa allar vélar Íslenska menntanetsins nafn sem endar á ismennt.is og vélar Háskóla Íslands hafa nafn sem endar á hi.is.

Á vélum sem miðla vefsíðum er fyrsti liður umdæmisheitis oft www (fyrir world wide web) eins og í www.ismennt.is og www.microsoft.com. Vélar sem sinna póstþjónustu hafa oft umdæmisheiti sem byrjar á pop eða mail eins og pop.ismennt.is og vélar sem bjóða upp á skráaflutning með ftp hafa gjarna umdæmisheiti sem hefst á ftp eins og ftp.ismennt.is.

URL og CGI-bin forrit

Sé vefsíða geymd í skrá sem heitir karamella.html í efnisskránni /nammi á vélinni www.fva.is þá er **veffang** (á ensku: „uniform resource locator“ skst. „**URL**“) hennar

```
http://www.fva.is/nammi/karamella.html
```

Þar sem www.fva.is stendur fyrir IP töluna 193.4.7.194 má eins rita veffangið

```
http://193.4.7.194/nammi/karamella.html
```

Veffang byrjar á skammstöfun sem segir til um hvers konar gögn er um að ræða og með hverjum hætti þau skuli flutt. Algengustu tegundir eru:

file	er skrá á eigin diski sem miðlað er af stýrikerfi vélarinnar.
ftp	(„file transfer protocol“) er skrá sem miðlað er af FTP þjóni.
http	(„hypertext transfer protocol“) er vefsíða sem miðlað er af vefþjóni.
news	er Usenet ráðstefna sem miðlað er af fréttaþjóni.
telnet	er tenging með skjáhermi.

Á eftir skammstöfun (sem oftast er „http“) kemur tvípunktur og tvö skástrik og svo nafn vélar (annað hvort IP tala eða umdæmisheiti), þá vísun í skrá sem geymd er á vélinni.

Á tölvum sem keyra stýrikerfið UNIX eða Linux fær hver notandi efnisskrá sem hægt að finna með því að setja ~ framan við nafn hans. Notandi sem tengist www.fva.is undir nafninu atli á til dæmis efnisskrá sem hægt er að komast í með því að biðja um ~atli. Vefsíður sem atli setur í efnisskrána sína er því hægt að nálgast með því að gefa upp veffangið http://www.fva.is/~atli/

Stundum vísar veffang ekki á venjulega vefsíðu heldur svokallað **CGI** (Common Gateway Interface) forrit. Það er forrit sem keyrir á sömu vél og vefsíður eru geymdar á. Ef vefsjá sendir vefþjóni URL sem vísar á slíkt forrit þá ræsir vefþjónninn forritið og sendir vefsjánni þann texta (streng) sem það skrifar út eða skilar.

URL sem vísar á CGI-bin forrit er ritað þannig að á eftir nafni forritsins kemur „?“ (spurningamerki) og þar fyrir aftan strengur sem forritið á að lesa. Dæmi:

```
http://www.lycos.com/cgi-bin/pursuit?query=Akranes
```

Þetta veffang vísar á forritið pursuit sem er leitarvél sem geymd er í efnisskránni /cgi-bin á tölvunni http://www.lycos.com og sendir forritinu strenginn "query=Akranes". Forritið bregst við með því að búa til vefsíðu (þ.e. streng með HTML kóða) með krækjum í síður um Akranes og láta vefþjóninn á http://www.lycos.com senda hana um hæl.

Strengir sem eru sendir CGI-bin forriti mega aðeins innihalda bókstafi úr enska stafrófinu og tölustafi. Eigi að biðja leitarvélina hjá www.lycos.com að leita að upplýsingum um Flateyri við Önundarfjörð er beðið um URL- ið

```
http://www.lycos.com/cgi-bin/pursuit?query= Flateyri+vi%F0+%D6nundarfj%F6r%F0
```

Í stað orðabila kemur + og í stað annarra tákna sem ekki tilheyra enska stafrófinu (t.d. séríslenskra stafa) kemur prósentumerki og svo númer táknsins ritað með hexadesímaltölum. Strengir sem eru kóðaðir svona til að skeyta aftan við URL sem vísar á CGI-bin forrit kallast **URL-kóðaðir** (á ensku: „URL-encoded“).

Miðlarar, biðlarar og hlið

Forrit sem miðlar gögnum til annarra véla, t.d. pósti eða vefsíðum kallast **miðlari** (á ensku: „**server**“). Vélin www.fva.is keyrir til dæmis vefmiðlara og miðlar vefsíðum til annarra véla.

Forrit sem nýtir sér þjónustu miðlara kallast **biðlari** (á ensku: „**client**“). Flestar heimilistölvur sem tengjast Internetinu um mótald keyra aðeins biðlara, þær nota sér þjónustu annarra véla en láta ekkert í té.

Sama vél getur keyrt bæði miðlara og biðlara. Munurinn á þessum forritaflokkum er sá að miðlararnir bíða eftir að biðlari hafi sambandi við sig og eru þá til þjónustu reiðubúnir en eiga aldrei upphaf að neinum samskiptum. Biðlarar eru forrit sem hafa samband við miðlara, yfirleitt til að biðja um einhverja þjónustu, t.d. að fá senda vefsíðu eða mynd.

Tæknilega er ekkert því til fyrirstöðu að venjuleg heimilistölva sem keyrir Windows 95 eða Mac OS keyri miðlara. Hún getur til dæmis keyrt vefþjón og veitt öðrum vélum aðgang að vefsíðum sem geymdar eru á diski hennar. Yfirleitt er þó lítið vit í að láta vél bjóða upp á þjónustu nema hún sé í stöðugu sambandi og hafi a.m.k. fasta IP tölu og helst líka umdæmisheiti. Það getur enginn notað þjónustuna nema

hann nái sambandi við vélina og það er vandkvæðum bundið ef hún er ekki tengd nema við og við eða hefur eina IP tölu í dag og aðra á morgun.

Tölvur sem veita þjónustu á Internetinu keyra yfirleitt marga miðlara. Hver þeirra hefur númer sem er kallað **hlið** (á ensku: „port“). Til að ná sambandi við miðlara þarf biðlari bæði að vita IP tölu (eða umdæmisheiti) vélarinnar og númerið á hliðinu sem miðlarinn situr við.

Hlið með númer neðan við 1024 eru frátekin fyrir algengar gerðir miðlara á Internetinu, dæmi:

20	fyrir ftp gögn
21	fyrir ftp skipanir
25	til að senda póst
80	fyrir vefsíður
110	til að sækja póst
119	fyrir Usenet ráðstefnur

Númer frá 1024 og upp úr (upp í 65535) mega forritarar nota að vild.

Í undantekningartilvikum eru vefsíður afgreiddar um annað hlið en númer 80. Þá er hægt að nálgast þær með því að rita númer hliðsins á eftir umdæmisheitinu. Dæmi:

http://www.afbrigdileg_vjel.is:81/nammi/karamella.html

Verkefni 12.1*

Finndu einhverja leitarvél á vefnum og prófaðu að leita að einhverju (t.d. Flateyri við Önundarfjörð) og taktu eftir því hvernig strengurinn er kóðaður. (Í Netscape og Internet Explorer sést hann í línunni þar sem slóðin er rituð.)

12.b. `InetAddress`

Forrit_12_01 finnur IP tölu vélar ef umdæmisheiti er gefið. Til að það virki þarf að vera í sambandi við Internetið.

Forritið er sjálfstætt, enda hafa `Applet`, sem keyrð eru í vefsjá, aðeins mjög takmarkaðan aðgang að Internetinu, geta raunar bara haft samband við þá tölvu sem þau eru sótt af. Klasinn `Starta` er nánast eins og samnefndur klasi í forrit_09_01 og forritunum í kafla F.

```
//
// forrit_12_01
// Starta
// Byggt á samnefndum klasa í forrit_09_01
//
public class Starta
{
   public static void main(String args[])
   {
      FinnaIPTolu s = new FinnaIPTolu("IP-tölu snuðrari.");
      s.init();
      s.resize(250, 200);
      s.move(50, 100);
      s.show();
   }
```

```
}
```

Klasinn `FinnaIPTolu` notar hluti af tegundinni `InetAddress` sem er í pakkanum `java.net`. Slíkir hlutir hafa samband við umdæmisheitaþjónustur á Internetinu til að finna IP tölu ef þeir fá umdæmisheiti.

Klasabreytan `thessiTolva` er af tegundinni `InetAddress` og er látin geyma IP tölu og umdæmisheiti tölvunnar sem forritið er keyrt á. Klasabreytan `onnurTolva` er af sömu tegund og hún er notuð til að finna IP-tölur annarra véla.

Klasinn `InetAddress` er óvenjulegur að því leyti að smiður hans er ekki `public` og þar með ekki aðgengilegur. Til að búa til hlut af tegundinni `InetAddress` er því ekki notaður smiður heldur `static` aðferðirnar `getLocalHost` og `getByName`, sem skila hlut af tegundinni `InetAddress`. Til að búa til hlut af tegundinni `InetAddress` sem geymir IP-tölu og umdæmisheiti vélarinnar sem forritið er keyrt á er hægt að nota skipunina `getLocalHost` svona

```
InetAddress i;
i = InetAddress.getLocalHost();
```

Eigi svo að setja umdæmisheiti og IP-tölu (á formi eins og "193.4.1.2") í breytur af tegundinni String er hægt að nota

```
String sIPTala = i.getHostAddress();
String sUmdaemisheiti = i.getHostName();
```

Til að finna IP tölu vélar með umdæmisheitið www.fva.is og geyma það í streng er hægt að nota

```
InetAddress j;
j = InetAddress.getByName("www.fva.is");
String s = j.getHostAddress();
```

Hér kemur svo klasinn `FinnaIPTolu`.

```
import java.awt.*;
import java.net.*;
//
// Forrit_12_01
// FinnaIPTolu
// Finnur IP tölur ef umdæmisheiti eru gefin.
//
public class FinnaIPTolu extends Frame
{
   // Klasinn InetAddress geymir umdæmisheiti og
   // IP tölu tölvu á Internetinu.
   InetAddress thessiTolva, onnurTolva;
   Button bHaetta;
   Button bFinnaIPTolu;
   Button bEiginIPTalaOgHeiti;
   TextField tUmdaemisheiti;
   TextField tIPTala;
   Label skilabod;


   FinnaIPTolu(String s)
   {
      super(s);
   }
```

```
public void init()
{
   FlowLayout f = new FlowLayout();
   this.setLayout(f);
   bFinnaIPTolu = new Button("Finna IP-tölu.");
   bEiginIPTalaOgHeiti = new Button("Eigin IP-tala og heiti");
   bHaetta = new Button("Hætta");
   tUmdaemisheiti = new TextField(20);
   tIPTala = new TextField(20);
   skilabod = new Label("Umdæmisheiti má skrifa í efri reitinn");
   this.add(skilabod);
   this.add(tUmdaemisheiti);
   this.add(tIPTala);
   this.add(bFinnaIPTolu);
   this.add(bEiginIPTalaOgHeiti);
   this.add(bHaetta);
}

public boolean action(Event e, Object o)
{
   if (e.target == bHaetta)
   {
      dispose();
      System.exit(0);
      return(true);
   }

   if (e.target == bFinnaIPTolu)
   {
      String umdaemisheiti = tUmdaemisheiti.getText();
      try
      {
         // InetAddress hefur engan public smið en í
         // staðinn static aðferðina getByName sem
         // tekur við umdæmisheiti sem streng og skilar
         // hlut af gerðinni InetAddress.
         onnurTolva = InetAddress.getByName(umdaemisheiti);
         // Aðferðin getHostAddress finnur IP-tölu
         // hlutar af gerðinni InetAddress. Til að aðferðin
         // virki þarf tölvan að vera í sambandi við
         // Internetið því hún notar umdæmisheitaþjónustu
         // á Internetinu til að finna hvaða IP-tala
         // samsvarar umdæmisheitinu sem hluturinn
         // (í þessu tilviki onnurTolva) geymir.
         String ipTala = onnurTolva.getHostAddress();
         tIPTala.setText(ipTala);
      }
      catch (UnknownHostException ue)
      {
         tIPTala.setText("Finn ekki " + umdaemisheiti);
      }
      return true;
   }

   if (e.target == bEiginIPTalaOgHeiti)
   {
      try
      {
         // Aðferðin getLocalHost skilar hlut af
         // gerðinni InetAddress sem inniheldur
         // umdæmisheiti og ip-tölu tölvunnar
         // sem forritið er keyrt á. Hún er static.
         thessiTolva = InetAddress.getLocalHost();
         String eiginIPTala = thessiTolva.getHostAddress();
         tIPTala.setText(eiginIPTala);
         // Aðferðin getHostName skilar umdæmisheiti.
```

```
            // Hún er static eins og fleiri aðferðir í
            // klasanum InetAddress.
            String eigidUmdaemisheiti = thessiTolva.getHostName();
            tUmdaemisheiti.setText(eigidUmdaemisheiti);
         }
         catch (UnknownHostException ue)
         {
            tIPTala.setText("Er ég í sambandi?");
         }
         return true;
      }
      return false;
   }

}
```

Verkefni 12.2*
Keyrðu forrit_12_01 og finndu út hvaða IP-tölu og umdæmisheiti vélin sem þú notar hefur. Finndu líka IP tölur nokkurra annarra véla sem þú veist umdæmisheiti á.

12.c. `URL` og `URLEncoder`

Forrit_12_02 tekur við veffangi (URL-i) í textareit. Forritið hefur annan textareit fyrir streng sem á að kóða með URL-kóðun og hengja aftan við veffangið. Það hefur einn takka sem merktur er „Sækja gögn“. Sé smellt á hann sækir forritið innihald skráarinnar sem veffangið vísar á og skrifar það í textasvæði. Annar takki er merktur „Upplýsingar um URL“ og sé smellt á hann skrifar forritið upplýsingar um veffangið í textasvæðið. Um leið og smellt er á „Sækja gögn“ er innihaldi seinni textareitsins breytt og skrifað í hann allt veffangið með kóðuðum streng fyrir aftan.

Verkefni 12.3*
Keyrðu forrit_12_02 og prófaðu eftirfarandi:

a) Að setja veffangið
http://www.ismennt.is/not/atli/data/visa.txt
í efri textareitinn og ýta fyrst á „Sækja gögn“ og svo á „Upplýsingar um URL“.

b) Að setja í efri textareitinn veffangið
http://www.lycos.com/cgi-bin/pursuit?
(sem vísar á leitarvél) og í hinn einhvern streng t.d.
Flateyri við Önundarfjörð
og smella svo á „Sækja gögn“.

Forrit_12_02 er samsett úr þrem klösum `Starta`, `Villumelding` og `KannaURL`. `Starta` er eins og samnefndur klasi í forrit_12_01 og `Villumelding` er eins og samnefndur klasi í forrit_0F_03. `Starta` og `Villumelding` eru ekki prentaðir hér.

Klasinn `KannaUrl` fer hér á eftir. Hann notar þrjá klasa sem tilheyra pakkanum `java.net`. Þeir eru `URL`, `URLEncoder` og `InetAddress`. Sá síðastnefndi var kynntur í kafla 12.b. Hann er notaður í aðferðinni `skrifaUpplUmURL` og notkun hans þar skýrir sig væntanlega sjálf.

Klasinn `URLEncoder` er notaður í `action`-aðferðinni þar sem brugðist er við ef smellt er á hnappinn `bLesaURL`. Það eru ekki búnir til neinir hlutir af gerðinni `URLEncoder` enda hefur klasinn aðeins eina aðferð sem er `static` og heitir `encode`. Hún er til að URL-kóða streng. Ef gefnar eru skipanirnar

```
String s1 = "Flateyri við Önundarfjörð";
String s2 = URLEncoder.encode(s1);
```

þá fær `s2` gildið "Flateyri+vi%F0+%D6nundarfj%F6r%F0". Í stað orðabila eru komnir plúsar og í stað sérískensku stafanna eru prósentumerki með tveggja stafa hexadesímaltölum fyrir aftan.

Þegar smellt er á `bLesaURL` er strengurinn í neðri textareitnum (`tTilCGIForrits`) URL-kóðaður og útkoman úr því hengd aftan við strenginn í efri reitnum (`tURL`) og strengurinn sem út úr því kemur sendur aðferðinni `lesaGognFraURL`. Aðferðin tekur líka við textasvæði (`tSvar`) til að skrifa í. Hún opnar inntaksstraum frá veffangi og skrifar allt sem úr honum rennur í textasvæði. Aðferðin `lesaGognFraURL` skýrir sig annars að mestu sjálf.

Klasinn `URL` var kynntur lítillega í kafla F.d. Aðferðin `skrifaUpplUmURL`, sem framkvæmd er þegar smellt er á takkann `bUpplysingarUmURL`, sýnir hvernig hægt er að nota aðferðir í klasanum `URL` til að fá upplýsingar um veffang. Notkun þessara aðferða ætti að vera nokkuð ljós af samhenginu.

```
import java.awt.*;
import java.net.*;
import java.io.*;
//
// forrit_12_02
// KannaURL
//
public class KannaURL extends Frame
{
   Button bHaetta;
   Button bLesaURL;
   Button bUpplysingarUmURL;
   TextField tURL;
   TextField tTilCGIForrits;  // Strengur sem þarf að kóða
   TextArea tSvar;
   Label skilabod1;
   Label skilabod2;

   KannaURL(String s)
   {
      super(s);
   }

   public void init()
   {
      FlowLayout f = new FlowLayout();
      this.setLayout(f);
      bLesaURL = new Button("Sækja gögn");
      bUpplysingarUmURL = new Button("Upplýsingar um URL");
      bHaetta = new Button("Hætta");
      tURL = new TextField(40);
      tTilCGIForrits = new TextField(70);
      tSvar = new TextArea(15, 70);
      skilabod1 = new Label("Skrifaðu URL hér");
      skilabod2 = new Label("Ef URL vísar á CGI-bin
                        forrit fær það strenginn sem er hér.");
      this.add(skilabod1);
```

```
      this.add(tURL);
      this.add(skilabod2);
      this.add(tTilCGIForrits);
      this.add(bLesaURL);
      this.add(bUpplysingarUmURL);
      this.add(bHaetta);
      this.add(tSvar);
   }

   void lesaGognFraURL(String s, TextArea t) // Les skrána sem
   {                                         // s vísar á og
      DataInputStream dLes;                  // skrifar inni-
      URL u;                                 // hald hennar í t.
      try
      {
         u = new URL(s);
         // openStream í klasanum URL býr til straum úr veffangi.
         dLes = new DataInputStream(u.openStream());
         String lina = dLes.readLine();
         while (lina != null)
         {
            t.appendText(lina + "\n");
            lina = dLes.readLine();
         }
         dLes.close();
      }
      // Smiðurinn URL kastar fráviki af tegundinni
      // MalformedURLException sem er undirklasi
      // IOException.
      //   Klasinn Villumelding úr forrit_0F_03 er
      // notaður til að birta skilaboð ef fráviki er
      // kastað.
      catch (MalformedURLException fravik)
      {
         Villumelding v = new Villumelding("Rangt myndað URL");
      }
      // Aðferðin readLine kastar IOException.
      catch (IOException fravik)
      {
         Villumelding v = new Villumelding(fravik.toString());
      }
      catch (Exception fravik)
      {
         Villumelding v = new Villumelding(fravik.toString());
      }
   }
```

```
void skrifaUpplUmURL(String s, TextArea t)   // Hlutar URL sem er
                                             // skrifað í s sundur
{                                            // í parta og skrifar
   InetAddress tolva;                        // upplýsingar um þá
   URL u;                                    // í t.
   try
   {
      u = new URL(s);
      t.appendText("URL = " + u + "\n");
      t.appendText("Samskiptastaðallinn er = " +
                                   u.getProtocol() + "\n");
      t.appendText("Umdæmisheiti vefþjóns = " +
                                   u.getHost() + "\n");

      // Klasinn InetAddress sem kynntur var í forrit_12_01
      // er hér notaður til að finna IP tölu vefþjóns.
      tolva = InetAddress.getByName(u.getHost());
      String ipTala = tolva.getHostAddress();
      t.appendText("IP tala vefþjóns = " + ipTala + "\n");

      t.appendText("Portið er = " + u.getPort() + "\n");
      t.appendText("Skráin er = " + u.getFile() + "\n");
      t.appendText("Akkerið er = " + u.getRef() + "\n");
   }

   // Smiðurinn URL kastar fráviki af tegundinni
   // MalformedURLException sem er undirklasi
   // IOException.
   //   Klasinn Villumelding úr forrit_0F_03 er
   // notaður til að birta skilaboð ef fráviki er
   // kastað.
   catch (MalformedURLException fravik)
   {
      Villumelding v = new Villumelding("Rangt myndað URL.");
   }
   catch (UnknownHostException ue)
   {
      Villumelding v = new Villumelding("Vefþjónn finnst ekki.");
   }
}


public boolean action(Event e, Object o)
{
   String URLStrengur;
   String tilCGIStrengur;

   if (e.target == bHaetta)
   {
      dispose();
      System.exit(0);
      return(true);
   }

   if (e.target == bLesaURL)
   {
      // Innihald textareita sett í strengjabreytur.
      URLStrengur = tURL.getText();
      tilCGIStrengur = tTilCGIForrits.getText();
      // Ef seinni textareiturinn er ekki tómur,
      // þ.e. ef senda á kóðaðan streng til
      // CGI-bin forrits, þá er hann kóðaður og
      // skeytt aftan við fyrri strenginn (sem
      // geymir URL og ? og einhverja skipun til
      // CGI-bin forrits.
      if (tilCGIStrengur.length() > 0)
```

```
            {
               tilCGIStrengur = URLEncoder.encode(tilCGIStrengur);
               URLStrengur += tilCGIStrengur;
            }
            tSvar.setText("");

            // Strengurinn sem sendur verður aðferðinni
            // lesaGognFraURL birtur í seinni texta
            // reitnum (þeim sem skrifað er í hvað
            // senda skal CGI-bin forriti).
            tTilCGIForrits.setText(URLStrengur);
            lesaGognFraURL(URLStrengur, tSvar);
            return true;
         }

         if (e.target == bUpplysingarUmURL)
         {
            tSvar.setText("");
            URLStrengur = tURL.getText();
            skrifaUpplUmURL(URLStrengur, tSvar);
            return true;
         }
         return false;
      }
   }
```

12.d. `Applet` og samskipti þeirra við vefþjón

Þótt `Applet` hafi fremur takmarkaðan aðgang að Internetinu er hægt að láta þau opna vefsíður, sækja myndir og hljóð af vélinni sem þau eru vistuð á og vinna ýmis verk í samspili við vefsjár.

Forrit_12_03 sýnir hvernig hægt er að láta `Applet` birta mynd, spila hljóð (úr au-skrá) og sækja vefsíður. Til að keyra forritið þarf að hafa eftirtaldar skrár í sömu efnisskrá og klasana `AppletSaekirGogn` og `Villumelding`:

gelt.au	Hljóðskrá
hundur.gif	Mynd
hundur.html	Vefsíða
testa_forrit_12_03.html	Vefsíða

Þegar búið er að þýða klasana er hægt að keyra forritið með því að hlaða testa_forrit_12_03.html inn í vefsjá.

Verkefni 12.4
Prófaðu að keyra forrit_12_03 fyrst af diski í eigin vél og svo af neti. Til að keyra forritið af neti þarftu fyrst að koma klösunum, og skránum sem taldar voru upp hér að ofan, fyrir á vél sem keyrir vefþjón, annað hvort á innra neti eða úti á Internetinu.

Klasinn `Villumelding` í forrit_12_03 er eins og í forrit_0F_03 og fleiri forritum og er því ekki prentaður hér. Klasinn `AppletSaekirGogn`, sem er meginhluti forritsins fer hér á eftir.

Klasinn hefur eina klasabreytu af tegundinni `AppletContext`, sem er viðmót en ekki klasi. Hlutur af þessari tegund gerir `Applet`-i mögulegt að láta vefsjá sækja myndir, hljóð eða heilar vefsíður. Ekki er til smiður fyrir þessa tegund en það er hægt að sækja `AppletContext` með aðferðinni `getAppletContext` í klasanum `Applet`. Við getum litið svo á að þessi aðferð skili hlut sem inniheldur vefsjána sem `Applet`-ið er

keyrt í og tekur á sig viðmótið `AppletContext` (þ.e. getur framkvæmt þær aðferðir sem viðmótið inniheldur). Í forrit_12_03 er `getAppletContext`-aðferðin notuð (í aðferðinni `init`) til að gefa breytunni `ac`, sem er af tegundinni `AppletContext`, gildi.

`AppletContext`-ið sem geymt er í ac er svo notað til að sækja mynd, hljóð og heila vefsíðu.

Skipanirnar til að sækja myndina eru inni í `init`-aðferðinni. Þær eru

```
URL mynd = new URL(this.getCodeBase(), "hundur.gif");
hundur = ac.getImage(mynd);
```

Sú fyrri býr til veffang, `URL`, sem vísar á myndina. Smiðurinn `URL` tekur við veffangi og streng. Hann hagar sér þannig að ef hann fær t.d. veffangið

http://www.ismennt.is/not/atli/data/visa.txt

og strenginn "skrimsli.int" þá býr hann til veffangið

http://www.ismennt.is/not/atli/data/skrimsli.int

Hann klippir sem sagt skráarnafnið af veffanginu og setur strenginn í staðinn.

Strengurinn er einfaldlega nafnið á myndinni. Veffangið vísar á staðinn þar sem hana er að finna. Það er fundið með aðferðinni `getCodeBase` sem tilheyrir klasanum `Applet`. Þessi aðferð skilar `URL`-i sem vísar á `Applet`-ið sjálft. Einnig hefði mátt nota aðferðina `getDocumentBase` sem skilar `URL`-i sem vísar á vefsíðuna sem `Applet`-ið er á. Hér er búið til veffang sem er eins og veffang sem vísar á `Applet`-ið nema hvað nafnið „hundur.gif“ er komin í stað nafnsins á `Applet`-inu.

`AppletContext` hluturinn, `ac`, er hér látinn nota `getImage` aðferð sína til að sækja myndina. Þessi aðferð hagar sé næstum eins og samnefnd aðferð í klasanum `Applet`, munurinn er sá að aðferðin sem tilheyrir `AppletContext` notar flýtiminni og bráðabirgðageymslur („cache“) vefsjárinnar og getur því verið töluvert fljótvirkari.

Aðferðin `getImage` setur í gang þráð sem sækir myndina. Það er því engan vegin tryggt að búið sé að sækja myndina þegar kemur að því að framkvæma næstu skipun fyrir neðan skipunina

```
hundur = ac.getImage(mynd);
```

Skipanirnar um að sýna myndina eru í `paint`-aðferðinni og því framkvæmdar í hvert sinn sem myndflötur af `Applet`-sins er uppfærður á skjánum Til að tryggja að ekki sé reynt að sýna myndina fyrr en búið er að sækja hana er aðferðin `prepareImage` í klasanum `Applet` notuð. Sú aðferð skilar `true` ef mynd er tilbúin til birtingar.

Í aðferðinni `action` eru aðferðir `AppletContext` til að sækja hljóð og sýna vefsíðu notaðar.

Hljóðið er sótt þegar smellt er á takkann `bSaekjaHljod`. Veffangið er myndað eins og veffang myndarinnar. Það er geymt í klasabreytu af tegundinni `AudioClip` og sótt með aðferðinni `getAudioClip`. Eins og `getImage` hefur `getAudioclip` aðferð klasans `AppletContext` það fram yfir samnefnda aðferð í klasanum `Applet` að geta notað flýtiminni og bráðabirgðageymslur („cache“) vefsjárinnar. Til að láta hljóðið heyrast er `play`-aðferð klasans `AudioClip` notuð.

Vefsíðan er sótt þegar smellt er á takkann `bSaekjaVefsidu`. Veffangið er myndað eins og veffang myndarinnar og hljóðsins. Til að sýna vefsíðuna er notuð skipunin

```
ac.showDocument(vefsida, "_blank");
```

Aðferðin `showDocument` í viðmótinu `AppletContext` tekur við hlut af tegundinni `URL` og streng. `URL`-ið vísar á vefsíðuna sem sýna skal og strengurinn segir í hvaða ramma hún skal sýnd. Til að tiltaka ramma eru sömu strengir notaðir og í HTML skipanamálinu.

_self	Vefsíða sett í sama ramma og vefsíðan sem Applet-ið er á.
_parent	Vefsíða sett í ramman sem _self ramminn æxlaðist af
_top	Vefsíða sett í þann ramma sem er í forgrunni.
_blank	Vefsíða sett í nýjan auðan ramma.

Til viðbótar við þessa fjóra möguleika er hægt að nota hvaða nafn sem er ef rammi með því nafni er opinn. (Um allt þetta má fræðast í handbókum um HTML skipanamálið.)

```
import java.applet.*;
import java.awt.*;
import java.net.*;
//
// forrit_12_03
// AppletSaekirGogn
//
public class AppletSaekirGogn extends Applet
{
   Button bSaekjaHljod;
   Button bSaekjaVefsidu;
   Graphics g;
   AudioClip gelt;
   Image hundur;
   AppletContext ac;                // AppletContext er viðmót
                                    // sem inniheldur aðferðir
   // til að hafa samskipti við vefsjána sem keyrir Applet-ið.
   // Applet nota aðferðina getAppletContext til að búa til
   // hlut sem hefur þetta viðmót. Það má líta svo á að þessi
   // hlutur sé vefsjáin sem Applet-ið keyrir í.

   public void init()
   {
      bSaekjaHljod    = new Button("Hundur gelti");
      bSaekjaVefsidu = new Button("Vefsíða um hund");
      this.add(bSaekjaHljod);
      this.add(bSaekjaVefsidu);
      g = this.getGraphics();
      ac = this.getAppletContext();

      try
      {
         // Búið til URL sem vísar á mynd.
         URL mynd = new URL(this.getCodeBase(), "hundur.gif");
         // Myndin sótt. getImage aðferðin setur af stað þráð sem
         // hleður myndinni inn. Í paint aðferðinni er myndin sýnd
         // þegar hún er komin inn.
         hundur = ac.getImage(mynd);
      }
      catch (Exception fravik)
      {
         Villumelding v = new Villumelding("Get ekki sótt mynd.");
      }
   }

   public void paint(Graphics g)
   {
         // Aðferðin prepareImage skilar true ef búið er
         // að hlaða mynd inn og hún er tilbúin til að
```

```
            // birtast á skjánum.
            if (this.prepareImage(hundur, this))
            {
               g.drawImage(hundur, 10, 50, this);
            }
            else
            {
               g.drawString("Augnablik", 12, 60);
            }
      }

      public boolean action(Event e, Object o)
      {
         if (e.target == bSaekjaHljod)
         {
            try
            {
               // Búið til URL sem vísar á hljóðskrána.
               URL hljod = new URL(this.getCodeBase(), "gelt.au");
               // Hljóðið sótt og spilað.
               gelt = ac.getAudioClip(hljod);
               gelt.play();
            }
            catch (Exception fravik)
            {
               Villumelding v =
                     new Villumelding("Get ekki spilað hljóð.");
            }
            return true;
         }

         if (e.target == bSaekjaVefsidu)
         {
            try
            {
               URL vefsida = new URL(this.getCodeBase(), "hundur.html");
               // showDocument aðferðin lætur vefsjána
               // (AppletContext-hlutinn) birta vefsíðu.
               ac.showDocument(vefsida, "_blank");
            }
            catch (Exception fravik)
            {
               Villumelding v =
                     new Villumelding("Get ekki birt vefsíðu.");
            }
            return true;
         }
         return false;
      }
}
```

Verkefni 12.5*
Búðu til tvær vefsíður og `Applet` sem hefur tvo hnappa. Þegar smellt er á annan hnappinn á önnur vefsíðan að birtast og þegar smellt er á hinn hnappinn á hin vefsíðan að birtast. Láttu aðra vefsíðuna birtast í rammanum `_self` og hina í `_blank`.

Verkefni 12.6
Sumar vefsíður haga sér þannig að um leið og þeim er hlaðið inn opnast gluggi með annarri síðu. Búðu til svona vefsíðu. Best er að láta hana innhalda lítt áberandi `Applet` sem gerir ekkert annað en að sýna aðra síðu í `_blank` ramma.

Verkefni 12.7

Ef vefur ef færður (hann breytir um veffang) er gott að hafa síðu á gamla veffanginu sem vísar á það nýja. Ef lausninni á verkefni 12.6 er breytt þannig að síðan sem `Applet`-ið opnar lendir í `_self` en ekki í `_blank` er lesendum vefsins beint á nýja veffangið með sjálfvirkum hætti.

Búðu til `Applet` sem notar aðferðina `getParameter` til að sækja veffang og sendir lesanda sjálfkrafa á það veffang um leið og hann sækir síðuna með `Applet`-inu.

Verkefni 12.8
Þegar smellt er á annan hnappinn í forrit_12_03 spilar það hljóð einu sinni. Breyttu forritinu þannig að það spili hljóðið 10 sinnum með smá þögn á milli. Til að gera þetta er best að búa til þráð og láta `run` aðferð hans spila hljóðið aftur og aftur og gera stutt hlé á milli.

12.e. Biðlarar og miðlarar

Forrit_12_01, forrit_12_02 og forrit_12_03 nota sér miðlara á annarri tölvu. Hlutir af gerinni `InetAddress` nýta sér umdæmisheitaþjónustu á Internetinu og `URL` hafa samband við vefþjóna (eða vef-miðlara) og nota þjónustu þeirra t.d. til að opna strauma af Internetinu.

Við getum notað klasana `InetAddress` og `URL` án þess að velta því mikið fyrir okkur hvernig þeir ná sambandi við miðlara og senda þeim boð. En ef við ætlum að láta tvær vélar á Internetinu senda boð sín á milli og stjórna því fullkomlega hvernig þær túlka boðin og vinna úr þeim þurfum við að búa til tvö forrit, miðlara og biðlara og láta biðlarann koma sér í samband við miðlarann.

Í pakkanum `java.net` er klasi sem heitir `Socket`. Hlutir af þessari gerð eru notaðir til að koma á sambandi milli tveggja véla á TCP/IP neti. `Socket` hefur klasabreytur sem geyma eftirtaldar upplýsingar:

IP tala og umdæmisheiti eigin vélar, af tegundinni `InetAddress`.
Hlið á eigin vél af tegundinni `int`.
IP tala og umdæmisheiti vélar sem samband er við, af tegundinni `InetAddress`.
Hlið á vél sem samband er við, af tegundinni `int`.

Biðlari sem ætlar að ná sambandi við miðlara þarf að smíða hlut af tegundinni `Socket` og senda smiðnum hlut af tegundinni `InetAddress` sem geymir IP tölu eða umdæmisheiti miðlara og tölu (`int`) sem er númerið á hliðinu sem miðlarinn er stilltur á. Smiðurinn getur sjálfur fundið út IP-tölu vélarinnar sem biðlarinn keyrir á og valið sér hlið (hann velur yfirleitt lægstu tölu ofan við 1023 sem er á lausu).

Ef miðlari er staðsettur á vélinni www.hundur.is og er stilltur á að nota hlið númer 9000 þá getur biðlari tengst honum og opnað inntaks- og úttaksstrauma, sem nota má til að sækja gögn frá miðlara og senda gögn til hans, með eftirfarandi skipunum.

```
Socket tenging;
PrintStream pSenda;
DataInputStream dLesa;
try
{
   InetAddress midlari = InetAddress.getByName("www.hundur.is");
   tenging = new Socket(midlari, 9000);
   // Straumar búnir til.
```

```
      pSenda = new PrintStream(tenging.getOutputStream());
      dLesa = new DataInputStream(tenging.getInputStream());
   }

   catch (UnknownHostException fravik)
   {
   }
   catch (IOException fravik)
   {
   }
```

Smiðurinn `Socket` getur kastað tvenns konar frávikum `UnknownHostException` (ef vélin sem vísað er á finnst ekki) og `IOException` (ef ekki tekst að ná sambandi við miðlara). Aðferðin `InetAddress.getByName` kastar `UnknownHostException` ef ekki finnst vél með því umdæmisheiti, eða þeirri IP tölu, sem tiltekið er. Aðferðirnar `getOutputStream` og `getInputStream` kasta `IOException`. `Catch`-blokkirnar grípa þessar tvær gerðir frávika.

Þegar samskiptum við miðlara er lokið þarf að loka bæði `Socket` og straumum með skipunum á borð við

```
dLesa.close();
pSenda.close();
tenging.close();
```

Til að láta miðlara bregðast við þegar biðlari biður um samband við hann er klasinn `ServerSocket` notaður. Hlutir af þeirri gerð kunna aðferð sem heitir `accept`. Hún bíður eftir beiðni um samband frá biðlara og skilar hlut af tegundinni `Socket` þegar slík beiðni kemur. Meðan `accept` aðferðin er framkvæmd bíður miðlari (eða þráður sem hann hefur sett af stað) rólegur eftir að beðið sé um samband. Ekkert er því til fyrirstöðu að `accept` aðferðin bíði tímunum saman.

Hægt er að láta miðlara, sem stilltur er á hlið númer 9000, bíða eftir sambandsbeiðni frá biðlara, búa til `Socket` og opna inntaks- og úttaksstrauma með eftirfarandi skipunum

```
ServerSocket s;
Socket tenging;
DataInputStream dLesa;
PrintStream pSenda;
try
{
      s = new ServerSocket(9000);

      // Aðferðin accept lætur ServerSocket bíða eftir beiðni
      // um tengingu frá biðlara og býr til Socket til að
      // tala við biðlara um leið og slík beiðni berst.
      tenging = s.accept();

      // Straumar búnir til.
      pSenda = new PrintStream(tenging.getOutputStream());
      dLesa = new DataInputStream(tenging.getInputStream());

}
catch (IOException fravik)
{
}
```

Smiðurinn `ServerSocket` tekur við númeri hliðs og þar með er miðlarinn stilltur á að nota það hlið. Hann kastar fráviki af gerðinni `IOException`. Það gerir aðferðin `accept` líka sem og `getOutputStream` og `getInputStream`.

Aðferðin `accept` býr til `Socket`. Hún tekur við upplýsingum um IP tölu og hlið frá biðlara og finnur IP-tölu vélarinnar sem forritð er keyrt á.

Þegar samskiptum við biðlara er lokið þarf miðlari að loka bæði `Socket` og straumum með skipunum á borð við

```
pSenda.close();
dLesa.close();
tenging.close();
s.close();
```

Þegar samband er komið á milli miðlara og biðlara geta þeir sent hvers kyns gögn sín á milli. Svona samband er hægt að nota til að senda skilaboð, leika leiki, safna upplýsingum um tölvur á neti o.fl. o.fl. Internetið er ekki bara til að dreifa pósti og miðla vefsíðum. Möguleikar þess eru ótæmandi.

Forrit_12_04 er einfaldur miðlari sem gerir ekkert annað en taka við skilaboðum frá biðlara og skrifa þau í textareit. Forritið er myndað úr tveim klösum `Starta` og `Midlari`. `Starta` er nokkuð frábrugðinn samnefndum klösum í öðrum sjálfstæðum forritum í þessu hefti því hann keyrir ekki aðeins `init` aðferðir hins klasans heldur líka aðferðirnar sem vinna verkin, þær eru semsagt ekki settar af stað í `action`-aðferð, við það að smellt er á takka, heldur keyrðar úr `main`-aðferðinni í `Starta`.

```
//
// forrit_12_04
// Starta
// Byggt á samnefndum klasa í forrit_09_01
//
public class Starta
{
   public static void main(String args[])
   {
      Midlari m = new Midlari("Miðlari fyrir skilaboð.");
      m.init();
      m.resize(550, 450);
      m.move(20, 40);
      m.show();
      m.skrifaUpplysingarUmEiginIPToluHeitiOgHlid();
      m.tengjastOgTakaViðBodum();
   }
}
```

`Midlari` hefur einn takka til að hætta keyrslu og tvö textasvæði, annað til að skrifa upplýsingar um gang mála (láta vita þegar samband næst o.s.frv), hitt til að birta skilaboð frá biðlara.

Þessi miðlari er takmarkaður að því leyti að hann getur aðeins verið í sambandi við einn biðlara í senn. Til að láta miðlara þjóna mörgum biðlurum samtímis er heppilegast að láta þræði sjá um samskiptin og setja nýjan þráð af stað í hvert sinn sem `accept` aðferðin í klasanum `ServerSocket` skilar nýjum `Socket`.

Aðferðin `init` gerir ekki annað en að raða textasvæðum, takka og merkjum á myndflötinn. `action`-aðferðin bregst við ef smellt er á takkann `bHaetta`. Ef það er gert meðan samband við biðlara er opið er `Socket` og straumum ekki lokað með eðlilegum hætti. Í fullkomnum miðlara sem á að nota í alvöru þarf að gæta þess að forritið loki örugglega öllum tengingum og straumum.

Meginhlutar forritsins eru aðferðirnar `skrifaUpplysingarUmEiginIPToluHeitiOgHlid` (sem gerir nákvæmlega það sem nafn hennar bendir til) og `tengjastOgTakaViðBodum`. Sú síðarnefnda kemur á sambandi um leið og beiðni þar um kemur frá

biðlara, skrifar öll skilaboð frá biðlaranum í textasvæðið `tFraBidlara` og rýfur sambandið þegar hún fær senda línu sem í stendur „bless“ og ekkert annað.

Þessar aðferðir skýra sig að mestu leyti sjálfar. Þó er vert að nefna að í hvert sinn sem úttaksstraumurinn `pSenda` í aðferðinni `tengjastOgTakaViðBodum` er látinn senda svar til miðlara er hann látinn framkvæma aðferðina `flush`. Úttaksstraumar í Java hafa aðferð með þessu nafni til þess að dæla öllu innihaldi sínu strax á áfangastað. Þegar gögn eru sett í úttaksstraum bíða þau stundum þar í góða stund áður en þau eru send lengra áfram. Þegar skrifað er í skrá á diski er þetta heppilegt. Það er þá ekkert verið að setja diskinn í gang fyrr en safnað hefur verið góðum slatta af gögnum í strauminn. Þetta hefur þær afleiðingar að skrifað er sjaldnar á diskinn og meira í senn, sem er mun fljótlegra heldur en að vera sífellt að skrifa eitthvert smáræði. En miðlari sem á að svara öllum boðum frá biðlara strax getur ekki leyft sér að láta gögn safnast upp í úttaksstraumi. Hann verður því að beita `flush`-aðferð á strauminn í hvert sinn sem boð eru send.

```
import java.awt.*;
import java.net.*;
import java.io.*;
//
// Forrit_12_04
// Midlari
//   Tekur við skilaboðum frá biðlara eins og í forrit_12_05
// og birtir þau í textareit og sendir svar til baka um að
// skilaboð hafi borist.
//   Slítur sambandi um leið og biðlari sendir línu sem í
// stendur "bless" og ekkert annað.
//
public class Midlari extends Frame
{
   // Þetta forrit á að svara boðum sem berast á port númer 9000.
   final int HLID = 9000;

   Button bHaetta;
   Label lUpplysingar, lFraBidlara;
   TextArea tFraBidlara, tUpplysingar;

   Midlari(String s)
   {
      super(s);
   }

   public void init()
   {
      FlowLayout f = new FlowLayout();
      this.setLayout(f);

      // Merki, takkar og textareitir smíðaðir og settir á skjáinn.
      lFraBidlara = new Label("Boð frá biðlara.");
      lUpplysingar = new Label("Upplýsingar um gang mála.");
      bHaetta = new Button("Hætta");
      tFraBidlara = new TextArea(10, 70);
      tUpplysingar = new TextArea(10, 70);

      this.add(tFraBidlara);
      this.add(lFraBidlara);
      this.add(tUpplysingar);
      this.add(lUpplysingar);
      this.add(bHaetta);
   }
```

```
public void skrifaUpplysingarUmEiginIPToluHeitiOgHlid()
{
   // Upplýsingar um eigin IP tölu og umdæmisheiti fundnar.
   try
   {
      InetAddress thessiTolva;
      thessiTolva = InetAddress.getLocalHost();
      String eiginIPTala = thessiTolva.getHostAddress();
      String eigidUmdaemisheiti = thessiTolva.getHostName();
      tUpplysingar.appendText("IP tala mín er: "
                              + eiginIPTala + "\n");
      tUpplysingar.appendText("Umdæmisheiti mitt er: "
                              + eigidUmdaemisheiti + "\n");
      tUpplysingar.appendText("Ég tek við boðum sem berast á " +
                              "port númer: " + HLID + "\n");
   }
   catch (UnknownHostException fravik)
   {
         tUpplysingar.appendText("Eitthvað klikkaði" + "\n" +
                                 fravik.toString() + "\n");
   }
}


public void tengjastOgTakaViðBodum()
{
   ServerSocket s;
   Socket tenging;
   DataInputStream dLesa;
   PrintStream pSenda;
   String lina;
   try
   {
      s = new ServerSocket(HLID);
      tUpplysingar.appendText(s.toString() + "\n");

      // Aðferðin accept lætur ServerSocket bíða eftir beiðni
      // um tengingu frá biðlara og býr til Socket til að
      // tala við biðlara um leið og slík beiðni berst.
      tenging = s.accept();
      tUpplysingar.appendText(tenging.toString() + "\n");

      // Straumar búnir til.
      pSenda = new PrintStream(tenging.getOutputStream());
      dLesa = new DataInputStream(tenging.getInputStream());

      // Skilaboð frá biðlara lesin og birt í tFraBidlara
      int linunumer = 1;
      do
      {
         lina = dLesa.readLine();
         tFraBidlara.appendText(lina + "\n");
         pSenda.print("Lína númer " + linunumer +
                      "móttekin." + "\n");
         linunumer++;
         // Aðferðin flush tryggir að það sem printLine
         // sendir af stað fari strax alla leið en sitji
         // ekki í biðminni.
         pSenda.flush();
      } while(!lina.equals("bless"));
      tUpplysingar.appendText("Samskiptum lokið." + "\n" +
                              "Rýf samband." + "\n");

      // Tenginar rofnar.
      tenging.close();
      s.close();
```

```
            pSenda.close();
            dLesa.close();
            tUpplysingar.appendText("Samband rofið.");
         }
         catch (IOException fravik)
         {
               tUpplysingar.appendText("Samband klikkaði" + "\n" +
                                       fravik.toString() + "\n");
         }
         catch (Exception fravik)
         {
               tUpplysingar.appendText("Eitthvað klikkaði" + "\n" +
                                       fravik.toString() + "\n");
         }
      }


      public boolean action(Event e, Object o)
      {
         if (e.target == bHaetta)
         {
            dispose();
            System.exit(0);
            return true;
         }
         return false;
      }
} // Hér endar klasinn Midlari.
```

Forrit_12_05 er biðlari sem getur tengst miðlara eins og forrit_12_04 og sent honum skilaboð. Forritið er samsett út tveim klösum, `Starta` og `Bidlari`. Sá fyrrnefndi er næstum eins og samnefndir klasar í mörgum öðrum forritum og er því ekki birtur hér.

`Bidlari` hefur textareiti til að skrifa í umdæmisheiti eða IP-tölu vélar sem miðlari keyrir á og númer hliðs. Hann hefur einnig þrjá takka, einn til að hætta keyrslu, einn til að tengjast og einn til að senda skilaboð. Klasinn hefur enn fremur þrjú textasvæði, eitt til að skrifa í skilaboð til að senda til miðlara, eitt til að birta svör frá miðlara og eitt til að birta upplýsingar um gang mála. Þegar smellt er á takkann `bSendaSkilabod` er allt innihald textasvæðisins `tSkilabod` sent til miðlara.

Auk aðferðanna `init` og `action` hefur `Bidlari` þrjár aðferðir. Þær heita `tengjast`, `sendaSkilabod` og `rjufaSamband`. Nöfnin segja allt sem segja þarf um hlutverk þeirra og skýringar eru skrifaðar í forritskóðann.

```
import java.awt.*;
import java.net.*;
import java.io.*;
//
// Forrit_12_05
// Bidlari
// Sendir skilaboð til miðlara eins og í forrit_12_04.
// Slítur samband með því að senda línu sem í stendur
// "bless" og ekkert annað.
//
public class Bidlari extends Frame
{
   // Tengi og straumar eru klasabreytur því þessi hlutir
   // eru notaðir af mörgum aðferðum (tengjast, sendaSkilabod
   // og rjufaSamband). Í miðlaranum (forrit_12_04) eru samsvarandi
   // hlutir staðværir því þeir eru aðeins notaðir af einni
   // aðferð sem er tengjastOgTakaViðBodum.
```

```
Socket tenging;
DataInputStream dLesa;
PrintStream pSenda;

// Boolean breyta sem geymir upplýsingar um hvort samband er á
// eða ekki. Hefur upphaflega gildið false en fær gildið true um
// leið og samband kemst á og aftur gildið false þegar það rofnar.
boolean iSambandi = false;

// Sýnilegir hlutir úr java.awt
Label lIPTalaMidlara, lHlidMidlara;
TextField tIPTalaMidlara, tHlidMidlara;
Button bHaetta;
Button bTengjast;
Button bSendaSkilabod;
Label lSkilabod, lFraMidlara, lUpplysingar;
TextArea tSkilabod, tFraMidlara, tUpplysingar;

Bidlari(String s)
{
   super(s);
}

public void init()
{
   FlowLayout f = new FlowLayout();
   this.setLayout(f);

   // Merki, takkar og textareitir smíðaðir og settir á skjáinn.
   lIPTalaMidlara = new Label("IP tala miðlara");
   lHlidMidlara = new Label("Númer hliðs á miðlara");;

   tIPTalaMidlara = new TextField(15);
   tHlidMidlara = new TextField(5);

   lSkilabod = new Label("Skilaboð til miðlara");
   lFraMidlara = new Label("Svör frá miðlara.");
   lUpplysingar = new Label("Upplýsingar um gang mála.");

   tSkilabod = new TextArea(7, 70);
   tFraMidlara = new TextArea(4, 70);
   tUpplysingar = new TextArea(4, 70);

   bTengjast = new Button("Tengjast");
   bSendaSkilabod = new Button("Senda skilaboð");
   bHaetta = new Button("Hætta");

   this.add(lIPTalaMidlara); this.add( tIPTalaMidlara);
   this.add(lHlidMidlara); this.add(tHlidMidlara);
   this.add(bTengjast);
   this.add(tSkilabod); this.add(lSkilabod);
   this.add(bSendaSkilabod);
   this.add(tFraMidlara); this.add(lFraMidlara);
   this.add(tUpplysingar); this.add(lUpplysingar);
   this.add(bHaetta);
}


public void tengjast()
{
   String sIPTalaMidlara = tIPTalaMidlara.getText();
   // Streng í tHlidMidlara breytt í heiltölu af teg. int.
   Integer I = Integer.valueOf(tHlidMidlara.getText());
   int hlidMidlara = I.intValue();
   try
   {
```

```
            InetAddress midlari = InetAddress.getByName(sIPTalaMidlara);
            tenging = new Socket(midlari, hlidMidlara);
            tUpplysingar.appendText("Samband er komið á" + "\n");
            tUpplysingar.appendText(tenging.toString() + "\n");

            // Straumar búnir til.
            pSenda = new PrintStream(tenging.getOutputStream());
            dLesa = new DataInputStream(tenging.getInputStream());

            iSambandi = true;
        }

        catch (UnknownHostException fravik)
        {
            tUpplysingar.appendText("Miðlari finnst ekki" + "\n" +
                                    fravik.toString() + "\n");
        }

        catch (IOException fravik)
        {
            tUpplysingar.appendText("Ekki tókst að ná sambandi" + "\n" +
                                    fravik.toString() + "\n");
        }

        catch (Exception fravik)
        {
            tUpplysingar.appendText("Eitthvað klikkaði" + "\n" +
                                    fravik.toString() + "\n");
        }
    }

    public void sendaSkilabod()
    {
        try
        {
            String sSkilabod = tSkilabod.getText();
            pSenda.print(sSkilabod + "\n");
            pSenda.flush();
            String lina = dLesa.readLine();
            tFraMidlara.appendText(lina + "\n");
            if (sSkilabod.equals("bless"))
            {
                rjufaSamband();
            }
        }
        catch (IOException fravik)
        { // readLine í DataInputStream kastar IOException.
            tUpplysingar.appendText("Ekki barst svar frá miðlara" +
                                   "\n" + fravik.toString() + "\n");
        }
        catch (Exception fravik)
        {
            tUpplysingar.appendText("Eitthvað klikkaði" + "\n" +
                                    fravik.toString() + "\n");
        }

    }

    public void rjufaSamband()
    {
        try
        {
            dLesa.close();
            pSenda.close();
            tenging.close();
        }
```

```
      catch (IOException fravik)
      {
         tUpplysingar.appendText("Ekki tókst að loka straumum og
                     tengingu." + "\n" + fravik.toString() + "\n");
      }
      finally
      {
         iSambandi = false;
      }
   }


   public boolean action(Event e, Object o)
   {
      if (e.target == bTengjast)
      {
         if (!iSambandi)
         {
            tengjast();
         }
         return true;
      }

      if (e.target == bSendaSkilabod)
      {
         if (iSambandi)
         {
            sendaSkilabod();
         }
         return true;
      }

      if (e.target == bHaetta)
      {
         if (iSambandi)
         {
            rjufaSamband();
         }
         dispose();
         System.exit(0);
         return true;
      }
      return false;
   }
}
```

Verkefni 12.9*
Prófaðu að keyra forrit_12_04 (miðlarann) og forrit_12_05 (biðlarann) og láta það síðarnefnda senda því fyrrnefnda skilaboð. Prófaðu fyrst að keyra bæði forritin á sömu vél. (Þar sem vélar á TCP/IP neti láta sem IP talan 127.0.0.1 tilheyri sjálfum sér er best að biðja biðlarann að hafa samband við miðlara sem heitir 127.0.0.1 og stilla portið á 9000.) Þegar þú keyrir forritin sitt á hvorri vélinni þarftu e.t.v. að nýta þér upplýsingarnar sem miðlarinn skrifar um eigin IP tölu og hlið í neðri textasvæðið.

Verkefni 12.10
Bættu við forrit_12_04 og forrit_12_05 því sem þarf til að hægt sé að nota þau til að „talast við“. Það þarf meðal annars að bæta við miðlarann einhverjum möguleikum á að senda skilaboð.

Verkefni 12.11

Búðu til miðlara sem tekur við sambandi frá biðlara og skrifar, í textaskrá, IP tölu hans og upplýsingar um hvaða dag og klukkan hvað hann tengist og rýfur svo sambandið.

Búðu líka til biðlara sem gerir ekkert annað en ná sambandi við miðlarann og rjúfa það strax aftur.

Settu biðlarann svo upp á nokkrum vélum sem tengdar eru TCP/IP neti og miðlarann á einni. Ef biðlararnir eru alltaf keyrðir um leið og vél tengist netinu skráir miðlarinn upplýsingar um hvenær hver vél tengist.

12.x. Spurningar og umhugsunarefni

1. Hvaða munur er á Internetinu og innra neti?
2. Hvað er TCP og hvað er IP?
3. Hvað er IP tala?
4. Hvaða sérstöðu hafa IP tölur sem byrja á 10, 192 og 127?
5. Hvað er umdæmisheiti og til hvers eru þau höfð til viðbótar við IP-tölur?
6. Hvað er DNS?
7. Hvað er URL?
8. Hvað er CGI-bin forrit?
9. Hvað er http?
10. Hvað er URL-kóðaður strengur?
11. Hvað er hlið og til hvers eru þau notuð?
12. Hvaða munur er á miðlara og biðlara?
13. Til hvers er klasinn `InetAddress`?
14. Hvaða gildi fær breytan `s` ef þessar skipanir eru gefnar?

```
InetAddress j;
j = InetAddress.getByName("www.fva.is");
String s = j.getHostAddress();
```

15. Á hvað vísar `URL` sem er smíðað svona:

```
URL u = new URL(this.getCodeBase(), "kisa.html");
```

16. Til hvers er klasinn `URLEncoder`?
17. Hvað er `AppletContext`?
18. Hverju skila aðferðirnar `getCodeBase` og `getDocumentBase` í klasanum `Applet`?
19. Til hvers er klasinn `Socket` og hvaða upplýsingar geyma hlutir af þeirri gerð?
20. Til hvers er klasinn `ServerSocket` og hvaða hlutverki þjónar aðferðin `accept` sem tilheyrir honum?
21. Af hverju þarf að beita aðferðinni `flush` á úttaksstrauma þegar gögn eru send milli tölva?
22. Hvernig er hægt að láta miðlara þjóna mörgum biðlurum samtímis?

Til umhugsunar

Í kafla 12.e var fjallað um hvernig hægt er að láta tvær tölvur skiptast á boðum með því að önnur keyri miðlara og hin biðlara. Tölvur sem keyra miðlara eru yfirleitt stöðugt í gangi og alltaf til þjónustu reiðubúnar. Tölvurnar sem almenningur notar til að tengjast Internetinu keyra yfirleitt eintóma biðlara. Samt er algengt að tvær slíkar tölvur skiptist á skeytum. Menn senda t.d. boð gegnum spjallrásir (IRC) eða sitja hver að sínum biðlara og tefla skák.

Hvernig þarf að breyta forrit_12_04 og forrit_12_05 þannig að miðlarinn geti tengst tveim biðlurum og þeir sem nota biðlarana spjallað saman?

Viðauki I: Frátekin orð og aðgerðir

Orðaforði Java skiptist í frátekin orð, aðgerðir og nöfn. Fráteknu orðin og aðgerðirnar eru grunnorðaforði málsins. Frátekin orð í Java 1.0 og 1.1 eru:

```
abstract    boolean     break       byte        (byvalue)
case        (cast)      catch       char        class
(const)     continue    default     do          double
else        extends     false       final       finally
float       for         (future)    (generic)   (goto)
if          implements  import      (inner)     instanceof
int         interface   long        native      new
null        (operator)  (outer)     package     private
protected   public      (rest)      return      short
static      super       switch      synchronized this
throw       throws      true        transient   try
(var)       void        volatile    while
```

Þessi orð er ekki hægt að nota fyrir nöfn á klasa, aðferðir eða breytur. Þau sem eru innan sviga eru merkingarlaus og aldrei notuð. Sum þeirra eru höfð frátekin til þess að hægt sé að bæta þeim við málið síðar. Væru þau ekki frátekin gæti einhver notað þau fyrir nöfn á klasa, breytur eða aðferðir og þá væri ekki hægt að bæta þeim við málið án þess að gera öll forrit þar sem þau koma fyrir sem nöfn ónothæf.

Til viðbótar við fráteknu orðin eru aðgerðir innbyggðar í Java. Þær eru:

For-gangs-röð	**Aðgerðir**	**Viðfang**	**Skýring**
1	++, --,	tala	Hækkar / lækkar tölu um 1
1	+, -	tala	Formerki, beitt á eina tölu
1	~	heiltala	Neitun hvers bita
1	!	yrðing	Neitun
1	(<heiti teg.>)	hvað sem er	Breytir einni tegund í aðra
2	*, /, %	tala, tala	Margföldun, deiling og afgangur
3	+, -	tala, tala	Samlagning og frádráttur
3	+	strengur, strengur	Límir saman tvo strengi
4	<<, >>, >>>	heiltala	Hliðrun bita fyrir bita
5	<, <=, >, >=	tala, tala	Samanburður
6	==, !=	einföld tegund, einföld tegund	Er jafnt / er ekki jafnt
6	==, !=	hlutur, hlutur	Er sami / er ekki sami hlutur
7	&	heiltala, heiltala	Og - beitt bita fyrir bita
7	&	yrðing, yrðing	Og - báðar yrðingar metnar
8	^	heiltala, heiltala	Xor beitt bita fyrir bita
8	^	yrðing, yrðing	Xor
9	|	heiltala, heiltala	Eða - beitt bita fyrir bita
9	|	yrðing, yrðing	Eða - báðar yrðingar metnar
10	&&	yrðing, yrðing	Og - seinni ekki metin sé fyrri false
11	||	yrðing, yrðing	Eða - seinni ekki metin sé fyrri true
12	?:	yrðing, hvað sem er, hvað sem er	Skilyrt útkoma
13	=	breyta, gildi	gildisgjöf
13	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=	breyta, gildi	gildisgjöf með aðgerð

Forgangsröð aðgerða segir til um í hvaða röð þær eru framkvæmdar. Til dæmis er * á undan + svo margföldun er framkvæmd á undan samlagningu með þeim afleiðingum að 3+7*2 er 17 en ekki 20. Reikniaðgerðirnar * og + eru á undan samanburðaraðgerðinni > svo 3+7*2>30/2 þýðir 17>15 sem hefur gildið true.

Við komumst ekki langt með þessi fráteknu orð og aðgerðir ein að vopni. Auk þeirra notum við nöfn á klösum sem fylgja Java og aðferðum, breytum og föstum sem þeir innihalda. Föst hefð er að klasar og smiðir þeirra heiti nöfnum með stórum upphafsstaf (dæmi: `Button`, `Color`, `TextField`). Klasabreytur og aðferðir heita yfirleitt nöfnum með litlum upphafsstaf en stórum staf til að sýna upphaf orðs (dæmi: `valueOf()`, `toString()`). Fastar, þ.e. klasabreytur sem eru skilgreindar sem `final` og hafa því fast innihald, heita svo nöfnum úr eintómum stórum stöfum (dæmi `Math.PI`, `GridBagConstraints.VERTICAL`).

Það er ekki leyfilegt að búa nýja klasa, breytur og aðferðir sem hafa frátekin orð fyrir heiti. Hins vegar er leyfilegt að nota nöfn sem fyrir koma í þeim klösum sem fylgja með málinu. Við getum til dæmis búið til klasa og látið hann heita `Button`. En ef við gerum það þá er erfiðara að nota klasann `Button` sem fylgir með Java.

Viðauki II: Pakkar

Hver Java-klasi tilheyrir einhverjum pakka. Ef ekki er tekið fram hvaða pakka klasi skuli tilheyra telst hann vera í sjálfgefnum (default) nafnlausum pakka. Klasar sem eru í nafnlausum pakka og geymdir í sömu efnisskrá geta notað hver annan eins og aðrir klasar sem tilheyra sama pakka. (Um aðgang að klösum er fjallað í kafla 5.d.)

Til að stjórna því hvaða pakka klasi tilheyrir er `package`-skipun sett efst í kóðan (ofan við þessa skipun má ekki vera neitt nema athugasemdir og auðar línur). Eigi klasi til dæmis að tilheyra pakkanum `lifverur.randyr.kettir er` skipunin

```
package lifverur.randyr.kettir;
```

sett efst í forritskóðann.

Klasi sem tilheyrir öðrum pökkum en sjálfgefna, nafnlausa pakkanum þarf að vera staðsettur í efnisskrá sem heitir sama nafni og pakkinn og er staðsett inn af efnisskrá sem geymd er í umhverfisbreytunni CLASSPATH.

Öll verkfæri til að smíða Java forrit bjóða upp á einhverja leið til að tilgreina CLASSPATH, þ.e. efnisskrár sem klasar eru geymdir í. Á tölvu sem notar Windows stýrikerfið gæti CLASSPATH til dæmis haft gildið C:\javaklasar og þá ætti klasi sem tilheyrir pakkanum `lifverur.randyr.kettir` að vera í efnisskránni

```
C:\javaklasar\lifverur\randyr\kettir
```

og klasi sem tileyrir pakkanum `java.awt` í efnisskránni

```
C:\javaklasar\java\awt
```

CLASSPATH þarf ekki endilega að innihalda eina efnisskrá. Hún getur geymt nöfn margra efnisskráa. Ef vél notar Windows eru þau aðgreind með semikommu svo ef sumir klasar eiga að vera inn af C:\javaklasar og sumir inn af D:\forrit\klasar þá ætti CLASSPATH að hafa gildið C:\javaklasar;D:\forrit\klasar

Klasarnir sem fylgja Java 1.0 skiptast í 8 pakka. Þeir eru

`java.applet`	Inniheldur `Applet` klasann og lítið annað.
`java.awt`	Mjög stór pakki. Inniheldur alls konar sýnilega hluti (eins og takka, textareiti, valmyndir, myndfleti) og atburði sem þeir geta orðið fyrir.
`java.awt.image`	Inniheldur ýmsar aðferðir til myndvinnslu.
`java.awt.peer`	Inniheldur enga klasa heldur eingöngu viðmót. Gegnir hlutverki við að gera Javaforritum mögulegt að keyra í ólíkum stýrikerfum.
`java.io`	Allt sem þarf til að lesa úr og skrifa í skrár
`java.lang`	Inniheldur tegundir sem flest forrit þurfa á að halda eins og t.d. `Object`, `String` og `Math`.
`java.net`	Allt sem þarf til að sækja og senda gögn um Internetið.
`java.util`	Ýmis gagnleg verkfæri, t.d. `StringTokenizer`, `Random` og `Vector`.

Með Java 1.1 fylgja nokkrir pakkar til viðbótar. Þeir heita:

```
java.awt.datatransfer        java.awt.event
java.beans                   java.lang.reflect
java.math                    java.rmi
java.rmi.dgc                 java.rmi.registry
java.rmi.server              java.security
java.security.acl            java.security.interfaces
java.sql                     java.text
java.util.zip
```

Meðal pakka sem fylgja Java hefur `java.lang` þá sérstöðu að innihald hans er sjálfkrafa „flutt inn“ í öll forrit. Innihald annarra pakka þarf að „flytja inn“ með `import` skipunum.

Ef klasi er „fluttur inn“ með `import` skipun gerist í raun ekkert annað en að mögulegt verður að skammstafa nafn hans. Klasinn `Button` í pakkanum `java.awt` heitir til dæmis fullu nafni `java.awt.Button`. Hafi innihald `java.awt` verið „flutt inn“ með skipuninni

```
import java.awt.*;
```

er hægt að skrifa bara `Button` í stað `java.awt.Button`.

Það er hægt að nota tvo klasa sem heita sama nafni svo framarlega sem þeir tilheyra ekki sama pakka. Ef nafnlausi, sjálfgefni pakkinn inniheldur klasa sem heitir Button er samt hægt að nota Button í java.awt með því að nefna hann fullu nafni: `java.awt.Button`.

Klasar sem búnir eru til af forriturunum hjá Sun Microsystems, sem eru höfundar Java málsins, tilheyra pökkum með nafni sem byrjar á „sun“ eða „java“. Aðrir skyldu ekki geyma klasa sem þeir hafa búið til í pökkum með nöfnum sem hefjast á þessum orðum.

Höfundar Java mæla með því að pakkar heiti nöfnum sem hefjast á umdæmisheiti fyrirtækis þar sem orðin eru í öfugri röð. Ef starfsmenn stofnunar sem hefur umdæmisheitið fva.is á Internetinu búa til pakka sem þeir vilja kalla `lifverur.randyr.kettir` þá ættu þeir að láta hann heita fullu nafni

```
is.fva.lifverur.randyr.kettir
```

Sé þessari reglu fylgt geta forritarar notað pakka sem þeir fá hver hjá öðrum án þess að hætta sé á að sama nafn vísi á tvo ólíka klasa.

Viðauki III: *JDK* frá Sun Microsystems

Java Development Kit er safn forrita sem höfundar Java-málsins hafa smíðað. Nafn þessa forritasafns er iðulega skammstafað og það kallað *JDK*. Forritin liggja frammi, ásamt öllum klösunum sem fylgja með Java, á vefsíðum Sun Microsystems (http://www.javasoft.com).

Forritin í *JDK* duga til að smíða, þýða og kemba Javaforrit. Þau eru öll keyrð frá skipanalínu eins og gamaldags UNIX eða DOS forrit. Hin helstu þessara forrita eru:

appletviewer	Forrit til að keyra `Applet`.
java	Túlkur sem keyrir Bytecode forrit.
javac	Þýðandi sem breytir Java-kóða í Bytecode.
javap	Til að kanna innihald Bytecode (class) skráar.
jdb	Kembiforrit til að leita að villum í forritskóða.

Ef sjálfstætt Java forrit samanstendur af klösunum `Starta` og `Teikna` og sá fyrrnefndi hefur main-aðferð þá er hægt að smíða það með *JDK* og ritli (t.d. Notepad sem fylgir Windows) svona:

1. Klasarnir eru skrifaðir og vistaðir í skrám með nöfnunum Starta.java og Teikna.java.
2. Klasarnir eru þýddir með javac. Til að þýða Teikna.java er notuð skipunin:

   ```
   javac Teikna.java
   ```

   og til að þýða Starta.java er gefin skipunin:

   ```
   javac Starta.java
   ```

3. Nú er hægt að keyra `Starta` með túlkinum java. Til þess er gefin skipunin:

   ```
   java Starta
   ```

Þegar um er að ræða `Applet`, en ekki sjálfstætt forrit, er farið eins að því að skrifa klasana og þýða þá og í skrefum 1 og 2 hér að ofan. En til að keyra `Applet` þarf að búa til vefsíðu. Hugsum okkur að `Applet` heiti `Mynd` og búið sé að þýða Java-kóðann á Bytecode með skipuninni

```
javac Mynd.java
```

Þá þarf að búa til vefsíðu sem inniheldur eftirfarandi:

```
<HTML>
<HEAD>

</HEAD>

<BODY>
   <APPLET CODE=Mynd.class WIDTH=300 HEIGHT=300></APPLET>
</BODY>
</HTML>
```

Ef vefsíðan er geymd í skrá sem heitir mynd.html er hægt að keyra forritið með appletviewer með því að gefa skipunina:

```
appletviewer mynd.html
```

Að sjálfsögðu má stilla breidd og hæð á annað en 300 í línunni:

```
<APPLET CODE=Mynd.class WIDTH=300 HEIGHT=300></APPLET>
```

Hægt er að nota forritin javac, java og appletviewer á ýmsa vegu með því að setja skipanir aftan við nöfn þeirra í skipanalínu. Listi yfir skipanir kemur upp á skjáinn ef forrit er keyrt án þess að tiltaka neitt viðfang. Ef appletviewer er t.d. keyrt með því að skipa bara

```
appletviewer
```

svarar forritið með því að skrifa:

```
usage: appletviewer [-debug] [-J<runtime flag>] url|file ...
```

Hornklofarnir þýða að það sem er innan þeirra megi vera í skipanalínunni en þurfi þess ekki. Þannig má t.d. keyra appletviewer með skipuninni -debug svona:

```
appletviewer -debug mynd.html
```

Ef þetta er gert þá er kembiforritið jdb keyrt upp samhliða appletviewer. Strikið „|“ þýðir *eða* svo aftast í skipanalínunni á að vera url eða heiti skráar.

Til að þægilegt sé að nota forritin í JDK þarf að vera hægt að keyra þau frá skipanalínu í öðrum efnisskrám en þeirri sem forritin eru geymd í. Ef tölva notar Windows stýrikerfið er heppilegast að efnisskráin sem forritin eru í sé tilgreind í umhverfisbreytunni PATH. Einnig getur þurft að gefa umhverfisbreytunni CLASSPATH gildi sem er heiti þeirrar efnisskráar sem geymir klasana sem fylgja með Java.

Viðauki IV: Java 1.0 og 1.1

Öll forrit í þessu hefti eru skrifuð í fyrstu útgáfu Java málsins, sem heitir Java 1.0. Nýrri útgáfur hafa allt sem þessi fyrsta útgáfa hefur. Nýjungarnar eru eingöngu fólgnar í viðbótum. Í viðauka II eru taldir upp allir pakkar sem fylgja Java 1.1 en eru ekki í Java 1.0. Auk þess sem nýir pakkar hafa bæst við hefur innihald sumra pakka eins og `java.io` og `java.awt` aukist verulega.

Þótt allar skipanir, aðgerðir og klasar sem fylgja Java 1.0 séu líka í Java 1.1. hafa sumar aðferðir í klösum sem fylgdu með upphaflegu útgáfunni fallið í ónáð. Á máli Java-þýðandans javac heitir þetta að þær hafi verið „deprecated“. Ef klasi notar aðferð sem er fallin í ónáð hjá þýðandanum, þá skrifar hann meldingu um að aðferðin sé „deprecated“. Forritið er vel nothæft þrátt fyrir þetta. Að aðferð sé „deprecated“ þýðir ekki að hún sé ónothæf heldur aðeins að nýrri útgáfur Java hafi betri aðferðir til að vinna sömu verk.

Í þessu hefti eru notaðar nokkrar aðferðir sem féllu í ónáð með tilkomu Java 1.1. Þær helstu eru `action`, `mouseDown`, `mouseDrag`, `mouseUp` og `keyDown`. Til að fá nákvæmar upplýsingar um hvaða aðferðir eru fallnar í ónáð er hægt að keyra þýðandann sem fylgir *JDK*, javac, með skipuninni -deprecation. Eigi til dæmis að finna hvaða aðferðir í Teikna.java eru í ónáð er hægt að þýða klasann með skipuninni:

```
javac -deprecation Teikna.java
```

Auk þess sem nýir pakkar bætast við og klösum er bætt í gamla pakka hefur Java málið tekið nokkrum breytingum frá því útgáfa 1.0 leit dagsins ljós og það er enn að breytast og þróast. Nýjasta útgáfan gengur nú ýmist undir nafninu Java 1.2 eða Java 2. Það sem kennt er í þessu kveri úreldist þó varla í bráð því breytingarnar á málinu eru allar fólgnar í viðbótum, nýjum klösum, nýjum skipunum og nýjum möguleikum.

Hér verða þessar nýjungar ekki allar taldar heldur látið duga að nefna þær merkustu, enda er langt frá því að textinn í þessu hefti geri grein fyrir öllum þeim möguleikum sem boðið var upp á þegar í fyrstu útgáfu Java málsins.

Meðal þess sem bættist við í Java 1.1 má merkast telja:

1. Innri klasar. Hér er um að ræða breytingu á málfræðireglum Java sem leyfa að klasi sé skilgreindur inni í öðrum klasa.
2. Aukinn stuðningur við önnur tungumál en ensku. (M.a. í klösunum í `java.txt.`)
3. Verulegar viðbætur við java.`awt` og nýjar gerðir atburða og aðferðir til að bregðast við þeim (í `java.awt.event`).
4. Klasar til að senda fyrirspurnir í gagnagrunna og taka við svörum frá þeim (í `java.sql`).
5. Aðferðir til að tryggja öryggi gagna sem send eru um tölvunet, m.a. með rafrænum undirskriftum (í `java.security`).
6. Möguleikar á dreifðri vinnslu þar sem Java forrit sem fer af stað á einni vél lætur hluti sem eru í gangi á öðrum vélum framkvæma aðferðir. (Hlutir sem annast þetta eru í pökkum sem heita nöfnum sem byrja á `java.rmi.`)
7. Tæki til að rækta „baunir“ (í `java.beans`) sem meðal annars auðvelda smíði tegunda sem hægt er að setja inn í forrit með tækjum til sjálfvirkrar kóðunar á notendaskilum.