Java-opiskelumateriaali: HARJOITUKSET

- JCreatorilla työskenneltäessä ei ole tarpeellista perustaa projekteja. Riittää kun .java tiedostot avataan sellaisenaan JCreatoriin ja tarvittavat tiedostot ovat kaikki samassa kansiossa.
- Varo antamasta Java-ohjelmallesi esim. nimeä String.java, koska String on Javan standardiluokka. Anna ohjelmallesi mieluiten pitkähkö nimi tyyliin KivaPeliApplet.java.

Kari Laitinen
http://www.naturalprogramming.com
2004-09-13 Tiedosto luotu.
2014-10-28 Viimeisin muutos

Java-harjoitus: Lämpötiloja muuntava ohjelma

Maailmalla on kaksi yleisesti käytettyä tapaa lämpötilojen esittämiseen. Fahrenheit-asteet (ºF) ovat käytössä USA:ssa ja joissain muissa valtioissa, kun taas Celsius-asteet (ºC) ovat käytössä useimmissa Euroopan maissa ja monissa muissa maissa maapallolla. Veden jäätymispiste on 0 Celsius-astetta ja 32 Fahrenheit-astetta, 10ºC on 50ºF, 20ºC on 68ºF, 30ºC on 86ºF, ja niin edelleen. 10 Celsius-astetta siis vastaa 18-astetta Fahrenheit-asteikossa.

1. Kirjoita ohjelma (esim. Lampo.java) joka osaa muuntaan annetut Fahrenheit-asteet Celsius-asteiksi. Voit halutessasi tehdä myös toiseen suuntaan muuntavan ohjelman.

2. Paranna ohjelmaa siten että se muuntaa sille syötetyt asteet sekä Celsius-asteiksi että Fahrenheit-asteiksi. Ohjelma siis ottaa sille syötetyn lämpötila-arvon ja tekee automaattisesti muunnoksen molempiin suuntiin.

3. Paranna ohjelmaa lisäämällä siihen ominaisuus, että jos käyttäjä syöttää lämpötilaksi arvon nolla, niin se tulostaa taulukon jossa Fahrenheit-asteet muutetaan 10 asteen välein Celsius-asteiksi. Tähän ohjelmaan tarvitset if-rakenteen, jolla tutkit onko annettu arvo nolla. Lisäksi tarvitset silmukan joka tulostaa lämpötilataulukon. (Esim. ohjelmassa SuurinLuku.java on käytössä if-rakenteita ja mm. ohjelmassa Summaussilmukka.java on esimerkki while-silmukan käytöstä. Ohjelmia Etaisyys.java ja Formatointeja.java tutkimalla saat selville kuinka tulostus voidaan formatoida printf()-metodia käytettäessä.)

4. Paranna ohjelmaa vielä siten että lämpötilataulukon tulostuksen suorittaa erillinen kutsuttava metodi. Metodi voi alkaa esim. seuraavasti

```
public static void tulosta_lampotilataulukko()
{
   ...
}
```

ja sitä voidaan kutsua main-metodista seuraavaan tapaan

```
if ( annettu_lampotila  ==  0 )
{
   tulosta_lampotilataulukko() ;
}
```

Esim. ohjelmasta Kirjaimet.java näet kuinka parametritonta metodia voidaan kutsua.

5. Jos intoa piisaa, paranna ohjelmaa siten että sillä saa tulostettua lämpötilataulukon tiedostoon. Ohjelmasta http://www.naturalprogramming.com/javaohjelmat/javaohjelmat3/KopioiTiedosto.java näet kuinka tekstitiedostoon voidaan kirjoittaa tekstiä.

Java-harjoitus: SILMUKKA JOKA TULOSTAA KERTOTAULUN

1. Tee ohjelma "Kerro.java" joka tulostaa 4:n kertotaulun while- tai for-silmukassa tyyliin

```
 1 * 4  =   4
 2 * 4  =   8
 3 * 4  =  12
 4 * 4  =  16
       ...
 9 * 4  =  36
10 * 4  =  40
```

Ohjelmassa tulee siis numeroarvojen (muuttujien) väliin saada tulostettua tekstit " * " ja " = ". Muista että kertolaskuoperaattori on Javassa * kuten C++:ssakin.

2. Saatuasi aikaan 4:n kertotaulun tulostuksen, voit parantaa ohjelmaa vielä siten että se kysyy käyttäjältä, minkä luvun kertotaulu tulostetaan. Kuten esim. ohjelmasta Peli.java voit nähdä, voidaan int-tyypin luku lukea näppäimistöltä seuraavasti

```
kerrottava  =  nappaimisto.nextInt() ;
```

Jotta tämä lause on mahdollinen, on näppäimistöolio määriteltävä ensin main()-metodin alussa ja ohjelmassa on oltava alussa lause import java.util.* ;

3. Paranna ohjelmaa siten että ohjelma tulostaa myös annettujen lukujen jakolasku ja jakojäännösoperaatiot kertotaulun sivuun

```
1 * 4  =   4     1 / 4  =  0    1 % 4  =  1
2 * 4  =   8     2 / 4  =  0    2 % 4  =  2
3 * 4  =  12     3 / 4  =  0    3 % 4  =  3
4 * 4  =  16     4 / 4  =  1    4 % 4  =  0
5 * 4  =  20     5 / 4  =  1    5 % 4  =  1
  ... jne
```

4. Muuta ohjelmaa vielä siten että aritmeettiset tulostukset tehdään erillisellä staattisella metodilla jota kutsutaan main() -metodista sen jälkeen kun käyttäjä on antanut näppäimistöltä luvun. Tulostuksen suorittava metodi kirjoitetaan luokan Kerro sisään esim. seuraavaan tapaan

```
public static void tulosta_kertotaulu_yms(
                                        int  kerrottava )
{
   ...
}
```

Esim. ohjelmasta Suurin.java näet kuinka erillistä metodia kutsutaan

5. Lisää ohjelmaan vielä metodi

```
public static void tulosta_kertotaulu_yms_heksana(
                                          int  kerrottava )
{
   ...
}
```

Tämän metodin saa helposti aikaiseksi kun kopioi edellisessä kohdassa tehdyn metodin ja muuttaa sitä sopivasti tulostuksen osalta. Luvun saa tulostettua heksadesimaalisena käyttämällä Integer -luokan metodia toHexString() seuraavaan tapaan

```
System.out.print( "joku_luku heksana on  "
                +   Integer.toHexString( joku_luku ) ) ;
```

Vaihtoehtoisesti luku voidaan tulostaa heksadesimaalisena käyttämällä metodia System.out.printf() ja formatointimäärittelyä "%X".

HARJOITUKSIA OHJELMALLA Elaimia.java

Harjoitus 1:

Lisää luokkaan Elain metodi tyhjenna_vatsa(), jonka avulla Elain-olion vatsa tyhjennetään siten että sinne kirjoitetaan tyhjä stringi "". Metodia tulee voida kutsua esim. seuraavasti

```
kissaolio.tyhjenna_vatsa() ;
koiraolio.tyhjenna_vatsa() ;
```

Tämän muutoksen onnistumisen voi testata kutsumalla anna_puhua() -metodia ja tutkimalla onko vatsa todella tyhjentynyt.

Harjoitus 2:

Lisää luokkaan Elain uusi datakenttä

```
String  elaimen_nimi ;
```

Tässä on muutettava luokan ensimmäistä konstruktoria siten että Elain-olio voidaan luoda esim. lauseella

```
Elain nimetty_kissa  =  new Elain( "kissa", "Miuku" ) ;
```

Myöskin kopiointikonstruktoria on muutettava siten että uusi datakenttä tulee kopioiduksi. Metodia anna_puhua() tulee modifioida siten että se tekee seuraavantapaisen tulostuksen

```
Hei. Mina olen kissa nimelta Miuku.
Olen syonyt: ...
```

Harjoitus 3:

Muuta metodia anna_puhua() siten että se tulostaa

```
Hei. Mina olen ... nimelta ...
Vatsani on tyhja.
```

siinä tapauksessa kun vatsan_sisalto viittaa tyhjään stringiin. Vatsa on tyhjä niin kauan kuin metodia ruoki() ei ole kutsutta. Voit käyttää String-luokan metodia length() tarkistamaan onko vatsa tyhjä. Tätä metodia voi käyttää esimerkiksi seuraavaan tapaan

```
if ( vatsan_sisalto.length()  ==  0 )
{
   // vatsan_sisalto viittaa tyhjaan stringiin.
   ...
```

Jos vatsa ei ole tyhjä, annetaan alkuperäisen kaltainen tulostus.

Harjoitus 4:

Lisää luokkaan Elain oletuskonstruktori eli konstruktori jota voidaan kutsua antamatta parametreja (argumentteja). Tällä konstruktorilla Elain-tyypin olio voidaan luoda esim. seuraavasti

```
Elain joku_elain  =  new Elain() ;
```

Oletuskonstruktori voi asettaa datakentän lajin_nimi arvoksi stringin "oletuselain" ja datakentän elaimen_nimi arvoksi "nimeton elain". Toisin sanoen, kun anna_puhua()-metodia kutsutaan oletuskonstruktorilla luodulle Elain-oliolle, syntyy seurvaavankaltainen tulostus

```
Hei. Mina olen oletuselain nimelta nimeton elain.
...
```

Luokassa on hyvä olla oletuskonstruktori siinä tapauksessa kun siitä johdetaan (periytetään) uusia luokkia. Kun periytetyn luokan olio luodaan, useimmissa tapauksissa kantaluokan oletuskonstruktoria kutsutaan automaattisesti ennenkuin periytetyn luokan konstruktori suoritetaan.

Harjoitus 5:

Kirjoita ohjelmaan, esimerkiksi luokan Elain jälkeen, uusi luokka nimeltä Elaintarha. Elaintarha-luokan on tarkoitus edustaa olioita jotka sisältävät joukon Elain-olioita. Tässä luokassa ei välttämättä tarvita konstruktoria kun datakentät alustetaan niiden esittelyn yhteydessä. Luokassa Elaintarha tulee olla metodi nimeltä lisaa_elain(), jolla uusi Elain-tyyppinen olio voidaan laittaa osaksi eläintarhaa. Lisäksi siinä tulee olla metodi anna_elainten_puhua(), jonka sisällä Elain-luokan anna_puhua()-metodia kutsutaan jokaisen tarhan eläimen suhteen. Elaintarha-luokka voi näyttää seuraavanlaiselta

```
class  Elaintarha
{
   Elain[]  tarhan_elaimet  =  new  Elain[ 20 ] ;

   int  elaimia_tarhassa  =  0 ;

   public void lisaa_elain( Elain  uusi_elain_tarhaan )
   {
      ...

   public void anna_elainten_puhua()
   {
      for ( int elaimen_indeksi  =  0 ;
                elaimen_indeksi  <  elaimia_tarhassa ;
                elaimen_indeksi  ++ )
      {
         ...
```

Tarkoitus on että Elaintarha-luokka sisältää Elain[]-tyyppisen taulukon joka sisältää viittaukset tarhan Elain-olioihin. Muuttujalla elaimia_tarhassa pidetään kirjaa siitä montako eläintä tarhassa on. Ohjelmassa Olympialaiset.java on esimerkki olioviittauksia sisältävän taulukon käytöstä.

Metodissa main() voidaan uutta Elaintarha-luokkaa testata esimerkiksi seuraavanlaisilla lauseilla:

```
Elaintarha  testitarha  =  new  Elaintarha() ;

testitarha.lisaa_elain( kissaolio ) ;
testitarha.lisaa_elain( koiraolio ) ;
testitarha.lisaa_elain( toinen_kissa ) ;
testitarha.lisaa_elain( joku_elain ) ;

testitarha.anna_elainten_puhua() ;
```

Harjoitus 6: (vanha harjoitus)

Muuta Elain-luokan datajäsen vatsan_sisalto String-olioita sisältäväksi taulukoksi joka määritellään luokan alussa seuraavaan tapaan

```
String[] vatsan_sisalto  =  new  String[ 30 ] ;
```

Kun vatsan_sisalto määritellään kuten yllä, siinä on tilaa 30 ruokastringille. Tarkoitus on että eläinoliota ruokittaessa ruokana annettava stringi lisätään tämän taulukon ensimmäiseen vapaaseen paikkaan. Ensimmäisellä ruokintakerralla ruokastringi lisätään taulukon ensimmäiseen paikkaan. Jotta taulukkoa voidaan käyttää, tulee luokkaan määritellä myös datajäsen

```
int ruokintojen_maara  =  0 ;
```

jota voidaan käyttää taulukon indeksinä. Tämä muutos vaatii muutoksia konstruktoreihin ja muihin luokan metodeihin. Metodeiden "signeerauksia", siis niiden ottamien parametrien (argumenttien) tyyppejä, ei tarvitse muuttaa. Näin myöskään metodia main() ei tarvitse tässä kohdassa muuttaa.

Esimerkiksi metodissa ruoki() tulee tehdä sellainen muutos että stringi annettu_ruoka kopioidaan taulukkoon vatsan_sisalto.
Tämä voi tapahtua seuraavasti

```
vatsan_sisalto[ ruokintojen_maara ]  =  annettu_ruoka ;
```

Metodi anna_puhua() voi tutkia vatsan sisällön tyhjyyttä käyttämällä hyväksi datajäsentä ruokintojen_maara. Vatsan sisältö tulee tulostaa silmukalla seuraavaan tapaan

```
for ( int ruoan_indeksi  =  0 ;
          ruoan_indeksi  <  ruokintojen_maara ;
          ruoan_indeksi  ++ )
{
   System.out.print( ...
           // tulostetaan yksi syöty ruoka kerrallaan
```

Harjoitus 7: (vanha harjoitus)

Johda luokasta Elain perinnan avulla uusi luokka Petoelain, ja kirjoita tähän uuteen luokkaan uusi versio metodista ruoki(). Tämä uusi metodi ruoki() tulee olla sellainen että Petoelain-oliolle voidaan syöttää toisia eläimiä. Tämä uusi ruoki()-metodi voi alkaa seuraavasti:

```
public void ruoki( Elain syotava_elain )
{
```

Toisen eläimen syominen voi tapahtua esimerkiksi siten että syötävän eläinparan datajäsen lajin_nimi kulkeutuu Petoelain-olion vatsaan. Uudessa ruoki() -metodissa voidaan viitata argumenttina (parametrina) tulevan Elain-olion datajäseneen lajin_nimi kirjoittamalla

```
syotava_elain.lajin_nimi
```

Syötävän eläimen datajäsen lajin_nimi voidaan kopioida Petoelain-olion vatsaan seuraavanlaisella käskylauseella

```
vatsan_sisalto[ ruokintojen_maara ]  =
                                syotava_elain.lajin_nimi ;
```

Uusi Petoelain-luokka kannattaa rakentaa siten että tekee perinnän avulla ensin uuden luokan, eikä lisää siihen muita metodeita kuin konstruktorin. Tämän jälkeen voi testata että uuden Petoelain-luokan olioita voi luoda, ja niille voi kutsua vanhoja metodeita. Vasta tämän jälkeen kannattaa tehdä uusi versio ruoki() -metodista. Katso mallia perinnän toteuttamiseen esimerkiksi ohjelmasta PankkiMonimuotoinen.java

Uusi luokka Petoelain voidaan kirjoittaa luokan Elain jälkeen jo olemassaolevaan ohjelmatiedostoon. (Useita luokkia voi olla kirjoitettuna samaan .java-tiedostoon jos niitä ei ole määritelty public-luokiksi.)

Petoelain-luokan toimintaa voidaan testata seuraavanlaisilla käskylauseilla:

```
Petoelain  tiikeri  =  new Petoelain( "..." ) ;
Elain      nauta    =  new Elain( "..." ) ;

tiikeri.ruoki( nauta ) ;
```

HARJOITUKSIA OHJELMALLA Olympialaiset.java

Harjoitus 1:

Nykyisellään ohjelma ei tiedä että vuonna 2012 olympialaiset pidetään Lontoossa Iso-Britanniassa. Korjaa ohjelmaa siten että myös noiden tulevien vuoden 2012 olympialaisten tiedot tulostuvat.

Harjoitus 2:

Nykyisellään ohjelmassa datan loppu kisataulukossa on merkitty "olympialaisilla" joiden olympiavuosi on 9999. Taulukosta tietoa hakeva algoritmi tunnistaa olympiadatan lopun tuon erikoisen olympiavuoden avulla. Toinen mahdollisuus tunnistaa datan loppu olioviittauksia sisältävästä taulukosta on etsiä milloin taulukosta löytyy null, eli tieto siitä että kyseisestä taulukon muistipaikasta ei viitata mihinkään olioon. Olioviittauksia sisältävä taulukko sisältää null-arvoja (nollia) heti taulukon luomisen jälkeen. Kun taulukon muistipaikasta ryhdytään viittaamaan johonkin olioon, ko. muistipaikka saa jonkin muun arvon kuin null. Muuta ohjelma sellaiseksi että datan loppu tunnistetaan null-viittausta etsimällä. Jotta null voidaan tunnistaa, on dataa hakevassa if-rakenteessa testattava ensin kohdattiinko taulukon loppu, ja vasta tämän jälkeen testataan onko taulukon kautta löydetyn olion olympiavuosi sama kuin annettu vuosi. if-rakenteen tulee siis alkaa seuraavasti

```
if ( kisataulukko[ kisojen_indeksi ]  ==  null )
{
   //  Data loppui taulukosta.
```


Tämän muutoksen ansiosta ohjelman toiminta ei saa muuttua. Tässä tehdään ohjelman rakenteesta hiukan järkevämpi.

Harjoitus 3:

Johda luokasta Olympialaiset uusi luokka nimeltä Talviolympialaiset. Voit kirjoittaa Talviolympialaiset-luokan heti Olympialaiset-luokan perään samaan tiedostoon. Jotta luokka Olympialaiset voi toimia "tehokkaasti" kantaluokkana toiselle luokalle on sinne lisättävä seuraavanlainen oletuskonstruktori

```
public Olympialaiset() {} // Tyhja oletuskonstruktori
```

Oletuskonstruktori tarvitaan koska se suoritetaan automaattisesti ennen johdetun luokan konstruktorin suorittamista.

Tee aluksi Talviolympialaiset-luokalle kostruktori joka on samanlainen kuin Olympialaiset-luokan konstruktori. Muista toki että Talviolympialaiset-luokan konstruktorin nimi tulee olla Talviolympialaiset.

Kun Talviolympialaiset-luokalla on konstruktori, voit kokeilla Talviolympialaiset-olion luontia ja taulukkoon tallettamista seuraavanlaisella lauseella

```
kisataulukko[ 28 ] = new Talviolympialaiset( 2006,
                                    "Torino", "Italia" ) ;
```

Jos ohjelmasi löytää Talviolympialaiset-olion tiedot, olet suorittanut tämän harjoituksen.

Harjoitus 4:

Edellisessä harjoituksessa tehty Talviolympialaiset-luokka ei eroa toiminnaltaan Olympialaiset-luokasta. Paranna Talviolympialaiset-luokkaa lisäämällä siihen uusi versio metodista tulosta_olympialaisten_tiedot() siten että metodin tulostamassa tekstissä mainitaan sana "talvi" esim. seuraavaan tapaan

```
Vuonna 2006 talviolympialaisten kaupunki oli Torino (Ita-
lia).
```

Tarkoitus on että Talviolympialaiset-luokassa ylikirjoitetaan luokassa Olympialaiset määritelty metodi. Kun tämmöinen metodi on olemassa, sitä käytetään automaattisesti silloin kun kisataulukosta löydetään Talviolympialaiset olio.

Harjoitus 5:

Aikaisempina vuosina talvi- ja kesäolympialaiset olivat samana vuonna. Esim. vuonna 1984 talviolympialaiset olivat Sarajevossa Jugoslaviassa. Jos tämmöisiä vanhojen talviolympialaisten tietoja lisätään kisataulukkoon, syntyy ongelma että ohjelma ei löydä niitä koska samana vuonna olleiden kesäkisojen tiedot tulostuvat ensin, ja tietojen etsintä lopetetaan tämän jälkeen. Muuta etsintäalgoritmia siten että etsintää jatketaan aina taulukon loppuun, siis ensimmäiseen null-olioviittaukseen saakka, ja tulostetaan kaikkien annettuna vuonna olleiden olympialaisten tiedot. Ohjelmaan tarvitaan todennäköisesti uusi boolean-tyypin muuttuja kuten

```
boolean olympiatietoja_loydetty  =  false ;
```

jolle annetaan arvo true silloin kun taulukosta löydetään annetun vuoden olympiatietoja. Jos tämä muuttuja on lopussa false, annetulle vuodelle ei löydetty olympiatietoja.

Harjoitus 6:

Muuta ohjelmaa siten että se tulostaa kaikki kisataulukossa olevat kesäolympialaisten tiedot siinä tapauksessa että sille annetaan vuodeksi nolla, tai jos sille annetaan vuodeksi 1, se tulostaa kaikkien talviolympialaisten tiedot. Tarkoitus on että ratkaiset tämän tehtävän oliosuuntautuneesti siten että käytät Javan instanceof-operaattoria. Tällä operaattorilla voidaan tutkia, onko taulukosta viitattu olio jotain tiettyä tyyppiä. instanceof-operaattoria voidaan käyttää seuraavaan tapaan

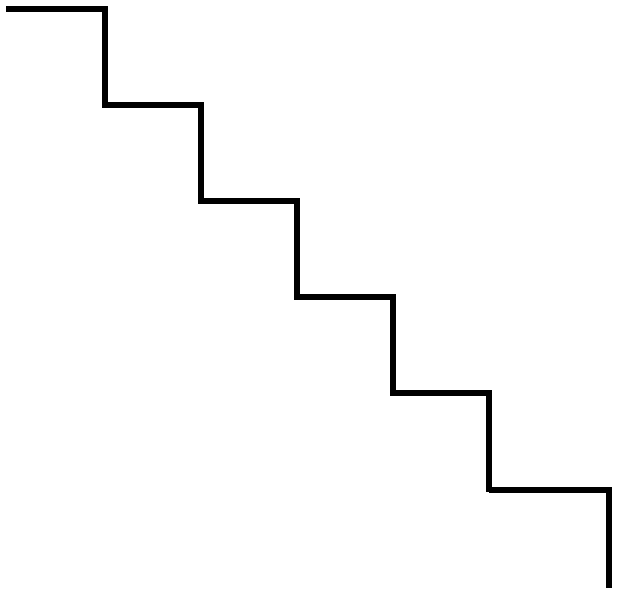
```
if ( annettu_vuosi  ==  1 )
{
   kisojen_indeksi  =  0 ;

   while ( kisataulukko[ kisojen_indeksi ]  !=  null )
   {
      if ( kisataulukko[ kisojen_indeksi ] instanceof
                                          Talviolympialaiset )
      {
         // Viitataan Talviolympialaiset-olioon.
```

instanceof palauttaa arvon true jos sen vasemmalla puolella viitataan olioon joka on oikealla puolella annetun luokan tai sen alaluokan olio. Jotta voit löytää instanceof-operaattorilla myös kesäkisojen tiedot, tulee sinun johtaa luokasta Olympialaiset luokka Kesaolympialaiset aivan samaan tapaan kuin siitä on jo johdettu luokka Talviolympialaiset. Kisataulukkoon talletettavat Olympialaiset-oliot muutat sitten (Find/Replace-toiminnolla) Kesaolympialaiset-olioiksi. Taulukon itsensä tyyppi pitää edelleen olla Olympialaiset.

HARJOITUKSIA OHJELMALLA HelloApplet.java

Muista että aplettiharjoituksia tehdessä tarvitset aina .html -tiedoston, joka määrittelee sivun jossa apletti toimii.

1. Ota pohjaksi "HelloApplet.java" ja muuta ohjelma sellaiseksi että se tulostaa apletille varatulle sivun osalle laskevat portaat seuraavaan tyyliin



Portaat voi tietysti piirtää jokaisen erikseen kutsumalla metodia drawLine() riittävän monta kertaa, mutta suositus on että piirrät portaat kutsumalla drawLine()-metodia silmukassa. paint()-metodin rakenne voi olla seuraavan tapainen

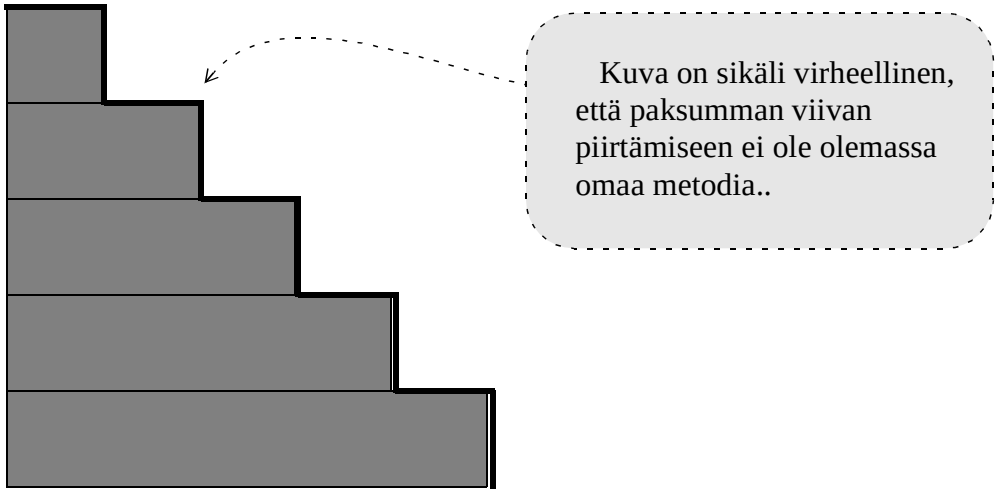
```
int viivan_alkupiste_x  =  10 ;
int viivan_alkupiste_y  =  10 ;

while ( ...
{
   graphics.drawLine( viivan_alkupiste_x,
                      viivan_alkupiste_y,
                      ...   // vaakaviivan piirto

   viivan_alkupiste_x  =  viivan_alkupiste_x  +  30 ;

   graphics.drawLine( viivan_alkupiste_x,
                      viivan_alkupiste_y,
                      ...   // pystyviivan piirto

   viivan_alkupiste_y  =  viivan_alkupiste_y  +  30 ;
}
```

paint()-metodissa voi tietysti käyttä esim muuttujaa

```
int porraslaskuri  =  0 ;
```

jolla lasketaan piirrettyjen portaiden lukumäärää ja kontrolloidaan silmukan päättyminen.

2. Lisää portaat piirtävään silmukkaan kutsu Graphics -luokan metodiin fillRect() siten että portaiden alle "perustukseksi" piirtyy joukko täytettyjä suorakaiteita alla esitetyllä tavalla. drawRect() -metodilla saat halutessasi suorakaiteille ääriviivat. Perustus kannattaa piirtää eri värillä kuin portaiden ääriviivat. EarthAndMoonApplet.java -ohjelmasta näet miten piirrosväri voidaan tarvittaessa muuttaa setColor() -metodilla. fillRect() -metodi on käytössä esimerkiksi GrafiikkademoApplet.java-ohjelmassa.



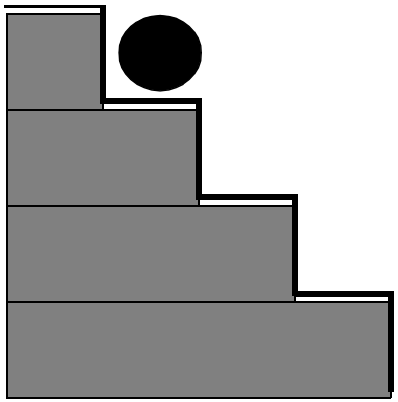
3. Määrittele aplettiluokkaan seuraava datajäsen

```
int  maalauslaskuri  =  0 ;
```

Kun lisäät paint()-metodiin lauseet

```
maalauslaskuri  ++ ;
graphics.drawString( "Maalattu "  +  maalauslaskuri  +
                     " kertaa.", 150, 10 ) ;
```

voit havaita että maalauslaskurin arvo kasvaa esimerkiksi kun pienennät ja taas suurennat apletin ikkunaa. paint()-metodia siis kutsutaan aina kun apletin ikkuna muuttuu suureksi. Tehtäväsi on em. lisäysten jälkeen muuttaa apletti sellaiseksi että ensimmäisellä kerralla ruutua maalattaessa ensimmäisen portaan kohdalle piirretään pallo seuraavaan tapaan.



Tässä portaita on vain neljä, mutta sinun ohjelmassa niitä saa olla enemmän..

Kun apletin ikkuna pienennetään ja suurennetaan tulee pallon siirtyä aina yhtä alemmalle portaalle paint()-metodin kutsun seurauksena. Alimman portaan kohdalta pallo siirtyy taas ylimmän portaan kohdalle. (Tässä tehdään eräänlaista käsikäyttöistä animaatiota.)

4. Metodin setColor() kutsulla

```
graphics.setColor(
                new Color( viivan_alkupiste_y * 0x10000 ) ) ;
```

portaiden "perustuspalkit" saadaan piirtymään punaisen eri sävyillä. Tutki luokkaa Color ja ota selville miten eri palkit tehdään vihreän tai sinisen eri sävyillä.

5. Paranna aplettia vielä siten että se ottaa selville apletille varatun alueen koon y-suunnassa, ja piirtää täsmälleen 7 porrasta kyseiselle alueelle. Piirrettävän pallon halkaisija tulee olla täsmälleen 3/4 portaan korkeudesta. (Kannattaa tutkia ohjelmaa GrafiikkademoApplet.java)

Aplettiin on myös mahdollista lisätä System.out.print()-metodin kutsuja, jotka tulostavat tekstiä AppletViewerin avaamaan komentorivi-ikkunaan. Näitä kutsuja voi käyttää esimerkiksi kun haluaa seurata joidenkin muuttujien arvojen muuttumista ohjelman suorituksen aikana.

HARJOITUKSIA OHJELMALLA SinglePictureApplet.java

Ota käyttöön ohjelma SinglePictureApplet.java ja sille tehty .html-tiedosto. Lisäksi tarvitset ohjelman käyttämän kuvatiedoston jonka löydät kansiosta www.naturalprogramming.com/javagui/.

Harjoitus 1:

Tee ohjelmasta SinglePictureApplet.java sellainen että kuva piirretään vain yhteen kertaan, sen luonnollisessa koossa, siten että kuva tulee täsmälleen apletin alueen keskelle.

Tässä voit käyttää jotain muutakin kuvaa kuin ohjelman alunperin käyttämää kuvaa. Tässä täytyy ottaa selville apletin leveys ja korkeus. Voit kopioida tarvittavat datajäsenet ja niiden tarvitsemat alustukset esim. ohjelmasta DrawingDemoApplet.java.

drawImage()-metodille annetaan parametreinä kuvan vasemman yläkulman paikka. Joudut siis tekemään laskutoimituksia jotta saat vasemman yläkulman sellaiseen paikkaan että kuva tulee alueen keskelle.

(Huom! Kuvatiedostoissa on joskus esiintynyt ongelmia siten että jokin kuva ei vaan suostu näkymään. Kokeile siis jotain toista kuvatiedostoa jos kuvan näkymisessä esiintyy jotain mystisyyttä.)

Harjoitus 2:

Lisää ominaisuus että kuvalle piirretään raamit jollain värillä. Raamit saat kätevästi aikaiseksi kun ennen kuvan piirtoa teet fillRect()-metodilla täytetyn suorakaiteen kuvan alle. Tuon suorakaiteen tulee olla hiukan kuvaa isompi, jotta raamit jäävät sopivasti näkyviin kun kuva piirretään suorakaiteen päälle.

Harjoitus 3:

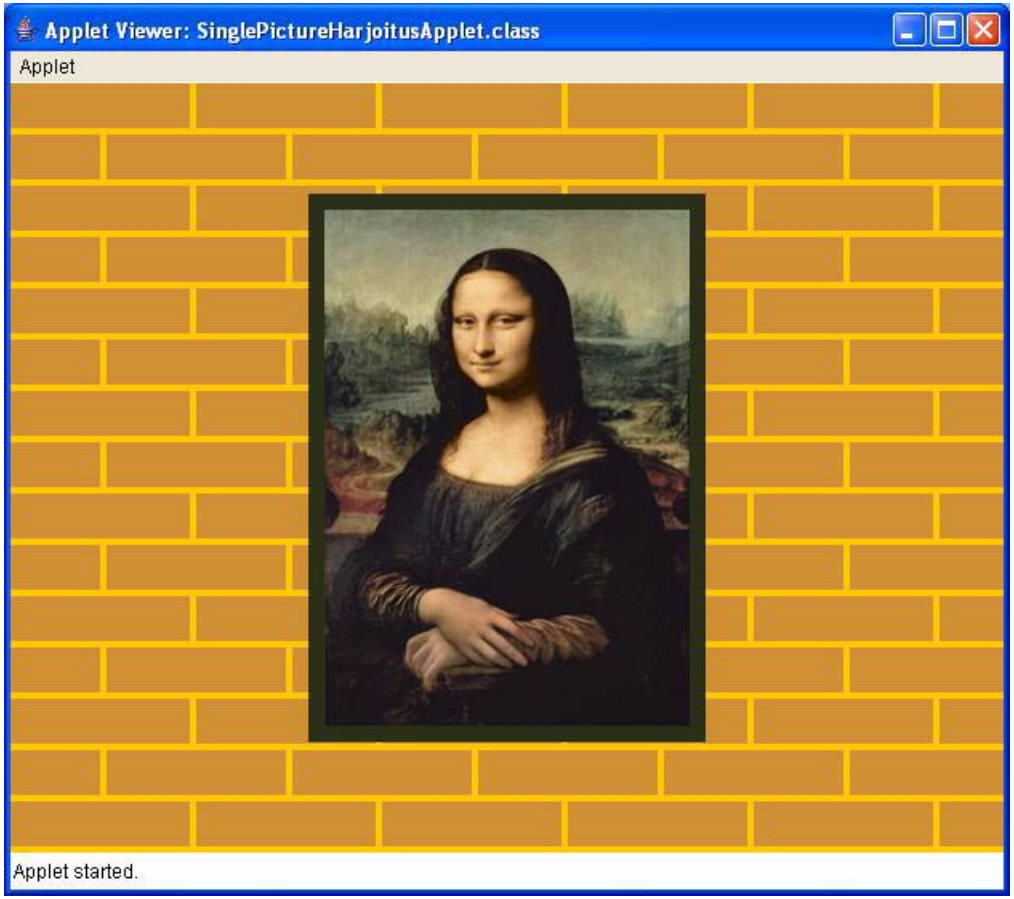
Tee ikkunassa näkyvän kuvan taustalle tiiliseinä. Tarkoitus on että tiiliseinä piirretään ennenkuin kuvan raameja tai itse kuvaa piirretään.

Tiiliseinän saat aikaiseksi kun ensin täytät koko apletin alueen fillRect()-metodilla sellaisella värillä joka on tiilien välissä olevan laastin väri. Kun olet värittänyt taustalle laastin voit piirtää tiiliä silmukoiden avulla siten että tiilien väliin jää hiukan tyhjää, joka on siis sitä "laastia".

Tiilien piirtämiseksi voi käyttää seuraavalla sivulla hahmoteltua silmukkarakennetta, jonka voit kopioida ja täydentää toimivaksi ohjelmaksi. Kun ruudulle piirretään, ei haittaa vaikka piirtäminen tapahtuu varsinaisen piirtoalueen ulkopuolle. Sellaiset piirrokset eivät vaan tule käyttäjälle näkyviin. Näin esim. puolikas tiili voidaan piirtää kun pannaan x-koordinaatti tiilirivin alussa negatiiviseksi, jolloin osa tiilestä piirtyy käyttäjälle näkymättömäksi.

```
graphics.setColor( new Color( 0xD0, 0x91, 0x32 ) ) ; // tiilien väri

int brick_position_x  =  0 ;  // Eka tiili piirretään vasempaan yläkulmaan
int brick_position_y  =  0 ;

int brick_height  =   28 ;     // tiilen korkeus
int brick_length  =  112 ;     // tiilen pituus
int gap_between_bricks =  4 ;  // Tämä on se laastin paksuus.
int row_counter = 0 ;          // Tällä lasketaan tiilirivejä.

while ( brick_position_y  <  applet_height )
{
   while ( brick_position_x < applet_width )
   {
      graphics.fillRect( brick_position_x, brick_position_y,
                         brick_length, brick_height ) ;  // Tiilen piirto

      // Tässä pitäisi kasvattaa tiilen paikan x-koordinaattia sopivasti.
   }

   // Tiilirivi on piirretty. Valmistaudutaan seuraavan rivin piirtoon.
   row_counter ++ ;

   if ( row_counter % 2 == 1 )  // Onko pariton tiilirivi?
   {
      // Tässä pitäisi tiilen paikan x-koordinaatti pistää puolen
      // tiilen verran negatiiviseksi.
   }
   else
   {
      // Tässäkin pitäisi x-koordinaattia asettaa.
   }

   // Tässä pitäisi tiilen paikan y-koordinaattia muuttaa sopivasti.
}
```

Kun tämä osatehtävä on suoritettu, ohjelman pitäisi tuottaa seuraavan kuvan kaltainen näkymä.



Harjoitus 4:

Lisää ohjelmaan ominaisuus että näytettävää kuvaa ja sen raameja voidaan pienentää tai isontaa Nuoli ylös - ja Nuoli alas -näppäimillä. Katso ohjelmasta KeyboardInputDemoApplet.java kuinka näppäimistöön voidaan reagoida

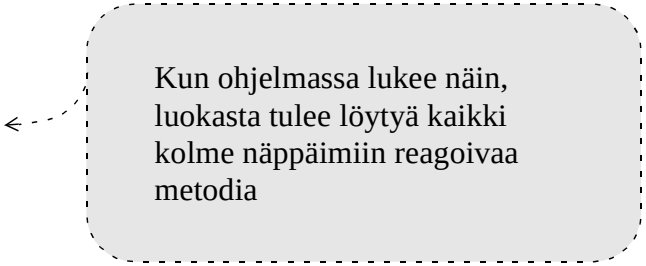
Näppäimistöön reagointia varten tulee tehdä mm. seuraavat operaatiot

- Ohjelman alussa tulee olla seuraava import -komento, jolla importoidaan kaikki java.awt.event -paketissa olevat luokat

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

- Luokan alussa tulee lukea

```
implements KeyListener
```



- init()-metodissa tulee lukea

```
public void init()
{
   ...

   addKeyListener( this ) ;
}
```

- Luokasta tulee löytyä kaikki kolme näppäimistöön reagoivaa metodia, vaikka sitten tyhjinä "dummy"-metodeina.

Eräs tapa ratkaista tämän tehtävän ongelma on määritellä luokkaan datajäsen

```
double  picture_enlargement = 1.0 ;
```

jonka arvoa muutetaan kun nuolinäppäimiä painellaan. paint()-metodissa voidaan sitten haluttu kuvan koko laskea esim. seuraavanlaisilla lauseilla:

```
int desired_picture_width = (int) ( picture_enlargement * picture_width ) ;
int desired_picture_height = (int) ( picture_enlargement * picture_height ) ;
```

Harjoitus 5:

Jos aikaa riittää ja intoa piisaa, niin lisää ohjelmaan vielä ominaisuus että näytettävää kuvaa voidaan vaihtaa Nuoli oikealle - ja Nuoli vasemmalle -näppäimillä. Tässä voi toimia niin että alussa ladataan näytettävät kuvat Image[] -tyyppiseen taulukkoon, ja pannaan olioviittaaja picture_to_show osoittamaan vuorollaan kuhunkin taulukon kuvaolioon.Tämä tarkoittaa että esim. keyPressed()-metodissa voidaan siis kirjoittaa

```
picture_to_show = pictures_to_be_shown[ index_of_current_picture ] ;
```

Kun nuolinäppäimiin reagoitaessa pannaan picture_to_show viittaamaan vuorollaan taulukon kuhunkin Image-olioon, vältytään siltä että paint()-metodiin pitäisi tässä osatehtävässä tehdä muutoksia.

Tässä osatehtävässä voi kuvataulukon määrittelyn ja luonnin sekä indeksimuuttujan kopioida suoraan ohjelmasta PictureShowApplet.java.

HARJOITUKSIA OHJELMALLA VilkkuvaPalloApplet.java

Muista että aplettiharjoituksia tehdessä tarvitset aina .html -tiedoston, joka määrittelee sivun jossa apletti toimii.

Tehtävät 1, 2 ja 3 voi ratkaista siten että metodeista muutetaan vain ruudun maalauksen suorittavaa paint()-metodia.

1. Nyt pallon värinä on Color.cyan silloin kun sitä näytetään. Muuta ohjelmaa siten että pallon väri muuttuu siten että se on joka toisella kerralla Color.cyan ja joka toisella kerralla Color.magenta silloin kun sitä näytetään. Kun käytetään Color-luokassa määriteltyjä vakiovärejä Color.cyan, Color.magenta, jne., voidaan väriolioita vertailla testaamalla olioviittaajia seuraavaan tapaan

```
if (  pallon_vari  ==  Color.cyan )
{
   ...
```

2. Muuta ohjelmaa siten että pallo alkaa hiljalleen liikkua alaspäin. Siis kun pallo tulee seuraavan kerran näkyviin, se näkyy muutamaa pikseliä alempana. Tässä pitää myös vartioida että pallon paikan y-koorinaatti ei kasva niin suureksi että pallo katoa kokonaan ruudulta. Tämän tehtävän ratkaisun seurauksena pallo jää vilkkumaan apletin alareunaan sitten kun alareuna on saavutettu.

3. Paranna edellisessä kohdassa tehtyä ominaisuutta siten että pallo lähtee nousemaan hiljalleen ylöspäin sitten kun se on saavuttanut apletin alareunan. Kun pallo on saavuttanut apletin yläreunan, se lähtee taas hiljalleen valumaan alaspäin. Tarkoitus on että pallo lopulta "sahaa" jatkuvasti apletin ala- ja yläreunan välissä. Aplettiin kannattaa määritellä seuraava datajäsen jolla pallon liikettä voiaan kontrolloida:

```
boolean  pallo_liikkuu_alaspain  =  true ;
```

Tälle muuttujalle voidaan antaa arvo false sitten kun palloa aletaan siirtämään ylöspäin.

4. Muuta ohjelma sellaiseksi että pallon ylöspäinliikkuminen tapahtuu suunnilleen kaksinkertaisella nopeudella alaspäinliikkumiseen verrattuna. Tämän tehtävän voit ratkaista siten, että erillistä säiettä suoritettavassa run()-metodissa annetaan Thread.sleep()-metodille pienempi nukkumisaika ylöspäinkulkemisen aikana.

5. Muuta ohjelmaa siten että apletin taustaväri muuttuu n. 15 sekunnin välein. Apletin taustaväri voidaan asettaa setBackground()-metodilla. Taustavärit voi tallettaa esim. erilliseen taulukkoon, josta otetaan aina uusi väri käyttöön silloin kun edellistä taustaväriä on käytetty jo mainitut 15 sekuntia. 15 sekunnin kulumisen voi mitata esim. System.currentTimeMillis()-metodin avulla. Kyseinen metodi palauttaa millisekunnit jotka ovat kuluneet vuoden 1970 alusta saakka.

Ohjelmaan voi määritellä esim. seuraavat uudet datajäsenet:

```
long  taustavarin_asetusaika ;
Color[]  taustavaritaulukko  =  new  Color[ 4 ] ;
int  taustavarin_indeksi  =  0 ;
```

init()-metodissa voidaan kiinnittää taustaväritaulukkoon värioliot ja asettaa ensimmäinen taustaväri käyttöön seuraavalla tavalla:

```
taustavaritaulukko[ 0 ]  =  Color.white ;
taustavaritaulukko[ 1 ]  =  Color.lightGray ;
taustavaritaulukko[ 2 ]  =  Color.gray ;
taustavaritaulukko[ 3 ]  =  Color.black ;

getContentPane().setBackground(
                taustavaritaulukko[ taustavarin_indeksi ]);

taustavarin_asetusaika  =  System.currentTimeMillis() ;
```

Ajan kulumista voidaan seurata esim. run()-metodissa ja asettaa uusi taustaväri käyttöön kun 15 sekuntia edellisestä värin asetuksesta on kulunut. 15 sekunnin kuluminen voidaan havaita esim. seuraavanlaisilla lauseilla:

```
long  nykyinen_aika  =  System.currentTimeMillis() ;

if ( nykyinen_aika  -  taustavarin_asetusaika  >=  15000 )
{
   ...
```

HARJOITUKSIA OHJELMALLA MatoPanelApplet.java

Kopioi työhakemistoosi MatoPanelApplet.java sekä MatoPanel.html

Harjoitus 1

Muuta ohjelmaa, siis käytännössä metodia run(), siten että mato ei "hyppää" takaisin oikeasta reunasta vasempaan reunaan, vaan kulkee tuonkin matkan oikeaoppisesti "luikertelemalla" eli venyen ja supistuen.

Tässä tehtävässä kannattaa kopioida oikeallepäin luikertelun suorittava while-silmukka ja tehdä siitä vasemmallepäin luikertelun suorittava silmukka. run() -metodiin tulee siis "ikuisen" while-silmukan sisälle peräkkäin kaksi while-silmukkaa joiden kummankin sisällä on kaksi for-silmukkaa. Vasemmallepäin luikertelun suorittava while-silmukan sisällä olevat for-silmukat on muutettava siten että mato jatkuu vasemmallepäin ja lyhenee oikelta. for-silmukoissa olevia laskurimuuttujia ei tarvitse muuttaa.

Harjoitus 2

Lisää luokkaan datajäsenet, esim.

```
int  edestakaisten_matkojen_maara  =  0 ;
int  ruudun_maalauslaskuri  =  0 ;
```

joiden avulla lasketaan kuinka monta kertaa mato käy ruudun oikeassa reunassa,ja kuinka monta kertaa paintComponent() -metodia kutsutaan. Näiden muuttujien arvot voi tulostaa esim. ruudun vasempaan yläreunaan.

Harjoitus 3

Katso mallia ohjelmasta KeyboardInputDemoApplet.java ja lisää matoaplettiin ominaisuus että madon liikkumisnopeutta voi säätää Arrow Up ja Arrow Down näppäimillä. Tämän ominaisuuden voi toteuttaa esimerkiksi siten että määrittelee datajäsenen

```
int  madon_hidastuskerroin  =  50 ;
```

joka saa suuremman arvon kun painetaan Arrow Down -näppäintä ja joka pienenee kun Arrow Up näppäintä painetaan. Kun tämmöinen datajäsen on käytössä, erilaisia hitauksia saa aikaan esim. kutsuilla

```
lepuuta_tata_saietta( madon_hidastuskerroin ) ;
lepuuta_tata_saietta( 5 * madon_hidastuskerroin ) ;
```

Lisäksi täytyy tehdä:

- Muuta koodiin

```
implements Runnable, KeyListener
```

- Lisää MatoPanel-luokan konstruktoriin kutsu

```
public MatoPanel( ...
{
   ...

   addKeyListener( this ) ;
}
```

Kun ohjelmassa lukee näin, luokasta tulee löytyä kaikki kolme näppäimiin reagoivaa metodia

- Lisää metodit näppäimiin reagointiin ja muista myös lisätä tarpeellinen import -komento, jolla importoidaan kaikki java.awt.event -paketissa olevat luokat.

Tässä tehtävässä on hyvä myös käyttää if-rakennetta jolla estetään madon hidastuskertoimen meneminen nollaksi tai negatiiviseksi. Hidastuskertoimen muuttaminen kannattaa laittaa keyPressed()-metodiin. (keyTyped()-tapahtumaa ei synny nuolinäppäimillä.) Näppäimistöön reagoivia ohjelmia tehtäessä voi joskus joutua klikkaamaan apletin ikkunaa hiirellä ennenkuin näppäimistön luku alkaa toimia.

Harjoitus 4:

Lisää ominaisuus että madon väri saadaan

- siniseksi näppäintä S painamalla,
- punaiseksi näppäintä P painamalla,
- mustaksi näppäintä M painamalla, ja
- vihreäksi näppäintä V painamalla.

Harjoitus 5:

Standardiluokan Math staattisella metodilla Math.random() saadaan aikaan satunnaislukuja. Lisää ohjelmaan ominaisuus että madolle annetaan satunnainen väri kun painetaan näppäintä R. Color-luokassa on konstruktorit joiden avulla väri voidaan määrittää "numeroina". Math.random()-metodia on käytetty mm. Matematiikkaa.java -ohjelmassa. Kyseinen ohjelma löytyy kansiosta jossa on esimerkkejä olio-ohjelmoinnin perusteisiin liittyen.

Harjoitus 6:

Ota käyttöön PacmanApplet.java-ohjelmassa käytetty "Pacman" ja pistä se suuta aukovana seuraamaan matoa. Pacmanin voi pistää madon perään kun lisää/vähentää sopivan arvon madon x-koordinaatista.

Tässä tehtävässä saattaa tarvita datajäsentä

```
boolean  mato_liikkuu_oikealle  =  true ;
```

jonka arvo muuttuu kun madon liikkumissuunta vaihtuu.

Pacmanin suun "tahdistuksen" voi laittaa metodin lepuuta_tata_saietta() sisään.

PacmanApplet.java löytyy joko www.naturalprogramming.com/javagui/-hakemistosta tai tunnilla annettavasta hakemistosta.

HARJOITUKSIA OHJELMALLA MouseDemoApplet.java

MouseDemoApplet.java on ohjelma jossa näytetään kuinka Java-sovellus voi reagoida hiiren nappien painamisiin ja hiiren liikkeisiin. Ohjelma on rakennettu siten että kaikki toiminnallisuus on ohjelmoitu luokkaan nimeltä MouseDemoPanel, ja apletin sisään on sitten pantu tämmöinen MouseDemoPanel-olio. Tutki ensin ohjelmaa ja tee sitten seuraavia harjoituksia.

Harjoitus 1:

Poista näytön sisällön "maalaavasta" metodista paintComponent() sen nykyiset toiminnot ja tee metodista sellainen että se tulostaa keskelle apletin aluetta tekstin "Et ole klikannut." Tässä kannattaa menetellä, seuraavaa osatehtävää silmälläpitäen, siten että teet luokkaan datajäsenen

```
String text_to_show = "Klikkaa!" ;
```

Ja tulostat tämän stringin arvon paintComponent()-metodissa. Ohjelmasta DrawingDemoApplet.java näet kuinka stringi piirretään näytölle ja kuinka apletin alueen koko saadaan selville.

Alussa stringi kannattaa tulostaa (piirtää) oletusfontilla. Kun saat tekstin näkyviin, voit tulostaa sen suurehkolla fontilla. Ohjelmasta EarthAndMoonApplet.java näet kuinka erilaisia fontteja saadaan käyttöön. Voit määritellä fontin datajäseneksi esim. seuraavasti

```
Font font_for_text = new Font( "Serif", Font.BOLD, 24 ) ;
```

Harjoitus 2:

Muuta ohjelma sellaiseksi että hiirellä apletin aluetta klikattaessa näytettävä teksti muuttuu. Tekstiksi pitää tulla "Koillinen" silloin kun klikataan sitä osaa apletin aluetta joka on keskipisteestä katsoen koillisessa, eli alueen pisteet ovat ovat apletin alueen keskipisteestä katsottuna x-koordinaatiltaan suurempia ja y-koordinaatiltaan pienempiä kuin keskipiste. Tekstiksi pitää tulla vastaavasti "Kaakko", "Lounas" tai "Luode" silloin kun klikkaus osuu apletin keskipisteestä katsoen näihin (väli-)ilmansuuntiin.

Tässä tehtävässä joudut siis vertaamaan klikatun pisteen koordinaatteja apletin keskipisteen koordinaatteihin. Tämän vertailun voit tehdä mousePressed()-metodissa, josta voit poistaa siellä olevan vanhan ohjelmakoodin. (Tietysti voi käyttää myös mouseClicked()-metodia.)

Voit muuttaa näytettävän tekstin (text_to_show) hiireen reagoivassa metodissa. paintComponent()-metodia ei tarvitse tässä tehtävässä muuttaa.

Harjoitus 3:

Paranna ohjelman toiminnan näyttävyyttä ("käyttäjäkokemusta") siten että seuraavan kuvan tapaan keskelle tulostuva teksti piirretään laatikon sisään, ja lisäksi viimeksi klikattu ikkunaneljännes väritetään eri värillä.

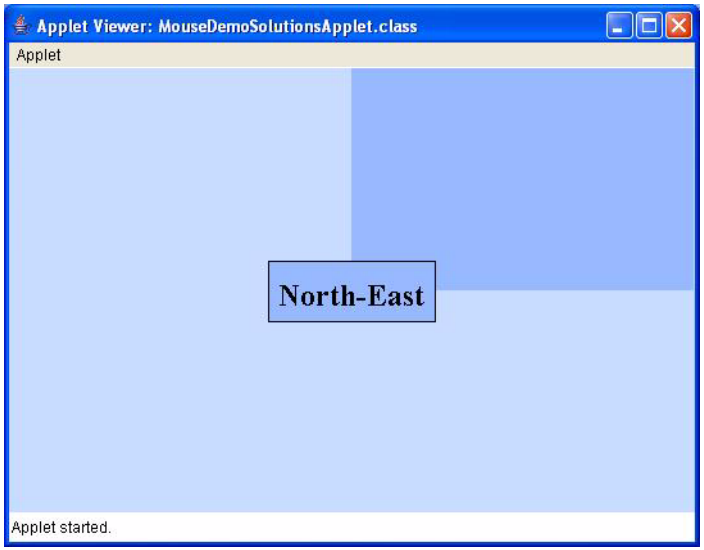
Voit paintComponent()-metodissa tutkia mikä on näytettävä teksti ja sen mukaisesti maalata neljänneksen alueesta toisella värillä. Stringejä vertailtaessa tulee käyttää equals()-metodia seuraavaan tapaan:

```
if ( text_to_show.equals( "North-East" ) )
```

Jotta saat tekstin ympärille oikean kokoisen laatikon, joudut ottamaan selville kuinka suuri kyseinen teksti on käytetyllä fontilla kirjoitettuna. Tämän saa selville käyttämällä FontMetrics-luokkaa piirtometodissa seuraavaan tapaan:

```
graphics.setFont( font_for_text ) ;

FontMetrics font_metrics = getFontMetrics( font_for_text ) ;

int text_width = font_metrics.stringWidth( text_to_show ) ;
int text_height = font_metrics.getAscent() + font_metrics.getDescent() ;
```

Kun nämä muutokset on ohjelmaan tehty, sen pitäisi tuottaa seuraavan kaltainen näkymä sen jälkeen kuo koilliseen (North-East) on klikattu:



Harjoitus 4:

Rajapinnan MouseWheelListener ja luokan MouseWheelEvent avulla pystyt lukemaan hiiren rullan liikkeitä. Muuta ohjelma sellaiseksi että keskellä olevan tekstin fonttia voidaan hiiren rullan avulla suurentaa ja pienentää.

Font-luokassa on metodi getSize() jolla saadaan selville fontin nykyinen koko. Lisäksi siellä on metodi deriveFont() joka palauttaa erikokoisen fontin silloin kun sille annetaan float-tyyppinen parametri.

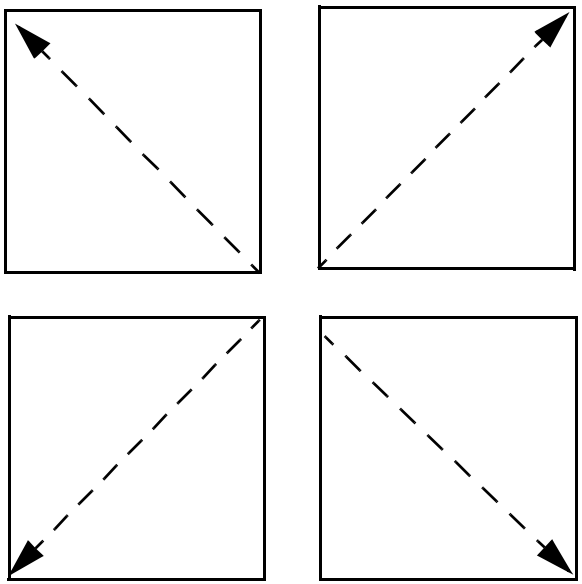
Tässä tehtävässä voidaan fontin koko muuttaa mouseWheelMoved()-metodissa. paintComponen()-metodia ei tarvitse muuttaa jos siellä on fontin koon laskenta tehty oikein edellisessä tehtävässä.

Harjoitus 5:

Muuta ohjelma sellaiseksi että se voidaan ’resetoida’ silloin kun painetaan hiiren oikeaa nappia. Resetointi tarkoittaa että näyttöön ilmestyy siellä alussa ollut teksti ja fontiksi tulee alussa käytetty fontti.

HARJOITUKSIA OHJELMALLA DrawingSuorakaiteitaApplet.java

DrawingSuorakaiteitaApplet.java, joka löytyy suomenkielisten GUI-ohjelmien kansiosta, on ohjelmasta DrawingLinesApplet.java muunnettu versio. Ohjelmaa on muutettu siten että se piirtää hiiren liikkeen määrittämiä suorakaiteita viivojen sijasta. Ohjelma osaa piirtää suorakaiteen "oikein" riippumatta siitä mikä on hiiren liikkumissuunta. Mahdolliset hiiren liikkumissuunnat ja syntyvät suorakaiteet on esitetty alla.



Ohjelmassa on metodi nimeltä piirra_suorakaide() joka suorittaa suorakaiteen piirtämisen. Tutki em. metodin toimintaa ja tee sitten seuraavat harjoitukset.

Harjoitus 1

Pistä suorakaiteen sisään piirtymään jokin valokuva (.jpg tai .gif tiedosto). Katso mallia ohjelmasta SinglePictureApplet.java.

Kuvan piirtämistoiminnon voi laittaa piirra_suorakaide()-metodin sisään. Kun kuva piirretään ennen suorakaidetta, suorakaiteesta tulee kuvalle raamit. Tarkoitus on että tässä käytetään sellaista drawImage()-metodia joka piirtää kuvan samankokoisena kuin suorakaidekin piirretään.

Huom! Kuvatiedostojen kanssa työskenneltäessä on joskus esiintynyt selittämättömiä ohjelman ’kaatumisia’. Jos tämmöistä ilmenee, kokeile jotain toista kuvatiedostoa. Saattaa olla että jotkut kuvatiedostot ovat jotenkin virheellisiä ja ohjelmia kaatavia.

Harjoitus 2

Kuten init() -metodista voit nähdä, kutsulla getContentPane().setBackground(...) voi asettaa apletin taustavärin. Tee "tyhjille" metodeille mouseEntered() ja mouseExited() toiminnot, joiden avulla taustaväri muuttuu keltaiseksi hiiren saapuessa apletin päälle, ja takaisin valkoiseksi hiiren poistuessa apletista. Ruutu on ehkä uudelleenmaalattava jotta asetettu uusi taustaväri tulee käyttöön.

Harjoitus 3

Lisää ohjelmaan ominaisuus että jos hiirtä näpäytetään Alt-nappulan ollessa alhaalla, ruudulle piirtyneet suorakaiteet ja kuvat pyyhkiytyvät pois. Luokkaan kannattaa jo seuraavaakin osatehtävää silmällä pitäen määritellä datajäsen

```
boolean piirto_operaatio_kaynnissa = false ;
```

Tämä muuttuja voidaan asettaa mousePressed()-metodissa arvoon true silloin kun Alt-nappula ei ole alhaalla ja taulukkoon mahtuu pisteiden koordinaatteja. Tällöin voidaan mouseDragged()- ja mouseReleased()-metodeissa tehdä piirtämiseen liittyviä asioita vain silloin kun em. muuttujalla on arvo true. Muuttuja pitää muistaa asettaa arvoon false silloin kun piirto-operaatio lopetetaan.

- "Poispyyhintä" voidaan suorittaa mousePressed() metodissa silloin kun Alt-nappula on alhaalla.
- "Poispyyhintä" saadaan aikaiseksi kun suorakaiteita laskeva muuttuja merkataan nollaksi ja ruutu maalataan uudestaan.
- MouseEvent -luokkaan on periytynyt metodi isAltDown() jolla Alt-nappulan tila voidaan tutkia.

Harjoitus 4

Lisää ohjelmaan viimeisen suorakaiteen ja kuvan poispyyhintäominaisuus joka toimii siten että jos hiirtä klikattaessa on käytössä jokin muu hiiren näppäin kuin vasen hiiren nappäin, niin viimeksi piirretty suorakaide ja sen sisältämä kuva "pyyhkiytyy" pois. Ohjelmasta HiirenKuunteluApplet.java näet kuinka käytetty hiiren näppäin (nappi) saadaan selville. Tämä tehtävä voidaan ratkaista helposti mousePressed()-metodissa jos edellinen osatehtävä on tehty mainittua boolean-muuttujaa käyttäen.

Harjoitus 5

Ota mallia ohjelmasta MatoOffscreenApplet.java ja toteuta tässä tekemääsi aplettiin ns. tuplapuskurointi jolla estetään ruudun välkkymistä. update()-metodia ei tuplapuskuroinnin toteutuksessa tarvita, mutta paint()-metodista on poistettava kutsu yläluokan paint()-metodiin.

Tarkoitus on että kaikki ruudulle tuleva tavara piirretään ensin "ruudun kuvaan" ja sitten tuo kuva piirretään oikealle ruudulle. Harjoituksessa 2 tehty ominaisuus lakkaa toimimasta kun taustaväri peittyy ruudun täyttävän kuvan alle. Mieti mitenkä saat kyseisen ominaisuuden käyttöön myös ruudun kuvaa käytettäessä.

HARJOITUKSIA OHJELMALLA MovingBallApplet.java

Harjoitus 1:

Lisää aplettiin uusia painonappeja joiden avulla pallon halkaisijaa voidaan kasvattaa ja pienentää. Nykyisellään pallon halkaisija on vakio, 100 pikseliä.

- Tarvitaan kaksi uutta JButton oliota. Lisää ohjelmaan ensin nämä oliot ja tarkista että ne ilmestyvät ruudulle. Vasta tämän jälkeen nappuloihin kannattaa lisätä toimintaa.
- Luokkaan tarvitaan datajäseneksi pallon kokoa kuvaava muuttuja jonka arvon tulee muuttua uusien painonappien avulla. Muuttuja voi olla esim.

```
int pallon_halkaisija  =  100 ;
```

Harjoitus 2:

Lisää ohjelmaan ominaisuus että pallon kokoa voi säätää myös ns. vierityspalkilla joka on luokan JScrollBar olio. Katso mallia ohjelmasta SuorakaideApplet.java ja Java API dokumentaatiosta. (Huomaa että applet_width- ja applet_height-muuttujille on saatava "hyvät" arvot ennenkuin niitä käytetään JScrollBar-olioiden luonnissa.)

Harjoitus 3:

Muuta ohjelma sellaiseksi että pallon ei sallita menevän apletin alueen ulkopuolelle. Tutki JButton-luokan dokumentaatiota ja ota selville kuinka painonappi deaktivoidaan silloin kun pallo on saavuttanut apletin alueen reunan. Painonappi tulee saattaa takaisin aktiiviseksi sitten kun palloa on liikutettu takaisin apletin keskikohtaa päin.

Harjoitus 4:

Muuta pallon liikuttamiseen käytettävät painonapit sellaisiksi että ne rakennetaan kuvien avulla. Tähän voi hyödyntää ImageIcon-luokkaa. ImageIcon-olio saadaan luotua painonappiin seuraavalla tavalla:

```
Image left_button_basic_image =
        getImage( getCodeBase(),
                  "arrow_left_light.gif" ) ;
left_button  =  new JButton(
      new ImageIcon( left_button_basic_image ) ) ;
```

Yllä olevassa esimerkissä luodaan ensin Image-olio jonka avulla tehdään sitten ImageIcon-olio. Näin on hyvä menetellä erityisesti apletteja tehtäessä kun on varauduttava siihen että kuva varmasti latatutuu palvelimelta. Java-applikaatioita tehtäessä voidaan ImageIcon luoda antamalla konstruktorille parametrina kuvatiedoston nimi. Kuvatiedoston tulee tässä tapauksessa olla samassa kansiossa kuin Java-ohjelman .class-tiedosto. Liikuttelunapeissa tarvittavia nuolten kuvia löydät esim. kansiosta http://www.oamk.fi/~karil/images/

Harjoitus 5:

Tutkimalla JButton-luokan tai tarkemmin sanoen sen yläluokan AbstractButton dokumentaatiota saat selville että painonappiin voidaan laittaa myös ns. "roll over icon" joka ilmestyy painonappiin silloin kun hiiri viedään painonapin päälle. Samoin voidaan nappiin laittaa "pressed icon" joka aktivoituu kun nappia painetaan. Laita ainakin yhteen painonappiin tällaiset ikonit. Em. kansiosta löytyy tähän tarkoitukseen erilaisia kuvia ainakin left-nappia varten.

Harjoitus 6:

Ohjelmasta saa näyttävämmän jos myös näytettävä pallo on tehdy kuvan avulla. Osoitteesta http://www.oamk.fi/~karil/images/ löydät ainakin kolme käyttökelpoista pallon kuvaa jotka ovat sellaisia että pallojen ympäristö on tehty kuvaan läpinäkyväksi. Nämä kuvat voi ottaa käyttöön seuraavanlaisilla lauseilla:

```
jalkapallon_kuva =  getImage( getCodeBase(),
                        "soccer_ball_1_round.png" ) ;

koripallon_kuva =  getImage( getCodeBase(),
                        "basketball_round.gif" ) ;

kummallisen_pallon_kuva =  getImage( getCodeBase(),
                        "weird_ball.png" ) ;
```

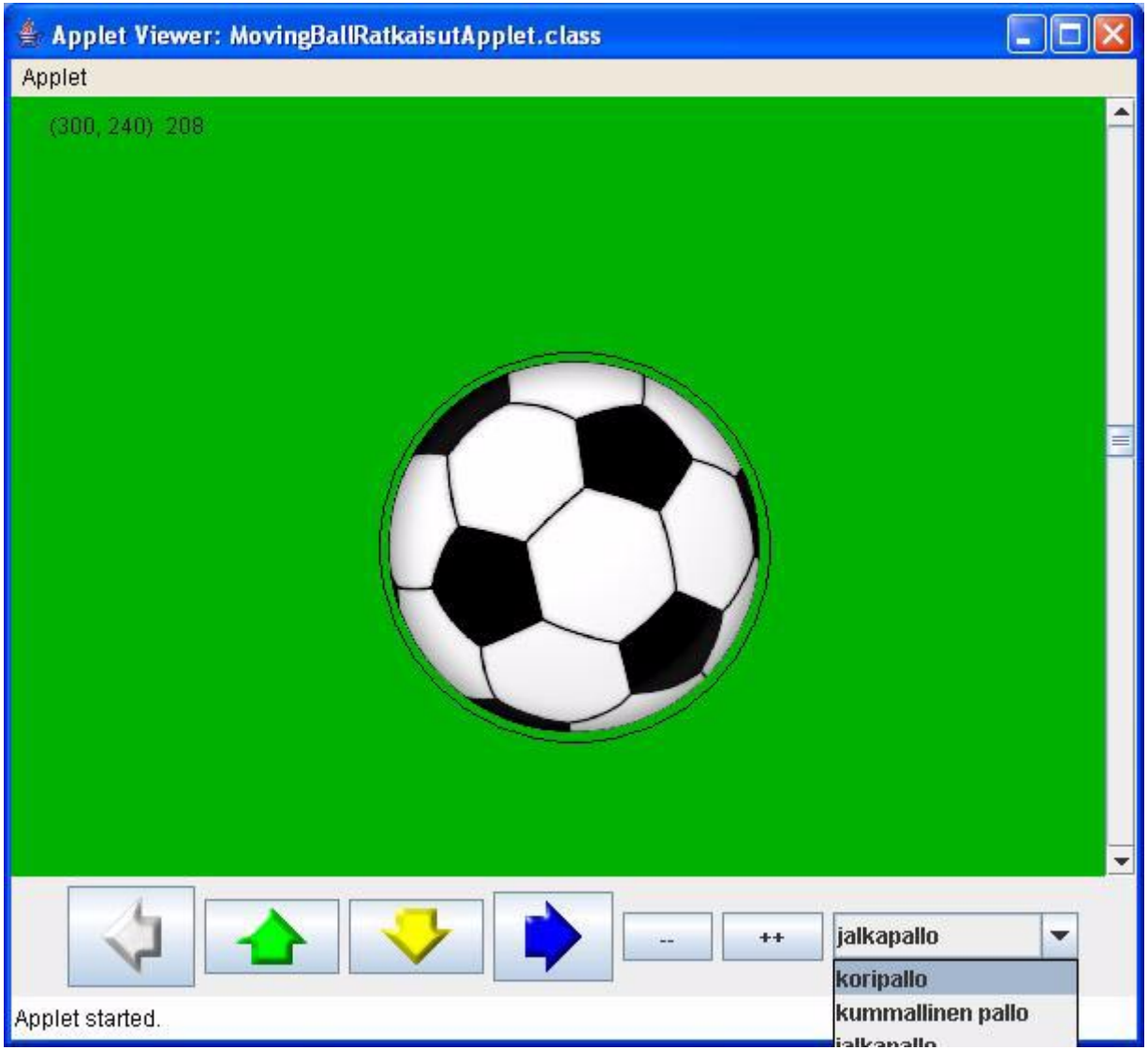
Itse asiassa pallojenkin kuvista kannattaa tehdä ImageIcon-olioita sen tähden että sillä tempulla saa varmistettua että kuvat varmasti latautuvat heti aplettiin. Kuvan voi piirtää seuraavanlaisella lauseella:

```
graphics.drawImage(
    current_ball_image,
    ball_center_point_x - pallon_halkaisija / 2,
    ball_center_point_y - pallon_halkaisija / 2,
    pallon_halkaisija,
    pallon_halkaisija , null) ;
```

Ohjelmasta saa vieläkin näyttävämmän jos eri palloille panee erilaisen taustavärin. itemStateChanged()-metodissa voi siten toimia seuraavalla tavalla:

```
public void itemStateChanged(
                    ItemEvent menu_selection )
{
   String selected_ball  =
                (String) menu_selection.getItem() ;

   if ( selected_ball.equals( "koripallo" ) )
   {
      current_ball_image  =  koripallon_kuva ;
      getContentPane().setBackground( Color.yellow ) ;
   }
   else if ...
```

Kun pallo on kuvana, ohjelman ulkoasu voi näyttää esim. seuraavanlaiselta:


Applet Viewer: MovingBallRatkaisutApplet.class
Applet
(300, 240) 208
--
++
jalkapallo
koripallo
kummallinen pallo
Applet started.

HARJOITUKSIA OHJELMALLA CurtainsApplet.java

Harjoitus 1:

Näissä harjoituksissa on tarkoitus päätyä siihen että ohjelman verhoista tehdään automaattiset siten että ne liikkuvat napin painalluksella animoidusti. Animaation aikaansaamiseksi täytyy käyttää säikeitä. Ennen säikeen tekoon ryhtymistä kannattaa kuitenkin ohjelma muuttaa sellaiseksi että siihen tulee napit verhojen säätöön.

Poista siis ohjelmasta JSlider-säätimet ja pane tilalle painonapit (JButton-oliot) joiden avulla verhot voidaan joko sulkea tai avata. Painonappien käyttöön liittyen voit katsoa mallia MovingBallApplet.java -ohjelmasta. Seuraavat rivit käyvät datakenttien määrittelyksi CurtainsPanel-luokkaan:

```
JButton verhojen_avausnappi    = new JButton( "Open" ) ;
JButton verhojen_sulkemisnappi = new JButton( "Close" ) ;

static final int VERHON_LEVEYS_TAYSIN_KIINNI = WINDOW_WIDTH / 2 + 8 ;
static final int VERHON_LEVEYS_TAYSIN_AUKI   = 10 ;

int verhon_leveys = VERHON_LEVEYS_TAYSIN_KIINNI ;
```

Ohjelma voidaan tehdä siten että verhon leveyttä muutetaan painonappien avulla ja piirtämisessä otetaan aina uusi verhon leveys huomioon.

Harjoitus 2:

Muuta ohjelma sellaiseksi että verhojen avausnapilla verhot avautuvat hiljalleen, ja verhojen sulkunapilla ne menevät hiljalleen kiinni. Tämän aikaansaamiseksi tarvitset säikeen joka hiljalleen muuttaa verhon leveyttä ja suorittaa näytön uudelleenpäivitystä. Säikeen rakentamiseksi seuraavat datajäsenet voivat olla hyödyllisiä:

```
static final int VERHOT_TAYSIN_AUKI     = 0 ;
static final int VERHOT_SULKEUTUMASSA   = 1 ;
static final int VERHOT_AVAUTUMASSA     = 2 ;
static final int VERHOT_TAYSIN_KIINNI   = 3 ;
static final int VERHOT_OSITTAIN_AUKI   = 4 ;

int verhojen_tila  =  VERHOT_TAYSIN_KIINNI ;

Thread  saie_verhojen_liikutteluun ;
```

Tämäntyyppisessä ohjelmassa on verhoilla monta tilaa joita yllä määritellyt tilamuuttujan arvot kuvaavat. Voit katsoa tämän ongelman ratkaisemiseen mallia ohjelmasta AurinkoApplet.java joka löytyy suomenkielisten GUI-ohjelmien kansiosta. Tämä verhojen leveyden säätöongelma on itse asiassa helpompi kuin tuo AurinkoApplet.javan auringonliikutteluongelma.

Harjoitus 3:

Säikeitä käytettäessä on vaarana että ohjelmaan saadaan aikaiseksi hyvin kummallisia ja vaikeasti löydettäviä virheitä. Voi esimerkiksi jäädä useita säikeitä yhtäaikaisesti toimintaan.

Verhojen liikuttelussa voi syntyä ongelmia jos painetaan jotain nappia silloin kun animaatio on kesken.

Tee niin että ohjelman painonapit deaktivoidaan setEnabled()-metodilla siksi ajaksi kun animaatio on käynnissä. Tällä voidaan estää mahdollisia ongelmia.

Harjoitus 4:

Lisää ohjelmaan Stop-nappi jolla verhojen liike voidaan pysäyttää siinä tapauksessa kun ne ovat joko sulkeutumassa tai avautumassa.

Stop-nappi pitää sitten jättä deaktivoimatta siinä tapauksessa kun animaatio aloitetaan. Stop-napin tulee olla deaktiivinen silloin kun avaamis- ja sulkemisnapit ovat aktiivisia.

Verhojen pysäytyksen jälkeen verhojen tilaksi tulee VERHOT_OSITTAIN_AUKI. Tuosta tilasta pitäisi päästä taas joko verhojen avaamiseen tai sulkemiseen kun nappeja käytetään.

Harjoitus 5:

Lisää ikkunaan verhojen taakse sälekaihtimet. Ne saat aikaiseksi piirtämällä silmukalla joukon vaakasuoria matalia suorakaiteita (säleitä) ennen varsinaisten verhojen piirtämistä. Yleensähän sälekaihtimet ovat vaaleanharmaita väriltään.

Sälekaihtimet ovat englanniksi Blinds tai Venetian Blinds.

Lisää sälekaihdinten säätöön Blinds-painonappi jolla sälekaihtimet saa enemmän tai vähemmän kaihtavaksi. Tämä tarkoittaa säleen korkeuden säätämistä painonapin avulla. Ohjelmassa voit käyttää datakenttää

```
int saleen_korkeus = 1 ;
```

jonka arvo muuttuu Blinds-napin painamisen jälkeen esim seuraavasti:

```
saleen_korkeus = saleen_korkeus + 4 ;

if ( saleen_korkeus > 16 )
{
   saleen_korkeus = 1 ;
}
```

Sälekaihtimien säleen korkeus voi siis painonapin avulla saada ensin suurempia arvoja ja sitten taas aloittaa alusta. Näin käyttäjä voi etsiä sopivan asennon kun painaa nappia riittävän monta kertaa. Säleiden korkeuden säätöön ei tarvita animaatiota.

HARJOITUKSIA OHJELMALLA PalloLiikkuuHiirellaApplet.java

Näissä harjoituksissa tulee olla ohjelma Pallo.java samassa kansiossa kuin PalloLiikkuuHiirellaApplet.java. Luokkaan Pallo ei tarvitse kuitenkaan tehdä mitään muutoksia.

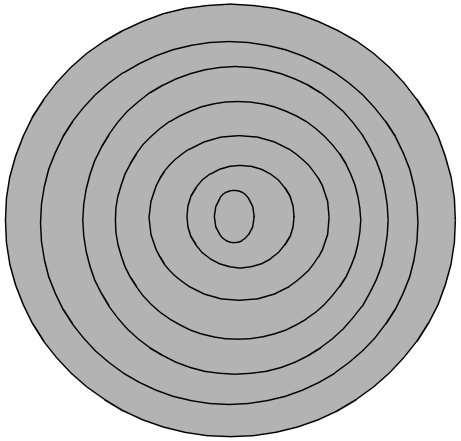
Harjoitus 1:

Muuta ohjelmaa ottamalla neljäs pallo käyttöön. Tässä tulee määritellä aplettiluokkaan neljäs Pallo-tyyppinen olioviittaaja datajäseneksi, sekä lisätä metodeihin toiminnot joilla tämä neljäskin pallo otetaan huomioon. Metodeihin mouseReleased() ja mouseDragged() ei tässä tarvitse tehdä muutoksia. Neljäs pallo tulee alussa keskimmäisen pallon kohdalle kun se luodaan seuraavanlaisella lauseella:

```
neljas_pallo  =  new  Pallo( apletin_leveys / 2,
                             apletin_korkeus / 2  + 100,
                             Color.cyan ) ;
```

Harjoitus 2:

Johda (periytä) luokasta Pallo uusi luokka nimeltä Rengaspallo. Voit kirjoittaa Rengaspallo-luokan esim. tiedoston PalloLiikkuuHiirelläApplet.java alkuun import-lauseiden jälkeen kunhan et tee uudesta luokasta public-luokkaa. Tarkoitus on että Rengaspallo-oliot ovat samanlaisia kuin Pallo-oliot sillä poikkeuksella että Rengaspallo-luokan pallo sisältää mustia renkaita seuraavaan tapaan:



Piirtoteknisistä syistä tässä esitetyt renkaat eivät ole ihan ympyröitä, mutta Graphics-luokan metodilla drawOval() on kätevä piirtää ihan oikeita ympyröitä..

Rengaspallo-luokkaan tulee kirjoittaa konstruktori ja uusi versio metodista piirra(). Uusia datajäseniä luokkaan ei tarvitse lisätä. Kun konstruktori tehdään seuraavalla tavalla, ei Pallo-luokassa tarvita oletuskonstruktoria:

```
public Rengaspallo( int annettu_paikka_x,
                    int annettu_paikka_y,
                    Color annettu_vari )
{
   super( annettu_paikka_x,
          annettu_paikka_y, annettu_vari ) ;
}
```

Metodi piirra() voidaan tehdä seuraavalla tavalla, jossa ensin kutsutaan Pallo-luokan vastaavaa metodia ja vasta sitten piirretään pallon sisään tarvittavat mustat renkaat:

```
public void piirra( Graphics graphics )
{
   super.piirra( graphics ) ;  //  Kutsu Pallo-luokan piirra()-metodiin

   graphics.setColor( Color.black ) ;

   int  renkaan_halkaisija  =  pallon_halkaisija ;

   while  ( renkaan_halkaisija  >  0 )
   {
      graphics.drawOval( pallon_keskipiste_x  -  renkaan_halkaisija / 2,
                         pallon_keskipiste_y  -  renkaan_halkaisija / 2,
                         renkaan_halkaisija, renkaan_halkaisija ) ;

      renkaan_halkaisija  =  renkaan_halkaisija  -  8 ;
   }
}
```

Kun olet saanut aikaiseksi uuden Rengaspallo-luokan, sinun tulee luonnollisesti ottaa käyttöön aplettiluokassa Rengaspallo-olio, jolla voit testata tekemääsi luokkaa. Rengaspallo-olion lisääminen voidaan tehdä samaan tapaan kuin harjoituksessa 1 lisättiin uusi Pallo-olio.

Harjoitus 3:

Nykyisessä PalloLiikkuuHiirellaAppletissa on se ongelma että viimeisenä liikutettu pallo ei välttämättä jää päälimmäiseksi jos pallot tulevat ruudulle osittain päällekkäin. Ongelma johtuu siitä että paint()-metodissa pallot piirretään aina samassa järjestyksessä.

Luonnollisesti olisi ohjelman käyttäjän kannalta loogista että viimeisenä liikutettu pallo on päällimmäisenä. Tässä harjoituksessa tehtäväsi on poistaa tämä ongelma ohjelmasta. Tämä ongelma voidaan poistaa mm. siten että aplettiluokkaan lisätään datajäseneksi taulukko johon myös tarvittavat olioviittaajat talletetaan. Kun taulukkona käytetään java.util -paketissa määriteltyyn ArrayList-luokkaan perustuvaa taulukkoa, on helppo poistaa juuri liikutettua palloa vastaava olio taulukosta ja lisätä se sitten taulukon loppuun, minkä seurauksena viimeisenä liikuteltu pallo piirtyy myös viimeisenä. Tarvittava ArrayList-taulukko voidaan määritellä ja luoda esim. seuraavalla lauseella

```
ArrayList<Pallo>  piirrettavat_pallot  =
                              new  ArrayList<Pallo>() ;
```

ArrayList-taulukkoa määriteltäessä annetaan kulmasulkeissa < > sen luokan nimi, jonka olioita tai jonka alaluokan olioita taulukkoon aiotaan tallettaa.

Kaikki olioviittaajat voidaan tallettaa ArrayList-taulukkoon esim. PalloLiikkuuHiirellaPanel -luokan konstruktorin lopussa seuraavaan tapaan:

```
piirrettavat_pallot.add( eka_pallo ) ;
piirrettavat_pallot.add( toka_pallo ) ;
jne.
```

Kun Pallo- ja Rengaspallo-olioihin viitataan ArrayList-taulukosta, kaikki pallot voidaan tulostaa paintComponent()-metodissa seuraavanlaisella silmukalla. size()-metodilla saadaan selville montako oliota ArrayList-taulukkoon on talletettu, ja get()-metodilla voidaan lukea tiettyä indeksiarvoa vastaava olio.

```
for ( int pallon_indeksi  =  0 ;
          pallon_indeksi  <  piirrettavat_pallot.size() ;
          pallon_indeksi  ++ )
{
   piirrettavat_pallot.get( pallon_indeksi ).piirra( graphics ) ;
}
```

Kun sitten halutaan asettaa viimeksi liikuteltu pallo viimeksi piirrettäväksi, se voidaan tehdä mousePressed()-metodissa heti kun on löydetty liikuteltava pallo. mousePressed()-metodissa voidaan liikuteltava pallo poistaa ensin ArrayList-taulukosta seuraavanlaisella lauseella:

```
piirrettavat_pallot.remove( liikuteltava_pallo ) ;
```

Kun liikuteltava_pallo:n viittaama olio lisätään tämän jälkeen add()-metodilla takaisin taulukkoon, se tulee taulukon viimeiseksi ja piirtyy siten viimeisenä. Nämä operaatiot on luonnollisesti suoritettava sen jälkeen kun liikuteltava_pallo on pantu viittaamaan oikeaan liikuteltavaan palloon.

Harjoitus 4:

Ohjelmassa on vielä sellainen vika että jos palloja on päällekkäin ruudulla ja klikataan päällimmäistä palloa, aina ei välttämättä tartuta päälimmäiseen pallon vaan hiiren kursoriin saattaa tarttua pallo jostain alta. Muuta mousePressed()-metodi sellaiseksi että tartutaan aina päällimmäiseen palloon. Edellisessä kohdassa käytettyä taulukkoa voidaan tässä hyödyntää. Kyseinen taulukko on sellaisessa järjestyksessä että päällimmäisenä olevat pallot ovat taulukon lopussa. Tämän tehtävän ratkaisussa voi käyttää seuraavantapaista ohjelmarakennetta, jossa taulukkoa käydään läpi lopusta alkuun:

```
int pallon_indeksi  =  piirrettavat_pallot.size() - 1 ;

boolean klikatun_pallon_etsinta_valmis  =  false ;

while ( klikatun_pallon_etsinta_valmis == false )
{
   // Tanne pitaisi jotain sopivaa ohjelmakoodia laittaa



   if ( pallon_indeksi  >  0 )
   {
      pallon_indeksi -- ;
   }
   else
   {
      klikatun_pallon_etsinta_valmis  =  true ;
   }
}
```

Harjoitus 5:

Kun sinulla nyt on olioviittaukset Pallo-olioihin ArrayList-taulukossa, voit helposti tehdä ohjelmasta sellaisen että ruudulla voi olla "äärettömästi" palloja. Muuta mousePressed()-metodi sellaiseksi että hiiren keskimmäisellä napilla ArrayList-taulukon loppuun lisätään uusi Rengaspallo-olio ja oikealla hiiren napilla sinne lisätään tavallinen Pallo-olio. Nämä uudet pallot piirtyvät ruudulle automaattisesti jos edelliset osatehtävät on tehty oikein. Kun kaikki viittaukset pallo-olioihin ovat ArrayList-taulukossa, voit niin halutessasi luopua kokonaan olioviittaajista eka_pallo, toka_pallo, jne.

HARJOITUKSIA OHJELMALLA PelikortitApplet.java

Harjoitus 1:

Muuta Kortti-luokan piirra()-metodia siten että punaisten maiden numerotiedot kirjoitetaan punaisella ja mustien maiden numerotiedot mustalla. Tässä muutos kohdistuu vain Kortti-luokan piirra()-metodiin, koska Kortti-olio on sellainen että se tietää itse kuinka se piirtyy ruudulle.

Harjoitus 2:

Muuta ohjelma sellaiseksi että korttipakka on valmiiksi sekoitettu silloin kun ohjelma käynnistyy.

Harjoitus 3:

Tee ohjelmasta yksinkertainen pokerikäden parannuspeli siten että rivi_kortteja -taulukosta ja siten myös näytöltä poistetaan klikatut kortit, ja sitten kun painetaan Jaa kortit -nappia niin poistettujen korttien tilalle otetaan taulukon loppuun uudet kortit. Tarkoitus on että pelaaja voi poistaa rivistä kortteja ja ottaa tilalle uusia kunnes hän on saanut aikaiseksi esim. jonkin pokeripelin "käden". ("Yksinäisen" kortin voi tässä jättää käyttämättömäksi. Kortit kannattaa heti alussa kääntää oikeinpäin.)
Korttien poistaminen ArrayList-pohjaisesta taulukosta voi tapahtua lauseella

```
rivi_kortteja.remove( kortin_indeksi ) ;
```

Tässä siis halutaan että Jaa kortit -nappi ei tyhjennä koko korttiriviä, vaan lisää sinne vaan

puuttuvan määrän kortteja loppuun. Kun kortteja halutaan lisätä epätäyteen korttiriviin, se voidaan tehdä seuraavanlaisella silmukalla:

```
int kortteja_rivissa = rivi_kortteja.size() ;

while ( kortteja_rivissa < 5 )
{
   Kortti uusi_kortti_riviin  =  korttipakka.ota_kortti() ;
   uusi_kortti_riviin.kaanna_kortti() ;

   rivi_kortteja.add( uusi_kortti_riviin ) ;

   kortteja_rivissa ++ ;
}
```

Korttirivissä olevien korttien paikat näytöllä voidaan asettaa uudelleen seuraavanlaisella silmukalla:

```
for ( int kortin_indeksi  =  0 ;
          kortin_indeksi  <  rivi_kortteja.size() ;
          kortin_indeksi  ++ )
{
   Point  kortin_paikka  =  new  Point(
      40 + ( Kortti.KORTIN_LEVEYS_PIKSELEINA +20 ) * kortin_indeksi,
      50 ) ;

   rivi_kortteja.get( kortin_indeksi ).
                          aseta_paikka_pikseleina( kortin_paikka ) ;
}
```

Harjoitus 4:

Lisää ohjelmaan ominaisuus että se tutkii ovatko rivi_kortteja-taulukossa olevat kortit samaa maata. Mikäli ne ovat samaa maata, eli kortit muodostavat pokeripelin värin, ohjelma ilmoittaa tästä jotenkin käyttäjälleen. Eräs tapa ilmoittaa että väri on löytynyt on muuttaa apletin taustaväri esim. punaiseksi. Kortti-luokassa on valmiina metodi nimeltä on_samaa_maata_kuin() jolla jotain Kortti-oliota voidaan verrata maan suhteen johonkin toiseen Kortti-olioon. Tässä voi toimia esim. siten että taulukon ensimmäistä korttia verrataan ehtolausekkeessa vuorotellen taulukon muihin kortteihin.

Harjoitus 5:

Muuta Sekoita pakka -nappi Uusi peli -napiksi. Kun tätä nappia painetaan rivi_kortteja -taulukko tyhjennetään, luodaan uusi korttipakka, sekoitetaan uusi pakka ja asetetaan taustaväri alkuperäiseen arvoonsa.

Harjoitus 6:

Muuta ohjelma enemmän peliksi siten että liität siihen yksinkertaisen ajanmittaustoiminnon. Eräs tapa ajan mittaukseen on käyttää System.currentTimeMillis()-metodia, joka palauttaa kutsuhetkiset koneen kellon millisekunnit. Metodia voidaan käyttää seuraavasti:

```
long nykyinen_aika = System.currentTimeMillis() ;
```

Kun alussa otetaan talteen pelin aloitusaika seuraavasti

```
long aloitusaika = System.currentTimeMillis() ;
```

saadaan kulunut aika millisekunteina selville kun nykyisestä ajasta vähennetään aloitusaika.

Kulunut aika voidaan aina päivittää ja näyttää sen jälkeen kun kortteja käännellään tai uusia kortteja jaetaan. Lopussa, kun väri on löytynyt, näytetään lopullinen kulunut aika.

Aloitusaikaa voidaan alkaa mitata esimerkiksi silloin kun käyttäjä ensimmäisen kerran painaa Jaa kortit -nappia. Tässä kannattaa käyttää datajäsenenä muuttujaa

```
boolean peli_kaynnissa = false ;
```

jolla kontrolloidaan pelin kulku.

Tässä pelissä mahdollisimman pieni aika on hyvä tulos. Aikaan vaikuttaa se kuinka nopeasti pelaaja kääntelee kortteja ja lisäksi se mitenkä kortit sattuvat pakassa olemaan.

Mieti mitenkä reaaliaikainen kuluneen ajan näyttö voitaisiin toteuttaa.

HARJOITUKSIA OHJELMALLA BouncingBallApplet.java

Harjoitus 1:

Laita aluksi ohjelmaan toinen pomppiva pallo. Tämän saat aikaan kun laitat BouncingBallPanel-luokkaan datajäsenen

```
ExplodingBouncer toinen_pallo ;
```

ja sitten ko. luokan konstruktorissa luot ExplodingBouncer-olion seuraavaan tapaan

```
toinen_pallo  =  new  ExplodingBouncer(
                 new Point2D.Double( applet_width / 2,
                                     applet_height / 2 ),
                 Color.CYAN, bouncing_area) ;
```

Yllä olevalla tavalla tehtynä toisella pallolla on alussa sama paikka kuin ’vanhalla’ pallolla. Uusi pallo lähtee kuitenkin lentämään eri suuntaan koska pallon suunta arvotaan satunnaisesti.

Jotta saat uuden pallon liikkumaan, tulee run()-metodiin lisätä seuraava kutsu

```
toinen_pallo.move() ;
```

Jotta saat uuden pallon piirtymään, tulee paintComponent()-metodiin lisätä seuraava kutsu

```
toinen_pallo.draw( graphics2D ) ;
```

Harjoitus 2:

Muuta ohjelma sellaiseksi että sen käynnistyessä lähtee 10 palloa pomppimaan näytöllä. Tämä voidaan tehdä loogisessa mielessä samaan tapaan kuin toimittiin edellisen kohdan yhden pallon kanssa. Tässä voi käyttää java.util -paketissa olevaa ArrayList-luokkaa jolla voidaan rakentaa dynaaminen taulukko. Taulukko BouncingBall-olioille voidaan määritellä seuraavasti

```
ArrayList<ExplodingBouncer> pelin_pallot =
                    new ArrayList<ExplodingBouncer>();
```

ja oliot taulukkoon voidaan tehdä seuraavasti

```
for ( int pallolaskuri = 0 ;
          pallolaskuri < 10 ;
          pallolaskuri ++ )
{
   ExplodingBouncer uusi_pallo =
      new ExplodingBouncer(
           new Point2D.Double( applet_width / 2,
                               applet_height / 2 ),
                               Color.MAGENTA,
                               bouncing_area) ;

   pelin_pallot.add( uusi_pallo ) ;

}
```

Taulukossa olevia palloja voidaan liikutella seuraavanlaisella silmukalla

```
for ( ExplodingBouncer liikuteltava_pallo : pelin_pallot )
{
   liikuteltava_pallo.move() ;
}
```

Taulukossa olevat pallot voidaan piirtää seuraavanlaisella silmukalla

```
for ( ExplodingBouncer piirrettava_pallo : pelin_pallot )
{
   piirrettava_pallo.draw( graphics2D ) ;
}
```

Kun olet saanut ArrayList-taulukossa olevat pallot toimimaan, voit poistaa alkuperäisen pallon ja alussa lisäämäsi pallon ohjelmasta. (Voit helposti lisätä palloja ArrayList-taulukkoon jos tuntuu että pelissäsi on liian vähän palloja.)

Harjoitus 3:

Tee kaikille palloille erilainen pohjaväri. Voit käyttää konstruktorissa seuraavaa taulukkoa, josta otat indeksoimalla pallon värin ennen pallo-olion luontia:

```
Color[] ball_colors =
    { Color.CYAN, Color.MAGENTA, Color.YELLOW,
      Color.PINK, Color.WHITE, Color.BLACK,
      Color.BLUE, Color.ORANGE, Color.RED,
      Color.GREEN.brighter() } ;
```

Tämän tehtävän teko ei ole edellytys seuraaavan tehtävän tekemiselle.

Harjoitus 4:

Ohjelmassa käytetyille ExplodingBouncer-olioille on käytössä metodi nimeltä contains_point(), jolla voidaan tutkia onko jokin piste pallon eli ’pomppijan’ alueen sisällä. Lisäksi on käytössä metodi explode_ball(), jolla pallo saadaan ’räjähtämään’.

Tehtäväsi on tässä lisätä hiiritoiminto mukaan ohjelmaan siten että hiirellä jotakin palloa klikattaessa se räjähtää ja hiljalleen katoaa näytöltä. Tässä siis ohjelmasta tulee eräänlainen pallojentuhoamispeli.

Esim. silloin kun hiiren nappi painetaan alas voidaan ’kysyä’ jokaiselta ArrayList-taulukossa olevalta pallolta että sattuiko klikattu piste pallon alueelle. Sitten tuhotaan kyseinen pallo jos klikkaus osui sen alueelle.

Huomaa, että pallon tuhoamiseen riittää kun kutsutaan ko. pallon suhteen explode_ball()-metodia. ExplodingBouncer-luokassa on jo valmiina automatiikka jolla pallo tuhoutuu hiljalleen sitten kun se on räjäytetty explode_ball()-metodilla.

Ohjelman alkuperäisessä versiossa explode_ball()-metodia kutsutaan silloin kun painetaan näppäimistön Esc-näppäintä.

Harjoitus 5:

Lisää peliin ominaisuus että se käynnistyy vasta kun Space-näppäintä eli välilyöntinäppäintä painetaan. Tämä on aika helppo tehdä kun määrittelee BouncingBallPanel-luokkaan datajäsenen

```
boolean peli_kaynnissa = false ;
```

jolle annetaan arvo true sitten kun Space-näppäintä on painettu.

Tarkoitus on että palloja aletaan liikuttelemaan vasta sitten kun yllä mainittu muuttuja on saanut arvon true. Pallot kyllä voidaan piirtää vaikka mainitulla muuttujalla on arvo false. Tällöin ne piirtyvät päällekäin keskelle ohjelman piirtoaluetta. Pallot sitten sinkoutuvat eri suuntiin automaattisesti kun Space-näppäintä on painettu.

Harjoitus 6:

Tee ohjelmaan ominaisuus että kun kaikki pallot on räjäytetty, siinä tuhotaan ArrayList-taulukosta kaiki tuhoutuneet pallot ja luodaan pallot uudestaan. Ohjelma voi taas tässä tilanteessa jäädä odottamaan Space-näppäimen painallusta.

Tässä on ExplodingBouncer-luokkaan tehtävä boolean-tyyppinen metodi is_exploded() joka palauttaa arvon true siinä tapauksessa kun pallon tila on BALL_EXPLODED. Tuon metodin avulla tulee sitten kaikkien pallojen tila tutkia esim. ennen niiden liikuttamista. Jos kaikki pallot ovat räjähtäneet, toimitaan tässä tehtävässä määritellyllä tavalla. Erillinen metodi tarvitaan koska pallo ei ole räjähtänyt heti kun explode_ball()-metodia on kutsuttu.

HARJOITUKSIA OHJELMALLA FlagsApplet.java

Harjoitus 1:

Ohjelma FlagsApplet.java on esimerkki mm. standardiluokan JTabbedPane käytöstä. Kyseisellä luokalla voidaan rakentaa graafinen käyttöliittymä joka koostuu "tabeista" eli välilehdistä. Tutki luokan JTabbedPane dokumentaatiota ja muuta FlagsApplet.java sellaiseksi että sen välilehdet voidaan valita näppäimistön avulla. Tämä pitäisi saada aikaiseksi setMnemonicAt()-metodin avulla.

Käytä myös setToolTipTextAt()-metodia jolla välilehdille voidaan asettaa "vinkkiteksti" joka tulee näkyviin kun hiiri viedään välilehden valinnan kohdalle.

Harjoitus 2:

Lisää ohjelmaan uusi välilehti "Fictitious" johon kuuluvaan FlagPanel-olioon lisätään kuvitteellisten maiden lippuja. Tätä ominaisuutta rakentaessasi voit sinne laittaa aluksi ihan jonkin jo määritellyn lippuolion, mutta suositus on että määrittelet CrossFlag-luokan avulla "AntiFinland" -nimisen maan lipun ja laitat tuon lipun tuohon "Fictitious"-lippujen välilehdelle. "AntiFinland" -lipussa on sininen tausta ja siinä valkoinen risti. "AntiFinland"-lipussa voi risti olla horisontaalisesti toisinpäin verrattuna Suomen lippuun.

Tätä harjoitusta tehdessäsi voit myös korjata FlagPanel-luokasta sen ongelman että se ei toimi jos paneeliin ei ole lisätty yhtään lippua. Tämän ongelman syy on se että ArrayList-taulukosta yritetään näyttää aina ensimmäistä lippua, ja jos tuota ensimmäistäkään lippua ei ole niin syntyy ongelmia. Tämä ongelma voidaan ratkaista tutkimalla size()-metodilla FlagPanel-luokan paintComponent()-metodissa, onko ArrayList-taulukko tyhjä.

Harjoitus 3:

FlagApplet.java-ohjelmassa on luokasta Flag johdettu useita alaluokkia joiden oliot edustavat erityyppisiä lippuja. Kussakin alaluokassa on draw()-metodi joka osaa piirtää juuri kyseisen tyyppisen lipun. Tuo draw()-metodi on abstrakti metodi Flag-luokassa. Kun tämmöinen metodi on käytössä, voi ArrayList<Flag> -tyyppisessä taulukossa olla erityyppisiä lippuolioita ja ohjelman suoriintuessa kunkin lippuolion piirtämiseksi valitaan automaattisesti oikea draw()-metodi.

Johda Flag-luokasta uusi luokka joka edustaa sellaisia lippuja joiden keskellä on pallonmuotoinen kuvio. Esim. Japanin lipussa on punainen pallo (keskellä) valkealla pohjalla. Bangladeshin lipussa on punertava pallo vihereällä pohjalla. Uuden luokan nimi voi olla BallFlag ja sen draw()-metodi piirtää siis automaattisesti määritellynvärisen pallon keskelle lippua. (Voit käyttää Bangladeshin lippua testilippuna, vaikka sen pallo ei taida virallisesti sattuakaan ihan keskelle.)

Tämän uuden luokan testaamiseksi tulee jollekin välilehdelle lisätä BallFlag-lippu(ja).

Harjoitus 4:

Johda Flag-luokasta uusi luokka nimeltä SingleStarFlag. Tuon luokan liput ovat sellaisia että niissä on yksi viisihaarainen tähti keskellä lippua. Tämän tyyppinen lippu on esim. Vietnamilla, Somalialla ja Marokolla, vaikka tähti ei olekaan kaikissa näissä lipuissa ihan samankokoinen.

Tähtikuvion piirtämiseksi kannattaa käyttää Javan 2D-grafiikkaa ja GeneralPath-luokkaa. Tutki ohjelmaa StarsApplet.java (löytyy kansiosta http://www.naturalprogramming/javagui/) jossa näitä Javan ominaisuuksia on käytetty. Ohjelmasta löytyy itse asiassa valmis luokka viisisakaraisen tähden piirtämiseksi. Voit kopioida tuon luokan ja sen tarvitseman yläluokan FlagsApplet.java-ohjelmaan. Tuota luokkaa StarShape5 käyttämällä lippuun tarvittava tähti voidaan määritellä oliona ja se voidaan piirtää helposti StarShape5-luokan draw()-metodin avulla.

HARJOITUKSIA OHJELMALLA DrawingGraphicalObjectsApplet.java

Harjoitus 1:

Lisää Settings-menun alamenuun "Select line thickness" mahdollisuus antaa viivan paksuudeksi "Medium" paksuus, joka voi olla esimerkiksi 6 pistettä eli pikseliä. Pistä tämä Medium-paksuus oletuspaksuudeksi. Tässä osatehtävässä tarvitsee muuttaa vain pääluokkaa eli DrawingGraphicalObjectsApplet-luokkaa.

Harjoitus 2:

Lisää menuihin ja ohjelmaan mahdollisuus valita käytettävä piirtoväri. Pistä menuihin esimerkiksi kolme eri värivaihtoehtoa. GraphicalObjectsPanel-luokassa on jo valmiina metodi set_drawing_color(), jolla valittu piirtoväri saadaan kyseisen luokan tiedoksi. Käytettävä väri voidaan antaa luokkien CurvyLine ja Circle konstruksoreille silloin kun näitä olioita luodaan. Näitä konstruktoreita on siis muutettava jotta ohjelman saa toimimaan. Luokassa GraphicalObject on jo tarpeellinen piirtovärin tallettava datakenttä eli datajäsen.

Harjoitus 3:

Lisää ohjelmaan mahdollisuus suorien viivojen piirtämiseen. Luokasta GraphicalObject täytyy tässä johtaa eli periyttää tarpeellinen alaluokka kuten esim. StraightLine. Tähän luokkaan tulee sitten kirjoittaa suoran viivan piirtävä draw()-metodi. Lisäksi ohjelmaan tulee tehdä tarpeelliset menu- yms. muutokset.

Harjoitus 4:

Lisää ohjelmaan Edit-menu, johon laitat Clear all - ja Clear last -toiminnot. Näistä toiminnoista ensimmäinen poistaa kaikki piirrokset ruudulta ja jälkimmäinen poistaa viimeksi tehdyn piirroksen. Tämän ominaisuuden tekemiseksi tulee ArrayList-taulukosta poistaa joko kaikki oliot tai viimeisin olio

Harjoitus 5:

Tämäntyyppisissä piirtelyohjelmissa on yleensä jonkinlainen palkki tai paletti josta piirrostoiminnot valitaan. Jos intoa piisaa laita ohjelmaan Toolbar, josta menuissa olevat toiminnot voi valita. MenuDemoWithToolbarApplication.java voi olla tässä hyödyllinen esimerkkiohjelma.

JAVA WINDOWS-SOVELLUKSIIN LIITTYVÄ HARJOITUS

Harjoitus 1:

MovingBallWindow.java on esimerkki Javalla tehdystä Windows-sovelluksesta. Katso mallia tuosta ohjelmasta, ja tee aplettia DrawingLinesApplet.java vastaava Windows-sovellus. Voit tehdä tämän homman kopioimalla ensin em. apletin nimelle DrawingLinesWindow.java, ja tekemällä siihen tarvittavat muutokset. Tarvittavat muutokset ovat:

- Luokka johdetaan standardiluokasta Frame ja se implementoi WindowListener -rajapinnan.
- Luokassa on hyvä olla staattiset datajäsenet jotka määrittelevät ikkunan koon.
- Luokassa tulee olla konstruktori init() -metodin sijaan, ja konstruktorin lopussa asetetaan ikkunan ominaisuudet.
- WindowListener -rajapinnan toteuttavat metodit voidaan kopioida suoraan ohjelmasta MovingBallWindow.java
- Tulee olla olemassa main()-metodi, jossa DrawingLinesWindow-luokan olio luodaan ja pannaan näkyviin.

Harjoitus 2:

Lisää ominaisuus, että ikkunan otsikoksi muutetaan WINDOW FULL silloin kun ikkunaan ei voi piirtää enempää viivoja.

Harjoitus 3:

Ota mallia ohjelmasta PerinteinenWindowsSovellus.java, ja tee viivojenpiirtosovellukseen Edit-menu jonka sisällä on valintamahdollisuus Clear. Kun Clear-valinta tehdään tulee ikkunasta pyyhkiytyä kaikkien viivojen pois.

Viivojen "poispyyhintä" voidaan tehdä yksinkertaisesti siten että viivojen määrää kuvaavan datajäsenen arvo merkitään nollaksi.