Oliosuuntautuneita apletteja

Luokka **Pallo** on yleiskäyttöinen luokka jota voidaan käyttää kun halutaan piirtää liikuteltavia palloja Java-apletteihin.

Pallo-oliolle määritellään sitä luotaessa paikka ja väri. Nämä tiedot tulee siis antaa **Pallo**-luokan konstruktorille. Pallo siis tietää paikkansa ruudulla ja tietää itse minkä värinen se on.

```
//  Pallo.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.geom.* ;

class Pallo
{
   int pallon_keskipiste_x  =  0 ;
   int pallon_keskipiste_y  =  0 ;

   Color pallon_vari  =  Color.red ;

   int pallon_halkaisija  =  100 ;

   boolean  pallo_on_aktivoitu  =  false ;

   public Pallo( int annettu_paikka_x,
                 int annettu_paikka_y,
                 Color annettu_vari )
   {
      pallon_keskipiste_x  =  annettu_paikka_x ;
      pallon_keskipiste_y  =  annettu_paikka_y ;
      pallon_vari      =  annettu_vari ;
   }

   public void aktivoi_pallo()
   {
      pallo_on_aktivoitu  =  true ;
   }

   public void deaktivoi_pallo()
   {
      pallo_on_aktivoitu  =  false ;
   }

   int lue_pallon_keskipiste_x()
   {
      return pallon_keskipiste_x ;
   }

   int lue_pallon_keskipiste_y()
   {
      return pallon_keskipiste_y ;
   }
```

Pallo.java - 1: Yleiskäyttöinen liikuteltavaa palloa edustava luokka.

Metodi **muuta_pallon_paikka()** siirtää pallon kokonaan uuteen paikkaan ruudulla. Muut palloa siirtelevät metodit muuttavat pallon paikkaa suhteessa sen entiseen paikkaan.

```
public int lue_pallon_halkaisija()
{
   return pallon_halkaisija ;
}

public void siirra_oikealle()
{
   pallon_keskipiste_x  +=  3 ;
}

public void siirra_vasemmalle()
{
   pallon_keskipiste_x  -=  3 ;
}

public void siirra_ylos()
{
   pallon_keskipiste_y  -=  3 ;
}

public void siirra_alas()
{
   pallon_keskipiste_y  +=  3 ;
}

public void  siirra_palloa( int siirtyma_suunnassa_x,
                            int siirtyma_suunnassa_y )
{
   pallon_keskipiste_x  =  pallon_keskipiste_x  +  siirtyma_suunnassa_x ;
   pallon_keskipiste_y  =  pallon_keskipiste_y  +  siirtyma_suunnassa_y ;
}

public void muuta_pallon_paikka( int uusi_paikka_x,
                                 int uusi_paikka_y )
{
   pallon_keskipiste_x  =  uusi_paikka_x ;
   pallon_keskipiste_y  =  uusi_paikka_y ;
}

public void pienenna()
{
   //  if-rakenteella varmistetaan etta pallo ei mene liian pieneksi.

   if ( pallon_halkaisija  > 10 )
   {
      pallon_halkaisija  -=  6 ;
   }
}
```

Pallo.java - 2: Pallo-luokan metodeita.

Metodi **suurenna()** muuntaa pallon halkaisijaa. Pallon paikkakoordinaatteja ei tarvitse muuttaa koska muuttunut halkaisija otetaan aina piirrettäessä huomioon.

Metodilla **muuta_halkaisija()** voidaan pallon kokoa muuttaa siten että uusi koko ei ole riippuvainen entisestä pallon koosta.

```
public void suurenna()
{
   pallon_halkaisija  +=  6 ;
}

// Myos seuraavalla metodilla voi muuttaa pallo-olion
// kokoa.

public void muuta_halkaisija( int uusi_halkaisija )
{
   if ( uusi_halkaisija  >  5 )
   {
      pallon_halkaisija  =  uusi_halkaisija ;

   }
}

public void muuta_vari( Color uusi_vari )
{
   pallon_vari  =  uusi_vari ;
}

public boolean piste_on_pallolla( Point annettu_piste )
{
   Area  taman_pallon_alue  =  new  Area(

         new  Ellipse2D.Float( pallon_keskipiste_x - pallon_halkaisija / 2,
                               pallon_keskipiste_y - pallon_halkaisija / 2,
                               pallon_halkaisija,
                               pallon_halkaisija ) ) ;

   return taman_pallon_alue.contains( annettu_piste.x,
                                      annettu_piste.y  ) ;
}
```

Metodin **piste_on_pallolla()** avulla voidaan **Pallo**-oliolta kysyä sattuuko annettu piste pallon alueelle. Tämän metodin avulla voidaan siis tarkistaa esimerkiksi että klikattiinko hiirellä juuri tämän pallon päällä.

Pallo.java - 3: Lisää Pallo-luokan metodeita.

Metodin **piirra()** avulla **Pallo**-olio saadaan piirtymään ruudulle. Koska **Pallo**-olio tietää värinsä, halkaisijansa ja paikkansa ruudulla, se osaa itse piirtää itsensä.

Kun pallo on aktivoitu, sille piirretään paksummat reunat mustalla värillä. Pallon aktivoimiseen ja deaktivoimiseen on olemassa valmiit metodit. Tarkoitus on että tämän luokan käyttäjä voi aktivoimalla tehdä pallosta eri näköisen esim. pallon siirtämisen ajaksi.

```
   public void piirra( Graphics graphics )
   {
      graphics.setColor( pallon_vari ) ;

      graphics.fillOval( pallon_keskipiste_x - pallon_halkaisija / 2,
                         pallon_keskipiste_y - pallon_halkaisija / 2,
                         pallon_halkaisija, pallon_halkaisija ) ;

      graphics.setColor( Color.black ) ;

      graphics.drawOval( pallon_keskipiste_x - pallon_halkaisija / 2,
                         pallon_keskipiste_y - pallon_halkaisija / 2,
                         pallon_halkaisija, pallon_halkaisija ) ;

      //  If this ball is activated, it will have a thick black edge

      if ( pallo_on_aktivoitu  ==  true )
      {
         graphics.drawOval( pallon_keskipiste_x - pallon_halkaisija / 2 + 1,
                            pallon_keskipiste_y - pallon_halkaisija / 2 + 1,
                            pallon_halkaisija - 2, pallon_halkaisija - 2 ) ;

         graphics.drawOval( pallon_keskipiste_x - pallon_halkaisija / 2 + 2,
                            pallon_keskipiste_y - pallon_halkaisija / 2 + 2,
                            pallon_halkaisija - 4, pallon_halkaisija - 4 ) ;
      }
   }

}
```

Pallo.java - 4. Pallo-olion piirtometodi.

Tämä hyödynnetään valmista **Pallo**-luokkaa, joka sijaitsee erillisessä ohjelmatiedostossa nimeltä **Pallo.java**. Kääntämisen aikana kyseinen tiedosto tulee sijaita samassa kansiossa tämän ohjelman kanssa.

Kolme **Pallo**-oliota luodaan apletin käynnistyessä. Palloille lasketaan tässä paikat siten että ne sijaitsevat alussa rivissä keskellä apletin aluetta.

```
//  PalloLiikkuuHiirellaApplet.java  (c)  Kari Laitinen

import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;

class PalloLiikkuuHiirellaPanel extends JPanel
                               implements MouseListener,
                                          MouseMotionListener
{
   int  apletin_leveys, apletin_korkeus ;

   Pallo  eka_pallo, toka_pallo, kolmas_pallo  ;

   Pallo  liikuteltava_pallo  =  null ;

   Point  edellinen_hiiren_paikka ;

   public PalloLiikkuuHiirellaPanel( int annettu_apletin_leveys,
                                     int annettu_apletin_korkeus )
   {
      apletin_leveys   =  annettu_apletin_leveys ;
      apletin_korkeus  =  annettu_apletin_korkeus ;


      eka_pallo  =  new  Pallo( apletin_leveys / 2  -  150,
                                apletin_korkeus / 2,
                                Color.red ) ;

      toka_pallo  =  new  Pallo( apletin_leveys / 2,
                                 apletin_korkeus / 2,
                                 Color.green ) ;

      kolmas_pallo  =  new  Pallo( apletin_leveys / 2  + 150,
                                   apletin_korkeus / 2,
                                   Color.blue ) ;

      addMouseListener(this);
      addMouseMotionListener(this);
   }

   public void mouseMoved( MouseEvent event ) {}
   public void mouseClicked( MouseEvent event ) {}
   public void mouseEntered( MouseEvent event ) {}
   public void mouseExited( MouseEvent event ) {}
```

PalloLiikkuuHiirellaApplet.java - 1: Pallo-luokkaa hyödyntävä apletti.

Jokaiselta **Pallo**-oliolta tiedustellaan tässä metodin **piste_on_pallolla()** avulla että kuuluuko klikkauspiste kyseisen pallon sisään. **liikuteltava_pallo** asetetaan viittaamaan siihen palloon jonka päälle hiiren klikkaus osui.

Liikuteltava pallo on liikuttelun ajan aktivoituna minkä vuoksi se piirtyy hiukan erinäköisenä. Kun pallon liikuttelu lopetetaan, pallo deaktivoidaan.

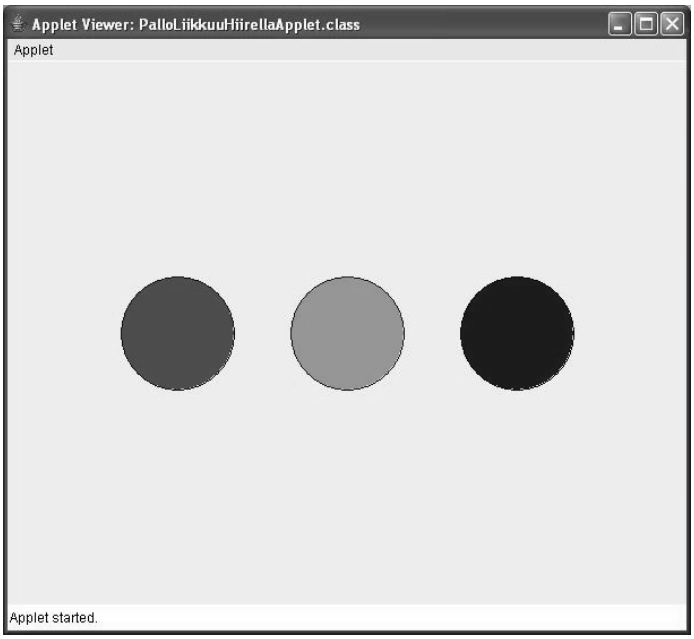
```
public void mousePressed( MouseEvent event )
{
   Point klikkauspiste  =  event.getPoint() ;

   if ( eka_pallo.piste_on_pallolla( klikkauspiste )  )
   {
      liikuteltava_pallo  =  eka_pallo ;
      liikuteltava_pallo.aktivoi_pallo() ;
   }
   else if ( toka_pallo.piste_on_pallolla( klikkauspiste )  )
   {
      liikuteltava_pallo  =  toka_pallo ;
      liikuteltava_pallo.aktivoi_pallo() ;
   }
   else if ( kolmas_pallo.piste_on_pallolla( klikkauspiste )  )
   {
      liikuteltava_pallo  =  kolmas_pallo ;
      liikuteltava_pallo.aktivoi_pallo() ;
   }

   edellinen_hiiren_paikka  =  klikkauspiste ;
   repaint() ;
}

public void mouseReleased( MouseEvent event )
{
   liikuteltava_pallo.deaktivoi_pallo() ;
   liikuteltava_pallo  =  null ;
   repaint() ;
}

public void mouseDragged( MouseEvent event )
{
   if ( liikuteltava_pallo  !=  null )
   {
      Point uusi_hiiren_paikka  =  event.getPoint() ;

      int  tarvittava_siirtyma_x  =  uusi_hiiren_paikka.x
                                     -  edellinen_hiiren_paikka.x ;
      int  tarvittava_siirtyma_y  =  uusi_hiiren_paikka.y
                                     -  edellinen_hiiren_paikka.y ;
      edellinen_hiiren_paikka  =  uusi_hiiren_paikka ;

      liikuteltava_pallo.siirra_palloa( tarvittava_siirtyma_x,
                                        tarvittava_siirtyma_y ) ;
      repaint() ;
   }
}
```

PalloLiikkuuHiirellaApplet.java - 2: Apletin hiireen reagoivat metodit.

Luokan **Pallo** piirtometodia kutsumalla komennetaan jokainen **Pallo**-olio piirtämään itse itsensä.

```
   public void paintComponent( Graphics graphics )
   {
      super.paintComponent( graphics ) ;

      eka_pallo.piirra( graphics ) ;
      toka_pallo.piirra( graphics ) ;
      kolmas_pallo.piirra( graphics ) ;
   }
}


public class PalloLiikkuuHiirellaApplet extends JApplet
{
   public void init()
   {
      getContentPane().add( new PalloLiikkuuHiirellaPanel(
                                 getSize().width,  getSize().height ) ) ;
   }
}
```

PalloLiikkuuHiirellaApplet.java - 3. paintComponent()-metodi ja lyhyt aplettiluokka.



PalloLiikkuuHiirellaApplet.java - X. Apletti heti käynnistymisen jälkeen.

```
// PelikortitApplet.java (c) Kari Laitinen

import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;
import java.util.ArrayList ;
import java.util.Collections ;

class Kortti
{
   int kortin_arvo ;
   int kortin_maa ;

   //  oikeinpain  =  kortti piirretaan tietopuoli nakyviin
   //  nurinpain   =  kortti piirretaan selkapuoli nakyviin

   boolean  tama_kortti_on_oikeinpain  =  false ;
   Point  kortin_paikka_pikseleina  =  null ;
   static final Font fontti_kortin_teksteihin
                           = new Font( "SansSerif", Font.BOLD, 20 ) ;

   public static final int  HERTTA  =  1 ;
   public static final int  RUUTU   =  2 ;
   public static final int  PATA    =  3 ;
   public static final int  RISTI   =  4 ;

   public static final int  KORTIN_LEVEYS_PIKSELEINA   =  128 ;
   public static final int  KORTIN_KORKEUS_PIKSELEINA  =  176 ;

   public Kortti ( int annettu_kortin_arvo,
                   int annettu_kortin_maa )
   {
      kortin_arvo  = annettu_kortin_arvo ;
      kortin_maa   = annettu_kortin_maa ;
   }

   public int lue_arvo()
   {
      return kortin_arvo  ;
   }

   public int lue_maa()
   {
      return kortin_maa ;
   }

   public void  kaanna_kortti()
   {
      if ( tama_kortti_on_oikeinpain  ==  true )
      {
         tama_kortti_on_oikeinpain  =  false ;
      }
      else
      {
         tama_kortti_on_oikeinpain  =  true ;
      }
   }
```

Mahdolliset maat määritellään näillä staattisilla jäsenillä. Näitä voi muista luokista käyttää kun kirjoittaa Kortti.HERTTA, Kortti.RUUTU, jne.

Luokan konstruktorilla ei määritellä kortin paikkaa ruudulla. Kortin paikan asettamiseen on oma metodi nimeltä **aseta_paikka_pikseleina()**. Kortilla ei tarvitse olla paikkaa jos se on vielä esim. pakassa oleva kortti.

PelikortitApplet.java - 1: Pelikorttien näyttämiseen liittyvä esimerkkiohjelma.

Nämä metodit kääntävät korttia loogisessa mielessä. Riippuen siitä mikä on tässä asetettavan **boolean**-muuttujan arvo, **piirra()**-metodi piirtää kortin joko oikeinpäin tai nurinpäin.

```
public void  kaanna_kortti_oikeinpain()
{
   tama_kortti_on_oikeinpain  =  true ;
}

public void  kaanna_kortti_nurinpain()
{
   tama_kortti_on_oikeinpain  =  false ;
}

public boolean  kortti_on_oikeinpain()
{
   return  (  tama_kortti_on_oikeinpain  ==  true  ) ;
}

public boolean  kortti_on_nurinpain()
{
   return  (  tama_kortti_on_oikeinpain  ==  false  ) ;
}

public void  aseta_paikka_pikseleina(
                   Point  annettu_kortin_paikka_pikseleina )
{
   kortin_paikka_pikseleina  =  annettu_kortin_paikka_pikseleina ;
}

public String lue_maa_stringina()
{
   String  palautettava_stringi  =  "" ;

   switch( kortin_maa )
   {
   case  HERTTA :
      palautettava_stringi  =  "Hertta" ;
      break ;
   case  RUUTU :
      palautettava_stringi  =  "Ruutu" ;
      break ;
   case  PATA :
      palautettava_stringi  =  "Pata" ;
      break ;
   case  RISTI :
      palautettava_stringi  =  "Risti" ;
      break ;
   default:
      palautettava_stringi  =  "Ohjelmavirhe!!!" ;
   }

   return  palautettava_stringi ;
}
```

Kortti-luokan sisällä maa on kuvattu numeerisilla vakioilla. Tämä metodi palauttaa vakiota vastaavan merkkijonon.

PelikortitApplet.java - 2: Kortti-luokan metodeja.

Kortti-luokka sisältää joukon metodeja joilla voidaan verrata "tätä" korttia parametrina annettuun **Kortti**-olioon.

```
public boolean on_samaa_maata_kuin( Kortti toinen_kortti )
{
   return ( kortin_maa  ==  toinen_kortti.kortin_maa ) ;
}

public boolean on_eri_maata_kuin( Kortti toinen_kortti )
{
   return ( kortin_maa  !=  toinen_kortti.kortin_maa ) ;
}

public boolean on_pienempi_kuin( Kortti toinen_kortti )
{
   return ( kortin_arvo  <  toinen_kortti.kortin_arvo ) ;
}

public boolean on_suurempi_kuin( Kortti toinen_kortti )
{
   return ( kortin_arvo  >  toinen_kortti.kortin_arvo ) ;
}

public boolean on_yhtasuuri_kuin( Kortti toinen_kortti )
{
   return ( kortin_arvo  ==  toinen_kortti.kortin_arvo ) ;
}

public boolean on_erisuuri_kuin( Kortti toinen_kortti )
{
   return ( kortin_arvo  !=  toinen_kortti.kortin_arvo ) ;
}

public boolean piste_on_talla_kortilla( Point annettu_piste )
{
   return ( kortin_paikka_pikseleina  !=  null  &&
      annettu_piste.x  >=  kortin_paikka_pikseleina.x  &&
      annettu_piste.x  <=  kortin_paikka_pikseleina.x  +
                                KORTIN_LEVEYS_PIKSELEINA  &&
      annettu_piste.y  >=  kortin_paikka_pikseleina.y  &&
      annettu_piste.y  <=  kortin_paikka_pikseleina.y  +
                                KORTIN_KORKEUS_PIKSELEINA ) ;
}
```

Tämä metodi palauttaa joko arvon **true** tai **false** riippuen siitä, osuiko annettu **Point**-tyyppinen piste sille ruudun alueelle jolla kortti fyysisesti ruudulla sijaitsee. Tällä metodilla voidaan mm. tutkia osuiko klikkauspiste kortin alueelle.

PelikortitApplet.java - 3: Kortti-luokan vertailumetodeja.

```
public void piirra( Graphics  graphics )
{
   Font alkuperainen_fontti = graphics.getFont() ;
   graphics.setFont( fontti_kortin_teksteihin ) ;

   if ( tama_kortti_on_oikeinpain   ==  true  )
   {
      graphics.setColor( Color.YELLOW ) ;
   }
   else
   {
      graphics.setColor( Color.LIGHT_GRAY ) ;
   }

   graphics.fillRect( kortin_paikka_pikseleina.x,
                      kortin_paikka_pikseleina.y,
                      KORTIN_LEVEYS_PIKSELEINA,
                      KORTIN_KORKEUS_PIKSELEINA ) ;

   graphics.setColor( Color.black ) ;
   graphics.drawRect( kortin_paikka_pikseleina.x,
                      kortin_paikka_pikseleina.y,
                      KORTIN_LEVEYS_PIKSELEINA,
                      KORTIN_KORKEUS_PIKSELEINA ) ;

   if ( tama_kortti_on_oikeinpain  ==  true )
   {
      graphics.drawString( lue_maa_stringina(),
                           kortin_paikka_pikseleina.x  + 20,
                           kortin_paikka_pikseleina.y  +
                            ( ( KORTIN_KORKEUS_PIKSELEINA * 2 ) / 3 ) ) ;

      String  kortin_arvo_stringi  =  ""  +  kortin_arvo ;

      if ( kortin_arvo  ==  1  ||  kortin_arvo  ==  14 )
      {
         kortin_arvo_stringi  =  "1 Ace" ;
      }
      else if ( kortin_arvo  ==  13 )
      {
         kortin_arvo_stringi  =  "13 King" ;
      }
      else if ( kortin_arvo  ==  12 )
      {
         kortin_arvo_stringi  =  "12 Queen" ;
      }
      else if ( kortin_arvo  ==  11 )
      {
         kortin_arvo_stringi  =  "11 Jack" ;
      }

      graphics.drawString( kortin_arvo_stringi,
                           kortin_paikka_pikseleina.x  + 20,
                           kortin_paikka_pikseleina.y  +
                             ( KORTIN_KORKEUS_PIKSELEINA  / 3  ) ) ;
   }
```

Silloin kun kortti on nurinpäin, eli sen maa ja arvo eivät ole näkyvissä, se piirretään vaalean harmaana.

Ässän ja kuvakorttien tapauksessa kortin arvo ilmaistaan tekstuaalisemmin. Pikkukortit näytetään vain pelkkänä numerona.

PelikortitApplet.java - 4: Kortti-luokan piirra()-metodi.

```
      graphics.setFont( alkuperainen_fontti ) ;

   }
} // Kortti-luokan loppu


class Korttipakka
{
   ArrayList<Kortti>  pakan_kortit   =  new ArrayList<Kortti>() ;

   int[]  maat  =  { Kortti.HERTTA, Kortti.RUUTU, Kortti.PATA, Kortti.RISTI } ;


   public void sekoita_pakka()
   {
      //  ArrayList-pohjainen dynaaminen taulukko voidaan
      //  pistää satunnaiseen järjestykseen Collections-luokan
      //  staattisella shuffle()-metodilla.

      Collections.shuffle( pakan_kortit ) ;
   }

   public void lisaa_kortti( Kortti uusi_kortti )
   {
      if ( pakan_kortit.size()  <  52 )
      {
         pakan_kortit.add( uusi_kortti ) ;
      }
   }

   public Kortti ota_kortti()
   {
      Kortti  annettava_kortti  =  null ;

      if ( pakan_kortit.size()  >  0 )
      {
         annettava_kortti  =  pakan_kortit.get( pakan_kortit.size() - 1 ) ;

         pakan_kortit.remove( pakan_kortit.size() - 1 ) ;
      }

      return  annettava_kortti  ;
   }
}
```

`Korttipakka`-luokka on tarkoitettu kantaluokaksi siitä johdetuille luokille.

Metodi **`ota_kortti()`** palauttaa viittauksen **`Kortti`**-olioon ja poistaa kyseisen kortin pakasta. Koska tässä otetaan viimeinen taulukossa oleva kortti, voidaan ajatella että kortti jaetaan "pakan alta". Metodi palauttaa **`null`**-viittauksen jos pakassa ei ole kortteja.

PelikortitApplet.java - 5: Luokka nimeltä Korttipakka.

```
//  Kuningaspakka on korttipakka jossa ässä on pienin kortti
//  ja kuningas on suurin kortti. Tätä taidetaan loppujen lopuksi
//  tarvita aika harvassa korttipelissä.

class Kuningaspakka  extends  Korttipakka
{
   public Kuningaspakka()  // Konstruktori eli muodostin.
   {
      for ( int maan_indeksi  =  0 ;
                maan_indeksi  <  4 ;
                maan_indeksi  ++ )
      {
         for ( int kortin_arvo  =  1 ;
                   kortin_arvo  <  14 ;
                   kortin_arvo  ++ )
         {
            lisaa_kortti( new  Kortti( kortin_arvo, maat[ maan_indeksi ] ) ) ;
         }
      }
   }
}


class Pokeripakka  extends  Korttipakka
{
   public Pokeripakka()
   {
      for ( int maan_indeksi  =  0 ;
                maan_indeksi  <  4 ;
                maan_indeksi  ++ )
      {
         for ( int kortin_arvo  =  2 ;
                   kortin_arvo  <  15 ;
                   kortin_arvo  ++ )
         {
            lisaa_kortti( new  Kortti( kortin_arvo, maat[ maan_indeksi ] ) ) ;
         }
      }
   }
}
```

Näiden silmukoiden avulla luodaan 52 kappaletta **Kortti**-olioita ja lisätään nämä kortit "tähän" korttipakkaan.

Kuningaspakka-luokka ja **Pokeripakka**-luokka on tässä johdettu eli periytetty luokasta **Korttipakka**. Luokka **Pokeripakka** on kuten **Kuningaspakka**, mutta siina ässän numeroarvo on 14. Tuo kortti piirtyy luokan **Kortti** metodilla **piirra()** aivan kuten arvon 1 sisältävä ässä, mutta **Kortti**-luokan vertailumetodit ymmartavat numeroarvon 14 sisältävän ässän suurimmaksi.

PelikortitApplet.java - 6: Luokkien Kuningaspakka ja Pokeripakka määrittely.

Tässä **korttipakka** on tyypin **Korttipakka** olioviittaaja, ja se asetetaan viittaamaan tyypin **Pokeripakka** olioon. Tämä on täysin sallittua ja normaalia koska **Pokeripakka** on luokan **Korttipakka** alaluokka. Olioviittaaja voi viitata itsensä tyyppisiin olioihin sekä olioihin jotka ovat jonkun olioviittaajan tyypin alaluokan olioita.

```
class PelikortitPanel extends JPanel
                      implements ActionListener, MouseListener
{
   JButton  jaa_kortit_button     =  new  JButton( "Jaa kortit" ) ;
   JButton  sekoita_pakka_button  =  new  JButton( "Sekoita pakka" ) ;

   //  Pokeripakka-oliossa assa on suurin.

   Korttipakka  korttipakka  =  new  Pokeripakka() ;

   ArrayList<Kortti>  rivi_kortteja  =  new ArrayList<Kortti>() ;

   Kortti  yksinainen_kortti ;

   Kortti  valittu_kortti ;

   //  Tässä käytetään samaa heksadesimaalista taustaväriä kun
   //  on käytetty .html-tiedostossa. Pannaan se apletin taustalle
   //  vähän vaaleampana.

   Color   pelialueen_taustavari = new Color( 0x41C300 ).brighter() ;

   public PelikortitPanel()
   {
      setLayout( new BorderLayout() ) ;

      setBackground( pelialueen_taustavari ) ;

      JPanel alareunan_paneeli  =  new  JPanel() ;

      alareunan_paneeli.setBackground( pelialueen_taustavari ) ;
      jaa_kortit_button.setBackground( pelialueen_taustavari.darker() ) ;
      sekoita_pakka_button.setBackground( pelialueen_taustavari.darker() ) ;

      jaa_kortit_button.addActionListener( this ) ;
      sekoita_pakka_button.addActionListener( this ) ;

      alareunan_paneeli.add( jaa_kortit_button ) ;
      alareunan_paneeli.add( sekoita_pakka_button ) ;

      add( "South", alareunan_paneeli ) ;

      addMouseListener( this ) ;
   }
```

PelikortitApplet.java - 7: PelikortitPanel-luokan alkuosa.

ota_kortti()-metodilla otetaan korttipakasta kortteja ja talletetaan ne taulukkoon johon viittaa **rivi_kortteja**. Korttipakassa oleville korteille ei ole määritelty paikkaa ruudulle, joten kun ne otetaan pakasta pois niille määritellään paikka johon ne piirretään **piirra()**-metodin avulla.

```
public void actionPerformed( ActionEvent event )
{
   if ( event.getSource() instanceof JButton )
   {
      if ( event.getSource()  ==  jaa_kortit_button )
      {
         //  Tyhjennetää ArrayList-taulukko ensin vanhoista
         //  korteista.

         rivi_kortteja.clear() ;

         for ( int kortin_indeksi  =  0 ;
                   kortin_indeksi  <  5 ;
                   kortin_indeksi  ++ )
         {
            Kortti uusi_kortti_riviin  =  korttipakka.ota_kortti() ;

            Point  kortin_paikka  =  new  Point(
               40 + ( Kortti.KORTIN_LEVEYS_PIKSELEINA +20 ) * kortin_indeksi,
                50 ) ;

            uusi_kortti_riviin.aseta_paikka_pikseleina( kortin_paikka ) ;

            rivi_kortteja.add( uusi_kortti_riviin ) ;
         }

         yksinainen_kortti  =  korttipakka.ota_kortti() ;
         yksinainen_kortti.aseta_paikka_pikseleina( new  Point( 188, 300 ) );
      }
      else if ( event.getSource()  ==  sekoita_pakka_button )
      {
         korttipakka.sekoita_pakka() ;
      }

      repaint() ;
   }
}
```

sekoita_pakka()-metodi sekoittaa korttipakassa kulloinkin jäljellä olevat kortit.

PelikortitApplet.java - 8: Painonappien painamisiin regoiva metodi.

Klikkauspaikka saadaan selville tutkimalla tälle metodille parametrina annettavaa **MouseEvent**-tyypin oliota.

Kortti-luokkaan kuuluvalla **piste_on_talla_kortilla()**-metodin avulla tutkitaan jokainen ruudulla näytettävä kortti, ja otetaan selville minkä kortin päälle klikkaus sattui. On mahdollista että klikkaus ei osu minkään kortin päälle.

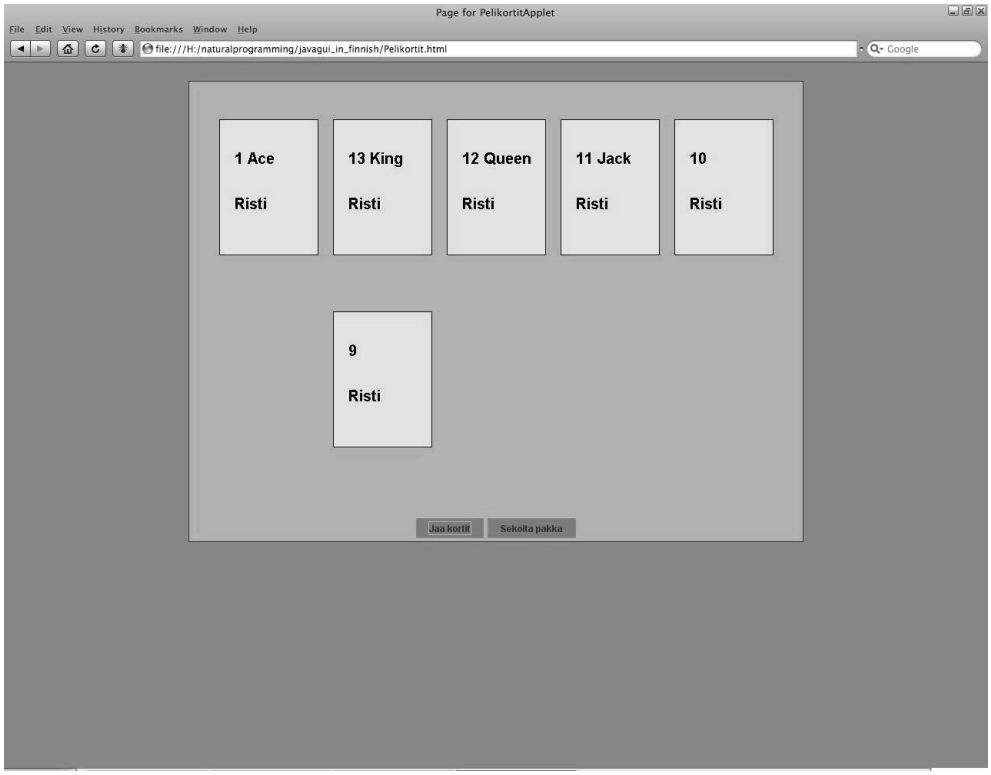
```
public void mouseClicked( MouseEvent event )
{
   Point  klikkauspaikka  =  new Point( event.getX(), event.getY() );

   if ( rivi_kortteja.size() == 5 )
   {
      //  Viisi korttia on rivissä. Tarkastetaan osuiko klikkaus
      //  millekään noista korteista.

      for ( int kortin_indeksi  =  0 ;
                kortin_indeksi  <  rivi_kortteja.size() ;
                kortin_indeksi  ++ )
      {
         if ( rivi_kortteja.get( kortin_indeksi ).
                              piste_on_talla_kortilla( klikkauspaikka ) )
         {
            rivi_kortteja.get( kortin_indeksi ).kaanna_kortti() ;

            //  valittu_kortti asetetaan viittaamaan klikattuun
            //  korttiin rivi_kortteja -taulukossa, mutta
            //  valitulla kortilla ei muuten tehda mitaan tassa
            //  ohjelmassa.

            valittu_kortti  =  rivi_kortteja.get( kortin_indeksi ) ;
         }
      }

      if ( yksinainen_kortti != null  &&
           yksinainen_kortti.piste_on_talla_kortilla( klikkauspaikka ) )
      {
         yksinainen_kortti.kaanna_kortti() ;
      }

      repaint() ;
   }
}

public void mouseEntered( MouseEvent event ) {}
public void mouseExited( MouseEvent event ) {}
public void mousePressed( MouseEvent event ) {}
public void mouseReleased( MouseEvent event ) {}
```

PelikortitApplet.java - 9: Hiiren klikkaukseen reagoiva metodi.

Jos taulukkoon johon **rivi_kortteja** viittaa on jo pantu kortit, piirretään ne yksitellen ruudulle.

```
   public void paint( Graphics graphics )
   {
      super.paint( graphics ) ;

      //  Piirretään apletin alueelle mustalla raamit.
      graphics.drawRect( 0, 0, getWidth() - 1, getHeight() - 1 ) ;

      //  Jos rivi_kortteja-taulukkoon on jo pantu kortit,
      //  piirretaan ne yksitellen.

      if ( rivi_kortteja.size() > 0 )
      {
         // Käytetään Kortti-olioiden piirtämisessä ns. "foreach"
         // silmukkaa joka on erikoisversio perinteisestä for-silmukasta.
         // "foreach"-silmukka käy aina lävitse annetun taulukon kaikki
         // Kortti-oliot.

         for ( Kortti kortti_rivissa : rivi_kortteja )
         {
            kortti_rivissa.piirra( graphics ) ;
         }

         // Huomaa että yksinäinen kortti piirretään vain jos löytyi
         // rivistä piirrettäviä kortteja.

         if ( yksinainen_kortti != null )
         {
            yksinainen_kortti.piirra( graphics ) ;
         }
      }
      else
      {
         graphics.drawString( "KORTTEJA EI OLE JAETTU", 100, 100 ) ;
      }
   }
}

public class PelikortitApplet extends JApplet
{
   public void init()
   {
      getContentPane().add( new PelikortitPanel() ) ;
   }
}
```

PelikortitApplet.java - 10. PelikortitPanel-luokan loppu ja lyhyt aplettiluokka.



Tässä Jaa kortit -nappia on painettu ilman että pakkaa on vielä sekoitettu. Apletti on jakanut pakasta kuusi korttia, joita on klikattu sillä seurauksella että ne ovat kääntyneet oikein-päin.

PelikortitApplet.java - X. Kuusi alinta pakan korttia jaettuna.

Lisää JPanel-luokan avulla rakennettuja apletteja

Javan luokkakirjastoihin kuuluu ns. Swing-luokkakirjasto joka täydentää perinteistä AWT (Abstract Windowing Toolkit) -kirjastoa. Swing-luokat tunnistaa niiden nimen alkukirjaimesta joka on J. Swing-luokkien avulla voidaan rakentaa sekä apletteja että Windows-sovelluksia (aplikaatioita). Swing-luokilla toteutetuissa ohjelmissa on seuraavia ominaisuuksia:

- Swing-luokan JPanel avulla rakennettu apletti voidaan vähällä vaivalla muutaa Windows-sovellukseksi ja päinvastoin.
- JPanel-luokkaan on sisäänrakennettuna ns. tuplapuskurointi, minkä vuoksi niillä rakennetut sovellukset eivät välkytä ruutua.
- Swing-luokat toteuttavat painonapit, valikot yms. Windows-komponentit itse, joten ruudulla näkyvät komponentit ovat aina saman näköisiä riippumatta ajetaanko esim. aplettia Windows-koneessa, UNIX-koneessa tai esim. Macintosh-tietokoneessa. (Perinteisillä AWT-luokilla toteutetuissa apleteissa painonapit ja muut ikkunakomponentit näyttävät siltä kuin ne kohdejäjelestelmässä (esim. Windows) yleensä näyttävät.)

Seuraavilla sivuilla esitellään lisää JPanel-luokan avulla rakennettuja Java-ohjelmia. Osa apleteista on muunnoksia aiemmin esitetyistä ohjelmista.

Näillä merkinnöillä määritellään että tämä paneeli toteuttaa rajapinnat **ActionListener** ja **ItemListener**. Käytännössä tämä tarkoittaa että aplettiin kuuluu metodit **actionPerformed()** ja **itemStateChanged()**, joita kutsutaan automaattisesti silloin kun paneelissa olevia nappuloita painetaan, tai kun paneelissa olevan valintamenun valintaa muutetaan.

```
//  MovingBallPanelApplet.java

import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;

class MovingBallPanel extends JPanel
                      implements ActionListener, ItemListener
{
   JButton  left_button, up_button, down_button, right_button ;

   int  applet_width, applet_height ;

   int  ball_center_point_x, ball_center_point_y ;

   Color  current_ball_color   =  Color.red ;

   public MovingBallPanel( int given_applet_width,
                           int given_applet_height )
   {
      applet_width  =  given_applet_width ;
      applet_height =  given_applet_height ;

      ball_center_point_x  =  applet_width / 2 ;
      ball_center_point_y  =  applet_height / 2 ;

      setLayout( new BorderLayout() ) ;

      JPanel operations_panel  =  new  JPanel() ;

      left_button     =  new JButton( " < " ) ;
      up_button       =  new JButton( " Up " ) ;
      down_button     =  new JButton( " Down " ) ;
      right_button    =  new JButton( " > " ) ;

      left_button.addActionListener( this ) ;
      up_button.addActionListener( this ) ;
      down_button.addActionListener( this ) ;
      right_button.addActionListener( this ) ;

      operations_panel.add( left_button ) ;
      operations_panel.add( up_button ) ;
      operations_panel.add( down_button ) ;
      operations_panel.add( right_button ) ;
```

Näillä metodin **addActionListener()** kutsuilla määritetään että tämä paneeli on määriteltyjen nappuloiden painalluksia "kuunteleva" olio. Tämä tarkoittaa käytännössä että tässä luokassa olevaa **actionPerformed()** -metodia kutsutaan kun nappuloita painellaan.

Tässä aiemmin luotuun **JPanel**-luokan olioon kiinnitetään määriteltyjä nappulaolioita. Seuraavalla sivulla paneeliolioon kiinnitetään vielä valintamenuolio.

MovingBallPanelApplet.java - 1: Apletti joka näyttää napeilla liikuteltavaa palloa.

Ja tässä on loppuosa **MovingBallPanel**-luokan konstruktoria. Tällä operaatiolla määrätään että tämä apletti käsittelee valintamenussa tapahtuvat valinnat. Käytännössä tämän luokan **itemStateChanged()**-metodia kutsutaan silloin kun värivalintamenussa muutetaan värin valintaa.

```
   JComboBox  color_selection_menu  =  new JComboBox() ;

   color_selection_menu.addItem( "red" ) ;
   color_selection_menu.addItem( "orange" ) ;
   color_selection_menu.addItem( "yellow" ) ;
   color_selection_menu.addItem( "green" ) ;
   color_selection_menu.addItem( "blue" ) ;
   color_selection_menu.addItem( "magenta" ) ;
   color_selection_menu.addItem( "cyan" ) ;
   color_selection_menu.addItem( "pink" ) ;
   color_selection_menu.addItem( "lightGray" ) ;

   color_selection_menu.addItemListener( this ) ;

   operations_panel.add( color_selection_menu ) ;

   add( "South", operations_panel ) ;
}

public void actionPerformed( ActionEvent event )
{
   if ( event.getSource() instanceof JButton )
   {
      if ( event.getSource()  ==  left_button )
      {
         ball_center_point_x  -=  3 ;
      }
      else if ( event.getSource()  ==  up_button )
      {
         ball_center_point_y  -=  3 ;
      }
      else if ( event.getSource()  ==  down_button )
      {
         ball_center_point_y  +=  3 ;
      }
      else if ( event.getSource()  ==  right_button )
      {
         ball_center_point_x  +=  3 ;
      }

      repaint() ;
   }
}
```

Tämän konstruktorin alussa määriteltiin että tässä paneelissa on käytössä ns. **BorderLayout**, mikä tarkoittaa että erilaiset paneeliin lisättävät komponentit voidaan kiinnittää eri kohtiin ilmansuuntia merkitsevien sanojen avulla. Tässä operaatiopaneeli kiinnitetään alareunaan, kun **add()**-metodille annetaan parametrina "South". "North" kiinnittäisi paneelin yläreunaan. Lisäksi voidaan käyttää sanoja "West", "East" ja "Center".

Tätä metodia kutsutaan silloin kun jotain ruudulla olevista nappuloista painetaan. Metodille tulee parametrina **ActionEvent**-tyypin olio, jonka avulla saadaan tietää mikä nappulaolio aiheutti metodin kutsumisen.

MovingBallPanelApplet.java - 2: Konstruktorin loppuosa ja actionPerformed()-metodi.

Tätä metodia kutsutaan kun värivalinta-menussa suoritetaan värin muutos. Tällä tavalla valittu menun item muutetaan `String`-olioksi, jonka sisältöä voidaan sitten tutkia.

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

   if ( selected_color.equals( "red" ) )
   {
      current_ball_color  =  Color.red ;
   }
   else if ( selected_color.equals( "orange" ) )
   {
      current_ball_color  =  Color.orange ;
   }
   else if ( selected_color.equals( "yellow" ) )
   {
      current_ball_color  =  Color.yellow ;
   }
   else if ( selected_color.equals( "green" ) )
   {
      current_ball_color  =  Color.green ;
   }
   else if ( selected_color.equals( "blue" ) )
   {
      current_ball_color  =  Color.blue ;
   }
   else if ( selected_color.equals( "magenta" ) )
   {
      current_ball_color  =  Color.magenta ;
   }
   else if ( selected_color.equals( "cyan" ) )
   {
      current_ball_color  =  Color.cyan ;
   }
   else if ( selected_color.equals( "pink" ) )
   {
      current_ball_color  =  Color.pink ;
   }
   else if ( selected_color.equals( "lightGray" ) )
   {
      current_ball_color  =  Color.lightGray ;
   }

   repaint() ;
}
```

Kun Javan `String`-olioita vertaillaan käytetään metodia `equals()`, joka palauttaa arvon `true` silloin kun `String`-oliot sisältävät saman tekstin.

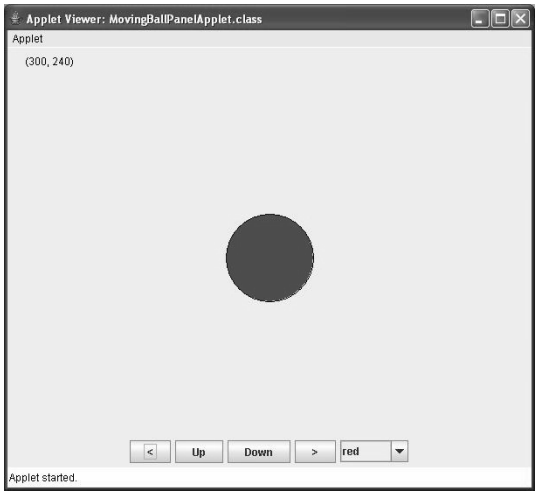
Kutsumalla `repaint()`-metodia apletti, tai tarkemmin sanoen tämä paneeliolio, pyytää ruudun päivitystä. `repaint()`-kutsun jäleen tämän luokan `paintComponent()`-metodia kutsutaan automaattisesti ja pallo piirtyy valitulla uudella värillä.

MovingBallPanelApplet.java - 3: itemStateChanged()-metodi.

Tässä on **MovingBallPanel**-luokan metodi **paintComponent()**. Aluksi asetetaan käyttöön nykyinen pallon väri, ja sen jälkeen piirretään pallo **fillOval()**-metodin avulla käyttäen pallon senhetkisiä koordinaatteja. Pallon halkaisija on 80 pikseliä. Pallon ympärille piirretään vielä viiva mustalla värillä, ja pallon nykyiset koordinaatit pannaan näkyville piirtoalueen vasempaan yläkulmaan.

```
   public void paintComponent( Graphics graphics )
   {
      super.paintComponent( graphics ) ;

      graphics.setColor( current_ball_color ) ;

      graphics.fillOval( ball_center_point_x - 50, ball_center_point_y - 50,
                         100, 100 ) ;

      graphics.setColor( Color.black ) ;

      graphics.drawString( "("   +  ball_center_point_x  +
                           ", "  +  ball_center_point_y  +
                           ")",  20, 20 ) ;

      graphics.drawOval( ball_center_point_x - 50, ball_center_point_y - 50,
                         100, 100 ) ;
   }
}

public class MovingBallPanelApplet extends JApplet
{
   public void init()
   {
      getContentPane().add( new MovingBallPanel( getSize().width,
                                                 getSize().height ) ) ;
   }
}
```

MovingBallPanelApplet.java - 4. Paneeliluokan loppu ja lyhyt aplettiluokka.



Tämä apletti toimii samoin kuin aiemmin nähty **MovingBallApplet.java**. (Palvelimelta löytyy muuten sellainenkin versio jonka nimi on **MovingBallAWTApplet.java**. Tuo on pelkillä AWT-luokilla tehty ’vanhan tyylin’ apletti.)

MovingBallPanelApplet.java - X. Palloa ei ole tässä vielä liikutettu.

Tämä apletti on toteutettu siten että kaikkea ruudulla näkyvää edustaa **MatoPanel**-luokan olio. Itse aplettiluokka on hyvin lyhyt luokka ohjelman lopussa. Tämä panel toteuttaa **Runnable**-rajapinnan, mikä tarkoittaa että siinä on **run()**-metodi, joka toimii itsenäisenä animaatiota tahdistavana säikeenä.

```
//  MatoPanelApplet.java

import java.awt.* ;
import java.util.* ;
import javax.swing.* ;

class MatoPanel extends JPanel implements Runnable
{
   Thread  tahdistussaie ;
   boolean  saietta_on_suoritettava  =  false ;

   Color   kaytossa_oleva_vari      =  Color.red ;
   int  madon_paikka_x  ;
   int  madon_paikka_y  =  150 ;

   int  madon_pituus    =  100 ;
   int  madon_korkeus   =   30 ;

   int  apletin_leveys, apletin_korkeus ;

   public MatoPanel( int annettu_apletin_leveys,
                     int annettu_apletin_korkeus )
   {
      apletin_leveys   =  annettu_apletin_leveys ;
      apletin_korkeus  =  annettu_apletin_korkeus ;
   }

   public void kaynnista_saie()
   {
      if ( tahdistussaie  ==  null )
      {
         tahdistussaie  =  new  Thread( this ) ;
         saietta_on_suoritettava  =  true ;

         tahdistussaie.start() ;
      }
   }

   public void lopeta_saie()
   {
      if ( tahdistussaie  !=  null )
      {
         tahdistussaie.interrupt() ;  //  keskeytä mahdollisesti nukkuva säie
         saietta_on_suoritettava  =  false ;
         tahdistussaie  =  null ;
      }
   }
```

Säikeen suoriintumista kontrolloidaan muuttujan **saietta_on_suoritettava** avaulla. Tämä muuttuja saa arvon **true** silloin kun halutaan suoriintuvan. **run()**-metodi päättyy, ja säie "kuolee" automaattisesti kun **saietta_on_suoritettava** pannanna arvoo **false**.

MatoPanelApplet.java - 1: Matoa ruudulla liikutteleva apletti.

Tämä apletti piirtää ruudulle eräänlaisen madon, joka liikkuu toistuvasti ruudun vasemmasta reunasta oikeaan reunaan. Madon liikkeiden tahdistamisesta vastaa tämä `run()`-metodi. Ennenkuin mato lähtee liikkeelle se on paikallaan ruudun vasemmassa reunassa 2 sekuntia (2000 millisekuntia).

```
public void run()
{
   while ( saietta_on_suoritettava  ==  true )
   {
      madon_paikka_x  =  50 ;
      madon_pituus    = 100 ;

      repaint() ;

      lepuuta_tata_saietta( 2000 ) ;

      while ( madon_paikka_x  +  madon_pituus  <  apletin_leveys  &&
              saietta_on_suoritettava  ==  true )
      {
         for ( int venytyslaskuri  =  0 ;
                   venytyslaskuri  <  7 ;
                   venytyslaskuri  ++ )
         {
            madon_pituus  =  madon_pituus  + 5 ;

            repaint() ;
            lepuuta_tata_saietta( 50 ) ;
         }

         for ( int supistuslaskuri  =  0 ;
                   supistuslaskuri  <  7 ;
                   supistuslaskuri  ++ )
         {
            madon_pituus    =  madon_pituus    -  5 ;
            madon_paikka_x  =  madon_paikka_x  +  5 ;

            repaint() ;
            lepuuta_tata_saietta( 50 ) ;
         }
      }

      lepuuta_tata_saietta( 4000 ) ;
   }
}
```

Mato saadaan näyttämään liikkuvalta, kun madon pituutta ja sen paikan x-koordinaattia modifioidaan ennen ruudun päivityksen pyytämistä `repaint()`-metodin avulla. Ruutua päivitetään tässä kohden 20 kertaa sekunnissa. Tämä ei kuitenkaan aiheuta ruudun välkkymisongelmaa koska Javan Swing-komponenteissa on käytössä automaattinen ns. tuplapuskurointi joka estää välkkymisen.

MatoPanelApplet.java - 2: Itsenäisenä säikeenä toimiva run()-metodi.

Tämän metodin sisään on "piilotettu" kutsu metodiin **`Thread.sleep()`**. Koska tuon metodin käyttö vaatii **`try-catch`**-rakenteen, on sitä helpompi kutsua yhdestä metodista. Tämän metodin erikoisuutena on että säikeen lepuutusta ei tapahdu silloin kun säie on "kuoletettu".

Apletteja testatessa voidaa **`System.out.print()`**-kutsuilla tulostaa tekstiä mustaan komentorivi-ikkunaan. Tässä testauksessa käytetyt ohjelmarivit on kommentoitu pois ohjelmasta.

```
   void  lepuuta_tata_saietta( int lepuutusaika_millisekunteina )
   {
      //  Ilman seuraavaa if-rakennetta syntyy sellainen ongelma
      //  että säie ei pysähdy nopeasti mm. silloin kun matoa näyttävä
      //  ikkuna minimoidaan. Esim. MatoOlioAppletissa tämä metodi on
      //  kirjoitettu ilman if-rakennetta ja siinä on sellainen ongelma
      //  että jos ikkuna minimoidaan ja maksimoidaan heti perään madon
      //  ollessa ruudun keskivaiheilla, mato alkaa uuden säikeen
      //  käynnistyttyä käyttäytyä hyvin omituisesti. Ilmeisesti tämä
      //  johtuu siitä että sekä uusi tahdistussäie että vanha säie
      //  muuttelevat yhtä aikaa madon pituutta ja madon x-paikkaa.
      //  Vanha säie jatkaa elämäänsä kun saietta_on_suoritettava
      //  pannaan uutta säiettä varten arvoon true.

      if ( saietta_on_suoritettava  ==  true )
      {
         try
         {
            Thread.sleep( lepuutusaika_millisekunteina ) ;
            // System.out.print( " Z" ) ;  //  Rivi testausta varten
         }
         catch ( InterruptedException vangittu_poikkeus )
         {
            // System.out.print( " X" ) ;  //  Rivi testausta varten
         }
      }
   }

   public void paintComponent( Graphics graphics )
   {
      super.paintComponent( graphics ) ;

      graphics.setColor( kaytossa_oleva_vari ) ;

      graphics.fillRoundRect( madon_paikka_x, madon_paikka_y,
                              madon_pituus, madon_korkeus, 20, 20  ) ;
   }
}
```

Mato saadaan aikaiseksi piirtämällä ruudulle kylmistaan pyöristetty suorakaide. Tässä kaksi viimeistä parametria kuvaavat pyöristyneisyyden määrää.

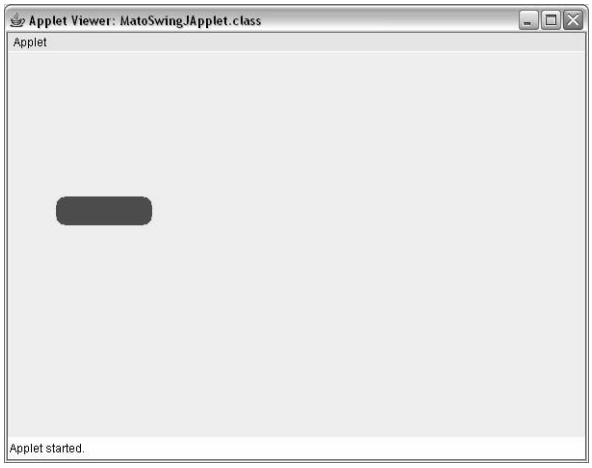
MatoPanelApplet.java - 3: MatoPanel-luokan loppuosa.

MatoPanel-luokan konstruktorille annetaan tässä parametreina apletille varatun ruudun alueen leveys ja korkeus.

```
public class MatoPanelApplet extends JApplet
{
   MatoPanel  paneeli_madolle ;

   public void init()
   {
      paneeli_madolle  =  new MatoPanel( getSize().width, getSize().height ) ;
      getContentPane().add( paneeli_madolle ) ;
   }

   public void start()
   {
      paneeli_madolle.kaynnista_saie() ;
   }

   public void stop()
   {
      paneeli_madolle.lopeta_saie() ;
   }
}
```

Selain tai Applet Viewer kutsuu aplettiluokan **start()**-metodia silloin kun apletti käynnistyy tai kun apletin ikkuna maksimoidaan. **stop()**-metodia kutsutaan puolestaan silloin kun ikkuna minimoidaan. Animaatio pysäytetään tässä siksi ajaksi kun ikkuna on minimoituna.

MatoPanelApplet.java - 5. JApplet-luokasta periytetty aplettiluokka.



MatoPanelApplet.java - X. Apletti on tässä juuri käynnistynyt ja mato ei ole vielä liikkunut.

```
//  SumApplet.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;
import javax.swing.event.* ;
import java.util.Scanner ;

class SumPanel extends JPanel
               implements DocumentListener
{
   JTextField  first_text_field, second_text_field ;
   JLabel  result_label ;

   SumPanel()
   {
      setLayout( null ) ;

      Font font_for_numbers  =  new  Font( "Courier", Font.BOLD, 20 ) ;

      first_text_field  =  new  JTextField( "" ) ;
      first_text_field.setHorizontalAlignment( JTextField.RIGHT ) ;
      first_text_field.setFont( font_for_numbers ) ;
      first_text_field.getDocument().addDocumentListener( this ) ;
      second_text_field  =  new  JTextField( "" ) ;
      second_text_field.setHorizontalAlignment( JTextField.RIGHT ) ;
      second_text_field.setFont( font_for_numbers ) ;
      second_text_field.getDocument().addDocumentListener( this ) ;

      result_label  =  new  JLabel( "0", JLabel.RIGHT ) ;
      result_label.setFont( font_for_numbers ) ;

     JPanel  calculation_panel  =  new JPanel( new GridLayout( 3, 2, 10, 10 ) ) ;

      calculation_panel.setBounds( 100, 50, 300, 150 ) ;

      calculation_panel.add( new JLabel( "First integer:" ) ) ;
      calculation_panel.add( first_text_field ) ;
      calculation_panel.add( new JLabel( "Second integer:" ) ) ;
      calculation_panel.add( second_text_field ) ;
      calculation_panel.add( new JLabel( "Calculated sum:" ) ) ;
      calculation_panel.add( result_label ) ;

      add( calculation_panel ) ;
   }
```

Tämä ohjelma demonstroi mm. `JTextField`- ja `JLabel`-luokkien käyttöä. Sekä tekstikentät että label määritellään tässä sellaisiksi että tekstit ilmestyvät niihin oikealle tasattuina.

`calculation_panel` viittaa `JPanel`-olioon jolle annetaan tässä rajat `setBounds()`-metodilla. Laskentapaneelin paikaksi tulee (100, 50), ja sen leveydeksi tulee 300 pikseliä ja korkeudeksi 150 pikseliä. Koska tämän `SumPanel`-paneelin layoutiksi on annettu `null`, tähän voidaan komponentteja sijoittaa absoluuttisia arvoja käyttäen.

Laskentapaneelissa on käytössä `GridLayout`, minkä ansiosta paneelissa on 3 riviä komponentteja kahdessa sarakkeessa. `JLabel`- ja `JTextField`-oliot lisätään laskentapaneeliin sellaisessa järjestyksessä että "laskukone" näyttää loogiselta ruudulla.

SumApplet.java - 1: Ohjelma joka laskee tekstikenttiin kirjoitettavien lukujen summan.

`calculate_sum()`-metodia kutsutaan aina kun jomman kumman tekstikentän sisältöä muutetaan. Tekstikenttien sisällöt tutkitaan siten että ne muutetaan ensin `Scanner`-olioiksi joista on kätevä skannata dataa. Ensin käytetään `Scanner`-luokan metodia `hasNext()`, jolla voidaan tutkia sisältääkö `Scanner`-olio yhtään tesktiä. Jos `Scanner`-olioissa, eli siis tekstikentissä, ei ole ollenkaan tekstiä, oletetaan niiden sisältävän arvon 0 (nolla). `hasNextInt()`-metodi palauttaa arvon `true` kun tekstikentissä on `int`-arvoksi tulkittava teksti.

```
   void calculate_sum()
   {
      Scanner first_text_to_scan  =  new Scanner( first_text_field.getText() ) ;
      Scanner second_text_to_scan  =  new Scanner( second_text_field.getText());

      //  The following two if constructs ensure that empty text fields
      //  are treated as if they contained a zero.

      if ( ! first_text_to_scan.hasNext() )
      {
         first_text_to_scan  =  new  Scanner( "0" ) ;
      }

      if ( ! second_text_to_scan.hasNext() )
      {
         second_text_to_scan  =  new  Scanner( "0" ) ;
      }

      if ( first_text_to_scan.hasNextInt() &&
           second_text_to_scan.hasNextInt() )
      {
         int first_integer  =  first_text_to_scan.nextInt() ;
         int second_integer  =  second_text_to_scan.nextInt() ;
         int sum_of_two_integers  =  first_integer  +  second_integer ;
         result_label.setText( "" + sum_of_two_integers  ) ;
      }
      else
      {
         result_label.setText( "INPUT ERROR!" ) ;
      }
   }

   public void insertUpdate( DocumentEvent event )
   {
      calculate_sum() ;
   }

   public void removeUpdate( DocumentEvent event )
   {
      calculate_sum() ;
   }

   public void changedUpdate( DocumentEvent event )
   {
   }  //  This kind of update is not possible in the case of JTextField
}
```

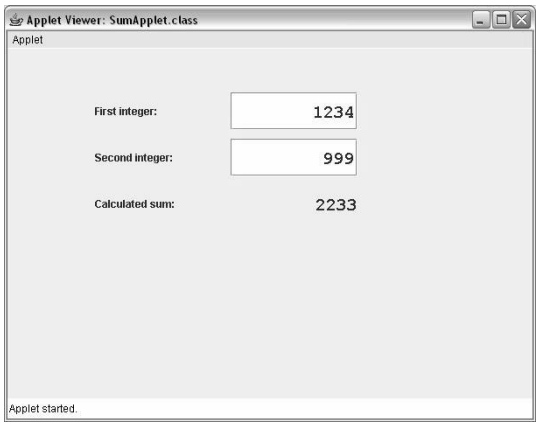
Tämä luokka toteuttaa `DocumentListener`-rajapinnan. Näitä metodeja kutsutaan automaattisesti kun tekstikentissä tehdään muutoksia.

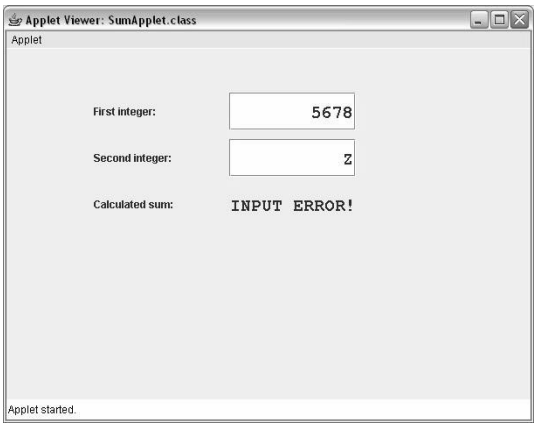
SumApplet.java - 2: SumPanel-luokan loppuosa.

SumPanel-olio tulee tässä **SumApplet**-olion sisälle. **SumPanel**-olion sisällä on puolestaan **JPanel**-tyyppinen "laskentapaneeli", jossa on kiinni **JLabel**- ja **JTextField**-tyyppisiä olioita. "content pane" on puolestaan luokassa **JApplet** määritelty "laatta" johon mm. paneeliolio voidaan kiinnittää.

```
public class SumApplet extends JApplet
{
   public void init()
   {
      getContentPane().add( new SumPanel() ) ;
   }
}
```

SumApplet.java - 3. Lyhyt aplettiluokka jossa luodaan SumPanel-olio.





Tässä toiseen tekstikenttään on syötetty stringi "Z", joka ei edusta mitään kokonaislukua. Tämän vuoksi **JLabel**-olioon johon viittaa **result_label** pannaan teksti "INPUT ERROR!".

SumApplet.java - X. Kaksi kuvaa apletin suoriintumisesta.

Java 2D -grafiikan käyttö

Seuraavilla sivuilla esitellään apletteja joissa on käytetty ns. Java 2D -grafiikkaa. Tämä tarkoittaa käytännössä että piirtämisessä on käytössä luokka `Graphics2D` luokan `Graphics` sijasta.

Javan 2D-grafiikkaa päästään käyttämään kun piirtometodin alkuun lisätään tällainen lause.

Tässä alustetaan taulukot jotka sisältävät nuolen muotoisen kuvion koordinaatit. Nämä koordinaatit syötetään sitten myöhemmin `GeneralPath`-oliolle `moveTo()`- ja `lineTo()`-metodien avulla. Kun `GeneralPath`-olio sitten myöhemmin piirretään, ruudulle piirtyy näiden koordinaattien mukainen kuvio.

```
//  FlyingArrowApplet.java

import java.awt.* ;
import java.awt.geom.* ; //  Classes GeneralPath etc.
import javax.swing.* ;

class FlyingArrowPanel extends JPanel
{
   public void paintComponent( Graphics graphics )
   {
      super.paintComponent( graphics ) ;

      Graphics2D  graphics2D  =  (Graphics2D) graphics ;

      //  The arrow coordinates are selected so that Point (0, 0)
      //  is at the tip of the arrow, and the arrow points upwards.

      int[]  arrow_shape_coordinates_x  =
             { 0,  15,   5,   5,  15,   0, -15,  -5,  -5, -15 } ;
      int[]  arrow_shape_coordinates_y  =
             { 0,  40,  30, 120, 160, 130, 160, 120,  30,  40 } ;

      GeneralPath  flying_arrow  =  new  GeneralPath() ;

      flying_arrow.moveTo( arrow_shape_coordinates_x[ 0 ],
                           arrow_shape_coordinates_y[ 0 ] ) ;

      for ( int point_index  =  1 ;
                point_index  <  arrow_shape_coordinates_x.length ;
                point_index  ++ )
      {
         flying_arrow.lineTo( arrow_shape_coordinates_x[ point_index ],
                              arrow_shape_coordinates_y[ point_index ] ) ;
      }

      flying_arrow.closePath() ;

      graphics2D.translate( 150, 250 ) ;  // arrow tip to point (150, 250 )
      graphics2D.fill( flying_arrow ) ;   // draw solid arrow

      graphics2D.rotate( Math.PI / 4 ) ;  // 45 degrees clockwise
      graphics2D.draw( flying_arrow ) ;   // draw a hollow arrow
```

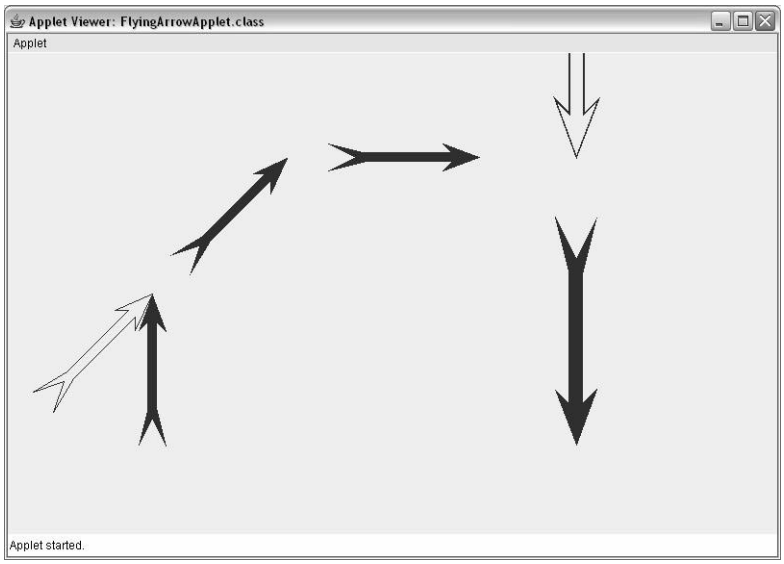
`Graphics2D`-luokan `draw()`-metodilla voidaan piirtää graafisia kuvioita edustavia olioita. `translate()`-metodilla voidaan siirtää piirtopinnan nollakohtaa, ja `rotate()`-metodilla piirtopintaa voidaan "pyörittää".

FlyingArrowApplet.java - 1: Graphics2D-metodien käytön demonstrointia.

Tässä piirtopaikka siirtyy oikealle, vaikka **translate()**-metodille annettavien parametrien mukaan ollaan siirtymässä ylöspäin piirtopinnalla. Tämä johtuu siitä että koordinaatistoa on käännetty kahdessa vaiheessa yhteensä 90 astetta myötäpäivään.

```
        graphics2D.translate( 0, -200 ) ;   // flying "up" 200 points
        graphics2D.fill( flying_arrow ) ;

        graphics2D.rotate( Math.PI / 4 ) ;  // 45 degrees clockwise
        graphics2D.translate( 0, -200 ) ;   // flying "up" (i.e. to the right)
        graphics2D.fill( flying_arrow ) ;

        graphics2D.translate( 0, -100 ) ;   // flying "up" 100 points
        graphics2D.rotate( Math.PI / 2 ) ;  // 90 degrees clockwise
        graphics2D.scale( 1.5, 1.5 ) ;      // magnify everything by 1.5
        graphics2D.draw( flying_arrow ) ;   // draw a hollow arrow
       graphics2D.translate( 0, -200 ) ;   // flying "up" (i.e. down) 300 points
        graphics2D.fill( flying_arrow ) ;
    }
}

public class FlyingArrowApplet extends JApplet
{
   public void init()
   {
      getContentPane().add( new FlyingArrowPanel() ) ;
   }
}
```

FlyingArrowApplet.java - 2. paintComponen()-metodin loppuosa ja aplettiluokka.



Graphics2D-luokan **draw()**-metodia käyttämällä on saatu aikaan ontot nuolet. Metodilla **fill()** puolestaan on piirretty kiinteät nuolet. Nuolen koko on lopussa kasvanut 1.5-kertaiseksi **scale()**-metodin käytön ansiosta.

FlyingArrowApplet.java - X. Sama nuoliolio piirrettynä useaan kertaan.

Ohjelman alussä määritellään **Clock2DPanel**-luokka ja lopussa on varsinainen aplettiluokka jossa luodaan kaksi kappaletta **Clock2DPanel**-olioita. Eri kello-paneelit pannaan näyttämään eri kellonaikaa.

```
//  DoubleClockApplet.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.geom.* ;  //  Classes Line2D, Ellipse2D, etc.
import java.util.* ;      //  Calendar, GregorianCalendar, TimeZone
import javax.swing.* ;

class Clock2DPanel extends JPanel implements Runnable
{
   static final int HOUR_HAND_LENGTH    = 152 ;
   static final int MINUTE_HAND_LENGTH  = 188 ;
   static final int SECOND_HAND_LENGTH  = 200 ;

   Thread   thread_that_runs_the_clock ;

   boolean  thread_must_be_executed ;

   TimeZone clock_time_zone ;

   String  time_zone_text ;

   Font font_for_time_zone_text = new Font( "Serif", Font.BOLD, 20 ) ;

   int  time_zone_text_width ;

   BasicStroke wide_stroke = new BasicStroke( 12, BasicStroke.CAP_ROUND,
                                                  BasicStroke.JOIN_ROUND ) ;

   BasicStroke medium_stroke = new BasicStroke( 6, BasicStroke.CAP_ROUND,
                                                   BasicStroke.JOIN_ROUND ) ;

   BasicStroke narrow_stroke = new BasicStroke( 2 ) ;

   Color background_color = new Color( 0xE7, 0xE7, 0xFF ) ;
```

Kellon viisareiden piirtämisessä käytettävät **BasicStroke**-tyyppiset viivaoliot määritellään tässä datajäseniksi. Kun näitä olioita ei luoda uudelleen joka kerta kelloa piirrettäessä, voidaan säästää vähän prosessointiaikaa.

DoubleClockApplet.java - 1: Kahta eriaikaista kelloa näyttävä apletti.

Jos **Clock2DPanel**-oliota luotaessa ei anneta parametreja, paneelin aikavyöhykkeeksi tulee tietokoneeseen asetetttu aikavyöhyke.

Aikavyöhyke voidaan vaihtoehtoisesti antaa konstruktorille parametrina. Tällöin tulee antaa myös paikan nimi.

```
public Clock2DPanel()
{
   clock_time_zone = TimeZone.getDefault() ;

   time_zone_text = "LOCAL (" + clock_time_zone.getDisplayName() + ")" ;

   FontMetrics metrics_for_font_for_time_zone_text =
                        getFontMetrics( font_for_time_zone_text ) ;

   char[] time_zone_text_as_array = time_zone_text.toCharArray() ;

   time_zone_text_width =
        metrics_for_font_for_time_zone_text.charsWidth(
               time_zone_text_as_array, 0,
               time_zone_text_as_array.length ) ;

   setBackground( background_color ) ;
}

public Clock2DPanel( String   given_place_name,
                     TimeZone given_time_zone )
{
   clock_time_zone = given_time_zone ;

   time_zone_text = given_place_name + " (" +
                    clock_time_zone.getDisplayName() + ")" ;

   FontMetrics metrics_for_font_for_time_zone_text =
                        getFontMetrics( font_for_time_zone_text ) ;

   char[] time_zone_text_as_array = time_zone_text.toCharArray() ;

   time_zone_text_width =
        metrics_for_font_for_time_zone_text.charsWidth(
               time_zone_text_as_array, 0,
               time_zone_text_as_array.length ) ;

   setBackground( background_color ) ;
}
```

Näytölle piirrettävän graafisen kellon alle piirretään teksti johon viittaa **time_zone_text**. Tässä otetaan **FontMetrics**-luokan avulla selville kyseisen tekstin leveys. Näin teksti voidaan piirrettäessä keskittää tarkasti kellon alapuolelle.

DoubleClockApplet.java - 2: Clock2DPanel-luokan konstruktorit.

Animaatiota ylläpitävän säikeen käynnistäminen ja pysäyttäminen tapahtuu tässä ohjelmassa aiemmin nähdyllä tavalla. Molemmat ohjelmassa käytettävät **Clock2DPanel**-oliot käynnistävät oman animaatiosäikeen. Kun tässä säie käynnistetään **start()**-metodia kutsumalla, säikeidenhallintasysteemi generoi automaattisesti kutsun tässä luokassa määriteltyyn **run()**-metodiin.

Metodeita **start_animation_thread()** ja **stop_animation_thread()** kutsutaan aplettiluokan **start()**- ja **stop()**-metodeista. Käytännössä näiden metodien avulla animaatio keskeytetään silloin kun apletin sisältävä ikkuna on minimoituna eikä ole näkyvissä ruudulla.

```
public void start_animation_thread()
{
   if ( thread_that_runs_the_clock  ==  null )
   {
      thread_must_be_executed     =  true ;
      thread_that_runs_the_clock  =  new  Thread( this ) ;

      thread_that_runs_the_clock.start() ;
   }
}

public void stop_animation_thread()
{
   if ( thread_that_runs_the_clock  !=  null )
   {
      thread_that_runs_the_clock.interrupt() ;
      thread_must_be_executed  =  false ;

      thread_that_runs_the_clock  =  null ;
   }
}

public void run()
{
   while ( thread_must_be_executed  ==  true )
   {
      repaint() ;

      try
      {
         Thread.sleep( 200 ) ;  // Suspend for 0.2 seconds.
      }
      catch ( InterruptedException caught_exception )
      {
         // No actions to handle the exception.
      }
   }
}
```

DoubleClockApplet.java - 3: Animaatiota ylläpitävän säikeen tarvitsemat metodit.

Tässä nykyiset aikatiedot otetaan selville **GregorianCalendar**-tyyppisen olion avulla. Kun olio luodaan tähän tapaan, sen sisällöksi tulee tietokoneen senhetkiset aikatiedot.

Nykyisen aikatiedon sisältävä **GregorianCalendar**-olio saadaan siirtymään tarvittaessa eri aikavyöhykkeelle kutsumalla metodia **setTimeZone()**. Tuolle metodille annetaan parametrina aiemmin määritelty **TimeZone**-olio.

```
public void paintComponent( Graphics graphics )
{
   super.paintComponent( graphics ) ;

   Graphics2D  graphics2D  =  (Graphics2D) graphics ;

   Calendar time_now  = new GregorianCalendar() ;

   time_now.setTimeZone( clock_time_zone ) ;

   int current_hours    =  time_now.get( Calendar.HOUR_OF_DAY ) ;
   int current_minutes  =  time_now.get( Calendar.MINUTE ) ;
   int current_seconds  =  time_now.get( Calendar.SECOND ) ;
   int current_milliseconds  =  time_now.get( Calendar.MILLISECOND ) ;


   int  clock_center_point_x  =  getWidth() / 2 ;
   int  clock_center_point_y  =  getHeight() / 2 - 30 ;


   Ellipse2D.Double ring_around_the_clock = new Ellipse2D.Double(
                     clock_center_point_x - ( SECOND_HAND_LENGTH + 16 ),
                     clock_center_point_y - ( SECOND_HAND_LENGTH + 16 ),
                     2 * ( SECOND_HAND_LENGTH + 16 ),
                     2 * ( SECOND_HAND_LENGTH + 16 ) ) ;

   //  The background of the clock face will be white.

   graphics2D.setColor( Color.WHITE ) ;

   graphics2D.fill( ring_around_the_clock ) ;

   graphics2D.setColor( Color.BLACK ) ;
```

getWidth()- ja **getHeight()**-metodeilla saadaan selville "tämän" paneelin leveys ja korkeus. Kun tätä paneelioliota käytetään apletissa ja katsotaan selaimella, paneelin koko ei voi muuttua ohjelman suorituksen aikana, mutta esim. Applet Viewerissä paneelin leveyttä voi muuttaa.

Kellon kehä määritellään tässä **Ellipse2D.Double**-tyyppisenä oliona. Kyseistä kehämäärittelyä käyttäen kellon tausta täytetään valkoisella värillä.

DoubleClockApplet.java - 4: Clock2DPanel-luokan paintComponent()-metodin alku.

Kellon piirrossa hyödynnetään trigonometrisiä staattisia metodeita jotka löytyvät **Math**-luokan sisältä.

Tällä silmukalla kellotatulun kehälle piirretään pisteet. Kellotaulussa on 60 pistettä, joista joka viides piste piirretään hiukan suuremmalla halkaisijalla.

```
// The following loop prints dots on the clock dial.

int dot_index  =  0 ;

while ( dot_index  <  60 )
{
   double dot_angle = dot_index * Math.PI / 30 - Math.PI / 2 ;

   int dot_position_x =  (int) (Math.cos( dot_angle ) * SECOND_HAND_LENGTH
                          +  clock_center_point_x ) ;
   int dot_position_y =  (int) (Math.sin( dot_angle ) * SECOND_HAND_LENGTH
                          +  clock_center_point_y ) ;

   int dot_diameter = 4 ;

   if ( ( dot_index % 5 ) == 0 )
   {
      // Every 5th dot on the clock circle is a larger dot.
      dot_diameter = 8 ;
   }

   graphics.fillOval( dot_position_x - dot_diameter / 2,
                      dot_position_y - dot_diameter / 2,
                      dot_diameter, dot_diameter ) ;

   dot_index  =  dot_index  +  1  ;
}


double hour_hand_angle = ( current_hours * 30  +  current_minutes / 2 )
                         * Math.PI / 180 - Math.PI /2 ;

int hour_hand_end_x = (int) ( Math.cos( hour_hand_angle ) *
                              HOUR_HAND_LENGTH + clock_center_point_x ) ;
int hour_hand_end_y = (int) ( Math.sin( hour_hand_angle ) *
                              HOUR_HAND_LENGTH + clock_center_point_y ) ;

Line2D.Double  current_hour_hand =  new  Line2D.Double(

                   clock_center_point_x, clock_center_point_y,
                   hour_hand_end_x, hour_hand_end_y ) ;
```

DoubleClockApplet.java - 5: ClockPanel-luokan paintComponent()-metodin jatko.

Kellon viisarit määritellään **Line2D.Double**-olioina. Nämä viivaoliot sitten myöhemmin piirretään **Graphics2D**-luokan **draw()**-metodilla. **Line2D.Double**-luokka on luokan **Line2D** sisällä oleva ns. sisäinen luokka (inner class), mutta sitä käytetään kuten mitä tahansa luokkaa.

wide_stroke viittaa luokan alussa luotuun **BasicStroke**-olioon, joka määrittelee mm. tunti- ja minuuttiviisareiden paksuuden sekä kellon ympärille tulevan renkaan paksuuden.

```
double minute_hand_angle = ( (double) current_minutes +
                             (double) current_seconds / 60.0 )
                           * Math.PI / 30 - Math.PI /2 ;

int minute_hand_end_x = (int) ( Math.cos( minute_hand_angle ) *
                           MINUTE_HAND_LENGTH + clock_center_point_x ) ;

int minute_hand_end_y = (int) ( Math.sin( minute_hand_angle ) *
                           MINUTE_HAND_LENGTH + clock_center_point_y ) ;

Line2D.Double  current_minute_hand  =  new  Line2D.Double (

                  clock_center_point_x, clock_center_point_y,
                  minute_hand_end_x, minute_hand_end_y ) ;

graphics2D.setStroke( wide_stroke ) ;

//  The following statement draws a black ring around the clock face.

graphics2D.draw( ring_around_the_clock ) ;

//  Let's draw the hour hand as thick black line.

graphics2D.draw( current_hour_hand ) ;

//  Now we change color and line width to draw a 'phosphorous'
//  line on the hour hand.

graphics2D.setPaint( Color.GREEN ) ;
graphics2D.setStroke( medium_stroke ) ;
graphics2D.draw( current_hour_hand ) ;

//  Next we'll draw the minute hand in the same way as the
//  hour hand was drawn.

graphics2D.setPaint( Color.BLACK ) ;
graphics2D.setStroke( wide_stroke ) ;
graphics2D.draw( current_minute_hand ) ;

graphics2D.setPaint( Color.GREEN ) ;
graphics2D.setStroke( medium_stroke ) ;
graphics2D.draw( current_minute_hand ) ;
```

Englanniksi kellon viisareita kuvataan sanalla "hand". Englantilaisilla kelloilla on siis kädet.

DoubleClockApplet.java - 6: Lisää ClockPanel-luokan paintComponent()-metodia.

Graphics2D-luokassa on yksi yleiskäyttöinen piirtometodi nimeltä **draw()**, jolla voidaan piirtää kaikki graafiset peruskuviot kuten viivat, suorakaiteet ja ellipsit. Nämä kuviot määritellään ensin olioina. Tässä piirretään **Ellipse2D.Double**-olio.

Sekuntiviisari piirretään lopuksi kaiken muun päälle ohuella viivanleveydellä.

```
        //  Now, as the stroke width is 'medium', let's further
        //  decorate the ring around the panel area.

        graphics2D.setPaint( Color.LIGHT_GRAY ) ;
        graphics2D.draw( ring_around_the_clock ) ;

        // Let's print a 16-point dot into the center of the clock.

        graphics2D.setPaint( Color.BLACK ) ;

        graphics2D.fill( new Ellipse2D.Double(
                                clock_center_point_x - 8,
                                clock_center_point_y - 8, 16, 16 ) ) ;


        // And finally we'll draw the second hand.

        graphics2D.setStroke( narrow_stroke ) ;
        graphics2D.setPaint( Color.DARK_GRAY ) ;

        double second_hand_angle = ((double) current_seconds +
                                    (double) current_milliseconds / 1000.0 )
                                    * Math.PI / 30 - Math.PI /2 ;

        int second_hand_end_x = (int) ( Math.cos( second_hand_angle ) *
                                   SECOND_HAND_LENGTH + clock_center_point_x ) ;
        int second_hand_end_y = (int) ( Math.sin( second_hand_angle) *
                                   SECOND_HAND_LENGTH + clock_center_point_y ) ;

        graphics2D.draw( new Line2D.Double(
                           clock_center_point_x, clock_center_point_y,
                           second_hand_end_x, second_hand_end_y  ) ) ;


        graphics2D.setFont( font_for_time_zone_text ) ;

        graphics2D.drawString( time_zone_text,
                               clock_center_point_x - time_zone_text_width / 2,
                              clock_center_point_y + SECOND_HAND_LENGTH + 80 ) ;
    }
}
```

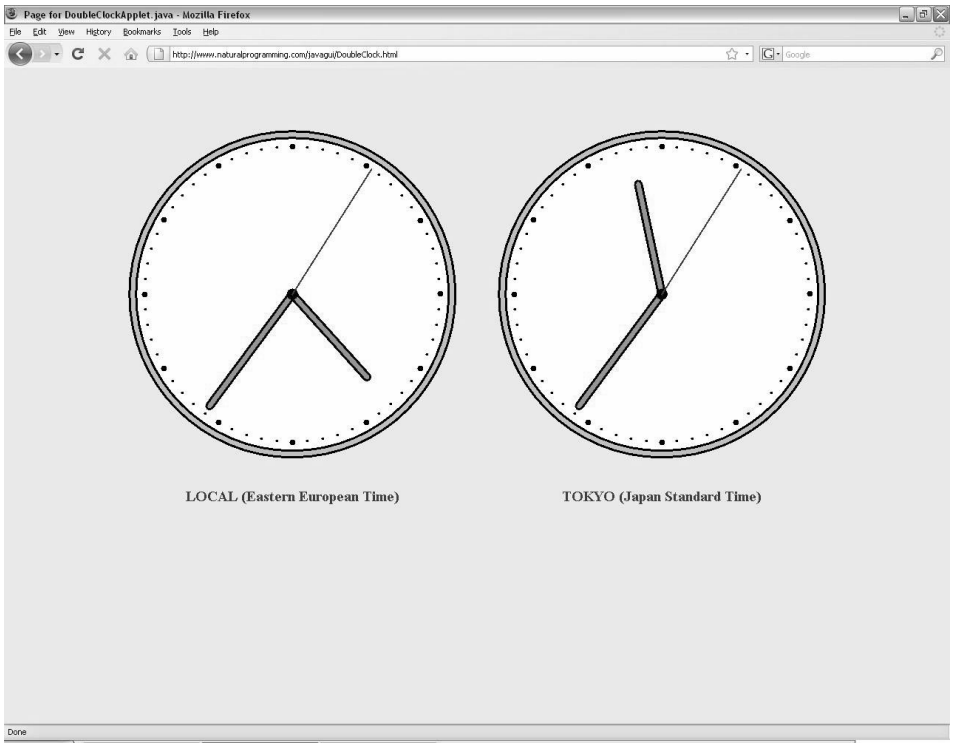
Sekuntiviisaria vastaava viivaolio luodaan tässä juuri ennen sen piirtämistä. Viimeisenä piirretään kellotaulun alle kellon aikavyöhykettä kuvaava teksti.

DoubleClockApplet.java - 7: ClockPanel-luokan paintComponent()-metodin loppuosa.

Tässä aplettiluokassa luodaan kaksi **`Clock2DPanel`**-oliota erilaisilla aikavyöhykkeillä. Vasemman kellon aikavyöhykeeksi tulee tietokoneeseen asetettu aikavyöhyke. Paneelit kiinnitetään aplettiin **`add()`**-metodilla. Apletin layoutiksi asetetaan tähän tapaan **`GridLayout`** jolla paneelit tulevat vierekkäin ruudulle.

```
public class DoubleClockApplet extends JApplet
{
   Clock2DPanel  panel_for_left_clock ;
   Clock2DPanel  panel_for_right_clock ;

   public void init()
   {
      Container content_pane_of_this_applet  =  getContentPane() ;

      content_pane_of_this_applet.setLayout( new GridLayout( 1, 2 ) ) ;

      panel_for_left_clock   =  new Clock2DPanel() ;
      panel_for_right_clock  =  new Clock2DPanel( "TOKYO",
                                                  TimeZone.getTimeZone(
                                                       "Asia/Tokyo" ) ) ;

      content_pane_of_this_applet.add( panel_for_left_clock ) ;
      content_pane_of_this_applet.add( panel_for_right_clock ) ;
   }

   public void start()
   {
      panel_for_left_clock.start_animation_thread() ;
      panel_for_right_clock.start_animation_thread() ;
   }

   public void stop()
   {
      panel_for_left_clock.stop_animation_thread() ;
      panel_for_right_clock.stop_animation_thread() ;
   }

}
```

DoubleClockApplet.java - 8. Paneeliolioita käyttävä aplettiluokka.



DoubleClockApplet.java - X. Ohjelmaa suoritetaan tässä Suomessa.

Luokka **Wheel** edustaa tässä ohjelmassa ruudulle piirrettävää pyörää (rengasta). **Wheel**-luokassa on datajäseninä **wheel_radius** eli pyörän säde, **rotation_increment** joka on astemäärä jolla pyörä pyörähtää yhdellä **rotate()**-metodin kutsulla, **current_rotation** joka on astemäärä (0 ... 360) jonka pyörä on nykyisellään pyörinyt, sekä **wheel_image** joka on erillinen piirtopinta jolle pyörä piirretään. **Wheel**-luokassa ei ole datajäseninä muuttujia jotka kuvaisivat pyörä sijaintia ruudulla. Vaikka yleensä tämäntyyppisissä luokissa tällaisia muuttujia käytetään, tästä luokasta paikkatieto puuttuu koska pyörä sijaintia ei ohjelmassa muuteta.

```
//  WheelsApplet.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.event.* ;
import java.awt.geom.* ;
import java.awt.image.* ;
import javax.swing.* ;

class Wheel
{
   int  wheel_radius ;

   int  rotation_increment  = 6 ;  //  in degrees

   int  current_rotation  =  0 ;  //  in degrees

   BufferedImage wheel_image ;

   public Wheel( int   given_radius )
   {
      wheel_radius  =  given_radius ;

      //  A BufferedImage object is used only by the draw() method, but it
      //  is best to create the image only once, as creating it
      //  something like 25 times a second inside the draw() method
      //  requires quite a lot of processing power and memory.

      wheel_image  =  new BufferedImage( wheel_radius * 2,
                                         wheel_radius * 2,
                                         BufferedImage.TYPE_INT_RGB ) ;
   }
```

Pyörä piirretään siten että se piirretään ensin kokonaan **BufferedImage**-tyyppisen olion piirtopintaan ja lopuksi kyseinen **BufferedImage**-olio piirretään varsinaiselle ruudulle. **BufferedImage**-olio kannattaa kuitenkin luoda vain kertaalleen ohjelmassa.

WheelsApplet.java - 1: Apletti jonka näyttämät renkaat pyörivät ja haluttaessa pysähtyvät.

rotate()-metodilla pyörä saadaan pyörähtämään niin monta astetta kun on **rotation_increment**-muuttujan arvo. Kyseisen muuttujan arvoa voidaan puolestaan säätää **increase_speed()**- ja **decrease_speed()**-metodeilla. Jos **rotation_increment**-muuttujan arvo on negatiivinen pyörä pyörii vastapäivään. **rotation_increment** voi tässä ohjelmassa saada arvot -12, -9, -6, -3, 0, 3, 6, 9, ja 12, jotka ovat mahdollisia pyörähdyksessä käytettäviä astemääriä.

```
public void rotate()  //  This is to be called by an animation thread.
{
   if ( rotation_increment  >= 0 )
   {
      //  Pyörä on pyörimässä myötäpäivään.

      if ( current_rotation  <=  0 )
      {
         current_rotation  =  360 - rotation_increment ;
      }
      else
      {
         current_rotation  =  current_rotation - rotation_increment ;
      }
   }
   else
   {
      //  Pyörän pyörimissuunta on vastapäivään.
      if ( current_rotation  -  rotation_increment < 360 )
      {
         current_rotation  =  current_rotation - rotation_increment ;
      }
      else
      {
         current_rotation  =  0 ;
      }
   }
}

public void increase_speed()
{
   if (  rotation_increment  <  12 )
   {
      rotation_increment  =  rotation_increment  +  3 ;
   }
}

public void decrease_speed()
{
   if (  rotation_increment  >  -12 )
   {
      rotation_increment  =  rotation_increment  -  3 ;
   }
}
```

WheelsApplet.java - 2: Wheel-luokan metodeja.

Koordinaatiston nollapiste siirretään tässä pyärän kuvan keskipisteeseen eri pyörän napaan.

Pyörien pinnat ja vanne saadaan aikaiseksi piirtämällä piirakkapalan (PIE) muotoisia kuvioita pyörän keskipisteestä lähtien. 30 on täsä pinnojen välinen kulma asteina. Tekemällä sama piirrosoperaatio kahdella eri värillä ja viivanleveydellä saadaa pyöriin vauhtiraidat.

```
   public void draw( Graphics2D  graphics2D,
                     Point       wheel_upper_left_corner )
   {
      Graphics2D  image_graphics  =  wheel_image.createGraphics() ;

      image_graphics.setPaint( Color.white ) ;
      image_graphics.fill( new Rectangle2D.Double( 0, 0, wheel_radius * 2,
                                                   wheel_radius * 2 ));
      image_graphics.translate( wheel_radius, wheel_radius ) ;

      int  right_spoke_angle  =  0 ;

      while ( right_spoke_angle  <  360 )
      {
         image_graphics.setPaint( Color.black ) ;
         image_graphics.setStroke( new BasicStroke( 8 ) ) ;

         image_graphics.draw( new Arc2D.Double( -wheel_radius + 4,
                                                -wheel_radius + 4,
                                                wheel_radius * 2 - 8,
                                                wheel_radius * 2 - 8,
                                                right_spoke_angle
                                                   +  current_rotation,
                                                30, Arc2D.PIE ) ) ;

         image_graphics.setPaint( Color.yellow ) ;
         image_graphics.setStroke( new BasicStroke( 2 ) ) ;

         image_graphics.draw( new Arc2D.Double( -wheel_radius + 4,
                                                -wheel_radius + 4,
                                                wheel_radius * 2 - 8,
                                                wheel_radius * 2 - 8,
                                                right_spoke_angle
                                                   +  current_rotation,
                                                30, Arc2D.PIE ) ) ;
         right_spoke_angle  +=  30 ;
      }

      //  Seuraavat kaksi riviä piirtävät renkaalle punaisen navan.
      image_graphics.setPaint( Color.red ) ;
      image_graphics.fill( new Ellipse2D.Double( -8, -8, 16, 16 ) ) ;

      //  Renkaan kuva piirretään lopuksi annettuun paikkaan paneelilla.
      graphics2D.drawImage( wheel_image,
                            wheel_upper_left_corner.x,
                            wheel_upper_left_corner.y, null ) ;
   }
}
```

WheelsApplet.java - 3: Wheel-luokan draw()-metodi.

Wheel-olion paikkaa **Wheel-Panel**-olion sisällä ei tässä määritellä. **WheelPael**-luokan **paintComponent()**-metodissa pyörä asetetaan keskelle paneelia.

Pyörän pyörimistä kontrolloiva "vaihteisto" on tässä saatu aikaiseksi kahdella napilla joista toinen muuttaa nopeutta pykälää suuremmaksi ja toinen pykälää pienemmäksi. Nopeus voi mennä negatiiviseksi, jolloin pyörä alkaa pyörimään vastapäivään. Pyörän kontrollointinappeihin lisätään **ActionListener**it sillä tavalla että luodaan nimettömän sisäisen luokan olio. Sisäinen luokka sisältää tarvittavan **actionPerformed()**-metodin jota sitten kutsutaan automaattisesti kun nappia painetaan.

```
class WheelPanel extends JPanel implements Runnable
{
   Thread   thread_to_rotate_the_wheels ;
   boolean  thread_must_be_active  =  false ;

   int    wheel_radius  ;
   Wheel  wheel_on_this_panel ;

   public WheelPanel( int given_wheel_radius )
   {
      wheel_radius  =  given_wheel_radius ;
      wheel_on_this_panel  = new  Wheel( wheel_radius ) ;

      setLayout( new BorderLayout() ) ;

      JButton  button_to_gear_up    =  new  JButton( "GEAR UP" ) ;
      JButton  button_to_gear_down  =  new  JButton( "GEAR DOWN" ) ;

      button_to_gear_up.addActionListener( new ActionListener()
      {
         public void actionPerformed( ActionEvent event )
         {
            wheel_on_this_panel.increase_speed() ;
         }
      } ) ;

      button_to_gear_down.addActionListener( new ActionListener()
      {
         public void actionPerformed( ActionEvent event )
         {
            wheel_on_this_panel.decrease_speed() ;
         }
      } ) ;

      JPanel  wheel_control_panel  =  new  JPanel() ;

      wheel_control_panel.add( button_to_gear_down ) ;
      wheel_control_panel.add( button_to_gear_up ) ;

      add( "South", wheel_control_panel ) ;
   }
```

WheelsApplet.java - 4: WheelPanel luokan datakentät ja konstruktori.

Animaatio, siis pyörän pyöriminen, käynnistetään aplettiluokasta käsin näiden metodien avulla. Kun **Thread**-oliolle kutsutaan **start()**-metodia, **run()**-metodin suoritus aktivoituu automaattisesti. **run()**-metodia suoritetaan kunnes se päättyy, eli niin kauan kuin muuttujan **thread_must_be_active** arvo on **true**.

```
public void start_animation_thread()
{
   if ( thread_to_rotate_the_wheels  ==  null )
   {
      thread_to_rotate_the_wheels  =  new  Thread( this ) ;
      thread_must_be_active  =  true ;

      thread_to_rotate_the_wheels.start() ;
   }
}

public void stop_animation_thread()
{
   if ( thread_to_rotate_the_wheels  !=  null )
   {
      thread_must_be_active        =  false ;
      thread_to_rotate_the_wheels  =  null ;
   }
}

public void run()
{
   while ( thread_must_be_active  ==  true )
   {
      wheel_on_this_panel.rotate() ;
      repaint() ;

      try
      {
         Thread.sleep( 40 ) ;  //  25 times per second
      }
      catch ( InterruptedException caught_exception )
      {
         // No actions to handle the exception.
      }
   }
}
```

run()-metodin suoriintuessa **rotate()**-metodia kutsutaan pyöräolion suhteen 25 kertaa sekunnissa. Ruudulle piirtyy näin usein uusi kuva pyörästä, ja pyörä näyttää pyörivän. Pyörä piirretään ruudulle 25 kertaa sekunnissa myös silloin kun sen nopeus on nolla eli pyörä on pysähdyksissä.

WheelsApplet.java - 5: Animointisäikeen suoriintumista kontrolloivat metodit.

Pyörän piirtäminen paneelille on varsin yksinkertainen operaatio koska sen tekee **`Wheel`**-luokan **`draw()`**-metodi. Ennen pyörän piirtämistä otetaan selville paneelin leveys ja korkeus. Pyörä piirretään vaakasuunnassa keskelle paneelia. Pystysuunnassa pyörän yläpuolelle jätetään sopivasti tilaa lokasuojaa varten.

```
   public void paintComponent( Graphics graphics )
   {
      super.paintComponent( graphics ) ;
      Graphics2D  graphics2D  =  (Graphics2D) graphics ;

      setBackground( Color.white ) ;

      //  The 'earth level' for the wheels is 75 points from the
      //  southern border of the panel.

      Point wheel_upper_left_corner = new Point(

                              getWidth() / 2 - wheel_radius,
                              getHeight() - wheel_radius * 2 - 75 ) ;

      wheel_on_this_panel.draw( graphics2D, wheel_upper_left_corner ) ;

      //  Now we'll print a "mudguard" for the wheel.
      //  The mudguard is an open arc whose distance from the wheel
      //  is 0.5 times the radius of the wheel.

      graphics2D.setStroke( new BasicStroke( 8 ) ) ;

      graphics2D.draw( new Arc2D.Double(
                 wheel_upper_left_corner.x - wheel_radius * 0.5 + 4,
                 wheel_upper_left_corner.y - wheel_radius * 0.5 + 4,
                 wheel_radius * 2 * 1.5 - 8,
                 wheel_radius * 2 * 1.5 - 8,
                 0, 180, Arc2D.OPEN ) ) ;

      graphics2D.setPaint( Color.red ) ;
      graphics2D.setStroke( new BasicStroke( 2 ) ) ;

      graphics2D.draw( new Arc2D.Double(
                 wheel_upper_left_corner.x - wheel_radius * 0.5 + 4,
                 wheel_upper_left_corner.y - wheel_radius * 0.5 + 4,
                 wheel_radius * 2 * 1.5 - 8,
                 wheel_radius * 2 * 1.5 - 8,
                 0, 180, Arc2D.OPEN ) ) ;
   }
}
```

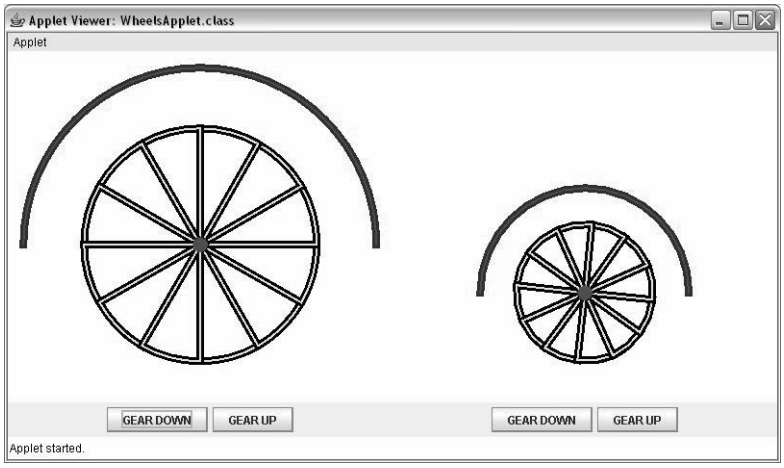
Vauhtiviivallinen lokasuoja saadaan aikaiseksi piirtämällä sama 180 asteen kaari kahdella eri viivanleveydellä.

WheelsApplet.java - 6: WheelPanel-luokan paintComponent()-metodi.

Kun apletin layout on **GridLayout**, voivat pyörä-paneelit jakaa apletin alueen tasan puoliksi.

```
public class WheelsApplet extends JApplet
{
   WheelPanel  left_wheel_panel, right_wheel_panel ;

   public void init()
   {
      Container content_pane_of_this_applet  =  getContentPane() ;

      content_pane_of_this_applet.setLayout( new GridLayout( 1, 2 ) ) ;

      left_wheel_panel   =  new WheelPanel( 125 ) ; // wheel radius: 125 points
      right_wheel_panel  =  new WheelPanel(  75 ) ; // wheel radius:  75 points

      content_pane_of_this_applet.add( left_wheel_panel ) ;
      content_pane_of_this_applet.add( right_wheel_panel ) ;
   }

   public void start()
   {
      left_wheel_panel.start_animation_thread() ;
      right_wheel_panel.start_animation_thread() ;
   }

   public void stop()
   {
      left_wheel_panel.stop_animation_thread() ;
      right_wheel_panel.stop_animation_thread() ;
   }
}
```

WheelsApplet.java - 7. Aplettiluokka jossa käytetään kahta WheelPanel-oliota.



Pyörien pyöriminen näkyy huonosti näin paperikuvassa. Pyörät voivat tosin olla pysähtyneenä jos nappuloilla on valittu sopiva "vaihde".

WheelsApplet.java - X. Kaksi rinnakkaista WheelPanel-oliota muodostavat tämän apletin.

Menuja sisältävä Java-sovellus (ja Java-apletti)

Tämä on Swing-luokkiin perustuva sovellus (siis ei apletti). Tämä on rakennettu siten että sovelluksen toiminnot ovat kaikki **JFrame**-luokasta johdetussa luokassa. Erillistä paneeliluokkaa ei tässä käytetä.

Ohjelman nimenä on "Perinteinen Windows-sovellus" koska tässä demonstroidaan menujen käyttöä ja perinteisesti Windows-sovelluksissa on ollut sellaisia menuja kuten **File**-, **Edit**-, **Settings**- ja **Help**-menut. Menuja ei voida kiinnittä **JPanel**-luokkaan perustuviin olioihin, vaan menuja käytettäessä on käytettävä kantaluokkana **JFrame**-luokkaa.

Tästä ohjelmasta löytyy aplettiversio sähköisessä muodossa tiedostosta **PerinteinenWindowsSovellusApplet.java**.

```
//  PerinteinenWindowsSovellus.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.event.*;
import javax.swing.* ;

class PerinteinenWindowsFrame extends JFrame implements ActionListener
{
   static final int WINDOW_WIDTH   =  500 ;
   static final int WINDOW_HEIGHT  =  400 ;

   JMenuItem  file_open_menu_item  =  new  JMenuItem( "Open" ) ;
   JMenuItem  file_save_menu_item  =  new  JMenuItem( "Save" ) ;
   JMenuItem  file_exit_menu_item  =  new  JMenuItem( "Exit" ) ;

   JMenuItem  edit_copy_menu_item  =  new  JMenuItem( "Copy" ) ;
   JMenuItem  edit_paste_menu_item =  new  JMenuItem( "Paste" ) ;

   JCheckBoxMenuItem  cyan_background_menu_item  =
                         new JCheckBoxMenuItem( "Cyan background" ) ;

   JRadioButtonMenuItem  small_text_menu_item  =
                                 new JRadioButtonMenuItem( "Small" ) ;
   JRadioButtonMenuItem  medium_text_menu_item  =
                                 new JRadioButtonMenuItem( "Medium" ) ;
   JRadioButtonMenuItem  large_text_menu_item  =
                                 new JRadioButtonMenuItem( "Large" ) ;

   JMenuItem  about_menu_item  =  new  JMenuItem( "About" ) ;

   String  viimeksi_valitun_menu_itemin_nimi  =  "EI VALINTOJA TEHTY." ;
```

Menut ovat sellaisia että yhteen **JMenu**-tyypin olioon voidaan sijoittaa joukko menu-itemeitä. Nämä menu-itemit ovat tässä sovelluksessa luokkien **JMenuItem**, **JCheckBoxMenuItem** ja **JRadioButtonMenuItem** olioita. Luokan alussa määritellään datajäseniksi nämä menu-itemit, jotka myöhemmin konstruktorissa kiinnitetään menuihin.

PerinteinenWindowsSovellus.java - 1: Menujen avulla käytettävä Windows-sovellus.

Jokaiseen menu-itemiin lisätään "kuuntelijaksi" tämä luokka. Tässä luokassa on **actionPerformed()**-metodi, jota kutsutaan automaattisesti silloin kun joku menu-itemi valitaan. Menu-itemeitä kuunnellaan siis samannimisellä metodilla kuin esim. **JButton**-olioita. Itse asiassa luokat **JMenuItem** ja **JButton** ovat "sukulaisia" keskenään sillä molempien yläluokka on **AbstractButton**.

Tässä määritellään **setMnemonic()**-metodin avulla että **File**-menussa olevan **Save**-itemin voi valita näppäilemällä Alt-S.

```
public PerinteinenWindowsFrame()
{
   getContentPane().setBackground( Color.white ) ;

   setSize( WINDOW_WIDTH, WINDOW_HEIGHT ) ;

   setTitle( "PERINTEINEN WINDOWS-SOVELLUS MENUILLA" ) ;

   file_open_menu_item.setMnemonic( 'O' ) ;
   file_open_menu_item.addActionListener( this ) ;
   file_save_menu_item.setMnemonic( 'S' ) ;
   file_save_menu_item.addActionListener( this ) ;
   file_exit_menu_item.setMnemonic( 'x' ) ;
   file_exit_menu_item.addActionListener( this ) ;

   JMenu  file_menu  =  new  JMenu( "File" ) ;
   file_menu.setMnemonic( 'F' ) ;

   file_menu.add( file_open_menu_item ) ;
   file_menu.add( file_save_menu_item ) ;
   file_menu.add( file_exit_menu_item ) ;

   edit_copy_menu_item.addActionListener( this ) ;
   edit_paste_menu_item.addActionListener( this ) ;

   JMenu  edit_menu  =  new  JMenu( "Edit" ) ;

   edit_menu.add( edit_copy_menu_item ) ;
   edit_menu.add( edit_paste_menu_item ) ;

   JMenu  settings_menu  =  new  JMenu( "Settings" ) ;

   cyan_background_menu_item.addActionListener( this ) ;

   settings_menu.add( cyan_background_menu_item ) ;
```

Jokainen menu-itemi lisätään **add()**-metodilla aiemmin määritelty **JMenu**-olioon. Kyseinen **add()**-metodi on määritelty **JMenu**-luokassa tai jossain sen yläluokassa. Tässä kiinnitetään menuun **JCheckBoxMenuItem**-olio aivan samaan tapaan kuin aiemmin menuihin lisätyt **JMenuItem**-oliot.

PerinteinenWindowsSovellus.java - 2: PerinteinenWindowsFrame-luokan konstruktorin alku.

Text size -menu on **Settings**-menun alimenu, josta voidaan valita näytössä käytettävän tekstin koko kolmesta eri vaihtoehdosta. Nuo kolme tekstin kokoa tarkoittavaa menu-itemiä ovat `JRadioButtonMenuItem`-olioita. Nämä oliot kiinnitetään tässä ensin `ButtonGroup`-olioon ja sitten ne kiinnitetään varsinaiseen **Text size** -menuun. Kun `JRadioButtonMenuItem`-oliot kuuluvat johonkin `ButtonGroup`-olioon, vain yksi nappi voi kerrallaan tulla valituksi. `setSelected()`-metodilla pistetään keskikokoinen teksti alussa valituksi.

```
        small_text_menu_item.addActionListener( this ) ;
        medium_text_menu_item.addActionListener( this ) ;
        large_text_menu_item.addActionListener( this ) ;

        medium_text_menu_item.setSelected( true ) ;

        ButtonGroup  text_size_button_group  =  new  ButtonGroup() ;

        text_size_button_group.add( small_text_menu_item ) ;
        text_size_button_group.add( medium_text_menu_item ) ;
        text_size_button_group.add( large_text_menu_item ) ;

        JMenu  text_size_menu  =  new  JMenu( "Text size" ) ;

        text_size_menu.add( small_text_menu_item ) ;
        text_size_menu.add( medium_text_menu_item ) ;
        text_size_menu.add( large_text_menu_item ) ;

        settings_menu.add( text_size_menu ) ;

        JMenu  help_menu  =  new  JMenu( "Help" ) ;

        about_menu_item.addActionListener( this ) ;
        help_menu.add( about_menu_item ) ;

        JMenuBar menubar_of_this_window  =  new JMenuBar() ;

        menubar_of_this_window.add( file_menu ) ;
        menubar_of_this_window.add( edit_menu ) ;
        menubar_of_this_window.add( settings_menu ) ;
        menubar_of_this_window.add( help_menu ) ;

        setJMenuBar( menubar_of_this_window ) ;

        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE ) ;
    }
```

Tällä metodikutsulla määritellään että tästä sovelluksesta poistutaan siinä tapauksessa kun tässä määriteltävä ikkuna suljetaan ikkunan yläkulman "sulkupainikkeella".

Kun kaikki menut on saatu rakennettua, luodaan `JMenuBar()`-olio ja tähän olioon kiinnitetään taas `add()`-metodilla aiemmin luodut menut. Lopuksi `JMenuBar`-olio asetetaan tämän ikkunan "menutangoksi" `setMenuBar()`-metodilla.

PerinteinenWindowsSovellus.java - 3: PerinteinenWindowsFrame-konstruktorin loppuosa.

`actionPeformed()`-metodia kutsutaan automaattisesti kun jossain menussa jokin menu-item valitaan. `if`-rakenteella tutkitaan ensin että aiheuttiko tapahtuman `JMenuItem`-luokkaan (tai sen johonkin alaluokkaan) kuuluva olio. Ohjelma kyllä toimisi oikein ilman tätä testaustakin.

Tässä otetaan talteen ruudulle tulostettavaksi valitun menu-itemin teksti. Sen jälkeen asetetaan taustaväriksi Cyan jos tämmöisen valinnan mahdollistava `JCheckBoxMenuItem` on valittuna.

```
public void actionPerformed( ActionEvent event )
{
   if ( event.getSource() instanceof JMenuItem )
   {
      JMenuItem valittu_menu_item  = (JMenuItem) event.getSource() ;

      viimeksi_valitun_menu_itemin_nimi  =  valittu_menu_item.getText() ;

      if ( cyan_background_menu_item.isSelected() == true )
      {
         getContentPane().setBackground( Color.cyan ) ;
      }
      else
      {
         getContentPane().setBackground( Color.white ) ;
      }

      if ( valittu_menu_item  ==  file_exit_menu_item )
      {
         setVisible( false ) ;
         System.exit( 0 ) ;
      }
      else if ( valittu_menu_item  ==  about_menu_item )
      {
         JOptionPane.showMessageDialog( this,
                                 "Tämä ohjelma matkii perinteistä"
                               + "\nmenuihin perustuvaa windows-sovellusta",
                                 "About", JOptionPane.PLAIN_MESSAGE ) ;
      }

      repaint() ;
   }
}
```

Metodikutsulla `System.exit( 0 );` saadaan ohjelman suoriintuminen päättymään siinä tapauksessa kun **File**-menusta on valittu **Exit**-toiminto. `System`-luokan `exit()`-metodi pysäyttää Java-virtuaalikoneen. Parametrina annettava nolla tarkoittaa normaalia ohjelman päättymistä.

`JOptionPane`-luokka sisältää useita staattisia metodeita joilla voidään näyttää käyttäjälle erillistä dialogi-ikkunaa. Metodilla `showMessageDialog()` saadaan näkymään dialogi jossa on näytettävän viestin lisäksi OK-nappi.

PerinteinenWindowsSovellus.java - 4: Menu-valintoihin reagoiva actionPeformed()-metodi.

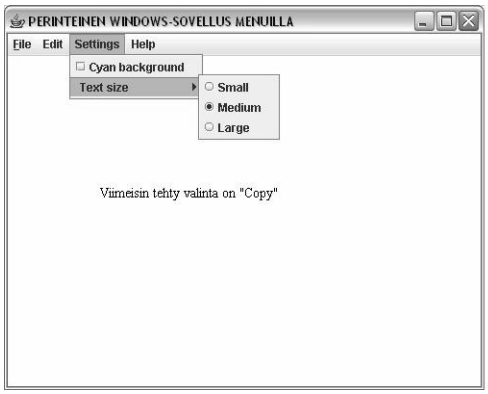
JFrame-luokkaan pohjautuvaan luokkaan kirjoitettava ruudunmaalausmetodi on nimeltään **paint()**. **JPanel**-luokkaan perustuvissa luokissa se on **paintComponent()**. Metodin **paint()** alussa tulee kutsua varatun sanan **super** avulla yläluokan vastaavaa metodia.

Metodin **isSelected()** avulla tutkitaan, mikä **JRadioButtonMenuItem**-olioista on valittuna, ja sen mukaan asetetaan käytettävä kirjasintyyppi. Tuolla kirjasintyypillä tulostetaan myöhemmin ruudulle valitun menu-itemin teksti.

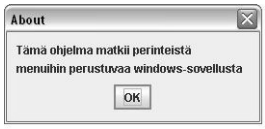
```
   public void paint( Graphics graphics )
   {
      super.paint( graphics ) ;

      if ( small_text_menu_item.isSelected() == true )
      {
         graphics.setFont( new Font( "Serif", Font.PLAIN, 10 ) ) ;
      }

      if ( medium_text_menu_item.isSelected() == true )
      {
         graphics.setFont( new Font( "Serif", Font.PLAIN, 14 ) ) ;
      }

      if ( large_text_menu_item.isSelected() == true )
      {
         graphics.setFont( new Font( "Serif", Font.PLAIN, 18 ) ) ;
      }

      graphics.drawString( "Viimeisin tehty valinta on \""
                        +   viimeksi_valitun_menu_itemin_nimi  +  "\"",
                           100, 200 ) ;
   }
}

public class PerinteinenWindowsSovellus
{
   public static void main( String[] ei_kaytossa )
   {
      PerinteinenWindowsFrame  taman_sovelluksen_ikkuna  =
                                  new  PerinteinenWindowsFrame() ;
      taman_sovelluksen_ikkuna.setVisible( true ) ;
   }
}
```

Luotu **PerinteinenWindowsFrame**-olio pannaan näkyville ruudulle kutsumalla **setVisible()**-metodia. Parametrilla **true** määrätään että on totta että ikkuna on näkyvillä ruudulla.

PerinteinenWindowsSovellus.java - 5. Ohjelman loppuosa.



Hiirellä on tässä klikattu **Settings**-menua ja menty siinä **Text size** -menuun. Lopullista valintaa ei tässä kuitenkaan ole vielä suoritettu. Menusta on aiemmin valittu **Edit** > **Copy**, minkä ikkunassa oleva teksti ilmoittaa. Menuista valittavat toiminnot **File** > **Open**, **File** > **Save**, **Edit** > **Copy** ja **Edit** > **Paste** eivät aiheuta muuta toimintaa kuin maininnan asiasta tekstinä.



Kun suoritetaan valinnat **Help** > **About**, aukeaa tämän näköinen dialogi-ikkuna, joka täytyy sulkea OK-nappia painamalla. Tätä dialogia varten ei ohjelmassa tarvitse määritellä mitään erityistä luokkaa, vaan dialogi-ikkuna saadaan aikaiseksi ja toimimaan yksinkertaisesti `JOptionPane.showMessageDialog()`-metodia kutsumalla.

PerinteinenWindowsSovellus.java - X.

Sähköisessä muodossa on saatavilla myös ohjelma nimeltä **PerinteinenWindowsSovellusApplet.java**, joka on aplettiversio tässä esitetystä ohjelmasta. Kyseinen ohjelma on lähes "sama" ohjelma kuin tässä esitetty. Oleelliset erot ovat seuraavat:

- Aplettiversiossa toiminnot sisältävä luokka on johdettu `JApplet`-luokasta eikä `JFrame`-luokasta. `JApplet`-luokan sisään rakennettuihin apletteihin voidaan siis liittää menuja, muttä tällöin apletin toiminnot on rakennettava varsinaisen aplettiluokan sisään eikä erillistä **JPanel**-luokkaan perustuvaa paneelia kannata käyttää.
- Aplettiluokassa on `init()`-metodi eikä lainkaan konstruktoria. `init()`-metodissa luodaan menut, mutta siellä ei voi säätää ikkunan kokoa eikä asettaa ikkunan otsikkoa.
- Apletissa ei voi poistua sovelluksesta valitsemalla **File** > **Exit**.

Esimerkki tiedoston käsittelystä Java-sovelluksessa

Tällä ohjelmalla voi luoda uuden tekstitiedoston. Teksti kirjoitetaan ruudulla `JTextArea`-olion sisään. `JTextArea`-olio käsittää koko ruudun alueen lukuunottamatta alareunassa olevaa paneelia.

Tekstin "File name:" tilalle ilmestyy teksti "INVALID FILE NAME:" silloin jos ohjelman käyttäjä antaa tiedostonimen jonka laajennnus on jotain muuta kuin **.txt** tai jos nimi on vähemmän kuin 5 merkkiä pitkä. Save-nappi tulee aktiiviseksi siinä tilanteessa kun ruudulla on tekstiä jota ei vielä ole talletettu tiedostoon.

NewTextFileApplication.java - X. Tekstiä ei ole tässä vielä talletettu tiedostoon.

Tällä ohjelmalla voi luoda uuden tekstitiedoston. Ohjelmalla ei voi lukea jo olemassa olevan tiedoston sisältöä tiedostosta ja alkaa muuttamaan sitä.

Käytettävä ikkunaluokka toteuttaa kolme eri rajapintaa. `DocumentListener`-rajapinnan avulla tiedetään milloin `JTextArea`-komponentin tekstiä muutetaan, `ActionListener` huolehtii Save-nappiin reagoinnista, ja `WindowListener`in avulla kuunnellaan ikkunan sulkeutumista.

```
//  NewTextFileApplication.java  (c) Kari Laitinen

import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;
import javax.swing.event.* ;
import java.io.* ;    //  Classes for file handling.

class NewTextFileFrame extends JFrame
                       implements DocumentListener, ActionListener,
                                  WindowListener
{
   public static final int FRAME_WIDTH   =  500 ;
   public static final int FRAME_HEIGHT  =  400 ;

   JTextArea   writing_area ;
   JLabel      file_name_label ;
   JTextField  file_name_field ;
   JButton     save_button ;

   boolean  text_has_to_be_saved_to_file  =  false ;
```

NewTextFileApplication.java - 1: Ohjelma jolla voi luoda uuden tekstitiedoston.

Luokan **JTextArea** avulla voidaan luoda ruudulle alue, johon voidaan kirjoittaa useampi rivi tekstiä. Tässä luotavan tekstialueen ohjeellinen rivimäärä on 20 riviä ja 60 on ohjeellinen määrä merkkejä rivillä.

Luokka **JTextField** on luokan **JTextArea** sisarluokka, sillä molempien yläluokka on **JTextComponent**. **JTextField**-olio voi sisältää vain yhden rivin tekstiä. Molempien tekstiä sisältävien näyttökomponenttien tekstin muuttumista voidaan kuunnella **DocumentListener**-rajapinnan metodeilla.

```
public NewTextFileFrame()
{
   setTitle( "CREATE A NEW TEXT FILE" ) ;
   setSize( FRAME_WIDTH, FRAME_HEIGHT ) ;
   setBackground( Color.white ) ;

   setLayout( new BorderLayout() ) ;

   writing_area  =  new  JTextArea( 20, 60 ) ;
   writing_area.getDocument().addDocumentListener( this ) ;
   add( "Center", writing_area ) ;

   file_name_label  =  new  JLabel( " File name:" ) ;

   file_name_field  =  new  JTextField( "tmp.txt", 20 ) ;
   file_name_field.getDocument().addDocumentListener( this ) ;

   save_button  =  new  JButton( "Save" ) ;
   save_button.setEnabled( false ) ;
   save_button.addActionListener( this ) ;

   JPanel  operations_panel  =  new  JPanel() ;

   operations_panel.add( file_name_label ) ;
   operations_panel.add( file_name_field ) ;
   operations_panel.add( save_button ) ;

   add( "South", operations_panel ) ;

   addWindowListener( this ) ;

   //  When the default close operation is DO_NOTHING_ON_CLOSE
   //  the windowClosing() method determines what actually happens
   //  when the window is being closed.

   setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE ) ;

}
```

Tässä sovelluksessa ikkunan sulkeminen hallitaan **WindowListener**-rajapinnan metodilla **windowClosing()**. Jotta näin voidaan tehdä, tulee oletusarvoiseksi ikkunan sulkemisoperaatioksi määrätään tässä "ÄLÄ TEE MITÄÄN".

NewTextFileApplication.java - 2: NewTextFileFrame-luokan konstruktori.

Tässä tutkitaan ensin muutettiinko tekstiä **JTextArea**-oliossa vai **JTextField**-oliossa. Jos tiedostonimen sisältävää tekstikenttää muutettiin, halutaan varmistua siitä että tiedostonimi on hyväksyttävä. Sen pitää päättyä .txt ja siinä pitää olla vähintaään viisi merkkiä.

Jos tiedostoon talletettavaa tekstiä muutettiin, aktivoidaan Save-nappi ja asetetaan asiantilaa kuvaava muuttuja arvoon **true**.

```
void process_document_event( DocumentEvent event )
{
   if ( event.getDocument()  ==  file_name_field.getDocument() )
   {
      //  The file name field was modified. It is required that
      //  the length of the file name is at least 5 characters,
      //  and the file name extension is .txt

      if ( file_name_field.getText().length() <  5  ||
           ! file_name_field.getText().endsWith( ".txt" ) )
      {
         file_name_label.setText( "INVALID FILE NAME: " ) ;
      }
      else
      {
         file_name_label.setText( "        File name:" ) ;
      }
   }
   else if ( event.getDocument()  ==  writing_area.getDocument() )
   {
      save_button.setEnabled( true ) ;
      text_has_to_be_saved_to_file  =  true ;
   }
}

public void insertUpdate( DocumentEvent event )
{
   process_document_event( event ) ;
}

public void removeUpdate( DocumentEvent event )
{
   process_document_event( event ) ;
}

public void changedUpdate( DocumentEvent event )
{
}  //  This kind of update is not possible in the case of JTextField
```

Näitä metodeita kutsutaan automaattisesti kun joko kirjoitettavan tekstin sisältävällä tekstialueella tai tiedostonimen sisältävässä tekstikentässä tehdään muutos tekstiin. Modelmmat metodit siirtävät "dokumenttitapahtuman" prosessoinnin **process_document_event()**-metodille.

NewTextFileApplication.java - 3: DocumentListener-rajapinnan toteutus.

Tekstin talletus tiedostoon suoritetaan vain jos tiedostonimi on kirjoitettu "oikeilla" säännöillä. Kun tiedostonimi on kirjoitettu oikein, tiedostonimen sisältävän tekstikentän vieressä oleva label ei sisällä tekstiä "INVALID".

Javalla voidaan avata tekstitiedosto kirjoittamista varten siten että ensin luodaan **FileWriter**-luokan olio. Tuo **FileWriter**-olio sitten "syötetään" parametrina kun luodaan **PrintWriter**-luokan olio. **FileWriter**-luokka huolehtii varsinaisesta tiedoston avaamisesta. **PrintWriter**-luokka tarjoaa metodeja tiedostoon kirjoitusta varten.

```
public void actionPerformed( ActionEvent event )
{
   if ( event.getSource()  ==  save_button  &&
        ! file_name_label.getText().contains( "INVALID" ) )
   {
      String file_name  =  file_name_field.getText() ;

      try
      {
         PrintWriter new_text_file  =  new PrintWriter(
                                           new FileWriter( file_name ) ) ;

         writing_area.write( new_text_file ) ;

         new_text_file.close() ;
         save_button.setEnabled( false ) ;
         text_has_to_be_saved_to_file  =  false ;
      }
      catch ( IOException  caught_io_exception )
      {
         System.out.print( "\n\n  File error. Cannot write to \""
                           +  file_name + "\".\n" ) ;
      }
   }
}
```

Tekstialueella oleva teksti voidaan kokonaisuudessaan tallettaa tiedostoon kun kutsutaan **JTextArea**-oliolle metodia **write()**. Tuolle **write()**-metodille annetaan parametrina avattua tekstitiedostoa edustava **PrintWriter**-olio. Javan tiedostonkäsittelyoperaatiot suoritetaan yleensä aina **try-catch** -rakenteen sisällä.

Kun teksti on saatu talletetuksi tiedostoon, Save-nappi "disabloidaan". Tuo nappi aktivoituu heti uudelleen kun tekstialueen tekstiä seuraavan kerran muutetaan.

NewTextFileApplication.java - 4: Tekstin talletuksen tiedostoon hoiteleva metodi.

WindowListener-rajapinnan muodostavista metodeista on tässä tehty "jotain tekeväksi" ainoastaan **windowClosing()**-metodi, jota kutsutaan automaattisesti silloin kun ikkuna suljetaan ikkunan yläkulmassa olevalla sulkunapilla. Tässä sulkeutuminen on estetty siinä tapauksessa että muutettua tekstiä ei ole talletettu tiedostoon.

```
   public void windowClosing( WindowEvent event )
   {
      //  If the text on the writing area has not been saved after
      //  last modification, window closing operation will not be
      //  permitted.

      if ( text_has_to_be_saved_to_file  ==  false )
      {
         setVisible( false ) ;
         System.exit( 0 ) ;
      }
   }

   //  Other methods required by the WindowListener interface:
   public void windowActivated( WindowEvent event ) {}
   public void windowClosed( WindowEvent event ) {}
   public void windowDeactivated( WindowEvent event ) {}
   public void windowDeiconified( WindowEvent event ) {}
   public void windowIconified( WindowEvent event ) {}
   public void windowOpened( WindowEvent event ) {}
}


public class NewTextFileApplication
{
   public static void main( String[] not_in_use )
   {
      NewTextFileFrame  new_text_file_frame  =  new  NewTextFileFrame() ;

      new_text_file_frame.setVisible( true ) ;
   }
}
```

NewTextFileApplication.java - 5. Ohjelman loppuosa.