# LUKU 18

## JAVA-MIDLETIT

Java-midletit ovat kännyköissä suoritettavia Java-ohjelmia.

2006-03-03 Tiedosto luotu.
2006-10-24 Virheellisiä paragraph tagejä korjattu.
2009-10-02 Uudistettu versio ohjelmasta ClockMIDlet.java
2009-10-04 CustomItemDemoMIDlet.java lisätty.

## *GraphicsDemoMIDlet.java -- piirtometodien demonstrointia*

Kun halutaan tehdä midlettiohjelma jossa piirretään jotain näytölle **Graphics**-luokan metodeilla, tulee määritellä erillinen **Canvas**-luokasta johdettu luokka jonne piirtämisen suorittava **paint()**-metodi sijoitetaan. Tässä tapauksessa tuon luokan nimi on **GraphicsDemoCanvas** ja se esitellään myöhemmin tässä ohjelmassa.

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

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class GraphicsDemoMIDlet extends MIDlet
{
   Display             display_of_this_midlet ;
   GraphicsDemoCanvas  canvas_of_this_midlet ;

   public GraphicsDemoMIDlet()
   {
      display_of_this_midlet  =  Display.getDisplay( this ) ;
      canvas_of_this_midlet   =  new  GraphicsDemoCanvas() ;
   }

   protected void destroyApp( boolean conditional_destruction_disabled )
   {

   }

   protected void pauseApp()
   {

   }

   protected void startApp() throws MIDletStateChangeException
   {
      display_of_this_midlet.setCurrent( canvas_of_this_midlet ) ;
   }
}
```

**GraphicsDemoMIDlet.java - 1: Graphics-luokan piirtometodeja demonstroiva midletti.**

Englanninkielen sana *canvas* tarkoittaa piirtoalustaa tai -kangasta johon taidemaalarit maalaavat tauluja. Javassa **Canvas** on luokka josta johdetut luokat sisältävät mahdollisuuden maalata jotain luokan edustamaan olioon. **Canvas**-luokasta johdettuun luokkaan kirjoitetaan **paint()**-metodi, joka sitten suorittaa tarvittavat maalaamiset. **paint()**-metodi kutsuuntuu automaattisesti ohjelman suoriintumisen aikana ja sille tulee parametrina viittaus **Graphics**-olioon, jonka avulla varsinainen piirtäminen tapahtuu.

Tässä asetetaan ensin piirtoväriksi valkoinen ja sitten tuolla värillä täytetään koko kanvaksen alue. Tällä tavalla kaikki näytöllä oleva tulee aluksi ylikirjoitettua.

```
class GraphicsDemoCanvas extends Canvas
{
   int  canvas_width, canvas_height ;

   public GraphicsDemoCanvas()
   {
      canvas_width   =  getWidth() ;
      canvas_height  =  getHeight() ;
   }

   public void paint( Graphics graphics )
   {
      graphics.setColor( 255, 255, 255 ) ;  // white color clears the canvas
      graphics.fillRect( 0, 0, canvas_width - 1, canvas_height - 1 ) ;

      graphics.setColor( 0, 0, 0 ) ;  // black color is used for drawing

      graphics.drawString( "Canvas size is "  +  canvas_width
                        +  " x "  +  canvas_height, 20, 20,
                           Graphics.TOP | Graphics.LEFT ) ;

      //  Drawing a horizontal line into the middle of canvas area.
      graphics.drawLine( 0, canvas_height / 2,
                         canvas_width, canvas_height / 2 ) ;

      graphics.fillRect( 20, 70, 100, 40 ) ;

      graphics.fillArc(  20, 170, 100, 80,  45, 270 ) ;
      graphics.drawArc( 100, 170, 100, 80, 315,  90 ) ;
   }
}
```

**GraphicsDemoMIDlet.java - 2. GraphicsDemoCanvas-luokka.**


Midlettejä suunniteltaessa tulee huomioida että kaikissa puhelimissa näyttö ei välttämättä ole yhtä suuri kuin tässä emulaattorin puhelimessa.

Kaksi alimmaista kuviota on saatu aikaiseksi `fillArc()`- ja `drawArc()`-metodeilla joista ensin mainittu täyttää määritellyn kaarenmuotoisen alueen piirtovärillä ja toinen piirtää pelkän kaaren. Näillä metodeilla saadaan aikaan pallo tai ympyrä, kun piirretään 360 asteen kaari ja määritellään kaaren rajoittavan suorakaiteen molemmat sivut saman kokoisiksi.

**GraphicsDemoMIDlet.java - X. Midlettiä suoritetaan tässä emulaattorissa.**

## *SumMIDlet - midletti joka laskee kahden annetun luvun summan*

Tämä ohjelma on ns. midletti joka kännykässä toimiessaan osaa pyytää käyttäjältä kaksi kokonaislukua ja laskea ne yhteen.

Midlettejä ohjelmoitaessa tarvitaan yleensä nämä **import**-komennot ohjelman alkuun.

Midletit rakennetaan siten että ne johdetaan standardi-Javaluokasta nimeltä **MIDlet**. Tämä midletti toteuttaa **CommandListener**- ja **ItemStateListener**-rajapinnat, mikä tarkoittaa että midletissä on metodit nimeltä **commanAction()** ja **itemStateChanged()**.

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

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class SumMIDlet extends  MIDlet
                       implements CommandListener, ItemStateListener
{
   private Display display_of_this_midlet ;

   private TextField first_integer_text_field ;
   private TextField second_integer_text_field ;
   private TextField result_text_field ;

   private Form  form_of_this_midlet  =  new  Form( "SumMIDlet" ) ;

   private Command exit_command  =  new  Command( "EXIT", Command.EXIT, 1 ) ;
```

Midlettejä rakennettaessa käytetään hyväksi lukuisia Javan standardiluokkia. **Display** on luokka joka edustaa kännykän näyttöä. Yleensä midleteissä on aina määritelty **Display** tähän tapaan.

Tässä midletissä näytöllä esitettävät kokonaisluvut näkyvät **TextField**-olioiden sisällä. **TextField**-oliot ovat tekstikenttiä jotka kiinnitetään **Form**-olioon. **Form**-olio sitten kiinnitetään midletin **Display**-olioon. Nämä kiinnittämiset tehdään konstruktorissa ja **startApp()**-metodissa.

**SumMIDlet.java - 1: Midlettiohjelma jolla voidaan laskea kahden kokonaisluvun summa.**

Konstruktori on metodi jolla luokan olio luodaan eli rakennetaan. Midlettioliohan luodaan automaattisesti kännykässä silloin kun midletti käynnistetään.

Midlettiluokan konstruktorissa rakennetaan näytöllä käyttäjälle esitettävä näkymä. **TextField**-olioiden tarkoitus on sisältää näytettävät kokonaisluvut. Jokaiseen **TextField**-olioon liittyy ns. label-teksi jota näytetään varsinaisen tekstikentän edellä. Parametreina **TextField**-luokan konstuktorille annetaan tuo label-teksti, tekstikentän alkuteksti, tekstikentän leveys merkkeinä sekä tekstikentän rajoitteet. Nuo rajoitteet kuvataan vakioiden avulla. Esimerkiksi vakio **TextField.NUMERIC** tarkoitaa että tekstikenttään voidaan kännykän mäppimistöltä syöttää vain kokonaislukuja.

```
public SumMIDlet()
{
   display_of_this_midlet  =  Display.getDisplay( this ) ;

   first_integer_text_field   = new TextField( "First integer: ",
                                               "", 8, TextField.NUMERIC ) ;

   first_integer_text_field.setLayout( Item.LAYOUT_CENTER ) ;

   second_integer_text_field  = new TextField( "Second integer:",
                                               "", 8, TextField.NUMERIC ) ;
   result_text_field     = new TextField( "Calculated sum:",
                                          "0", 8, TextField.NUMERIC |
                                                  TextField.UNEDITABLE ) ;

   form_of_this_midlet.append( first_integer_text_field ) ;
   form_of_this_midlet.append( second_integer_text_field ) ;
   form_of_this_midlet.append( result_text_field ) ;

   form_of_this_midlet.setItemStateListener( this ) ;

   form_of_this_midlet.addCommand( exit_command ) ;
   form_of_this_midlet.setCommandListener( this ) ;
}
```

Luodut tekstikenttäoliot kiinnitetään (lisätään) tässä aiemmin luotuun **Form**-olioon. **Form**-oliot voivat sisältää tekstikenttiä ja muita näyttökomponenttiolioita (itemejä). **Form**-oliolle asetetaan "tämä" luokka kuuntelijaksi. Käytännössä tämä tarkoittaa että tässä luokassa olevaa **itemStateChanged()**-metodia kutsutaan automaattisesti silloin kun jonkin **Form**-oliossa olevan tekstikentän tekstiä muutetaan kännykän näppäimistön avulla.

**Form**-olioon lisätään vielä aiemmin luotu komento eli **Command**-olio. Lisäksi asetetaan tuolle oliolle kuuntelija siten että tässä luokassa olevaa **commandAction()**-metodia kutsutaan automaattisesti silloin kun komento annetaan.

**SumMIDlet.java - 2: SumMIDlet-luokan konstruktori.**

Midletteihin tulee kirjottaa tämän nimiset metodit vaikka niille ei ohjelmassa olisi varsinaisia toiminteita suunniteltukaan.

Midlettien ajoympäristöstä kutsutaan metodeita **`destroyApp()`** ja **`pauseApp()`** automaattisesti joissain tilanteissa jos/kun ajoympäristö haluaa jostain syystä tuhota midletin tai väliaikaisesti pysäyttää sen.

Nämä metodit on midletissä oltava mutta niille ei tarvitse ainakaan yksinkertaisissa oppikirjaohjelmissa antaa mitään tehtävää.

```
protected void destroyApp( boolean conditional_destruction_disabled )
{
}

protected void pauseApp()
{
}

protected void startApp() throws MIDletStateChangeException
{
   display_of_this_midlet.setCurrent( form_of_this_midlet ) ;
}

public void commandAction( Command     given_command,
                           Displayable display_content )
{
   if ( given_command  ==  exit_command )
   {
      destroyApp( false ) ;
      notifyDestroyed() ;
   }
}
```

Metodia **`startApp()`** kutsutaan silloin kun midletti ajoympäristössään (siis kännykässä) käynnistetään. Ennen **`startApp()`** -metodin kutsua on midlettiolio jo luotu ja sen konstruktori suoritettu.

Tässä kiinnitetään komento midletin näyttöön aiemmin rakennettu formi.

Metodia nimeltä **`commandAction()`** kutsutaan automaattisesti silloin kun käyttäjä antaa komennon. Tässä midletissä ainoa annettava komento on EXIT-komento.

**SumMIDlet.java - 3: Midletin "pakolliset" metodit ja commandAction()-metodi.**

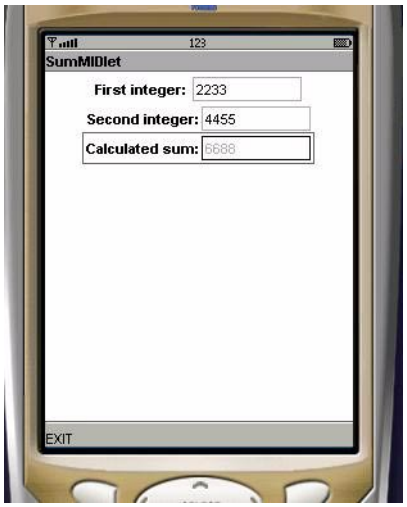
Tätä **itemStateChanged()**-metodia kutsutaan automaattisesti silloin kun jossakin **Form**-olion tekstikentässä tehdään muutos. Olisi mahdollista ottaa selville missä tekstikentässä tuo muutos tehtiin, sillä **item_which_changed_state** viittaa juuri tuohon **TextField**-olioon. (**TextField** on **Item**-luokan alaluokka.) Tässä ei kuitenkaan oteta selville, missä tekstikentässä tekstiä muutettiin, vaan aina lasketaan uudelleen tekstikentissä olevien lukujen summa.

```
public void itemStateChanged( Item item_which_changed_state )
{
   String first_integer_text   =  first_integer_text_field.getString() ;
   String second_integer_text  =  second_integer_text_field.getString() ;

   if ( first_integer_text.length() ==  0 )
   {
      first_integer_text  =  "0" ;
   }

   if ( second_integer_text.length()  ==  0 )
   {
      second_integer_text  =  "0" ;
   }

   int first_integer   =  Integer.parseInt( first_integer_text ) ;
   int second_integer  =  Integer.parseInt( second_integer_text ) ;

   int sum_of_two_integers  =  first_integer  +  second_integer ;

   String  sum_text  =  "" + sum_of_two_integers ;

   result_text_field.setString( sum_text ) ;
}
}
```

**TextField**-luokassa on käytössä metodit **getString()** ja **setString()**, joilla tekstikentässä oleva teksti voidaan lukea ja kirjoittaa. Lopuksi asetetaan **setString()**-metodin avulla yhteenlaskun tules tuloskenttään.

Jos tekstikentät olivat tyhjiä, eli niihin ei oltu vielä kokonaislukuja kirjoitettu, tekstikenttien arvoksi oletetaan luku nolla. Arvoksi tulee asettaa "0", eli nollan sisältävä stringi, jotta **Integer.parseInt()**-metodilla tehtävä muunnos onnistuu varmasti. **Integer**-luokan staattinen **parseInt()**-metodi muuntaa annetun stringin **int**-arvoksi. Jos annettu stringi ei ole muunnettavissa **int**-arvoksi, heitetään poikkeus.

**SumMIDlet.java - 4. Tekstikentissä tehtäviin muutoksiin reagoiva metodi.**




Yhteenlaskun tuloksen sisältävä viimeinen tekstikenttä on sellainen että sen arvoa ei midletin käyttäjä voi muuttaa. Tuommoisen tekstikentän teksti näkyy himmeämpänä kuin muiden tekstikenttien tekstit. Tämän tekstikenttäolion luonnissa on käytetty vakiota `TextField.UNEDITABLE` jolla saadaan aikaan ei-editoitava tekstikenttä.

**SumMIDlet.java - X. Midletillä on tässä laskettu yhteen luvut 2233 ja 4455.**

## *KeyCodesMIDlet.java -- näppäinpainalluksiin heti reagoiva midletti*

Tässä esimerkissä **KeyCodesMIDlet** -luokka ei tee pajon mitään, vaan siellä luodaan olio jäljempänä esiteltävästä **KeyCodesCanvas** -luokasta. Tämä **KeyCodesCanvas** -olio on sitten se joka "hoitaa hommat".

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

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class KeyCodesMIDlet extends MIDlet
{
   private  Display display_of_this_midlet ;
   private  KeyCodesCanvas  key_codes_canvas ;

   public KeyCodesMIDlet()
   {
      display_of_this_midlet  =  Display.getDisplay( this ) ;
      key_codes_canvas        =  new KeyCodesCanvas() ;
   }

   protected void destroyApp( boolean unconditional_destruction_required )
   {
   }

   protected void pauseApp()
   {
   }

   protected void startApp() throws MIDletStateChangeException
   {
      display_of_this_midlet.setCurrent( key_codes_canvas ) ;
   }
}
```

Tässä aiemmin luotuun **Display**-olioon kiinnitetään **key_codes_canvas** -olio. Käytännössä siis näytössä näytetään sitä mitä **KeyCodesCanvas** -luokassa määrätään.

**KeyCodesMIDlet.java - 1: Midlettiluokan määrittely.**

**KeyCodesCanvas** johdetaan tässä standardiluokasta nimeltä **Canvas**. Kun luokka toteuttaa **CommandListener** -rajapinnan, se tarkoittaa että se sisältää **commandAction()** -metodin joka reagoi annettuihin komentoihin. Komentoihin voivat reagoida erityyppiset oliot. Aiemmassa esimerkissä **SumMIDlet.java** **CommandListener** -rajapinnan toteutti luokasta **MIDlet** johdettu luokka.

```
class KeyCodesCanvas extends Canvas
                     implements CommandListener
{
   String  key_code_as_string  =  "No keys pressed" ;

   int  key_code_numerical  =  0 ;

   int  game_action_code  =  0 ;

   Command select_hexadecimal_printing  =  new Command( "Hexadecimal",
                                                        Command.SCREEN, 1 ) ;
   Command select_decimal_printing      =  new Command( "Decimal",
                                                        Command.SCREEN, 1 ) ;
   Command select_binary_printing       =  new Command( "Binary",
                                                        Command.SCREEN, 1 ) ;

   Command last_given_command  =  select_decimal_printing ;

   public KeyCodesCanvas()
   {
      addCommand( select_hexadecimal_printing ) ;
      addCommand( select_decimal_printing ) ;
      addCommand( select_binary_printing ) ;

      setCommandListener( this ) ;
   }
```

Aiemmin määritellyt komennot kiinnitetään tässä tähän **KeyCodesCanvas** -olioon. Lisäksi määritellään että tämä olio toimii komentojen kuuntelijana.

**KeyCodesMIDlet.java - 2: KeyCodesCanvas -luokan datajäsenet ja konstruktori.**

**Canvas**-luokasta johdetuissa luokissa on yleensä **paint()** -metodi, jolla on suurinpiirtein sama tehtävä kuin apleteissakin olevilla **paint()** -metodeilla.

Tässä oletetaan ensin että tulostuksessa numerot näytetään desimaalisina, mutta jos valittuna on heksadesimaalinen tai binaarinen numeroiden näyttötapa, tässä luodut **String**-oliot hylätään ja stringiviittaajiin kiinnitetään uudet oliot.

```
protected void paint( Graphics graphics )
{
   graphics.setColor( 255, 255, 255 ) ;
   graphics.fillRect( 0, 0, getWidth() - 1, getHeight() - 1 ) ;

   graphics.setColor( 0, 0, 0 ) ;

   String game_action_code_to_print  =  "" + game_action_code ;
   String key_code_numerical_to_print  =  "" + key_code_numerical ;

   if ( last_given_command  ==  select_hexadecimal_printing )
   {
      game_action_code_to_print  =
               Integer.toHexString( game_action_code ) + "H" ;
      key_code_numerical_to_print  =
               Integer.toHexString( key_code_numerical ) + "H" ;
   }
   else if ( last_given_command  ==  select_binary_printing )
   {
      game_action_code_to_print  =
               Integer.toBinaryString( game_action_code ) + "B" ;
      key_code_numerical_to_print  =
               Integer.toBinaryString( key_code_numerical ) + "B" ;
   }

   graphics.drawString( "game_action_code:    "  +  game_action_code_to_print,
                        10, 20,
                        Graphics.TOP | Graphics.LEFT ) ;

  graphics.drawString( "key_code_numerical: "  +  key_code_numerical_to_print,
                        10, 40,
                        Graphics.TOP | Graphics.LEFT ) ;

   graphics.drawString( "key_code_as_string:  "  +  key_code_as_string,
                        10, 60,
                        Graphics.TOP | Graphics.LEFT ) ;
}
```

**Graphics** -luokan metodit ovat midleteissä samantapaisia kuin apleteissa, mutta eroja löytyy. Metodien kuvaukset kannattaa tutkia elektronisesta dokumentaatiosta.

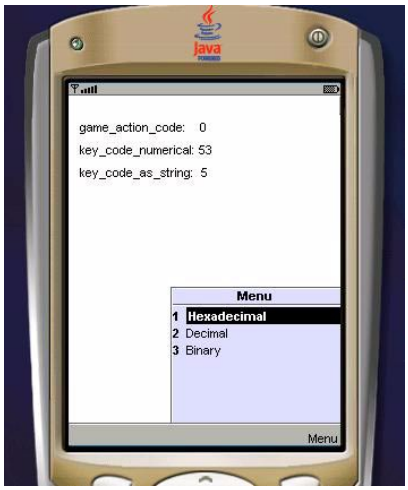
**KeyCodesMIDlet.java - 3: KeyCodesCanvas -luokan metodi paint().**

**`keyPressed()`** -metodia kutsutaan automaattisesti silloin kun jotain kännykän näppäintä painetaan, ja metodi saa silloin parametrinä kyseisen näppäimen numerokoodin. Tässä näppäimen numerokoodi muutetaan sekä ns. Game Action -koodiksi että stringimuotoiseksi näppäimen nimeksi.

```
   public void keyPressed( int key_code )
   {
      game_action_code = getGameAction( key_code ) ;

      key_code_numerical  =  key_code ;

      key_code_as_string  =  getKeyName( key_code ) ;

      repaint() ;
   }

   public void commandAction( Command     given_command,
                              Displayable display_content )
   {
      last_given_command  =  given_command ;
   }
}
```

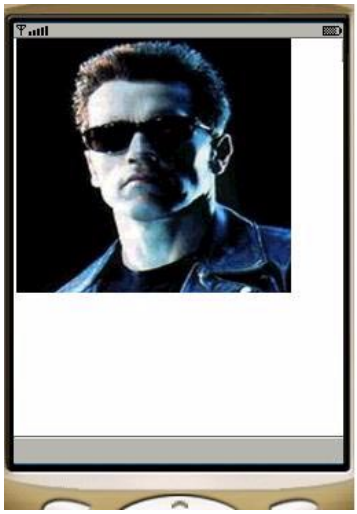
**`last_given_command`** on tämän luokan **`Command`**-tyyppinen datajäsen johon talletetaan tieto siitä mikä komento on annettu viimeksi. Metodi **`paint()`** tutkii "muuttujan" **`last_given_command`** arvon ja näyttää näppäinkoodit joko desimaalisina, heksadesimaalisina tai binaarisina riippuen tämän muuttujan arvosta.

**KeyCodesMIDlet.java - 4. KeyCodesCanvas -luokan metodit keyPressed() ja commandAction().**




**KeyCodesMIDlet.java - X. Tässä on ensin painettu näppäintä 5 ja sitten otettu menu esille.**

## *KuvashowMIDlet.java – kuvia näyttävä midlettiohjelma*



Tässä esiteltävä ohjelma on midletti joka näyttää näytöllä **.png**-tiedostoina olevia kuvia. Kuvia voidaan kelata eteenpäin nuoli oikealle - ja nuoli vasemmalle -näppäimillä. Tässä kuvia on kelattu etenpäin Arnold Schwarzeneggerin kuvan kohdalle.

**KuvashowMIDlet.java - X. Kuvia näyttävä midletti toiminnassa.**

Tässä ohjelmassa varsinainen midlettiluokka, siis **`MIDlet`**-luokasta johdettu luokka, on hyvin lyhyt. Siinä luodaan olio **`KuvashowCanvas`**-luokasta ja pannaan tuo olio näyttöön.

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

import java.io.* ;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class KuvashowMIDlet extends MIDlet
{
   Display  taman_midletin_naytto  =  Display.getDisplay( this ) ;
   KuvashowCanvas  kuvashowcanvas  =  new KuvashowCanvas() ;

   protected void destroyApp( boolean unconditional_destruction_required )
   {
   }

   protected void pauseApp()
   {
   }

   protected void startApp() throws MIDletStateChangeException
   {
      taman_midletin_naytto.setCurrent( kuvashowcanvas ) ;
   }
}
```

**KuvashowMIDlet.java - 1: Kuvia näyttävä midletti.**

Kuvatiedostoihin viitataan luonnollisesti niiden tiedostonimillä. Sun Java Wireless Toolkitia käytettäessä kuvatiedostot pannaan projektin **res**-kansioon. Tässä kuvatiedostojen nimet on pantu alustettuun `String[]`-tyypin taulukkoon. Heti perään luodaan samankokoinen tyyppiä `Image[]` oleva taulukko johon konstruktorissa luotavat `Image`-oliot talletetaan.

```
class KuvashowCanvas extends Canvas
{
   int  nayton_leveys   =  getWidth() ;
   int  nayton_korkeus  =  getHeight() ;

   int  naytettavan_kuvan_indeksi  =  0 ;

   String[]  kuvatiedostot  =  { "/pengbrew_160x160.png",
                                 "/risto_kimari_2000.png",
                                 "/marilyn_by_warhol.png",
                                 "/risto_kimari_blueface.png",
                                 "/tarja_halonen_2000.png",
                                 "/nicole_kidman.png",
                                 "/terminator2.png",
                                 "/kate_winslet.png" } ;

   Image[]   naytettavat_kuvat  =  new  Image[ kuvatiedostot.length ] ;

   public KuvashowCanvas()
   {
      for ( int kuvan_indeksi  =  0 ;
                kuvan_indeksi  <  kuvatiedostot.length ;
                kuvan_indeksi  ++ )
      {
         try
         {
            naytettavat_kuvat[ kuvan_indeksi ]  =
                   Image.createImage( kuvatiedostot[ kuvan_indeksi ] ) ;
         }
         catch ( IOException  kasittelematon_poikkeus )
         {
            System.out.print( "\n Kuvaolion luontiongelma ....  "
                           +  kuvatiedostot[ kuvan_indeksi ] ) ;
         }
      }
   }
```

Jokaisen annetun kuvatiedoston perusteella luodaan tässä `Image`-olio. Koska on mahdollista että metodi `createImage()` heittää poikkeuksen siinä tapauksessa että kuvatiedoston lukeminen jostain syystä epäonnistuu, täytyy kuvaoliot luoda `try-catch` -rakenteen sisällä.

**KuvashowMIDlet.java - 2: KuvashowCanvas-luokan alkuosa.**

Yksi **Image**-tyyppinen kuvaolio taulukosta piirretään näytölle **Graphics**-luokan **drawImage()**-metodilla.

```
    protected void paint( Graphics graphics )
    {
       graphics.setColor( 255, 255, 255 ) ;  //  white
       graphics.fillRect( 0, 0, getWidth() - 1, getHeight() - 1 ) ;


       graphics.drawImage( naytettavat_kuvat[ naytettavan_kuvan_indeksi ],
                           2, 0,
                           Graphics.TOP | Graphics.LEFT ) ;
    }

    public void keyPressed( int nappainkoodi )
    {
       int game_action_code = getGameAction( nappainkoodi ) ;

       switch ( game_action_code )
       {
       case UP:
       case LEFT:

          if ( naytettavan_kuvan_indeksi  >  0 )
          {
             naytettavan_kuvan_indeksi  --  ;
          }
          else
          {
             naytettavan_kuvan_indeksi  =  naytettavat_kuvat.length  -  1 ;
          }

          break ;

       case DOWN:
       case RIGHT:

          if ( naytettavan_kuvan_indeksi  <  ( naytettavat_kuvat.length  -  1 ))
          {
             naytettavan_kuvan_indeksi  ++  ;
          }
          else
          {
             naytettavan_kuvan_indeksi  =  0 ;
          }

          break ;
       }

       repaint() ;
    }
}
```

Muuttujan (datakentän) **naytettavan_kuvan_indeksi** arvo määrää, minkä kuvan metodi **paint()** piirtää näytölle. Tässä kyseisen muuttujan arvoa kasvatetaan yhdellä tai se asetetaan nollaksi siinä tapauksessa jos se on jo saavuttanut maksimiarvonsa.

**KuvashowMIDlet.java - 3. KuvashowCanvas-luokan ja koko ohjelman loppuosa.**

## *LiikkuvaPalloMIDlet.java – List-luokkaa hyödyntävä ohjelma*

Tässä ohjelmassa tehdään **List**-luokan avulla valintamenu josta ohjelman käyttäjä voi valita värin liikuteltavalle pallolle. **List**-olio luodaan **List.IMPLICIT**-parametrilla, mika tarkoittaa että listasta voidaan siirtyä heti valinnan jälkeen pallon liikuttelutilaan.

**Command.BACK**-määrittelyllä komennosta tulee sellainen että se kiinnittyy siihen Soft Key -näppäimeen joka normaalisti toimii laitteen BACK-näppäimenä.

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

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class LiikkuvaPalloMIDlet extends MIDlet
                                 implements  CommandListener
{
   private  Display              taman_midletin_naytto ;
   private  LiikkuvaPalloCanvas  liikkuvan_pallon_canvas ;

   String[] pallon_varivaihtoehdot  =  { "Punainen pallo",
                                         "Vihrea pallo",
                                         "Sininen pallo",
                                         "Musta pallo",
                                         "Tummanpunainen pallo",
                                         "Tummanvihrea pallo",
                                         "Tummansininen pallo" } ;

   List  varin_valintamenu  =  new  List( "Valitse pallon vari",
                                          List.IMPLICIT,
                                          pallon_varivaihtoehdot,
                                          null ) ;

   Command  varinvaihtokomento  =  new  Command( "Muu vari",
                                                 Command.SCREEN, 5 ) ;

   Command  harmahtavanasetuskomento  =  new Command( "Harmahtava",
                                                      Command.BACK, 5 ) ;

   public LiikkuvaPalloMIDlet()
   {
      taman_midletin_naytto     =  Display.getDisplay( this ) ;
      liikkuvan_pallon_canvas   =  new LiikkuvaPalloCanvas( this ) ;

      liikkuvan_pallon_canvas.addCommand( varinvaihtokomento ) ;
      liikkuvan_pallon_canvas.addCommand( harmahtavanasetuskomento ) ;
      liikkuvan_pallon_canvas.setCommandListener( this ) ;

      varin_valintamenu.setCommandListener( this ) ;
   }
```

**LiikkuvaPalloMIDlet.java - 1: Esimerkki List-luokkaa hyödyntävästä ohjelmasta.**

```
protected void destroyApp( boolean unconditional_destruction_required )
{
}

protected void pauseApp()
{
}

protected void startApp() throws MIDletStateChangeException
{
   taman_midletin_naytto.setCurrent( varin_valintamenu ) ;
}

public void commandAction( Command     annettu_komento,
                           Displayable display_content )
{
   if ( annettu_komento  ==  varinvaihtokomento )
   {
      taman_midletin_naytto.setCurrent( varin_valintamenu ) ;
   }
   else if ( annettu_komento  ==  harmahtavanasetuskomento )
   {
      liikkuvan_pallon_canvas.muuta_pallon_vari( "Harmahtava pallo" ) ;

      liikkuvan_pallon_canvas.repaint() ;
   }
   else if ( annettu_komento  ==  List.SELECT_COMMAND )
   {
      int valitun_varin_indeksi  =
                           varin_valintamenu.getSelectedIndex() ;

      String varin_nimi_listasta  =
                  varin_valintamenu.getString( valitun_varin_indeksi ) ;

      liikkuvan_pallon_canvas.muuta_pallon_vari( varin_nimi_listasta ) ;

      taman_midletin_naytto.setCurrent( liikkuvan_pallon_canvas ) ;
   }
}
}
```

Alussa näytön sisältönä on **List**-olio eli värin valintamenu.

Näytön sisällöksi pannaan värin valintamenu silloin kun käyttäjä on antanut Muu väri -komennon. **Display**-luokan **setCurrent()**-metodilla voidaan siis näytön sisällöksi muuttaa erilaisia olioita.

**List.SELECT_COMMAND** on eräänlainen automaattisesti generoitu komento joka tulee käsittelyyn kun **List.IMPLICIT**-tyyppiseltä listalta tehdään valinta. Tätä komentoa käsiteltäessä otetaan käyttöön valittu väri ja asetetaan ohjelma "pelitilaan" pistämällä näytön sisällöksi liikkuvan pallon canvas.

**LiikkuvaPalloMIDlet.java - 2: Varsinaisen midlettiluokan loppuosa.**

RGB-väri koostuu kolmesta värikomponentista Red, Green ja Blue. Väri voidaan määrittää heksadesimaaliluvulla joka on muotoa 0x00RRGGBB. Jokaiselle värikomponentille on siis numeroarvot väliltä 0 ... 255.

```
class LiikkuvaPalloCanvas extends Canvas
{
   MIDlet  canvasin_luonut_midletti ;

   int pallon_paikka_x, pallon_paikka_y ;

   int  rgb_varin_maarittely   =    0x00000000 ;

   public LiikkuvaPalloCanvas( MIDlet   annettu_midletti )
   {
      canvasin_luonut_midletti  =  annettu_midletti ;

      pallon_paikka_x  =  getWidth()  / 2  -  20 ;
      pallon_paikka_y  =  getHeight() / 2  -  20 ;
   }

   public void muuta_pallon_vari( String annettu_varin_nimi )
   {
      if ( annettu_varin_nimi.equals( "Punainen pallo" ) )
      {
         rgb_varin_maarittely  =  0x00FF0000 ;
      }
      else if ( annettu_varin_nimi.equals( "Vihrea pallo" ) )
      {
         rgb_varin_maarittely  =  0x0000FF00 ;
      }
      else if ( annettu_varin_nimi.equals( "Sininen pallo" ) )
      {
         rgb_varin_maarittely  =  0x000000FF ;
      }
      else if ( annettu_varin_nimi.equals( "Musta pallo" ) )
      {
         rgb_varin_maarittely  =  0x00000000 ;
      }
      else if ( annettu_varin_nimi.equals( "Tummanpunainen pallo" ) )
      {
         rgb_varin_maarittely  =  0x007F0000 ;
      }
      else if ( annettu_varin_nimi.equals( "Tummanvihrea pallo" ) )
      {
         rgb_varin_maarittely  =  0x00007F00 ;
      }
      else if ( annettu_varin_nimi.equals( "Tummansininen pallo" ) )
      {
         rgb_varin_maarittely  =  0x0000007F ;
      }
      else if ( annettu_varin_nimi.equals( "Harmahtava pallo" ) )
      {
         rgb_varin_maarittely  =  0x007F7F7F ;
      }
   }
```

**LiikkuvaPalloMIDlet.java - 3: LiikkuvaPalloCanvas-luokan alkuosa.**

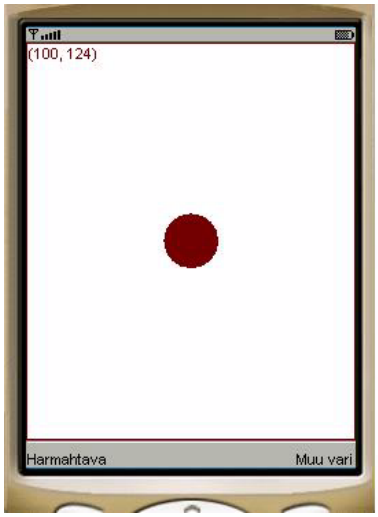
**Graphics**-luokan **drawRect()**-metodilla piirretään canvaksen ympärille raamit. **getWidth()**- ja **getHeight()**-metodeilla otetaan selville canvaksen koko.

Pallo piirretään tässä **fillArc()**-metodilla, joka täyttää suorakaiteen sisällä olevaa soikiota annettujen kulmamäärittelyjen mukaisesti. Pallo piirretään 40 x 40 pikselin kokoisen suorakaiteen (neliön) sisään. Kun pallon kaareksi määritetään 360 astetta, syntyy kokonainen pallo.

```
   protected void paint( Graphics graphics )
   {
      // Aluksi koko naytto tyhjennetaan maalaamalla se valkoiseksi.

      graphics.setColor( 255, 255, 255 ) ;  //  Valkoinen vari
      graphics.fillRect( 0, 0, getWidth() - 1, getHeight() - 1 ) ;


      graphics.setColor( rgb_varin_maarittely ) ;

      graphics.drawRect( 0, 0, getWidth() - 1, getHeight() - 1 ) ;

      graphics.fillArc( pallon_paikka_x, pallon_paikka_y, 40, 40, 0, 360 ) ;

      graphics.drawString(  "(" + pallon_paikka_x
                          + ", "  +  pallon_paikka_y  +  ")",
                            2, 0, Graphics.TOP | Graphics.LEFT ) ;
   }

   public void keyPressed( int nappaimen_koodi )
   {
      int game_action_code = getGameAction( nappaimen_koodi ) ;

      switch ( game_action_code )
      {
      case UP:
          pallon_paikka_y  -=  3 ;
          break;

      case DOWN:
          pallon_paikka_y  +=  3 ;
          break;

      case RIGHT:
          pallon_paikka_x  +=  3 ;
          break;

      case LEFT:
          pallon_paikka_x  -=  3 ;
          break;
      }

      repaint() ;
   }
}
```

Metodi nimeltä **keyPressed()** reagoi "pelitilassa" nuolinäppäinten painalluksiin. Pelitilalla tarkoitetaan tilaa jolloin **LiikkuvaPalloCanvas**-olio on näytön sisältönä. Pallon paikan koordinaatteja muutetaan sopivasti sen mukaan mitä nuolinäppäintä painettiin.

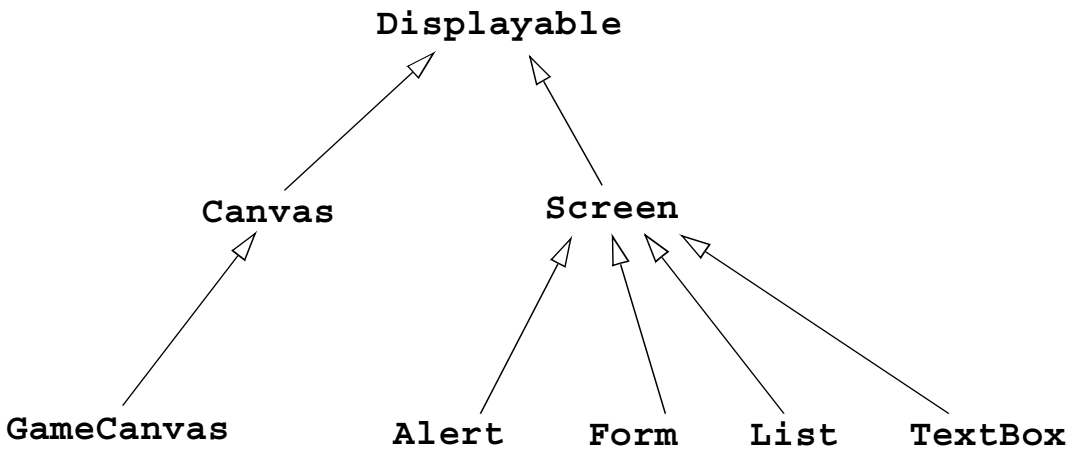
**LiikkuvaPalloMIDlet.java - 4. LiikkuvaPalloCanvas-luokan ja koko ohjelman loppuosa.**


Vasemmalla on esitetty värinvalintavalikko joka tulee midletin käynnistyessä esiin. Kyseisen valikon saa näkyviin myös myöhemmin kun tekee Muu väri -valinnan. Oikealla on midletin varsinainen "pelinäkymä". Ruudulla olevaa palloa voi liikutella nuolinäppäimillä. Vasemmassa yläkulmassa näkyvät pallon koordinaatit. Harmahtava-valinnalla pallon värin voi muuttaa nopeasti.

**LiikkuvaPalloMIDlet.java - X. "Pelin" alkunäkymä ja pelinäkymä.**

Tällä sivulla on kuvattu eräiden tärkeiden midlettien rakentamisessa tarvittavien luokkien suhteita, eli luokkien välistä periytymistä. Esimerkiksi `Dispalyable`-luokan alaluokkien oliot ovat sellaisia jotka voidaan kiinnittää näyttöön. `Display`-luokan `setCurrent()`-metodilla.




`Item` on toinen tärkeä luokka. `Item`-luokan alaluokkien olioita voidaan kiinnittää `Form`-olioon ja näin rakentaa kännykän näytölle pienemmistä palasista koostuva näkymä. Tutki dokumentaatiota ja piirrä `Item`-luokan alaluokat samaan tapaan kuin yllä on tehty.

## *SatunnaislukujaMIDlet.java – luokkien Random ja ChoiceGroup käyttöä*



**double**-tyyppisen satunnaisluvun generointi on tässä valittuna. Lukualueen valinta ei vaikuta **double**-tyyppisen satunnaisluvun suuruuteen. **double**-tyypin valinta kuuluu **ChoiceGroup**-olioon jossa on tehtävissä vain yksi valinta.

**SatunnaislukujaMIDlet.java - X. Ohjelman canvas ja ChoiceGroup-oliot sisältävä formi.**

Midlettiluokka on jälleen lyhyt koska kaikki toiminnot tapahtuvat **Canvas**-luokasta johdetussa luokassa. Tällainen **import**-komento tarvitaan ohjelman alussa koska standardiluokka **Random** sijaitsee paketissa **java.util**.

```
//  SatunnaislukujaMIDlet.java

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Random ;

public class SatunnaislukujaMIDlet extends MIDlet
{
   Display  taman_midletin_naytto  =  Display.getDisplay( this ) ;
   SatunnaislukuCanvas  satunnaislukucanvas  =
                        new SatunnaislukuCanvas( taman_midletin_naytto ) ;

   protected void destroyApp( boolean unconditional_destruction_required )
   {
   }

   protected void pauseApp()
   {
   }

   protected void startApp() throws MIDletStateChangeException
   {
      taman_midletin_naytto.setCurrent( satunnaislukucanvas ) ;
   }
}
```

**SatunnaislukujaMIDlet.java - 1: Satunnaislukuja generoiva midletti.**

**ChoiceGroup**-olio edustaan **Form**-olioon kiinnitettävää näyttökomponenttia jolla voidaan suorittaa valintoja. Tässä luodaan kolme valintamahdollisuutta sisältävä **ChoiceGroup**-olio. Koska valinnan tyyppi on **Choice.EXCLUSIVE** vain yksi kolmesta mahdollisuudesta voi olla kerrallaan valittuna.

Toinen **ChoiceGroup**-olio sisältää vain yhden valintamahdollisuuden. Kyseisen valinnan tyyppi on **Choice.MULTIPLE**.

```
class SatunnaislukuCanvas extends Canvas
                          implements CommandListener
{
   Display  taman_midletin_naytto ;

   int  arvottu_satunnaisluku_int  =  0 ;
   double  arvottu_satunnaisluku_double  =  0 ;

   Form  valinnat_form  =  new  Form( "Valitse lukualue yms." ) ;

   String[]  lukualueet  =  { "0 ... 9", "0 ... 99", "0 ... 999" } ;

   ChoiceGroup  lukualueen_valinta  =  new  ChoiceGroup( "LUKUALUE (int):",
                                                         Choice.EXCLUSIVE,
                                                         lukualueet, null ) ;

   String[]  double_valinnan_teksti  =  { "Näytä vain double-arvo" } ;

   ChoiceGroup  double_valinta  =  new ChoiceGroup( "double-tyyppi:",
                                                    Choice.MULTIPLE,
                                                    double_valinnan_teksti,
                                                    null ) ;

   Command  komento_valintojen_suorittamiseksi  =
                       new  Command( "Valinnat", Command.SCREEN, 1 ) ;

   Command  komento_valinnoista_poistumiseen  =
                       new  Command( "Poistu", Command.SCREEN, 1 ) ;

   public SatunnaislukuCanvas( Display annettu_naytto )
   {
      taman_midletin_naytto  =  annettu_naytto ;

      valinnat_form.append( lukualueen_valinta ) ;
      valinnat_form.append( double_valinta ) ;
      valinnat_form.addCommand( komento_valinnoista_poistumiseen ) ;
      valinnat_form.setCommandListener( this ) ;

      addCommand( komento_valintojen_suorittamiseksi ) ;
      setCommandListener( this ) ;
   }
```

Komento jolla siirrytään valintoihin liitetään tähän canvasluokkaan, mutta komento jolla poistutaan valinnoista liitetään formiin.

**SatunnaislukujaMIDlet.java - 2: SatunnaislukuCanvas-luokan datakentät ja konstruktori.**

Soft Key -näppäimiin kytkenttyjä komentoja antamalla saadaan valintojentekoformi haluttaessa näyttöön tai otettua pois näytöltä.

Midletti generoi satunnaisluvun kun painetaan jotain numeronäppäintä. Aluksi tutkitaan onko valittu **double**-tyypin satunnaisluvun generointi. Jos **double**-valintaa ei ole tehty, generoidaan **int**-tyypin satunnaisluku.

```
public void commandAction( Command     annettu_komento,
                           Displayable nayton_nykyinen_sisalto )
{
   if ( annettu_komento  ==  komento_valintojen_suorittamiseksi )
   {
      taman_midletin_naytto.setCurrent( valinnat_form ) ;
   }
   else if ( annettu_komento  ==  komento_valinnoista_poistumiseen )
   {
      taman_midletin_naytto.setCurrent( this ) ;
   }
}

public void keyPressed( int nappainkoodi )
{
   if ( nappainkoodi  >=  '0'  &&  nappainkoodi  <=  '9' )
   {
      Random  satunnaislukugeneraattori  =  new  Random() ;

      if ( double_valinta.isSelected( 0 ) )
      {
         arvottu_satunnaisluku_double  =
                     satunnaislukugeneraattori.nextDouble() ;
      }
      else if ( lukualueen_valinta.getSelectedIndex()  ==  0 )
      {
         arvottu_satunnaisluku_int  =
                    satunnaislukugeneraattori.nextInt( 10 ) ;
      }
      else if ( lukualueen_valinta.getSelectedIndex()  ==  1 )
      {
         arvottu_satunnaisluku_int  =
                    satunnaislukugeneraattori.nextInt( 100 ) ;
      }
      else if ( lukualueen_valinta.getSelectedIndex()  ==  2 )
      {
         arvottu_satunnaisluku_int  =
                    satunnaislukugeneraattori.nextInt( 1000 ) ;
      }
   }

   repaint() ;
}
```

Satunnaislukuja saadaan generoitua **nextDouble()**- ja **nextInt()**-metodeilla sen jälkeen kun on ensin luotu **Random**-tyyppinen satunnaislukugeneraattori. **nextDouble()** palauttaa satunnaisen **double**-arvon joka on suurempi tai yhtäsuuri kuin nolla ja pienempi kuin yksi.

**SatunnaislukujaMIDlet.java - 3: commandAction()- ja keyPressed()-metodit.**

**paint()**-metodissa tulostetaan **keyPressed()**-metodissa generoitu satunnaisluku joka on talletettu joko datakenttään **arvottu_satunnaisluku_int** tai datakenttään **arvottu_satunnaisluku_double** sen mukaan miten midletti on "konfiguroitu". Tässä joudutaan tutkimaan, halusiko käyttäjä **double**-tyyppisen vai **int**-tyyppisen satunnaisluvun. Metodi **isSelected()** on **ChoiceGroup**-luokan metodi jolla voidaan tutkia onko jokin tietty **ChoiceGroup**-olioon kuuluva valinta suoritettu. Metodille **isSelected()** annettava parametri on valinnan indeksi. Nolla viittaa ensimmäiseen valintaan. Siinä **ChoiceGroup**-oliossa johon **double_valinta** viittaa on vain yksi valintamahdollisuus ja siihen tässä viitataan indeksillä 0.

```
   protected void paint( Graphics graphics )
   {
      graphics.setColor( 255, 255, 255 ) ;  //  Valkoinen vari kayttoon
      graphics.fillRect( 0, 0, getWidth(), getHeight() ) ;

      graphics.setColor( 0, 0, 0 ) ;  //  Musta vari kayttoon

      graphics.drawString( "VIIMEKSI ARVOTTU SATUNNAISLUKU:",
                           10, 20, Graphics.TOP | Graphics.LEFT ) ;

      if ( double_valinta.isSelected( 0 ) )
      {
         graphics.drawString( ""  +  arvottu_satunnaisluku_double,
                              10, 40, Graphics.TOP | Graphics.LEFT ) ;
      }
      else
      {
         graphics.drawString( ""  +  arvottu_satunnaisluku_int,
                              10, 40, Graphics.TOP | Graphics.LEFT ) ;
      }
   }
}
```

**SatunnaislukujaMIDlet.java - 4. SatunnaislukuCanvas-luokan ja koko ohjelman loppuosa.**

## *CustomItemDemoMIDlet.java – CustomItem-luokan käyttö*

**CustomItem** on standardiluokka jota voidaan käyttää kun halutaan midlettiin piirtoalue joka ei käsitä koko näyttöä. **CustomItem** on abstrakti luokka, mikä tarkoittaa että siitä aina johdetaan uusi luokka kun halutaan "kustomoitu item" tehdä.

**CustomItem**-luokasta johdettuun luokkaan tulee aina kirjoittaa tässä näkyvät metodit joiden avulla midlettiä suorittava ajoympäristö voi saada tietoa minkä kokoisena "kustomoitu item" halutaan näkyvän.

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

import javax.microedition.midlet.* ;
import javax.microedition.lcdui.* ;

class PaintableArea extends CustomItem
{
   static final int DRAWING_AREA_WIDTH  = 180 ;
   static final int DRAWING_AREA_HEIGHT = 140 ;

   CustomItemDemoMIDlet master_midlet ;

   public PaintableArea( CustomItemDemoMIDlet given_midlet )
   {
      super( "PAINTABLE AREA" ) ;

      master_midlet = given_midlet ;
   }

   protected int getMinContentWidth()
   {
      return DRAWING_AREA_WIDTH ;
   }

   protected int getMinContentHeight()
   {
      return DRAWING_AREA_HEIGHT ;
   }

   protected int getPrefContentWidth( int tentative_content_height )
   {
      return DRAWING_AREA_WIDTH ;
   }

   protected int getPrefContentHeight( int tentative_content_width )
   {
      return DRAWING_AREA_HEIGHT ;
   }

   public void request_repaint()
   {
      repaint() ;
   }
```

**CustomItemDemoMIDlet.java - 1: Esimerkki CustomItem-luokan käytöstä.**

```
    protected void paint( Graphics graphics,
                          int current_item_width,
                          int current_item_height )
    {
       graphics.setColor( 255, 255, 0 ) ;  //  Yellow color
       graphics.fillRect( 0, 0, current_item_width, current_item_height ) ;

       graphics.setColor( 255, 0, 0 ) ;  //  Red color


       if ( master_midlet.ball_drawing_is_selected() == true )
       {
          int ball_diameter = current_item_height / 2 ;

          graphics.fillArc( current_item_width / 2 - ball_diameter / 2,
                            current_item_height / 2 - ball_diameter / 2,
                            ball_diameter, ball_diameter,
                            0, 360 ) ;
       }
       else if ( master_midlet.triangle_drawing_is_selected() == true )
       {
          int triangle_height = current_item_height / 2 ;
          int triangle_width  = triangle_height * 3 / 2 ;

          int triangle_top_point_x  =  current_item_width / 2 ;
          int triangle_top_point_y  =  current_item_height / 4 ;

          graphics.fillTriangle( triangle_top_point_x,
                                 triangle_top_point_y,
                                 triangle_top_point_x  -  triangle_width / 2,
                                 triangle_top_point_y  +  triangle_height,
                                 triangle_top_point_x  +  triangle_width / 2,
                                 triangle_top_point_y  +  triangle_height ) ;
       }
       else if ( master_midlet.square_drawing_is_selected() == true )
       {
          int square_height = current_item_height / 2 ;

          graphics.fillRect( current_item_width / 2 - square_height / 2,
                             current_item_height / 2 - square_height / 2,
                             square_height, square_height ) ;
       }
    }
}
```

Tällä sivulla olevalle **`paint()`**-metodille tulee parametreinä tieto siitä minkä kokoisena piirtoalue on juuri nyt näytöllä. Kun midlettiä suoritetaan, piirtoalueen koko voi muuttua, ainakin monimutkaisemmissa sovelluksissa.

Esimerkiksi tässä piirretään **`fillRect()`**-metodin avulla näytölle neliö siinä tapauksessa jos neliön piirto on valittuna **`ChoiceGroup`**-valikossa. Midlettiluokassa, siis **`MIDlet`**-luokassa johdetussa luokassa, on metodit joiden avulla tämä **`paint()`**-metodi voi tarkastaa mikä kuvio on valittu piirrettäväksi.

**CustomItemDemoMIDlet.java - 2: CustomItemistä johdetun PaintableArea-luokan loppuosa.**

**PaintableArea**-olio on kuin mikä tahansa muu **Item**-luokan alaluokan olio, eli se voidaan kiinnittää **Form**-olioon metodin **append()** avulla. Tässä kiinnitetään formiin ensin piirtoalue ja sitten **ChoiceGroup**-olio. Sekä **ChoiceGroup** että **CustomItem** ovat **Item**-luokan alaluokkia.

Tässä rakennetaan **ChoiceGroup**-valikko eli "valintaryhmä" jonka avulla voidaan valita piirtoalueelle piirrettävä kuvio. Valittavat kuviot ovat pallo, kolmio ja neliö. Kun valikokko luodaan parametrilla **Choice.EXCLUSIVE**, siinä voi kerrallaan olla vain yksi valinta tehtynä.

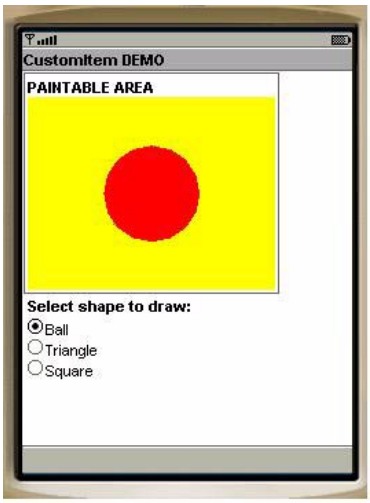
```
public class CustomItemDemoMIDlet extends MIDlet
                                  implements ItemStateListener
{
   Display  midlet_display  =  Display.getDisplay( this ) ;
   PaintableArea  paintable_area  = new PaintableArea( this ) ;

   Form  form_for_items  =  new  Form( "CustomItem DEMO" ) ;

   String[]  selectable_shapes  =  { "Ball", "Triangle", "Square" } ;

   ChoiceGroup  shape_choices  =  new  ChoiceGroup(
                                       "Select shape to draw:",
                                       Choice.EXCLUSIVE,
                                       selectable_shapes, null ) ;

   public CustomItemDemoMIDlet()
   {
      form_for_items.append( paintable_area ) ;
      form_for_items.append( shape_choices ) ;

      form_for_items.setItemStateListener( this ) ;
   }

   protected void startApp() throws MIDletStateChangeException
   {
      midlet_display.setCurrent( form_for_items ) ;
   }

   protected void pauseApp()
   {
   }

   protected void destroyApp( boolean unconditional_destruction_requested )
   {
   }

   public boolean ball_drawing_is_selected()
   {
      return ( shape_choices.getSelectedIndex() == 0 ) ;
   }
```

**CustomItemDemoMIDlet.java - 3: Midlettiluokan alkuosa.**

```
    public boolean triangle_drawing_is_selected()
    {
       return ( shape_choices.getSelectedIndex() == 1 ) ;
    }

    public boolean square_drawing_is_selected()
    {
       return ( shape_choices.getSelectedIndex() == 2 ) ;
    }


    public void itemStateChanged( Item item_which_changed_state )
    {
       if ( item_which_changed_state == shape_choices )
       {
          paintable_area.request_repaint() ;
       }
    }
}
```

**PaintableArea**-oliota pyydetään tällä metodikutsulla pyytämään näyttöalueensa sisällön päivitys siinä tapauksessa kun piirrettävä kuvio on valittu uudelleen.

**CustomItemDemoMIDlet.java - 4. Midlettiluokan loppuosa.**




PaintableArea-olio täyttää tässä vain osan näytöstä. Näin näyttöön sopii samaan aikaan myös valikko jolla piirrettävä kuvio valitaan.

**CustomItemDemoMIDlet.java - X. Ohjelma on tässä juuri käynnistetty.**

## *ClockMIDlet.java – säikeen sisältävä midletti*

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

import java.util.* ;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


class ClockCanvas extends Canvas
                  implements Runnable
{
   Thread   thread_that_runs_the_clock ;
   boolean  thread_must_be_executed  =  false ;

   int  canvas_width, canvas_height ;

   int  clock_center_point_x, clock_center_point_y  ;

   public ClockCanvas()
   {
      setFullScreenMode( true ) ;

      canvas_width  =  getWidth() ;
      canvas_height =  getHeight() ;

      clock_center_point_x  =  canvas_width / 2 ;
      clock_center_point_y  =  canvas_height / 2 + 10 ;

   }

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

Tässä ohjelmassa **Canvas**-luokasta johdettu luokka esitellään ensin ja **MIDlet**-luokasta johdettu midlettiluokka on ohjelman lopussa.

Tässä luodaan **Thread**-luokan olio, joka toimii rinnakkaisesti suoritettavana säikeenä midletin rinnalla. Säie lähtee käyntiin kun kutsutaan säieolion suhteen **Thread**-luokan metodia **start()**. Välittömästi **start()**-metodin kutsun jälkeen generoituu automaattisesti kutsu **run()**-metodiin, joka sisältää rinnakkaisesti suoritettavan säikeen ohjelmakoodin. **Thread**-luokan konstruktorille annetaan viittaus "tähän" **ClockCanvas**-luokan olioon, minkä perusteella **Thread**-oliossa tiedetään missä suoritettavaksi tarkoitettu **run()**-metodi sijaitseen.

**ClockMIDlet.java - 1: Kello jota "käytetään" säikeen avulla.**

Metodeita **start_animation_thread()** ja **stop_animation_thread()** kutsutaan midlettiluokan **destroyApp()**, **pauseApp()**- ja **startApp()**-metodeista. Käytännössä näiden metodien avulla animaatio keskeytetään silloin kun midletti asetetaan puhelimessa epäaktiiviseen tilaan tai lopetetaan kokonaan..

Säie saadaan pysätytettyä kun muuttujalle **thread_must_be_executed** asetetaan arvoksi **false**. Tällöin **run()**-metodissa oleva **while**-silmukka päättyy, ja säie "kuolee" silloin kun **run()**-metodi päättyy. Kun vielä kutsutaan **Thread**-luokan **interrupt()**-metodia tähän tapaan, varmistutaan siitä että säie "herää kuolemaan" myös siinä tapauksessa kun se on nukkumistilassa.

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

      thread_that_runs_the_clock  =  null ;
   }
}


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

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

Metodia **run()** kutsutaan automaattisesti sen jälkeen kun säie on aktivoitu **Thread**-luokan **start()**-metodia kutsumalla. Metodi **run()** siis edustaa ohjelmassa varsinaisen midletin rinnalla toimivaa itsenäistä säiettä. Tämä **run()**-metodi aiheuttaa sekunnin välein näytön päivittämisen, ja kun **paint()**-metodi puolestaan piirtää näytölle aina nykyisen kellonajan, näytölle piirrettävä kello näyttää käyvän.

**ClockMIDlet.java - 2: ClockCanvas-luokan metodeita.**

Kun **paint()**-metodia kutsutaan sekunnin välein, ja **paint()** maalaa näytölle aina kulloisenkin kellonajan, saadaan aikaan illuusio käyvästä kellosta.

Midletissä saadaan kännykän kulloinenkin kellonaika selville luomalla **Calendar**-tyyppinen olio tähän tapaan. **Calendar**-oliosta voidaan sitten poimia erilaista aikainformaatiota **get()**-metodin avulla. Esimerkiksi kun **get()**-metodille annetaan parametrina vakio **Calendar.YEAR**, saadaan selville **Calendar**-olioon talletettu vuosi.

```
public void paint( Graphics graphics )
{
   String[]  days_of_week  =  { "Sun", "Mon", "Tue",
            "Wed", "Thu", "Fri", "Sat" } ;

   String[]  names_of_months  =  { "Jan", "Feb", "Mar", "Apr",
                                   "May", "Jun", "Jul", "Aug",
                                   "Sep", "Oct", "Nov", "Dec" } ;

   Calendar time_now  =  Calendar.getInstance() ;

   int current_year  =  time_now.get( Calendar.YEAR ) ;
   int current_day   =  time_now.get( Calendar.DAY_OF_MONTH ) ;
   int month_index   =  time_now.get( Calendar.MONTH ) ;
   int number_of_day_of_week  =  time_now.get( Calendar.DAY_OF_WEEK ) ;

   String current_month  =  names_of_months[ month_index ] ;

   String current_day_of_week =  days_of_week[ number_of_day_of_week - 1 ] ;

   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 ) ;

   graphics.setColor( 200, 200, 255 ) ;  // light blue
   graphics.fillRect( 0, 0, canvas_width, canvas_height ) ;

   graphics.setColor( 0, 0, 0 ) ;  // black

   graphics.drawString( ""   + current_day_of_week  +
                        " "  + current_month +
                        " "  + current_day   +
                        ", " + current_year,
                        2, 0, Graphics.TOP | Graphics.LEFT ) ;
```

**ClockMIDlet.java - 3: ClockCanvas-luokan paint()-metodin alku.**

Kellonaika näytetään myös tekstimuodossa. Näillä lauseilla varmistetaan että minuutit ja sekunnit näkyvä tarvittaessa etunollilla varustettuna. Tämä tarkoittaa että esimerkiksi kellonaika "viisi minuuttia ja kolme sekuntia yli seitsemän" näytetään 7:05:03 eikä 7:5:3.

Tällä silmukalla piirretään 60 pistettä kellotaulun kehälle. Trigonometristen funktioiden avulla lasketaan pisteiden paikat pitäen kellotaulun säteenä arvoa 100.

```
String  minutes_string  =  "00" + current_minutes ;

minutes_string  =   minutes_string.substring(
                            minutes_string.length() - 2,
                            minutes_string.length()  ) ;

String  seconds_string  =  "00" + current_seconds ;

seconds_string  =   seconds_string.substring(
                            seconds_string.length() - 2,
                            seconds_string.length()  ) ;

graphics.drawString( current_hours + ":" + minutes_string  +
                     ":" + seconds_string,
                     2, 18, Graphics.TOP | Graphics.LEFT ) ;

// Let's print an 8-point dot in the center of the clock.

graphics.fillArc( clock_center_point_x - 4,
                  clock_center_point_y - 4, 8, 8, 0, 360 ) ;

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 ) * 100
                        +  clock_center_point_x ) ;
   int  dot_position_y  =  (int) (Math.sin( dot_angle ) * 100
                        +  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 = 6 ;
   }

   graphics.fillArc( dot_position_x - dot_diameter / 2,
                     dot_position_y - dot_diameter / 2,
                     dot_diameter, dot_diameter, 0, 360 ) ;

   dot_index  =  dot_index  +  1  ;
}
```

**ClockMIDlet.java - 4: ClockCanvas-luokan paint()-metodin jatkoa.**

Tässä lasketaan ensin kellonajasta riippuva tuntiviisarin kulma. Kun kulma ja viisarin pituus tiedetään, voidaan laskea viisarin loppupaikka trigonometrisilla funktioilla. Samalla tavalla lasketaan minuutti- ja sekuntiviisarin paikka.

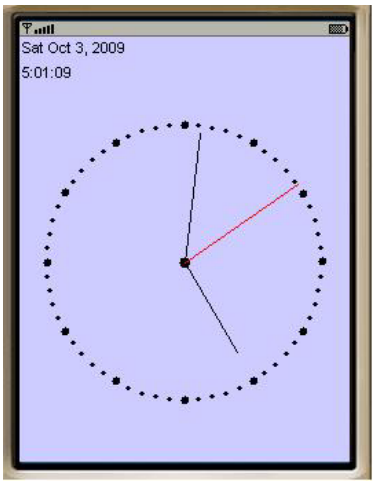
```
        double hour_hand_angle = ( current_hours * 30  +  current_minutes / 2 )
                                 * Math.PI / 180 - Math.PI /2 ;

        // 76 is the length of the hour hand

        int  hour_hand_end_x  =  (int) (Math.cos( hour_hand_angle ) * 76
                                 +  clock_center_point_x ) ;
        int  hour_hand_end_y  =  (int) (Math.sin( hour_hand_angle ) * 76
                                 +  clock_center_point_y ) ;

        graphics.drawLine( clock_center_point_x, clock_center_point_y,
                           hour_hand_end_x, hour_hand_end_y ) ;



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

        //  Minute hand length is 94.

        int  minute_hand_end_x  =  (int) (Math.cos( minute_hand_angle ) * 94
                                   +  clock_center_point_x ) ;

        int  minute_hand_end_y  =  (int) (Math.sin( minute_hand_angle ) * 94
                                   +  clock_center_point_y ) ;

        graphics.drawLine( clock_center_point_x, clock_center_point_y,
                           minute_hand_end_x, minute_hand_end_y ) ;


        graphics.setColor( 255, 0, 0 ) ; // red

        double seconds_hand_angle = ((double) current_seconds +
                                     (double) current_milliseconds / 1000.0 )
                                     * Math.PI / 30 - Math.PI /2 ;
        //  100 is the length of the seconds hand

        int  seconds_hand_end_x  =  (int) (Math.cos( seconds_hand_angle ) * 100
                                   +  clock_center_point_x ) ;
        int  seconds_hand_end_y  =  (int) (Math.sin( seconds_hand_angle) * 100
                                   +  clock_center_point_y ) ;

        graphics.drawLine( clock_center_point_x, clock_center_point_y,
                           seconds_hand_end_x, seconds_hand_end_y ) ;
    }
}
```

**ClockMIDlet.java - 5: ClockCanvas-luokan loppuosa.**

```
public class ClockMIDlet extends MIDlet
                                 implements CommandListener
{
   Display      display_of_this_midlet  =  Display.getDisplay( this ) ;
   ClockCanvas  clock_canvas            =  new  ClockCanvas() ;

   Command exit_command  =  new  Command( "Exit Clock", Command.EXIT, 1 ) ;

   protected void startApp() throws MIDletStateChangeException
   {
      display_of_this_midlet.setCurrent( clock_canvas ) ;
      clock_canvas.start_animation_thread() ;

      clock_canvas.addCommand( exit_command ) ;
      clock_canvas.setCommandListener( this ) ;
   }

   protected void pauseApp()
   {
      clock_canvas.stop_animation_thread() ;
   }

   protected void destroyApp( boolean unconditional_destruction_requested )
   {
      clock_canvas.stop_animation_thread() ;
   }


   public void commandAction( Command      given_command,
                              Displayable  display_content )
   {
      if ( given_command  ==  exit_command )
      {
         destroyApp( false ) ;
         notifyDestroyed() ;
      }
   }
}
```

Tämänkin midletin tapauksessa suurin osa toiminnoista tapahtuu **`Canvas`**-luokasta johdetussa **`ClockCanvas`**-luokassa. Midlettiluokasta käsin pannaan toimintaan ja tarvittaessa pysäytetään säie joka käyttää näytettävää kelloa. Säikeen toiminnan käynnistävät ja pysäyttävät metodit **`start_animation_thread()`** ja **`stop_animation_thread()`** ovat **`ClockCanvas`**-luokassa.

**ClockMIDlet.java - 6. ClockMIDlet luokka jossa luodaan ClockCanvas-olio.**




Kellotaulun halkaisija on 200 pistettä (pikseliä). Nykyisissä puhelimissa näytön leveys yleensä ylittää tämän pikselimäärän joten kellon pitäisi sopia puhelinten näyttöön.

Kello asettaa itsensä automaattisesti jotakuinkin keskelle näyttöä. Huomaa että ohjelma on asettanut näytön Full Screen -moodiin.

**ClockMIDlet.java - X. Midlettiä suoritetaan lauantaina 3 lokakuuta 2009 kello 05:01:09.**

## *PelikortitMIDlet.java – oliosuuntautuneesti rakennettu midletti*

Rajapinta **Korttivakiot** sisältää joukon vakioita joita hyödynnetään sekä **Kortti**- että **Korttipakka**-luokassa.

Tämän datakentän arvon perusteella **piirra()**-metodi piirtää "tämän" korttiolion joko oikeinpäin (tietopuoli näkyvissä) tai nurinpäin. Tämän muuttujan arvon voi muuttaa esim. metodilla **kaanna_kortti()**.

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

import javax.microedition.midlet.* ;
import javax.microedition.lcdui.* ;
import java.util.* ;

interface Korttivakiot
{
   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   =   75 ;
   public static final int  KORTIN_KORKEUS_PIKSELEINA  =  100 ;
}


class Kortti implements Korttivakiot
{
   int kortin_arvo ;
   int kortin_maa ;

   boolean  tama_kortti_on_oikeinpain  =  false ;

   int kortin_paikka_x  =  0 ;
   int kortin_paikka_y  =  0 ;

   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 ;
   }
```

**PelikortitMIDlet.java - 1: Pelikorttien käyttöä esittelevä midletti.**

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

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( int annettu_kortin_paikka_x,
                                      int annettu_kortin_paikka_y )
{
   kortin_paikka_x  =  annettu_kortin_paikka_x ;
   kortin_paikka_y  =  annettu_kortin_paikka_y ;
}
```

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

**Kortti**-oliota luotaessa 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.

**PelikortitMIDlet.java - 2: Kortti-luokan metodeja.**

```
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  =  "Kumma juttu" ;
   }

   return  palautettava_stringi ;
}

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 ) ;
}
```

Kortti-luokan sisällä kortin maa talletetaan numeroarvoina joita edustavat vakiot **HERTTA**, **RUUTU**, **PATA** ja **RISTI**. Maa voidaan haluttaessa muuttaa stringiksi tämän metodin avulla.

**PelikortitMIDlet.java - 3: Lisää Kortti-luokan metodeja.**

```
    public void piirra( Graphics  graphics )
    {

       if ( tama_kortti_on_oikeinpain   ==  true  )
       {
          graphics.setColor( 0x00FFFF00 ) ;  // keltainen
       }
       else
       {
          graphics.setColor( 0x0000FFFF ) ;  // cyan
       }

       graphics.fillRect( kortin_paikka_x, kortin_paikka_y,
                          KORTIN_LEVEYS_PIKSELEINA,
                          KORTIN_KORKEUS_PIKSELEINA ) ;

       graphics.setColor( 0x00000000 ) ;  //  musta

       graphics.drawRect( kortin_paikka_x, kortin_paikka_y,
                          KORTIN_LEVEYS_PIKSELEINA,
                          KORTIN_KORKEUS_PIKSELEINA ) ;

       if ( tama_kortti_on_oikeinpain  ==  true )
       {
          graphics.drawString( lue_maa_stringina(),
                               kortin_paikka_x  + 20,
                               kortin_paikka_y  +
                                   ( ( KORTIN_KORKEUS_PIKSELEINA * 2 ) / 3 ),
                               Graphics.TOP | Graphics.LEFT ) ;

          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_x  + 20,
                               kortin_paikka_y  +
                                    ( KORTIN_KORKEUS_PIKSELEINA  / 3  ),
                               Graphics.TOP | Graphics.LEFT ) ;
       }
    }
}
```

Silloin kun kortti on nurinpäin, eli sen maa ja arvo eivät ole näkyvissä, se piirretään cyan-värisenä.

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

**PelikortitMIDlet.java - 4: Kortti-luokan lopussa oleva piirra()-metodi.**

```
class Korttipakka  implements Korttivakiot
{
   static int[]  maat  =  { HERTTA, RUUTU, PATA, RISTI } ;

   Kortti[]  pakan_kortit   =  new  Kortti[ 52 ] ;

   int    kortteja_pakassa  =  0 ;

   public void sekoita_pakka()
   {
      if ( kortteja_pakassa  >  0 )
      {
         Random  satunnaislukugeneraattori  =  new  Random() ;

         for ( int kortin_indeksi  =  0 ;
                   kortin_indeksi  <  kortteja_pakassa ;
                   kortin_indeksi  ++ )
         {
            int satunnainen_indeksi  =
                      satunnaislukugeneraattori.nextInt( kortteja_pakassa ) ;

           Kortti satunnainen_kortti  =  pakan_kortit[ satunnainen_indeksi ] ;

            pakan_kortit[ satunnainen_indeksi ]  =
                                        pakan_kortit[ kortin_indeksi ] ;
            pakan_kortit[ kortin_indeksi ]  =  satunnainen_kortti ;
         }
      }
   }

   public void lisaa_kortti( Kortti uusi_kortti )
   {
      if ( kortteja_pakassa  <  52 )
      {
         pakan_kortit[ kortteja_pakassa ]  =  uusi_kortti ;
         kortteja_pakassa  ++  ;
      }
   }

   public Kortti ota_kortti()
   {
      Kortti  annettava_kortti  =  null ;

      if ( kortteja_pakassa  >  0 )
      {
         annettava_kortti  =  pakan_kortit[ kortteja_pakassa - 1 ] ;
         pakan_kortit[ kortteja_pakassa - 1 ]  =  null ;
         kortteja_pakassa  -- ;
      }

      return  annettava_kortti  ;
   }
}
```

**Korttipakka**-luokassa ei ole yhtään korttia. Tämä luokka on tarkoitettu kanta-luokaksi siitä johdetuille luokille.

Tämä metodi poistaa alimmaisen kortin pakasta ja antaa kutsujalleen viittauksen poistettuun korttiin. Metodi palauttaa arvon **null**, jos pakassa ei ole enää kortteja.

**PelikortitMIDlet.java - 5: Korttipakka-luokan määrittely.**

```
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.

**PelikortitMIDlet.java - 6: Kuningaspakka- ja Pokeripakka-luokat.**

Midlettiluokka on hyvin yksinkertainen. Luotu **PelikortitCanvas**-olio pannaan näyttöön kun midletti käynnistyy.

Tässäkin ohjelmassa varsinainen 'action' tapahtuu **Canvas**-luokasta johdetussa luokassa.

```
public class PelikortitMIDlet extends MIDlet
{
   Display  taman_midletin_naytto  =  Display.getDisplay( this ) ;

   PelikortitCanvas pelikortit_canvas  =  new  PelikortitCanvas() ;

   public PelikortitMIDlet()
   {
   }

   public void startApp()
   {
      taman_midletin_naytto.setCurrent( pelikortit_canvas ) ;
   }

   public void pauseApp()
   {
   }

   public void destroyApp( boolean ehdoton_tuhoutuminen_vaadittu )
   {
   }
}


class PelikortitCanvas extends Canvas
                       implements  CommandListener
{
   Korttipakka  korttipakka  =  new  Pokeripakka() ;

   Kortti edellinen_kortti  =  null ;

   Kortti viimeksi_otettu_kortti  =  null ;

   Command sekoita_pakka_komento = new Command( "Sekoita pakka",
                                                Command.EXIT, 1) ;
   Command uusi_kortti_komento   = new Command( "Uusi kortti",
                                                Command.ITEM, 1) ;

   public PelikortitCanvas()
   {
      addCommand( sekoita_pakka_komento ) ;
      addCommand( uusi_kortti_komento ) ;
      setCommandListener( this ) ;
   }
```

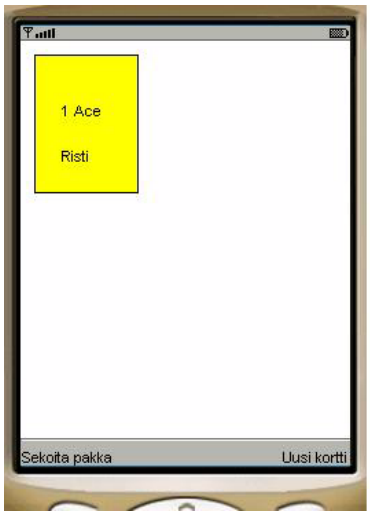
**PelikortitMIDlet.java - 7: Midlettiluokka ja PelikortitCanvas-luokan alku.**

**Kortti**-oliolle asetetaan paikka näytölle vasta sen jälkeen kun se on otettu pakasta. Metodin **kaanna_kortti()** avulla kortti saadaan piirtymään näytölle oikeinpäin.

```
   public void commandAction( Command      annettu_komento,
                              Displayable  nayton_sisalto )
   {
      if ( annettu_komento == uusi_kortti_komento )
      {
         edellinen_kortti  =  viimeksi_otettu_kortti ;

         viimeksi_otettu_kortti  =  korttipakka.ota_kortti() ;
         viimeksi_otettu_kortti.aseta_paikka_pikseleina( 10, 10 ) ;
         viimeksi_otettu_kortti.kaanna_kortti() ;
      }
      else if ( annettu_komento  ==  sekoita_pakka_komento )
      {
         korttipakka.sekoita_pakka() ;
      }

      repaint() ;
   }

   protected void paint( Graphics graphics )
   {
      graphics.setColor( 255, 255, 255 ) ;  // Valkoinen
      graphics.fillRect( 0, 0, getWidth(), getHeight() ) ;

      graphics.setColor( 0, 0, 0 ) ;

      if ( viimeksi_otettu_kortti  !=  null )
      {
         viimeksi_otettu_kortti.piirra( graphics ) ;

         if ( edellinen_kortti  !=  null  &&
              edellinen_kortti.on_eri_maata_kuin( viimeksi_otettu_kortti ) )
         {
            graphics.drawString( "\n MAA VAIHTUI!", 10, 130,
                                 Graphics.TOP | Graphics.LEFT ) ;
         }
      }
   }
}
```

Kun pakasta otetaan näytettäväksi uusi kortti, viittaus edelliseen korttiolioon pannaan talteen. Silloin kun huomataan että uusi pakasta otettu kortti on eri maata kuin edellinen kortti, tulostetaan teksti "MAA VAIHTUI" kortin alapuolelle. Olioviittaajien arvo on alussa **null** kun kortteja ei vielä ole pakasta otettu.

**PelikortitMIDlet.java - 8. PelikortitCanvas-luokan ja koko ohjelman loppu.**




Tässä on painettu Uusi kortti -nappulaa sekoittamatta pakkaa ensin. Kun midletti käynnistetään eikä pakkaa sekoiteta, ristiässä on ensimmäinen näytettävä kortti. Sekoittamattomassa pakassa kortit ovan suuruusjärjestyksessä siten että ristit ovat pantu pakkaan viimeisenä.

**PelikortitMIDlet.java - X. Midletti suoriintumassa emulaattorissa.**

## *TekstiNetistaMIDlet.java – Internetista tekstitiedostoja lukeva ohjelma*

Tämä ohjelma esittää aluksi käyttäjälle listan Internetissä sijaitsevia tekstitiedostoja, joista käyttäjä valitsee yhden noudettavaksi. Tälle listalle päästään takaisin kun annetaan nettiosoitteen vaihtokomento.

```
//  TekstiNetistaMIDlet.java

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TekstiNetistaMIDlet extends MIDlet
                                 implements CommandListener, Runnable
{
   Thread  tekstin_vastaanottava_saie ;

   Display  taman_midletin_naytto ;

   TextBox tekstiboxi_naytolla ;

   String  valittu_nettiosoite  =  "Ei URLia valittu." ;

   String[]  valittavat_nettiosoitteet  =
    {
      "http://www.oamk.fi/~karil/shortstory.txt",
      "http://www.oamk.fi/~karil/longerstory.txt",
      "http://www.oamk.fi/~karil/numberstory.txt",
      "http://www.bof.fi/ohi/fin/0_new/0.1_valuuttak/fix-rec.txt",
      "http://www.naturalprogramming.com/javabookprograms/javafiles1/Sum.java",
    } ;

   List  nettiosoitteen_valintamenu  =  new  List( "VALITSE NETTIOSOITE:",
                                                   List.IMPLICIT,
                                                   valittavat_nettiosoitteet,
                                                   null ) ;

   Command  nettiosoitteen_vaihtokomento = new Command( "Vaihda URL",
                                                        Command.SCREEN, 1 ) ;

   public TekstiNetistaMIDlet()
   {
      taman_midletin_naytto  =  Display.getDisplay( this ) ;

      tekstiboxi_naytolla = new TextBox( "TEKSTI NETISTÄ:", valittu_nettiosoite,
                                         4092, TextField.ANY ) ;

      tekstiboxi_naytolla.addCommand( nettiosoitteen_vaihtokomento ) ;
      tekstiboxi_naytolla.setCommandListener( this ) ;

      nettiosoitteen_valintamenu.setCommandListener( this ) ;
   }
```

**TekstiNetistaMIDlet.java - 1: Tekstitiedostoja Internetistä lukeva midletti.**

```
protected void destroyApp( boolean ehdoton_tuhoutuminen_vaadittu )
{
}

protected void pauseApp()
{
}

protected void startApp() throws MIDletStateChangeException
{
   taman_midletin_naytto.setCurrent( nettiosoitteen_valintamenu ) ;
}

public void commandAction( Command      annettu_komento,
                           Displayable  display_content )
{
   if ( annettu_komento == nettiosoitteen_vaihtokomento )
   {
      taman_midletin_naytto.setCurrent( nettiosoitteen_valintamenu ) ;
   }
   else if ( annettu_komento  ==  List.SELECT_COMMAND )
   {
      /* List.SELECT_COMMAND on eraanlainen oletuskomento
         joka tulee kasittelyyn kun IMPLICIT-tyypin
         listasta suoritetaan valinta.  */

      int  valitun_osoitteen_indeksi  =
               nettiosoitteen_valintamenu.getSelectedIndex() ;

      valittu_nettiosoite  =  nettiosoitteen_valintamenu.getString(
                                        valitun_osoitteen_indeksi ) ;

      taman_midletin_naytto.setCurrent( tekstiboxi_naytolla ) ;

      tekstin_vastaanottava_saie  =  new  Thread( this ) ;
      tekstin_vastaanottava_saie.start() ;
   }
}
public void run()
{
   //  Kun run() -metodi ei sisälla ikuista silmukaa,
   //  tapahtuu niin etta kun run()-metodi on suoritettu,
   //  säie lakkaa olemasta ja kuolee pois.

   vastaanota_teksti_netista() ;
}
```

Teksti luetaan Internetistä heti kun uusi nettiosoite on valittu. Internet-yhteyden hallintaa varten on käynnistettävä erillinen säie joka suorittaa tekstin lukemisen verkosta. Säiettä edustava **`run()`**-metodi käynnistyy automaattisesti kun **`start()`**-metodia kutsutaan luodun säieolion suhteen.

**TekstiNetistaMIDlet.java - 2: Ohjelman toinen osa.**

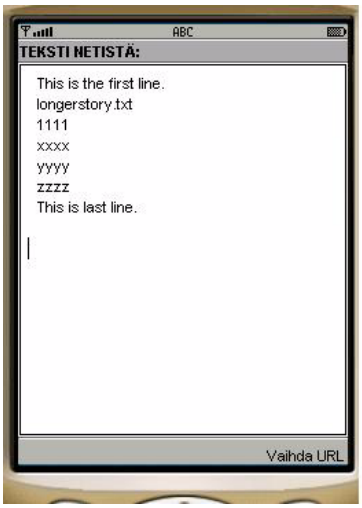
Internet-yhteys luodaan käyttämällä hyväksi standardiluokkia **InputConnection**, **Connector** ja **InputStream**.

```
   void vastaanota_teksti_netista()
   {
      InputConnection  input_connection  =  null ;
      InputStream      input_stream      =  null ;

      StringBuffer     luetut_merkit  =  new StringBuffer( "" ) ;

      try
      {
         input_connection  = (InputConnection)
                   Connector.open( valittu_nettiosoite, Connector.READ ) ;

         input_stream  =  input_connection.openInputStream() ;

         input_connection.close() ;

         int luettu_merkkikoodi ;

         while ( ( luettu_merkkikoodi  =  input_stream.read() ) != -1 )
         {
            luetut_merkit.append( (char) luettu_merkkikoodi ) ;
         }

         tekstiboxi_naytolla.setString( luetut_merkit.toString() ) ;
      }
      catch ( Exception caught_exception )
      {
         System.out.print( "\n " + caught_exception.getMessage()
                         + "\n " + valittu_nettiosoite ) ;
      }
      finally
      {
         try
         {
            if ( input_stream  !=  null )
            {
               input_stream.close() ;
            }

            if ( input_connection  !=  null )
            {
               input_connection.close() ;
            }
         }
         catch ( IOException poikkeus )
         {
         }
      }
   }
}
```

Yhteyden sulkeminen tässä vaiheessa ei vaikuta ohjelman toimintaan.

Tekstiä luetaan verkosta merkki kerrallaan ja luetut merkit talletetaan **StringBuffer**-olioon. Lopuksi teksti muutetaan stringiksi ja talletetaan **TextBox**-olioon jossa se tulee näkyviin näytölle.

**TekstiNetistaMIDlet.java - 3. Tekstitiedoston Internetistä lukeva metodi.**

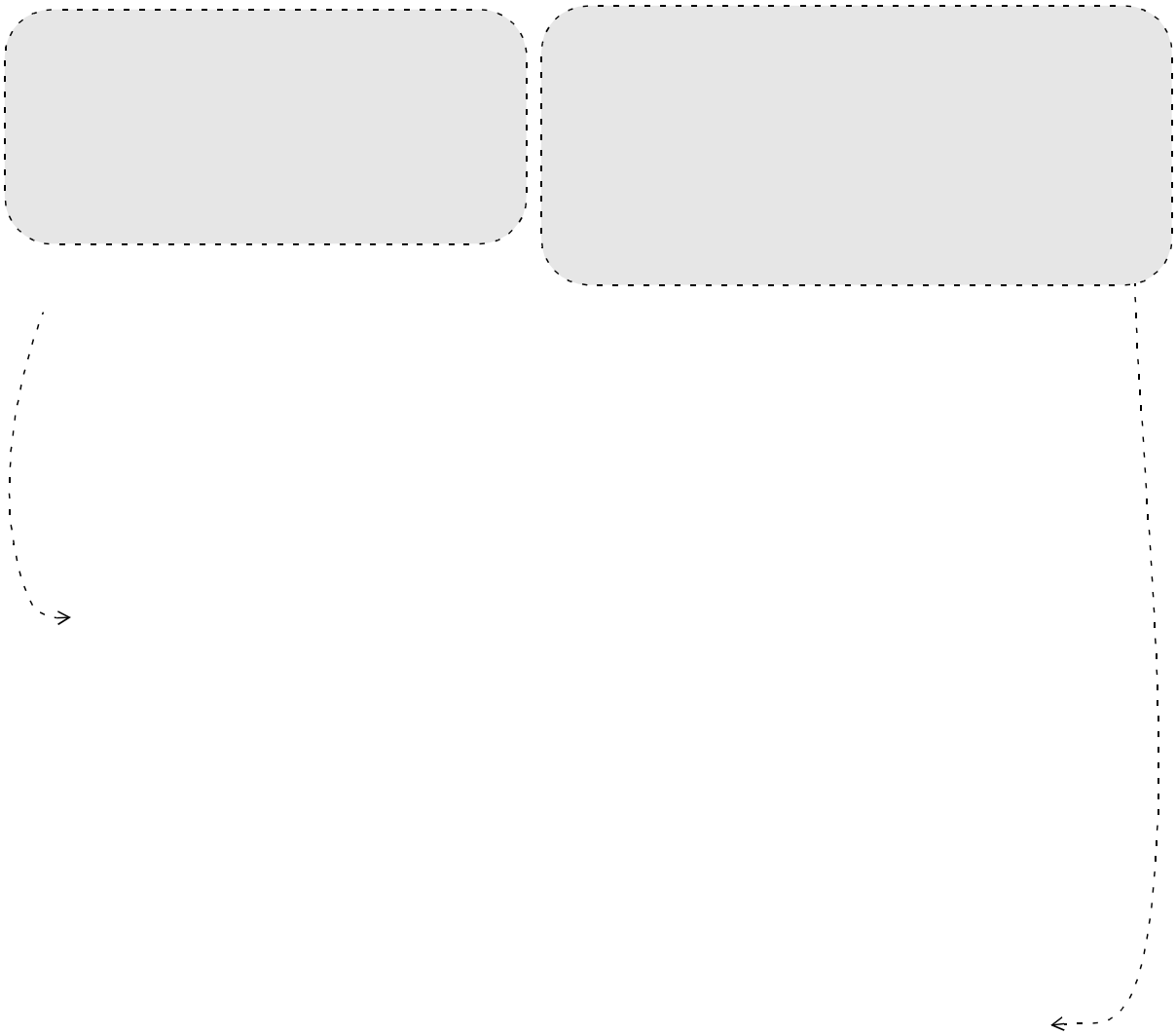




Tässä on esillä midletin aloitusnäkymä, jossa on valittavana viisi eri nettiosoitetta. Kun käyttäjä suorittaa listalta valinnan, tekstin lataaminen verkosta alkaa välittömästi. Yleensä ennen kuin tekstin lataaminen alkaa, puhelin ja myös simulaattori näyttää automaattisesti näkymän jossa varmistetaan että käyttäjä todella haluaa käyttää maksullista verkkoyhteyttä.

Tässä on tekstitiedosto **http://www.oamk.fi/~karil/longerstory.txt** ladattu verkosta ja pantu näkyviin `TextBox`iin näytölle.

**TekstiNetistaMIDlet.java - X. Kaksi näkymää ohjelman suoriintumisesta.**

### *Otsikko*



**MIDlet.java - 1:**