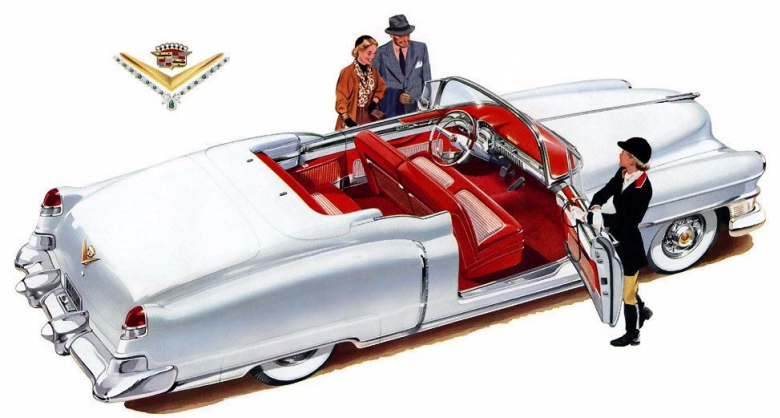
HARJOITUKSIA JAVA-MIDLETTEIHIN LIITTYEN

- JCreatorilla työskenneltäessä ei ole tarpeellista perustaa projekteja. Kun editoit midlettiohjelmaa JCreatorilla ja suoritat sitten ohjelman kääntämisen ja suorittamisen Sun Java Wireless Toolkitillä, muista TALLETTAA ohjelma ennen kääntämistä.
- Älä pidä tiedostoja verkkolevyllä kun teet ohjelmointiharjoituksia.
- Varo antamasta Java-ohjelmallesi nimeä String.java, koska String on Javan standardiluokka. Anna midlettiohjelmallesi mieluiten pitkähkö nimi tyyliin KivaPeliMIDlet.java.



Cadillac 1953

Cadillac 1957

Kari Laitinen
http://www.naturalprogramming.com
2006-03-13 Tiedosto luotu.
2013-10-09 Viimeisin muutos

HARJOITUKSIA OHJELMALLA SumMIDlet.java

1. Paranna ohjelmaa aluksi siten että lisäät summattavat luvut sisältävien tekstikenttien väliin uuden tekstikentän joka on operaattoritekstikenttä joka sisältää alussa tekstin "+". Tarkoitus on että ohjelman käyttäjä voi editoida tätä tekstikenttää ja kirjoittaa siihen toisen aritmeettisen operaattorin kuten "-", "*", "/" tai "%". TextField-olio tulee luoda seuraavaan tapaan parametrilla TextField.ANY jotta kenttään hyväksytään kaikki merkit:

```
operator_text_field  = new TextField( "OPERATOR:",
                                      "+", 1, TextField.ANY ) ;
```

Wireless Toolkitin simulaattorissa saa erikoismerkkejä kirjoitettua kun painelee vasemman alakulman tähtinäppäintä.

Lisää aluksi ohjelmaan uusi tekstikenttä ja pistä vasta tämän jälkeen se toimintaan. Uuden tekstikentän sisältöä voi laskennan suorittavassa metodissa tutkia esim. seuraavaan tapaan:

```
if ( operator_text_field.getString().equals( "-" ) )
{
   calculation_result  =
                first_integer  -  second_integer ;
}
```

Javassa käytetään String-olioiden sisältöjen vertailuun equals()-metodia. Huomaa, että jakolaskua ja jakojäännöslaskua ei saa suorittaa jos jakaja on nolla.

2. Lisää ohjelmaan viides tekstikenttä, siis TextField-olio, joka näyttää laskennan tuloksen heksadesimaalisena lukuna. Laskennan tulos voidaan muuttaa heksadesimaaliseksi stringiksi käyttämällä Integer-luokan staattista toHexString()-metodia seuraavaan tapaan

```
String hexadecimal_result_text  =
          Integer.toHexString( calculation_result ) ;
```

Heksadesimaalisen luvun sisältävä TextField-olio täytyy luoda parametrilla TextField.ANY, koska sen pitää pystyä sisältämään myös kirjaimia.

3. Ensimmäisessä osatehtävässä tehty ominaisuus on hiukan vaikeakäyttöinen. Lisää tämän vuoksi ohjelmaan ominaisuus että käytettävä aritmeettinen operaatio voidaan valita komentojen, siis Command-olioiden avulla. Ohjelmassa on jo käytössä Command-olio johon viittaa exit_command. Voit esim. määritellä ohjelmaan seuraavat oliot:

```
private Command add_command  =
              new Command( "ADD", Command.SCREEN, 1 ) ;
private Command subtract_command  =
             new Command( "SUBTRACT", Command.SCREEN, 1 ) ;
private Command multiply_command  =
             new Command( "MULTIPLY", Command.SCREEN, 1 ) ;
private Command divide_command    =
              new Command( "DIVIDE", Command.SCREEN, 1 ) ;
private Command remainder_command   =
             new Command( "REMAINDER", Command.SCREEN, 1);
```

Kun nämä Command-oliot sitten kiinnitetään midletin formiin, niistä muodostuu automaattisesti menu, josta voidaan tehdä valintoja.

commandAction()-metodissa tulee sitten reagoida näihin uusiin komentoihin samaan tapaan kuin siellä nyt reagoidaan Command-olioon johon viittaa exit_command. Riippuen siitä mikä komento on annettu, voidaan operaattorin sisältävän tekstikentän sisältö muuttaa. Esimerkiksi seuraavanlainen lause voisi ottaa kertolaskuoperaattorin käyttöön:

```
operator_text_field.setString( "*" ) ;
```

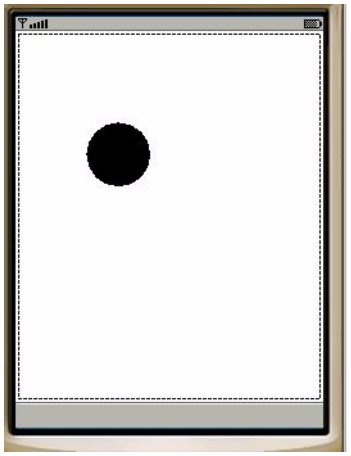
Kun operaattorin määrittelevä stringi on muutettu tekstikenttään, voidaan commandAction()-metodin lopussa kutsua itemStateChanged()-metodia seuraavaan tapaan

```
itemStateChanged( null ) ;
```

Tällä tavalla saadaan tekstikentät päivitetyiksi siinä tapauksessa että käytetty aritmeettinen operaattori muuttui. (Ei ehkä ole hienoa ohjelmointityyliä kutsua itemStateChanged()-metodia näin, mutta se näyttää toimivan. Parempi tapa olisi tehdä erillinen metodi laskutoimitusten suoritukseen ja kutsua tuota metodia sekä commandAction()- että itemStateChanged()-metodeista.)

HARJOITUKSIA OHJELMALLA KeyCodesMIDlet.java

HARJOITUS 1: Tee ohjelmasta aluksi sellainen, että se piirtää käynnistyttyään ruudulle kiinteän pallon. Tämä tarkoittaa että voit poistaa KeyCodesCanvas-luokan paint()-metodista entiset kutsut jotka piirtävät stringejä. Jätä kuitenkin alkuun ohjelmarivit jotka tyhjentävät näytön ja muuttavat piirtovärin mustaksi. Kiinteän pallon saa piirrettyä Graphics-luokan fillArc()-metodilla. Pallo voi ruudulla sijaita suurinpiirtein seuraavan kuvan esittämällä tavalla



HARJOITUS 2: Kun olet saanut edellisessä kohdassa vaaditun pallon näkymään ruudulla, tee ohjelmasta sellainen että pallon saa pois näkyvistä, kun painaa näppäintä 0 (Nolla), ja pallon saa uudelleen näkyviin kun painaa näppäintä 0 toistamiseen. Tämän voi tehdä esimerkiksi siten että määrittelet KeyCodesCanvas-luokkaan seuraavan datakentän (datajäsenen):

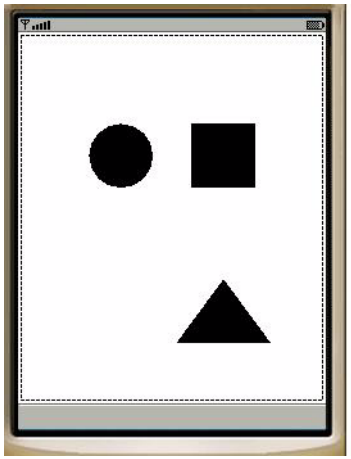
```
boolean  pallo_on_piirrettava   =  true ;
```

Tämän boolean-muuttujan arvoa voi sitten tutkia näppäimiin reagoivassa keyPressed()-metodissa ja asettaa arvoksi false jos entinen arvo on true ja päinvastoin. Luonnollisesti keyPressed()-metodissa tulee ensin tutkia oliko painettu 0-näppäintä. Tämä voidaan tehdä seuraavasti:

```
public void keyPressed( int key_code )
{
   if ( key_code  ==  '0' )
   {
      if ( pallo_on_piirrettava  ==  true )
      {
         pallo_on_piirrettava  =  false ;
         ...
```

Luonnollisesti tämän pallo_on_piirrettava-muuttujan arvoa pitää tutkia vielä paint()-metodissa ja pallo piirretään vain siinä tapauksessa jos muuttujalla on arvo true.

HARJOITUS 3: Jatka ohjelman kehittelyä siten että piirrät näytölle pallon lisäksi vielä sisältätäytetyn neliöön ja kolmion. Nämä saadaan aikaiseksi Graphics-luokan fillRect()- ja fillTriangle()-metodeilla. Tutki elektronista dokumentaatiota jotta tiedät mitä parametreja näille metodeille annetaan. Tämän harjoituksen jälkeen midletti voi käynnistyessään näyttää seuraavanlaiselta



HARJOITUS 4: Tee edellisessä kohdassa tehdyistä kolmiosta ja neliöstä sellaisia että ne saa halutessaan pois ruudulta ja takaisin ruudulle tiettyä näppäintä painamalla. Esimerkiksi näppäin 3 voisi vaikuttaa kolmion näkymiseen ruudulla ja näppäin 4 neliön näkymiseen ruudulla. Tarkoitus on että nämä kuvioiden näkyvyyteen vaikuttavat ominaisuudet ovat samantapaiset kuin kohdassa 2 tehty. Tässä voi käyttää seuraavia muuttujia apuna:

```
boolean  nelio_on_piirrettava   =  true ;
boolean  kolmio_on_piirrettava  =  true ;
```

HARJOITUS 5: Lisää midlettiin ominaisuus että näytölle tulostetaan "raamit". Raamit voivat olla näytön reunaa kiertävä suorakaide joten ne saadaan aikaiseksi drawRect()-metodilla. Tee raameista kuitenkin sellaiset että ne piirretään katkoviivalla. Graphics-luokassa on metodi nimeltä setStrokeStyle(), jolla viivan piirtotyylin voi muuttaa joko katkoviivaksi tai jatkuvaksi viivaksi. Graphics-luokassa on määritelty vakiot DOTTED ja SOLID, joita voidaan antaa parametrina setStrokeStyle()-metodille. Viivan piirtotyyli tulee asettaa ennen kuin viivoja piirretään. Viivat ovat oletusarvoisesti SOLID-tyyppisiä.

HARJOITUS 6: Lisää ominaisuus että näppäintä 1 painamalla edellisessä kohdassa piirretyt raamit tulostuvat jatkuvalla viivalla, ja kun näppäintä 1 painetaan uudestaan raamit piirtyvät katkoviivalla. Tässä voi käyttää hyväksi seuraavanlaista muuttujaa:

```
boolean  raamit_piirretaan_katkoviivalla  =  true ;
```

HARJOITUS 7: Display-luokassa on metodi jolla puhelimen taustavalot saa palamaan tietyksi määräksi millisekunteja. Tutki Display-luokan dokumentaatiota ja muuta midlettisi sellaiseksi, että aina kun raamien piirtotyyliä muutetaan, siis aina kun näppäintä 1 painetaan, puhelimen taustavaloa näytetään kahden sekunnin aika.

Tämä tehtävä on muuten melkoisen yksinkertainen, mutta ongelma on siinä että midletin Display-olio on määritelty midlettiluokassa kun taas näppäimiin reagoiva metodi on toisessa luokassa. Eräs tapa ratkaista tämä ongelma on antaa konstruktorille parametrina "this" siinä vaiheessa kun Canvas-pohjaisen luokan olio luodaan. Canvas-luokasta johdettuun luokkaan voidaan siis kirjoittaa seuraavannäköinen konstruktori:

```
KeyCodesHarjoitusMIDlet host_midlet ;

public KeyCodesCanvas(
                KeyCodesHarjoitusMIDlet  given_midlet )
{
   host_midlet  =  given_midlet ;
}
```

Kun tämmöiset määrittelyt on tehty, Canvas-pohjaisesta luokasta päästään käsiksi midlettiluokan datajäseniin seuraavalla tavalla

```
host_midlet.display_of_this_midlet ...
```

Jotta tämä onnistuu, on midlettiluokan datajäsenistä poistettava private-määrittely.

HARJOITUS 8: Lisää ohjelmaan vielä toinen Canvas-pohjainen luokka, jonka olio pannaan näyttöön siinä tapauksessa kun näppäintä 9 painetaan. Tämän toisen Canvas-luokasta johdetun luokan pitää näyttää jotain kuvaa midletin näytöllä. Ohjelmasta KuvashowMIDlet.java näet kuinka kuva saadaan näytettyä. Kuvatiedosto pitää pistää midlettiprojektin res-kansioon.

Kun näytössä on kuvaa näyttävä canvas, on sen toimittava siten että päästään takaisin alkuperäiseen graafisia kuvioita sisältävään canvasiin mitä tahansa näppäintä painamalla. Tämän vuoksi kuvaa näyttävän canvas-luokan konstruktorille on annettava viittaus midlettiolioon, jotta sieltä päästää käsiksi Display-olioon. Kuvaa näyttävä canvas-luokka voi näin ollen alkaa seuraavaan tapaan:

```
class KuvaCanvas extends Canvas
{
   Image   naytettava_kuva  ;

   KeyCodesHarjoitusMIDlet host_midlet ;

   public KuvaCanvas(
               KeyCodesHarjoitusMIDlet  given_midlet )
   {
      host_midlet  =  given_midlet ;
      ...
```

Kuvaa näyttävän canvas-luokan keyPressed()-metodi voi sitten asettaa toisen canvas-olion näyttöön seuraavalla tavalla:

```
   public void keyPressed( int nappainkoodi )
   {

      host_midlet.display_of_this_midlet.setCurrent(
                        host_midlet.key_codes_canvas ) ;
      ...
```

HARJOITUKSIA OHJELMALLA LiikkuvaPalloMIDlet.java

1. Muuta pallon piirtämistä siten että pallolle piirretään aina mustalla värillä ääriviivat. Tarkoitus on että ensin täytetään valitulla värillä pallon sisus, ja sitten vaihdetaan väri mustaksi ääriviivojen piirtoa varten.

2. Lisää värinvalintalistaan uudeksi väriksi keltainen. Keltainen saadaan aikaan RGB-värinä kun punaisen ja vihreän arvot pannaan maksimiin.

3. Nykyisellään midletissä toimii toinen Soft Key pikanäppäimenä jolla pallon värin saa nopeasti muutettua harmaaksi. Muuta tuon näppäimen toiminta sellaiseksi että sillä saa käyttöön aina pallon edellisen värin. Tarkoitus on että jos esim. pallo on ensin asetettu vihreäksi ja sen jälkeen puhaiseksi, värin saa pikanäppäimellä nopeasti takaisin punaiseksi. Tässä täytyy aina värimuutosten yhteydessä ottaa talteen pallon edellinen väri. Tämän voi tallettaa esim. seuraavanlaiseen LiikkuvaPalloCanvas-luokan datakenttään:

```
int  edellinen_vari  =  0x007F7F7F ;
```

Edelliseen väriin siirtymisen mahdollistava komento voidaan määritellä esim. seuraavaan tapaan

```
Command  edellisen_varin_asetuskomento  =
                    new Command( "Edellinen vari",
                                 Command.BACK, 5 ) ;
```

Tarkoitus on että tämä komento korvaa aiemman harmahtavanasetuskomennon.

4. Lisää värivalintalistalle "Satunnaisvarinen pallo". Tarkoitus on että jos midletin käyttäjä valitsee tämän värin, niin midletti arpoo satunnaislukugeneraattorin avulla pallolle värin. Jos jokainen värikomponentti arvotaan erikseen siten että ne ovat alueella 0 ... 255, värikomponentit voidaan yhdistää yhdeksi RGB-väriksi seuraavanlaisella lauseella:

```
rgb_varin_maarittely  =  0x10000 * punaisen_arvo  +
                         0x100   * vihrean_arvo +
                         sinisen_arvo ;
```

5. Muuta ohjelma sellaiseksi että ennen alussa näytettävää värinvalintalistaa käyttäjälle näytetään lista josta käyttäjä voi valita pallolle muodon. Tämä lista saadaan aikaan esimerkiksi seuraavanlaisilla datajäsenillä:

```
String[] pallon_muotovaihtoehdot  =
                           { "Pieni pyorea pallo",
                             "Iso pyorea pallo",
                             "Pieni soikea pallo",
                             "Iso soikea pallo" } ;

List pallon_muodon_valintamenu  =
               new List( "Valitse pallon muoto",
                         List.IMPLICIT,
                         pallon_muotovaihtoehdot,
                         null ) ;
```

Tarkoitus on että tämä muodonvalintamenu heitetään midletin näyttöön heti kun midletti käynnistyy, ja sitten kun käyttäjä on valinnut pallolle muodon, heitetään näyttöön nykyinen värinvalintamenu. Tämä muutos ei vaadi uusien komentojen lisäämistä midlettiin jos ohjelma rakennetaan siten että pallon muodon voi asettaa ainoastaan ohjelman alussa.

Uudelle List-oliolle täytyy kuitenkin muistaa laittaa kuuntelija ja kuuntelijana voi toimia "tämä" midlettiluokka. Pallon muodon valintamenusta tehtävään valintaan voidaan reagoida samassa metodissa kuin reagoidaan värinvalintamenusta tehtäviin valintoihin. Tällöin voidaan seuraavaan tapaan tutkia mitä näytössä on ja reagoida sen mukaan

```
else if ( annettu_komento  ==  List.SELECT_COMMAND )
{
   if ( display_content  ==  pallon_muodon_valintamenu )
   {
      ...
```

commandAction()-metodille tulee siis parametrina tieto siitä mitä näytössä on komennonantohetkellä, ja metodi voi ottaa tämän huomioon komentoja käsitellessään.

Halutusta pallon muodosta pitää jotenkin saada tieto midlettiluokasta LiikkuvaPalloCanvas-luokkaan jossa pallo varsinaisesti piirretään. Eräs tapa tehdä tämä tiedonsiirto on kirjoittaa LiikkuvaPalloCanvas-luokkaan seuraavalla tavalla alkava metodi

```
public void muuta_pallon_muoto(
                        String annettu_muodon_kuvaus )
{
   if ( annettu_muodon_kuvaus.equals(
                                  "Pieni pyorea pallo" ) )
   {
      pallon_leveys   =  40 ;
      pallon_korkeus  =  40 ;
   }
   else if ( annettu_muodon_kuvaus.equals(
                                     "Iso pyorea pallo" ) )
   {
      ...
```

Tässä metodissa oletetaan että LiikkuvaPalloCanvas-luokassa on datakentät

```
int  pallon_leveys   =  40 ;
int  pallon_korkeus  =  40 ;
```

jotka sisältävät pallon kokotiedot. Soikeassa "pallossa" näiden datakenttien arvot eivät luonnollisestikaan voi olla samat.

6. Lisää ohjelmaan ominaisuus että se antaa hälytyksen jos pallon yrittää viedä näyttöalueen ulkopuolelle. Hälytyksen voi antaa esim. silloin kun pallon keskipiste viedään canvasin reunojen yli.

Anna hälytys käyttämällä hyväksi Alert-luokkaa. Tähän löytyy vinkkiä AlertGaugeDemoMIDlet.java -ohjelman loppupuolelta.

Tässä(kin) osatehtävässä muodostuu ongelmaksi se että Canvas-pohjaisen luokan keyPressed() tms. metodista pitää päästä käsiksi midlettiluokassa määriteltyyn Display-olioon. Asia voidaan ratkaista esim. siten että viittaus ko. Display-olioon siirretään konstruktorin parametrina LiikkuvaPalloCanvas-luokkaan.

HARJOITUKSIA OHJELMALLA ClockMIDlet.java

TEHTÄVÄ 1. Muuta ohjelmaa siten että kellon sekuntiviisari piirretään katkoviivalla. Viivan piirtotyylin voi muuttaa Graphics-luokan setStrokeStyle()-metodilla, jolle voidaan antaa parametrina Graphics luokassa määritelty vakio DOTTED. Tämän osatehtävän suoritus vaatii yhden rivin koodia sopivaan paikkaan ohjelmassa.

TEHTÄVÄ 2. Tee ClockCanvas-luokkaan Asetukset-valikko ja laita tähän valikkoon aluksi mahdollisuus muuttaa kellon tausta mustaksi tai valkoiseksi. (Jos kellon tausta on musta, tulee itse kello tällöin piirtää valkoisena.) Valikon rakentamiseen voit katsoa mallia ohjelmasta SatunnaislukujaMIDlet.java. Tällainen valikko täytyy rakentaa Form-luokkaa käyttäen. Form-olioon voidaan sitten kiinnittää ChoiceGroup-olio joka sisältää eksklusiiviset valinnat mustalle ja valkoiselle kellon taustalle. ChoiceGroup-olio voidaan luoda esim. seuraavanlaisella lauseella:

```
ChoiceGroup  kellon_taustan_valinta  =
               new  ChoiceGroup( "KELLON TAUSTA:",
                                  Choice.EXCLUSIVE,
                                taustavaihtoehdot, null );
```

Tarvittava Form-olio voidaan luoda seuraavanlaisella lauseella:

```
Form  asetukset_form  =  new  Form( "ASETUKSET" ) ;
```

Tarkoitus on että Form-perustainen valikko rakennetaan nimenomaan ClockCanvas-luokassa. Tätä varten tarvitaan ClockCanvas-luokkaan datajäseniksi myös pari Command-oliota, joilla Form-olio heitetään näyttöön ja poistetaan näytöstä. Command-oliot voidaan määritellä esim. seuraavasti:

```
Command asetukset_komento  =
       new  Command( "Asetukset", Command.SCREEN, 1 );
Command valmis_komento  =
       new  Command( "Valmis", Command.SCREEN, 1 ) ;
```

Jotta näitä olioita voidaan käyttää, ClockCanvas-luokan tulee toteuttaa CommandListener-rajapinta ja siellä täytyy olla commandAction()-metodi joissa reagoidaan komentoihin.

Koska ClockMIDlet-luokassa on jo ennestään komentoihin reagoiva CommandAction-metodi, ClockCanvas-luokan saman nimistä metodia ei saada helposti toimintaan muuten kuin siten että ClockMIDlet-luokan kyseinen metodi "rampautetaan". Tämä saadaan aikaiseksi siten että kommentoidaan kuuntelijan asettelu ClockMIDlet-luokan startApp()-metodista pois seuraavaan tapaan:

```
//       clock_canvas.setCommandListener( this ) ;
```

Tämän seurauksena Exit-komento lakkaa toimimasta mutta se ei tässä pitäisi haitata mitään.

Tässä tehtävässä kannattaa edetä siten että saat ensin Asetukset-valikon tulemaan esiin ja poistumaan näkyvistä, ja vasta tämän jälkeen kannattaa Asetuksissa asetetut asiat pistää toimintaan itse kellossa. Ohjelmasta kannattaa poistaa myös Full Screen -ominaisuus näitä harjoituksia tehtäessä.

Jotta saat kellon taustan piirtymään mustana tai valkoisena sen mukaan mitä ChoiceGroup-oliossa on valittu, tulee valinnan tila lukea ClockCanvas-luokan paint()-metodissa ja valita tämän perusteella taustan väri ja piirtoväri. Valinnan tila voidaan tutkia isSelected()-metodilla seuraavaan tapaan

```
if ( kellon_taustan_valinta.isSelected( 0 ) )
{
   //  ChoiceGroupin ensimmäinen valinta on tehty
```

TEHTÄVÄ 3. Tässä on tehtävänä muuntaa kello sellaiseksi että siitä tulee herätyskello. Herätyskellotoiminnon toteuttamiseksi tulee olla mahdollisuus asettaa herätysaika, ja lisäksi tulee olla mahdollisuus aktivoida/deaktivoida herätystoiminto.

Herätysajan asettamiseksi kannattaa käyttää valmista luokkaa nimeltä DateField. Tällainen DateField-luokan olio voidaan kiinnittää Form-olioon siten että se on näkyvillä Asetukset-valikossa joka tehtiin edellisessä osatehtävässä. Käytettävä DateField-olio voidaan luoda seuraavalla tavalla:

```
DateField  wakeup_time_field  =
           new DateField( "WAKE-UP TIME:",
                          DateField.DATE_TIME ) ;
```

Tällä lauseella DateField-olio luodaan siten että sekä päivämäärä että aika voidaan asettaa. Jotta saat DateField-olion käyttöösi, se tulee alustaa ja lisätä Asetukset-valikkoon seuraavanlaisilla lauseilla ClockCanvas-luokan konstruktorissa:

```
Calendar default_wakeup_time  =
                    Calendar.getInstance() ;
wakeup_time_field.setDate(
                    default_wakeup_time.getTime() ) ;
asetukset_form.append( wakeup_time_field ) ;
```

Ennenkuin jatkat tästä tehtävän tekemistä kannattaa varmistaa että DateField todella tulee näkyviin Asetukset-formiin.

Herätystoiminnon aktivoimiseksi voit lisätä Asetukset-valikkoon uuden ChoiceGroup-valinnan, joka voidaan määritellä seuraavilla lauseilla:

```
String[]  alarm_activated_choice  =
                    { "Herätys aktivoitu" } ;

ChoiceGroup  alarm_activation_choice_group  =
       new  ChoiceGroup( "HERÄTYKSEN AKTIVOINTI:",
                          Choice.MULTIPLE,
                         alarm_activated_choice, null ) ;
```

Tässä ChoiceGroup-valintaryhmässä on vain yksi valinta joka voidaan enabloida tai disabloida. Tarkoitus on että herätyskellon käyttäjän tulee aktivoida herätys tämän ChoiceGroupin avulla sen jälkeen kun hän on asettanut herätysajan DateField-kentän kautta.

Tässäkin kannattaa varmistua ensin että ChoiceGroup tulee näkyviin Asetukset-formiin ennenkuin jatkaa tehtävän tekemistä.

Herätyshälytyksen antamista voidaan kontrolloida uudella if-rakenteella paint()-metodin lopussa. Ensin täytyy testata onko herätystoiminto aktivoitu, ja jos se on aktivoitu, täytyy testata pitääkö sillä hetkellä antaa hälytys. Tarvittava if-rakenne voi alkaa seuraavalla tavalla:

```
if ( alarm_activation_choice_group.isSelected( 0 ) )
{
   Calendar wakeup_time  =  Calendar.getInstance() ;

   wakeup_time.setTime( wakeup_time_field.getDate() ) ;

   int wakeup_hour   = wakeup_time.get( Calendar.HOUR_OF_DAY ) ;
   int wakeup_minute = wakeup_time.get( Calendar.MINUTE ) ;

   if ( ...
```

Yllä olevista lauseista nähdään kuinka haluttu herätysaika luetaan DateField-oliosta. Tämä toimenpide on hiukan monimutkainen koska metodi DateField-luokan getDate()-metodi palauttaa Date-olion joka sitten talletetaan Calendar-olioon jotta päästään lukemaan haluttuja aikakenttiä (herätystuntia ja herätysminuuttia).

Ohjelman pitäisi tulostaa näytölle "Herätys aktivoitu" silloin kun herätys on aktivoitu mutta kello ei ole vielä saavuttanut herätysaikaa. Ohjelman pitäisi tulostaa "OLISITKO KILTTI JA HERÄISIT!" sitten kun herätysaika on saavutettu. Kello toimii riittävän tarkasti jos ohjelma vertailee nykyistä tuntia ja minuuttia asetetun herätysajan tuntiin ja minuuttiin.

Huomaa että tämä herätyskello on "visuaalinen herätyskello" jota nukkujan pitää koko ajan katsella jotta hän tietää milloin on aika herätä.

TEHTÄVÄ 4: Jos intoa piisaa, herätyskelloon voi vielä rakentaa ominaisuuden, että herätyksen saa pois päältä mitä tahansa näppäintä painamalla. (Tämmöistä ominaisuutta en ole kyllä itsekään toteuttanut.)

Hyvässä herätyskellossa olisi vielä äänellä toimiva herätys, mutta äänien käyttöä emme ole tällä kurssilla toistaiseksi opiskelleet.

VANHA VERSIO OSATEHTÄVÄSTÄ 3: Tee ohjelmasta herätyskello lisäämällä edellisessä osatehtävässä rakennettuun Asetukset-valikkoon sopivat tekstikentät joiden avulla voidaan herätystunti ja herätysminuutti asettaa. (Tekstikenttien eli TextField-olioiden käytöstä on esimerkki SumMIDlet.java-ohjelmassa.) Lisäksi Asetukset-valikkoon pitäisi lisätä ChoiceGroup-olio, jonka avulla herätystoiminto aktivoidaan. Tämä ChoiceGroup-olio voidaan luoda esim. seuraavasti

```
String[]  heratyksen_aktivointiteksti  =
                          { "Heratys aktivoitu" } ;

ChoiceGroup  heratyksen_aktivointi  =
         new  ChoiceGroup( "HERATYKSEN AKTIVOINTI",
                           Choice.MULTIPLE,
                           heratyksen_aktivointiteksti,
                           null ) ;
```

Tämmöisessä ChoiceGroupissa eli "valintaryhmässä" on vain yksi valittava kohta.

Herätystunnin ja herätysminuutin valintaa varten kannattaa laittaa kaksi erillistä tekstikenttää joihon voi syöttää vain numeerista tietoa. Toiseen kenttään kirjoitetaan siis tunti ja toiseen minuutti.

Herätystoiminnot voidaan hoitaa esim. paint()-metodin lopussa ohjelmarakenteella joka alkaa esim. seuraavasti

```
if ( heratyksen_aktivointi.isSelected( 0 ) )
{
  String heratystunti_stringina  =  heratystunti_tekstikentta.getString();
   String heratysminuutti_stringina  =  heratysminuutti_tekstikentta.
                                                    getString() ;
   if ( heratystunti_stringina.length() != 0 &&
        heratysminuutti_stringina.length() != 0 )
   {
      int heratystunti  =  Integer.parseInt( heratystunti_stringina ) ;
      ...
```

Herätystoimintojen pitää toimia siten että silloin kun herätys on aktivoitu, mutta kello ei ole vielä saavuttanut herätysaikaa, kellon alla lukee näytöllä "Herätys aktivoitu". Kun kello saavuttaa herätystilassa herätysajan, kellon alle tulee em. tekstin tilalle teksti "HERÄÄ JO!". Tämä on siis visuaalinen herätyskello jota nukkujan pitää koko ajan katsella jotta hän tietää milloin pitää herätä. Jos herätys on aktivoitu, mutta herätysajan (herätysminuutin ja herätystunnin) sisältävissä tekstikentissä on ainakin toisessa pelkkä tyhjä stringi, kellon alla tulee lukea teksti "Herätysaika ei hyväksyttävä".

HARJOITUKSIA OHJELMALLA PelikortitMIDlet.java

Seuraavissa harjoituksissa tarvitsee muuttaa ainoastaan PelikortitCanvas-luokkaa.

1. Muuta ohjelma sellaiseksi että korttipakka on valmiiksi sekoitettuna heti kun midletti käynnistyy.

2. Lisää ominaisuus että ohjelma laskee Uusi kortti -komennolla pakasta otettujen korttien arvojen summan. Kortin maa ei tässä summan laskennassa ole merkityksellinen. Kortti-luokassa on metodi jolla kortin arvo voidaan lukea. Otettujen korttien summa näytetään viimeiseksi otetun kortin alapuolella näytöllä. (Tässä vaiheessa ohjelmasta voi poistaa ominaisuuden että maan vaihtumista seurataan kortteja otettaessa.) Tässä tarvitaan PelikortitCanvas-luokkaan uusi datakenttä kuten esim:

```
int pelaajan_korttien_summa  =  0 ;
```

Jo tässä vaiheessa kannattaa käyttää nimeä jossa esiintyy sana ’pelaaja’, koska seuraavassa osatehtävässä ohjelmasta kehitetään ihan oikea peli.

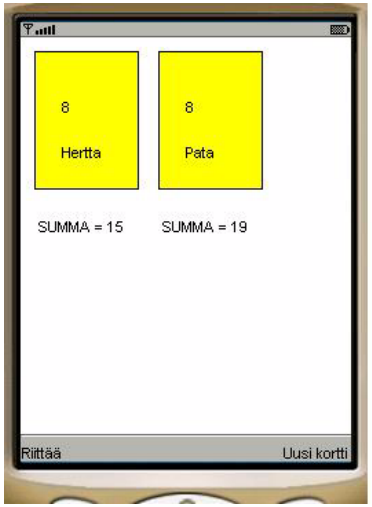
3. Tee ohjelmasta alkeellinen venttipeli siten että muutat Sekoita pakka -komennon Riittää-komennoksi. Tarkoitus on että kun käyttäjä on ensin ottanut kortteja Uusi kortti -komennolla, hän voi painaa Riittää-komentonappia, minkä jälkeem midletti alkaa ottamaan kortteja itselleen. Uudistettu komento voidaan määritellä esim. seuraavasti:

```
Command riittaa_komento = new Command( "Riittää", Command.EXIT, 1) ;
```

Kun midletti eli "kone" alkaa ottamaan kortteja ne pitäisi näyttää samaan tapaan kuin pelaajan kortit näytetään. Koneen viimeisin kortti voidaan näyttää pelaajan kortin oikealla puolella, ja myöskin koneen korttien numeroarvojen summa pitäisi näyttää ja esittää koneen kortin oikealla puolella. Tässä tarvitaan PelikortitCanvas-luokkaan lisää datakenttiä jotka voisivat olla esim. seuraavanlaiset:

```
Kortti koneen_viimeisin_kortti  =  null ;
int koneen_korttien_summa    =  0 ;
```

Näytöllä koneen viimeisin kortti ja korttien summa voisivat näkyä pelaajan kortin oikealla puolella seuraavaan tapaan:



paint()-metodissa täytyy tehdä niin että koneen korttia ei piirretä jos ko. korttia ei ole vielä jaettu eli olioviittaaja (koneen_viimeisin_kortti) on null.

Tässä tehtävässä voisi edetä siten että aluksi otetaan koneelle vain yksi kortti Riittää-komennon antamisen jälkeen. Näin varmistutaan siitä että koneen korttien ottaminen todella onnistuu ja ne tulevat näkyviin ruudulle.

Koneen eli midletin pitäisi ottaa pakasta useita kortteja siten että se pääsisi mahdollisimman lähelle venttiä eli arvoa 21. Koneen korttien pakasta ottamisessa ja näyttämisessä muodostuu ongelmaksi koneen nopeus. Koneen ottamat kortit tulevat näkyviin näytölle niin nopeasti että pelaaja ei ehdi nähdä kuin viimeisimmän kortin. Tämä ongelma voidaan ratkaista siten että koneen kortit otetaan erillisen säikeen avulla, ja säie pitää taukoa korttien välillä. Tuon tauon ansiosta pelaaja ehtii nähdä kortit näytöllä. Tarvittava säie voidaan saada aikaan siten että ensin määritellään seuraavat datajäsenet

```
Thread  koneen_kortit_ottava_saie  =  null ;
boolean  saietta_on_suoritettava  =  false ;
```

Sitten luodaan Thread-olio Riittää-komentoon reagoivassa metodissa seuraavaan tapaan:

```
koneen_kortit_ottava_saie  =  new  Thread( this );
koneen_kortit_ottava_saie.start() ;
```

Säiettä varten tarvitaan PelikortitCanvas-luokkaan run()-metodi, jossa kortit otetaan pakasta siten että korttien välillä pidetään Thread.sleep()-metodin avulla pieni tauko. Tarvittavan run()-metodin rakenne on esitetty seuraavalla sivulla. Kun säikeen toteuttava run()-metodi päättyy, säiekin lakkaa olemasta olemassa.

```
public void run()
{
   saietta_on_suoritettava  =  true ;

   while ( saietta_on_suoritettava  ==  true )
   {
      try
      {
         Thread.sleep( 1500 ) ;  // 1.5 sekunnin tauko

         //  Tänne pitäisi keksiä jotain ohjelmakoodia.

         //  Alla olevan if-rakenteen tilalle voisit kehittää
         //  jonkin vaikeammman ehdon korttien ottamiselle.
         if ( koneen_korttien_summa  >=  17  )
         {
            saietta_on_suoritettava  =  false ;
         }

         repaint() ;
      }
      catch ( InterruptedException vangittu_poikkeus )
      {
      }
   }
   koneen_kortit_ottava_saie  =  null ;
}
```

4. Lisää ohjelmaan ominaisuus että uusi peli voidaan aloittaa painamalla puhelimen numeronäppäintä 1. Seuraavasta keyPressed()-metodista saattaa olla apua tämän ominaisuuden toteutuksessa:

```
public void keyPressed( int nappaimen_koodi )
{
   if ( nappaimen_koodi ==  '1' )
   {
      if ( koneen_kortit_ottava_saie  !=  null )
      {
         //  Säie on käynnissä ja elossa. Tapetaan se!

         koneen_kortit_ottava_saie.interrupt() ;
         koneen_kortit_ottava_saie  =  null ;
         saietta_on_suoritettava    =  false ;
      }

      viimeksi_otettu_kortti  =  null ;
      koneen_viimeisin_kortti =  null ;

      pelaajan_korttien_summa  =  0 ;
      koneen_korttien_summa    =  0 ;
   }

   repaint() ;
}
```

5. Ota mallia ohjelmasta TickerDemoMIDlet.java ja pistä ns. Ticker-teksti pyörimään näytölle silloin kun peli on päättynyt. Tekstin tulee luonnollisesti kertoa voittiko pelin kone vai pelaaja. Ticker-teksti pitää poistaa sitten kun uusi peli alkaa.

HARJOITUKSIA OHJELMALLA WalkingMarioMIDlet.java

WalkingMarioMIDlet.java on ohjelma jossa käytetään hyväksi Sprite-luokalla tehtyä animaatiota. Sprite-luokan olion ’sisään’ laitetaan oliota luotaessa kuva joka sisältää eräänlaisia filmiruutuja eli freimejä (frame). Näitä filmiruutuja voidaan sitten esim. nextFrame()-metodin avulla kelata läpi ohjelmassa, mikä tuottaa animaation liikkeestä. Sprite-olion sisältämistä filmiruuduista voidaan valita kelattavaksi vain tietyt ruudut. Ohjelmassa WalkingMarioMIDlet.java käytetään eri ruutuja esim. sen mukaan ollaanko liikkumassa vasemmalle vai oikealle.

Sprite-luokan käytöstä on esimerkkinä myös ohjelma SpriteDemoMIDlet.java.

Tee WalkingMarioMIDlet.java -ohjelmaan seuraavat muutokset. Huomaa että kuvatiedosto mario_frames.png tulee laittaa projektin resurssikansioon.

Harjoitus 1:

Muuta ohjelmassa tallusteleva Mario sellaiseksi että sitä voidaan ohjata nuolinäppäimillä.

- Tarvittavan keyPressed()-metodin aihion voit kopioida esim. MovingBallMIDlet.java-ohjelmasta.
- Voit laittaa Mario-luokkaan seuraavat metodit joita kutsutaan silloin kun nuolinäppäimiä painellaan:

```
public void start_moving_right()
{
   mario_sprite.setFrameSequence( frames_to_walk_right ) ;
   mario_state = MOVING_RIGHT ;
}

public void start_moving_left()
{
   mario_sprite.setFrameSequence( frames_to_walk_left ) ;
   mario_state = MOVING_LEFT ;
}

public void start_moving_up()
{
   mario_sprite.setFrameSequence( frames_to_walk_up ) ;
   mario_state = MOVING_UP ;
}

public void start_moving_down()
{
   mario_sprite.setFrameSequence( frames_to_walk_down ) ;
   mario_state = MOVING_DOWN ;
}
```

Harjoitus 2:

Muuta ohjelma sellaiseksi että Mario saadaan pysähtymään kun painetaan jotain muuta näppäintä kuin nuolinäppäintä. Tarkoitus on että liike jatkuu kun pysähtymisen jälkeen painetaan jotain nuolinäppäintä.

Seuraavat uudet datajäsenet voivat tässä tehtävässä olla hyödyllisiä Mario-luokassa:

```
int[] frame_standing_down = { 16 } ;
int[] frame_standing_right = { 10 } ;
int[] frame_standing_up = { 4 } ;
int[] frame_standing_left = { 22 } ;

static final int STOPPED_IN_DIRECTION_DOWN  = 4 ;
static final int STOPPED_IN_DIRECTION_RIGHT = 5 ;
static final int STOPPED_IN_DIRECTION_UP    = 6 ;
static final int STOPPED_IN_DIRECTION_LEFT  = 7 ;
```

Seuraavasti alkava metodi voi olla hyödyllinen Mario-luokassa:

```
public void stop_moving()
{
   if ( mario_state == MOVING_RIGHT )
   {
      mario_sprite.setFrameSequence( frame_standing_right ) ;
      mario_state = STOPPED_IN_DIRECTION_RIGHT ;
   }
   else if ( mario_state == ...
```

Harjoitus 3:

Lisää ohjelmaan uusi Sprite-olio joka pyörittää asteroidia näytön keskellä. Voit tehdä tämän Sprite-olion WalkingMarioCanvas-luokan datajäseneksi. Kansiosta http://www.naturalprogramming.com/javamidlets/ löydät tiedoston asteroid_sprites.png joka sisältää 72 freimiä asteroidinpyörityskuvaa.

Harjoitus 4:

Muuta ohjelma sellaiseksi että edellisessä tehtävässä rakennettu näytön keskellä pyörivä asteroidi räjähtää kun Mario kävelee siihen pahki.

- Voit lisätä Mario-luokkaan seuraavan metodin jolla Mario-olion ja Sprite-olion yhteentörmäys voidaan tutkia:

```
public boolean collides_with( Sprite given_sprite )
{
   return mario_sprite.collidesWith( given_sprite, true ) ;
}
```

- Räjähdystä varten voit luoda uuden Sprite-olion joka otetaan käyttöön sitten kun yhteentörmäys on tapahtunut. Edellä mainitusta kansiosta löydät tiedoston explosive_sprites.png joka sisältää 72 ruutua ’räjähdyselokuvaa’
- Tarkoitus on että räjähdystä kuvaava Sprite ’toimii’ vain yhden kierroksen ja katoaa sitten näkyvistä. Sprite-luokassa on metodi jolla voidaan tutkia milloin Spriten viimeinen ruutu (frame) on käytössä. Seuraavanlaiset asteroidin tilaa kuvaavat datajäsenet voivat olla hyödyksi WalkingMarioCanvas-luokassa:

```
static int ASTEROID_ALIVE_AND_WELL = 0 ;
static int ASTEROID_IS_EXPLODING   = 1 ;
static int ASTEROID_HAS_EXPLODED   = 2 ;

int asteroid_state = ASTEROID_ALIVE_AND_WELL ;
```

HARJOITUKSIA OHJELMALLA TekstiNetistaMIDlet.java

1. Muuta ohjelma aluksi sellaiseksi että se lukee käynnistyttyään automaattisesti valuuttakurssitiedoston netistä ja näyttää sen TextBoxissa. Alussa siis näytön sisältönä on tekstiboxi_naytolla ja tekstin vastaanottava säie tulee laittaa käyntiin heti startApp()-metodissa. Myöskin käytettävä URL kannattaa heti määritellä esim. seuraavalla tavalla

```
String  valittu_nettiosoite  =
    "http://www.oamk.fi/~karil/valuuttakurssit_testaukseen.xml" ;
```

Seuraavissa osatehtävissä on tarkoitus tehdä tästä midletistä valuuttamuunnin, joka osaa käyttää netistä luettua viimeisintä valuuttakurssia. Voit kokeilla myös seuraavaa osoitetta, joka on Euroopan Keskuspankin valuuttakurssitiedoston osoite:

```
// "http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml" ;
```

2. Muuta ohjelma aluksi sellaiseksi että se näyttää vain yhden valitun valuutan kurssin TextBoxissa. Tässä täytyy siis "parseroida" eli etsiä yhden valuutan kurssi Suomen Pankista saatavasta tekstitiedostosta. Kannattaa ensin varmistaa että haluttu valuuttakurssi saadaan talteen, ja vasta sitten voi ryhtyä seuraavan osatehtävän mukaisesti rakentamaan varsinaista valuuttamuunninta.

Valitun valuutan nimen ja muunnoskertoimen tallentamiseksi midlettiluokkaan kannattaa lisätä seuraavat datajäsenet ja antaa niille oletusarvot:

```
double  valuuttamuunnoskerroin  =  1.2217 ;
String  valittu_valuutta  =  "USD" ;
```

Tässä vaiheessa kannattaa jo muuttaa käytetty List-olio valuuttojen nimiä sisältäväksi listaksi seuraavaan tapaan (Valuuttojen "niminä" käytetään tässä samoja stringejä jotka löytyvät valuuttakurssisivulta. Näin nimeä on helpompi etsiä XML-tekstitiedostosta.)

```
String[]  valittavat_valuutat  =
 {
   "USD",  "JPY",   "CAD",   "AUD",  "NZD",
   "ISK",  "SEK",   "NOK",   "RUB",  "GBP",
 } ;

List  valuutan_valintamenu  =  new  List( "VALITSE VALUUTTA:",
                                          List.IMPLICIT,
                                          valittavat_valuutat,
                                          null ) ;
```

Lisäksi kannattaa muuuttaa käytössä oleva komento valuutan vaihtokomennoksi seuraavaan tapaan.

```
Command  valuutan_vaihtokomento = new Command( "Vaihda valuutta",
                                               Command.SCREEN, 1 ) ;
```

Kun käyttäjä antaa tämän komennon, heitetään valuutanvaihtomenu näyttöön. Kun käyttäjä valitsee uuden valuutan listalta, valuuttakurssit sisältävä teksti haetaan netistä ja valitun valuutan kurssi parseroidaan saadusta tekstistä. (Ensimmäisellä kerralla, kun midletti käynnistyy, voidaan hakea kurssi sille valuutalle joka listalta löytyy automaattisesti.)

Valuuttakurssin etsintä tekstin seasta voidaan suorittaa metodissa vastaanota_teksti_netista(). Kyseinen metodi etsii valitun valuutan kurssitekstin ja pistää sen näkymään tekstiboksiin.

Valittu valuutta voidaan lukea listalta seuraavaan tapaan

```
int  valuutan_indeksi_listalla  =
             valuutan_valintamenu.getSelectedIndex() ;

valittu_valuutta  =
         valuutan_valintamenu.getString( valuutan_indeksi_listalla ) ;
```

Valittu valuutta voidaan parseroida Suomen Pankista saadusta tekstistä String-luokan metodeita käyttäen seuraavaan tapaan

```
String luettu_teksti_stringina  =  luetut_merkit.toString() ;

int  valuutan_paikka_tekstissa  =
                luettu_teksti_stringina.indexOf( valittu_valuutta ) ;

//  Tässä oletetaan että valuuttojen "nimet" on 3-merkkisiä,
//  kuten esim. USD, ja "nimeä" seuraa XML-koodissa teksti
//  "rate=" jonka jälkeen valuuttakurssi on annettu heittomerkeissä.
//  Yksittäinen kurssi näkyy siis seuraavaan tapaan:

//  <Cube currency='USD' rate='1.4090'/>
```

```
int  valuuttakurssin_alkukohta  =
      luettu_teksti_stringina.indexOf( "rate=\'",
                                        valuutan_paikka_tekstissa ) + 6;

int  valuuttakurssin_loppukohta  =
        luettu_teksti_stringina.indexOf( "\'",
                                          valuuttakurssin_alkukohta );

String valitun_valuutan_kurssi_stringina  =

         luettu_teksti_stringina.substring(
               valuuttakurssin_alkukohta, valuuttakurssin_loppukohta );

valuuttamuunnoskerroin  =  Double.parseDouble(
                               valitun_valuutan_kurssi_stringina ) ;
```

Voit näyttää pankista saadun valuuttakurssin aluksi jo olemassaolevassa TextBoxissa.

3. Valuuttamuuntimen rakentamiseksi midlettiin pitäisi liittää formi, jossa on kaksi tekstikentää. Tämä formi korvaa ohjelmassa aiemmin käytetyn TextBoxin. Kun jommassa kummassa tekstikentässä olevaa valuuttamäärää muutetaan, tulee toiseen tekstikenttään päivittyä valuuttamäärä käytössä olevan kurssin mukaan. Tässä kannattaa katsoa mallia ohjelmasta SumMIDlet.java, jossa tekstikentät on käytössä.

Formi ja siihen kiinnitettävät tekstikentät voidaan määritellä seuraavaan tapaan:

```
Form  valuutan_muunnos_form  =  new  Form( "VALUUTTAMUUNNIN" ) ;

TextField  eka_valuutta_tekstikentta  =  new TextField( "EURO", "1", 10,
                                                        TextField.NUMERIC ) ;

TextField  toka_valuutta_tekstikentta  =  new TextField( "USD", "", 10,
                                                        TextField.NUMERIC ) ;
```

Tekstikenttien muutoksiin reagoiva metodi voi alkaa seuraavasti:

```
public void itemStateChanged( Item item_which_changed_state )
{
   String eka_valuuttamaara_stringina  =
                                 eka_valuutta_tekstikentta.getString() ;
   String toka_valuuttamaara_stringina  =
                                 toka_valuutta_tekstikentta.getString() ;

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

   if ( toka_valuuttamaara_stringina.length()  ==  0 )
   {
      toka_valuuttamaara_stringina  =  "0" ;
   }
```

```
int eka_valuuttamaara, toka_valuuttamaara ;

if ( item_which_changed_state  ==  eka_valuutta_tekstikentta )
{
   eka_valuuttamaara  =
                Integer.parseInt( eka_valuuttamaara_stringina ) ;

   toka_valuuttamaara  =
             (int) ( valuuttamuunnoskerroin * eka_valuuttamaara ) ;

   toka_valuutta_tekstikentta.setString( "" + toka_valuuttamaara ) ;
}
else if ( item_which_changed_state  ==  toka_valuutta_tekstikentta )
{
   ...
```

Muista että midlettiluokka toteuttaa erään rajapinnan silloin kun siellä on tämä metodi. Lisäksi midlettiluokan täytyy rekisteröityä tekstikenttämuutosten kuuntelijaksi.

4. Lisää vielä käytettyyn formiin kolmas UNEDITABLE-tekstikenttä jossa näytät kulloinkin käytetyn valuuttakurssin. Lisäksi voit muuttaa Form-olion otsikkoa siten että se näyttää kulloinkin käytössä olevan valuutan. Lisäksi pitäisi toisen tekstikentän label muuttaa aina kulloinkin käytössä olevan valuutan mukaiseksi. (Form-luokkaan on periytynyt Displayable-luokasta metodi jolla näytettävän olion otsikko voidaan asettaa. Samoin on TextField-luokkaan periytynyt Item-luokasta metodi jolla itemin label voidaan asettaa.)

Näiden muutosten jälkeen valuuttamuuntimen käyttöliittymän pitäisi näyttää seuraavanlaiselta:

Mahdollisia myöhemmin kehiteltäviä harjoituksia (Ei tehdä toistaiseksi).

List-luokkaan perustuva valuuttalista voitaisiin tehdä netistä saatavan valuuttakurssit sisältävän XML-tekstitiedoston perusteella siten että tekstitiedostosta parseroitaisiin kaikki valuuttjen nimet ja ne pantaisiin listan elementeiksi.

Kunkin valuutan kurssit voitaisiin tallettaa taulukkoon. Näin listan haku pankista tarvitsisi suorittaa vain kerran.

Valuuttakurssit voitaisiin tallettaa Record Storeen ja valuuttakurssien päivityksestä voitaisiin tehdä erikoistoiminto. Uutta valuuttakurssilistaa hakiessa ohjelma voisi ilmoittaa tapahtuneista valuuttakurssien muutoksista. Ohjelmasta voisi kehittää myös valuuttakurssien seuraajasovelluksen joka (automaattisesti) tarkistaisi päivittäin valuuttakurssit.

Tekstikenttiin voisi olla mahdollista syöttää desimaalipisteellisiä lukuja. (Onnistunee varsin helposti kun TextFieldin tyyppi määritellään sopivasti.)

XML-tiedostojen parserointiin saattaa löytyä parempia tapoja kuin tässä harjoituksessa esitetty.

HARJOITUKSIA OHJELMALLA BluetoothJokesMIDlet.java

BluetoothJokesMIDlet.java on ohjelma jota voidaan ajaa joko serverinä tai clienttinä. Serverinä toimiessaan ohjelma odottaa että jossain toisessa puhelimessa clienttinä ajettava ohjelma ottaa Bluetooth-yhteyden ja pyytää serveriltä vitsiä. Kun serveri on saanut vitsipyynnön, se lähettää satunnaisesti valitun vitsitekstin clientille, ja palaa uudestaan odottamaan että client pyytää vitsiä.

Tämän ohjelman käyttämiseksi tarvitaan kaksi puhelinta. Sun Java Wireless Toolkitissä ohjelmaa voidaan käyttää kun käynnistetään kaksi puhelinsimulaatiota ja valitaan toinen serveriksi ja toinen clientiksi.

Harjoitus 1:

Lisää serveripuolelle uusi vitsi taulukkoon ja totea ohjelman toiminnan muuttuminen.

Harjoitus 2:

Muuta serveripuolen toiminta sellaiseksi että vitsejä ei lähetetä satunnaisessa järjestyksessä, vaan lähetettävät vitsit otetaan vitsitaulukosta alusta alkaen. Tarkoitus on että serveriä käyttävä client voi lukea kaikki serverin tarjoamat vitsit järjestyksessä. Serverin tulee toimia siten että jos kaikki vitsit taulukosta on lähetetty, se lähettää clientille tekstin "No more jokes available." Tämän jälkeen, jos client edelleen pyytää uutta vitsiä, serveri alkaa lähettää vitsejä taas taulukon alusta alkaen, eli serveri toimii kuten silloin kun vitsien pyytely aloitettiin. Tässä tarvitaaan serveriluokkaan uusi datajäsen, kuten esim.

```
int index_of_next_joke_to_be_sent = 0 ;
```

Harjoitus 3:

Muuta serveripuolen toiminta sellaiseksi että käyttäjällä on mahdollisuus lisätä uusi vitsi lähetettävien vitsien listaan. Koska kyseinen lista on tällä hetkellä alustettu taulukko, jonka koko ei voi kasvaa, kannattaa kyseinen taulukko ensin muuttaa Vector-luokkaa käyttäen dynaamiseksi taulukoksi. Kun vitsitaulukko on dynaaminen, voidaan uusi vitsi lisätä sen alkuun metodia insertElementAt() käyttäen. Kun uusi vitsi pannaan taulukon alkuun, se on aina ensin lähetettävä vitsi. Alussa vitsistringit voidaan lisätä dynaamiseen taulukkoon addElement()-metodin avulla.

Kun olet muuttanut ohjelman vitsitaulukon sellaiseksi että sinne voidaan lisätä uusia vitsejä, voit toteuttaa uuden vitsin kirjoitusmahdollisuuden siten että lisäät ohjelmaan uuden TextBox-olion johon vitsi voidaan kirjoittaa. Ohjelma voisi serverinä toimia siten että siinä on Soft key -näppäimessä komento "New Joke". Kun "New Joke" -komento annetaan, ohjelma pistää näyttöön uuden TextBox-olion. Tähän uuteen TextBox-olioon täytyy olla kiinnitettynä "Done"-komento, jonka antamisen jälkeen uuteen TextBoxiin kirjoitettu vitsi lisätään vitsilistaan ja näyttöön heitetään alkuperäinen TextBox johon viittaa textbox_for_messages.

TextBox- ja Command-olioiden käsittely voidaan pistää "pääluokkaan" eli MIDlet-pohjaiseen luokkaan. Uusi vitsi saadaan siirrettyä BluetoothJokesServer-luokan sisälle siten että sinne laitetaan metodi jolla uusi vitsi lisätään ja jota kutsutaan "pääluokasta" silloin kun uusi vitsi on kirjoitettu ja Done-komento annettu.

Seuraavat uudet datajäsenet MIDlet-pohjaiseen luokkaan voivat olla tämän osatehtävän ratkaisussa hyödyllisiä:

```
TextBox textbox_to_write_new_joke  =
              new TextBox( "WRITE NEW JOKE:", "",
                           512, TextField.ANY ) ;

Command new_joke_command  =
       new  Command( "New Joke", Command.SCREEN, 1 ) ;
Command done_command      =
       new  Command( "Done", Command.SCREEN, 1 ) ;
```

Harjoitus 4:

Lisää ohjelmaan ominaisuus, että clientinä toimiessaan ohjelmassa on käytössä "Latest Joke" -komento, joka antamalla pyydetään serveriltä viimeisin vitsi. Käytännössä tämä tarkoittaa että serveri menee alkutilaansa ja lähettää vitsilistalta (vitsitaulukosta) ensimmäisen vitsin. Tässä on tehtävä muutos myös serverin puolelle ja otettava käyttöön Bluetooth-yhteyden yli siirrettävä teksti jolla uusin vitsi pyydetään.

Harjoitus 5:

Ota serverin puolelle käyttöön maksimiaika vitsipyynnön odotteluun. Jos siis client ei ole pyytänyt uutta vitsiä tietyn ajan, esim. 2 minuuttia, kuluessa, serveri menee alkutilaan ja alkaa odottelemaan yhteydenottoa joltain clientiltä.

Harjoitus 6:

Muuta serveri sellaiseksi että se pystyy palvelemaan useita clientejä. Tässä täytynee tehdä jokaista clientiä varten oma säie.

Harjoitus 7:

Tee client-toimintoon ominaisuus, että client menee alkutilaan jos serveri ei lähetä uutta vitsiä tietyn ajan kuluessa vitsipyynnön lähettämisestä.

HARJOITUKSIA OHJELMALLA VideoPlayerMIDlet.java

Jotta VideoPlayerMIDlet.java ohjelman saa toimimaan tulee käytetty .mpg-tiedosto sijoittaa res-kansioon.

1. Lisää ohjelmaan aluksi valintalista joka mahdollistaa valitsemisen yhden monista videoista. Tarkoitus on että kun valintalistasta on valittu videotiedosto, siirrytään suoraan kyseisen videon katselutilaan. Tässä tehdävässä voit ottaa mallia esim. ohjelmasta TekstiNetistaMIDlet.java. Form-olion tilalle näyttöön pannaan tässä siis List-olio. Tarvittava List-olio saadaan aikaan esim. seuraavanlaisilla midlettiluokan datajäsenillä:

```
String[]  video_file_addresses  =
 {
   "resource:/test_mpeg_file_from_sun.mpg",
   "http://www.oamk.fi/~karil/media/david_bowie.mpg",
   "http://joku muu osoite tanne",
 } ;

List  video_selection_list  =  new  List( "CHOOSE VIDEO:",
                                          List.IMPLICIT,
                                          video_file_addresses,
                                          null ) ;
```

Kun videon osoite alkaa stringillä "resource:", ohjelma tajuaa sen automaattisesti res-kansiossa olevaksi tiedostoksi. "http:"-alkuiset osoitteet käsitellään puolestaan nettiosoitteina. Voit luonnollisesti käyttää omia videotiedostoja tässä harjoituksessa.

2. Lisää ohjelmaan Pause-toiminto siten että VideoCanvas-luokassa kiinnitetään vasempaan Soft Key -näppäimeen Pause-komento. Tarvittava Command-olio on jo ohjelmassa luotuna, mutta sitä ei ole vielä otettu käyttöön. Pistä Pause-komento aluksi käyttöön ja testaa että video todella pysähtyy sillä ja saat kuvan takaisin pyörimään Replay-komennolla.

Kun tiedät että Pause-komento todella toimii, muuta ohjelma sellaiseksi että kun Pause-komento on annettu, tuo Pause-komento poistetaan vasemmasta Soft Key -näppäimestä ja tilalle pannaan Play-komento. Kun taas Play-komentoa on painettu, pannaan sen tilalle uudelleen Pause-komento, jne. Voit määritellä tarvittavan Play-komennon samaan tapaan kuin Pause-komentokin on tehty. (Jätä toistaiseksi Replay-komento ohjelmaan.) Aivan samaan tapaan kuin komento voidaan lisätä addCommand()-metodilla se voidaan poistaa canvasista removeCommand()-metodilla.

3. Paranna vielä edellisen kohdan Pause/Play ominaisuutta siten että Pause-komento otetaan pois sitten kun videon ’soitto’ on päättynyt. Tämä saadaan aikaiseksi esim. playerUpdate()-metodissa jota kutsutaan aina automaattisesti kun jotain ’tärkeää’ tapahtuu. Esimerkiksi kun video on päättynyt, se saadaa ko. metodissa selville seuraavanlaisella if-rakenteella:

```
if ( event ==  PlayerListener.END_OF_MEDIA )
{
   // video on päättynyt
}
```

Kun Pause-toiminto poistetaan videon päätyttyä, tulee se lisätä takaisin sitten jos video käynnistetään uudelleen Replay-komennolla.

4. Ota selvää miten saat soitettua videon 'luupissa' 2 kertaa. Kannattaa tutkia mitä Player-rajapinnan dokumentaatiossa sanotaan.

5. Ota selville miten saat selville videon keston mikrosekunteina.

EXTRA-HARJOITUS: Tutustuminen Java-servletteihin

Java-servletit ovat http-palvelimilla suoriintuvia Java-ohjelmia. Niitä pystyy käytännössä kehittämään esim. NetBeans-työkalulla. NetBeans on IDE (Integrated Development Environment) josta on olemassa versio jonka mukana tulee tarvittava http-server jonka avulla servlettejä voi testata paikallisesti kehityskoneella.

Java-servletit saadaan kommunikoidaan jouhevasti Javalla tehtyjen client-sovellusten avulla. Servletit saadaan kommunikoimaan myös midlettien kanssa mutta tässä harjoituksessa client-sovelluksina käytetään perinteisiä Java SE -sovelluksia. Servlettien ja Java SE-sovellusten välinen tiedonsiirto voidaan tehdä siten että siirretään kokonaisia olioita. Tämä on mahdollista kun olioiden luokat toteuttavat `Serializable`-rajapinnan. Tällöin oliot serialisoituvat ja deserialisoituvat automaattisesti tiedonsiirron yhteydessä.

Kansiosta **http://www.naturalprogramming.com/javaservlets/netbeans/** löytyy kaksi Java-servlettiä EchoServlet ja MovingBallServlet NetBeans-projekteina.

Näiden kanssa kommunikoivat Java-sovellukset löytyvät kansiosta
http://www.naturalprogramming.com/javagui/communications/

Seuraavat harjoitukset liittyvät ohjelmaan **MovingBallServlet.java**, jonka alussa on kommenteissa selitetty kommunikointi **MovingBallManager.java**- ja **MovingBallViewer.java**-sovellusten kanssa.

Harjoitus 1:

Servlettejä voi myös ’katsoa’ selaimella kun selaimen osoitekenttään pannaan servletin osoite eli URL. Tällä hetkellä **MovingBallServlet.java**-ohjelmassa ei ole `doGet`()-metodia minkä vuoksi servlettiä selaimella ’katsottaessa’ se antaa virheilmoituksen. Tee siis `MovingBallServlet`-luokkaan kyseinen metodi ja pistä se ’tulostamaan’ jotain HTML-koodia. Katso tähän mallia ohjelmasta **EchoServlet.java**.

Harjoitus 2:

Ohjelmassa **EchoServlet.java** on ominaisuus että se pistää taltteen `ArrayList`-pohjaiseen taulukkoon ne stringit jotka se on ’kaiuttanut’ sen kanssa kommunikoivalle sovellukselle. Selaimella on mahdollista ’katsoa’ servlettiä jolloin se näyttää siihen mennessä kaiuttamansa stringit.

Tee `MovingBallServlet`-luokkaan sellainen ominaisuus että kun sitä selaimella ’katsotaan’ se näyttää pallon liikkeen viimeiset koordinaatit. Tämä voidaan tehdä esim. siten että luokkaan tehdään datajäsen

```
ArrayList<Ball> moved_balls = new ArrayList<Ball>() ;
```

johon talletetaan `MovingBallManager`-sovelluksen lähettämät pallot, aivan samaan tapaan kuin **EchoServlet.java**-ohjelmassa talletetaan kaiutettuja stringejä. Kyseisten pallojen koordinaatit voidaan sitten lukea `doGet()`-metodissa ja lähettää HTML-informaationa selaimelle.

Harjoitus 3:

Tee kokonaan uusi client-sovellus joka haluttaessa pyytää MovingBallServletiltä historiatiedon pallon liikkeistä. Tämän voit tehdä esim. ottamalla kopion ohjelmasta **MovingBallViewer.java** ja muuttamalla sen eri nimiseksi (esim. **MovingBallHistoryViewer.java**).

Tämän **MovingBallHistoryViewer.java**-ohjelman pitää lähettää jokin erityinen merkkistringi MovingBallServletille ja kun servlet huomaa saaneensa tuon erityisen stringin se lähettää clientille takaisin koko `ArrayList`-taulukon jossa on kaikki liikutellut pallot. Tässä siis muutetaan servletin `doPost()`-metodia. Koko `ArrayList`-taulukko voidaan lähettää clientille yhtenä oliona aivan samaan tapaan kuin yksittäinen `Ball`-oliokin lähetetään.

MovingBallHistoryViewer.java-ohjelma tulostaa sitten kaikki taulukossa olevat pallot näytölle.