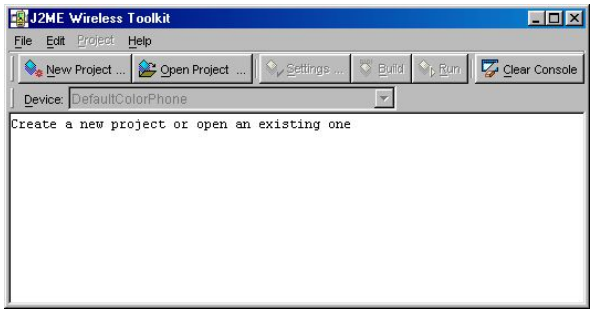
J2ME

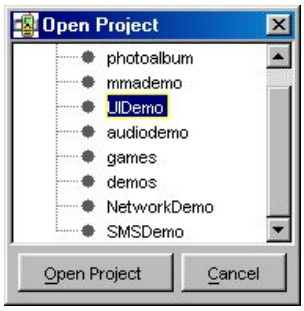
Mobiilirakendused, graafika, salvestamine, võrguühendus

Miniseadmete programmide koostamiseks pakub SUN J2ME Wireless Toolkiti nimelist keskkonda aitamaks koodi kompileerida, kontrollida ja käivitada. Samad toimingud õnnestub teha ka käsurealt, kuid vähemasti alustada tundub mugavam olema abikeskkonnast, KToolbari nimelisest vahendist.

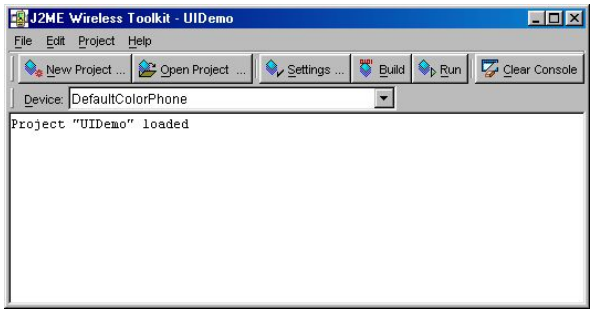
Demonstratsiooniprogrammid

Võimalustega tutvumiseks sobivad kaasatulevad demoprogrammid. Vajutades nupule "Open Project", võib valida kümnekonna väikese valmisrakenduse seast.





Kui projekt valitud, muutuvad töötavaks ka ülejäänud nupud. Nagu aimata võib, Build kompileerib ja teeb muud vajalikud ettevalmistustööd, Run käivitab.

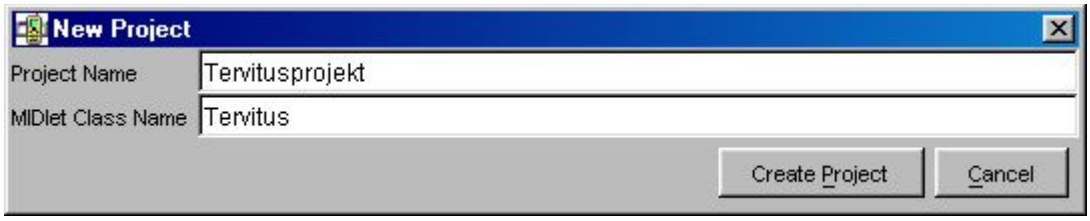


Seadme saab valida paari erikujulise emulaatori hulgast, et õnnestuks kontrollida rakenduse töövõimet mitmesuguse ekraaniresolutsiooni ning nuppude asetuse korral. Konkreetsete telefonimudelite emulaatoreid võib juurde laadida tootjate veebilehtedelt.

Valitud UIDemo tutvustab kasutajaliidese kujundamiseks tarvilikke elemente.

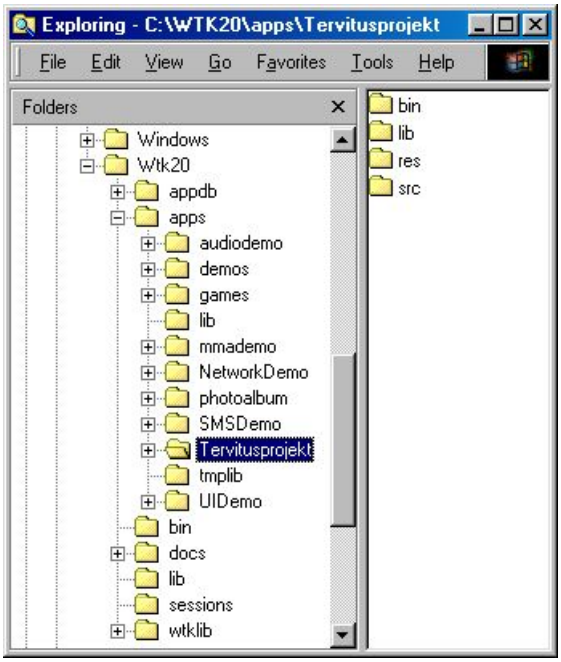
Omaloodud projekt

Ka demokoode uurides, muutes ja tulemust piiludes võib enesele küllalt meelepärase tulemuse saada. Mingil hetkel aga tekib tahtmine või vajadus "päris omi" programme kokku panema hakata. Selleks tasub luua uus projekt.



Projektile nimi ning kõrvale käivitatava MIDlet-i klassi nimi, millest programmi töö peale algab.

Kui failisüsteemi piiluda, siis võib näha, et iga projekti tavis on loodud \Wtk20\apps kataloogi alla omaette kataloog. Nii ka nüüd tehtud tervitusprojekti tarbeks. Ning projekti kataloogi sisse luuakse kohe hulk alamkatalooge, millest esialgu meile tarvilik on src oma programmikoodide paigutamiseks. Nii teatab ka arenduskeskkond:



Place Java source files in "C:\WTK20\apps\Tervitusprojekt\src"

Ülejäänud WTK20 all olevatest kataloogidest on esiotsa tarvilikum docs. Selle seest leiab nii J2ME API täieliku kirjelduse kui mõningaid näiteid ja seletusi.

Tervitav programm

Esimene programm nagu ikka – võimalikult lihtne ja lühike. Lihtne tervitus võtab siin küll mõnevõrra enam koodiridu kui käsurealt käivitatavas programmis, kuid mitte midagi hirmsat. Mobiilseadmes käivitatav programm peab olema kirjutatud javax.microedition.midlet.MIDlet-i alamklassina sarnaselt nagu näiteks veebilehel

töötav rakend vajab enesele ülemklassiks java.applet.Applet-i. Kohustuslikeks ülekaetavateks meetoditeks on startApp, pauseApp ja destroyApp. Nagu nimedki ütlevad, esimene käivitatakse rakenduse töö alustamisel, teine töö ajutisel katkemisel (näiteks sissetuleva kõne korral) ning kolmas töö lõppemisel. Siin näites on nad aga kõik tühjaks jäetud ning lihtsalt klassi konstruktoris luuakse teade ja paigutatakse see rakenduse ekraanile. Display.getDisplay(this) annab juurdepääsu rakenduse ekraanile ning käsklus setCurrent määrab, milllise objektiga ekraan täidetakse.

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

public class Tervitus extends MIDlet{
  public Tervitus(){
    Alert a=new Alert("Tervitamine", "Tere", null,
           AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(a);
  }
  protected void startApp() throws MIDletStateChangeException{}
  protected void pauseApp(){}
  protected void destroyApp( boolean p1 )
                     throws MIDletStateChangeException{}
}
```

Pärast kompileerimist käivitades võib rakenduse tööd imetleda. Esmalt tuleb ette projektis määratud MIDlet-i nimi ning selle valides avaneb rakenduse sisu, milleks siin vaid ekraanile näidatud tervitus.

Nagu näha toimib sama rakendus loetavana ka suurema seadme peal.

Kalkulaator

... ehk lihtsaim rakendus, mille töö tulemustega ka juba midagi peale hakata on. Nagu ikka, tuleb valmis ehitada nii kujundus kui arvutusloogika. Ning nagu lihtsamate programmide puhul ikka, kulub rohkem koodiridu väljanägemise sättimise tarvis. Tekstiväli nii kummagi liidetava sisestamiseks kui vastuse vaatamiseks. Tüübiks NUMERIC, mis – nagu nimestki näha – lubab välja sisse vaid numbreid kirjutada. Soovides J2SE vahenditega taolist tekstivälja luua tuleks mõnevõrra rohkem nikerdada. Vorm üksikute tükkide koos hoidmiseks. Üks käsklusnupp arvutamise ning teine väljumise tarbeks. Ning vormi küljes käsklus setCommandListener(this), mis tähendab, et vormis loodud käsklused antakse töötlemiseks this-ile ehk jooksvale kalkulaatori eksemplarile.Võrrelduna J2SE-ga on J2ME-s ressursside kokkuhoiu mõttes tavaks teated saata töötlemiseks korraga vaid ühte kohta. Nii ongi addActionListeneri asemel siin kirjas setCommandListener.

Sündmuste püüdmiseks realiseerib Kalkulaatori klass liidest CommandListener, mis annab kohustuse luua meetod CommandAction. Esimese parameetri järgi saab otsustada, milline käsklus saabus, teine näitab aktiivset ekraani, millel käsklusele vajutati. Nagu käskluste nimedestki näha, ühe ülesandeks on paluda tulemus arvutada, teise omaks rakenduse töö lõpetada. Et tekstivälja väärtust küsitakse sõnena, liitmise tulemust arvutatakse aga arvuna, siis tuleb ette võtta mõlemasuunaline tüübimuundus, mis teebki käsu pikaks. J2ME arvutuskäsud kasutavad vaid täisarve, kuna vastavate seadmete protsessorid ei pruugi suuta osata muude arvudega otse tehteid teha ning "ümber nurga" arvutamine võtab jälle märgatavalt ressursse.

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

public class Kalkulaator extends MIDlet implements CommandListener{
  TextField tf1=new TextField("Esimene arv", "", 5,
                    TextField.NUMERIC);
```

```
  TextField tf2=new TextField("Teine arv", "", 5, TextField.NUMERIC);
  TextField tf3=new TextField("Vastus", "", 5, TextField.NUMERIC);
  Command c1=new Command("Arvuta", Command.SCREEN, 1);
  Command c2=new Command("Välju", Command.EXIT, 1);
  Form f=new Form("Arvutaja");
  public Kalkulaator(){
    f.append(tf1);
    f.append(tf2);
    f.append(tf3);
    f.addCommand(c1);
    f.addCommand(c2);
    f.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
  }
  protected void startApp( ) throws
                 MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 )
   throws MIDletStateChangeException{}

  public void commandAction(
            Command p1, Displayable p2 ){
     if(p1==c1){
       tf3.setString(String.valueOf(
           Integer.parseInt(tf1.getString())+
           Integer.parseInt(tf2.getString()))
       );
     }
     if(p1==c2){
       notifyDestroyed();
     }
  }
}
```

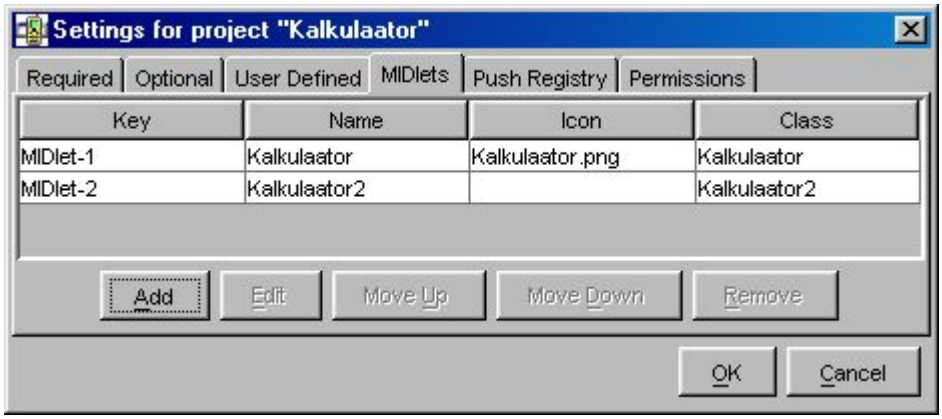
Tehtevalikuga kalkulaator

Eelmise rakenduse täiendus. Liitmise asemel võib kasutada nelja põhitehet. Ühel vormil endiselt tekstiväljad andmete sisestuseks ning koht vastuse tarvis. Sedakorda vastuse näitamiseks StringItem, mille sisu saab küll programmi poolt määrata, kuid mitte kasutaja ise kirjutada. Ka parasjagu aktiivne tehe näidatakse StringItem-i abil. Tehtevalikuks kasutatakse ekraanisuurust komponenti List, mis siis vastava käsu peale välja kutsutakse. Kui tehe valitud, asendatakse ekraaninäit jälle endise vormiga ning valitud element paigutatakse vormi sisse tehte kohale.

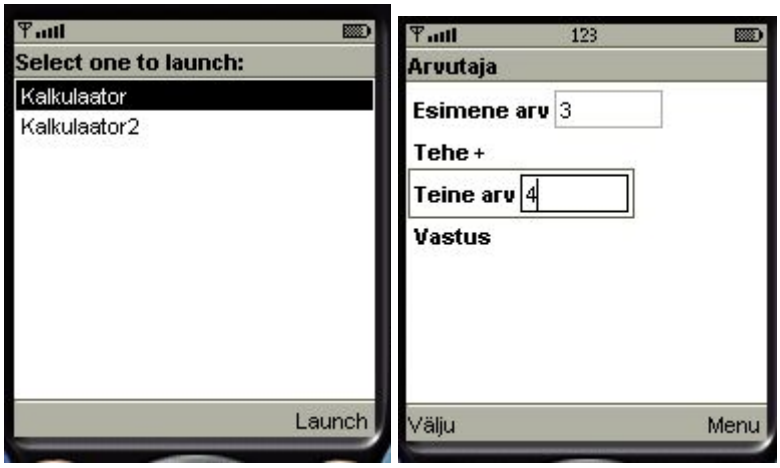
```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Kalkulaator2 extends MIDlet implements CommandListener{
  TextField tf1=new TextField("Esimene arv", "", 5, TextField.NUMERIC);
  TextField tf2=new TextField("Teine arv", "", 5, TextField.NUMERIC);
  StringItem vastus=new StringItem("Vastus", "");
  String[] tehted={"+", "-", "*", "/"};
  List nimistu=new List("Tehte valik", List.EXCLUSIVE, tehted, null);
  StringItem silt=new StringItem("Tehe", "+");
  Command c1=new Command("Arvuta", Command.SCREEN, 1);
  Command c2=new Command("Välju", Command.EXIT, 1);
  Command c3=new Command("Vali tehe", Command.SCREEN, 1);
  Command c4=new Command("Tehe valitud", Command.SCREEN, 1);
  Form f=new Form("Arvutaja");
  public Kalkulaator2(){
    f.append(tf1);
    f.append(silt);
    f.append(tf2);
    f.append(vastus);
```

```
    f.addCommand(c1);
    f.addCommand(c2);
    f.addCommand(c3);
    nimistu.addCommand(c4);
    f.setCommandListener(this);
    nimistu.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
  }
  protected void startApp( ) throws MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 ) throws MIDletStateChangeException{       }

  void arvuta(){
    int arv1=Integer.parseInt(tf1.getString());
    int arv2=Integer.parseInt(tf2.getString());
    String tulemus="";
    if(silt.getText().equals("+")){tulemus=String.valueOf(arv1+arv2);}
    if(silt.getText().equals("-")){tulemus=String.valueOf(arv1-arv2);}
    if(silt.getText().equals("*")){tulemus=String.valueOf(arv1*arv2);}
    if(silt.getText().equals("/")){tulemus=String.valueOf(arv1/arv2);}
    vastus.setText(tulemus);
  }

  public void commandAction( Command p1, Displayable p2 ){
    if (p1==c1){
      arvuta();
    }
    if(p1==c2){
      notifyDestroyed();
    }
    if(p1==c3){
      Display.getDisplay(this).setCurrent(nimistu);
    }
    if(p1==c4){
      silt.setText(nimistu.getString(nimistu.getSelectedIndex()));
      Display.getDisplay(this).setCurrent(f);
    }
  }
}
```

Kui loodud klassi fail lihtsalt eelmise kõrvale kopeeritakse, siis fail küll kompileeritakse, kuid kusagilt ei leia kohta rakenduse uue versiooni käivitamiseks. Valides "Settings" ning sealt alt "MIDlets", saab vaadata, muuta ja lisada, millise nime all millises klassis olev kood käima tõmmatakse. Kõik rakenduses paiknevad klassid ei pea sugugi eraldi käivitatavad olema, mõnigi võib olla lihtsalt abiks rakenduse avaklassile. Ning samas kataloogis asuvaid abiklasse saab ka vajadusel kasutada mitme teise klassi koodi sees. Tahtes aga loodud MIDleti teha rakenduse avamisel valitavaks, tuleb ta vastavasse loetellu lisada.

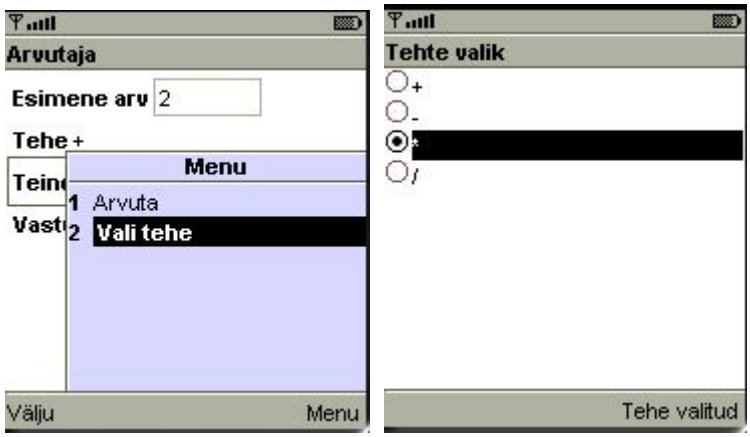


Nii võibki nüüd käivitamisel näha projekti sees kahte rakendust. Valides neist teise, tuleb ette tehtevalikuga kalkulaator.



Kui andmed sisestatud ning menüüst vajutada "arvuta", näeb tulemust.

Paludes aga tehe valida, ilmub eraldi List, mille hulgast siis tuleb sobiv enesele välja valida, kinnitades valikut käsuga "Tehe valitud".



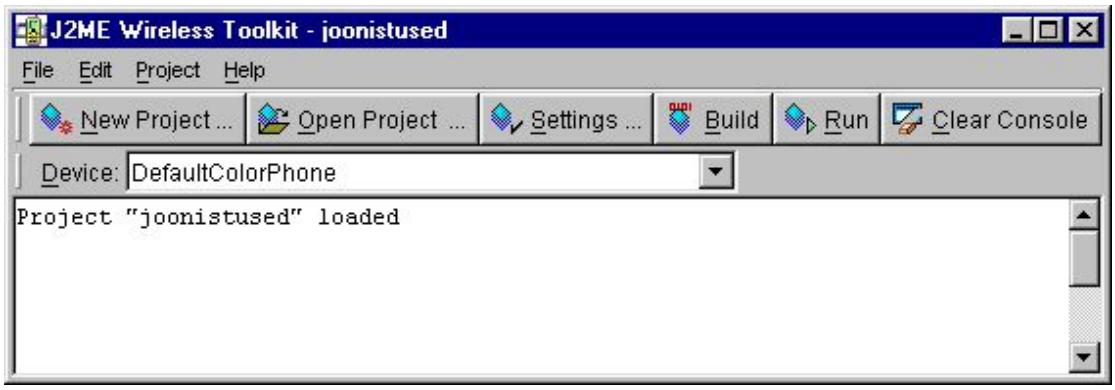
Siis pääseb tagasi avaekraanile ning võib vastava tehtega arvutustulemust vaadata.

Joonistused

Nagu enamike programmeerimisvahendite puhul, nii ka siin saab otse ekraanipunktidega arvutada ning sinna kujundeid joonistada. Nagu J2SE puhul, nii ka siin sobib joonistamise aluspinnaks Canvas ehk lõuend. Tavaline lõuend jätab enese alt välja nii tiitliriba kui käsuriba, kuid enamasti rakenduse ajal sellist lahendust vajataksegi. Kel tahtmist täisekraaniga tegelda, see vajab alates MIDP 2.0-st kättesaadavat GameCanvast. Joonistamiskäske antakse ikka meetodisse paint saabuva Graphicsi eksemplari kaudu. Kuid nagu mitmete muude klassidega, on ka osa siinsetest joonistusvahenditest samanimeliste J2SE klasside omadest mõnevõrra erinevad. Ning kui ka "hariliku" Java puhul on viisakas arvestada ekraani suuruse ja võimalustega, siis mobiilirakenduste puhul peab arvestama seadme parameetrite küllalt suure kõikumisega.

Üksik joon

Joonistuste tarbeks eraldi projekt ning koos sellega ka kataloog. Nii on sama teema rakendused ilusti ühes paigas koos.



Kui joonistuse oskused paigutada eraldi klassi, siis peaprogrammi saab jätta küllalt lühikeseks. Vaid rakenduse avamisel määratakse lõuendi eksemplar rakenduse paneelil nähtavaks.

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

public class Joonistus1 extends MIDlet{
  Canvas louend=new Louend1();
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
               throws MIDletStateChangeException{}
}
```

Lõuendi joonis ise kuvatakse ekraanile paint-meetodis, nii nagu tavarakendustegi puhul kombeks. Et vana pilt ekraanile paistma ei jääks, tuleb kõigepealt taustale ristkülik joonistada. Esimene joonis koosneb vaid ühest joonest, käsu parameetriteks nagu ikka kummagi otspunkti kaks koordinaati.

```
import javax.microedition.lcdui.*;
class Louend1 extends Canvas{
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawLine(10, 10, 80, 10); //x1, y1, x2, y2
  }
}
```

Ehkki rakendus koosneb kahest klassist, piisab, kui eraldi MIDletina on kirjas vaid Joonis1. Louend1 leitakse kataloogist üles.

Mitmekülgsem joonis

Nagu ennegi mainitud, käsud sarnanevad J2SE omadele, kuid ei pruugi kattuda. Puudub näiteks fillOval, tema ülesanded aga täidab fillArc, tuleb lihtsalt sobivad parameetrid ette anda. Punktiirjoon, mille tekitamiseks muidu oli vaja Graphics2D poole pöörduda lubatakse siin harilikus Graphics-klassis ekraanile paigutada. Teksti puhul võib määrata, kuhu jääb etteantud punkt teksti joonistamisel.

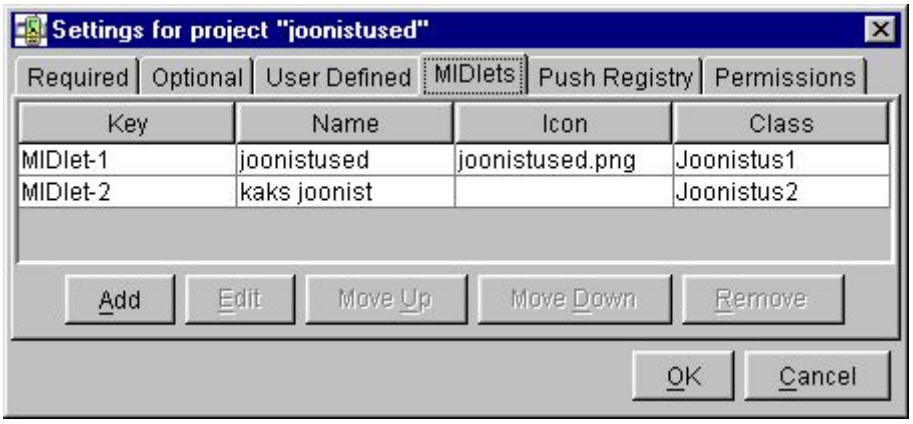
```
import javax.microedition.lcdui.*;

class Louend2 extends Canvas{
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.fillArc(10, 10, getWidth()-20, 50, 0, 180);
      //vasakult, ülalt, laius, kõrgus, algnurk, kaarenurk
     g.setStrokeStyle(Graphics.DOTTED);
     g.drawLine(10, 30, 10, getHeight());
     g.drawLine(getWidth()-10, 30, getWidth()-10, getHeight());
     g.drawString("Tervitus", getWidth()/2, getHeight()/2,
                              Graphics.BOTTOM | Graphics.HCENTER);
  }
}
```

Kaks joonist

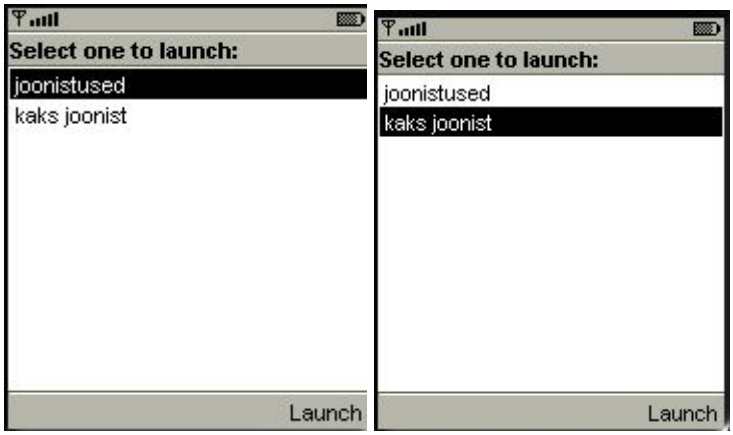
Peaprogrammis võimaldatakse valida, millist lõuendit vaadata. Kusjuures esimese lõuendi küljes on käsklus teise lõuendi vaatamiseks ning teisel jälle esimese juurde jõudmiseks. Käskude sisu nagu ikka kirjas commandAction'is. Ning algselt määratakse nähtavaks lõuend1.

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

public class Joonistus2 extends MIDlet implements CommandListener{
  Canvas louend1=new Louend1();
  Canvas louend2=new Louend2();
  Command c1=new Command("Esimene", Command.ITEM, 1);
  Command c2=new Command("Teine", Command.ITEM, 1);
  public Joonistus2(){
    louend1.addCommand(c2);
    louend2.addCommand(c1);
    louend1.setCommandListener(this);
    louend2.setCommandListener(this);
  }
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend1);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
  public void commandAction(Command c, Displayable d){
    if(c==c1){Display.getDisplay(this).setCurrent(louend1);}
    if(c==c2){Display.getDisplay(this).setCurrent(louend2);}
  }
}
```

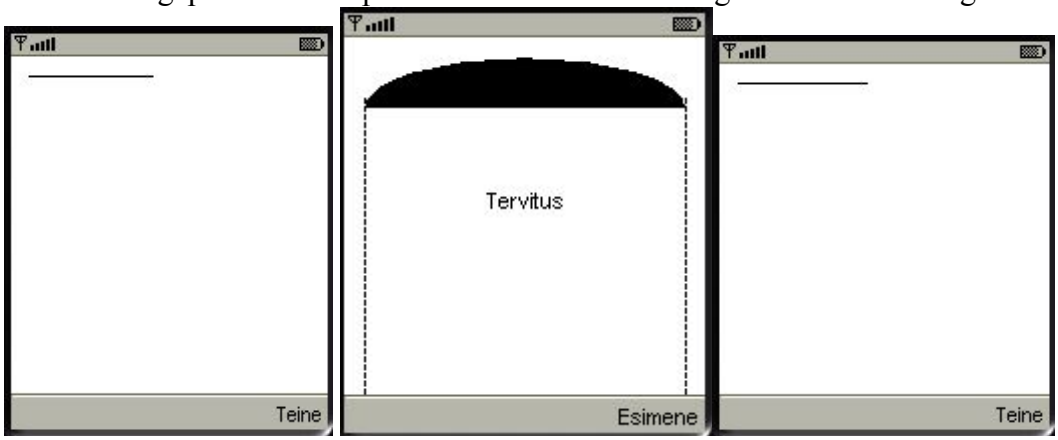
Taas vajab märkimist, et ja Joonistus2 võiks eraldi rakendusena tööle hakata.



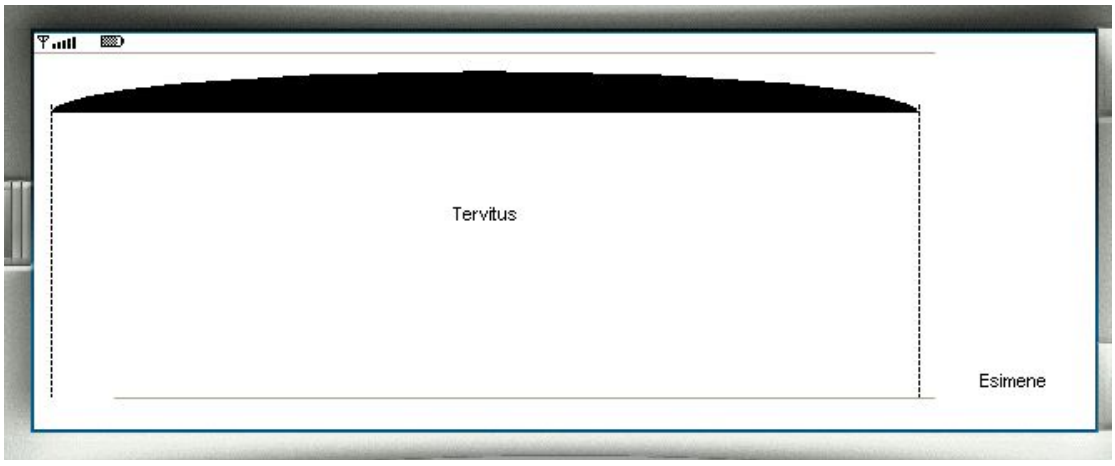
Ning edasi võibki rakenduse valida ja käivitada.



Vaadata kõigepealt esimest pilti. Siis liikuda teisele ning taas esimesele tagasi.



Teine joonis arvestab ekraani mõõtudega ning on suurema ekraani peal ka tunduvalt suuremalt nähtav.



Liigutamine

Olgu tegemist andmebaasi graafilise väljundiga, mänguga või lihtsalt olukorra simulatsiooniga – ikka soovitakse ekraanipilti kasutaja tegevuse tulemusena muuta. Põhimõte sarnane kui Java tavarakenduste juures – andmemuutuse järel tuleb

ekraanipilt uuesti joonistada. Klahvisündmuste kuulamiseks piisab üle katta vaid keyPressed – liikuvad teated peaksid automaatselt sinna alamprogrammi jõudma nii nagu näiteks Java esimese versiooni puhul. Et mitmesuguse ehitusega masinatega hakkama saada, kasutatakse kaht väärtust. Meetodi parameetrisse jõudev arv vastab konkreetsele klahvile. Canvase käsklus getGameAction aga leiab koodile vastava loogilise tähenduse. Nõnda võib eri masinatel hoopis isesugune klahv või nende kombinatsioon tähistada noolt vasakule. Käsu getGameAction abil saab aga kontrollida, et kas konkreetse masina ja klahvi puhul sooviti anda vasakule liikumise käsklust. Meetod repaint nagu ikka palub pildi uuesti joonistada.

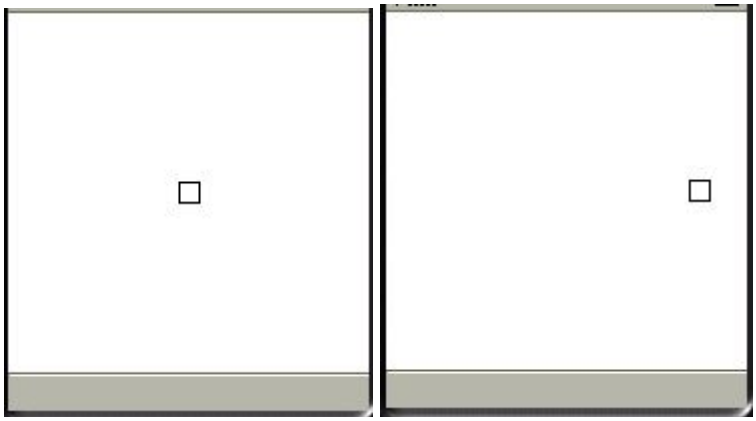
Et soovin ruudu algselt paigutada ekraani keskele, siis käsu showNotify käivitumise ajal leian vastavad koordinaadid. Varem poleks see võimalik, sest ekraani suurus ei pruugi programmile teada olla.

```
import javax.microedition.lcdui.*;

class Louend3 extends Canvas{
  int x, y;
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawRect(x-5, y-5, 10, 10);
  }
  protected void keyPressed(int kood){
    if(getGameAction(kood)==Canvas.LEFT){x--;}
    if(getGameAction(kood)==Canvas.RIGHT){x++;}
    repaint();
  }
  public void showNotify(){
    x=getWidth()/2;
    y=getHeight()/2;
  }
}
```

Käivitamislõik sama lühike nagu mujalgi.

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

public class Joonistus3 extends MIDlet{
  Canvas louend=new Louend3();
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
}
```

Valides liigutamist näitava rakenduse võib edaspidi vastavate klahvide abil liigutada ekraanil paiknevat ruutu.



Liikumine

Soovides, et ekraanil miski iseeneslikult liiguks, tuleb pilt iga natukese aja tagant uuesti arvutada ja joonistada. Üheks võimaluseks on kasutada klasside Timer ja TimerTask abi. Viimases kirjeldatakse, mida iga sammu jooksul tegema peab. Esimene aga palub tegevust soovitud aja tagant korrata. Liikumisega seotud teevused koondati meetodisse liigu, TimerTaski eksemplari ülesandeks on vaid vastav käsklus välja kutsuda. Selleks kaetakse üle run-meetod ning pannakse liikumiskäsklus sinna sisse. Käsklus schedule käsib kokkupandud tegevust iga saja millisekundi tagant välja kutsuda, alustades esimest korda 500 millisekundit pärast schedule käsu väljakutset. Ekraani peitmisel välja kutsutav hideNotify katkestab Timeri kordamistöö.

Liikumisel hoolitsetakse, et ruut üle ekraani serva ei liiguks. Kui käiguga satutaks üle serva, siis enne seda pannakse samm nulliks ning uuesti liikuda kannatab vaid klahvivajutuse puhul ja suunaga seinast eemale.

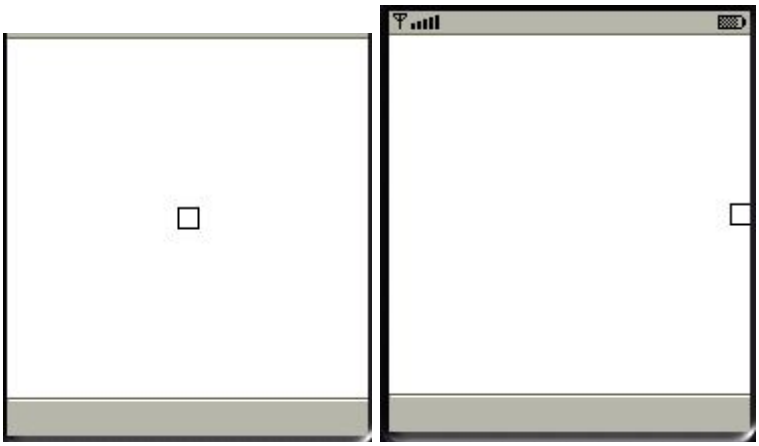
```
import javax.microedition.lcdui.*;
import java.util.*;

class Louend4 extends Canvas{
  int x, y, samm=0, r=5;
  Timer t;
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawRect(x-r, y-r, 2*r, 2*r);
  }
  protected void keyPressed(int kood){
    if(getGameAction(kood)==Canvas.LEFT){samm=-1;}
    if(getGameAction(kood)==Canvas.RIGHT){samm=1;}
  }
  public void showNotify(){
    x=getWidth()/2;
    y=getHeight()/2;
    t=new Timer();
    TimerTask tt=new TimerTask(){
      public void run(){
        liigu();
      }
    };
    t.schedule(tt, 500, 100); //tegevus, viivitus, intervall
  }

  public void hideNotify(){
    t.cancel();
  }

  void liigu(){
     if((samm>0 && x+samm+r>getWidth()) ||
        (samm<0 && x+samm-r<0)){
       samm=0;
     } else {
```

```
        x+=samm;
        repaint();
      }
  }
}
```

Käivituskood jälle võimalikult lühike.

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

public class Joonistus4 extends MIDlet{
  Canvas louend=new Louend4();
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
}
```

Nagu pildilt näha, jääb liikuv ruut servani jõudnult pidama.



Andmed veebist

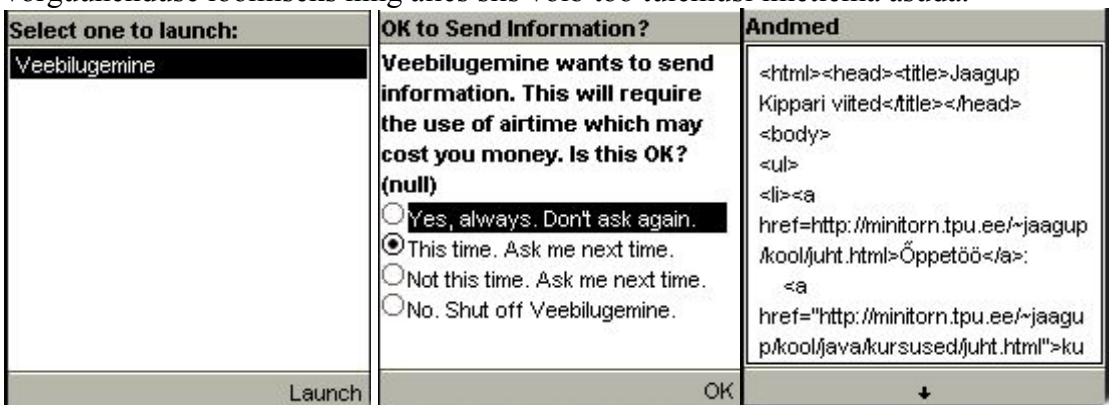
Et mobiiltelefon küllalt hulgaliselt andmeid välisilmaga vahetab, siis oleks patt jätta sealsed Java-programmid suhtlusvõimalusest välja. Samas aga igasugune ühendamine võib kulutada nii telefoniomaniku raha kui pakub võõrale programmile võimalusi veebiserverite värateid lõgistada ning muidki salapäraseid toimetusi ette võtta. Et Java puhul on püütud turvataset küllalt kõrgel hoida, siis ligipääs seadme võimalustele on küllaltki piiratud. Tavavahenditega ei pääse ligi ei telefoniraamatule, äratuskellale ega mujalegi. Mis ära keelatud, sealt pole ka ohtu karta. Samuti pole ju kõik J2ME seadmed sugugi mobiiltelefonid ning nende soovid ja võimalused sootuks teistsugused. Võrguliiklust aga mõnevõrra siiski lubatakse. Esimestest J2ME versioonidest alates töötab HTTP, vaikselt lisandub muidki ühenduskanaleid. Ärirakenduste loojate südameid soojendab HTTPS. Sest päris paljud ei raatsi oma paroole ja teadmisi vabalt üle võrgu liikuma saata. Samuti võib sidet pidada pistikliidese kaudu – seal juba antakse programmeerijale päris piisavalt ise otsustamisvõimalusi. Õnnelikul kombel saavad igasugused ühendused alguse klassist Connector. Nõnda tuleb siis lihtsalt käskude ja parameetrite abil proovida ja otsustada, et milline ühendusliik sobivaks osutub.

Kui liigutatavad andmed keerukamad, siis tasub alumise kihi peale ehitada/paigutada mõni paindlikum ülekandevahend. Näiteks XMLi-põhine, kui andmete suurem maht liialt raskusi ei valmista. Siin aga avatakse lihtsalt veebiaadres

ning loetakse sealsed andmed tekstialasse. Nähtud näite järgi kannatab aga tunduvalt keerukamaid andmete küsimise ja salvestamise rakendusi kokku panna.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.*;
public class Veebilugeja1 extends MIDlet {
  TextBox t1;
  public Veebilugeja1(){
   try{
    String aadress="http://www.tpu.ee/~jaagup/";
    InputStream sisse=Connector.openInputStream(aadress);
    ByteArrayOutputStream baos=new ByteArrayOutputStream();
    int arv=sisse.read();
    while(arv!=-1){
      baos.write(arv);
      arv=sisse.read();
    }
    byte[] b=baos.toByteArray();
    String tekst=new String(b);
    t1=new TextBox("Andmed", tekst, tekst.length(), TextField.ANY);
    Display.getDisplay(this).setCurrent(t1);
   }catch(Exception e){
     e.printStackTrace();
   }
  }
  protected void startApp( ) throws MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 ) throws MIDletStateChangeException{}
}
```

Nagu kopeeritud piltidelt paistab, küsitakse kõigepealt kasutajalt luba võrguühenduse loomiseks ning alles siis võib töö tulemusi imetlema asuda.



Salvestus.

Miniseadmetes talletatakse andmeid kirjetena. Java käskudega õnnestub andmeid talletada vaid baidimassiivi. Ülejäänud muundused tuleb kirjutamisel ja lugemisel ise läbi teha. Andmete hoidmisega seotud vahendid paiknevad paketis javax.microedition.rms. Järgneva näitrakendusena koostatakse loendur teatamaks, mitmendat korda rakendusse sisenetakse.

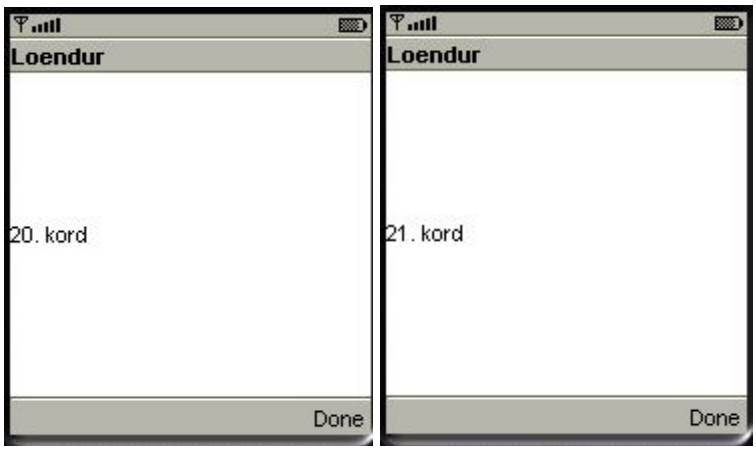
Loendur

Hoidla avatakse käsuga RecordStore.openRecordStore parameetriteks hoidla nimi sõnena ning tõeväärtusmuutuja teatamaks, kas vastava nimega hoidla puudumisel hoidla luuakse. Kui hoidlas pole ühtegi kirjet (`rs.getNumRecords()` väljastab 0), siis lisatakse hoidlasse ühebaidine kirje, mille arvuliseks väärtuseks siis sisenemiskordade arv järgmisel korral. Kui aga kirje baasis juba olemas, siis küsitakse

esimese kirje bait number 0 (ehk selle kirje ainuke bait). Jäetakse selle väärtus muutujasse meelde ning kirjesse kirjutatakse eelmisest ühe võrra suurem väärtus. Käsklus `rs.setRecord(1, new byte[]{(byte)(arv+1)}, 0, 1)` tähendab lahtiseletatult, et kirje nr. 1 väärtuseks määratakse baidimassiivi baidid alates numbrist 0 üks bait. Ning massiiv luuaks kohapeal, pannes selle ainukese elemendi väärtuseks endisest arvust ühe võrra suurema väärtuse. Teade antakse välja Alert-akna abil.

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

public class Salvestus1 extends MIDlet{
  String hoidlanimi="hoidla1";
  protected void startApp() throws MIDletStateChangeException{
   try{
    RecordStore rs=RecordStore.openRecordStore(hoidlanimi, true);
    byte arv=1;
    if(rs.getNumRecords()==0){
      rs.addRecord(new byte[]{(byte)(arv+1)}, 0, 1);
    } else {
      arv=rs.getRecord(1)[0];
      rs.setRecord(1, new byte[]{(byte)(arv+1)}, 0, 1);
    }
    Alert a=new Alert("Loendur", arv+". kord", null, AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(a);
   }catch(Exception e){e.printStackTrace();}
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
              throws MIDletStateChangeException{}
}
```

Nagu allolevalt pildilt näha võib, iga avamisega loenduri väärtus kasvab.



Neljabaidine salvestus

Vähegi rohkemate andmete salvestamisel tuleb liht- ja struktuuritüüpides paiknevaid väärtusi baidimassiiviks ja tagasi muundama hakata. Üheks võimaluseks on ise bitte nihutada, |-märkidega ühendada ning pärast &-märkidega eraldama asuda. Teiseks võimaluseks on ka J2ME-sse kaasa võetud java.io pakettidega arvestada. ByteArrayOutputStream võimaldab ette määramata hulga baite mällu saata ning sealt pärast massiivina välja lugeda. DataOutputStream'i ülesandeks on etteantud lihttüübid või sõned baitidena teele saata.. Teistpidise lugemise juures aitavad DataInputStream ning ByteArrayInputStream. Vastavalt siis allpool näha funktsioonid nimedega

intBaitideks ning baididIntiks. Kui andmed sobivale kujule teisendatud on, siis edasine salvestamine ja küsimine käib sarnaselt kui eelmisegi näite puhul. rs.addRecord(bm, 0, bm.length) salvestab lihtsalt massiivi kogu pikkuses kirjena ning rs.getRecord(1) annab vastavalt kirje numbrile massiivi taas välja. Rakendus töötab sarnaselt kui eelminegi, oskab aga ühebaidilise väärtuse piiridest hulga kaugemale lugeda.

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

public class Salvestus1a extends MIDlet{
  String hoidlanimi="hoidla1a";
  protected void startApp() throws MIDletStateChangeException{
   try{
    RecordStore rs=RecordStore.openRecordStore(hoidlanimi, true);
    int arv=1;
    if(rs.getNumRecords()==0){
      byte[] bm=intBaitideks(arv+1);
      rs.addRecord(bm, 0, bm.length);
    } else {
      arv=baididIntiks(rs.getRecord(1));
      byte[] bm=intBaitideks(arv+1);
      rs.setRecord(1, bm, 0, bm.length);
    }
    rs.closeRecordStore();
    Alert a=new Alert("Loendur",
          arv+". kord", null, AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(a);
   }catch(Exception e){e.printStackTrace();}
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
                throws MIDletStateChangeException{}
  byte[] intBaitideks(int a) throws IOException{
     ByteArrayOutputStream bos=new ByteArrayOutputStream();
     DataOutputStream dos=new DataOutputStream(bos);
     dos.writeInt(a);
     dos.close();
     return bos.toByteArray();
  }
  int baididIntiks(byte[] b) throws IOException {
    ByteArrayInputStream bis=new ByteArrayInputStream(b);
    DataInputStream dis=new DataInputStream(bis);
    return dis.readInt();
  }
}
```

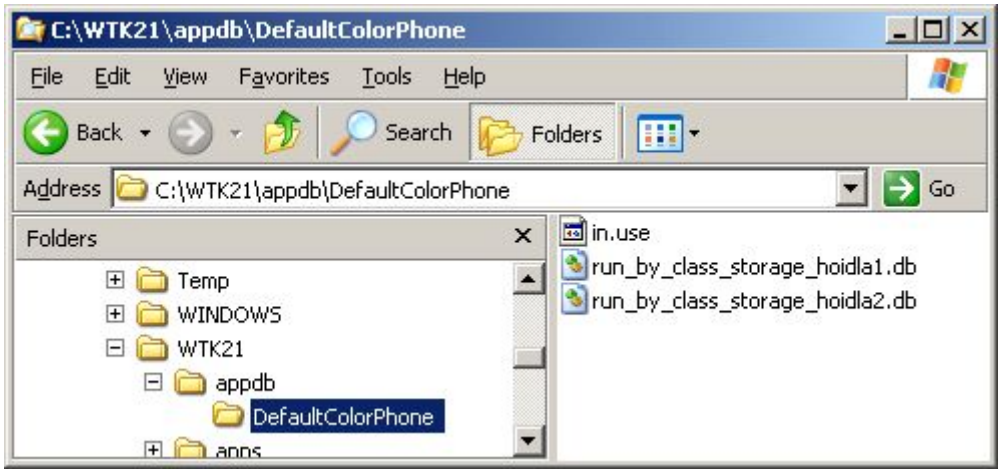
Salvestus vormist

Järgnev näide salvestab ühise kirjena inimese nime ja ta sünnipäeva. Lugemisnäide toodud hiljem. Ühendus andmehoidlaga avatakse kohe rakenduse algul ning suletakse rakenduse sulgumisel destroyApp-funktsiooni sees. (Vea)teadete edastamiseks loodi eraldi väike alamprogramm.

```
  void teade(String teade){
    Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
    a.setTimeout(5);
    Display.getDisplay(this).setCurrent(a);
  }
```

Nõnda õnnestub teade kergema vaevaga kasutajale ekraanile manada.

Kõrvalepilguks: arenduskeskkonnas salvestatakse andmed eraldi kataloogi. Keskkonna juurkataloogist alates appdb ning vastava emulaatori kataloog. Iga andmebaasi tarvis luuakse omaette fail ning fail in.use näitab, kui baas parajasti rakenduse poolt hõivatud on.



Järgnevalt salvestusvõimelise rakenduse kood.

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

public class Salvestus2 extends MIDlet implements CommandListener{
  String hoidlanimi="hoidla2";
  Form f=new Form("Sünnipäevad");
  Command lisa=new Command("Lisa", Command.SCREEN, 1);
  TextField tfnimi=new TextField("Eesnimi", "", 20, TextField.ANY);
  DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
  RecordStore rs;
  public Salvestus2(){
    f.append(tfnimi);
    f.append(dfsynniaeg);
    f.addCommand(lisa);
    try{
     rs=RecordStore.openRecordStore(hoidlanimi, true);
    }catch(Exception e){teade(e.getMessage());}
  }
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(f);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
                throws MIDletStateChangeException{
    try{
      rs.closeRecordStore();
    } catch (Exception e){teade(e.getMessage());}
  }
  public void commandAction(Command c, Displayable d){
    if(c==lisa){
      lisaKirje();
    }
  }

  void lisaKirje(){
    byte[] bm=leiaBaidiMassiiv();
    try{
     rs.addRecord(bm, 0, bm.length);
    }catch(Exception e){
      teade(e.getMessage());
    }
  }
  byte[] leiaBaidiMassiiv(){
   ByteArrayOutputStream bos=new ByteArrayOutputStream();
   DataOutputStream dos=new DataOutputStream(bos);
   try{
    dos.writeUTF(tfnimi.getString());
    dos.writeLong(dfsynniaeg.getDate().getTime());
   }catch(Exception e){teade(e.getMessage());}
   return bos.toByteArray();
  }

  void teade(String teade){
    Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
```

```
    a.setTimeout(5);
    Display.getDisplay(this).setCurrent(a);
  }
}
```

Mitu ekraanivormi

Eelmise täiendatud variant, kus peale andmete salvestamise ka neid näidata saab. Tegutsemine on jäetud mitme ekraanivormi peale. Vormide vahel liikumine on püütud korraldada nõnda nagu suuremate rakenduste puhul tavaks. Et liikudes mööda puud sügavamale õnnestub jälle tuldud teed pidi tagasi tulla. Põhiprogrammi sisse on paigutatud lisaks kolm klassi: valikuvormi, lisamise ning vaatamise tarbeks. MIDleti klassi kasutatakse vaid käivitamise tarbeks ning üldiste väärtuste ja vahendite kogumina.

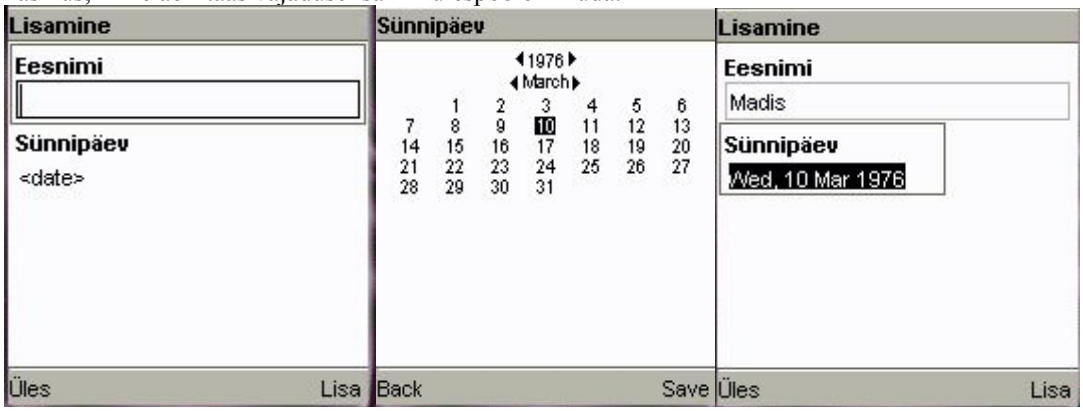
Valikuklass on loodud Listi alamklassina. List nagu Form või TextBox on üle kogu ekraani paiknev element, nõnda ka selle alamklass. Vaatamisvalikul paistab ees olema väike pilt.

Listi elementide lisamiseks kasutatakse käsku append. Pildi puudumisel on teiseks parameetriks null, pidi leidumisel aga Image-tüüpi objekt.

```
super("Tegevuse valik", List.IMPLICIT);
append("Lisa", null);
Image vaatamispilt=null;
try{
  vaatamispilt=Image.createImage("/silmad.png");
} catch(IOException e){
  teade("Probleem pildi laadimisel");
}
append("Vaata", vaatamispilt);
```

Pildid ja muud ressursifailid tuleb paigutada rakenduse alamkataloogi res. Sealt leitakse need üles.

On kord lisamisvorm valitud, võib nime lihtsalt sisse tippida. Sünnipäeva sisestamiseks sobib aga DateField, mille abil siis õige kuupäev välja vaadata. Ning iga ekraanitäie vasakul pool asub käsklus, mille abil taas vajadusel samm ülespoole liikuda.



Nii lisamisvormi kui vaatamisvormile antakse kaasa nende ülemakna muutuja – praegusel juhul siis valikuvormi oma. Nii õnnestub kohe sobivat teed pidi taas puud pidi samm tagasi astuda.

```
LisamisVorm lisamine=new LisamisVorm(this);
VaatamisVorm vaatamine=new VaatamisVorm(this);
```

Peremehe ehk väljakutsuva akna meeles pidamiseks hoitakse meeles Displayable-tüüpi muutuja.

```
Displayable avaja;
LisamisVorm(Displayable avaja){
   super("Lisamine");
   this.avaja=avaja;
   ...
}
```

Ülesliikumise nupule vajutades saab siis lihtsalt avaja-ekraani aktiivseks muuta. Põhiklassi saab kätte konstruktsiooniga Salvestus3.this. Mõnes paigas soovitatakse selle asemel kasutada ka vastava klassi külge pandud staatilist meetodit, mis siis väljastaks vastava klassi ainukese eksemplari. Et aga siin ehitus sisemiste klasside abil kokku pandud, õnnestub ka praegusel kujul ligi pääseda.

```
public void commandAction(Command c, Displayable d){
  if(c==yles){
    Display.getDisplay(Salvestus3.this).setCurrent(avaja);
  }
  ....
}
```

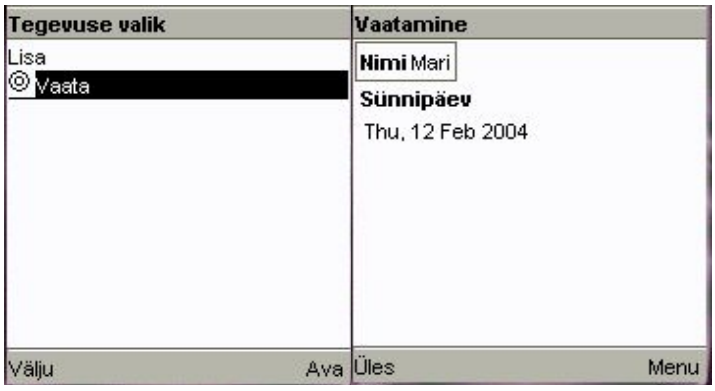
Vaatamise puhul siis õnnestub lihtsalt mööda kirjeid edasi ja tagasi liikuda ning väärtusi vaadata. Liikumist hõlbustab

```
RecordEnumeration re;
```

Kui see kord baasiühenduse käest küsitud ning viimase true abil uuendatavaks muudetud, siis õnnestub lihtsalt käskude abil mööda kirjeid edasi/tagasi liikuda.

```
re=rs.enumerateRecords(null, null, true);
```

Eelnevate null-väärtuseliste parameetrite abil võinuks omaloodud filtri abil osa kirjeid välja sorteerida või siis enesele soovitud järjekorda sättida. Sellisel juhul tuleks aga kirje baidijada järgi otsustada, milline kirje enesele sobib või mille järgi kirjeid omavahel reastada. Nagu pildi pealt paistab, võib vaatamise alt inimese andmeid näha.



Ja lõpuks siis selle salvestusrakenduse kood.

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import java.io.*;
import java.util.*;
public class Salvestus3 extends MIDlet{
  String hoidlanimi="hoidla2";
  RecordStore rs;
  RecordEnumeration re;
  ValikuVorm valik=new ValikuVorm();
  public Salvestus3(){
    try{
     rs=RecordStore.openRecordStore(hoidlanimi, true);
     re=rs.enumerateRecords(null, null, true);
    }catch(Exception e){teade(e.getMessage());}
  }
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(valik);
```

```
}
protected void pauseApp(){}
protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{
  try{
    rs.closeRecordStore();
  } catch (Exception e){teade(e.getMessage());}
}


void teade(String teade){
  Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
  a.setTimeout(5);
  Display.getDisplay(this).setCurrent(a);
}
class ValikuVorm extends List implements CommandListener{
   LisamisVorm lisamine=new LisamisVorm(this);
   VaatamisVorm vaatamine=new VaatamisVorm(this);
   Command ava=new Command("Ava", Command.SCREEN, 1);
   Command valju=new Command("Välju", Command.EXIT, 1);
   ValikuVorm(){
     super("Tegevuse valik", List.IMPLICIT);
     append("Lisa", null);
     Image vaatamispilt=null;
     try{
       vaatamispilt=Image.createImage("/silmad.png");
     } catch(IOException e){
       teade("Probleem pildi laadimisel");
     }
     append("Vaata", vaatamispilt);
     setSelectCommand(ava);
     addCommand(valju);
     addCommand(ava);
     setCommandListener(this);
   }
   public void commandAction(Command c, Displayable d){
     if(c==valju){
        notifyDestroyed();
     }
     if(c==ava){
       if(getSelectedIndex()==0){
         Display.getDisplay(Salvestus3.this).setCurrent(lisamine);
       }
       if(getSelectedIndex()==1){
         Display.getDisplay(Salvestus3.this).setCurrent(vaatamine);
       }
     }
   }
}
class LisamisVorm extends Form implements CommandListener{
  TextField tfnimi=new TextField("Eesnimi", "", 20, TextField.ANY);
  DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
  Command lisa=new Command("Lisa", Command.SCREEN, 1);
  Command yles=new Command("Üles", Command.EXIT, 1);
  Displayable avaja;
  LisamisVorm(Displayable avaja){
     super("Lisamine");
     this.avaja=avaja;
     append(tfnimi);
     append(dfsynniaeg);
     addCommand(lisa);
     addCommand(yles);
     setCommandListener(this);
  }
  public void commandAction(Command c, Displayable d){
    if(c==yles){
      Display.getDisplay(Salvestus3.this).setCurrent(avaja);
    }
    if(c==lisa){
      lisaKirje();
    }
  }

  void lisaKirje(){
    byte[] bm=leiaBaidiMassiiv();
    try{
     rs.addRecord(bm, 0, bm.length);
     dfsynniaeg.setDate(null);
     tfnimi.setString("");
    }catch(Exception e){
      teade(e.getMessage());
    }
  }
```

```
    byte[] leiaBaidiMassiiv(){
     ByteArrayOutputStream bos=new ByteArrayOutputStream();
     DataOutputStream dos=new DataOutputStream(bos);
     try{
      dos.writeUTF(tfnimi.getString());
      dos.writeLong(dfsynniaeg.getDate().getTime());
     }catch(Exception e){teade(e.getMessage());}
     return bos.toByteArray();
    }
  }

  class VaatamisVorm extends Form implements CommandListener{
    StringItem snimi=new StringItem("Nimi", "");
    DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
    Command edasi=new Command("Edasi", Command.SCREEN, 1);
    Command tagasi=new Command("Tagasi", Command.SCREEN, 1);
    Command yles=new Command("Üles", Command.BACK, 1);
    Displayable avaja;
    VaatamisVorm(Displayable avaja){
       super("Vaatamine");
       this.avaja=avaja;
       append(snimi);
       append(dfsynniaeg);
       addCommand(edasi);
       addCommand(tagasi);
       addCommand(yles);
       setCommandListener(this);
    }
    public void commandAction(Command c, Displayable d){
     try{
      if(c==yles){
        Display.getDisplay(Salvestus3.this).setCurrent(avaja);
      }
      if(c==edasi){
       if(re.hasNextElement()){
        loeBaidiMassiivist(re.nextRecord());
       }
      }
      if(c==tagasi){
        if(re.hasPreviousElement()){
          loeBaidiMassiivist(re.previousRecord());
        }
      }
     }catch(Exception e){
       teade(e.getMessage());
     }
    }
    void loeBaidiMassiivist(byte[] b){
     try{
      DataInputStream dis=
          new DataInputStream(new ByteArrayInputStream(b));
      snimi.setText(dis.readUTF());
      dfsynniaeg.setDate(new Date(dis.readLong()));
     }catch(IOException e){
       teade(e.getMessage());
     }
    }
  }
}
```

Ülesandeid

Mobiiliprogrammidega tutvumine.

* Installeeri arendusvahend.
* Tutvu kaasatulnud demonäidetega.
* Loo uus projekt.
* Lisa src-kataloogi tervitusnäide.
* Kompileeri, käivita.
* Muuda tervituse teksti. Testi.

* Lisa samasse projekti kalkulaatori näide. Testi.
* Lisa tehtevalikuga kalkulaatori näide.
* Lisa kalkulaatorile astendustehe.

Hinnaotsing

* Massiivis on meeles kõneminuti hinnad (sentides) võrkude kaupa, teises massiivis võrkude nimed. Valitakse võrk ning sisestatakse minutite arv ning väljastatakse kõne maksumus.
* Iga võrgu kohta on kirjas, millisest kellaajast alates kehtib milline tariif. Valitakse võrk, sisestatakse alguskellaaeg ja minutite arv ning väljastatakse kõne maksumus.
* Lisaks eelmisele arvestatakse maksumus õigesti ka juhul, kui kõne jooksul minutihind muutub.

Joonistus

Tutvu joonistusnäidetega.
* Koosta pilt: päike, kuusepuu, istepink.
* Arvesta joonistamisel ekraani suurusega.
* Kasuta juhuarve nii, et igal avamisel tuleks pilt mõnevõrra erinev.

Aardepüüdmismäng

* Tutvu liigutamisnäitega
* Võimalda kujundil noolte abil liikuda kõigi nelja ilmakaare suunas.
* Ekraanil juhuslikku kohta tekib ring. Selleni jõudmisel hüppab ring uude
  kohta.
* Platsi keskel on sein, millest ei saa liikumisel läbi minna.
* Iga tabamusega tuleb üks sein juurde.

Munapüüdja

Kaheksakümnendatel levinud mängu mobiilivariant.
* Joonista ekraani kummagisse serva kaks kaldteed.
* Mööda ühte teed veereb alla muna.
* Kui üks muna on alla jõudnud, hakkab kukkuma teine.
* Mune võib ühel teel liikuda korraga mitu.
* Mune võib igal teel liikuda korraga mitu.
* Joonistamisel ja arvutamisel arvestatakse ekraani mõõtmetega.

* Klahvivajutusega saab määrata, millise kaldtee all on püüdmislaud.
* Loetakse kinni püütud ning maha kukkunud mune.
* Lihtsa laua asemel on iga asendi puhul püüdja pilt.

Salvestusrakendus

* Tutvu salvestusnäidetega.
* Muuda andmevoogude abil baidimassiiviks punkti tekstiväljadest loetud punkti koordinaate tähistavad kaks täisarvu.
* Salvesta andmed kirjena hoidlasse.
* Loo eraldi nupp andmete lugemiseks hoidlast ja välja näitamiseks.

Kaart

* Nooltega saab liigutada ekraanil paiknevat ristkülikut.

* Jooksva asukoha koordinaadid talletatakse vastava käsu peale kirjena ning joonistatakse pildil ringina. Testi ringide säilimist rakenduse taasavamisel.
* Olemasoleva ringi kohal olles saab vastava käskluse abil ringi kustutada.
* Iga punkti juurde saab lisada tekstilise seletuse.

Veebirakenduse mobiililiides

* Tutvu näitega J2ME veebiühenduse kohta.
* Otsi üles/koosta veebi näidatav andmetabel laulude ja esitajatega.
* Loo veebileht, kus oleks näha vaid laulude nimed, üks nimi ühel real.
* Loe lehe sisu tekstialasse mobiiliekraanil.

* Loo veebi väljastav leht, kus oleks real näha laulu id, tühik ning laulu nimi.
* Loo mobiili neist valik, kui kasutaja saab soovitud laulu märgistada.
* Pärast märgistamist ja valimist näidatakse selle laulu andmeid eraldi mobiiliekraanil.

* Loo vahend laulude ja esitajate lisamiseks mobiili kaudu.
* Loo vahend andmete kustutamiseks.

* Hoolitse, et lisada ja kustutada saaks vaid registreeritud kasutajanime ja parooliga.