Java ME

Grunderna i ME

- En konfiguration tillhandahåller den viktigaste delmängden av paket/klasser och egenskaper för den virtuella maskinen för ett stort antal apparater,
- En profil är ett API för en mindre mängd apparater
- Dessutom finns valfria paket för specifika teknologier (bluetooth etc).

Konfigurationer

- Java ME plattformen har delats in i två baskonfigurationer, en för mobiltelefoner och liknande, och en inriktad mot mer avancerade mobila apparater som smart-phones och set top boxes.
- Konfiguration för begränsade apparater kallas Connected Limited Device Configration (CLDC)
- Den mer kapabla konfigurationen kallas Connected Device Configuration (CDC).

CLDC

- Konfigurationen för apparater med begränsad kapacitet som tex mobiltelefoner kallas *Connected Limited Device Configuration* (CLDC).
- Den är specifikt designad för att möta behovet av att köra Java på apparater med begränsat minne, processorkraft och begränsad skärmstorlek.
  - Minimum 160 KB ROM och 32 KB RAM Någon form av nätverksuppkoppling som fömodligen är rätt begränsad och långsam

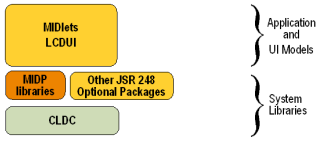
CLDC versioner

- 1.0
  - Inget stöd för flyttalsdatatyper (float, double). Man måste använda fixpunktsdatatyper (i form av klasser istället)
- 1.1
  - Introducerar bla flyttalsdatatyper
  - Bakåtkompatibel med 1.0

CLDC och profiler

- Ovanpå de olika konfigurationerna specificerar JavaME plattformen också ett antal profiler med ytterligare paket/klasser som man också kan använda sig av. Vanligt är att man kombinerar CLDC med Mobile Information Device Profile (MIDP) för att tillhandahålla en komplett Java programmiljö for mobiltelefoner och liknande apparater.
- Utifrån konfigurationen och profilen definierar vi senare våra program.

Hur hänger det ihop



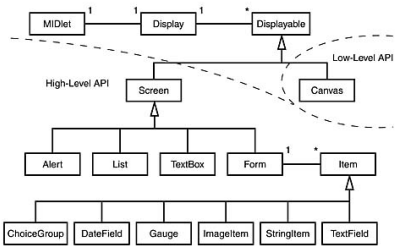
MIDP

- MIDP (Mobile Information Device Profile) är JavaME:s stöd för att utveckla mobiltelefon-användargränssnitt.
- MIDP tillhandahåller ett Java API (känt som LCDUI), som har funktionalitet liknande den hos Java Abstract Windows Toolkit (AWT) och Swing för vanliga datorer.

MIDP

- High level API
  - Tillhandahåller ett antal användarggränssnittsklasser för grundläggande UI komponenter på en relativt hög abstraktionsnivå
- Low level API
  - Ger utvecklare kontroll över hela skärmen
  - Möjlighet att rita på varje enskild pixel om man skulle vilja det
  - Rita grundläggande geometriska figurer
  - Samt möjlighet att rita text med olika fonter mm.

Viktiga klasser



MIDlet

- En MIDlet kräver en apparat som implementerar Java ME, MIDP för att köras.
- Liksom andra java program är det möjligt att kompilera MIDlets en gång och sedan köra dem på olika plattformar.
- MIDlet distributioner består också av en .jad-fil som beskriver innehållet i JAR-filen (innehåller delvis samma information som manifest filen i jar-filen).

- En MIDlet måste uppfylla följande krav för att kunna köras på en mobiltelefon:
  - Huvudklassen måste vara en subklass av javax.microedition.midlet.MIDlet
  - The MIDlet behöver vara hoppackad i en jar-fil
  - Jar-filen behöver preverifieras mha en preverifier. Verifieringen av klasser sköts i vanliga java av virtuella maskinen, men detta kräver rätt mkt resurser.
  - Ibland kan jar-filen behöva signeras

Display

- MIDlets kan antingen vara rena bakgrundsapplikationer eller applikationer som interagerar med användaren.
- En MIDlet kan komma åt dess display genom en instans av Display klassen. Denna instans får man tag på genom att anropa Display.getDisplay (MIDlet) (statisk metod). Som parameter skickar man en referens till MIDlet:en och tillbaka får man ett Display-objekt
- Display-klassen och de andra användargränssnittsklasserna i MIDP finns i paketet javax.microedition.lcdui.

Display forts.

- `Display`-klassen har en metod `setCurrent()` som specificerar vilket innehåll som ska visas på skärmen. Det är inte nödvändigtvis så att skärmen uppdateras omedelbart efter anropet.
- Skillnaden mellan `Display` och `Displayable` är att `Display`-klassen representerar skärm hårdvaran medans `Displayable` är något som kan visas på skärmen.
- MIDlet:en kan anropa metoden `isShown()` i Displayable för att avgöra om innehållet verkligen ska visas på skärmen.

Screen

- Form
  - Viktigaste subklassen till Screen. Kan innehålla ett godtyckligt antal Items
- Alert
  - Möjlighet att visa en dialog under en viss tid, eller tills användaren bekräftar meddelandet. Har en titel. Kan även ha en innehållssträng och bild.
- List
  - En lista där en eller flera värden kan vara valda
- TextBox
  - Textinmatning. Liknande TextField, men kan vara flera rader

Items

- `ChoiceGroup`
  - Samling av radiobuttons eller checkboxes
- `TextField`
  - För att mata in en rad med text (eller siffror). Finns möjlighet att begränsa vad som kan matas in och att inte visa upp vad som skrivits in (tex för lösenord)
- `StringItem`
  - För att visa en sträng. Kan endast ändras av programmet
- `ImageItem`
  - För att visa upp en bild. Det enda bildformatet som måste stödjas är .png

Items forts.

- `Gauge`
  - Grafisk representation av ett värde. Kan antingen vara interaktiv (alltså tillåta användaren att ändra värdet) eller inte.
- `DateField`
  - För att mata in datum och/eller tider

Händelser

- Det finns två olika typer
  - Command och Item
- Command-händelser svarar mot kommandon (soft buttons) som man kan lägga till till sin applikation
- Item-händelser uppstår då användaren förändrar värden i de lika typerna av komponenter (som kan få sina värden ändrade)

Command-händelser

- Tre grundsteg
  - Skapa ett `Command`-objekt
  - Lägg till det till en `Form, TextBox, List` eller `Canvas`
  - Sätt en en lyssnare (som implementerar interfacet `ItemCommandListener`) till att lyssna efter händelsen
- Då händelsen inträffar anropas metoden `public void commandAction(Command c, Displayable s)` hos lyssnaren. I denna metod så kan man via den `Command`-referens som skickas avgöra vilket `Command`-objekt som genererade händelsen, och hantera detta på det sätt man vill i sitt program

Command-händelser

- Skapa Command-objektet
  - `exitCmd = new Command("Exit", Command.EXIT,0);`
- Lägg till det
  - `mForm.addCommand(exitCmd);`
- Lägg till lyssnare
  - `mForm.setCommandListener(this);`

Command-händelser för Items

- Även Items kan ha Command:s associerade med sig.
- Dessa kommer vara tillgängliga endast då det Item-objektet är valt
- Command-objektet läggs till mha `addCommand` i Item-klassen
- En lyssnare måste registreras mha `setItemCommandListener(ItemCommandListener l)` hos Item-objektet
- Och när händelsen inträffar anropas `commandAction(Command c, Item item)` hos lyssnaren

Item-händelser

- Item händelser genereras då värdet på Items ändas.
- Registrera en lyssnare på händelsen genom att lägga till en lyssnare i Form-objektet som visar komponenten mha metoden `setItemStateListener(ItemStateListener iListener)` i Form.
- Då händelsen inträffas anropas metoden `itemStateChanged(Item item)` hos lyssnaren. Mha Item-referensen kan man avgöra vilket Item som genererade händelsen (och tex kolla Item:ets värde)

UI med lite lägre abstraktionsnivå

- Ibland räcker inte klasserna som vi tittat på hittills inte riktigt till för allt man vill kunna göra…
- Vi vill själv kunna styra lite mer detaljerat hur våra program ska se ut
- … och vad som ska hända då användaren interagerar med vårt program

Canvas

- `Canvas`-klassen är en basklass för att skriva program som behöver hantera lågnivåhändelser och rita på displayen.
- Abstrakt klass=> vi måste ärva från den och omdefiniera den abstrakta metoden `public void paint(Graphics g)`
- För att begära att Canvas:en ska ritas om anropas metoden `repaint()`
- Kan sättas i helskärmsläge mha `setFullScreenMode(boolean mode)`

Graphics

- javax.microedition.lcdui.Graphics
- Klass för att rita på tex Canvasens rityta
- Lite mer primitiv än Graphics i AWT
- Ett `Graphics`-objekt fås som parameter till `paint`-metoden
- Metoder för att rita linjer, geometriska figurer, text (i olika fonter)
- Vi kan ställa in i vilken färg saker ska ritas mha metoderna `setColor` (ingen Color-klass finns i ME utan man får jobba mha int:s istället)

Lågnivåhändelser i Canvas

- Canvas har ett antal metoder som anropas vid olika lågnivåhändelser och som kan omdefinieras till att utföra det man vill ska inträffa vid händelsen.
  - `showNotify()`
  - `hideNotify()`
  - `keyPressed(int keyCode)`
  - `keyRepeated(int keyCode)`
  - `keyReleased(int keyCode)`
  - `pointerPressed(int x,int y)`
  - `pointerDragged(int x,int y)`
  - `pointerReleased(int x,int y)`
- Till key metoderna skickas en int som svarar mot unicode värdet för motsvarande tecken (konstanter finns i klassen för de vanligaste) om sådant finns, eller ett negativt värde för andra tecken