LCDUI. Baix nivell

Autor: Sergi Tur Badenas

Java ME API docs

- **És molt útil disposar de la documentació (javadocs) de Java ME**
  - Les podeu trobar online a:

    http://java.sun.com/javame/reference/apis.jsp#api

  - Amb Netbeans trobareu la documentació a:

    ~/netbeans-6.5/mobility8/WTK2.5.2/docs/api

  - LCDUI forma part del perfil MIDP. Dues versions
    - MIDP 1.0 (JSR 37)
    - MIDP 2.0 (JSR 118)
  - Consulteu
    - Wiki del curs

Autor: Sergi Tur Badenas

Display

- **La interfície gràfica amb CLDC es realitza amb l'API LCDUI**
  - LCDUI = Limited Connected Devices User Interface
  - El codi es troba al paquet **javax.microedition.lcdui**
- **L'objecte Display representa una pantalla de mòbil**
  - Permet accedir a la pantalla i al teclat
  - Cada MIDlet té associat un sol D*isplay*
- **Es pot obtenir el display amb:**

```
Display display = Display.getDisplay(midlet);
```

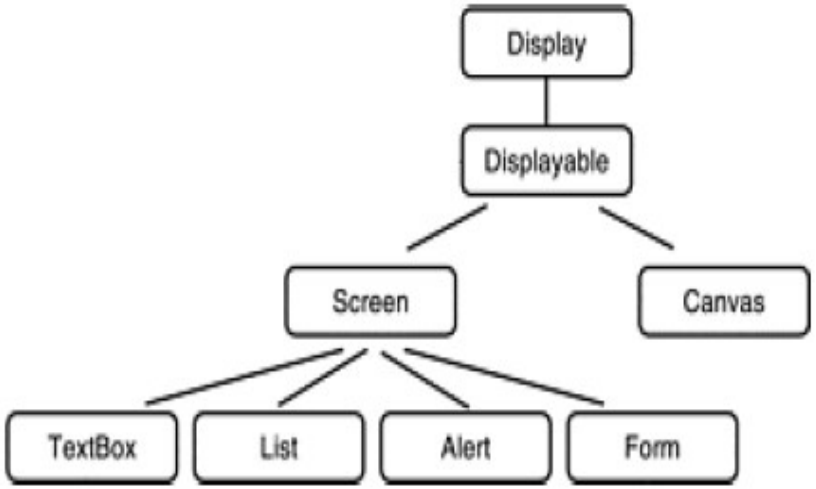
  - Normalment s'executa des de un MIDlet i pot ser un Bean Pattern:

```
public Display getDisplay () {
        return Display.getDisplay(this);
}
```


Autor: Sergi Tur Badenas

Components gràfics

- **Són els objectes que es poden mostrar per pantalla**
  - Tots són extensions (hereten) de la classe abstracta Displayable
  - Només es pot mostrar un component gràfic a l'hora.
  - **Canvas (baix nivell)**
    - Dibuix lliure
  - **Screen (alt nivell)**
    - Pantalles de la interfície gràfica



Autor: Sergi Tur Badenas

LCDUI. Baix nivell

- **MIDP 1.0**
  - Falta de suport per a aritmètica de punt flotant fa molt complicat implementar jocs en 3 dimensions.
  - No hi ha suport per a àudio. Només suporta fer "bips"
  - No és possible llegir o escriure en un píxel concret fet que no permet fer manipulacions bàsiques de imatges.
  - La classe Graphics no suporta cap tipus de transparència
  - L'únic protocol suportat és HTTP.
- **MIDP 2.0:**
  - Proveeix de l'api javax.microedition.lcdui.game

Autor: Sergi Tur Badenas

Dibuixant per pantalla

- **Objecte Canvas (llenç)**
  - El procés de pintat sobre un llenç és molt similar al procés de pintat amb AWT o Swing.
  - La idea bàsica és que l'objecte Canvas proporciona una àrea de dibuix que és accedida cada cop que el **mètode paint()** és invocat.

```
protected abstract void paint(Graphics g)
```

  - Les tasques de dibuix són realitzades per una classe anomenada **Graphics**. Aquesta classe proporciona mètodes primitius per a dibuixar sobre el llenç.
  - Wiki del curs

Autor: Sergi Tur Badenas

Dibuixant per pantalla

- **Canvas personalitzats**
  - La classe llenç (Canvas) és abstracta. Cal implementar una classe filla de Canvas que implementi el mètode paint():

```
public class MyCanvas
 extends Canvas {
 ...
  public void paint(Graphics g) {
  //Aquí posem el codi per pintar al Canvas
  }
 ...
 }
```


Autor: Sergi Tur Badenas

L'objecte Graphics

- **Proporciona primitives de dibuix.**
  - **Línies**: *drawLine()*
  - **Arcs**: *drawArc()*. Pot ser omplert per colors sòlids.
  - **Rectangles**: *drawRect()* i *drawRoundRect()*. Pot ser omplert per colors sòlids (mètodes *fillX*). També poden tenir els cantons arrodonits.
  - Totes les primitives poden ser pintades amb línia sòlides (SOLID) o puntejades (DOTTED).

```
public void setStrokeStyle(int style)
```


Autor: Sergi Tur Badenas

L'objecte Graphics

- **Objectes complexos**
  - **Imatges**: mètode drawImage()

```
public void drawImage(Image img, int x, int y,int anchor)
```

  - **Text:**

```
public void drawString(String img, int x, int y,int anchor)
public void drawChars(char[] data, int offset, int length, int x, int y, int anchor)
public void drawChar(char character, int x, int y, int anchor)
Public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)
```

  - **Transparències:**

```
drawRGB(int[] rgbData, int offset, int scanlength, int x, int y, int width, int height,
boolean processAlpha)
```

  - **Regions:**

```
drawRegion(Image src, int x_src, int y_src, int width, int height,
int transform, int x_dest, int y_dest, int anchor)
```


Autor: Sergi Tur Badenas

L'objecte Graphics

- **Espai de 2 dimensions**
  - Coordenades (X,Y) on la coordenada (0,0) és el cantó superior-esquerre de la pantalla.
- **Anchor points**
  - Per pintar imatges i text s'utilitzen anchor points
  - Permeten col·locar les imatges i el text en posicions molt concretes de la pantalla.

```
g.drawString(mMessage, mWidth / 2, mHeight, Graphics.BOTTOM | Graphics.HCENTER);
```

  - Es poden combinar és valors (amb |):
    - **Posició horitzontal**: (LEFT, HCENTER, RIGHT)
    - **Posició vertical**: (TOP, BASELINE, BOTTOM)

Autor: Sergi Tur Badenas

Color

- **Model de color RGB (red,green,blue)**
  - Establir el color

```
public void setColor(int red, int green, int blue);
public void setColor(int RGB);
public void setGrayScale(int value);
```

  - Els colors es codifiquen en 24 bits. Podem trobar amb dispositius amb limitacions:
    - El dispositiu no suporta colors. Pot ser blanc o negre o d'escala de grisos.
    - El número màxim de colors diferents és limitat
  - Consultar el color actual amb:

```
public int getColor()
```

  - O el valor de gris amb:

```
public int getGrayScale()
```


Autor: Sergi Tur Badenas

Color

- **Es recomana comprovar el tipus de dispositiu abans de començar a utilitzar colors.**
  - Mètodes de la classe Display per comprovar les limitacions del dispositiu

```
public boolean isColor();
public int numColors();
```

  - **isColor():** torna true si el dispositiu suporta colors
  - **numColors():** retorna el número de colors que són suportats pel dispositiu. Si el dispositiu no suporta colors, aleshores retorna el nombre de nivells de gris que suporta (en pantalles blanc i negre aquest valor és 2).

Autor: Sergi Tur Badenas

Fonts

- **Objecte Font**
  - Controla el tipus de fonts que es poden utilitzar. Hi han diversos mètodes per dibuixar text:

```
public void drawString(String str, int x, int y, int anchor)
public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)
public void drawChar(char character, int x, int y, int anchor)
```

- **La font actual és consulta amb:** `public Font getFont();`
- **I s'estableix amb:** `public void setFont(Font font);`

```
font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN, Font.SIZE_LARGE);
```


Autor: Sergi Tur Badenas

Fonts

- **No s'indica un tipus de font concret**
  - S'especifiquen característiques generals de la font
  - El mòbil retornarà la font més similar de la que disposi.
- **Tenim mètodes que ens permeten consultar l'alçada de la font i així pode escriure diverses línies sense xocar. Un exemple:**

```
height = f.getHeight();
for (i = 0; i < txt.length; i++) {
  g.drawString(txt[i], x, y, TOP|LEFT);
  y += height;
}
```


Autor: Sergi Tur Badenas

Repintar

- **Mètode paint**
  - És cridat pel propi mòbil cada cop que necessita tornar a mostrar la pantalla. Per tant, **NO** és el MIDlet qui controla quan s'ha de tornar a dibuixar la pantalla.
- **En alguns cassos voldrem forçar el repintat**
  - Per a realitzar animacions per pantalla
  - Un joc que necessita fer aparèixer un nou element gràfic per pantalla (per exemple un enemic)...

La instància de l'objecte **Graphics** utilitzat pel **mètode paint**, només és vàlida durant la execució d'aquest mètode. No podem accedir a l'objecte Graphics directament i aleshores invocar el mètode paint.
El mètode paint no és mai cridat directament pel MIDlet.

Autor: Sergi Tur Badenas

Repintar

- **Mètode repaint()**
  - Aquest mètode no crida el mètode paint() directament sinó que demana a l'entorn d'execució que cridi al mètode paint.
- **Exemple:**

```
public class MyCanvas extends Canvas {
   private String mMessage=null;
   ...
   public void setMessage(String s) {
     mMessage = s; repaint();
   }
   public void paint(Graphics g) {
     if (mMessage != null)
       g.drawString(mMessage, mWidth / 2, mHeight, Graphics.BOTTOM | Graphics.HCENTER);
   ...
   }
 ...
 }
```


Autor: Sergi Tur Badenas

Double-Buffering

- **Animacions complexes**
  - És possible que per problemes de rendiment noteu un **efecte indesitjable de parpelleig**.
  - **Solució**: la tècnica anomenada **double-buffering**.

Cal tenir en compte que hi ha dispositius que ja suporten per defecte double-buffering

- **MIDP suporta double-buffering**
  - S'implementa mitjançant la classe **Image**
  - Es crea una imatge off-screen
  - Es pinta sobre la imatge directament
  - Després es mostra la imatge a la pantalla.

Autor: Sergi Tur Badenas

Double-buffering

- **Exemple de double-buffering**

```
private Image offscreen;
...
offscreen = Image.createImage(getWidth(), getHeight());
g = offscreen.getGraphics();
paint(g);
...
 public void paint(Graphics g) {
 ...
 originalG = g;
 g = offscreen.getGraphics();
 ...
 }
```


Autor: Sergi Tur Badenas

Control d'esdeveniments

- **En Java per tractar els esdeveniments s'utilitzen *Listeners***
  - Els canvas també suporten l'ús de comandes
  - Funciona exactament igual que en LCDUI d'alt nivell
- **Tres tipus nous d'esdeveniments**
  - **Esdeveniments de teclat (keystrokes)**
  - **Accions de joc (command actions)**
  - **Esdeveniments de punter**
- **El Canvas ha de sobrescriure certs mètodes**

```
protected void keyPressed(int keyCode)
protected void keyReleased(int keyCode)
protected void keyRepeated(int keyCode)
protected void pointerDragged(int x, int y)
protected void pointerPressed(int x, int y)
protected void pointerReleased(int x, int y)
```


Autor: Sergi Tur Badenas

Esdeveniments de teclat (Keystrokes)

- **L'objecte Canvas pot controlar esdeveniments de teclat amb els mètodes**

```
protected void keyPressed(int keyCode)
protected void keyReleased(int keyCode)
protected void keyRepeated(int keyCode)
```

  - Es sobrescriu aquests mètodes a les classes filles de Canvas que utilitzeu. Els control d'esdeveniments per defecte d'aquests mètodes és no fer res.
  - **Constants predefinides**
    - **Tecles numèriques**: KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5...
    - **Asterisc i coixinet**: KEY_STAR,KEY_POUND

Són codis estàndards de la ITU-T. Poden existir altres tecles amb altres codis però si es vol garantir la portabilitat de l'aplicació només podem utilitzar les tecles estàndard.

Autor: Sergi Tur Badenas

Accions de joc (game actions)

- **Portabilitat**
  - Cal utilitzar accions de joc en comptes de codis de tecla per tal de fer aplicacions portables
- **MIDP defineix les següents accions de joc:**
  - **Fletxes de direcció**: UP, DOWN, LEFT, RIGHT
  - **Altres accions**: FIRE, GAME_A, GAME_B, GAME_C i GAME_D
  - Cada codi de tecla està associat com a màxim a una acció de joc però una acció de joc pot estar associada a més d'un codi de joc.
- **Mètodes:**

```
int getGameAction(int keyCode)
Int getKeyCode(int gameAction)
```


Autor: Sergi Tur Badenas

Accions de joc (game actions)

- **Cal tenir en compte que**

```
g == getGameAction(getKeyCode(g))     // (1)
k == getKeyCode(getGameAction(k))     // (2)
```

  - La expressió 1 sempre serà vàlida. No podem dir el mateix de la expressió (2).
  - Cal tenir en compte que per exemple les tecles de fletxa poden ser especifiques de cada dispositiu, i que alguns dispositius sense tecles de fletxa potser retornen els codis de les tecles 2,4,6 i 8 com a tecles de fletxa.

Autor: Sergi Tur Badenas

Accions de punter

Cal que tenir en compte que no tots els mòbils suporten esdeveniments de punter

- **Control d'esdeveniments de punter:**

```
protected void pointerDragged(int x, int y)
protected void pointerPressed(int x, int y)
protected void pointerReleased(int x, int y)
```

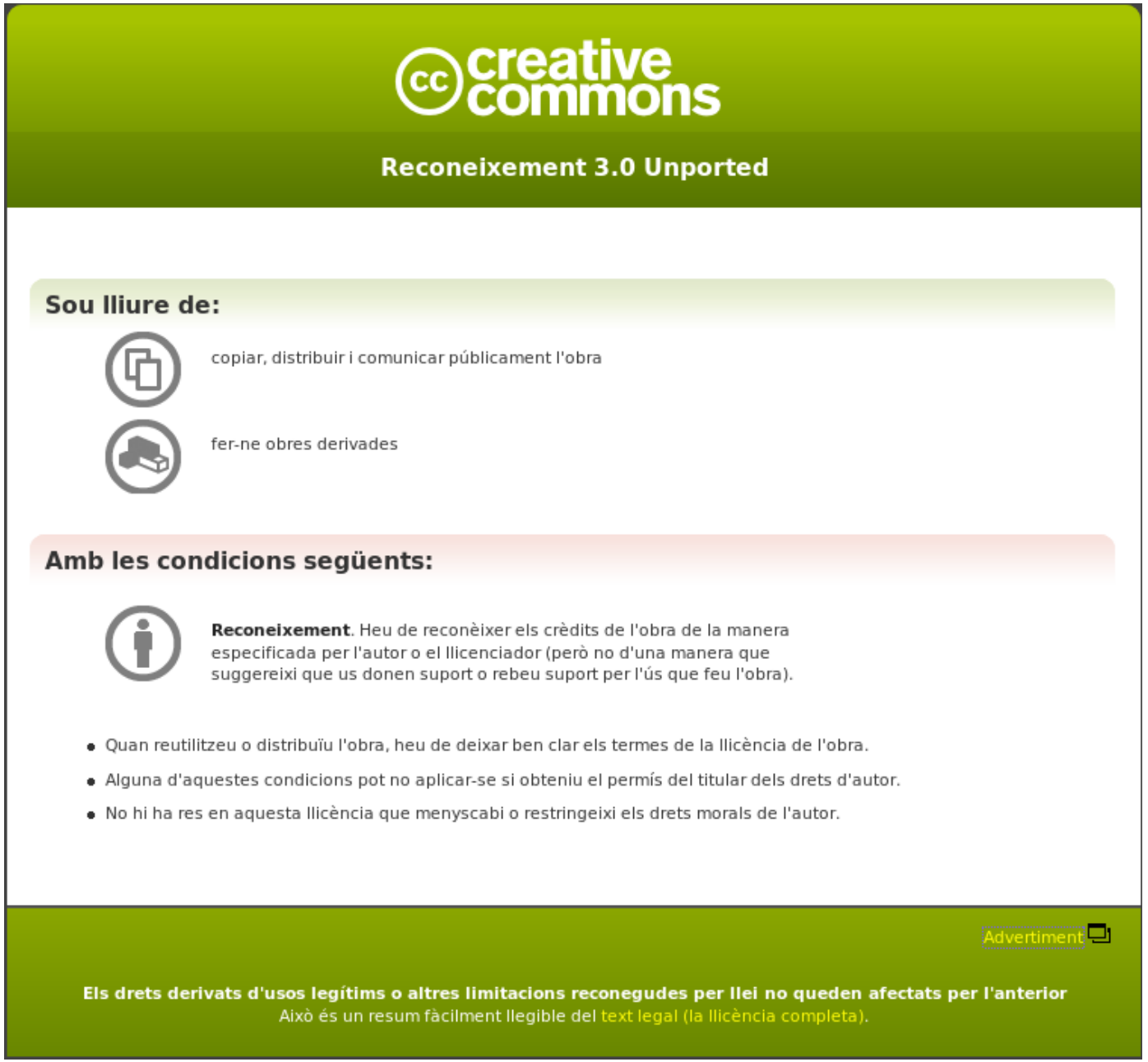
- **Comprovació del suport del mòbil**

```
public boolean hasPointerEvents()
public boolean hasPointerMotionEvents()
```

  - El segon mètode comprova si el mòbil permet controlar drags (arrossegament) del punter.

Autor: Sergi Tur Badenas

Llicència Creative Commons



http://creativecommons.org/licenses/by/3.0/deed.ca

Autor: Sergi Tur Badenas