Tema 5: Programación de dispositivos móviles con Java (J2ME)

Parte II: Interfaz de usuario

Tecnologías Web

1. Lenguaje Java para MIDs

Diferencias y parecidos con J2SE

Tecnologías Web

Configuración CLDC

- Características básicas del lenguaje
  - Mantiene la sintaxis y tipos de datos básicos del lenguaje Java
- No soporta números reales
  - No existen los tipos `float` y `double`
- Similar a la API de J2SE
  - Mantiene un pequeño subconjunto de las clases básicas de J2SE
    - Con una interfaz más limitada en muchos casos
  - Soporta hilos, excepciones, etc

CLDC y los números reales

- En CLDC 1.0 no tenemos soporte para números reales
  - Los tipos `float` y `double` no existen
- En muchas aplicaciones podemos necesitar trabajar con este tipo de números
  - P.ej. para cantidades monetarias
- Podemos implementar números de coma fija usando enteros
  - Existen librerías como *MathFP* que realizan esta tarea
- En CLDC 1.1 ya existe soporte para los tipos `float` y `double`

Números reales sobre enteros

- Podemos representar números de coma fija como enteros
  - Consideramos que los últimos N dígitos son decimales
  - Por ejemplo, 1395 podría representar 13.95
- Podremos hacer operaciones aritméticas con ellos
  - Suma y resta
    - Se realiza la operación sobre los números enteros
    - El resultado tendrá tantos decimales como los operandos

      13.95 + 5.20 → 1395 + 520 = 1915 → 19.15
  - Multiplicación
    - Se realiza la operación sobre los números reales
    - El resultado tendrá tantos decimales como la suma del número de decimales de ambos operandos

      19.15 * 1.16 → 1915 * 116 = 222140 → 22.2140

Conversión de reales a enteros

- Deberemos convertir el entero a real para mostrarlo al usuario

```
public String imprimeReal(int numero) {
  int entero = numero / 100;
  int fraccion = numero % 100;
  return entero + "." + fraccion;
}
```

- Cuando el usuario introduzca un valor real deberemos convertirlo a entero

```
public int leeReal(String numero) {
  int pos_coma = numero.indexOf('.');
  String entero = numero.substring(0, pos_coma - 1);
  String fraccion = numero.substring(pos_coma + 1, pos_coma + 2);
  return Integer.parseInt(entero)*100+Integer.parseInt(fraccion);
}
```

Hilos

- Permiten realizar múltiples tareas al mismo tiempo
  - Cada hilo es un flujo de ejecución independiente
- Todos acceden al mismo espacio de memoria
  - Necesidad de sincronizar cuando se accede concurrentemente a los recursos
- Se pueden crear de dos formas:
  - Heredando de `Thread`
    - Problema: No hay herencia múltiple en Java
  - Implementando `Runnable`
- Utilizaremos hilos para
  - Dar respuesta a más de un evento simultáneamente
  - Permitir que la aplicación responda mientras está ocupada

Tecnologías Web

Heredando de Thread

- Crear una clase que herede de `Thread`
- Sobrescribir el método `run`

```
public class MiHilo extends Thread {
    public void run() {
        // Codigo de la tarea a ejecutar en el hilo
    }
}
```

- En este método introduciremos el código que será ejecutado por nuestro hilo
- Instanciar el hilo

```
Thread t = new MiHilo();
```

Tecnologías Web

Implementando Runnable

- Crear una clase que implemente `Runnable`

```
public class MiHilo implements Runnable {
    public void run() {
        // Codigo de la tarea a ejecutar en el hilo
    }
}
```

- Definir en el método `run` el código de la tarea que ejecutará nuestro hilo
- Crear un hilo a partir de la clase anterior

```
Thread t = new Thread(new MiHilo());
```

Tecnologías Web

Estados de los hilos

- Nuevo hilo
  - El hilo acaba de instanciarse
- Hilo vivo
  - Llamando al método `start` pasa al estado de hilo vivo

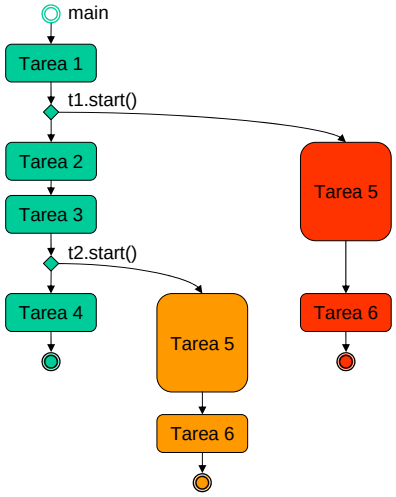
```
t.start();
```

  - El hilo se comienza a ejecutar
- Hilo muerto
  - Se ha terminado de ejecutar el código del método `run`
  - Se detiene la ejecución del hilo

Tecnologías Web

Ejemplo de hilos

```
public void main(String [] args) {
   tarea1();
   Thread t1 = new MiHilo();
   t1.start();
   tarea2();
   tarea3();
   Thread t2 = new MiHilo();
   t2.start();
   tarea4();
}

class MiHilo extends Thread {
   public void run() {
         tarea5();
         tarea6();
   }
}
```



Tecnologías Web

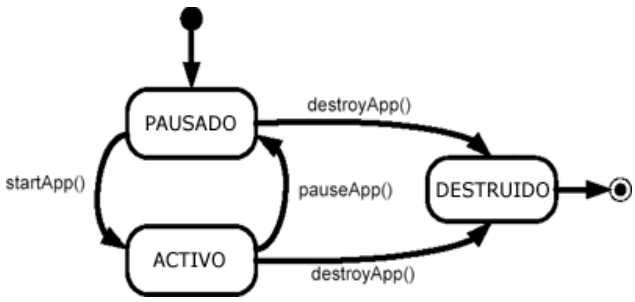
2. MIDlets

Ciclo de vida y propiedades

Tecnologías Web

Ciclo de vida

- La clase principal de la aplicación debe heredar de `MIDlet`
- Componente que se ejecuta en un contenedor
  - AMS = Software Gestor de Aplicaciones
- El AMS controla su ciclo de vida



Tecnologías Web

Esqueleto de un MIDlet

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

public class MiMIDlet extends MIDlet {
  protected void startApp()
              throws MIDletStateChangeException {
        // Estado activo -> comenzar
  }

  protected void pauseApp() {
        // Estado pausa -> detener hilos
  }

  protected void destroyApp(boolean incondicional)
              throws MIDletStateChangeException {
        // Estado destruido -> liberar recursos
  }
}
```

Tecnologías Web

Propiedades

- Leer propiedades de configuración (JAD)

```
String valor = getAppProperty(String key);
```

- Salir de la aplicación

```
public void salir() {
  try {
        destroyApp(false);
        notifyDestroyed();
  } catch(MIDletStateChangeException e) { }
}
```

Tecnologías Web

3. Interfaz gráfica

Display y pantallas

Tecnologías Web

Display

- La interfaz gráfica se realizará con la API LCDUI
  - LCDUI = *Limited Connected Devices User Interface*
  - Se encuentra en el paquete `javax.microedition.lcdui`
- El *display* representa el visor del móvil
  - Nos permite acceder a la pantalla
  - Nos permite acceder al teclado
- Cada MIDlet tiene asociado uno y sólo un *display*

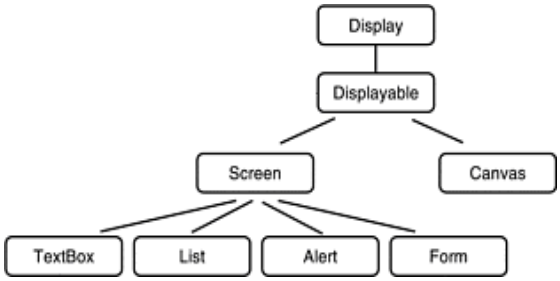
```
Display display = Display.getDisplay(midlet);
```

- El *display* sólo mostrará su contenido en la pantalla y leerá la entrada del teclado cuando el MIDlet esté en primer plano

Tecnologías Web

Componentes displayables

- Son los elementos que pueden mostrarse en el *display*
- El *display* sólo puede mostrar un *displayable* simultáneamente



- Establecemos el *displayable* a mostrar con

```
display.setCurrent(displayable);
```

Tecnologías Web

Alto nivel vs Bajo nivel

- Podemos distinguir dos APIs:
  - Alto nivel
    - Componentes predefinidos: listas, formularios, campos de texto
    - Se implementan de forma nativa
    - Aplicaciones portables
    - Adecuados para *front-ends* de aplicaciones corporativas
  - Bajo nivel
    - Componentes personalizables: *canvas*
    - Debemos especificar en el código cómo dibujar su contenido
    - Tenemos control sobre los eventos de entrada del teclado
    - Se reduce la portabilidad
    - Adecuado para juegos

Tecnologías Web

Campos de texto

```
TextBox tb = new TextBox("Contraseña",
             "", 8, TextField.ANY |
             TextField.PASSWORD);

Display d = Display.getDisplay(this);

d.setCurrent(tb);
```

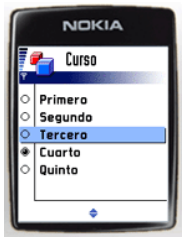
Tecnologías Web

Listas

```
List l = new List("Menu",
                  Choice.IMPLICIT);
l.append("Nuevo juego", null);
l.append("Continuar", null);
l.append("Instrucciones", null);
l.append("Hi-score", null);
l.append("Salir", null);

Display d =
   Display.getDisplay(this);

d.setCurrent(l);
```

Implícita

Múltiple

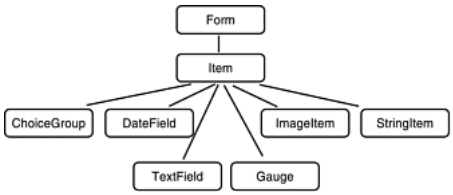


Exclusiva

Tecnologías Web

Formularios

```
Form f = new Form("Formulario");

Item itemEtiqueta = new StringItem(
        "Etiqueta:",
        "Texto de la etiqueta");
Item itemTexto = new TextField(
        "Telefono:", "", 8,
        TextField.PHONENUMBER);
Item itemFecha = new DateField(
        "Fecha",
        DateField.DATE_TIME);
Item itemBarra = new Gauge("Volumen",
        true, 10, 8);

f.append(itemEtiqueta);
f.append(itemTexto);
f.append(itemFecha);
f.append(itemBarra);

Display d = Display.getDisplay(this);
d.setCurrent(f);
```



Tecnologías Web

Alertas

- Mensaje de transición entre pantallas

```
Alert a = new Alert("Error",
       "No hay ninguna nota seleccionada",
       null, AlertType.ERROR);

Display d = Display.getDisplay(midlet);
d.setCurrent(a, d.getCurrent());
```

Tecnologías Web

Imágenes en MIDP

- En muchos componentes podemos incluir imágenes
- Las imágenes se encapsulan en la clase `Image`
- El único formato reconocido por MIDP es PNG
- Podemos crear una imagen de dos formas:
  - A partir de un fichero PNG contenido en el JAR

```
Image img = Image.createImage("/logo.png");
```

  - A partir de un array de bytes leído de un fichero PNG

```
Image img = Image.createImage(datos,offset,longitud);
```

- Para cargar imágenes de la red
  - Leemos un fichero PNG a través de la red.
  - Almacenamos los datos leídos en forma de array de bytes.
  - Creamos una imagen a partir del array de bytes

Tecnologías Web

Cargar imágenes de la red

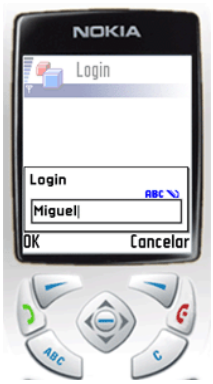
```
// Abre una conexion en red con la URL de la imagen
String url = "http://localhost:8080/subastas/imag/logo.png";
ContentConnection con =
                 (ContentConnection) Connector.open(url);
InputStream in = con.openInputStream();

// Lee bytes de la imagen
int c;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
while( (c=in.read()) != -1 ) {
  baos.write(c);
}

// Crea imagen a partir de array de bytes
byte [] datos = baos.toByteArray();
Image img = Image.createImage(datos,0,datos.length);
```

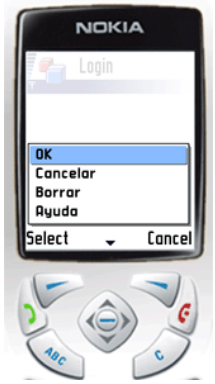
4. Comandos

Modelo de eventos e hilos

Comandos de entrada

- La entrada de usuario se realiza mediante comandos





Tecnologías Web

Creación de comandos

- Podemos crear comandos y añadirlos a un *displayable*

```
TextBox tb = new TextBox("Login", "", 8, TextField.ANY);
Command cmdOK = new Command("OK", Command.OK, 1);
Command cmdAyuda = new Command("Ayuda", Command.HELP, 1);
Command cmdSalir = new Command("Salir", Command.EXIT, 1);
Command cmdBorrar = new Command("Borrar", Command.SCREEN, 1);
Command cmdCancelar = new Command("Cancelar", Command.CANCEL, 1);

tb.addCommand(cmdOK);
tb.addCommand(cmdAyuda);
tb.addCommand(cmdSalir);
tb.addCommand(cmdBorrar);
tb.addCommand(cmdCancelar);

Display d = Display.getDisplay(this);
d.setCurrent(tb);
```

Tecnologías Web

Modelo de eventos

- En Java para tratar los eventos se utilizan listeners
- Listener
  - Componente que “escucha” un determinado evento
  - Cuando el evento sucede, se ejecuta el código que hayamos escrito en el listener
- Para crear un listener
  - Creamos una clase que implemente la interfaz correspondiente al tipo de listener deseado
    - Por ejemplo, `CommandListener` para “escuchar” la ejecución de comandos
  - Implementamos los métodos de la interfaz
    - Por ejemplo, `commandAction`
  - Registramos el listener en el componente del cuál queremos recibir eventos
- Una vez registrado, permanece a la “escucha” de eventos
  - Cada vez que se produzca un evento, se ejecutará el método correspondiente
    - Por ejemplo, si se pulsa un comando se ejecuta `commandAction`

Tecnologías Web

Listener de comandos

- Debemos crear un *listener* para dar respuesta a los comandos

```
class ListenerLogin implements CommandListener {
  public void commandAction(Command c, Displayable d) {
        if(c == cmdOK) {
                // Aceptar
        } else if(c == cmdCancelar) {
                // Cancelar
        } else if(c == cmdSalir) {
                // Salir
        } else if(c == cmdAyuda) {
                // Ayuda
        } else if(c == cmdBorrar) {
                // Borra r
        }
  }
}
```

- Registrar el *listener* en el *displayable*

```
tb.setCommandListener(new ListenerLogin());
```

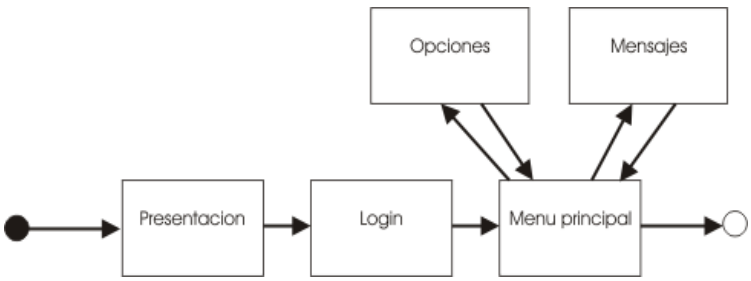
Tecnologías Web

5. Diseño de pantallas

Mapa de pantallas y patrones

Mapa de pantallas

- Cada *displayable* es una pantalla de la aplicación
- Conviene realizar un mapa de pantallas en la fase de diseño de la aplicación



Capa de presentación

- Conviene seguir un patrón de diseño para realizar la capa de presentación de nuestra aplicación
- Definiremos una clase por cada pantalla
- Encapsularemos en ella:
  - Creación de la interfaz
  - Definición de comandos
  - Respuesta a los comandos
- La clase deberá:
  - Heredar del tipo de *displayable* que vayamos a utilizar
  - Implementar `CommandListener` (u otros listeners) para dar respuesta a los comandos
  - Guardar una referencia al MIDlet, para poder cambiar de pantalla

Tecnologías Web

Creación de la pantalla

```
public class MenuPrincipalUI extends List implements CommandListener {

  ControladorUI controlador;
  Command selec;
  int itemNuevo;
  int itemSalir;

  public MenuPrincipalUI(ControladorUI controlador) {
    super("Menu", List.IMPLICIT);
    this.controlador = controlador;
    // Añade opciones al menu
    itemNuevo = this.append("Nuevo juego", null);
    itemSalir = this.append("Salir", null);
    // Crea comandos
    selec = new Command("Seleccionar", Command.SCREEN, 1);
    this.addCommand(selec);
    this.setCommandListener(this);
  }
  ...
```

Tecnologías Web

Respuesta a los comandos

- En la misma clase capturamos los eventos del usuario

```
...
public void commandAction(Command c, Displayable d) {
  if(c == selec || c == List.SELECT_COMMAND) {
    if(getSelectedIndex() == itemNuevo) {
      // Nuevo juego
      controlador.comenzarJuego();
    } else if(getSelectedIndex() == itemSalir) {
      // Salir de la aplicación
      controlador.salir();
    }
  }
}
}
```

Tecnologías Web

Controlador

- Controla transiciones entre pantallas y ejecuta acciones

```
public class ControladorUI {
  MiMIDlet owner;

  public Controlador(MiMIDlet owner) {
    this.owner = owner;
  }

  public void comenzarJuego() {
    Display display = Display.getDisplay(owner);
    PantallaJuego pj = new PantallaJuego(this);
    display.setCurrent(pj);
  }

  public void salir() {
    owner.salir();
  }
}
```

Tecnologías Web