Programación de Dispositivos Móviles

Sesión 3:
Interfaz gráfica

Índice

- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Interfaz gráfica

- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Display

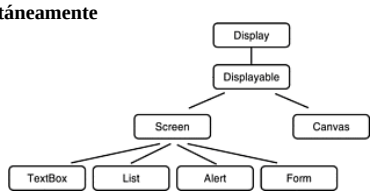
- La interfaz gráfica se realizará con la API LCDUI
  - LCDUI = *Limited Connected Devices User Interface*
  - Se encuentra en el paquete `javax.microedition.lcdui`
- El *display* representa el visor del móvil
  - Nos permite acceder a la pantalla
  - Nos permite acceder al teclado
- Cada MIDlet tiene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

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);
```


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

Interfaz gráfica

- Interfaz gráfica
- **Componentes de alto nivel**
- Imágenes
- Comandos
- Diseño de pantallas

Campos de texto

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

Display d = Display.getDisplay(this);

d.setCurrent(tb);
```

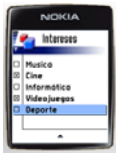
Listas

```
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

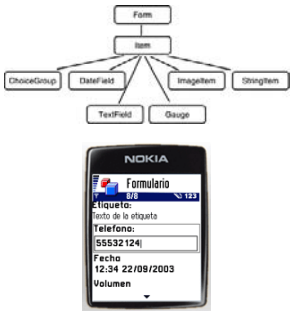
Formularios

```
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);
```



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());
```

Interfaz gráfica

- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Imágenes en MIDP

- En muchos componentes podemos incluir imágenes
- Las imágenes se encapsulan en la clase `Image`
- Encontramos dos tipos de imágenes
  - Imágenes mutables:
    - Podemos editar su contenido desde nuestra aplicación
    - Se crea con:

```
Image img_mut = Image.createImage(ancho, alto);
```

    - Al crearla estará vacía. Deberemos dibujar gráficos en ella.
  - Imágenes inmutables:
    - Una vez creada, ya no se puede modificar su contenido
    - En los componentes de alto nivel sólo podremos usar este tipo

Imágenes inmutables

- El único formato reconocido por MIDP es PNG
- Las imágenes inmutables se crean:
  - 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
    - Podemos leer un fichero PNG a través de la red.
    - Almacenamos los datos leídos en forma de array de bytes.

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

  - A partir de una imagen mutable
    - Nos permitirá usar en componentes de alto nivel imágenes creadas como mutables, y editadas en el código

```
Image img_inmut = Image.createImage(img_mut);
```


Interfaz gráfica

- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

Comandos de entrada

- La entrada de usuario se realiza mediante comandos

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);
```


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*

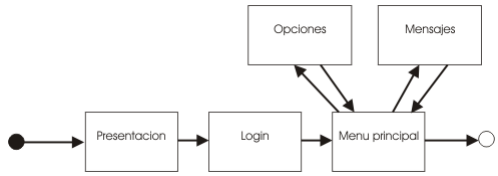
```
tb.setCommandListener(new ListenerLogin());
```

Interfaz gráfica

- Interfaz gráfica
- Componentes de alto nivel
- Imágenes
- Comandos
- Diseño de pantallas

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

Creación de la pantalla

```
public class MenuPrincipal extends List implements CommandListener {

  MiMIDlet owner;
  Command selec;
  int itemNuevo;
  int itemSalir;

  public MenuPrincipal(MiMIDlet owner) {
    super("Menu", List.IMPLICIT);
    this.owner = owner;

    // 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);
  }
  ...
```


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
        Display display = Display.getDisplay(owner);
        PantallaJuego pj = new PantallaJuego(owner, this);
        display.setCurrent(pj);
      } else if(getSelectedIndex() == itemSalir) {
        // Salir de la aplicación
        owner.salir();
      }
    }
  }
}
```