Informática Móvil

Diseño de aplicaciones con J2ME

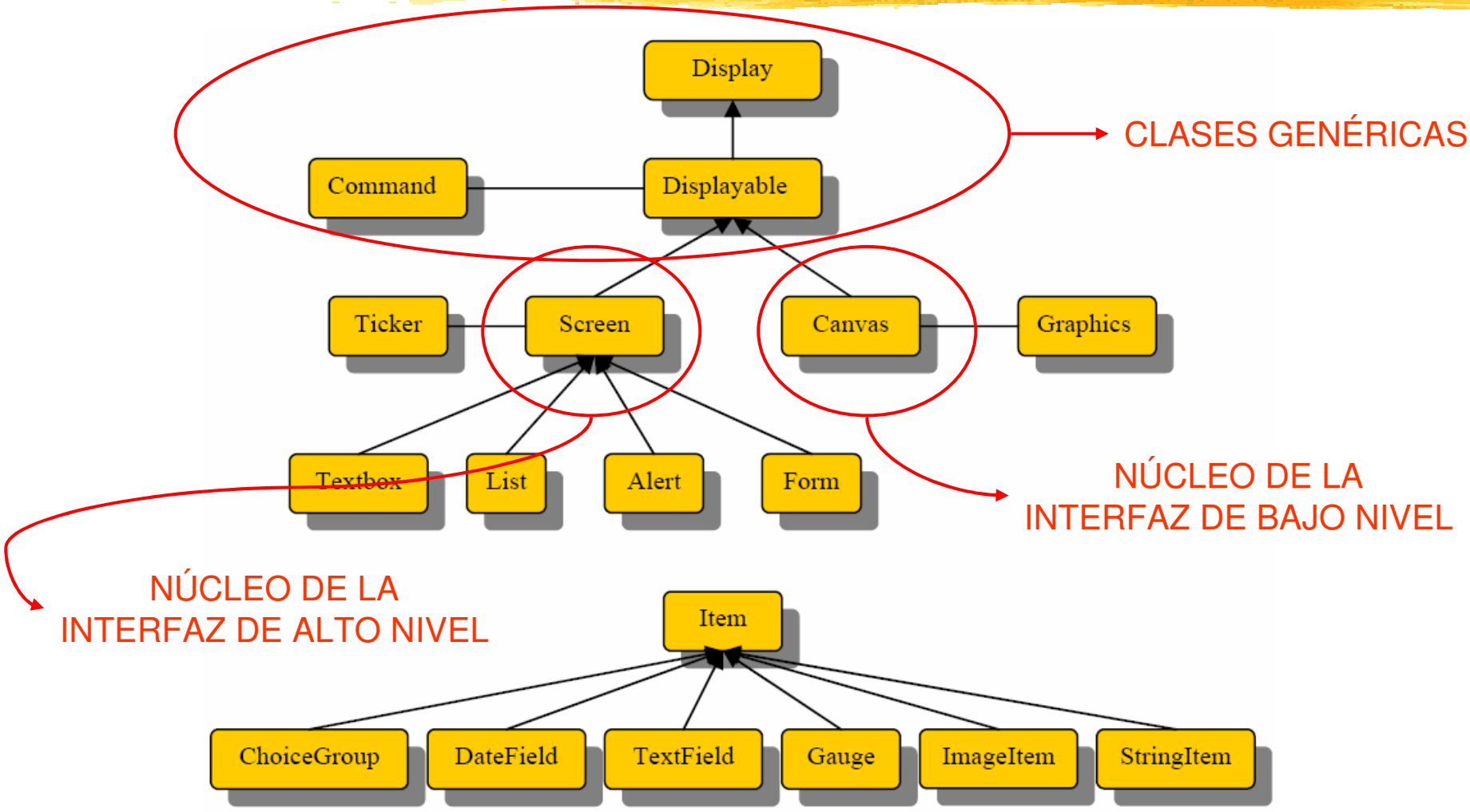
Interfaz de Usuario: API de Bajo Nivel

José Ramón Arias García
Ignacio Marín Prendes

UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005

Interfaz gráfica de usuario



Clases derivadas de Display e Item

Canvas

- Clase para representar objetos gráfico basada en el API de bajo nivel
  - *Screen*, para objetos gráficos basados en el API de alto nivel (controles)
  - El uso de ambas APIs no es exclusivo
    - Un mismo MIDLet puede ofrecer pantallas basadas en *Screen* y pantallas basadas en *Canvas*
- Permite manejar eventos de bajo nivel y dibujar cualquier tipo de elemento gráfico en pantalla

Ejemplo (I)

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

public class HolaMundo extends MIDlet {

   private MiClaseCanvas miCanvas;
   private Display pantalla;

   public HolaMundo() {
         pantalla = Display.getDisplay(this);
         miCanvas = new MiClaseCanvas(this);
    }

   public void startApp() {
         pantalla.setCurrent(miCanvas);
    }

    public void pauseApp() {} ;

    public void destroyApp() {} ;

    public void salir() {
        destroyApp(false);
        notifyDestroyed();
   }
}
```

HolaMundo.java

Ejemplo (II)

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

public class MiClaseCanvas extends Canvas implements CommandListener {

   private HolaMundo miHolaMundo;
   private Command salir;

   public MiClaseCanvas(HolaMundo miMidlet) {
        salir = new Command("Salir", Command.EXIT, 1);
        this.miHolaMundo = miMidlet;
        this.addCommand(salir);
        this.setCommandListener(this);
    }

   public void paint(Graphics g) {
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(0, 0, 0);
        g.drawString("¡Hola Mundo!", 10, 4,
                  Graphics.BASELINE | Graphics.HCENTER);
    }

    public void commandAction(Command c, Displayable d) {
        if (c==salir)
           miHolaMundo.salir();
    } ;
}
```

Encargado de repintar la pantalla

MiClaseCanvas.java

Eventos de bajo nivel

- Los eventos dentro de un *Canvas* se pueden gestionar de dos formas:
  - A través de *Commands*
    - Visto en el ejemplo anterior (gestión de eventos de alto nivel)
  - A través de códigos de teclas (*key codes*)
    - Gestión de eventos de bajo nivel
    - Son valores numéricos asociados a las diferentes teclas de un MID
      - (0-9,*,#)
      - Cada tecla tiene asociado un nombre simbólico al valor numérico

Tabla de códigos de teclas

NOMBRE SIMBÓLICO	VALOR
KEY_NUM0	48
KEY_NUM1	49
KEY_NUM2	50
KEY_NUM3	51
KEY_NUM4	52
KEY_NUM5	53
KEY_NUM6	54
KEY_NUM7	55
KEY_NUM8	56
KEY_NUM9	57
KEY_STAR	58
KEY_POUND	59

Métodos de gestión de pulsación

- Canvas proporciona una serie de métodos abstractos para gestionar estos eventos
  - Las clases que extienden a Canvas, deben implementarlos

Métodos	Descripción
boolean hasRepeatEvents()	Indica si el MID es capaz de detectar la repetición de teclas
String getKeyName(int codigo)	Devuelve una cadena de texto con el nombre del código de tecla asociado
void keyPressed(int codigo)	Se invoca cuando pulsamos una tecla
void keyReleased(int codigo)	Se invoca cuando soltamos una tecla
void keyRepeated(int codigo)	Se invoca cuando se deja pulsada una tecla

Métodos de gestión de dispositivo puntero

- Canvas proporciona una serie de métodos para conocer si el MID soporta algún dispositivo puntero
  - **boolean hasPointerEvents()**
    - Devuelve *true* si el dispositivo posee algún tipo de puntero
  - **boolean hasPointerMotionEvents()**
    - Devuelve *true* si el dispositivo puede detectar acciones como pulsar, arrastrar y soltar el puntero
- También métodos abstractos que deberán implementar las clases hijas de Canvas, para detectar eventos del dispositivo puntero
  - **void pointerDragged()**
    - Invocado al arrastrar el puntero
  - **void pointerPressed()**
    - Invocado al pulsar con el puntero
  - **void pointerReleased()**
    - Invocado cuando se deja de pulsar el puntero

Mecanismo de dibujo en pantalla

- El API de bajo nivel cumple dos funciones principales
  - Controlar los eventos de bajo nivel
  - Controlar qué aparece en pantalla en el dispositivo
- Gestión de dibujo de pantalla
  - Método abstracto paint() de la clase Canvas
    - No es llamado explícitamente (lo llama el AMS)
    - Para repintar la pantalla explícitamente, se debe llamar a *repaint()*
      - Esta petición se introduce en la cola de eventos
      - La aplicación atiende los eventos en orden de llegada
      - Otros eventos en la cola: eventos de teclado y de puntero
    - El MID no limpia la pantalla antes de llamar a paint()
      - *paint()* debe pintar cada pixel de la pantalla

Clase *Graphics*

- Clase que es pasada como parámetro de *paint()*
- Permite dibujar en una pantalla *Canvas*
- Se puede utilizar de dos maneras:
  - Dentro del método *paint()* de la clase *Canvas*
  - A partir de un objeto *Image*:

```
Image img = Image.createImage(width, height);
Graphics g = img.getGraphics();
```

- *Graphics* posee métodos para:
  - Seleccionar colores
  - Dibujar texto, líneas y figuras geométricas

Sistema de coordenadas

- Canvas proporciona dos métodos:
  - *getWidth()* – Obtiene el ancho de la pantalla
  - *getHeight()* – Obtiene el alto de la pantalla
- Posición de puntos:
  - Esquina superior izquierda: (0,0)
  - Esquina inferior derecha: (getWidth()-1, getHeight()-1)
  - Un punto intermedio: (x, y)
    - x = desplazamiento horizontal respecto de (0, 0)
    - y = desplazamiento vertical respecto de (0, 0)
- Se puede cambiar el origen de coordenadas:
  - Método *translate(int x, int y)* de la clase *Graphics*
    - Cambia el origen de coordenadas de (0, 0) a (x, y)
    - Provoca un desplazamiento de todos los objetos en pantalla
    - Útil para efectos de scroll

Gestión de colores (I)

- 2 métodos para detectar soporte de colores
  - *boolean Display.isColor()*
    - *true* indica pantalla en color; *false*, niveles de gris
  - *int Display.numColors()*
    - Número de colores o de niveles de gris soportados
- Graphics proporciona un modelo de color de 24 bits en formato RGB (8 bits para cada componente)
  - Método setColor (2 versiones sobrecargadas)
    - int Graphics.setColor(int RGB)
    - int Graphics.setColor(int rojo, int verde, int azul)

Gestión de colores (II)

- Ejemplos de llamadas a setColor()

```
Graphics g;
...
g.setColor(0,0,0);            // Selección de color negro
g.setColor(255,255,255); // Selección de color blanco
g.setColor(0,0,255);          // Azul
g.setColor(#00FF00);          // Verde
g.setColor(255,0,0);          // Rojo
g.fillRect(0,0,10,10);        // Rectángulo rojo comprendido
                              // entre los puntos (0,0) y (10,10)
```

Cómo dibujar texto

- Clase *Font*
  - Permite seleccionar un tipo de letra
    - La selección de tipo se hace en función de tres atributos: aspecto, estilo y tamaño
  - Fuentes disponibles:

    Aspecto: FACE_SYSTEM, FACE_MONOSPACE, FACE_PROPORTIONAL

    Estilo: STYLE_PLAIN, STYLE_BOLD, STYLE_ITALIC, STYLE_UNDERLINED

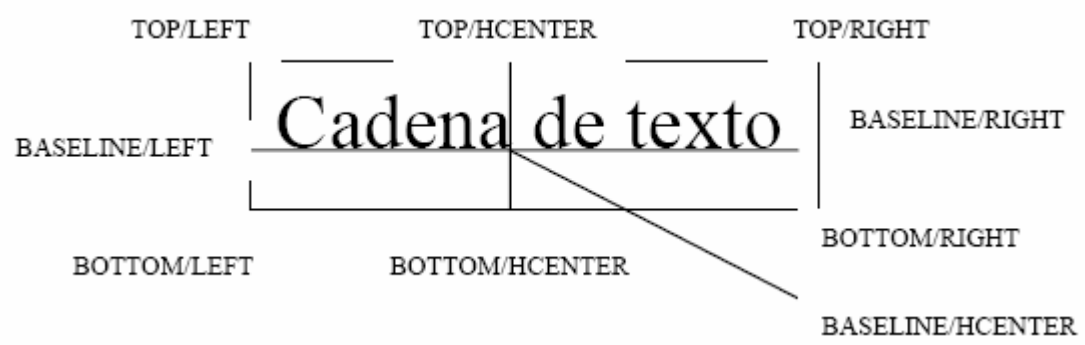
    Tamaño: SIZE_SMALL, SIZE_MEDIUM, SIZE_LARGE

```
Font fuente = Font.getFont(FACE_SYSTEM, STYLE_PLAIN, SIZE_MEDIUM);
g.setFont(fuente);
g.drawString("Hola mundo", 23, 45, BASELINE | HCENTER);
// Escribe la cadena de texto centrándola en el punto (23, 45)
```

Posicionamiento de texto

g.drawString("Hola mundo", 23, 45, BASELINE | HCENTER);

- Se indica el punto de anclaje: (23, 45)
- Se indica el alineamiento del texto respecto del punto de anclaje

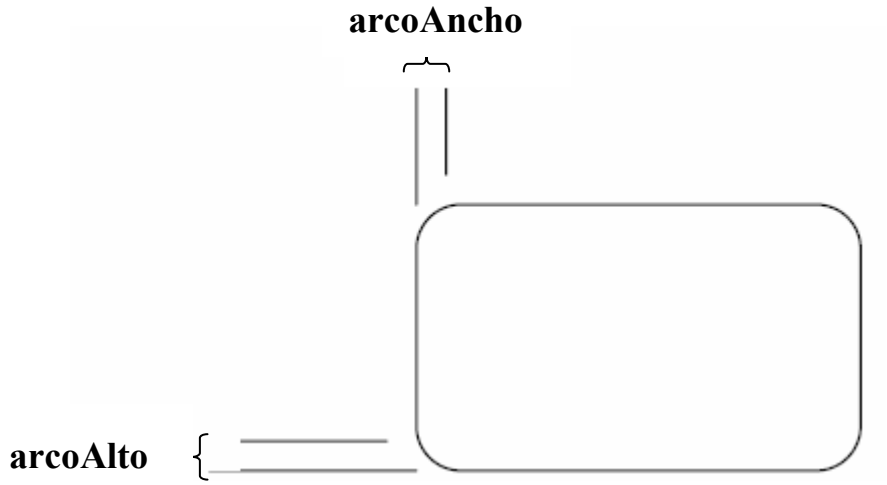


Representación de figuras geométricas

- Graphics permite dibujar 3 figuras geométricas:
  - Líneas
    - g.drawLine(x1,y1,x2,y2) // Línea de (x1,y1) a (x2,y2)
  - Rectángulos (hay 4 tipos)
    - Planos
      - g.drawRect(x,y,ancho,alto); // sin relleno
      - g.fillRect(x,y,ancho,alto); // con relleno
    - Redondeados
      - g.drawRoundRect(x,y,ancho,alto,arcoAncho,arcoAlto);
      - g.fillRoundRect(x,y,ancho,alto,arcoAncho,arcoAlto);
  - Arcos
    - Simple
    - Con color de relleno

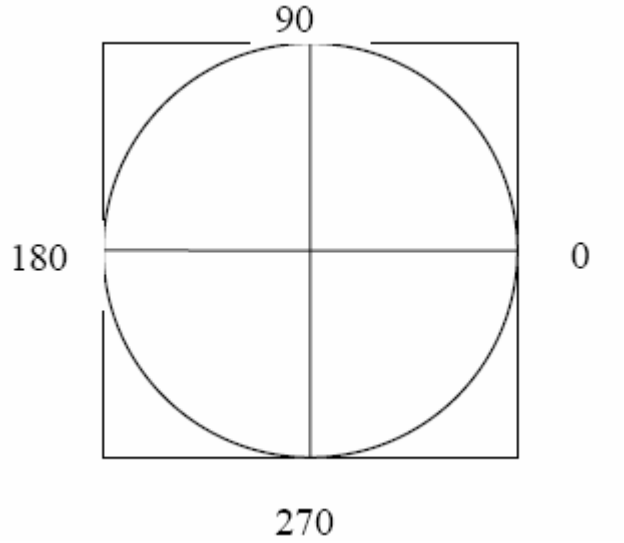
Rectángulos redondeados

g.drawRoundRect(x,y,ancho,alto,arcoAncho,arcoAlto);



Arcos

- 2 tipos: Simples y Con color de relleno
- El arco está virtualmente inscrito en una caja



- g.drawArc(1,1,20,20,0,270);
  - (1,1)-(20,20) Esquinas de la caja
  - 0º = Ángulo Inicial ; 270º = Ángulo Final

Arcos: Ejemplos

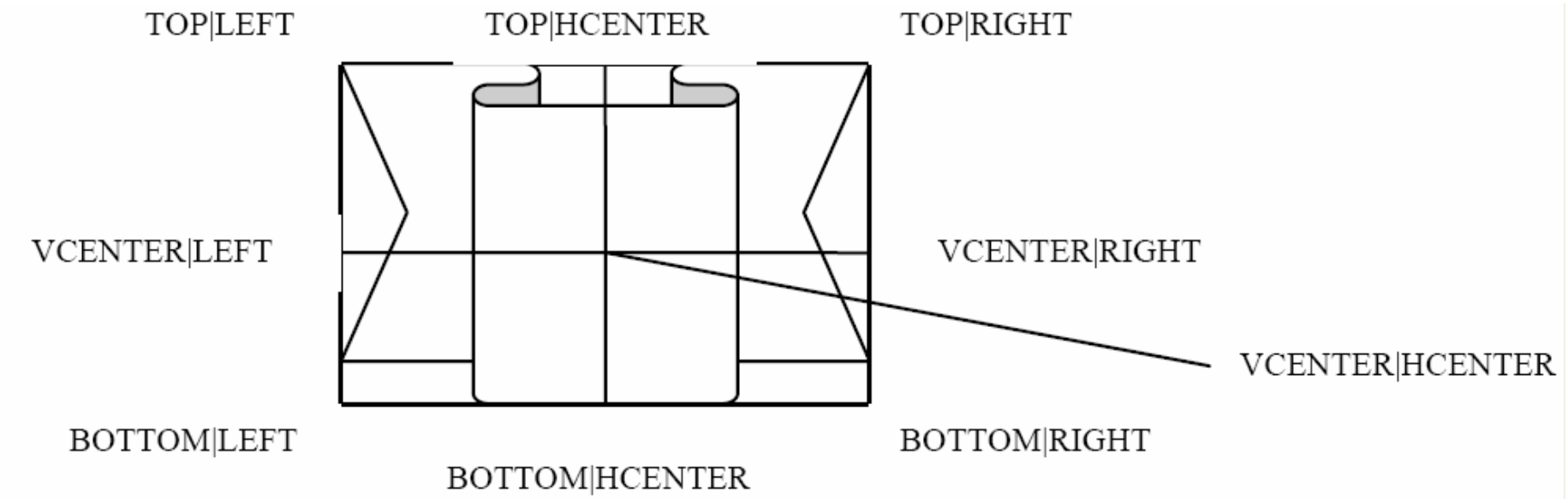
g.drawArc(1,1,getWidth()-1,getHeight()-1,0,270);

g.fillArc(1,1,getWidth()-1,getHeight()-1,90,290);

Imágenes

- 2 tipos distintos de objetos Image
  - Inmutables:
    - Image img = createImage("foto.png");
    - Creadas a partir de un mapa de bits en un fichero
    - No se puede variar su aspecto
  - Mutables:
    - *Image img = Image.createImage(75,25);*
    - Se trabaja sobre una matriz de puntos
    - Podemos crear la imagen deseada, modificarla, etc
- Mostrar imagen en pantalla
  - Graphics.drawImage(imagen,x,y,puntoAnclaje)

Puntos de anclaje para imágenes



Eventos de teclado para juegos

- Existen códigos genéricos de teclado relativos a entrada de usuario de un juego
  - El programador no tiene porqué saber a qué tecla física del MID está asociada una acción concreta
- Tabla de códigos:

Constante	Código
UP	1
DOWN	6
LEFT	2
RIGHT	5
FIRE	8
GAME_A	9
GAME_B	10
GAME_C	11
GAME_D	12

Eventos de teclado para juegos (II)

- Existen métodos para convertir los códigos de teclado genéricos a códigos de juego

Métodos	Descripción
int getKeyCode(int gameAction)	Devuelve el código genérico asociado *gameAction*.
int getGameAction(int keyCode)	Devuelve si existe el código de juego asociado a *keyCode*.
string getKeyName(int keyCode)	Obtiene el nombre del *keyCode*.

- Finalidad:
  - keyPressed, keyRepeated y keyReleased trabajan con códigos genéricos
  - Hay que usar getGameAction para traducir

Eventos de teclado para juegos (III)

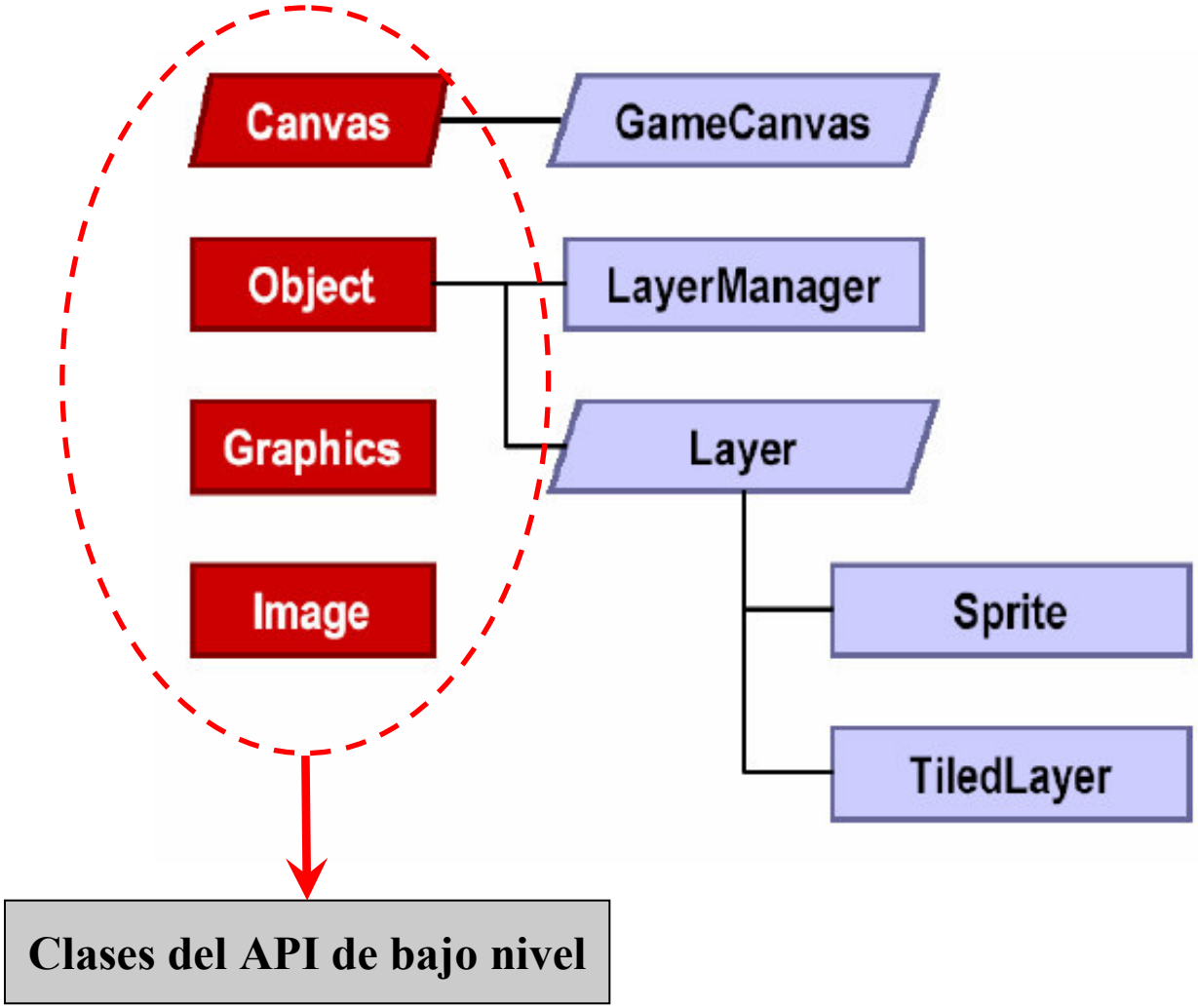
- Ejemplo:

```
protected void keyPressed(int keyCode) {
    switch (getGameAction(keyCode)) {
        case Canvas.FIRE: dibujarDisparo(); break;
        case Canvas.UP: dibujarSalto(); break;
        case Canvas.DOWN: dibujarAgachado(); break;
        ...
    }
}
```

javax.microedition.lcdui.game

- Paquete Java con diversas clases útiles en la creación de juegos, optimizadas para:
  - Aumentar rendimiento
  - Minimizar el trabajo del programador
  - Reducir el tamaño de la aplicación
- Contiene 5 clases que pueden utilizarse junto con las clases gráficas de bajo nivel
  - GameCanvas
  - Layer
  - LayerManager
  - Sprite
  - TiledLayer

Jerarquía de clases del API de juegos



Clase *GameCanvas*

- Clase abstracta que hereda de Canvas
- Incorpora funcionalidad adicional
  - Preguntar por el estado actual de las teclas
  - Sincronizar la visualización de gráficos en pantalla
- Debe implementarse el método ***public void run()***, encargado del bucle que gestiona el ciclo de vida del GameCanvas

Ejemplo de bucle en *GameCanvas.run()*

```
Graphics g=getGraphics();
while (true) {
    int teclaActual = getKeyStates();
    if ((teclaActual & LEFT_PRESSED) !=0) {
            sprite.move(-1,0);
    }else if ((teclaActual & RIGHT_PRESSED) !=0) {
            sprite.move(1,0);
    }else if ((teclaActual & DOWN_PRESSED) !=0) {
            sprite.move(0,-1);
    }else if ((teclaActual & RIGHT_PRESSED) !=0) {
            sprite.move(0,1);
    }else if ((teclaActual & FIRE_PRESSED) !=0) {
            dibujarDisparo();
    }
    sprite.paint(g);
    flushGraphics();
}
```

Clase *Layer*

- Representa una capa, un elemento visual en un juego. Es una clase abstracta:
  - Sus subclases deben implementar paint()
- Métodos:
  - int getHeight(), int getWidth()
  - int getX(), int getY()
  - boolean isVisible()
  - void move(int x, int y)
  - **abstract void paint(Graphics g)**
  - void setPosition(int x, int y)
  - void setVisible(boolean visible)

Clase *Sprite*

- Layer animado básico
  - Puede mostrar uno de entre varios "fotogramas"
  - Cada uno de esos *frames* es de igual tamaño y es proporcionado por un único objeto Image
- Contiene métodos para:
  - Detección de colisiones
  - Transformaciones (p.e., rotaciones)
- Además de mostrar los *frames* secuencialmente, puede fijarse una nueva secuencia de forma arbitraria

Clase *TiledLayer*

- Layer formado por una rejilla de celdas
  - Cada celda puede mostrar uno de entre varios elementos gráficos (*tiles*), representados por un objeto Image
  - Tile = "baldosa"
- Las celdas pueden ser rellenadas con tiles animados, con datos de pixels que pueden variar muy rápidamente
  - Útil para animar grandes grupos de celdas, como áreas de agua

Clase *LayerManager*

- Gestiona una serie de *Layer*s
  - Usa una lista ordenada de objetos Layer
  - Permite insertar o borrar un Layer
  - Acceso a los *Layer*s mediante un índice (cuanto más pequeño, más cercano a la vista del usuario)
  - El dibujo de cada capa se hace automáticamente

Técnicas útiles (I)

- Double Buffering
  - Elimina el problema del parpadeo al actualizar la pantalla
  - Idea: actualizaciones gráficas en memoria principal y volcar a pantalla
  - boolean isDoubleBuffered()
    - *true*: El dispositivo utiliza esta técnica automáticamente
    - *false*: Hay que implementar el doble buffer

Técnicas útiles (II)

- Double Buffering: esquema de implementación

```
// 1) Crear una imagen mutable del tamaño de la pantalla
Image pantalla2;
if (!isDoubleBuffered()) {
      pantalla2 = Image.createImage(getWidth(),getHeight());
...

// 2) Implementar paint(Graphics g) adecuadamente
protected void paint(Graphics g) {
   Graphics pantalla1 = g;
   g=pantalla.getGraphics();
   // Actualizar elementos gráficos sobre g
   ...
   // Volcar a la pantalla para visualizar
   pantalla1.drawImage(pantalla2, 0, 0, Graphics.LEFT | Graphics.TOP);
 }
```

Técnicas útiles (III)

- Clipping
  - Actualizar sólo la parte de la pantalla que se haya modificado
  - Se puede definir una región que limite qué parte de la pantalla se pintará al llamar a *paint()*
    - ***void setClip(int x, int y, int ancho, int alto)***
  - Sólo puede existir una zona de clipping por cada objeto Graphics
    - Cualquier operación realizada con este objeto que esté fuera de la zona de clipping, será ignorada
  - Otros métodos relacionados con clipping:
    - *getClipX()*, *getClipY()*, *getClipHeight()*, *getClipWidth()*

Informática Móvil

Diseño de aplicaciones con J2ME

Interfaz de Usuario: API de Bajo Nivel

José Ramón Arias García
Ignacio Marín Prendes

UNIVERSIDAD DE OVIEDO
Área de Arquitectura y Tecnología de Computadores

Curso 2004/2005