Programando juegos para PDA con Software Libre

David Fernández Vaamonde

<david_fv(en)gpul(punto)org>

Indice

Parte I: Conociendo SuperWABA

Introducción a SuperWABA

- ¿Qué es SuperWaba?
  - Subconjunto de clases **LIBRE** de Java
  - Máquinas virtuales **LIBRES** para distintas plataformas usadas en PDAs:
    - Symbian OS
    - Palm OS
    - WindowsCE
  - Herramientas **LIBRES** para generación del formato para cada Plataforma (.pdb, .exe, ...).

Introducción a SuperWABA

- ¿Qué es SuperWaba? (II)
  - Licenciado bajo **GPL** (Para uso no comercial :])
  - Muchos ejemplos y algunos tutoriales libres (otros comerciales).
  - Página del proyecto: http://www.superwaba.com.br

+

Creando un entorno de trabajo.

- Elementos necesarios (ejemplo en Palm):
  - Entorno de desarrollo
    - Superwaba SDK
    - Java (Java SUN, Java IBM, GCJ, Kaffe,...)
    - Ant o Make
  - Entorno de pruebas
    - Pose
    - Palm ROMs

Creando un entorno de trabajo (II)

- Entorno de desarrollo:
  - Java (cualquier implementación)
    - Compilador: *javac, gcj* (jikes NO)
    - Máquina Virtual: *Kaffe, GIJ, Java* (Sun o IBM)
    - Gestor del proyecto: *Ant, Make*
    - En este taller:
      - Compilador: ***gcj***
      - Máquina Virtual: ***java***
  - SuperWaba SDK (Clases Java y Binarios)
    - Descarga en: http://www.superwaba.com.br/en/downloads.asp

Creando un entorno de trabajo (III)

- Configurando el entorno de desarrollo:
  - Configurando PATH y CLASSPATH
    - `export SUPERWABA_ROOT=~/superwabasdk_gpl`
    - `export CLASSPATH="$SUPERWABA_ROOT/lib/SuperWaba.jar:$SUPERWABA_ROOT/lib/SuperWabaTools.jar:."`
    - `export PATH="$PATH:$SUPERWABA_ROOT/bin"`

Creando un entorno de trabajo (IV)

- Entorno de pruebas:
  - Emulador **POSE**
  - Palm ROMS (http://www.palmos.com/dev/tools/emulator/)
  - **Waba** Applet
- Configuración del entorno de Pruebas:
  - Pose con la siguientes opciones desactivadas:
    - Settings/Debugging:
      - Hardware Register Acccess
      - Proscribed Function Call
      - Screen Access

Creando un entorno de trabajo (V)

- Entorno de producción:
  - El propio PDA :)
- Configurando el entorno de producción:
  - Máquina virtual para cada plataforma (`$SUPERWABA_ROOT/lib/vm/<plataforma>`)
    - Palm:
      - `SuperWaba.prc, SuperWaba.pdf y SWNatives.prc` en / lib/vm/palm/...
    - Windows CE:
      - `superwaba.exe, SDL.dll, SDL_net.dll, SuperWaba.pdb, MSW.pdb` en lib/vm/ce y lib/vm/win32

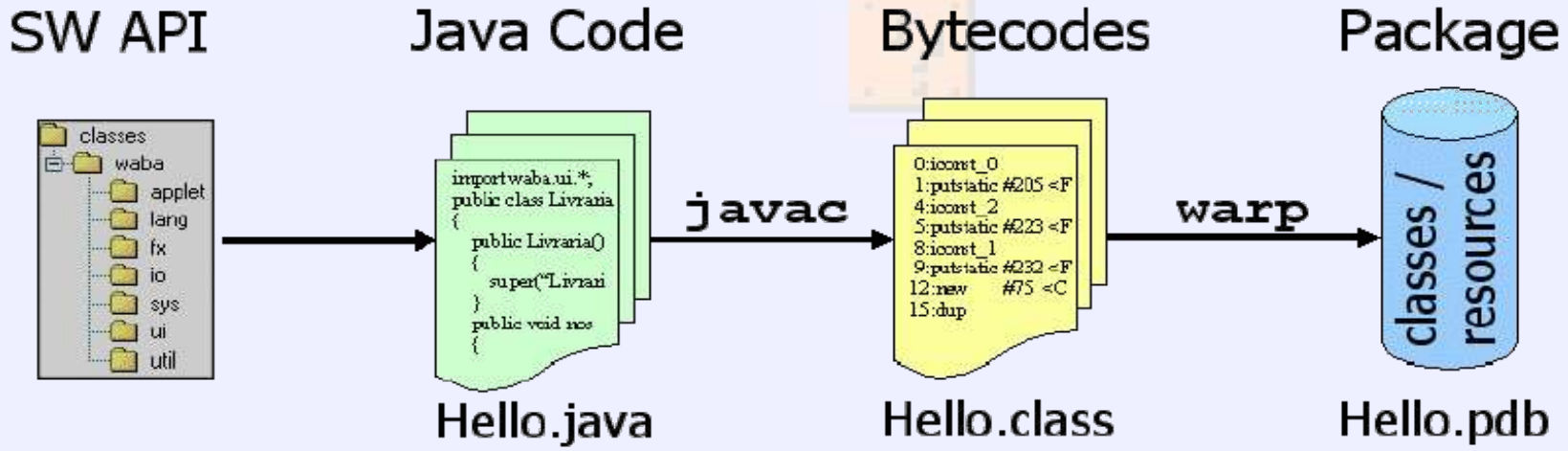
Creando un entorno de trabajo (VI)

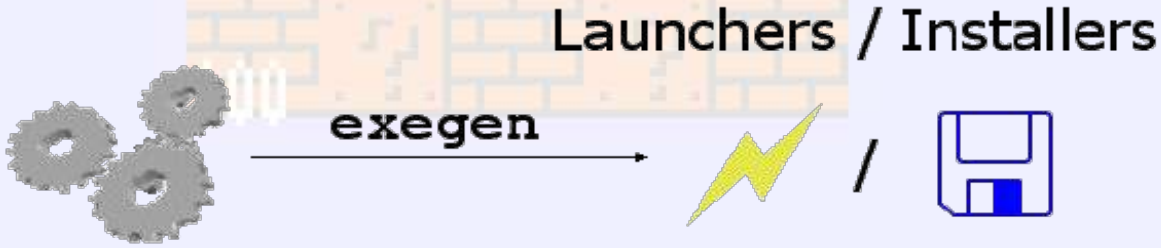
- Compilación de un programa
  - Ejemplo de programa: `bin/src/HelloWorld.java`
    - `gcj -C HelloWorld.java`
    - `javac HelloWorld.java`
  - Ejecución del ejemplo en la herramienta de prueba:
    - `kaffe -classpath $CLASSPATH waba.applet.Applet HelloWorld`
    - `java waba.applet.Applet HelloWorld`

Creando un entorno de trabajo (VII)

- Generando los formatos con `Exegen` y `Warp`
  - Genera los formatos necesarios para ejecutar en cada plataforma.
    - Para palm:
      - `Exegen <NombreFichero> <ClasePrincipal> <Nombreaplicación>`
        - `Exegen HolaMundo HolaMundo Hola`
        - Genera un fichero `.prc`
      - `Warp c <NombreAplicacion> <RecursosNecesarios>`
        - `Warp Hola HolaMundo.class`
        - Genera un fichero `.pdb`
    - Curiosidad: Cambio de los scripts para que funcionen con *Kaffe*

Proceso de compilación





Creando un entorno de trabajo (y VIII)

- Gestionando el proyecto:
  - Makefile (*Tips*)
    - Creación de "*targets*" para Exegen y Warp
    - Creación de "*targets*" para distintas arquitecturas
      - `make winCE, make palm,....`
  - Ant
    - Sistema de compilación en Java
    - Uso de targets "prefabricados"
    - Basado en XML
    - Más complejo que Make pero más específico y potente.

Creando un entorno de trabajo (y IX)

- Un ejemplo de Makefile:

```
JAVA_COMPILER= /usr/bin/gcj
JAVAVM= /usr/bin/kaffe
JAVA_OPTS= -C
EXEGEN=Exegen
WARP=Warp
APPLET=waba.applet.Applet
NAME=HolaMundo
CLASSES=HolaMundo.class

all: compile exegen warp

compile:
        ${JAVA_COMPILER} ${JAVA_OPTS} *.java

exegen:
        ${JAVAVM} ${EXEGEN} ${NAME} ${NAME} ${NAME}

warp:
        ${JAVAVM} ${WARP} c ${NAME} ${CLASSES}

run:
        ${JAVAVM} ${APPLET} ${NAME}
```

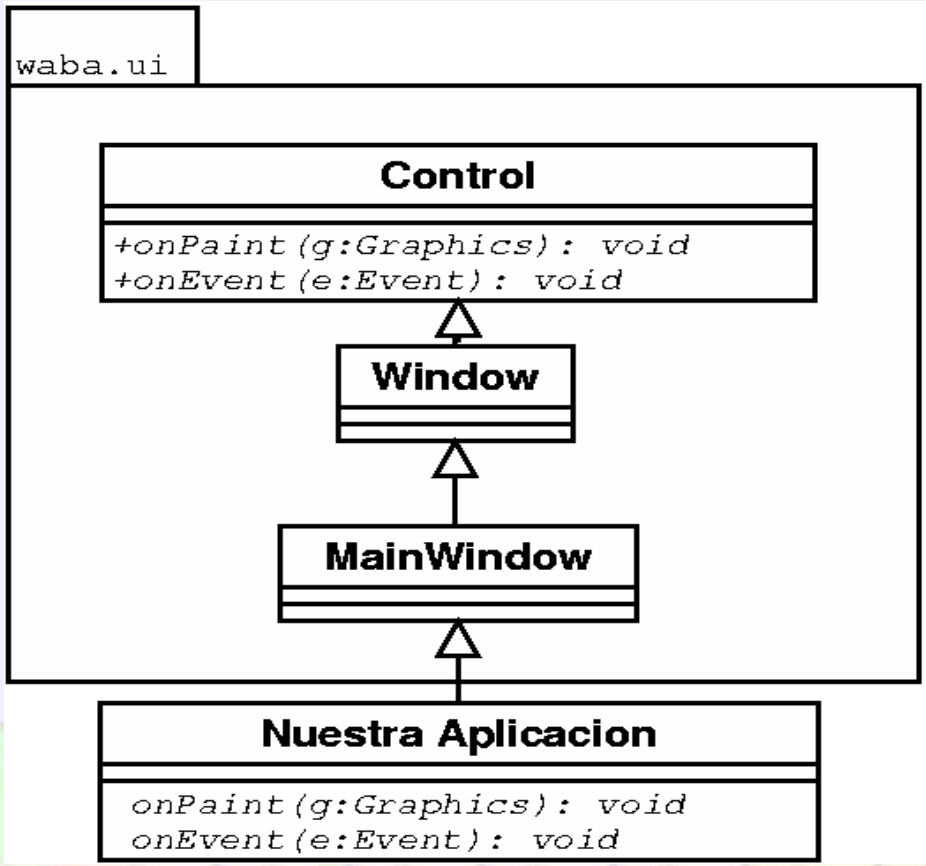
Radiografía de un programa en SuperWaba

- Un poco de código...

```
import waba.ui.*;
import waba.fx.*;

public class HelloWorld extends MainWindow
{
   public HelloWorld()
   {
      super("HelloWorld", NO_BORDER);
   }

   public void onPaint(Graphics g)
   {
      g.drawText("Welcome to SuperWaba", width >>
3, height >> 1);
   }
}
```

Radiografia de un programa en SuperWaba (II)

- Ciclo de vida de una aplicación clásica en SuperWaba



Radiografía de un programa en SuperWaba (y III)

- Métodos a redefinir en la aplicación:
  - `onStart()`
    - Se ejecuta al comenzar la aplicación
    - Ejecuta un `repaint()`
  - `onPaint()`
    - Se ejecuta al realizar un `repaint()`
  - `onEvent()`
    - Es llamado cuando hay un evento (pulsa botones, pinchar con el lapiz...)
    - Se tratan así los eventos de la aplicación:
      - Evento → Reacción

Resumen Parte I

- Superwaba = Clases java libres+Binarios+VM libres
- Proceso de desarrollo=programación+compilación+generación de formatos PDA.
- Elementos del entorno=gcj+Kaffe+Makefile+Exegen+Warp (Todo libre)
- Programación similar a un applet guiada por eventos.

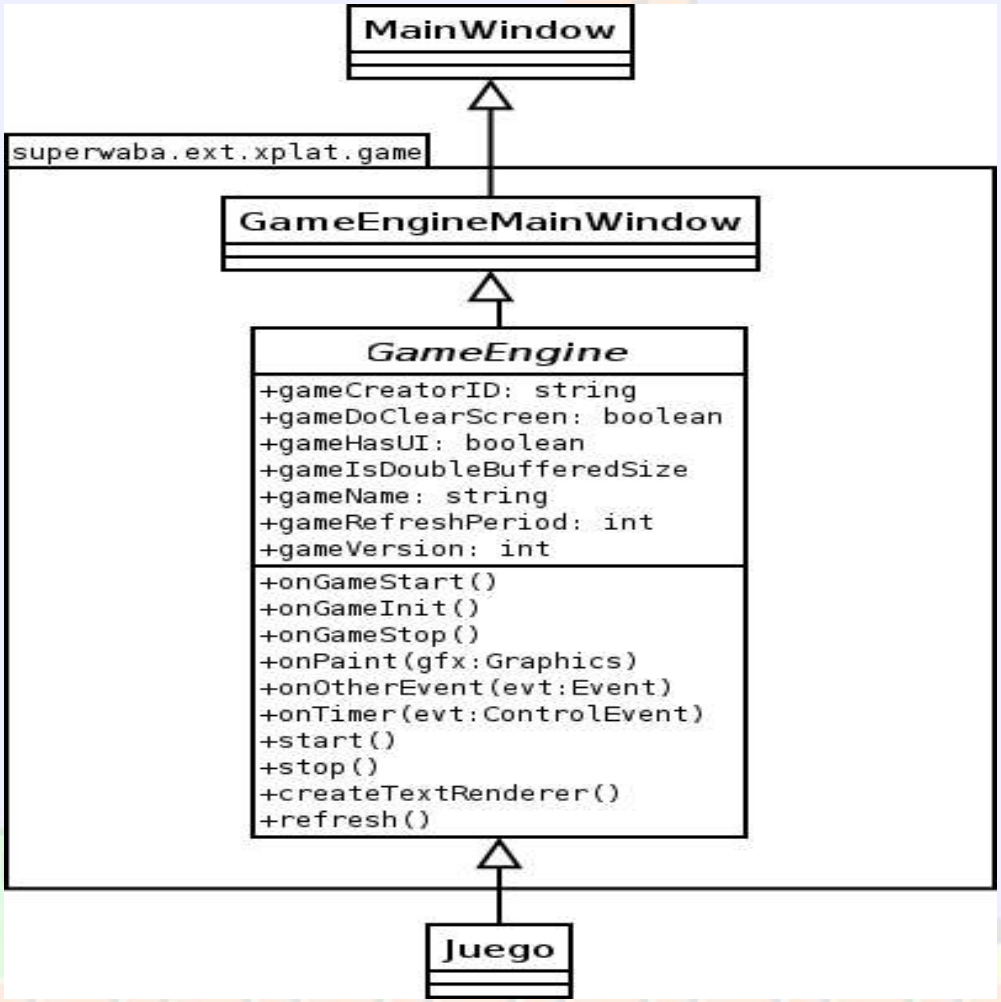
Parte II: Programamando juegos con SuperWaba

Programando Juegos: Introducción (I)

- Inclusión de nuevas clases en nuestro PDA:
  - `lib/vm/<arquitectura>/xplat/XPlatGame.pdb`
  - Otras pueden ser de utilidad (las iremos viendo si hacen falta).
- Se extiende la clase `GameEngine`
  - Esta clase redefine nuevos métodos que llamen a metodos derivados de la clase `MainWindow`
  - Ciertos métodos son llamados cuando ocurren ciertos eventos en el juego.
  - Los eventos pueden también ser provocados por funciones. (Ej.: `refresh()`)

Programando juegos: Introducción (II)

- Diagrama de clases del esqueleto de un juego:



Programando juegos: Introducción (III)

- Atributos de `GameEngine`
  - `gameCreatorID:` Cadena del creador del juego
  - `gameDoClearScreen:` El juego borra la pantalla antes de cada `onPaint()`
  - `gameHasUI:` El juego usa o no interfaz de ventanas (mucho más lento)
  - `gameIsDoubleBufferedSize:` El juego hace doble-buffer acelerando la presentacion y previniendo el parpadeo de pantalla.
  - `gameName:` Nombre del juego
  - `gameRefreshPeriod:` Tiempo en milisegundos en el que se llama periódicamente a `onPaint()`
  - `gameVersion:` Versión del juego (100 = 1.00).

Programando juegos: Introducción (IV)

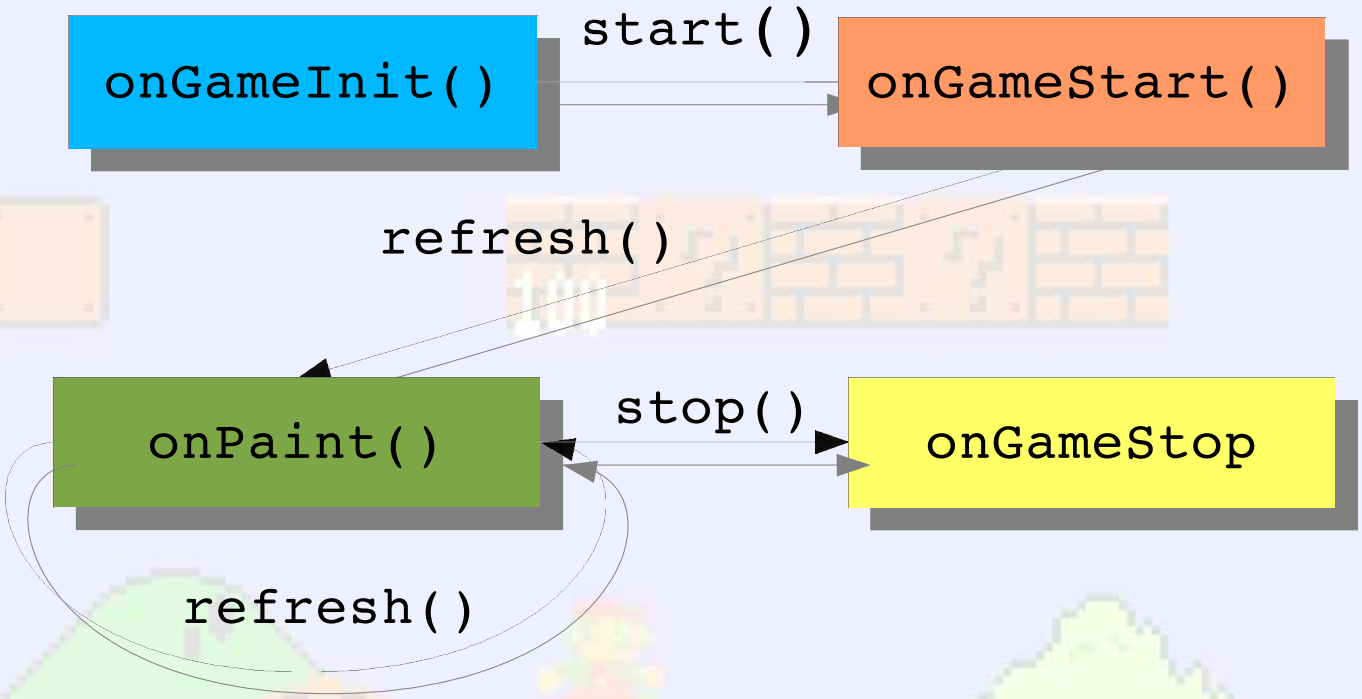
- Funciones llamadas por eventos en el juego a redefinir:
  - `onGameInit`: Llamada al inicio de un juego. Main.
  - `onGameStart`: Llamada por la función `start()`
  - `onGameStop`: Llamada por la función `stop()`
  - `onPaint`: Llamada cuando se hace un refresco de pantalla o explícitamente por la función `refresh()`
  - `onOtherEvent`: Llamada por otros eventos (UI, Teclas...)
  - `onTimer`: Invocada cuando vence un *timer*.

Programando juegos: Introducción (V)

- Funciones que pueden ser llamadas en `GameEngine`
  - `start()`: Inicia el contador de juego y llama a `onGameStart()`.
  - `stop()`: Para el contador del juego y llama a `onGameStop()`.
  - `refresh()`: Llama a la función `onPaint()`
  - `createTextRenderer()`: Crea un objeto de tipo `TextRender`.

Programando juegos: Introducción (VI)

- Ciclo de vida de un juego



Programando juegos: Introducción (VI)

- Ejemplo con `TextRender`

```
import superwaba.ext.xplat.game.*;
import waba.fx.*;
import waba.sys.*;
import waba.ui.*;

public class TextRender extends GameEngine{

    TextRenderer levelRenderer=createTextRenderer
(MainWindow.defaultFont,
new Color(255,255,0), "TEXTO RENDER!:", 1);

    public TextRender(){
        gameName= "TextRender";
        gameCreatorID= "davidfv";
        gameVersion=100;
        gameRefreshPeriod=20000;
        gameIsDoubleBuffered=false;
        gameDoClearScreen=false;
        gameHasUI=false;
```

Programando juegos: Introducción (VII)

```
    public void onGameInit(){
        levelRenderer.display(20,20,1,true);
        start();
      }

      public void onStartGame(){
        refresh();
      }

      public void onPaint(Graphics gfx){
        levelRenderer.display(20,20,2,true);
      }
  }
```

```
createTextRenderer(Font font, Color foreColor,
java.lang.String text, int maxDigits,
boolean zeroPadding)
```

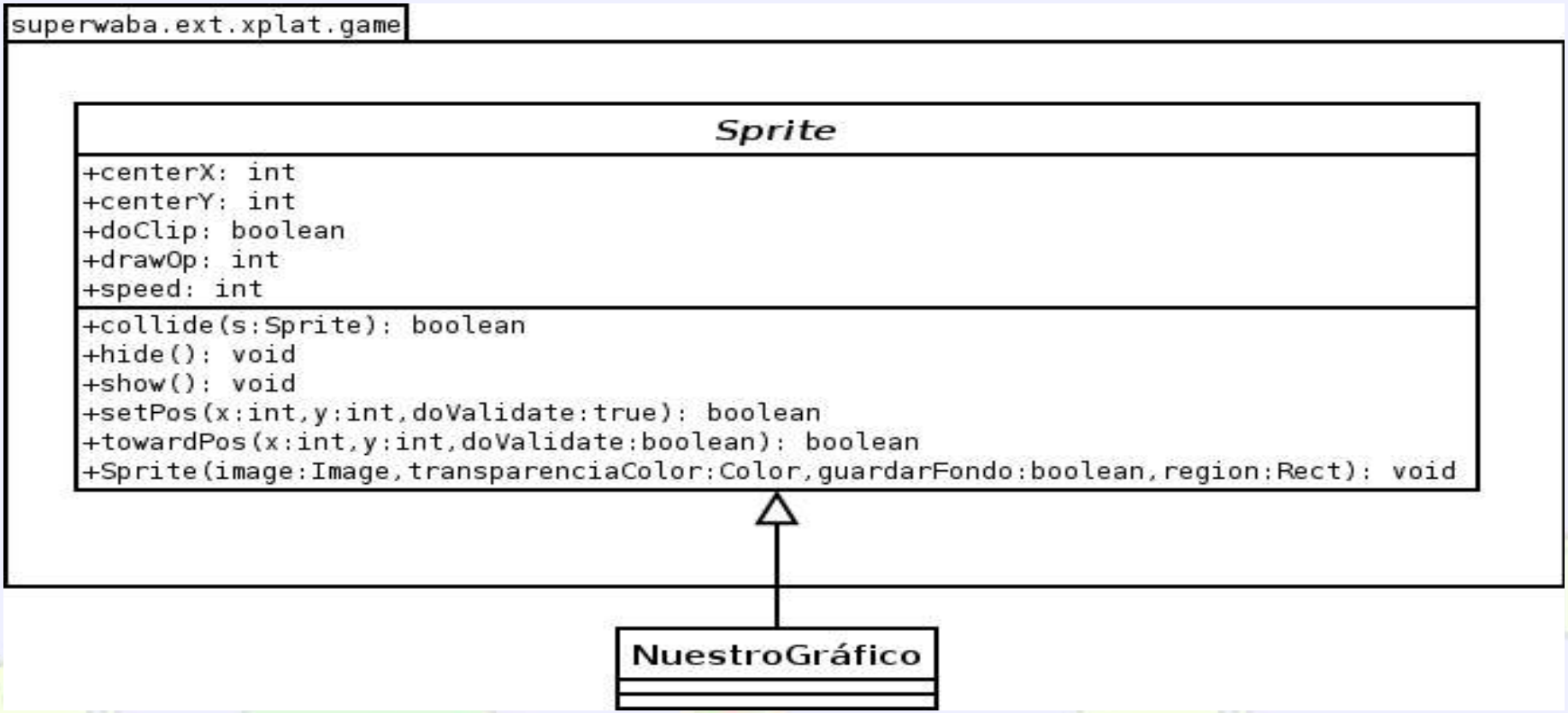
Programando juegos: Sprites y movimiento

Sprite: Figura que interviene en un juego

- Figuras que interactúan en un juego.
- Un *sprite* es un objeto que extiende la clase `Sprite`
- Se moverá, colisionará y podrá estar animada.
- *Sprite* significa hada en inglés ;).

Programando juegos: Sprites y movimiento (II)

- Diagrama de clase de un *sprite*



Programando Juegos: Sprites y Movimiento (III)

- Constructor de un `Sprite`
  - Imagen asociada al *sprite* (BMP)
  - Color de transparencia del *sprite (XOR...)*
  - Conservación del fondo del *sprite*
  - Region válida del *sprite* (null equivale a toda la pantalla)
- Atributos a redefinir del objeto `Sprite`
  - `centerX`: Centro X del sprite (Determina su posición).
  - `centerY`: Centro Y del sprite (Determina su posición).
  - `doClip`: Define si se permite salir al *sprite* de la pantalla
  - `drawOp`: Define como se dibujará el *sprite* en pantalla
    - `Graphics.DRAW_SPRITE, Graphics.DRAW_PAINT`
  - `speed`: Pixels con los que avanza a cada paso el *sprite*

¡CUIDADO CON LAS PATENTES!

- ¿Sabeis que el algoritmo *XOR* está patentado?
- ¿Eso quiere decir que nos pueden impedir usarlo para un juego? ... SI...
- Las patentes de software (de algoritmos en general) son un freno a la innovación y sirven para hacer *tradding* de patentes entre multinacionales, para nada más.
- ¿Sabeis que hay una directiva de la comunidad europea que aprueba **ilegalmente** las patentes de software en Europa?
- Más informacion: proinnova.hispalinux.es o www.gpul.org
- ¡Hay que movilizarse para mostrar nuestro desacuerdo¡

27 Abril- Movilización anti-patentes en la universidad
12:00 horas

Programando juegos: Sprites y movimiento (IV)

- Funciones básicas:
  - `show()`: Muestra el sprite en pantalla (posición dado por su centro).
  - `hide()`: Oculta el sprite de la pantalla.
  - Esto sucederá cuando se realice un refresco de pantalla.
  - Ejemplo: *Sprite1*

Programando juegos: Sprites y movimiento (V)

```
import superwaba.ext.xplat.game.*;
import waba.fx.*;

public class Sprites1 extends GameEngine{

    private Gnomo gnomo;

    public Sprites1(){
        waba.sys.Settings.setPalmOSStyle(true);
        gameName= "Sprites1";
        gameCreatorID= "davidfv";
        gameVersion=100;
        gameHighscoresSize=7;
        gameRefreshPeriod=75;
        gameIsDoubleBuffered=true;
        gameDoClearScreen=true;
        gameHasUI=false;
  }
```

Programando juegos: Sprites y movimiento (VI)

```
  public void onGameInit(){
        gnomo=new Gnomo();
        start();
    }

    public void onGameStart(){
      gnomo.show();
    }

    public void onPaint(Graphics Gfx){
      gnomo.show();
    }
}
```

Programando juegos: Sprites y movimiento (VII)

- Definicion del objeto Gnomo que extiende Sprite

```
import superwaba.ext.xplat.game.*;
import waba.fx.*;

public class Gnomo extends Sprite{

    public Gnomo(){
         super(new Image("gnomo.bmp"),
Color.WHITE,true,null);

        this.drawOp = Graphics.DRAW_SPRITE;
        this.speed=2;
        this.centerX=100;
        this.centerY=100;
    }
}
```

Programando juegos: Sprites y movimiento (VIII)

- Funciones relacionadas con el movimiento del *sprite*:
  - `setPos(x,y,doValidate)`: Sitúa el *sprite* en una posición validando o no si la región es correcta.
  - `towardPos(x,y,doValidate)`: Mueve el *sprite* con su velocidad hacia *x,y* usando el algoritmo de línea de *Freshman*.
  - Después de cada movimiento hemos de refrescar la pantalla.
  - Ejemplo de movimiento de sprite: *Sprites2mov*

Programando juegos: Sprites y movimiento (IX)

```
public class Sprites2 extends GameEngine{

    private Gnomo gnomo;
    private Gnomo gnomo2;

    int i=0;

    public Sprites2(){
        waba.sys.Settings.setPalmOSStyle(true);

        gameName= "Sprites2";
        gameCreatorID= "davidfv";
        gameVersion=100;
        gameHighscoresSize=7;
        gameRefreshPeriod=75;
        gameIsDoubleBuffered=true;
        gameDoClearScreen=true;
        gameHasUI=false;
    }
```

Programando juegos: Sprites y movimiento (X)

```
public void onGameInit(){
    gnomo=new Gnomo();
    gnomo2=new Gnomo();
    start();
}

public void onGameStart(){
  gnomo.show();
  gnomo2.show();
}

public void onPaint(Graphics Gfx){
  i++;
  gnomo.towardPos(100-(i*2),100-(i*2),false);
  gnomo.show();
  gnomo2.towardPos(100+(i*2),100+(i*2),false);
  gnomo2.show();

}
```

Programando juegos: Sprites y movimiento (XI)

- Detección de teclas
  - No usar `onKey`, es muy lento.
  - Se anuncia a la VM que se quieren interceptar las pulsaciones y se chequea despues su pulsación.
  - Ejemplo: *sprites3keys*

```
waba.sys.Vm.interceptSystemKeys (waba.ui.IKeys.PAGE_UP |
waba.ui.IKeys.PAGE_DOWN);
int sk=Vm.getSystemKeysPressed();
      if (sk!=0){
        if((sk & Vm.SK_PAGE_UP) != 0){
          gnomo.jump();
        }
      }
```

Programando juegos: Sprites y movimiento (XII)

```
public class Sprites3 extends GameEngine{

    private Gnomo gnomo;
    private Gnomo gnomo2;

    int i=0;

    public Sprites3(){
        waba.sys.Settings.setPalmOSStyle(true);

        gameName= "Sprites2";
        gameCreatorID= "davidfv";
        gameVersion=100;
        gameHighscoresSize=7;
        gameRefreshPeriod=75;
        gameIsDoubleBuffered=true;
        gameDoClearScreen=true;
        gameHasUI=false;

        waba.sys.Vm.interceptSystemKeys(waba.ui.IKeys.PAGE_UP |
waba.ui.IKeys.PAGE_DOWN);
    }
```

Programando juegos: Sprites y movimiento (XIII)

```
public void onGameInit(){
    gnomo=new Gnomo();
    gnomo2=new Gnomo();
    start();
}
public void onGameStart(){
  gnomo.show();
  gnomo2.show();
}
public void onPaint(Graphics Gfx){
  int sk=Vm.getSystemKeysPressed();
  if (sk!=0){
    if((sk & Vm.SK_PAGE_UP) != 0){
      gnomo.jump();
    }

    if((sk & Vm.SK_PAGE_DOWN) != 0){
      gnomo.down();
    }
  }
  gnomo.show();
}
}
```

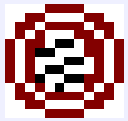
Programando juegos: Sprites y movimiento (XIV)

- Sprite Gnomo

```
public class Gnomo extends Sprite{

    public Gnomo(){
        super(new Image("gnomo.bmp"),
Color.WHITE,true,null);

        this.drawOp = Graphics.DRAW_PAINT;
        this.speed=1;
        this.centerX=20;
        this.centerY=100;
    }

    void jump(){
      if(centerY>40)
        setPos(centerX,centerY-5,false);
    }
    void down(){
      if(centerY<150)
      setPos(centerX,centerY+5,false);
    }
```

Programando juegos: Detección de colisiones

- Aspecto básico del desarrollo de algunos juegos.
- Se detecta cuando las regiones de cada *sprite* se solapan.
- Ideal para: Choques con balas, encontronazos con enemigos, etc... ;)
- `collide (s Sprite)`: Función de cada *sprite* que indica si se ha chocado con otro *sprite* s.
- Ejemplo: *Sprites4collide* (*El leprechaun y el barril...*)



Programando juegos: Detección de colisiones (II)

```
public class Sprites4 extends GameEngine{

    private Gnomo gnomo;
    private Barril barril;
    private int vidas=3;

    TextRenderer vidasText=createTextRenderer(getFont(), Color.BLACK,
"Vidas:", 1);

    public Sprites4(){
        waba.sys.Settings.setPalmOSStyle(true);

        gameName= "Sprites2";
        gameCreatorID= "davidfv";
        gameVersion=100;
        gameHighscoresSize=7;
        gameRefreshPeriod=75;
        gameIsDoubleBuffered=true;
        gameDoClearScreen=true;
        gameHasUI=false;

        waba.sys.Vm.interceptSystemKeys(waba.ui.IKeys.PAGE_UP |
waba.ui.IKeys.PAGE_DOWN);
```

Programando juegos: Detección de colisiones (III)

```
public void onGameInit(){
        gnomo=new Gnomo();
        barril=new Barril();
        vidasText.display(10,10,vidas,true);
        start();
    }

    public void onGameStart(){
      gnomo.show();
      barril.show();
    }
```

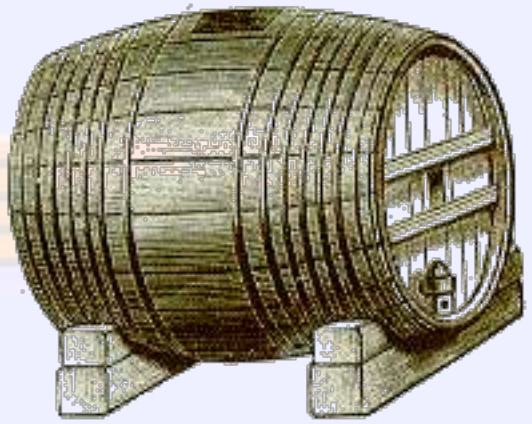
Programando juegos: Detección de colisiones (IV)

```
public void onPaint(Graphics Gfx){
    int sk=Vm.getSystemKeysPressed();
    if (sk!=0){
      if((sk & Vm.SK_PAGE_UP) != 0){
        gnomo.jump();
      }
      if((sk & Vm.SK_PAGE_DOWN) != 0){
        gnomo.down();
      }
    }
    barril.move();
    gnomo.show();
    barril.show();

    if(gnomo.collide(barril)){
      if(vidas==0){
        stop();
      } else{
        vidas--;
        barril.start();
      }
    }
    vidasText.display(10,10,vidas,true);
 }
```

Programando juegos: Detección de colisiones (y V)

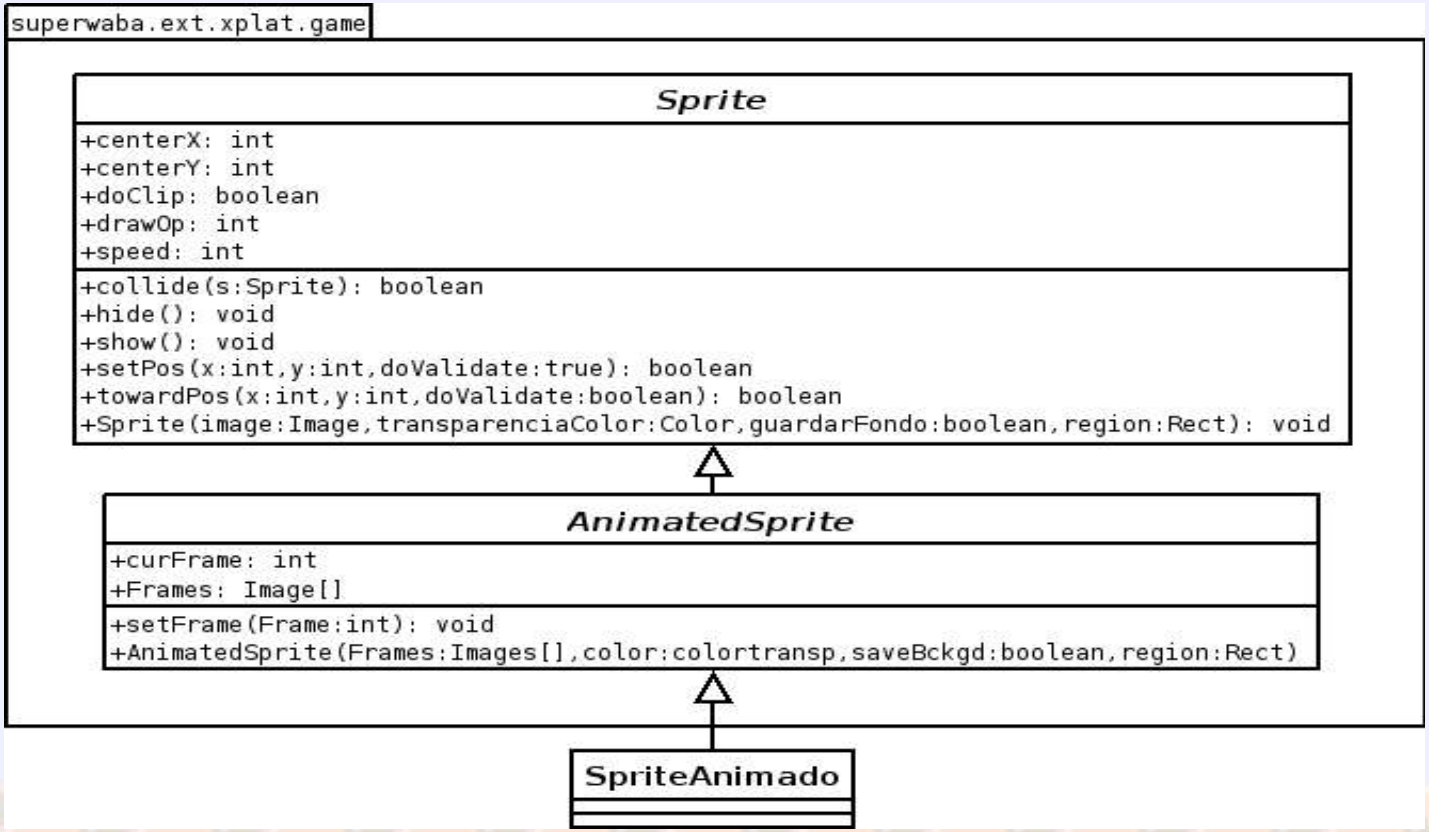
- Sprite `Barril`

```
public class Barril extends Sprite{

    public Barril(){
        super(new Image("barril.bmp"),Color.WHITE,true,null);

        this.drawOp = Graphics.DRAW_PAINT;
        this.speed=5;
        this.centerX=100;
        this.centerY=100;
    }

    void move(){
      if(centerX<1){
        centerX=100;
      }else{
        centerX--;
      }
    }

    void start(){
      centerX=100;
    }
```



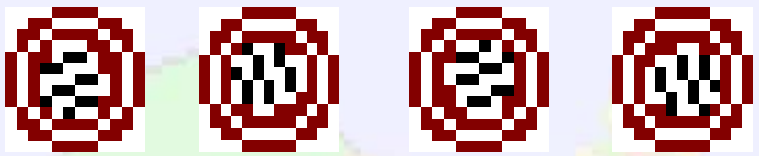
Programando juegos: Sprites animados

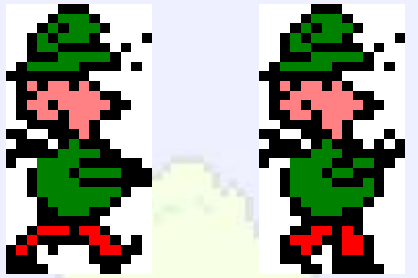
- *Sprites* que presentan movimiento (distintas imágenes conforman un mismo sprite).
- Clase `AnimatedSprite`. Diagrama de clases.



Programando juegos: Sprites animados (II)

- Cada *sprite* tiene una serie de dibujos.
- Somos los encargados de hacer cuadrar la animación
- `setFrame(int i)`: Fija el frame que deseamos que se vea en cada momento.
- Ejemplo: *sprites5anim*





Programando juegos: Sprites animados (III)

- Inicialización del *sprite* animado en la clase principal

```
public void onGameInit(){
        Image frames[] = new Image[4];

        frames[0]=new Image("barril.bmp");
        frames[1]=new Image("barril1.bmp");
        frames[2]=new Image("barril2.bmp");
        frames[3]=new Image("barril3.bmp");

        Image frames2[] = new Image[2];

        frames2[0]=new Image("gnomo.bmp");
        frames2[1]=new Image("gnomo2.bmp");

        barril=new Barril(frames);
        gnomo=new Gnomo(frames2);
        vidasText.display(10,10,vidas,false);
        start();
    }
```

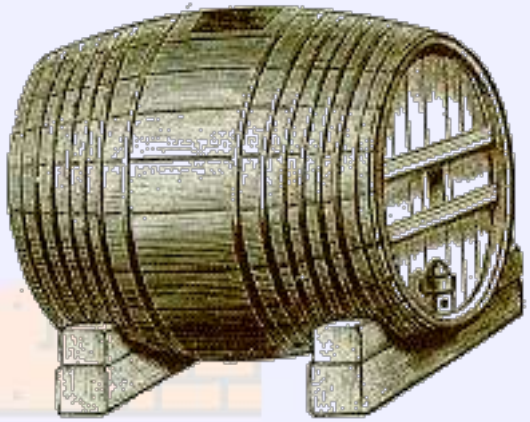
Programando juegos: Sprites animados (IV)

- *Sprite* animado del barril.

```
public class Barril extends AnimatedSprite{

    private int framesnum=0;

    public Barril(Image frames[]){

        super(frames,Color.WHITE,true,null);

        this.drawOp = Graphics.DRAW_PAINT;
        this.speed=5;
        this.centerX=100;
        this.centerY=100;
    }
```

Programando juegos: Sprites animados (V)

```
void move(){
   if(centerX<1){
     centerX=100;
   }else{
     centerX--;
   }

   framesnum++;
   if(framesnum>3){
     framesnum=0;
     }
   setFrame(framesnum);

 }

void start(){
   centerX=100;
 }
```



Cosas que se han quedado en el tintero...

- HighScores
- Animations
- Options
- Timers

Para la segunda parte: ***Programación Avanzada de juegos en PDAs*** ;)

Referencias

- Game Tutorial:
  - docs/GameTutorial.pdf
- Tutorial general de SuperWaba
  - docs/SuperWaba Companion.pdf