Programación de juegos para principiantes

Veremos conceptos intermedios sobre java (threads, AWT, Swing, etc.) y conceptos básicos para la programación de juegos (game loop, FPS, sprite, etc). Mi idea es que estos tutoriales sirvan tanto para los que desean programar juegos, como para que la gente con un nivel de principiante o intermedio en java pueda aprender y afianzar conceptos de programación java en general, de forma divertida.

Juegos en Java

Hoy día hay grandes juegos realizados completamente en java. La industria de los juegos para móviles está creciendo y java es el lenguaje para programar en Android. Android debe ser ya el sistema operativo para smart phones más usado en este momento. Por otro lado, juegos como Minecraft, tienen millones de usuarios a pesar de iniciarse como proyecto de un sólo programador, sin el apoyo de una gran empresa.

El juego: Mini Tenis

Desarrollaremos desde cero una versión de uno de los primeros juegos que alcanzaron popularidad en la era de la informática.

Índice

Programación de juegos:

JFrame, JPanel, método paint

Para dibujar algo necesitamos una superficie donde pintar. Esta superficie o lienzo (Canvas en inglés) donde pintaremos nuestro primer ejemplo es un objeto JPanel.

Así como un lienzo necesita un marco para sostenerse, nuestro JPanel estará enmarcado en una ventana modelada por la clase JFrame.

JFrame: La Ventana

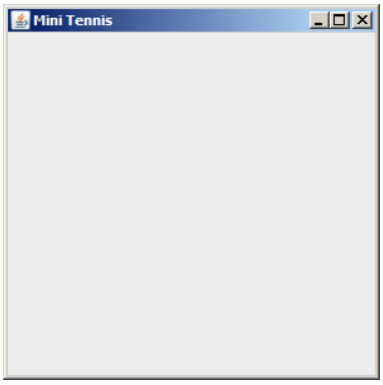
El siguiente código crea una ventana con titulo "Mini Tennis" de 300 pixels por 300 pixels.

La ventana no será visible hasta que llamemos setVisible(true).

Si no incluimos la última línea "frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)", cuando cerremos la ventana el programa no terminará y seguirá ejecutándose.

```
package com.edu4java.minitennis1;
import javax.swing.JFrame;
public class Game {
	public static void main(String[] args) {
		JFrame frame = new JFrame("Mini Tennis");
		frame.setSize(300, 300);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Si ejecutamos obtendremos:



Con estas pocas instrucciones obtenemos una ventana que se puede maximizar, minimizar, cambiar de tamaño con el ratón, etc.

En realidad cuando creamos un objeto JFrame iniciamos un motor que maneja la interfaz de usuario.

Este motor se comunica con el sistema operativo tanto para pintar en la pantalla como para recibir información del teclado o el ratón.

Llamaremos a este motor "Motor AWT" o "Motor Swing" ya que está compuesto por estas dos librerías.

En las primeras versiones de java solo existía AWT y luego se agregó Swing.

Este Motor utiliza varios hilos de ejecución.

¿Qué es un hilo o thread en java?

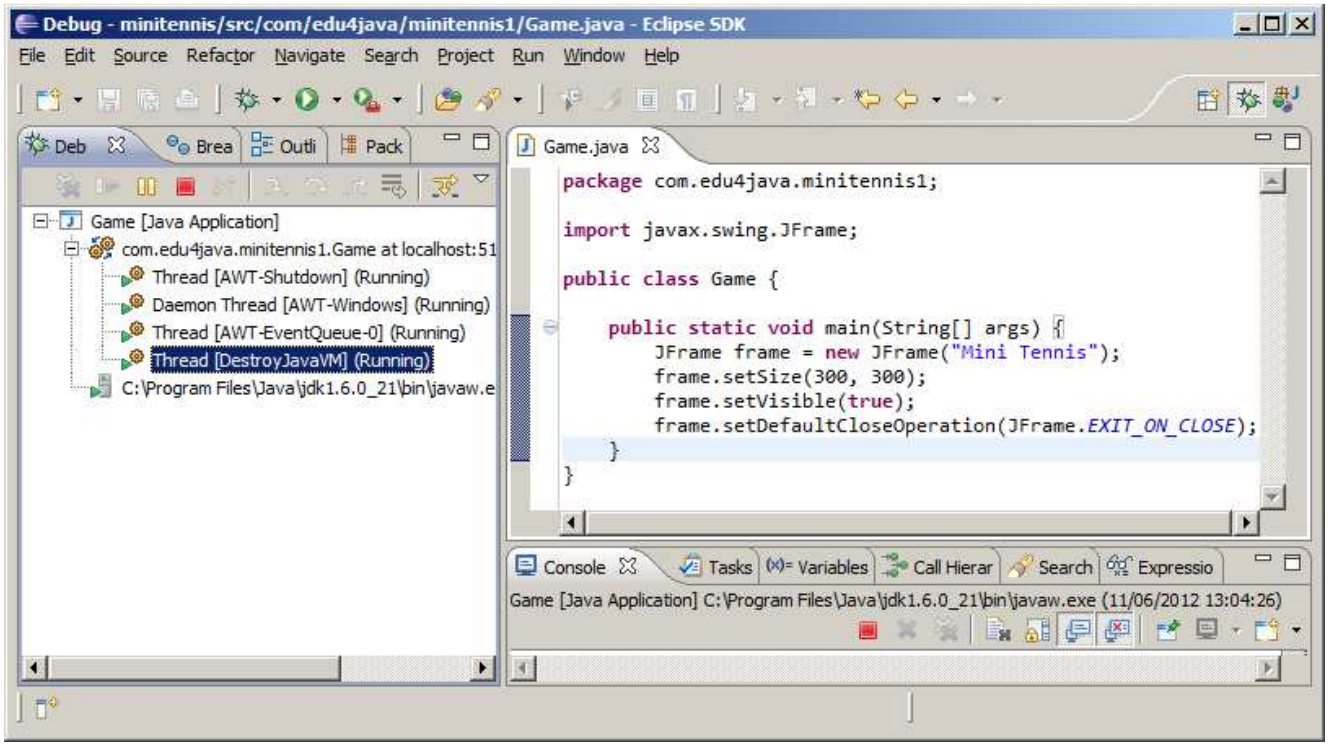
Normalmente un programa se ejecuta línea tras línea por un solo procesador en una sola línea o hilo de ejecución. El concepto de hilo (en inglés Thread) permite a un programa iniciar varias ejecuciones concurrentes. Esto es como si existieran varios procesadores ejecutando al mismo tiempo sus propias secuencias de instrucciones.

Aunque los hilos y la concurrencia son herramientas muy potentes puede traer muchos problemas como que dos hilos accedan a las mismas variables de forma conflictiva. Es interesante considerar que dos hilos pueden estar ejecutando el mismo código de un método a la vez.

Podemos pensar que un hilo es un cocinero preparando un plato leyendo una receta de cocina. Dos hilos concurrentes serían como dos cocineros trabajando en la misma cocina, preparando cada uno un plato leyendo cada uno una receta o también podrían estar leyendo la misma receta. Los conflictos surgen por ejemplo cuando los dos intentan usar una sartén al mismo tiempo.

Motor AWT e Hilo de cola de eventos - Thread AWT-EventQueue-0

El Motor AWT inicia varios Hilos (Threads) que podemos ver en la vista Debug si iniciamos la aplicación con debug y vamos a la perspectiva Debug. Cada hilo es como si fuera un programa independiente ejecutándose al mismo tiempo que los otros hilos. Más adelante veremos más sobre hilos, por lo pronto solo me interesa que recuerden el tercer hilo que vemos en la vista Debug llamado "Thread [AWT-EventQueue-0]" este hilo es el encargado de pintar la pantalla y recibir los eventos del teclado y el ratón.



JPanel: El lienzo (Canvas en inglés)

Para poder pintar necesitamos donde y el donde es un objeto JPanel que incluiremos en la ventana.

Extenderemos la clase JPanel para poder sobrescribir el método paint que es el método que llamará el Motor AWT para pintar lo que aparece en la pantalla.

```
package com.edu4java.minitennis1;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.Ellipse2D;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game2 extends JPanel {

	@Override
	public void paint(Graphics g) {
		Graphics2D g2d = (Graphics2D) g;
		g2d.setColor(Color.RED);
		g2d.fillOval(0, 0, 30, 30);
		g2d.drawOval(0, 50, 30, 30);
		g2d.fillRect(50, 0, 30, 30);
		g2d.drawRect(50, 50, 30, 30);
		g2d.draw(new Ellipse2D.Double(0, 100, 30, 30));
	}

	public static void main(String[] args) {
		JFrame frame = new JFrame("Mini Tennis");
		frame.add(new Game2());
		frame.setSize(300, 300);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

El método paint recibe por parámetro un objeto Graphics2D que extiende de Graphics. Graphics es la vieja clase usada por AWT que ha sido reemplazada por Graphics2D que tiene más y mejor funcionalidad. El parámetro sigue siendo de tipo Graphics por compatibilidad pero nosotros siempre utilizaremos Graphics2D por lo que es necesario crear una variable g2d:

"Graphics2D g2d = (Graphics2D) g;"

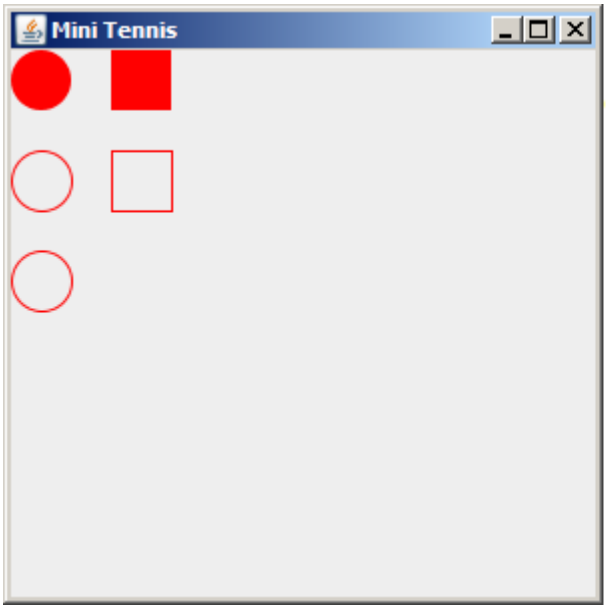
Una vez que tenemos g2d podemos utilizar todos los métodos de Graphics2D para dibujar.

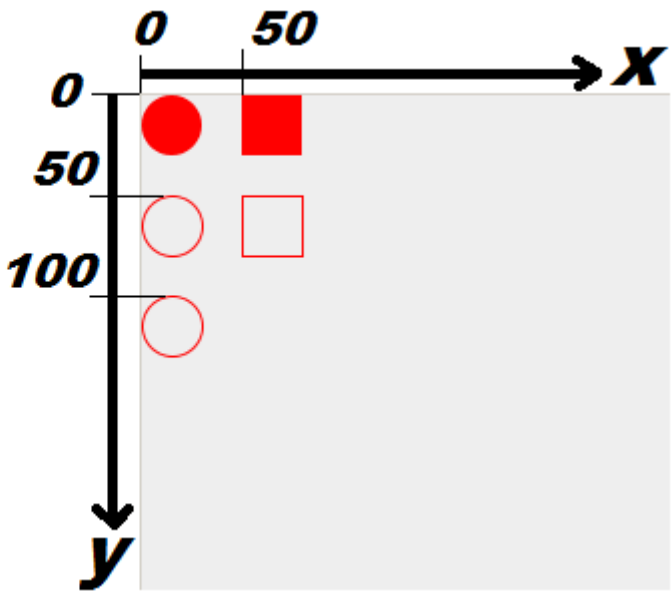
Lo primero que hacemos es elegir el color que utilizamos para dibujar: "g2d.setColor(Color.RED);".

Luego dibujamos unos círculos y cuadrados.

Posicionamiento en el lienzo. Coordenadas x e y

Para dibujar algo dentro del lienzo debemos indicar en qué posición comenzaremos a pintar. Para esto cada punto del lienzo tiene una posición (x,y) asociada siendo (0,0) el punto de la esquina superior izquierda.





El primer círculo rojo se pinta con "**g2d.fillOval(0, 0, 30, 30)**": los primeros dos parámetros son la posición (x,y) y luego se indica el ancho y alto. como resultado tenemos un circulo de 30 pixeles de diámetro en la posición (0,0).

El circulo vacío se pinta con "**g2d.drawOval(0, 50, 30, 30)**": el la posición x=0 (pegado al margen izquierdo) y la posición y=50 (50 pixeles más abajo del margen superior) pinta un circulo de 30 pixeles de alto y 30 de ancho.

Los rectángulos se pintan con "**g2d.fillRect(50, 0, 30, 30)**" y "**g2d.drawRect(50, 50, 30, 30)**" de forma similar a los círculos.

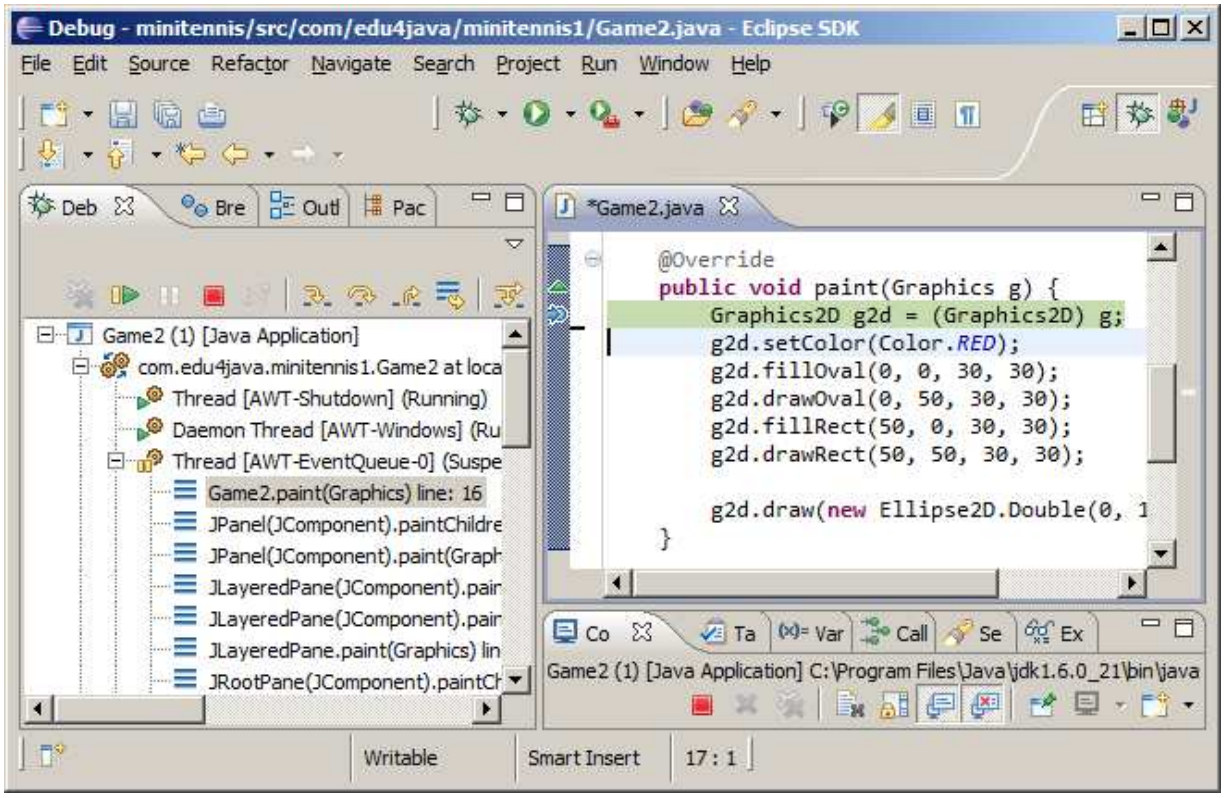
Por último "**g2d.draw(new Ellipse2D.Double(0, 100, 30, 30))**" pinta el ultimo circulo usando un objeto Ellipse2D.Double.

Existen muchísimos métodos en Graphics2D. Algunos los veremos en siguientes tutoriales.

¿Cuándo el motor AWT llama al método paint?

El motor AWT llama al método paint cada vez que el sistema operativo le informa que es necesario pintar el lienzo. Cuando se carga por primera vez la ventana se llama a paint, si minimizamos y luego recuperamos la ventana se llama a paint, si modificamos el tamaño de la ventana con el ratón se llama a paint.

Podemos comprobar este comportamiento si ponemos un breakpoint en la primera línea del método paint y ejecutamos en modo debug.



Es interesante ver que el método paint es ejecutado por el Hilo de cola de eventos (Thread AWT-EventQueue) que como indicamos antes es el encargado de pintar la pantalla.

Game loop y Animación de un objeto

En este tutorial veremos cómo hacer que un círculo se mueva sobre nuestro lienzo. Esta animación se consigue pintando el círculo en una posición y luego borrando y pintando el círculo en una posición cercana. El efecto logrado es un círculo en movimiento.

Posición del círculo

Como mencionamos antes cada vez que pintamos debemos definir la posición (x,y) donde dibujaremos en este caso el círculo. Para que el círculo se mueva debemos modificar la posición (x,y) cada cierto tiempo y volver a pintar el círculo en la nueva posición.

En nuestro ejemplo mantendremos en dos propiedades llamadas "x" e "y", la posición actual de nuestro círculo. También creamos un método moveBall() que incrementará en 1 tanto a "x" como a "y" cada vez que es llamado. En el método paint dibujamos un circulo de 30 pixeles de diámetro en la posición (x,y) dada por las propiedades antes mencionadas "g2d.fillOval(x, y, 30, 30);".

Game loop

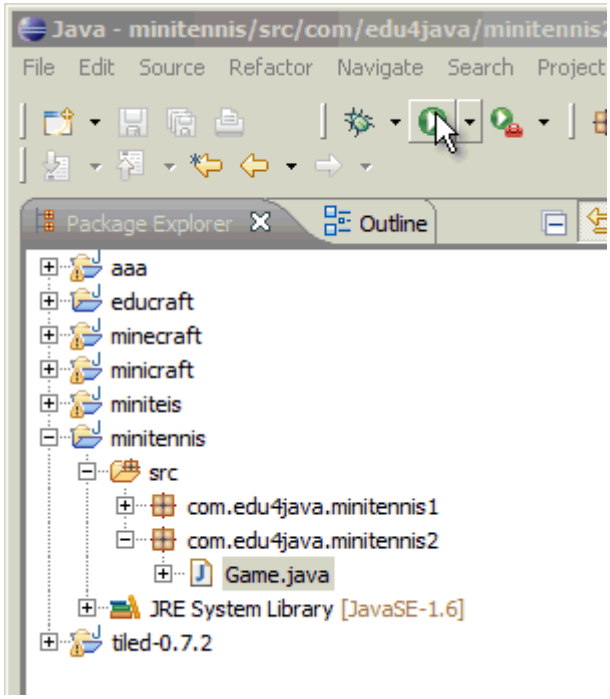
Al final del método main iniciamos un ciclo infinito "while (true)" donde repetidamente llamamos a moveBall() para cambiar la posición del circulo y luego llamamos a repaint() que fuerza al motor AWT a llamar al método paint para repintar el lienzo.

Este ciclo o repetición se conoce como "Game loop" y se caracteriza por realizar dos operaciones:

1. **Actualización (Update):** actualización de la física de nuestro mundo. En nuestro caso nuestra actualización está dada tan solo por el método moveBall() que incrementa las propiedades "x" e "y" en 1.
2. **Renderizado (Render):** aquí se dibuja según el estado actual de nuestro mundo reflejando los cambios realizados en el paso anterior. En nuestro ejemplo este renderizado esta dado por la llamada a repaint() y la subsecuente llamada a paint realizada por el motor AWT o más específicamente por el Hilo de cola de eventos.

```
package com.edu4java.minitennis2;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	int x = 0;
	int y = 0;

	private void moveBall() {
		x = x + 1;
		y = y + 1;
	}

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);
		g2d.fillOval(x, y, 30, 30);
	}

	public static void main(String[] args) throws InterruptedException {
		JFrame frame = new JFrame("Mini Tennis");
		Game game = new Game();
		frame.add(game);
		frame.setSize(300, 400);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		while (true) {
			game.moveBall();
			game.repaint();
			Thread.sleep(10);
		}
	}
}
```

Al ejecutar el código anterior obtendremos:

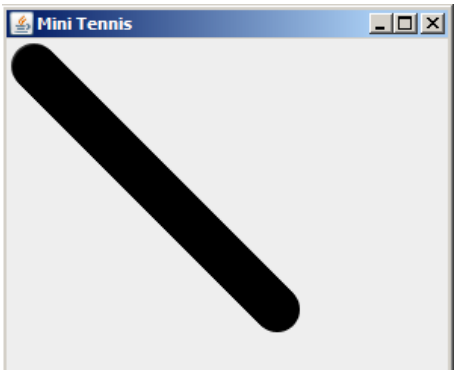


Analizando nuestro método paint

Como mencionamos en el tutorial anterior este método se ejecuta cada vez que el sistema operativo le indica a Motor AWT que es necesario pintar el lienzo. Si ejecutamos el método repaint() de un objeto JPanel lo que estamos haciendo es decirle al Motor AWT que ejecute el método paint tan pronto como pueda. La llamada a paint la realizará el Hilo de cola de eventos. Llamando a repaint() logramos que se repinte el lienzo y así poder reflejar el cambio en la posición del circulo.

```
@Override
public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.fillOval(x, y, 30, 30);
}
```

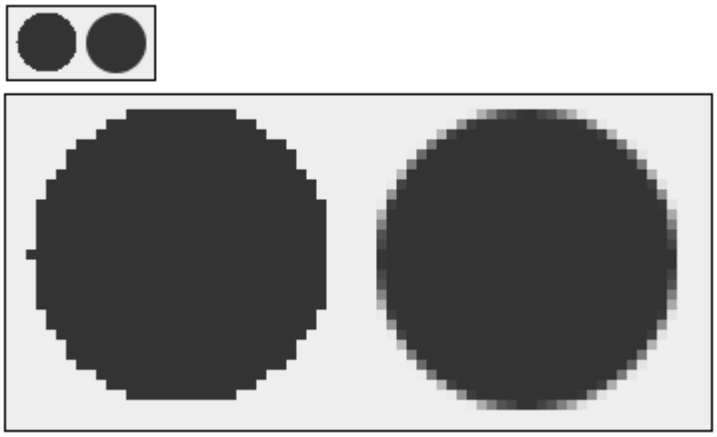
La llamada a *"super.paint(g)"* limpia la pantalla, si comentamos esta línea podemos ver el siguiente efecto:



La instrucción:

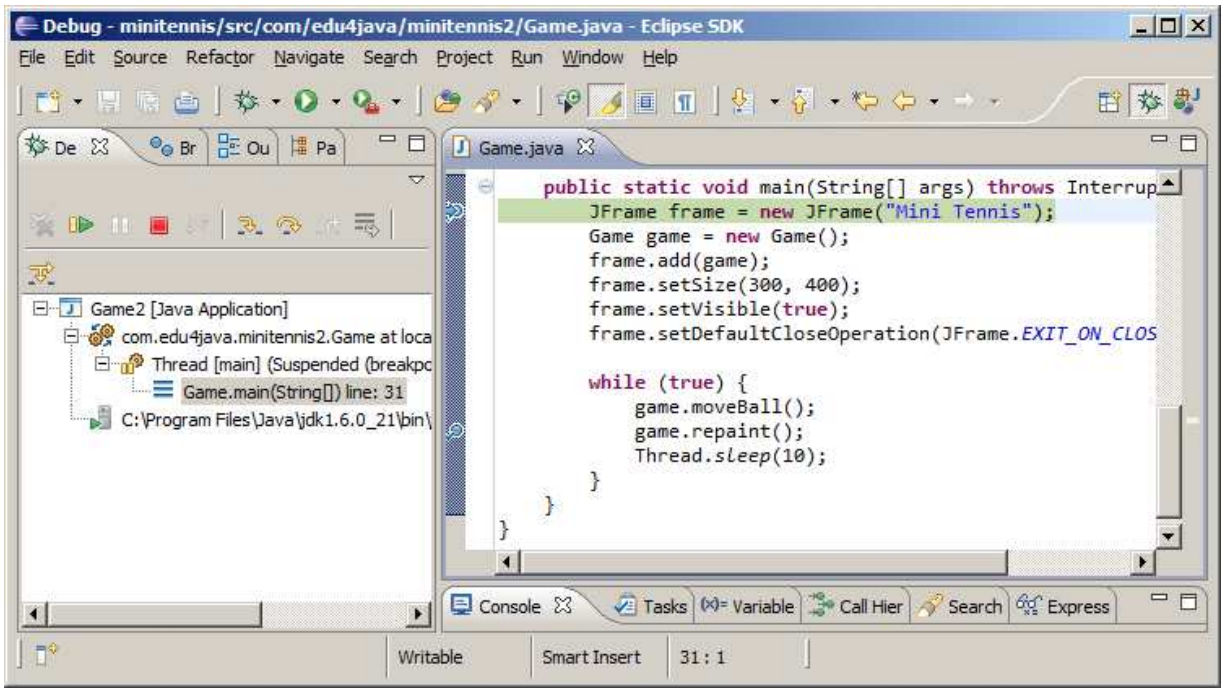
"g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON)"

Suaviza los bordes de las figuras como se puede ver en el siguiente gráfico. El círculo de la izquierda es sin aplicar ANTIALIAS y el de la derecha aplicando ANTIALIAS.

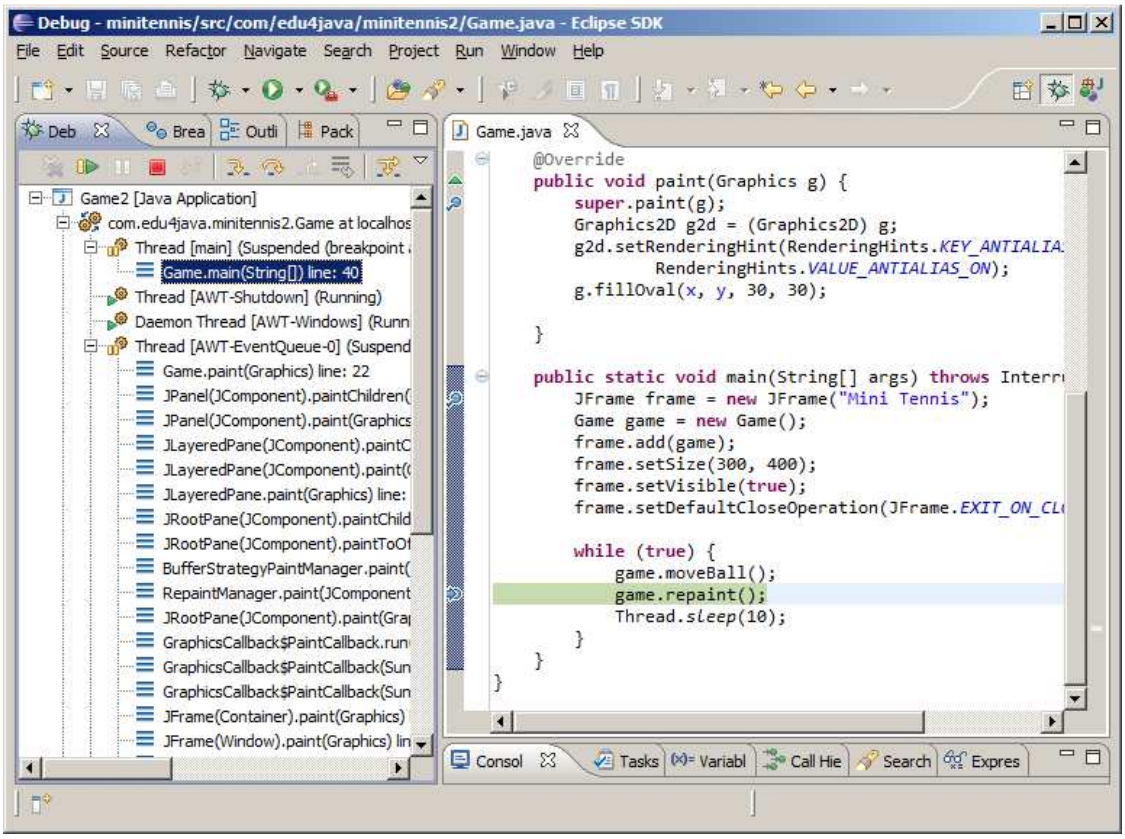


Analizando la concurrencia y el comportamiento de los hilos

Cuando se inicia la ejecución del método main sólo existe un hilo en ejecución. Esto se puede ver colocando un breakpoint en la primera línea del método main.

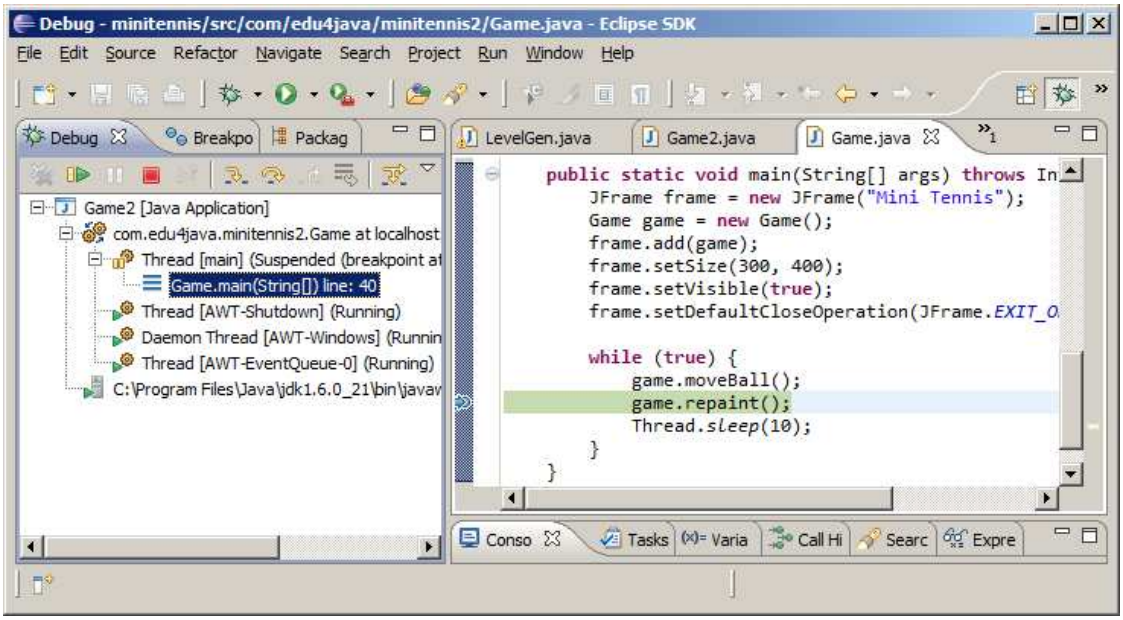


Si agregamos un breakpoint en la línea game.repaint() y en la primera línea del método paint y a continuación oprimimos F8 (Resume: ordena que continúe la ejecución hasta el final o hasta que encuentre el próximo breakpoint) obtendremos:



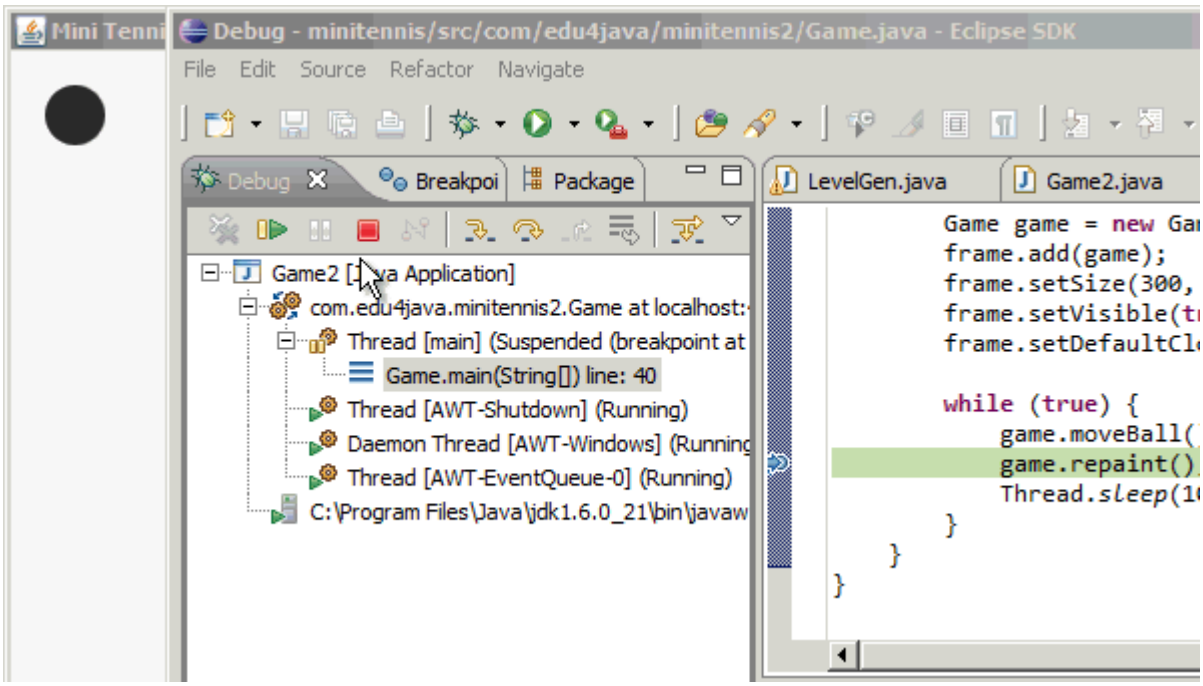
En la vista de la izquierda podemos ver que se han creado cuatro hilos de los cuales dos están detenidos en breakpoints. El Thread main está detenido en la línea 40 en la instrucción game.repaint(). El thread AWT-EventQueue está detenido en el método paint en la línea 22.

Si seleccionamos el thread AWT-EventQueue en la vista Debug y oprimimos F8 repetidamente (2 veces) veremos que no se detiene más en el metodo paint. Esto es porque el sistema operativo no ve motivo para solicitar un repintado del lienzo una vez inicializado.



Si oprimimos F6 (avanza la ejecución del hilo sólo una línea), esta vez sobre el thread main, veremos que el método paint es vuelto a llamar por el thread AWT-EventQueue. Ahora sacamos el breakpoint del método paint, oprimimos F8 y volvemos a tener sólo detenido el thread main.

La siguiente animación nos muestra que pasa en el lienzo cada vez que oprimimos resume (F8) repetidamente. Cada llamada a moveBall() incrementa la posición (x,y) del círculo y la llamada a repaint() le dice al thread AWT-EventQueue que repinte el lienzo.



Por último analicemos la línea "**Thread.sleep(10)**" (la última instrucción dentro del "Game loop"). Para esto comentamos la línea con // y ejecutamos sin debug. El resultado es que no se pinta el círculo en el lienzo. ¿Por qué pasa esto? Esto es debido a que el thread main se apodera del procesador y no lo comparte con el thread AWT-EventQueue que entonces no puede llamar al método paint.

"**Thread.sleep(10)**" le dice al procesador que el thread que se está ejecutando descanse por 10 milisegundos lo que permite que el procesador ejecute otros threads y en particular el thread AWT-EventQueue que llama al método paint.

Me gustaría aclarar que en este ejemplo la solución planteada es muy pobre y sólo pretende ilustrar los conceptos de "game loop", threads y concurrencia. Existen mejores formas de manejar el game loop y la concurrencia en un juego y las veremos en los próximos tutoriales.

Sprites - Velocidad y dirección

Cada objeto que se mueve en la pantalla tiene características propias como la posición (x,y), la velocidad y la dirección en que se mueve, etc. Todas estas características se pueden aislar en un objeto que llamaremos Sprite.

Velocidad y dirección

En el tutorial anterior logramos que la pelota (el círculo) se moviera hacia abajo y a la derecha a un píxel por vuelta en el Game Loop. Cuando llegaba al límite de la pantalla la pelota seguía su curso desapareciendo del lienzo. Lo que haremos a continuación es que la pelota rebote en los límites del lienzo cambiando su dirección.

```
package com.edu4java.minitennis3;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;

	private void moveBall() {
		if (x + xa < 0)
			xa = 1;
		if (x + xa > getWidth() - 30)
			xa = -1;
		if (y + ya < 0)
			ya = 1;
		if (y + ya > getHeight() - 30)
			ya = -1;

		x = x + xa;
		y = y + ya;
	}

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);
		g.fillOval(x, y, 30, 30);

	}

	public static void main(String[] args) throws InterruptedException {
		JFrame frame = new JFrame("Mini Tennis");
		Game game = new Game();
		frame.add(game);
		frame.setSize(300, 400);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		while (true) {
			game.moveBall();
			game.repaint();
			Thread.sleep(10);
		}
	}
}
```

En el código anterior se agregaron dos propiedades "xa" y "ya" que representan la velocidad en que se mueve la pelota. Si xa=1, la pelota se mueve hacia la derecha a un píxel por vuelta del Game Loop y si xa=-1, la pelota se mueve hacia la izquierda. Similarmente ya=1 mueve hacia abajo y ya=-1 mueve hacia arriba. Esto lo logramos con las líneas **"x = x + xa"** e **"y = y + ya"** del método moveBall().

Antes de ejecutar las instrucciones anteriores verificamos que la pelota no salga de los márgenes del lienzo. Por ejemplo cuando la pelota alcance el margen derecho o lo que es lo mismo cuando (x + xa > getWidth() - 30) lo que haremos es cambiar la dirección del movimiento sobre el eje x o lo que es lo mismo asignar menos uno a xa **"xa = -1"**.

```
	private void moveBall() {
		if (x + xa < 0)
			xa = 1;
		if (x + xa > getWidth() - 30)
			xa = -1;
		if (y + ya < 0)
			ya = 1;
		if (y + ya > getHeight() - 30)
			ya = -1;

		x = x + xa;
		y = y + ya;

	}
```

Cada sentencia if limita un borde del lienzo.

Crear el Sprite Ball (pelota en inglés)

La idea es crear una clase llamada Ball que aísle todo lo referente a la pelota. En el siguiente código podemos ver como extraemos todo el código referente a la pelota de la clase Game2 y lo incorporamos a nuestra nueva clase Ball.

```
package com.edu4java.minitennis3;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game2 extends JPanel {

	Ball ball = new Ball(this);

	private void move() {
		ball.move();
	}

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);
		ball.paint(g2d);
	}

	public static void main(String[] args) throws InterruptedException {
		JFrame frame = new JFrame("Mini Tennis");
		Game2 game = new Game2();
		frame.add(game);
		frame.setSize(300, 400);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		while (true) {
			game.move();
			game.repaint();
			Thread.sleep(10);
		}
	}
}
```

El Sprite Ball necesita que le envíen una referencia al objeto Game para obtener los limites del lienzo y así saber cuándo debe cambiar de dirección. En el método move() de la clase Ball se llama a **game.getWidth()** y **game.getHeight().**

```
package com.edu4java.minitennis3;

import java.awt.Graphics2D;

public class Ball {
	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game2 game;

	public Ball(Game2 game) {
		this.game= game;
	}

	void move() {
		if (x + xa < 0)
			xa = 1;
		if (x + xa > game.getWidth() - 30)
			xa = -1;
		if (y + ya < 0)
			ya = 1;
		if (y + ya > game.getHeight() - 30)
			ya = -1;

		x = x + xa;
		y = y + ya;
	}

	public void paint(Graphics2D g) {
		g.fillOval(x, y, 30, 30);
	}
}
```

Si ejecutamos Game2 obtendremos el mismo resultado que si ejecutamos la versión anterior Game. La conveniencia de esta separación del código referente a la pelota en una clase de tipo Sprite se vuelve más obvia cuando incluimos la raqueta mediante un nuevo Sprite en un próximo tutorial.

Eventos. Capturando las entrada por teclado

En este tutorial veremos cómo funcionan los eventos y en particular como obtener la información acerca de los eventos producidos en el teclado desde un programa java. Además explicaremos el concepto y uso de clases anónimas que es el método más comúnmente usado para manejar eventos en java.

Abandonaremos nuestro juego momentáneamente y haremos un simple ejemplo de captura de eventos.

Ejemplo de lectura del teclado

Para leer del teclado es necesario registrar un objeto que se encargue de "escuchar si una tecla es presionada". Este objeto conocido como "Listener" u "oyente" y tendrá métodos que serán llamados cuando alguien presione una tecla. En nuestro ejemplo el Listener se registra en el JPanel (o KeyboardExample) usando el método addKeyListener(KeyListener listener).

```
package com.edu4java.minitennis4;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class KeyboardExample extends JPanel {

	public KeyboardExample() {
		KeyListener listener = new MyKeyListener();
		addKeyListener(listener);
		setFocusable(true);
	}

	public static void main(String[] args) {
		JFrame frame = new JFrame("Mini Tennis");
		KeyboardExample keyboardExample = new KeyboardExample();
		frame.add(keyboardExample);
		frame.setSize(200, 200);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}

	public class MyKeyListener implements KeyListener {
		@Override
		public void keyTyped(KeyEvent e) {
		}

		@Override
		public void keyPressed(KeyEvent e) {

	System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
		}

		@Override
		public void keyReleased(KeyEvent e) {

	System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
		}
	}
}
```

En el constructor de la clase KeyboardExample creamos el listener y lo registramos. Para que un objeto JPanel reciba las notificaciones del teclado es necesario incluir la instrucción setFocusable(true) que permite que KeyboardExample reciba el foco.

```
public KeyboardExample() {
        KeyListener listener = new MyKeyListener();
        addKeyListener(listener);
        setFocusable(true);
}
```

La clase MyKeyListener es la que uso para crear el objeto Listener. Este Listener imprimirá en la consola el nombre del método y la tecla afectada por el evento.

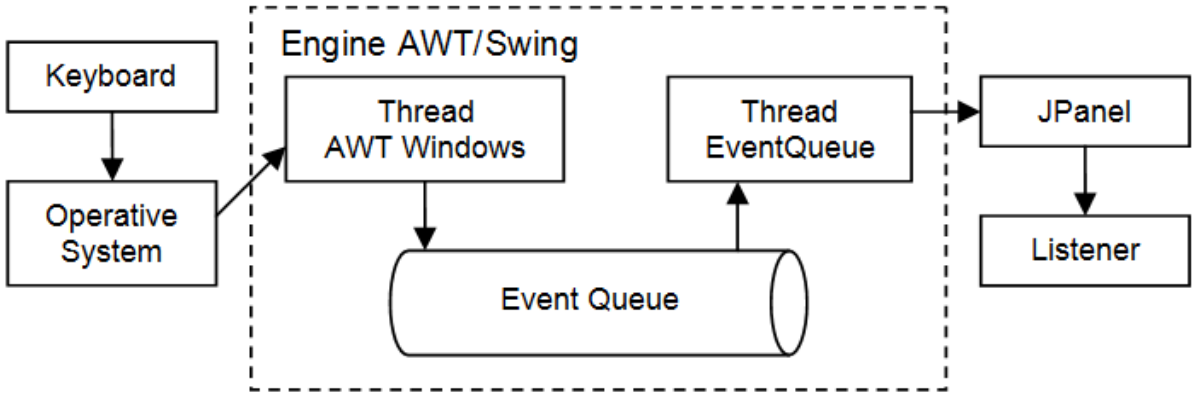
```
public class MyKeyListener implements KeyListener {
        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {

System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
        }

        @Override
        public void keyReleased(KeyEvent e) {

System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
        }
}
```

Una vez registrado, cuando KeyboardExample (nuestro JPanel) tenga el foco y alguien oprima una tecla KeyboardExample informará al objeto listener registrado. El objeto Listener de nuestro ejemplo implementa la interfaz KeyListener que tiene los métodos keyTyped(), keyPressed() y keyReleased(). El método keyPressed será llamado cada vez que una tecla sea oprimida (y varias veces si se mantiene oprimida). El método keyReleased será llamado cuando solemos una tecla.

Los métodos antes mencionados reciben como parámetro un objeto KeyEvent que contiene información sobre que tecla se ha oprimido o soltado. Usando e.getKeyCode() podemos obtener el código de la tecla y si le pasamos un código de tecla a la función estatica KeyEvent.getKeyText(...) podemos obtener el texto asociado a la tecla.

¿Cómo funcionan los eventos en AWT/Swing?

Lo eventos del ratón y el teclado son controlados por el sistema operativo. El motor AWT, en particular el thread AWT-Windows se comunica con el sistema operativo y se entera de si hubo un evento. Cuando encuentra un nuevo evento lo coloca en la "Cola de Eventos" para que sea atendido cuando le llegue su turno por el Thread AWT-EventQueue.



Cuando el Thread AWT-EventQueue atiende a un evento se fija a que componente afecta y le informa. En nuestro caso el componente es el JPanel que informa a todos los listeners que se hayan registrado para recibir notificaciones de ese evento.

En el caso del teclado la llamada addKeyListener(KeyListener listener) es la que realiza este registro. Si queremos registrar un objeto para escuchar los eventos del ratón podemos usar addMouseListener(MouseListener listener).

Si quieren profundizar en cómo funcionan los eventos en AWT/Swing les recomiendo el siguiente artículo.

Clase anónima

En el ejemplo anterior la clase MyKeyListener será solo usada una vez por lo que podríamos reemplazarla por una clase anónima. KeyboardExample2 muestra como sería:

```
package com.edu4java.minitennis4;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class KeyboardExample2 extends JPanel {

	public KeyboardExample2() {
		KeyListener listener = new KeyListener() {
			@Override
			public void keyTyped(KeyEvent e) {
			}

			@Override
			public void keyPressed(KeyEvent e) {

	System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
			}

			@Override
			public void keyReleased(KeyEvent e) {

	System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
			}
		};
		addKeyListener(listener);
		setFocusable(true);
	}

	public static void main(String[] args) {
		JFrame frame = new JFrame("Mini Tennis");
		KeyboardExample2 keyboardExample = new KeyboardExample2();
		frame.add(keyboardExample);
		frame.setSize(200, 200);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

En el constructor de la clase KeyboardExample2 podemos ver como se reemplaza

```
        KeyListener listener = new MyKeyListener();
por
                KeyListener listener = new KeyListener() {
                        @Override
                        public void keyTyped(KeyEvent e) {
                        }

                        @Override
                        public void keyPressed(KeyEvent e) {

         System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
                        }

                        @Override
                        public void keyReleased(KeyEvent e) {

         System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
                        }
                };
```

Esta instrucción tiene el mismo efecto que la anterior. Reemplaza la definición de la clase MyKeyListener por una clase anónima que hace exactamente lo mismo.

La forma de crear un objeto desde una clase anónima es reemplazar el nombre de la clase a crear por una definición que empieza por la interfaz a implementar seguida por () y luego dentro de {} la definición de la clase como hacemos normalmente.

Aunque parezca un poco extraño esta es la forma más cómoda de implementar Listeners de eventos y es la forma que más encontrarán en código java avanzado.

Ahora sigamos con el desarrollo de nuestro juego en el próximo tutorial.

Agregando el sprite raqueta

En este tutorial agregaremos la raqueta mediante un Sprite llamado Racquet. La raqueta se moverá hacia la izquierda o derecha cuando oprimamos las teclas del cursor por lo que nuestro programa necesita leer del teclado.

Nuevo Sprite Racquet

Lo primero que hacemos es agregar en la clase Game una nueva propiedad llamada racquet donde mantendremos el Sprite que maneja la raqueta. En el método move() añadimos una llamada a racquet.move() y en paint() una llamada a racquet.paint(). Hasta ahora todo es similar al sprite Ball pero como la posición de la raqueta responde al teclado tenemos que hacer algo más.

```
package com.edu4java.minitennis5;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	Ball ball = new Ball(this);
	Racquet racquet = new Racquet(this);

	public Game() {
		addKeyListener(new KeyListener() {
			@Override
			public void keyTyped(KeyEvent e) {
			}

			@Override
			public void keyReleased(KeyEvent e) {
				racquet.keyReleased(e);
			}

			@Override
			public void keyPressed(KeyEvent e) {
				racquet.keyPressed(e);
			}
		});
		setFocusable(true);
	}

	private void move() {
		ball.move();
		racquet.move();
	}

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);
		ball.paint(g2d);
		racquet.paint(g2d);
	}

	public static void main(String[] args) throws InterruptedException {
		JFrame frame = new JFrame("Mini Tennis");
		Game game = new Game();
		frame.add(game);
		frame.setSize(300, 400);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		while (true) {
			game.move();
			game.repaint();
			Thread.sleep(10);
		}
	}
}
```

En el constructor de la clase game se puede ver como se registra un listener para capturar los eventos del teclado. En el método keyPressed() del listener informamos a la raqueta que una tecla ha sido oprimida llamando a racquet.keyPressed(e). Lo mismo hacemos para keyReleased(). Con esto el Sprite racquet se enterará cuando una tecla sea oprimida. Veamos ahora las clases Ball y Racquet que implementan los sprites.

```
package com.edu4java.minitennis5;

import java.awt.Graphics2D;

public class Ball {
	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game game;

	public Ball(Game game) {
		this.game= game;
	}

	void move() {
		if (x + xa < 0)
			xa = 1;
		if (x + xa > game.getWidth() - 30)
			xa = -1;
		if (y + ya < 0)
			ya = 1;
		if (y + ya > game.getHeight() - 30)
			ya = -1;

		x = x + xa;
		y = y + ya;
	}

	public void paint(Graphics2D g) {
		g.fillOval(x, y, 30, 30);
	}
}
```

La clase Ball no tiene cambios. Comparémosla con la clase Racquet:

```
package com.edu4java.minitennis5;

import java.awt.Graphics2D;
import java.awt.event.KeyEvent;

public class Racquet {
        int x = 0;
        int xa = 0;
        private Game game;

        public Racquet(Game game) {
                this.game= game;
        }

        public void move() {
                if (x + xa > 0 && x + xa < game.getWidth()-60)
                        x = x + xa;
        }

        public void paint(Graphics2D g) {
                g.fillRect(x, 330, 60, 10);
        }

        public void keyReleased(KeyEvent e) {
                xa = 0;
        }

        public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_LEFT)
                        xa = -1;
                if (e.getKeyCode() == KeyEvent.VK_RIGHT)
                        xa = 1;
        }
}
```

A diferencia de Ball, Racquet no tiene propiedades para la posición "y" ni la velocidad "ya". Esto es debido a que la raqueta no variará su posición vertical, solo se moverá hacia la izquierda o derecha, nunca hacia arriba o abajo. En el método paint la instrucción g.fillRect(x, 330, 60, 10) define un rectángulo de 60 por 10 pixeles en la posición (x,y)=(x,330). Como vemos "x" puede variar pero "y" está fijada a 330 pixeles del limite superior del lienzo.

El método move() es similar al de Ball en el sentido de incrementar en "xa" la posición "x" y controlar que el sprite no se salga de los limites.

```
        public void move() {
                if (x + xa > 0 && x + xa < game.getWidth()-60)
                        x = x + xa;
        }
```

Inicialmente el valor de "x" es cero lo que indica que la raqueta estará en el limite izquierdo del lienzo. "xa" también está inicializado a cero, lo que hace que en principio la raqueta aparezca estática ya que x = x + xa no modificará "x" mientras "xa" sea cero.

Cuando alguien presione una tecla el método keyPressed de Racquet será llamado y este pondrá "xa" en 1 si la tecla presionada es la de dirección derecha (KeyEvent.VK_RIGHT) lo que a su vez hará que la raqueta se mueva a la derecha la próxima vez que se llame al método move (recordar x = x + xa). De la misma forma si se presiona la tecla KeyEvent.VK_LEFT se moverá a la izquierda.

```
public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_LEFT)
                xa = -1;
        if (e.getKeyCode() == KeyEvent.VK_RIGHT)
                xa = 1;
}
```

Cuando una tecla deja de ser presionada el método keyReleased es llamado y "xa" pasa a valer cero lo que hace que el movimiento de la raqueta se detenga.

```
public void keyReleased(KeyEvent e) {
        xa = 0;
}
```

Si ejecutamos el ejemplo podemos ver como la pelota se mueve rebotando contra los límites y la raqueta se mueve cuando presionamos las teclas de dirección correspondientes. Pero cuando la pelota choca con la raqueta la atraviesa pareciendo como si esta no existiese. En el próximo tutorial veremos como hacer que la pelota rebote sobre la raqueta.

Detección de colisiones

En este tutorial aprenderemos como detectar cuando un sprite choca con otro. En nuestro juego haremos que la pelota rebote contra la raqueta. Además haremos que el juego termine si la pelota alcanza el límite inferior del lienzo mostrando una ventana popup con el clásico mensaje "Game Over".

Game Over

A continuación vemos nuestra clase Game que es idéntica a la anterior con la sola diferencia de que se ha agregado el método gameOver();

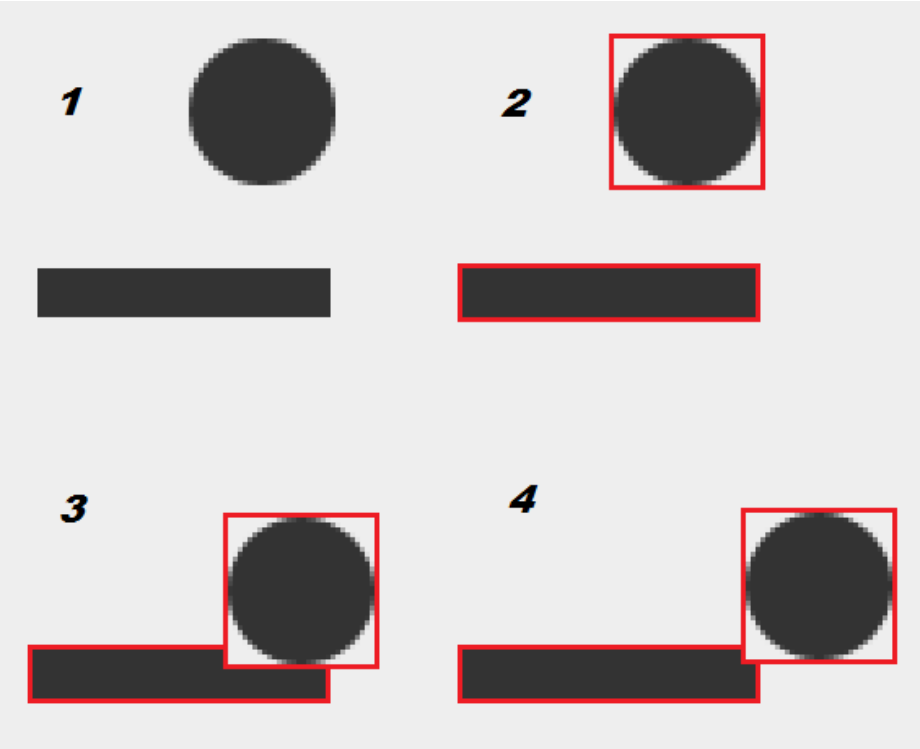
```
package com.edu4java.minitennis6;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	Ball ball = new Ball(this);
	Racquet racquet = new Racquet(this);

	public Game() {
		addKeyListener(new KeyListener() {
			@Override
			public void keyTyped(KeyEvent e) {
			}

			@Override
			public void keyReleased(KeyEvent e) {
				racquet.keyReleased(e);
			}

			@Override
			public void keyPressed(KeyEvent e) {
				racquet.keyPressed(e);
			}
		});
		setFocusable(true);
	}

	private void move() {
		ball.move();
		racquet.move();
	}

	@Override
	public void paint(Graphics g) {
		super.paint(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);
		ball.paint(g2d);
		racquet.paint(g2d);
	}

	public void gameOver() {
```

```
                JOptionPane.showMessageDialog(this, "Game Over", "Game Over",
JOptionPane.YES_NO_OPTION);
                System.exit(ABORT);
        }

        public static void main(String[] args) throws InterruptedException {
                JFrame frame = new JFrame("Mini Tennis");
                Game game = new Game();
                frame.add(game);
                frame.setSize(300, 400);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                while (true) {
                        game.move();
                        game.repaint();
                        Thread.sleep(10);
                }
        }
}
```

El método gameOver() lanza un popup usando JOptionPane.showMessageDialog con el mensaje "Game Over" y un solo botón "Aceptar". Después del popup, System.exit(ABORT) hace que se termine el programa. El método gameOver() es público ya que será llamado desde el sprite Ball cuando detecte que ha llegado al límite inferior del lienzo.

Colisión de Sprites

Para detectar la colisión entre la pelota y la raqueta usaremos rectángulos. El caso de la pelota crearemos un cuadrado alrededor de la pelota como se ve en la figura 2.



La clase java.awt.Rectangle tiene un método intersects(Rectangle r) que retorna true cuando dos rectángulos ocupan el mismo espacio como en el caso de la figura 3 o 4. Cabe destacar que este método no es exacto ya que en la figura 4 la pelota no toca a la raqueta pero para nuestro ejemplo será más que suficiente.

A continuación vemos la clase Racquet donde el único cambio funcional es que se ha agregado el método getBounds() que retorna un objeto de tipo rectángulo indicando la posición de la raqueta. Este método será usado por el sprite Ball para saber la posición de la raqueta y así detectar la colisión.

```
package com.edu4java.minitennis6;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class Racquet {
	private static final int Y = 330;
	private static final int WITH = 60;
	private static final int HEIGHT = 10;
	int x = 0;
	int xa = 0;
	private Game game;

	public Racquet(Game game) {
		this.game = game;
	}

	public void move() {
		if (x + xa > 0 && x + xa < game.getWidth() - WITH)
			x = x + xa;
	}

	public void paint(Graphics2D g) {
		g.fillRect(x, Y, WITH, HEIGHT);
	}

	public void keyReleased(KeyEvent e) {
		xa = 0;
	}

	public void keyPressed(KeyEvent e) {
		if (e.getKeyCode() == KeyEvent.VK_LEFT)
			xa = -1;
		if (e.getKeyCode() == KeyEvent.VK_RIGHT)
			xa = 1;
	}

	public Rectangle getBounds() {
		return new Rectangle(x, Y, WITH, HEIGHT);
	}

	public int getTopY() {
		return Y - HEIGHT;
	}
}
```

Otro cambio que funcionalmente no afecta pero que es una buena práctica de programación es la inclusión de constantes:

```
        private static final int Y = 330;
        private static final int WITH = 60;
        private static final int HEIGH = 20;
```

Como antes mencionamos el valor de posición "y" estaba fijo en 330. Este valor es usado tanto en el método paint como en getBounds. Si queremos cambiarlo ahora sólo tenemos que cambiarlo en un sólo lugar evitando el posible error que se produciría si lo cambiáramos en un método y en otro no.

La forma de definir una constante en java es declarando una propiedad "static final" y en mayúsculas. El compilador permite usar minúsculas pero el estándar dice que se deben usar mayúsculas para los nombres de las constantes.

Por último la clase Ball:

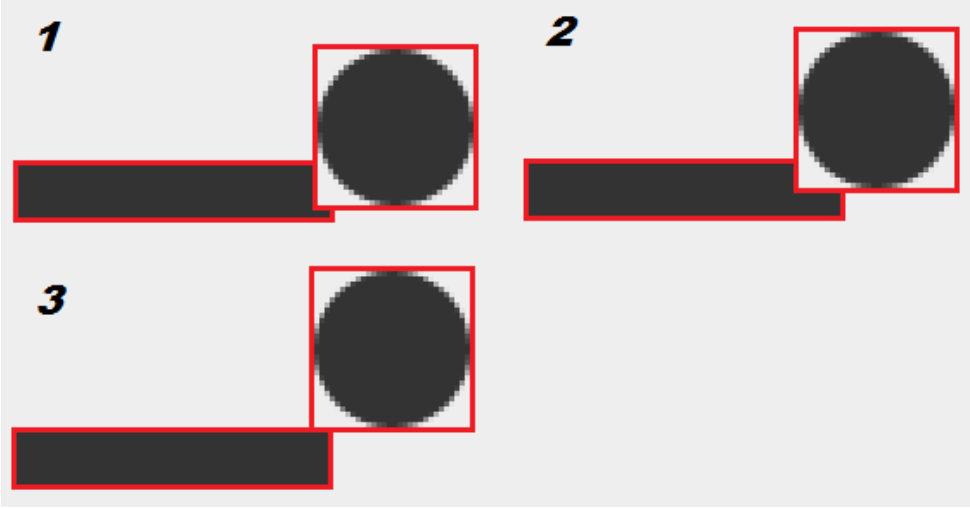
```
package com.edu4java.minitennis6;

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
        private static final int DIAMETER = 30;
        int x = 0;
        int y = 0;
        int xa = 1;
        int ya = 1;
        private Game game;

        public Ball(Game game) {
                this.game= game;
        }

        void move() {
                if (x + xa < 0)
                        xa = 1;
                if (x + xa > game.getWidth() - DIAMETER)
                        xa = -1;
                if (y + ya < 0)
                        ya = 1;
                if (y + ya > game.getHeight() - DIAMETER)
                        game.gameOver();
                if (collision()){
                        ya = -1;
                        y = game.racquet.getTopY() - DIAMETER;
                }
                x = x + xa;
                y = y + ya;
        }

        private boolean collision() {
                return game.racquet.getBounds().intersects(getBounds());
        }

        public void paint(Graphics2D g) {
                g.fillOval(x, y, DIAMETER, DIAMETER);
        }

        public Rectangle getBounds() {
                return new Rectangle(x, y, DIAMETER, DIAMETER);
        }
}
```

De forma similar a la clase Racquet se ha incluido el método getBounds() y la constante DIAMETER.

Más interesante es la aparición de un nuevo método llamado collision() que retorna true (verdadero) si el rectángulo ocupado por la raqueta "game.racquet.getBounds()" intersecta al rectángulo que encierra a la pelota "getBounds()".

```
private boolean collision() {
        return game.racquet.getBounds().intersects(getBounds());
}
```

Si la colisión se produce, además de cambiar la dirección ajustaremos la posición de la pelota. Si la colisión es por el lado (figura 1), la pelota podría estar varios pixeles por debajo de la cara superior de la raqueta. En el siguiente game loop aunque la pelota se movería hacia arriba (figura 2) podría todavía estar en colisión con la raqueta.



Para evitar esto colocamos a la pelota sobre la raqueta (figura 3) mediante:

```
y = game.racquet.getTopY() - DIAMETER;
```

El método getTopY() de Racquet nos da la posición en el eje y de la parte superior de la raqueta y restando DIAMETER conseguimos la posición y exacta donde colocar la pelota para que esté sobre la raqueta.

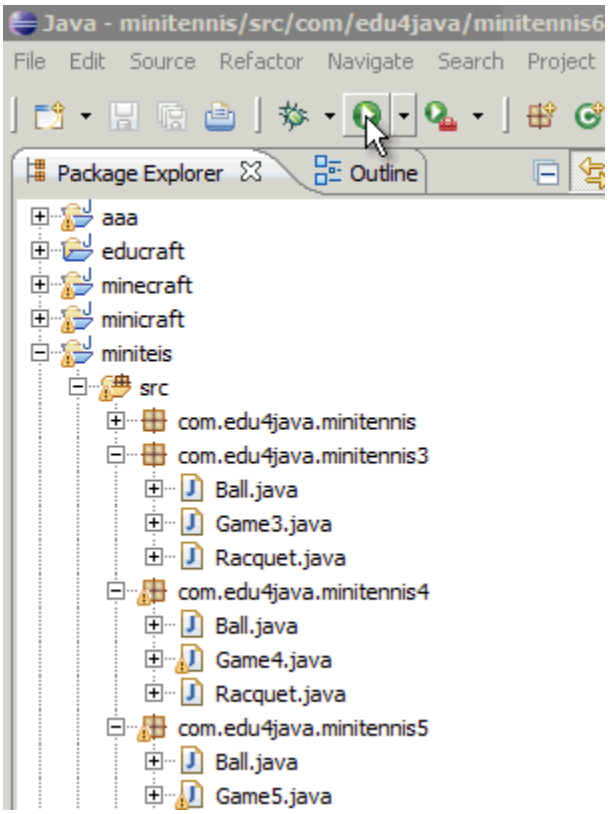
Por último es el método move() de la clase Ball el que usa los nuevos métodos collision() y gameOver() de la clase Game. El rebote al alcanzar el lÍmite inferior ha sido reemplazado por una llamada a game.gameOver().

```
if (y + ya > game.getHeight() - DIAMETER)
        game.gameOver();
```

Y poniendo un nuevo condicional usando el método collision() logramos que la pelota rebote hacia arriba si esta colisiona con la raqueta:

```
if (collision())
        ya = -1;
```

Si ejecutamos el ejemplo podemos ver:

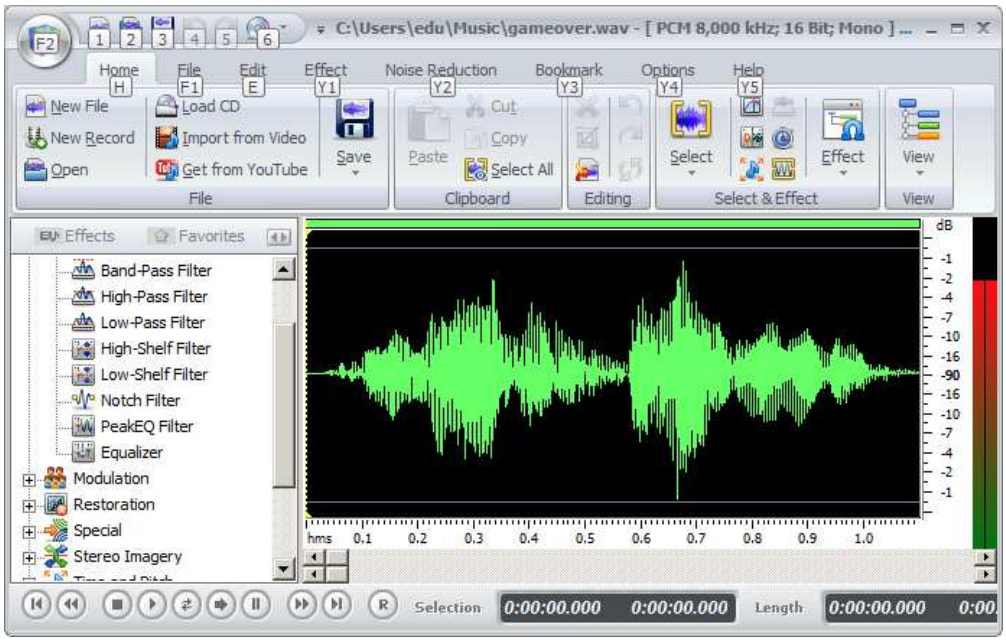


Agregando sonido a nuestro juego

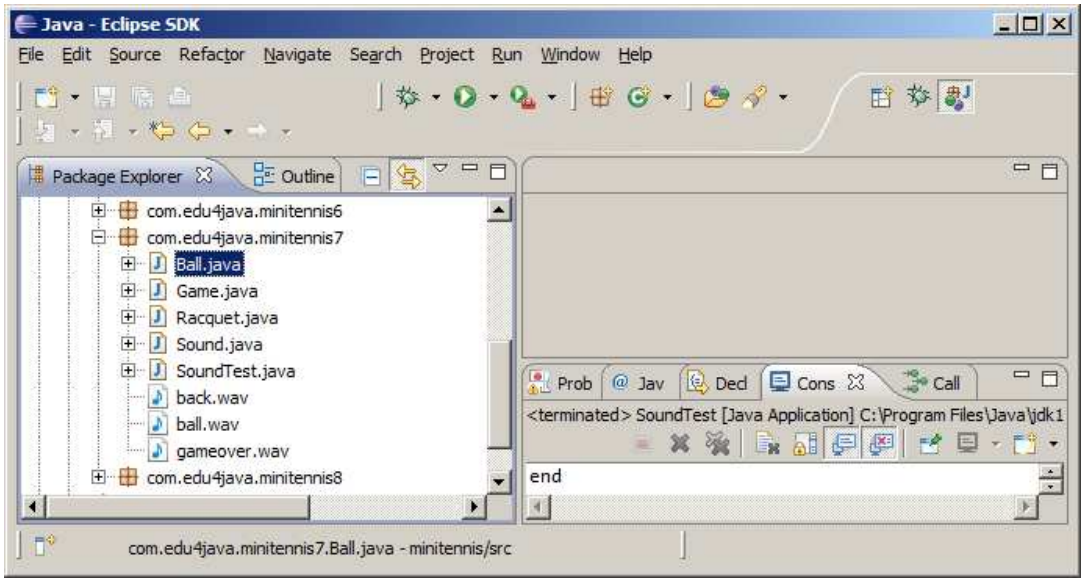
Un juego sin sonido no está completo. En este tutorial agregaremos música de fondo, el ruido del rebote de la pelota y un "Game Over" con voz graciosa al terminar el juego. Para evitar problemas de copyright vamos a crear nosotros mismos los sonidos.

Creando sonidos

Para crear los sonidos me tomé la libertad de buscar en Google "free audio editor" y como respuesta encontré http://free-audio-editor.com/. Tengo que decir que la versión gratis de este producto es potente y fácil de manejar.



Con este editor he creado los archivos: back.wav, gameover.wav y ball.wav. En el video de youtube pueden ver como lo hice y crearlos ustedes mismos. También pueden descargar y usar estos tres que en esta misma línea los declaro libres de copyright. Lo que tienen que hacer es copiar estos archivos al paquete com.edu4java.minitennis7.



Reproducir sonidos usando AudioClip

Para reproducir los archivos de sonido usaremos la clase AudioClip. Crearemos objetos AudioClip usando el método estático de la clase Applet: Applet.newAudioClip(URL url). Este método necesita un objeto URL que le indique donde está el archivo de audio que queremos cargar para luego reproducir. La siguiente instrucción crea un objeto URL utilizando una ubicación en Internet:

```
URL url = new URL("http://www.edu4java.com/es/game/sound/back.wav");
```

La siguiente utiliza un directorio dentro del sistema de archivos local:

```
URL url = new
URL("file:/C:/eclipseClasic/workspace/minitennis/src/com/edu4java/minitennis7/back.
wav");
```

Nosotros buscaremos nuestro archivo utilizando el classpath. Este es el sistema que usa java para cargar las clases o mejor dicho los archivos *.class que definen las clases del programa. Para obtener un URL desde el classpath se utiliza el método getResource(String name) de la clase Class donde name es el nombre del archivo que queremos obtener.

A continuación vemos dos formas de como conseguir el URL del archivo "back.wav" que está en el mismo paquete que la clase SoundTest o lo que es lo mismo en el mismo directorio donde esta el archivo SoundTest.class.

```
URL url = SoundTest.class.getResource("back.wav");
```

```
URL url = new SoundTest().getClass().getResource("back.wav");
```

Tanto "SoundTest.class" como "new SoundTest().getClass()" nos dan un objeto class que tiene el método getResource que queremos usar.

He creado la clase SoundTest con el sólo propósito de mostrarles como trabaja AudioClip y no es necesaria para nuestro juego. A continuación se muestra el código fuente de SoundTest completo:

```
package com.edu4java.minitennis7;

import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;

public class SoundTest {
	public static void main(String[] args) throws Exception {

//		System.out.println("1");
//		URL url = new URL("http://www.edu4java.com/es/game/sound/back.wav");
//		System.out.println("2");
//		AudioClip clip = Applet.newAudioClip(url);
//		System.out.println("3");
//		clip.play();
//		System.out.println("4");
//		Thread.sleep(1000);

//		URL url = new URL(
//
	"file:/C:/eclipseClasic/workspace/minitennis/src/com/edu4java/minitennis7/ba
ck.wav");

		URL url = SoundTest.class.getResource("back.wav");
		AudioClip clip = Applet.newAudioClip(url);
		AudioClip clip2 = Applet.newAudioClip(url);
		clip.play();
		Thread.sleep(1000);
		clip2.loop();
		Thread.sleep(20000);
		clip2.stop();

		System.out.println("end");
	}
}
```

De esta forma el archivo back.wav se obtienen desde el classpath. El classpath es el conjunto de directorios y archivos *.jar desde donde nuestro programa puede leer las clases (archivos *.class).

Una ventaja de esta metodología es que sólo tenemos que indicar la posición del archivo con respecto a la clase que lo usa. En nuestro caso como está en el mismo paquete basta con el nombre "back.wav". Otra ventaja es que los archivos de sonido se pueden incluir en un archivo *.jar. Veremos más sobre archivos *.jar más adelante. Una ves que tenemos el objeto URL podemos crear objetos AudioClip usando Applet.newAudioClip(url).

```
		AudioClip clip = Applet.newAudioClip(url);
		AudioClip clip2 = Applet.newAudioClip(url);
```

El objeto AudioClip tiene un método play() que inicia un thread independiente que reproduce sólo una vez el audio contenido en el archivo. Para reproducir el audio en forma repetitiva podemos usar el método loop() de AudioClip que reproducirá el sonido una y otra vez hasta que se llame al método stop sobre el mismo objeto AudioClip.

Dos audioClips pueden reproducirse al mismo tiempo. En el ejemplo creo dos audioClips con el mismo audio: clip y clip2. Reproduzco clip con play, espero un segundo Thread.sleep(1000) y reproduzco clip2 con loop. El resultado es una mezcla de los dos audios. Por último después de 20 segundos Thread.sleep(20000) llamo a clip2.stop() y detengo la repetición de clip2.

Creando una clase Sound para nuestro juego

Para guardar los audioclips de nuestro juego creamos una clase Sound que tendrá una constante con un audioclip por cada sonido que usemos. Estas constantes son públicas para que cualquier objeto que tenga acceso a ellas pueda reproducirlas. Por ejemplo en la clase Ball podemos reproducir el sonido del rebote de la pelota usando Sound.BALL.play() en el momento que detectamos que la pelota cambia de dirección.

```
package com.edu4java.minitennis7;

import java.applet.Applet;
import java.applet.AudioClip;

public class Sound {
	public static final AudioClip BALL =
Applet.newAudioClip(Sound.class.getResource("ball.wav"));
	public static final AudioClip GAMEOVER =
Applet.newAudioClip(Sound.class.getResource("gameover.wav"));
	public static final AudioClip BACK =
Applet.newAudioClip(Sound.class.getResource("back.wav"));
}
```

Los objetos audioclips se crearán al cargarse la clase Sound la primera vez que alguien use la clase Sound. A partir de este momento serán reutilizados una y otra vez. Ahora veamos las modificaciones en la clase Game:

```
package com.edu4java.minitennis7;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	Ball ball = new Ball(this);
	Racquet racquet = new Racquet(this);

	public Game() {
		addKeyListener(new KeyListener() {
			@Override
			public void keyTyped(KeyEvent e) {
			}

			@Override
			public void keyReleased(KeyEvent e) {
				racquet.keyReleased(e);
			}

			@Override
			public void keyPressed(KeyEvent e) {
				racquet.keyPressed(e);
			}
		});
		setFocusable(true);
		Sound.BACK.loop();
	}
```

```
        private void move() {
                ball.move();
                racquet.move();
        }

        @Override
        public void paint(Graphics g) {
                super.paint(g);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                RenderingHints.VALUE_ANTIALIAS_ON);
                ball.paint(g2d);
                racquet.paint(g2d);
        }

        public void gameOver() {
                Sound.BACK.stop();
                Sound.GAMEOVER.play();
                JOptionPane.showMessageDialog(this, "Game Over", "Game Over",
JOptionPane.YES_NO_OPTION);
                System.exit(ABORT);
        }

        public static void main(String[] args) throws InterruptedException {
                JFrame frame = new JFrame("Mini Tennis");
                Game game = new Game();
                frame.add(game);
                frame.setSize(300, 400);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                while (true) {
                        game.move();
                        game.repaint();
                        Thread.sleep(10);
                }
        }
}
```

En la última línea del constructor de la clase Game añadimos Sound.BACK.loop(), lo que iniciará la reproducción de nuestra música de fondo que se repetirá hasta que se alcance el método gameOver(), donde detenemos la música de fondo con Sound.BACK.stop(). A continuación de Sound.BACK.stop() y antes del popup informamos que se termino la partida reproduciendo "Game Over" Sound.GAMEOVER.play().

En la clase Ball modificamos el método move() para que se reproduzca Sound.BALL cuando la pelota rebote.

```
package com.edu4java.minitennis7;

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
	private static final int DIAMETER = 30;

	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game game;

	public Ball(Game game) {
		this.game = game;
	}

	void move() {
		boolean changeDirection = true;
		if (x + xa < 0)
			xa = 1;
		else if (x + xa > game.getWidth() - DIAMETER)
			xa = -1;
		else if (y + ya < 0)
			ya = 1;
		else if (y + ya > game.getHeight() - DIAMETER)
			game.gameOver();
		else if (collision()){
			ya = -1;
			y = game.racquet.getTopY() - DIAMETER;
		} else
			changeDirection = false;

		if (changeDirection)
			Sound.BALL.play();
		x = x + xa;
		y = y + ya;
	}

	private boolean collision() {
		return game.racquet.getBounds().intersects(getBounds());
	}

	public void paint(Graphics2D g) {
		g.fillOval(x, y, DIAMETER, DIAMETER);
	}

	public Rectangle getBounds() {
		return new Rectangle(x, y, DIAMETER, DIAMETER);
	}
}
```

Lo que hice en move() es agregar una variable changeDirection que inicializo a true. Añadiendo un else a cada if y colocando un changeDirection = false que sólo se ejecutará si ninguna condición en los if es cumplida, conseguimos enterarnos si la bola ha rebotado. Si la pelota ha rebotado changeDirection será verdadero y Sound.BALL.play() será ejecutado.

Agregando puntuación y aumentando la velocidad

Todo juego necesita una medida de logro o éxito. En nuestro caso incluiremos en el rincón izquierdo de la pantalla nuestra puntuación que no será más que la cantidad de veces que logramos pegarle a la pelota con la raqueta. Por otro lado el juego debería ser cada vez más difícil para no matar de aburrimiento al jugador. Para esto aumentaremos la velocidad del juego cada vez que rebote la pelota en la raqueta.

Los objetos móviles del juego son la pelota y la raqueta. Modificando la velocidad de movimiento de estos dos objetos modificaremos la velocidad del juego. Vamos a incluir una propiedad llamada speed en la clase Game para mantener la velocidad del juego. La propiedad speed será inicialmente 1 e irá incrementándose cada vez que le demos a la pelota con la raqueta.

Para la puntuación necesitaríamos otra propiedad a incrementar cada vez que golpeemos la pelota. En vez de crear una nueva propiedad se me ocurrió reutilizar speed. El único inconveniente es que las puntuaciones suelen iniciarse en 0 y no en 1 como speed. La solución que se me ocurrió fue agregar un método getScore() que retorne el valor de speed menos uno.

```
	private int getScore() {
		return speed - 1;
	}
```

Veamos las modificaciones hechas en la clase Game:

```
package com.edu4java.minitennis8;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

	Ball ball = new Ball(this);
	Racquet racquet = new Racquet(this);
	int speed = 1;

	private int getScore() {
		return speed - 1;
	}

	public Game() {
		addKeyListener(new KeyListener() {
			@Override
			public void keyTyped(KeyEvent e) {
			}

			@Override
			public void keyReleased(KeyEvent e) {
				racquet.keyReleased(e);
			}

			@Override
			public void keyPressed(KeyEvent e) {
				racquet.keyPressed(e);
			}
```

```
                });
                setFocusable(true);
                Sound.BACK.loop();
        }

        private void move() {
                ball.move();
                racquet.move();
        }

        @Override
        public void paint(Graphics g) {
                super.paint(g);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                RenderingHints.VALUE_ANTIALIAS_ON);
                ball.paint(g2d);
                racquet.paint(g2d);

                g2d.setColor(Color.GRAY);
                g2d.setFont(new Font("Verdana", Font.BOLD, 30));
                g2d.drawString(String.valueOf(getScore()), 10, 30);
        }

        public void gameOver() {
                Sound.BACK.stop();
                Sound.GAMEOVER.play();
                JOptionPane.showMessageDialog(this, "your score is: " + getScore(),
                                "Game Over", JOptionPane.YES_NO_OPTION);
                System.exit(ABORT);
        }

        public static void main(String[] args) throws InterruptedException {
                JFrame frame = new JFrame("Mini Tennis");
                Game game = new Game();
                frame.add(game);
                frame.setSize(300, 400);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                while (true) {
                        game.move();
                        game.repaint();
                        Thread.sleep(10);
                }
        }
}
```

Para pintar la puntuación en el rincón superior izquierdo al final del método paint he agregado:

```
                g2d.setColor(Color.GRAY);
                g2d.setFont(new Font("Verdana", Font.BOLD, 30));
                g2d.drawString(String.valueOf(getScore()), 10, 30);
```

En la primera línea elegimos el color gris, en la segunda línea el tipo de letra Verdana, negrita de 30 pixeles y finalmente en la posición (x,y) igual a (10,30) donde dibujamos la puntuación.

En el método gameOver() modificamos el segundo parámetro para mostrar la puntuación alcanzada:

```
                JOptionPane.showMessageDialog(this, "your score is: " + getScore(),
                                "Game Over", JOptionPane.YES_NO_OPTION);
```

En la clase Ball el método move() ha sido modificado para considerar la nueva propiedad de velocidad "game.speed". Cuando la pelota cambiaba de dirección las propiedades de velocidad xa y ya eran modificadas a 1 o -1. Ahora considerando la velocidad estas propiedades son cambiadas a game.speed o -game.speed. También se ha agregado en el condicional if(collision()) que la velocidad se incremente "game.speed++".

```
package com.edu4java.minitennis8;

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
	private static final int DIAMETER = 30;

	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game game;

	public Ball(Game game) {
		this.game = game;
	}

	void move() {
		boolean changeDirection = true;
		if (x + xa < 0)
			xa = game.speed;
		else if (x + xa > game.getWidth() - DIAMETER)
			xa = -game.speed;
		else if (y + ya < 0)
			ya = game.speed;
		else if (y + ya > game.getHeight() - DIAMETER)
			game.gameOver();
		else if (collision()){
			ya = -game.speed;
			y = game.racquet.getTopY() - DIAMETER;
			game.speed++;
		} else
			changeDirection = false;

		if (changeDirection)
			Sound.BALL.play();
		x = x + xa;
		y = y + ya;
	}

	private boolean collision() {
		return game.racquet.getBounds().intersects(getBounds());
	}

	public void paint(Graphics2D g) {
		g.fillOval(x, y, DIAMETER, DIAMETER);
	}

	public Rectangle getBounds() {
		return new Rectangle(x, y, DIAMETER, DIAMETER);
	}
}
```

A continuación vemos la clase Racquet:

```
package com.edu4java.minitennis8;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class Racquet {
	private static final int Y = 330;
	private static final int WITH = 60;
	private static final int HEIGHT = 10;
	int x = 0;
	int xa = 0;
	private Game game;

	public Racquet(Game game) {
		this.game = game;
	}

	public void move() {
		if (x + xa > 0 && x + xa < game.getWidth() - WITH)
			x = x + xa;
	}

	public void paint(Graphics2D g) {
		g.fillRect(x, Y, WITH, HEIGHT);
	}

	public void keyReleased(KeyEvent e) {
		xa = 0;
	}

	public void keyPressed(KeyEvent e) {
		if (e.getKeyCode() == KeyEvent.VK_LEFT)
			xa = -game.speed;
		if (e.getKeyCode() == KeyEvent.VK_RIGHT)
			xa = game.speed;
	}

	public Rectangle getBounds() {
		return new Rectangle(x, Y, WITH, HEIGHT);
	}

	public int getTopY() {
		return Y - HEIGHT;
	}
}
```

Aquí la modificación es similar que en Ball. En el método keyPressed(KeyEvent e) la modificación de la velocidad xa pasa de -1 y 1 a -game.speed y game.speed.

Nota: según el estándar de "Java Beans" el acceso a la propiedad "game.speed" debería hacerse usando un método de la forma "game.getSpeed()". El accesos directo a una propiedad es considerado casi un pecado mortal en el ámbito de java empresarial. Curiosamente en el entorno de desarrollo de juegos es muy común y está justificado por eficiencia. Esto es especialmente importante en programación para móviles donde los recursos suelen ser más escasos.

Los objetos móviles del juego son la pelota y la raqueta. Modificando la velocidad de movimiento de estos dos objetos modificaremos la velocidad del juego. Vamos a incluir una propiedad llamada speed en la clase Game para mantener la velocidad del juego. La propiedad speed será inicialmente 1 e irá incrementándose cada vez que le demos a la pelota con la raqueta.

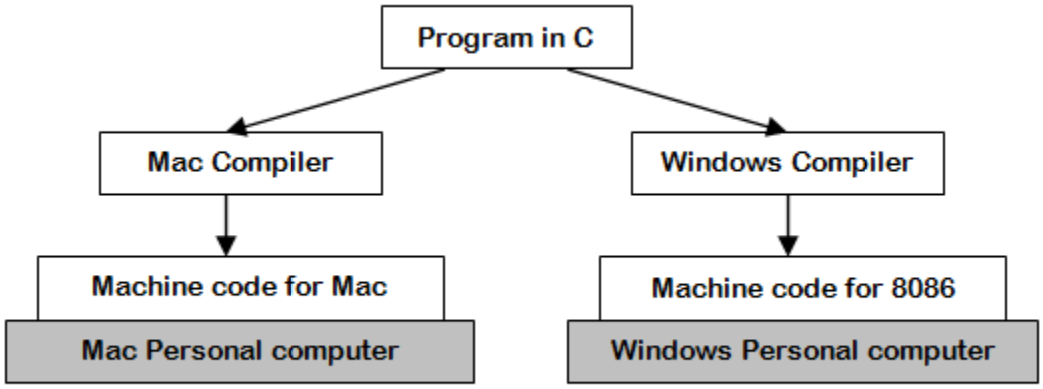
Creando archivo jar ejecutable y qué es la máquina virtual de java.

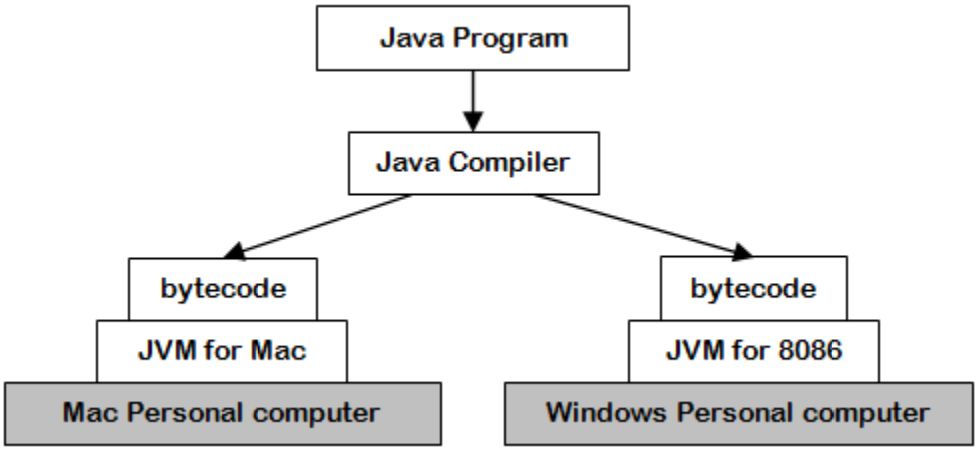
En este tutorial veremos cómo crear un archivo ejecutable para una aplicación java, en particular para nuestro juego. Un programa java necesita una maquina virtual para ser ejecutado. A continuación también explicaremos que es la máquina virtual de java y brevemente como funciona.

La máquina virtual de java; Java Virtual Machine (JVM)

Antes de java lo más normal era escribir un programa en un lenguaje de programación de alto nivel como C o Pascal y luego traducirlo a lenguaje de máquina con un compilador. El "lenguaje máquina" o "código máquina" es el lenguaje que entiende la máquina (ordenador o computadora). Una máquina con Windows y un Mac de Apple hablan distinto lenguaje de máquina. Luego se necesita un compilador diferente para cada máquina.

En el caso de java cuando usamos el compilador no obtenemos código máquina. Lo que obtenemos es un código llamado bytecode que no se ejecuta directamente sobre una máquina real. Este bytecode solo se puede ejecutar en una máquina virtual. Una máquina virtual es un programa que se hace pasar por una máquina. Para cada sistema operativo diferente existirá un programa de máquina virtual especifico pero el bytecode que ejecutan será el mismo.





Como el bytecode es el mismo potencialmente puede ser ejecutado es cualquier sistema operativo siempre y cuando exista una implementación de JVM para este SO. En esta idea se basa la famosa frase: "Write once, run anywhere" (WORA) "escribir una vez, ejecutar en cualquier parte".

Compilación y ejecución en java

Existen dos versiones de instalación de java para cada sistema operativo: JRE y JDK. JRE Java Runtime Environment, es una versión reducida que contiene la JVM pero que no incluye el compilador java. JDK Java Development Kit contiene la JVM, el compilador java y muchas herramientas adicionales para el desarrollo de aplicaciones java. Si no tiene instalada la versión JDK tendrán que instalarla para poder continuar con este tutorial.

Si tenemos instalado la JDK tendremos un directorio donde estarán todos los archivos que componen la plataforma java. Este directorio es conocido como java Home o JAVA_HOME. En mi caso este es "C:\Program Files (x86)\Java\jdk1.6.0_21".

Dentro de JAVA_HOME existe una carpeta bin que contiene los ejecutable entre los que podemos encontrar: El compilador: javac.exe y la máquina virtual: java.exe.

Para ejemplificar como funcionan estos programas vamos a crear un archivo llamado HelloWorld.java en un directorio C:\testjava con el siguiente contenido:

```
import javax.swing.JOptionPane;

public class HelloWorld {
	public static void main(String[] args) {
		System.out.println("Hello World ;)");
		JOptionPane.showMessageDialog(null, "Hello World");
	}
}
```

Luego abrimos una ventana de comandos, ejecutamos "cd C:\testjava" para posicionarnos en el directorio donde esta nuestro archivo java y luego para compilar ejecutamos:

```
javac HelloWorld.java
```

o

```
"C:\Program Files (x86)\Java\jdk1.7.0_05\bin\javac" HelloWorld.java
```

Como resultado podemos ver que se ha creado un nuevo archivo HellowWorld.class con el bytecode. Podemos ejecutar este bytecode con la siguiente instrucción:

```
java HelloWorld
```

o

```
"C:\Program Files (x86)\Java\jdk1.7.0_05\bin\java" HelloWorld
```

Un programa java normalmente está compuesto por varios archivos java y por consiguiente muchos archivos *.class. Además están los archivos de recursos como los sonidos en nuestra aplicación. Java permite empaquetar una aplicación con todos los archivos antes mencionados en un archivo *.jar.

Archivo JAR

Un archivo jar no es más que un archivo comprimido con el algoritmo de compresión ZIP que puede contener:

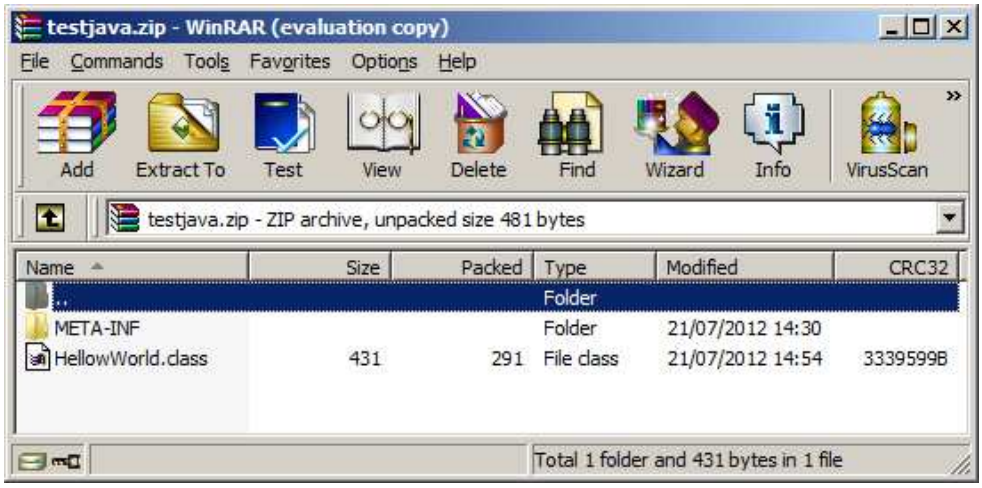
1. Los archivos *.class que se generan a partir de compilar los archivos *.java que componen nuestra aplicación.
2. Los archivos de recursos que necesita nuestra aplicación (Por ejemplo los archivo de sonido *.wav)
3. Opcionalmente se puede incluir los archivos de código fuente *.java
4. Opcionalmente puede existir un archivo de configuración "META-INF/MANIFEST.MF".

Crear un archivo JAR ejecutable

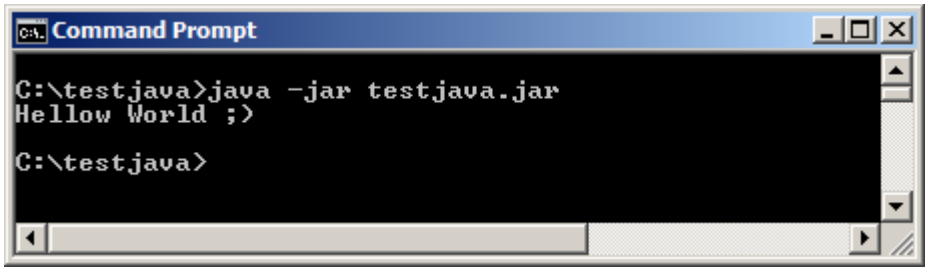
Para que el archivo jar sea ejecutable hay que incluir en el archivo MANIFEST.MF una línea indicando la clase que contiene el método estático main() que se usará para iniciar la aplicación. En nuestro ejemplo anterior sería:

```
Main-Class: HelloWorld
```

Es importante destacar que al final de la línea hay que agregar un retorno de carro para que funcione. Los invito a crear un archivo testjava.zip que contenga el archivo HelloWorld.class, el directorio META-INF y dentro el archivo MANIFEST.MF con la linea Main-Class: HelloWorld. Para esto pueden usar los programas Winzip o WinRAR que pueden descargar gratuitamente (buscar en Google).



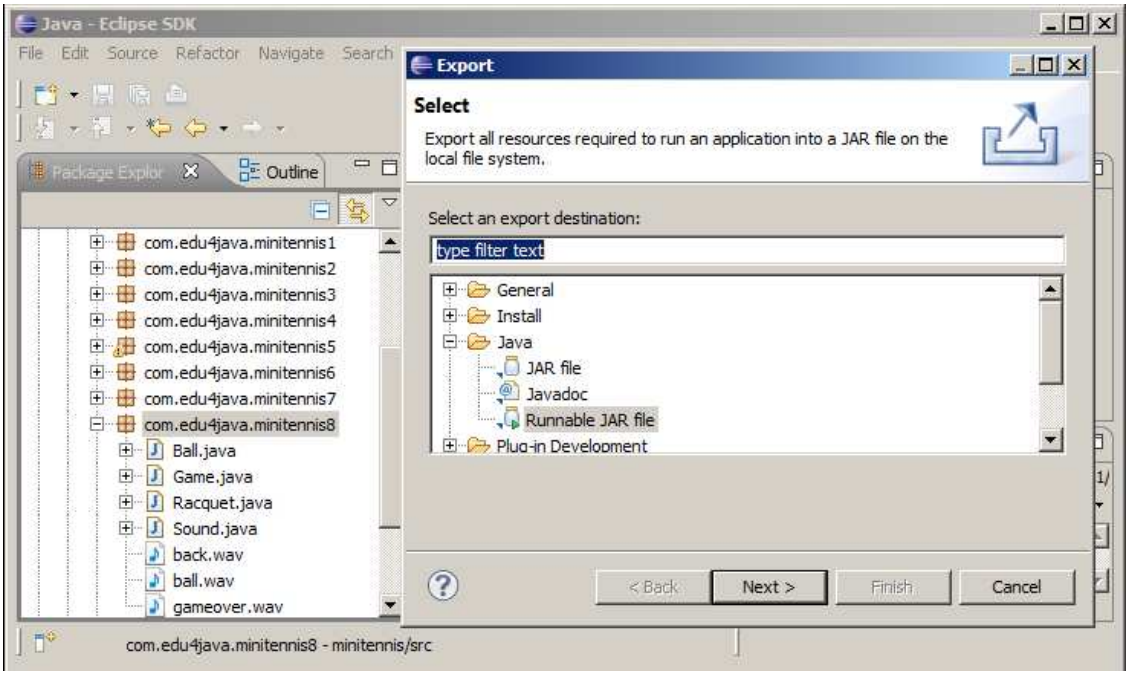
Una vez creado el archivo testjava.zip, lo renombramos a testjava.jar y lo ejecutamos desde la línea de comandos:



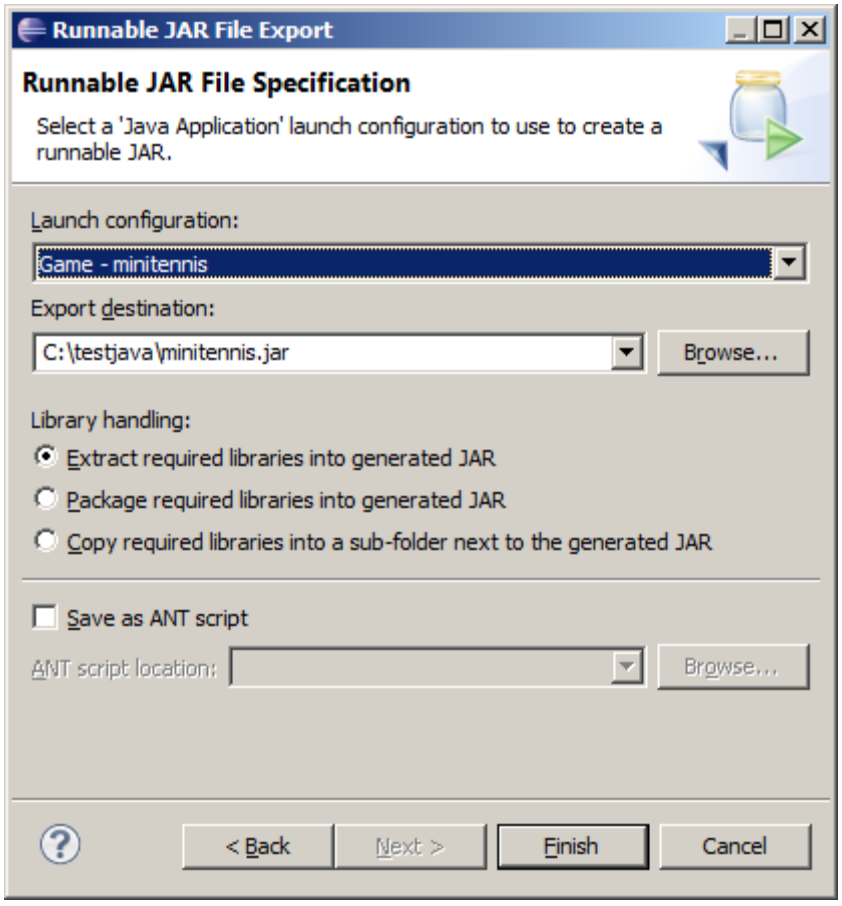
También podemos ejecutar haciendo doble click sobre el archivo JAR.

Como crear un archivo JAR ejecutable desde eclipse

Para crear un JAR ejecutable basta con ir a File-Export, seleccionar Runnable JAR file



Como se ve a continuación, en "Launch configuration" seleccionamos la que usamos para ejecutar la versión final de nuestra aplicación y en "Export destination" indicamos donde queremos guardar nuestro JAR y con qué nombre:

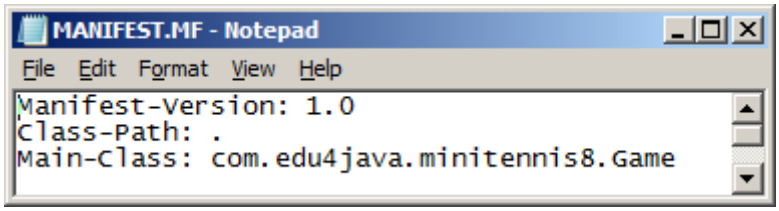


Si java está bien instalado sobre Windows, con un doble click sobre minitennis.jar sería suficiente para ejecutar nuestra aplicación.

Examinando minitennis.jar

Si descomprimimos nuestro archivo minitennis.jar encontraremos los archivos *.class que componen nuestro juego. Estos archivos están dentro del árbol de directorios con los nombres de los paquetes java que contienen a las clases.

Además dentro de META-INF/MANIFEST.MF podemos ver en la última línea como se indica que el juego debe iniciarse con el método main() de la clase Game que esta en el paquete com.edu4java.minitennis8.



Eclipse realiza un excelente trabajo compilando, ejecutando y creando archivos JAR pero es bueno entender que por debajo eclipse usa la instalación de java de forma similar a nuestro ejemplo HelloWorld.

Bueno, esto es todo ... nos vemos en el próximo tutorial ;)