Java, JavaME y el entorno Eclipse - Ejercicios

Índice

1. ¡Hola ME!

Vamos a hacer nuestra primera aplicación *"Hola mundo"* en J2ME. Para ello debemos:

a) Crear un nuevo proyecto `HolaME` con configuración CLDC1.1 y MIDP2.1. Crear en su carpeta de fuentes un MIDlet principal `es.ua.jtech.javame.holame.MIDletHolaME`.

b) Introduciremos en la clase del MIDlet principal el siguiente código:

```
package es.ua.jtech.javame.holame;

import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class MIDletHolaME extends MIDlet {
        protected void startApp() throws MIDletStateChangeException {
                Form form = new Form("¡Hola ME!");
                Display display = Display.getDisplay(this);
        display.setCurrent(form);
        }

        protected void destroyApp(boolean unconditional)
                throws MIDletStateChangeException {
        }

        protected void pauseApp() {
        }
}
```

c) Guardar el fichero y tras comprobar que Eclipse no da ningún error de compilación, ejecutar la aplicación en dos emuladores distintos, comprobando que funciona correctamente.

Busca desde Eclipse la opción del menú contextual que permite crear un paquete. Se generarán dos archivos en la carpeta `deployed`. Uno de ellos es de texto plano, el `.jar`. El otro es un paquete que puedes descomprimir como zip. Explora su estructura. ¿Encuentras un archivo con información similar al `.jar`?

2. Clases

Vamos a crear en el anterior proyecto `HolaME` un paquete llamado `es.ua.jtech.javame.holame.calculos` y en él vamos a añadir las clases `Factorial` y `EcuacionCuadratica` que vienen en las plantillas de la sesión. Para utilizar sus métodos vamos a introducir al final del método `MIDletHolaME.startApp()` dos formas de imprimir el resultado: en el formulario de la pantalla y por la salida estándar que puede verse en la consola de Eclipse.

```
        Factorial factorial = new Factorial();
        int f1 = factorial.factorialRec(4);
```

```
                System.out.println("Factorial = "+f1);
                form.append("Factorial = "+f1);
```

Añadid a la clase Factorial.java el código necesario para que calcule el factorial de un número. Intentad hacer tanto la versión recursiva como la iterativa

- La versión recursiva (en el método `factorialRec()`) consiste en un método que se llama a sí mismo hasta completar el resultado:
  `factorialRec(n) = n · factorialRec(n - 1)`
  Cuando n sea 0 se devuelve 1 y se termina la recursividad.
- La versión iterativa (en el método `factorialIter()`) consiste en realizar un bucle que vaya acumulando el resultado.

Ahora añadid a la clase `EcuacionCuadratica.java` el código necesario (dentro del método `solucion(...)`) para que resuelva una ecuación de segundo grado $ax^2 + bx + c = 0$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Raices de la ecuación

Si no hay solución, debe devolver `null`.

Nota: el método `Math.pow()` no está en JavaME. Tendremos que utilizar el producto en su lugar.

Probar `Factorial.factorialIter()`, `Factorial.factorialRec()` y `EcuacionCuadratica.solucion()` visualizando cadenas con los resultados tanto en el `Form` como por la salida estándar.

3. Métodos y campos de la clase

En JavaME no se utiliza el método main, se trata de un método que se utiliza más en aplicaciones de escritorio. Una de sus utilidades podría ser introducir algún pequeño test de que la clase funciona, o un ejemplo de uso. En el caso del ejercicio anterior podríamos introducir ejemplos de uso del Factorial o de la EcuacionCuadratica en sus respectivos métodos `main()`. Para ejecutarlos es suficiente con utilizar la opción "Run as / Java application" de Eclipse. El resultado se verá por la consola (salida estándar).

El método `EcuacionCuadratica.java` incluye dos ejemplos de uso pero uno de ellos, el del método sin parámetros, está comentado. Fíjate que ambos métodos (con parámetros y sin parámetros) se llaman exactamente igual, lo que los diferencia es el número de parámetros. Para que el método sin parámetros funcione necesitamos crear getters y setters para las variables, para ello utiliza el menú "Source / Generate getters and setters" de Eclipse. Descomenta el código y comprueba que funciona igual. Fíjate en que según el código comentado de `main()`, el getter `getRaices(i)` toma la posición del array como

parámetro. Modifica el getter o implementa otro, con esta funcionalidad.

Incluye una tercera forma de uso que, en lugar de usar los setters, utilice directamente el constructor para dar valor a los campos de la clase (las variables). Implementa dicho constructor usando la opción "Generate constructor using fields" del menú Source de Eclipse.

4. Métodos estáticos

¿Tiene sentido mantener las variables de la ecuación y las soluciones como campos de la clase? Mantener un estado del objeto `EcuacionCuadratica` del ejercicio anterior no tiene mucho sentido, ya que lo normal será que cada vez la utilicemos para resolver una ecuación diferente, y cada siguiente ecuación no depende en nada de la anterior. Así, en ambas clases del ejercicio anterior, sería suficiente con llamar a sus métodos pasándoles los parámetros necesarios y obteniendo como valor de retorno la solución.

Vamos a crear una segunda versión de las clases, duplicándolas desde Eclipse con nombres nuevos: `Factorial2` y `EcuacionCuadratica2`. Convierte en estáticos los métodos que podamos llamar sin depender de los campos de las clases. Elimina los campos de la clase `EcuacionCuadratica2` y el método que depende de ellos.

Cambia los métodos `main()` para que hagan un uso estático de los métodos. ¿Tiene ahora sentido crear objetos de las clases? Implementa un mecanismo que lo prohíba: el constructor por defecto privado. Comprueba que es imposible instanciarlas desde el MIDlet.

5. Librerías opcionales (*)

Vamos a añadir sonido a una aplicación JavaME. Para ello deberemos utilizar la API multimedia que es una API adicional. Crearemos un proyecto nuevo llamado `PruebaSonido` y le añadiremos un midlet `MIDletPruebaSonido`. A continuación deberemos:

a) Comprobar si está la librería MMAPI en nuestro proyecto en Eclipse. Se puede ver en la pestaña Libraries del Java Build Path de las preferencias del proyecto.

b) Una vez hecho esto podremos utilizar esta API multimedia en el editor de Eclipse sin que nos muestre errores en el código. Modificaremos el código del MIDlet de la siguiente forma:

```
package es.ua.jtech.javame.pruebasonido;

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

public class MIDletPruebaSonido extends MIDlet {
```

```
    protected void startApp() throws MIDletStateChangeException {
        Form f = new Form("Prueba de sonido");

        try {
            Manager.playTone(80, 1000, 100);
        } catch(MediaException e) {}

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

    protected void pauseApp() {
    }

    protected void destroyApp(boolean incondicional)
        throws MIDletStateChangeException {
    }
}
```

c) Guardar y comprobar que la aplicación funciona correctamente.

6. Temporizadores (*)

Vamos a incorporar un temporizador a una aplicación. Lo único que haremos será mostrar un mensaje de texto en la consola cuando se dispare el temporizador, por lo que no será una aplicación útil para visualizar en el móvil.

a) En el directorio `Temporizador` de las plantillas de la sesión se encuentra implementado este temporizador. Compilarlo y ejecutarlo.

b) Modificar este temporizador para que en lugar de dispararse pasado cierto intervalo, se dispare a una hora fija. Para ello puedes usar la clase `Date` y ayudarte de la clase `Calendar`.