Universidad del País Vasco
Euskal Herriko Unibertsitatea

Fundamentos de Programación:

Prácticas de Java

1º Ingeniería Técnica de Telecomunicación

Bilbao, febrero de 2009

Fundamentos de Programación: Prácticas de Java.

Copyright © 2007, 2008, 2009 Gorka Prieto Agujeta

Fundamentos de Programación: Prácticas de Java by Gorka Prieto is licensed under a Creative Commons Attribution-Share Alike 3.0 United States License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-sa/3.0/us/`; or, (b) send a letter to Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA.

Fundamentos de Programación: Prácticas de Java por Gorka Prieto está licenciado bajo una licencia Creative Commons Reconocimiento-Compartir bajo la misma licencia 2.5 España License. Para ver una copia de esta licencia, visita `http://creativecommons.org/licenses/by-sa/2.5/es/`; o, (b) manda una carta a Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA.

Resumen

Con las siguientes prácticas se pretende afianzar los conceptos de programación orientada a objetos (POO) que se han visto en las clases de teoría con Java.

Tras la realización de las prácticas, se habrá implementado en Java una aplicación que permita al usuario elegir entre diferentes juegos sencillos de consola y ejecutarlos.

Para ello se partirá de un programa sencillo y se irá incluyendo en cada práctica un concepto nuevo de los vistos en teoría comprobando la mejora que aporta sobre la versión anterior del programa.

Salvo en la primera práctica de familiarización con el entorno, en el resto de prácticas se realizará una copia del directorio con todos los ficheros de la práctica anterior y sobre esos ficheros se incluirán las nuevas funcionalidades.

Índice general

Práctica 1

Familiarización con el Entorno

> En esta primera práctica se trata de que el alumno se familiarice con el entorno de programación, tanto mediante la interfaz gráfica (Eclipse) como mediante la línea de comandos. Para ello se utilizará un programa sencillo que se compilará y ejecutará.

1.1. El "workspace"

En primer lugar creamos un directorio con nuestro nombre dentro del `home` del usuario alumno. En este directorio será donde guardemos todas las prácticas y a partir de ahora nos vamos a referir a él como `workspace`.

Se recomienda que antes de terminar cada clase, el alumno se copie su `workspace` a una memoria USB a modo de backup, ya que los ordenadores del centro de cálculo pueden ser formateados en cualquier momento a lo largo del curso y sin previo aviso.

1.2. "Hola Mundo" desde Consola

Mediante un editor de textos[1] escribir el siguiente programa de ejemplo:

```
public class HolaJava {
    public static void main( String args[] ) {
        System.out.println( "¡Hola Java!" );
    }
}
```

A continuación guardar el fichero fuente en el directorio `Practica1-1` de vuestro `workspace` con el nombre `HolaJava.java`.

[1] Si se usa un procesador de textos como OpenOffice, habrá que asegurarse de guardar el fichero en modo texto en lugar de en formato binario.

El siguiente paso consiste en compilar este programa de ejemplo. Para ello abrimos la consola y nos situamos en el directorio que contiene el código fuente y que será el directorio `Practica1-1` dentro del `workspace` que hemos creado:

```
$ cd tu_nombre
$ cd Practica1-1
```

Podemos comprobar los ficheros disponibles en este directorio con el comando:

```
$ ls
```

Desde este directorio invocamos el compilador de java pasándole como parámetro el fichero que queremos compilar:

```
$ javac HolaJava.java
```

Si volvemos a hacer un `ls` comprobaremos que se nos ha generado un fichero `HolaJava.class`. Este fichero contiene el bytecode de java que podrá ser interpretado por una máquina virtual java. Para invocarlo, escribimos lo siguiente:

```
$ java HolaJava
```

Como se puede observar, el parámetro que se le pasa no es el nombre del fichero java, sino que es el nombre de la clase que contiene el método `main`. El resultado de invocar este programa será que se muestre por pantalla:

```
¡Hola Java!
```

1.3. "Hola Mundo" desde Eclipse

A continuación realizamos un proceso equivalente desde Eclipse.

En primer lugar debemos indicar a Eclipse cuál es nuestro `workspace`. Esto se hace con la opción `File/Switch Workspace ...`, y le indicamos que es el directorio `/home/alumno/tu_nombre`. Es importante hacer esto al principio de cada práctica ya que puede que otra persona de otro grupo haya cambiado Eclipse a su propio `workspace`.

Ya podemos crear un nuevo proyecto con `File/New Project`. Indicamos que deseamos crear un `Java Project` y le damos el nombre `Practica1-2` al proyecto. Eclipse guarda los proyectos en directorios separados dentro del `workspace`.

Desde el árbol del proyecto, hacemos click con el botón derecho y damos a la opción de añadir una nueva clase. Llamamos `HolaJava` a esta nueva clase y a continuación vemos que automáticamente se crea un fichero fuente sobre el que podemos

escribir el fragmento de código mostrado anteriormente. Escribimos el código y guardamos el fichero.

Finalmente lo compilamos y ejecutamos sin más que ir al menú `Run` y dar a la opción `Run As/Java Application`. El resultado de la ejecución del programa se nos mostrará en la ventana `Console` de Eclipse.

Para el resto de las prácticas el alumno podrá usar el entorno de desarrollo que más cómodo le resulte.

1.4. Lectura de Teclado

Para los programas que se van a realizar en el resto de prácticas, resulta necesario el uso de la entrada por teclado. Como estos conceptos no se han explicado en la teoría, usaremos el siguiente código creado por el profesor y que proporciona los métodos `Teclado.LeeCaracter()`, `Teclado.LeeEntero()` y `Teclado.LeeCadena()`. Estos métodos devuelven un `char`, un `int` o un `String` respectivamente con la entrada introducida por el usuario en el teclado.

Copiar el siguiente código en un fichero `Teclado.java` dentro de un nuevo directorio `Practica1-3`:

```
import java.io.*;

public class Teclado {
    public static char LeeCaracter() {
        char ch;

        try {
            ch = LeeCadena().charAt(0);
        } catch( Exception e ) {
            ch = 0;
        }

        return ch;
    }

    public static String LeeCadena() {
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        String str;

        try {
            str = br.readLine();
        } catch( Exception e ) {
            str = "";
```

```
        }

        return str;
    }

    public static int LeeEntero() {
        int num;

        try {
            num = Integer.parseInt( LeeCadena().trim() );
        } catch( Exception e ) {
            num = 0;
        }

        return num;
    }
}
```

Para probar el funcionamiento del teclado, copiar el siguiente código en un fichero `HolaTeclado.java` dentro del mismo directorio y probar a compilar ambos ficheros desde la consola y desde un nuevo proyecto de Eclipse.

```
1   public class HolaTeclado {
2       public static void main( String args[] ) {
3           String nombre;
4           int edad;
5
6           System.out.print( "Dime tu nombre: " );
7           nombre = Teclado.LeeCadena();
8
9           System.out.print( "Dime tu edad: " );
10          edad = Teclado.LeeEntero();
11
12          System.out.println( "Hola " + nombre +
13                              ", tienes " + edad + " años" );
14      }
15  }
```

1.5. Depuración de Aplicaciones

Eclipse permite depurar aplicaciones de una forma muy cómoda desde la interfaz gráfica. Para que funcione la depuración de código java en Eclipse, debemos asegurarnos de que se esté usando un JRE que soporte la depuración de aplicaciones. Para ello en `Window/Preferences/Java/Installed JREs` nos aseguramos de que

esté presente y activado el JRE de SUN. Si no está presente, lo añadimos indicando el siguiente path: `/usr/lib/jvm/java-6-sun-1.6.0.03/jre`.

A continuación vamos a poner dos breakpoints, uno en la línea 6 del fichero `Teclado.java` y otro en la línea 9. Para poner un breakpoint basta con hacer click con el derecho en el borde izquierdo de la línea y dar a `Toggle Breakpoint`, o bien hacer doble click con el izquierdo en el borde de la línea. Lo mismo para quitar un breakpoint puesto previamente.

Una vez puestos los breakpoints, depuramos el programa sin más que ir al menú `Run` y dar a la opción `Debug As/Java Application`. El programa se detendrá cuando llegue a la línea 6. Si pulsamos la tecla F6, se ejecutará esa instrucción y se quedará detenido en la siguiente. Si a continuación pulsamos F8, el programa se continua ejecutando hasta llegar al siguiente breakpoint.

Repetir el procedimiento anterior pero pulsando F5 en lugar de F6 y comentar las diferencias observadas.

Práctica 2

Clases y Objetos

En esta segunda práctica comenzaremos ya con el programa que iremos ampliando a lo largo del resto de prácticas. El objetivo de esta práctica es que el alumno practique y se familiarice con los conceptos de clase y objeto.

2.1. Clase Juego

Implementar una clase `Juego` con las siguientes características:

- Atributos
  - Tiene como atributo público un entero que indica el número de vidas que le quedan al jugador en la partida actual.
- Métodos
  - Tiene como método el constructor que acepta un parámetro de tipo entero que indica el número de vidas iniciales con las que parte el jugador.
  - Tiene un método `MuestraVidasRestantes` que visualiza por pantalla el número de vidas que le quedan al jugador en la partida actual.
  - Además esta clase tiene también el método `main` que debe realizar lo siguiente:
    - Crea una instancia de la clase `Juego` indicando que el número de vidas es 5.
    - Llama al método `MuestraVidasRestantes` del objeto creado.
    - Resta una vida al valor del atributo con las vidas y vuelve a llamar a `MuestraVidasRestantes`.
    - Crea otra instancia de la clase `Juego` indicando que el número de vidas es también de 5.
    - Llama al método `MuestraVidasRestantes` de la nueva instancia y luego al de la instancia anterior

2.2. Ocultación de Atributos

En el ejercicio anterior no se ha prestado atención a la forma en que se permite que alguien modifique el atributo con las vidas de la clase `Juego`. En este ejercicio se utilizará un acceso controlado a ese atributo. Para ello se realizarán los siguientes cambios:

- Atributos
  - Debe ocultarse a cualquier otra clase el atributo con las vidas. Para poder modificar este atributo, se crearán los dos nuevos métodos que se explican más adelante.
  - Crear un nuevo atributo también privado que guarde el número de vidas que inicialmente se le pasaron al constructor del objeto. Este atributo se utilizará para poder reiniciar el juego.
  - Crear otro atributo también privado y de tipo entero que guarde el récord. A diferencia de los anteriores (que son atributos de instancia) éste es un atributo de clase, por lo que será común a todos los juegos que se implementen. Inicialmente este atributo tendrá el valor 0.
- Métodos
  - Añadir un método `QuitaVida` que disminuya en 1 el número de vidas del jugador y devuelva un `boolean` indicando si al jugador le quedan más vidas o no. En caso de que al jugador no le queden más vidas, este método debe mostrar un mensaje `Juego Terminado` por pantalla.
  - Añadir un método `ReiniciaPartida` que asigne al atributo vidas el número de vidas que se habían indicado al llamar al constructor del objeto. Para ello utilizará el nuevo atributo que se ha añadido.
  - Añadir un método `ActualizaRecord` que compare el valor actual de récord con el número de vidas restantes.
    - Si el número de vidas restantes es igual al récord, mostrará un mensaje indicando que se ha alcanzado el récord.
    - Si el número de vidas restante es mayor que el récord, actualizará el récord y mostrará un mensaje diciendo que éste se ha batido y cuál es su nuevo valor.
    - Si el número de vidas es menor, no hará nada.

Para probar la ocultación, la función `main` se va a poner ahora en una clase aparte llamada `Aplicacion` en un fichero `Aplicacion.java` dentro del mismo directorio:

- Antes de modificar esta función, comprobar que ahora el compilador nos muestra un mensaje de error al intentar modificar directamente el atributo con las vidas. A continuación proceder con las modificaciones que siguen.

- Llamar al método `QuitaVida` de una de las instancias de la clase `Juego` a continuación al método `MuestraVidasRestantes`.
- Posteriormente llamar al método `ReiniciaPartida` y de nuevo al método `MuestraVidasRestantes`.
- Llamar al método `ActualizaRecord` de la primera instancia de `Juego` y a continuación llamar a este mismo método pero en la segunda instancia. Explica los mensajes mostrados.

Práctica 3

Herencia y Polimorfismo

En esta práctica se introducen los conceptos de herencia y polimorfismo. La herencia permite a nuevas clases aprovechar código ya implementado por clases anteriores. El polimorfismo permite llamar a un método con diferentes resultados según la clase en la que se esté. Además se irá dando ya un poco de forma a la aplicación final.

3.1. Ejercicio 1

En este ejercicio se va a implementar un juego en el que el usuario[1] tenga que adivinar un número que conoce el programa. El código correspondiente a cada clase que se implemente deberá estar en un fichero java separado y que tenga el mismo nombre que la clase.

- Clase `Juego`
  - Añadirle un método abstracto `Juega` que no tome parámetros y que tendrán que implementar las clases derivadas.
  - La clase `Juego` ahora pasa a ser una clase abstracta por lo que ya no se podrán crear instancias de la misma.
  - La función `main` ya no estará dentro de esta clase.
- Clase `JuegoAdivinaNumero`
  - Deriva de la clase `Juego`.
  - Tiene un constructor que toma dos parámetros de tipo entero. El primero es el número de vidas que, a su vez, se lo pasará al constructor de la clase base. El segundo parámetro es un número a adivinar entre 0 y 10.
  - Implementa el método `Juega` de la clase base:

[1] Usando la clase de entrada por teclado copiada en el primera práctica, se leerá un entero del teclado

    - Llama al método `ReiniciaPartida` que ha heredado.
    - Muestra un mensaje al usuario pidiendo que adivine un número entre el 0 y el 10.
    - Lee un entero del teclado y lo compara con el valor predefinido por el programador:
        - Si es igual, muestra un mensaje `Acertaste!!` y, tras llamar a `ActualizaRecord`, sale del método.
        - Si es diferente, llama al método `QuitaVida` heredado.
        - Si el método `QuitaVida` devuelve `true`, significa que aún le quedan más vidas al jugador por lo que se muestra un mensaje indicando si el número a adivinar es mayor o menor y se le pide que lo intente de nuevo.
        - Si el método `QuitaVida` devuelve `false` significa que ya no le quedan más vidas al jugador, con lo que sale del método `Juega`.

- Clase `Aplicacion`

    - Contiene un método main que, tras crear una instancia de la nueva clase `JuegoAdivinaNumero` que se ha creado, llama al método `Juega`.

3.2. Ejercicio 2

A partir del juego anterior, se añadirán dos juegos más, uno de adivinación de números pares y otro de adivinación de números impares.

- Clase `JuegoAdivinaNumero`

    - Añade un nuevo método `ValidaNumero` que toma como parámetro el número introducido por el usuario y devuelve un `boolean` que, en este caso, será siempre `true`.
    - En el método `Juega` pedirá un nuevo número por teclado si el método `ValidaNumero` devuelve `false` y, en este caso, no restará vida.

- Clase `JuegoAdivinaPar`

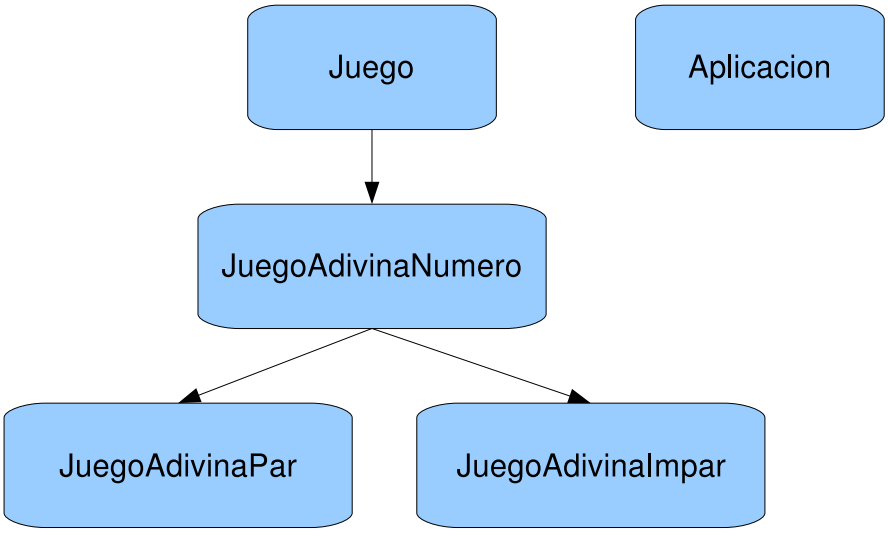
    - Deriva de la clase `JuegoAdivinaNumero`
    - Redefine el método `ValidaNumero` devolviendo `true` si el número es par. Si el número es impar, muestra un mensaje de error por pantalla y devuelve `false`.

- Clase `JuegoAdivinaImpar`

    - Deriva de la clase `JuegoAdivinaNumero`
    - Redefine el método `ValidaNumero` devolviendo `true` si el número es impar. Si el número es par, muestra un mensaje de error por pantalla y devuelve `false`.

- Clase `Aplicacion`
  - El método main crea una instancia de cada uno de los tres juegos creados: `JuegoAdivinaNumero`, `JuegoAdivinaPar` y `JuegoAdivinaImpar`. Como número de vidas de cada juego se pondrá 3 y como número a adivinar un número cualquiera, otro par y otro impar respectivamente, todos comprendidos entre el 0 y el 10.
  - Llama al método `Juega` de cada una de las tres instancias.

La jerarquía de clases resultante es la siguiente:



Práctica 4

Interfaces y Arrays

En esta práctica se hará uso del concepto de interfaz y se creará un array de interfaces que permita utilizar objetos de clases diferentes de una forma homogénea (polimorfismo).

4.1. Interfaces

En este ejercicio se va a implementar una interfaz `Jugable` que implementarán los juegos desarrollados hasta ahora y nuevos que se desarrollen. Esta interfaz nos permitirá especificar una serie de operaciones comunes que deben implementar todos los juegos y que nos permitirán manejarlos de forma genérica posteriormente.

- Clase `Juego`
  - Se eliminará su método abstracto `Juega`, pero la clase se seguirá manteniendo como abstracta ya que no interesa que se creen instancias de ellas directamente.
- Interfaz `Jugable`
  - Dispondrá de un método `Juega` que cumplirá el mismo objetivo que el que se ha quitado a la clase `Juego`.
  - Se incorporará un método `MuestraNombre` que no tome ningún parámetro y que obligue a las clases que implementen la interfaz a mostrar un mensaje por pantalla con el nombre del juego.
  - Se incorporará un método `MuestraInfo` que no tome ningún parámetro y que obligue a las clases que implementen la interfaz a mostrar un mensaje por pantalla con una descripción de cómo jugar al juego.
- Clase `JuegoAdivinaNumero`
  - Debe implementar la interfaz `Jugable`

    - El método `MuestraNombre` visualizará por pantalla el texto `Adivina un número`
    - El método `MuestraInfo` visualizará por pantalla una descripción de cómo se juega al juego, informando del número de intentos que se le dan al jugador.

- Clase `JuegoAdivinaPar`
    - Redefine el método `MuestraNombre` para que visualice por pantalla el texto `Adivina un número par`
    - Redefine el método `MuestraInfo`

- Clase `JuegoAdivinaImpar`
    - Redefine el método `MuestraNombre` para que visualice por pantalla el texto `Adivina un número impar`
    - Redefine el método `MuestraInfo`

- Clase `Aplicacion`
    - En el método `main` creará un objeto de cada uno de los juegos mencionados.
    - A continuación llama los métodos `MuestraNombre`, `MuestraInfo` y `Juega` de cada uno de los tres objetos creados.

4.2. Arrays

Para comprender la utilidad de las interfaces, implementamos en este ejercicio un array de interfaces que permitirá invocar a cualquiera de los tres juegos de forma genérica.

- Clase `Aplicacion`
    - Método `EligeJuego`
        - Método público y estático que no toma parámetros y devuelve un objeto del tipo `Jugable`.
        - Crea un objeto de cada uno de los tres juegos implementados.
        - Crea un array de tres elementos de tipo interfaz `Jugable`.
        - Rellena este array con los objetos creados para los distintos juegos. A partir de este momento, sólo se trabajará con este array de tipo interfaz `Jugable` para referirse a cualquiera de los juegos.
        - Muestra un menú por pantalla con el nombre de los tres juegos y pide al usuario que elija un juego introduciendo un número entre 0 y 2. Si el número introducido no es válido, seguirá pidiendo al usuario un número válido.

    - Devuelve el elemento del array correspondiente al número introducido por el usuario.

- Método `main`
    - Llama al método `EligeJuego` para obtener una referencia de tipo interfaz `Jugable` al juego seleccionado por el usuario.
    - Llama al método `MuestraNombre` de este juego.
    - A continuación llama al método `MuestraInfo` del juego.
    - Llama al método `Juega` del mismo para comenzar una nueva partida.
    - Finalmente, tras concluir la partida, pregunta al usuario si desea jugar de nuevo y en caso afirmativo vuelve a repetir los pasos anteriores.

Práctica 5

Paquetes

El número de clases y ficheros usados en la práctica ya va creciendo, por lo que vamos a estructurarlos en directorios y paquetes.

5.1. Paquetes

Se trata de organizar en diferentes paquetes las clases empleadas hasta ahora y de utilizarlos desde los diferentes ficheros de código.

- Paquete `juegos`, a su vez formado por:
  - Clase `Juego`
  - Paquete `interfaces`
    - Contiene la interfaz `Jugable`
  - Paquete `numeros`
    - Contiene las clases `JuegoAdivinaNumero`, `JuegoAdivinaPar` y `JuegoAdivinaImpar`
- Paquete `profesor`
  - Contiene la clase `Teclado`

Tener en cuenta que al estructurar el código en paquetes diferentes, el modificador por defecto `friendly` va a impedir que un paquete use código de otro salvo que éste lo ofrezca como público. Por lo tanto se deberán actualizar los modificadores de acceso de clases, métodos y atributos que se consideren necesarios. De la misma forma, al estar el código ahora en paquetes diferentes, será necesario hacer los `import` correspondientes.

Para crear un nuevo paquete desde Eclipse, basta con hacer click con el botón derecho sobre el proyecto y dar a `New/Package`. A continuación pedirá el nombre

del paquete y, tras introducirlo, creará un directorio con el nombre del paquete que nos aparecerá en el árbol del proyecto. Ahora ya podemos hacer click con el derecho sobre este paquete e ir añadiendo clases. Eclipse se encarga de poner por nosotros las directivas `package` necesarias en los ficheros fuente de las clases que vayamos añadiendo a ese paquete.

5.2. Classpath

5.2.1. Desde la Consola

Mover todos los ficheros salvo `Aplicacion.java` a un directorio `libs` en vuestro `workspace`:

```
$ mkdir -p /home/alumno/tu_nombre/libs
$ cd /home/alumno/tu_nombre/Practica5
$ mv * /home/alumno/tu_nombre/libs
$ mv /home/alumno/tu_nombre/libs/Ap* .
```

Si a continuación se ejecuta el programa, dará un error indicando que no encuentra clases. Solucionarlo añadiendo al classpath este directorio.

```
$ java Aplicacion
$ java -cp .:/home/alumno/tu_nombre/libs Aplicacion
```

Finalmente volver a poner los ficheros en su ubicación original.

```
$ mv /home/alumno/tu_nombre/libs/* .
$ rm -rf /home/alumno/tu_nombre/libs
```

5.2.2. Desde Eclipse

En primer lugar vamos a crear un fichero `JAR` con el paquete profesor. El fichero `JAR` no es más que un conjunto de ficheros `*.class` comprimidos.

1. Hacemos click con el derecho sobre el paquete y damos a `Export...`.
2. Seleccionamos `Java/JAR file` y damos a siguiente.
3. A continuación seleccionamos los ficheros java que nos interese meter en el paquete, en nuestro caso sólo `Teclado.java`.
4. En esta misma pantalla indicamos que el nombre del fichero que queremos generar es `profesor.jar` y damos a terminar.

Si abrimos el explorador de ficheros, veremos que se nos ha generado un nuevo fichero `profesor.jar` en el directorio de nuestro `workspace`.

Ahora borramos el paquete `profesor` de nuestro árbol de proyecto haciendo click con el derecho sobre él y dando a `Delete`. Al borrarlo se puede ver que Eclipse muestra varios errores en el resto de nuestro código, esto es debido a que no encuentra el paquete `profesor`.

Para solucionarlo vamos a incluir en nuestro `classpath` el fichero `JAR` que acabamos de generar:

1. Hacemos click con el derecho sobre el proyecto y damos a `Build Path/Add External Archives...`.
2. Seleccionamos el fichero `profesor.jar` y damos a aceptar.

Ahora comprobamos cómo desaparecen los errores de Eclipse y que podemos volver a ejecutar la aplicación sin problema. Esto es debido a que con el nuevo `classpath` ya se pueden encontrar todos los paquetes necesarios.

Práctica 6

El API

En esta práctica se trata de que el alumno sea capaz de leer documentación referente a un paquete externo y de utilizarlo. Para ello se utilizarán las clases `Random`, `String` y `Vector` del API de java[1].

6.1. Clase Random

Consultando la documentación del API de java, usar la clase `Random` del paquete `java.util` para que el número que hay que adivinar en los juegos de números sea un número aleatorio en lugar de un número predefinido por el programador. Para ello:

- Clase `JuegoAdivinaNumero`
  - Ahora el constructor ya no necesitará como parámetro el número a adivinar.
  - Añade como dato miembro un objeto de tipo `Random` que se usará para generar números aleatorios. A la hora de construir este objeto, es conveniente pasarle una semilla que evita que se genere siempre la misma secuencia de números pseudoaleatorios. Para ello puede usarse la clase `Date` de paquete `java.util`.
  - Redefine el método `ReiniciaPartida` para que, además de ejecutar el código definido en la clase `Juego`, asigne un valor aleatorio al dato miembro que contiene el número a adivinar.
- Clase `AdivinaNumeroPar`
  - Redefine el método `ReiniciaPartida` para que ejecute el código definido en la clase `JuegoAdivinaNumero` y además transforme el número aleatorio generado por ésta en un número par entre 0 y 10.

[1] `http://java.sun.com/javase/reference/api.jsp`

- Clase `AdivinaNumeroImpar`
  - Redefine el método `ReiniciaPartida` para que ejecute el código definido en la clase `JuegoAdivinaNumero` y además transforme el número aleatorio generado por ésta en un número impar entre 0 y 10.

6.2. Clase String

Haciendo uso de la clase `String` se va a implementar un nuevo juego `JuegoAhorcado` esta vez no basado en números.

- Clase `JuegoAhorcado`
  - Estará en el paquete `juegos.letras`.
  - Derivará de la clase `Juego`.
  - Tomará comoprimer parámetro del constructor el número de vidas y como segundo parámetro la cadena a adivinar.
  - Implementará la interfaz `Jugable`.
  - Para implementar el método `Juega` se recomienda seguir los siguientes pasos:
    - Llamar al método `ReiniciaPartida` de la clase base.
    - Crear una cadena del mismo tamaño que la cadena a adivinar pero en la que todos sus caracteres sean un '-'.
    - Mostrar al usuario la cadena con los '-'.
    - Pedir al usuario que introduzca un carácter y comprobar si está en la cadena a adivinar.
    - Si está en la cadena, reemplazar los '-' por el carácter en las posiciones que corresponda. Comparar esta cadena con la cadena a adivinar y, si son iguales, indicárselo al usuario y terminar la partida.
    - Si no está en la cadena, llamar al método `QuitaVida` comprobando si se ha terminado la partida o no. Si no se ha terminado la partida, volver a mostrar la cadena con '-' al usuario y repetir el proceso.
- Clase `Aplicacion`
  - Añadir al método `EligeJuego` el nuevo juego que se acaba de crear.

6.3. Clase Vector

Para practicar con la clase `Vector` del paquete `java.util`, vamos a substituir el array de interfaces de la clase `Aplicacion` por un vector.

- Clase `Aplicacion`
  - Añadir un método `InfoVector` que tome como parámetro un vector e indique por pantalla la capacidad y tamaño del mismo.
  - En el método `EligeJuego` crear un vector de capacidad 3 y con un incremento en saltos de 2 elementos. Llamar al método `InfoVector`.
  - Añadir los tres juegos de números a este vector y volver a llamar al método `InfoVector`.
  - A continuación añadir al vector el objeto correspondiente al juego del ahorcado y volver a llamar al método `InfoVector`.
  - Modificar el resto de la función para que trabaje con el vector en lugar de con el array de interfaces.

Práctica 7

Excepciones

En esta última práctica se incluirá el manejo de excepciones para validar los datos introducidos por el usuario.

7.1. Ejercicio 1

El método `LeeEntero` de la clase `Teclado` devuelve un 0 en caso de que el dato que se introduzca no sea un número. Modificar este método para que si se recibe una excepción del tipo `NumberFormatException`, se le indique al usuario que debe introducir un número válido y se vuelva a leer la entrada. En caso de recibir cualquier otra excepción, sigue devolviendo un 0.

7.2. Ejercicio 2

El constructor de las clases no puede devolver ningún código de retorno para indicar si ha funcionado bien o no. Una opción es que genere una excepción en caso de fallo.

- Clase `JuegoException`
  - Estará en el paquete `juegos.excepciones`.
  - Extiende la clase `Exception`.
  - Su constructor toma como parámetro una cadena de caracteres con la descripción del motivo de la excepción.
- Constructor de la clase `JuegoAhorcado`
  - Debe comprobar que ninguno de los caracteres de la palabra a adivinar sea un número, para ello puede valerse de los métodos de la clase `Character`. Si hay un número, lanzará una excepción `JuegoException` notificándolo.

- Clase `Aplicacion`
  - El método `EligeJuego` no captura la excepción y la pasa hacia arriba.
  - El método `main` debe capturar cualquier excepción e informar de la causa de fallo antes de terminar.
  - Tanto en caso de que ocurra una excepción como de que no ocurra ninguna, el programa debe terminar mostrando el mensaje `Fin del programa`.
  - Probar a compilar y ejecutar el programa empleando como palabra a adivinar una con número y otra sin ningún número.