Resumen

Con esta práctica 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 la práctica, 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 parte un concepto nuevo de los vistos en teoría comprobando la mejora que aporta sobre la versión anterior del programa.

Para la realización de la práctica es necesario utilizar la clase presente en el jar de nombre `librería.jar`. En esta clase se encuentran los métodos necesarios para recoger valores introducidos por el usuario desde consola.

1. Clases y Objetos

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

1.1. 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

1.2. 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`:

- 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.

2. Herencia y Polimorfismo

En esta parte 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.

2.1. Ejercicio 1

En este ejercicio se va a implementar un juego en el que el usuario 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:
    - 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`.

2.2 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.

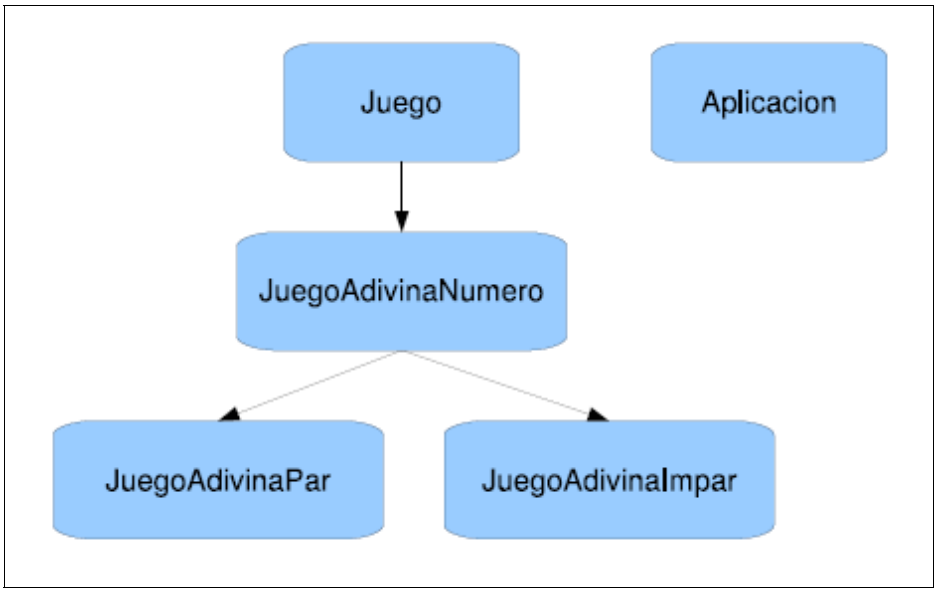
- Clase `JuegoAdivinaNumero`
  - Añade un nuevo método `ValidaNumero` que toma como parámetro el número introducido por el usuario y devuelve un `boolean` que, en este caso, será siempre true.
  - En el método `Juega` pedirá un nuevo número por teclado si el método `ValidaNumero` devuelve false y, en 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:



3. Interfaces y Arrays

En esta parte 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).

3.1. 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.

3.2. 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 1 y 3. 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.

4 Paquetes

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

4.1. 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`
- Paquete `aplicación`:
  - Contiene la clase `Aplicacion`

5 El API

En esta parte 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.

5.1. 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.

- 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.

5.2. Clase String y StringBuffer

Haciendo uso de la clase `String` o `StringBuffer` 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 como primer 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.

A continuación se muestra lo que debería salir por la consola al ejecutar el programa:

```
Elige un juego:
1.Adivina un numero entre 0 y 10
2.Adivina un numero par entre 0 y 10
3.Adivina un numero impar entre 0 y 10
4.Ahorcado
Opcion:1
Adivina un numero entre 0 y 10
Info. del juego: Tienes varias oportunidades
para acertar un numero comprendido entre 0 y 10
Introduce numero:3
Vidas restantes:2
El numero a adivinar es menor...Introduce numero:1
Vidas restantes:1
El numero a adivinar es menor...Introduce numero:0
Acertaste!!Se ha batido el record. El record esta ahora en 1 vidas
Quieres seguir jugando (s/n)?s

Elige un juego:
1.Adivina un numero entre 0 y 10
2.Adivina un numero par entre 0 y 10
3.Adivina un numero impar entre 0 y 10
4.Ahorcado
Opcion:3
Adivina un numero impar entre 0 y 10
Info. del juego: Tienes varias opotunidades
para acertar un numero impar comprendido entre 0 y 10
Introduce numero:5
Acertaste!!Se ha batido el record. El record esta ahora en 3 vidas
Quieres seguir jugando (s/n)?s

Elige un juego:
1.Adivina un numero entre 0 y 10
2.Adivina un numero par entre 0 y 10
3.Adivina un numero impar entre 0 y 10
4.Ahorcado
Opcion:4
Juego del ahorcado
Info. del juego: Tienes varias opotunidades
para acertar una palabra

Palabra a adivinar:----
Introduce caracter:c
La letra c esta en la palabraPalabra a adivinar:c---
Introduce caracter:a
La letra a esta en la palabraPalabra a adivinar:ca-a
Introduce caracter:r
La letra r NO esta en la palabraVidas restantes:2
Palabra a adivinar:ca-a
Introduce caracter:s
La letra s esta en la palabraHas acertado!!Palabra adivinada:casa
Quieres seguir jugando (s/n)?n
```

5.3. Clase Vector

Para practicar con la clase `Vector` del paquete `java.util`, vamos a sustituir 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.

6 EXCEPCIONES

Vamos a incluir el manejo de excepciones para validar los datos introducidos por el usuario.

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` noticándolo.

- 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.