Práctica 3: **Excepciones y lectura de fichero**

Programación II

Mayo 2021 (versión 23/04/2021)

DLSI – Universidad de Alicante

Alicia Garrido Alenda

Normas generales

- El plazo de entrega para esta práctica se abrirá el lunes 17 de mayo a las 9:00 horas y se **cerrará el viernes 21 de mayo a las 23:59 horas**. No se admitirán entregas fuera de plazo.
- Se debe entregar la práctica en un fichero comprimido de la siguiente manera:

  1. Abrir un terminal.
  2. Situarse en el directorio donde se encuentran los ficheros fuente (`.java`) de manera que al ejecutar `ls` se muestren los ficheros que hemos creado para la práctica y ningún directorio.
  3. Ejecutar:

     ```
     tar cfvz  practica3.tgz   *.java
     ```

Normas generales comunes a todas las prácticas

1. La práctica se debe entregar exclusivamente a través del servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos, al que se puede acceder desde la página principal del departamento (`http://www.dlsi.ua.es`, enlace "Entrega de prácticas") o directamente en `http://pracdlsi.dlsi.ua.es`.

   - **Bajo ningún concepto** se admitirán entregas de prácticas por otros medios (correo electrónico, Campus Virtual, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en el Campus Virtual.
   - La práctica se puede entregar varias veces (recomendable), pero sólo se corregirá la última entrega.

2. El programa debe poder ser compilado sin errores con el compilador de java existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.

3. La práctica debe ser un trabajo original de la persona que entrega; **en caso de detectarse indicios de copia de una o más prácticas (o compartición de código) la calificación de la práctica será `0`** y se enviará un informe al respecto tanto a la dirección del departamento como de la titulación.

4. Se recomienda que los ficheros fuente estén adecuadamente documentados, con comentarios donde se considere necesario.

5. La primera corrección de la práctica se realizará de forma automática, por lo que es imprescindible respetar estrictamente los formatos de salida que se indican en este enunciado.

6. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

7. Para evitar errores de compilación debido a codificación de caracteres que **descuenta `0.5` de la nota de la práctica**, se debe seguir una de estas dos opciones:

   a) Utilizar EXCLUSIVAMENTE caracteres ASCII (el alfabeto inglés) en vuestros ficheros, **incluídos los comentarios**. Es decir, no poner acentos, ni ñ, ni ç, etcétera.

   b) Entrar en el menú de Eclipse `Edit > Set Encoding`, aparecerá una ventana en la que hay que seleccionar `UTF-8` como codificación para los caracteres.

8. El tiempo estimado por el corrector para ejecutar cada prueba debe ser suficiente para que finalice su ejecución correctamente, en otro caso se invoca un proceso que obliga a la finalización de la ejecución de esa prueba y se considera que dicha prueba no funciona correctamente.

Descripción del problema

El objetivo de esta práctica es asentar los conocimientos propios del paradigma de programación orientado a objetos relativos al uso de excepciones y el acceso a ficheros de texto para su lectura. Para ello en esta práctica se pide implementar una serie de clases, algunas de cuales heredan de la clase **Exception** de java, para comprender de forma práctica como se crean, lanzan y capturan excepciones.

Se trata de implementar un juego en el que hay ir introduciendo fichas en una matriz hasta conseguir un cierto número de fichas consecutivas del mismo color. Las fichas pueden estar en horizontal, vertical o diagonal. Para ello serán necesarias las clases **Ubicacion**, **Ficha** y **Consecutivas**. Además también serán necesarias las clases **DatoNoValido**, **WinWin** y **UbicacionNoValida** que hereden de la clase *Exception* de java.

En esta práctica también realizaremos lectura de ficheros de texto y la implementación de una clase que contenga una aplicación. Esta aplicación tiene que utilizar la información leída de fichero para desarrollar una partida del juego utilizando objetos de las clases comentadas anteriormente.

Pasemos a una descripción más detallada de estos elementos:

La **Ubicacion** de una ficha en un tablero viene determinada por la fila y la columna que ocupa dicha ficha en la matriz. Sus variables de instancia son:

- *fila*: número que indica la fila (`int`);
- *columna*: número que indica la columna (`int`).

Y los siguientes métodos:

- constructor: se le pasan por parámetro dos enteros; el primero indica el número de fila mientras que el segundo indica el número de columna.
- *getFila*: devuelve el valor de *fila*.
- *getColumna*: devuelve el valor de *columna*.

Las variables de instancia de una **Ficha** son:

- *color*: cadena que indica el color de la ficha (`String`);
- *casilla*: lugar que ocupa la ficha en el tablero una vez colocada (*Ubicacion*).

Y sus métodos son:

- constructor: se le pasa por parámetro una cadena que indica el color de la ficha que se crea. Si dicha cadena es `null` o "" se lanza y propaga la excepción **DatoNoValido** que tendrá como mensaje asociado la cadena "`ficha sin color`". Inicialmente una ficha no está colocada en ningún tablero, por tanto no tiene una *casilla* asignada.
- *colocaFicha*: se le pasa por parámetro una *Ubicacion*, de manera que si la ficha no tenía una casilla asignada, actualiza su casilla con la posición que se le pasa por parámetro y devuelve cierto. En cualquier otro caso devuelve falso.

- ⊙ *getFila*: devuelve la fila de su *casilla* si tiene. En otro caso devuelve `-1`.
- ⊙ *getColumna*: devuelve la columna de su *casilla* si tiene. En otro caso devuelve `-1`.
- ⊙ *getColor*: devuelve el *color* de la ficha.
- ⊙ *resetea*: elimina de la ficha la información sobre su *casilla*.

Las variables de instancia de **Consecutivas** son:

- ⁕ *tablero*: matriz donde se colocan las fichas (`Ficha[][]`);
- ⁕ *nconsecutivas*: número de fichas del mismo color que tiene que haber consecutivas para ganar el juego (`int`);
- ⁕ *colores*: array que contiene las dos cadenas correspondientes a los colores con los que se juega (`String[]`);

Y sus métodos:

- ⊙ constructor: se le pasan por parámetro 3 enteros y dos cadenas. El primer entero se corresponde con el número de filas, el segundo con el número de columnas que tendrá el *tablero* y el tercero con el número de fichas consecutivas del mismo color que hay que conseguir, que tiene que ser menor o igual que los dos primeros y mayor que `2`. Las dos cadenas se corresponden con los colores con los que se va a jugar, que tienen que ser diferentes. Si alguno de los parámetros tiene un valor incorrecto, es decir:
  - sea menor o igual a `0` en el caso de los dos primeros enteros;
  - sea menor o igual a `2` o mayor que los dos anteriores en caso del tercer entero;
  - sea `null`, cadena vacía o sean iguales entre ellas, en el caso de las cadenas;

  se lanza y propaga una excepción **DatoNoValido** que tendrá como mensaje asociado el entero con valor incorrecto si se trata de uno de los tres primeros parámetros, o "`color no valido`" si se trata de alguna de las cadenas. Inicialmente el *tablero* no contiene ningún objeto.
- ⊙ *colocaFicha*: se le pasa por parámetro un objeto de tipo *Ficha* y un entero indicando la columna del *tablero* donde se quiere poner la ficha, de manera que la ficha se situa en la primera fila de esa columna que tenga una posición vacía (la primera fila vacía de una columna totalmente vacía es la correspondiente a la última fila[1]). Si el objeto pasado por parámetro se puede poner en el *tablero* se devuelve cierto, en cualquier otro caso se devuelve falso. Para poder poner la *Ficha* que se pasa por parámetro:
  1. La ficha tiene que ser de uno de los colores con los que se juega, en otro caso se lanza y propaga una excepción *DatoNoValido* pasando como parámetro el color de la ficha pasada por parámetro y el array de colores del juego.
  2. La ficha no puede estar ya colocada en otra posición.
  3. La columna tiene que ser una posición válida dentro del rango del *tablero*, en otro caso se lanza y propaga una excepción *UbicacionNoValida*, pasando como parámetro la columna al constructor de la excepción.

[1] Por ejemplo, en una matriz de `7x5` la última fila es la `6`.

4. En la columna en la que se quiere poner la ficha tiene que haber al menos una posición vacía.

Si se consigue poner la ficha se debe actualizar la *casilla* de dicha ficha con la posición que ocupa dentro del *tablero* y actualizar la posición del tablero con dicha ficha. A continuación se debe comprobar si con esta ficha se ha conseguido tener el número de fichas consecutivas del mismo color, y en ese caso el método tiene que lanzar y propagar la excepción *WinWin*, pasando como parámetro la ficha con la que se ha conseguido ganar el juego al constructor de la excepción.

- *iniciaTablero*: quita del *tablero* todas las fichas dejándolo vacío para empezar una nueva partida, eliminando de cada ficha la información sobre la posición que ocupaban en la partida jugada.
- *muestraTablero*: muestra por pantalla el contenido del tablero, una fila por línea, de la siguiente manera:

  1. si la posición contiene una ficha, muestra la primera letra del color de la ficha;
  2. si la posición no contiene una ficha, muestra por pantalla la letra 'X'.

  Por ejemplo para un tablero de tamaño 3x3 completamente vacío mostraría por pantalla:

```
XXX
XXX
XXX
```

- *getColor1*: devuelve el color que ocupa la posición 0 de *colores*.
- *getColor2*: devuelve el color que ocupa la posición 1 de *colores*.
- *getFilas*: devuelve el número de filas que tiene el tablero.
- *getColumnas*: devuelve el número de columnas que tiene el tablero.
- *getNumero*: devuelve el número de fichas consecutivas que hay que conseguir.

La clase **DatoNoValido**, que hereda de la clase *Exception* de java tiene tres constructores:

- Un constructor cuyo parámetro sea una cadena, de manera que el mensaje que devuelva al invocar a su método `getMessage` sea dicha cadena.
- Un constructor cuyo parámetro sea un entero, de manera que el mensaje que devuelva al invocar a su método `getMessage` sea dicho entero.
- Un constructor cuyos parámetros son una cadena y un array de dos cadenas, de manera que el mensaje que devuelva al invocar a su método `getMessage` sea el mensaje formado por la primera cadena y las dos contenidas en el array de la siguiente manera:
  `azul distinto de {lila,verde}` donde `azul` es el valor de la primera cadena pasada por parámetro y `lila` y `verde` las cadenas contenidas en el array pasado por parámetro en ese orden.

La clase **WinWin**, que hereda de la clase *Exception* de java tiene un constructor:

- Un constructor cuyo parámetro sea un objeto de tipo `Ficha`, de manera que el mensaje que devuelva al invocar a su método `getMessage` sea el mensaje formado por `color` de la ficha, paréntesis de apertura, la fila en la que está colocada, una coma, columna en la que está colocada y paréntesis de cierre. Es decir, el formato del mensaje será el siguiente para una ficha de color naranja colocada en la fila 5 y columna 1, por ejemplo:
  `naranja(5,1)`

La clase **UbicacionNoValida**, que hereda de la clase *`Exception`* de java tiene un constructor:

- Un constructor cuyo parámetro sea un entero, de manera que el mensaje que devuelva al invocar a su método `getMessage` sea dicho entero.

Como ya se comentó al inicio del enunciado, en esta práctica se tiene que leer de fichero de texto, con el objetivo de ser capaces de reproducir una partida de este juego a partir de la información contenida en el fichero. Un ejemplo de fichero de texto con información sobre una partida sería:

```
7 5 3 verde lila
lila 2
verde 3
lila 3
verde 2
lila 4
verde 1
```

Implementa una clase **Juego** con un método `main` que:

1. detecte el parámetro de la aplicación: un `String`. Si la aplicación no es invocada con un parámetro se debe emitir el siguiente mensaje por la salida estándar y no hacer nada más:

   ```
   Forma de uso: java Juego ficEntrada
   ```

   En otro caso, el primer parámetro se corresponde con el fichero de texto que se tiene que leer;

2. abra para lectura el fichero de texto correspondiente al parámetro;

3. cree un objeto de la clase *`Consecutivas`* con la información leída en la primera línea del fichero;

4. lea el fichero por líneas:

   - creando una ficha por línea con el color leído del fichero;
   - invocando el método *`colocaFicha`* del objeto *`Consecutivas`* creado pasando como parámetro la ficha creada y la columna correspondiente;
   - si se coloca la ficha muestra por pantalla el color de la ficha, fila y columna donde se ha colocado con el siguiente formato: color, guión, fila, coma, columna. Por ejemplo, para una ficha de color naranja colocada en la fila 5 y columna 1 mostrará:
     `naranja-5,1`

- si no se coloca la ficha solo muestra el color de la ficha:
  `naranja`

5. termine la ejecución por alguno de los siguientes motivos:

   a) no se puede crear el objeto *`Consecutivas`* porque hay algún dato no válido: en este caso no se puede jugar.

   b) se lanza la excepción *`WinWin`*: significa que alguien ha ganado, se muestra la excepción por pantalla y termina la ejecución;

   c) se acaba la información del fichero, por tanto no se puede continuar el juego;

6. implemente los manejadores de excepción de manera que muestre por pantalla la excepción capturada, ya que el `main` no propaga ninguna excepción, y continue la ejecución si no es uno de los casos comentados anteriormente;

Restricciones en la implementación

⊛ Los métodos deben ser públicos y tener una *`signatura`* concreta:

- En **Ubicacion**
  - `public Ubicacion(int,int)`
  - `public int getFila()`
  - `public int getColumna()`
- En **Ficha**
  - `public Ficha(String) throws DatoNoValido`
  - `public boolean colocaFicha(Ubicacion)`
  - `public int getFila()`
  - `public int getColumna()`
  - `public String getColor()`
  - `public void resetea()`
- En **Consecutivas**
  - `public Consecutivas(int,int,int,String,String) throws DatoNoValido`
  - `public boolean colocaFicha(Ficha,int) throws DatoNoValido,UbicacionNoValida,WinWin`
  - `public void iniciaTablero()`
  - `public String getColor1()`
  - `public String getColor2()`
  - `public int getFilas()`
  - `public int getColumnas()`
  - `public int getNumero()`
  - `public void muestraTablero()`

- En **DatoNoValido**
  - `public DatoNoValido(String)`
  - `public DatoNoValido(int)`
  - `public DatoNoValido(String,String[])`
  - `public String getMessage()`
- En **WinWin**
  - `public WinWin(Ficha)`
  - `public String getMessage()`
- En **UbicacionNoValida**
  - `public UbicacionNoValida(int)`
  - `public String getMessage()`
- En **Juego**
  - `public static void main(String[] args)`

⊛ De los ficheros entregados en esta práctica, solo el fichero `Juego.java` debe contener un método `public static void main(String[] args)`.

⊛ Todas las variables de instancia y de clase deben ser privadas. En otro caso se restará un `0.5` de la nota total de la práctica.

Segmentación de una cadena en distintos elementos

Una vez tenemos una línea del fichero en una variable de tipo `String`, por ejemplo la variable `linea`, hay que segmentarla para obtener de ella los distintos elementos que necesitamos para procesar la información contenida de manera conveniente.

Esta segmentación la podemos hacer usando el método `split` de la clase String. Este método devuelve un array de `Strings` a partir de uno dado usando el separador que se especifique. Por ejemplo, supongamos que tenemos la variable `linea` de tipo `String` que contiene la cadena "`verde 2`" de la cual queremos obtener por separado los dos datos que contiene: una cadena y un número entero. Para ello primero debemos indicar el separador y después segmentar usando el método `split` de la siguiente manera:

```
// indicamos el separador de campos
// en este caso un espacio en blanco
 String separador = " ";
// segmentamos
 String[] elems = linea.split(separador);
// asignamos la cadena verde contenida
// en elems[0] a la variable c1
 String c1=elems[0];
// convertimos a entero la cadena 2
// contenida en elems[1]
 int entero = Integer.parseInt(elems[1]);
```

Probar la práctica

- En el Campus Virtual se publicará un corrector de la práctica con un conjunto mínimo de pruebas (se recomienda realizar pruebas más exhaustivas de la práctica).
- Podéis enviar vuestras pruebas (fichero con extensión `java`, con un método `main` y sin errores de compilación) a los profesores de la asignatura mediante tutorías de UACloud, para obtener la salida correcta a esa prueba **a partir del 15 de mayo**. En ningún caso se modificará/corregirá el código de las pruebas. Los profesores contestarán a vuestra tutoría adjuntando la salida de vuestro `main`, si no da errores.
- El corrector viene en un archivo comprimido llamado `correctorPublicarP3.tgz`. Para descomprimirlo se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar:

  ```
  tar xfvz correctorPublicarP3.tgz
  ```

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica3-pruebas`: dentro de este directorio están los ficheros con extensión `.java`, programas en Java con un método main que realiza una serie de pruebas sobre la práctica, y los ficheros con el mismo nombre pero con extensión `.txt` con la salida correcta para la prueba correspondiente. Cuando la prueba requiere de un fichero de lectura éste tendrá la extensión `.lec`.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (sólo aquellos con extensión `.java`) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución si no los tiene. Para ello ejecutar:

  ```
  chmod +x corrige.sh
  corrige.sh
  ```

- Una vez se ejecuta `corrige.sh` los ficheros que se generarán son:

  - `errores.compilacion`: este fichero sólo se genera si el corrector emite por pantalla el mensaje "`Error de compilacion: 0`" y contiene los errores de compilación resultado de compilar los fuentes de una práctica particular. Para consultar el contenido de este fichero se puede abrir con cualquier editor de textos.
  - Ficheros con extensión `.tmp.err`: estos ficheros debe estar vacíos por regla general. Sólo contendrán información si el corrector emite el mensaje "`Prueba p01: Error de ejecucion`", por ejemplo para la prueba `p01`, y contendrá los errores de ejecución producidos al ejecutar el fuente `p01` con los ficheros de una práctica particular.
  - Ficheros con extensión `.tmp`: ficheros de texto con la salida generada por pantalla al ejecutar el fuente correspondiente, por ejemplo `p01.tmp` contendrá la salida generada al ejecutar el programa `p01` con los ficheros de una práctica particular.

Si en la prueba no sale el mensaje "`Prueba p01: Ok`", por ejemplo para la prueba `p01`, o algún otro de los comentados anteriormente, significa que hay diferencias en las salidas, por tanto se debe comprobar que diferencias puede haber entre los ficheros `p01.txt` y `p01.tmp`. Para ello ejecutar en línea de comando, dentro del directorio `practica3-pruebas`, la orden: `diff -i p01.txt p01.tmp`