Práctica 3: Excepciones y ficheros

Fecha de entrega: 10 de Diciembre de 2018, 9:00

Objetivo: Manejo de excepciones y lectura/escritura de ficheros.

1. Introducción

En esta práctica se ampliará la funcionalidad del juego *Plants vs Zombies* desarrollado en la práctica anterior. En particular, esta nueva funcionalidad se centra en dos aspectos principales:

- Incluir el manejo y tratamiento de excepciones. En ocasiones, existen estados en la ejecución del programa que deben ser tratados convenientemente. Además, cada estado debe proporcionar al usuario información relevante como, por ejemplo, errores producidos al procesar un determinado comando. En este caso, el objetivo es dotar al programa de mayor robustez, así como mejor la interoperabilidad con el usuario.
- Gestionar ficheros para poder grabar y cargar partidas en disco. De esta forma, se incluirán los comandos necesarios para poder realizar tanto la escritura del estado de una partida en disco, como su carga en memoria. Puesto que en la práctica anterior se utilizó el patrón *Command*, incluir los dos nuevos comandos se realizará de forma sencilla.

2. Manejo de excepciones

Es esta sección se detallarán las excepciones que deben tratarse durante el juego, así como detalles de su implementación y ejemplos de ejecución.

2.1. Descripción

El tratamiento de excepciones en Java resulta muy útil para controlar determinadas situaciones del juego en tiempo de ejecución, como por ejemplo, mostrar información relevante al usuario sobre la ejecución de un comando. En la práctica anterior, cada comando invocaba el método correspondiente - de la clase *Controller* o de la clase *Game* - para poder llevar a cabo las operaciones necesarias. Por ejemplo, para incluir una planta en el tablero, la clase *Game* debía comprobar si la casilla indicada por el usuario está libre y si se disponen de suficientes *sunCoins*. En caso de no poder incluir la planta indicada, se devolvía un valor `booleano` con valor `false`. De esta forma, el juego podía controlar que la planta no se había añadido, pero no se indicaba el motivo exacto (falta de sunCoins, casilla ocupada, nombre de la planta incorrecto, ...).

En esta práctica vamos a trabajar con el manejo de excepciones, de forma que cada clase pueda lanzar y procesar determinadas excepciones para tratar determinadas situaciones durante el juego. En algunos casos, estas situaciones consistirán únicamente en proporcionar un mensaje al usuario, mientras que en otras su tratamiento será más complejo. Cabe destacar que en esta sección no se van a tratar las excepciones relativas a los ficheros, las cuales serán explicadas en detalle en la sección siguiente.

Inicialmente se van a manejar las excepciones lanzadas por el sistema, es decir, aquellas que no son creadas ni lanzadas por el usuario. Al menos, debe tratarse las siguiente excepción:

- `NumberFormatException`, que será lanzada cuando se intente *parsear* un número, en formato String, que no pueda ser transformado al formato indicado.

Además, se deberán crear dos excepciones: *CommandParseException* y *CommandExecuteException*. La primera de ellas tratará los errores producidos al parsear un comando, es decir, aquéllos producidos durante la ejecución del método *parse*, tales como comando desconocido o parámetros incorrectos. La segunda se utilizará para tratar las situaciones de error al ejecutar el método *execute* de un comando como, por ejemplo, que una planta no pueda ser incluida en el tablero.

2.2. Implementación

Una de las principales modificaciones que realizaremos al incluir el manejo de excepciones en el juego consistirá en eliminar la comunicación directa entre los comandos y el controlador. De esta forma, no será necesario que cada comando indique al controlador cuándo se ha producido un error o cuándo se debe actualizar el tablero, sino que bastará con controlar las excepciones que puedan producirse durante la ejecución del comando. Además, puesto que ahora se van a tratar las situaciones de error tanto en el parseo como en la ejecución de los comandos, los mensajes de error mostrados al usuario serán mucho más descriptivos que en la práctica anterior. Básicamente, los cambios a realizar serán los siguientes:

1. Modificar la signatura del método *execute* - de la clase *Command* - para que devuelva un valor de tipo `boolean` en lugar de un `void`. Así, si el valor devuelto es `true`, el controlador podrá actualizar el tablero. En otro caso, el controlador no imprimirá el tablero.

2. Eliminar de la lista de parámetros, en los métodos *parse* y *execute* de los comandos, la referencia al objeto de la clase *Controller* . De forma similar, este parámetro deberá eliminarse del método *parseCommand* de la clase *CommandParser*.

3. El controlador deberá poder capturar las excepciones lanzadas por los métodos *execute* y *parse* de la clase *Command*.

4. Incluir el tratamiento de excepciones del sistema en las clases correspondientes para que puedan ser capturadas por el controlador e imprima por pantalla el mensaje correspondiente.

2.3. Ejemplos de ejecución

En esta sección se muestran algunos ejemplos que tratan las excepciones anteriormente descritas:

En la siguiente ejecución, el usuario intenta colocar un *Peashooter* en la coordenada (100, 200), la cual no existe. Por ello, el controlador debe capturar esta excepción y mostrar el mensaje correspondiente.

```
Command > 
Number of cycles: 12
Sun coins: 60
Remaining zombies: 6

   ---------------------------------------------------------------
  |       |       |       |       |       | N [4] | X [2] |       |
   ---------------------------------------------------------------
  | S [2] |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  |       |       |       |       |       | W [8] |       | X [2] |
   ---------------------------------------------------------------
  |       |       |       |       |       |       | X [2] |       |
   ---------------------------------------------------------------

Command > add p 100 200
ERROR: Coordinate (100, 200) is out of bounds.

Command > 
```

En la siguiente ejecución, el usuario intenta añadir una planta de tipo *Bananazooka*, la cual no existe.

```
Command > 
Number of cycles: 12
Sun coins: 60
Remaining zombies: 6

   ---------------------------------------------------------------
  |       |       |       |       |       | N [4] | X [2] |       |
   ---------------------------------------------------------------
  | S [2] |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  |       |       |       |       |       | W [8] |       | X [2] |
   ---------------------------------------------------------------
  |       |       |       |       |       |       | X [2] |       |
   ---------------------------------------------------------------

Command > add Bananazooka 2 2
ERROR: Plant Bananazooka does not exist.
Command > 
```

En la siguiente ejecución, el usuario intenta añadir una planta de tipo *Peashooter*. Sin embargo, no dispone de suficientes *sunCoins*.

```
Command > 
Number of cycles: 100
Sun coins: 10
Remaining zombies: 2

   ---------------------------------------------------------------
```

```
  |       |       |       |       |       | N [4] | X [2] |       |
   ---------------------------------------------------------------
  | S [2] |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  |       |       |       |       |       | W [8] |       | X [2] |
   ---------------------------------------------------------------
  |       |       |       |       |       |       | X [2] |       |
   ---------------------------------------------------------------

Command > add Peashooter 2 2
ERROR: Not enough sunCoins.
Command >
```

3. Ficheros

En esta sección se describe la nueva funcionalidad de la práctica en lo referente al tratamiento de ficheros. En esencia, la nueva funcionalidad consistirá en poder guardar y cargar partidas almacenadas en ficheros de texto.

3.1. Descripción

Para permitir guardar el estado de una partida en un fichero, así como cargar el estado de una partida previamente guardada, se van a crear dos comandos nuevos, SaveCommand y LoadCommand, tal que:

- `save fileName` guardará el estado actual de la partida en el fichero *fileName*. Hay que tener en cuenta que *fileName* es 'case-sensitive'y, por lo tanto, distingue entre mayúsculas y minúsculas.
- `load fileName` cargará la partida almacenada en el fichero *fileName*.

Se debe tener en cuenta que durante una misma partida se pueden guardar varios estados del juego, es decir, que se debe poder permitir al usuario guardar varios estados de la partida durante el transcurso de la misma utilizando, o no, ficheros diferentes. En el caso de que ya exista un fichero con el mismo nombre, se sobrescribirá el mismo.

3.2. Implementación

El formato que tendrá el fichero que contenga el estado de una partida será el siguiente:

```
cycle: yy
sunCoins: ss
level: ll
seed: ee
remZombies: zz
plantList: P1, P2, ..., Pn
zombieList: Z1, Z2, ..., Zm
```

donde *yy* es el número del ciclo actual, *ss* es el número de sunCoins que tiene el jugador, *ll* es el nivel del juego, *ee* es la seed de Random que se está utilizando, *zz* es el número de zombies que quedan por salir, *plantList* es la lista de plantas que están en el tablero y *zombieList* es la lista de los zombies que están en el tablero.

Las listas *plantList* y *zombieList* contienen todas las instancias de plantas y zombies que están en tablero, separadas por comas. Esto es, `P1, P2, ..., Pn` para las plantas y `Z1, Z2, ..., Zm` para los zombies. Para cada elemento de estas listas se guarda la siguiente información:

```
symbol:lr:lt:x:y:t
```

donde *symbol* es el símbolo del elemento, *lr* es la vida restante, *lt* es la vida total, x es el número de fila, y es el número de columna y t es el número de ciclos que faltan hasta que el objeto de juego tenga que realizar la siguiente acción.

En Java existen muchos mecanismos para manejar ficheros. En esta práctica usaremos flujos de caracteres en lugar de flujos de bytes. En particular, recomendamos el uso de BufferedWriter y FileWriter para escribir en un fichero, y BufferedReader y FileReader para leer de un fichero. Además, se recomienda el uso de bloques *try-with-resources* capturando `IOException` en cada uno de los métodos execute() de las clases LoadCommand y SaveCommand. Cada uno de estos métodos devuelve void.

Para poder guardar el estado de una partida, deberán tenerse en cuenta las siguientes consideraciones:

- Se debe comprobar que el nombre de un fichero es válido. En este caso, consideramos que el nombre de un fichero es válido si empieza por un carácter alfabético y contiene caracteres alfanuméricos y guión bajo (_). La extensión del fichero, que será siempre `.dat`, la añadirá automáticamente el programa. Así, el usuario únicamente deberá proporcionar el nombre sin extensión.

- Se debe comprobar que el fichero existe en el sistema de ficheros y puede abrirse.

- El método *execute()* de la clase SaveCommand invoca a un método llamado *getInfo()* de la clase *Game*, que a su vez deberá llamar a los métodos correspondientes de las listas para que devuelvan su contenido, esto es, la lista de plantas y la lista de zombies.

- La gestión de la escritura en disco, así como la apertura del fichero se realiza en el método execute del comando SaveCommand.

Para poder cargar el estado de una partida, deberán tenerse en cuenta las siguientes consideraciones:

- Se debe comprobar que el fichero existe en el sistema de ficheros y puede abrirse.

- La clase Game deberá almacenar el tamaño del tablero actual en atributos que puedan modificar su valor, ya que es posible cargar un tablero de un tamaño diferente al actual durante la partida.

- El método execute() de la clase LoadCommand invocará a los métodos correspondientes de la clase *Game* para establecer los datos leídos del fichero. Además, deberá llamar a los métodos correspondientes de las listas para que éstas creen los elementos correspondientes.

- La gestión de la lectura de disco, así como la apertura del fichero se realiza en el método execute del comando LoadCommand.

- Tras cargar una partida, se utiliza el pintado del tablero en modo `Release`.

3.3. Ejemplos de ejecución

En la siguiente ejecución, el usuario intenta cargar un fichero con un nombre de fichero no válido.

```
Command > load juego-1
ERROR: nombre de fichero incorrecto!
Command >
```

En la siguiente ejecución, el usuario intenta cargar un fichero que no existe.

```
Command > load juego123
ERROR: fichero ’juego123’ no encontrado!
Command >
```

Suponemos que existe un fichero llamado `tablero` con el siguiente contenido:

```
cycle: 10
sunCoins: 50
level: insane
seed: 0
remZombies: 5
plantList: S:2:3:1:0:1
zombieList: W:8:8:2:5:2, X:2:2:3:6:0
```

En la siguiente ejecución, el usuario carga el fichero llamado `tablero`:

```
Command > load tablero

Number of cycles: 10
Sun coins: 50
Remaining zombies: 5

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  | S [2] |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  |       |       |       |       |       | W [8] |       |       |
   ---------------------------------------------------------------
  |       |       |       |       |       |       | X [2] |       |
   ---------------------------------------------------------------
```

Suponemos que existe un fichero llamado `tablero2` con el siguiente contenido:

```
cycle: 25
sunCoins: 10
level: insane
seed: 0
remZombies: 1
plantList: S:2:3:0:1:1
zombieList: W:8:8:1:1:2
```

En la siguiente ejecución, el usuario carga el fichero llamado `tablero2`:

```

Number of cycles: 10
Sun coins: 50
Remaining zombies: 5

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------
  | S [2] |       |       |       |       |       |       |       |
```

```
 -------------------------------------------------------------------------------
|         |         |         |         |         |  W [8]  |         |         |
 -------------------------------------------------------------------------------
|         |         |         |         |         |         |  X [2]  |         |
 -------------------------------------------------------------------------------
```