Cuadernillo de prácticas
Tecnología de la Programación

Curso: 2017/2018

Departamento de Ingeniería del Software e Inteligencia Artificial
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Práctica 3: Manejo de excepciones y Ficheros

Fecha de entrega: 15 de Enero de 2018, 09:00

Objetivo: Desarrollo de aplicaciones que incluyen manejo de excepciones y ficheros.

1. Introducción

En esta práctica, ampliamos la práctica anterior de dos formas:

- Utilizamos el mecanismo de manejo de excepciones de Java para conseguir dos objetivos: que nuestra aplicación sea más robusta y mejorar su estructura.
- Vamos a permitir guardar el estado de la partida actual en un fichero de texto para poder cargarla posteriormente. En particular esto supone la introducción de dos nuevos comandos:
  - Guardar el estado actual de la partida en curso en un fichero.
  - Cargar el estado de una partida que ha sido guardado previamente en un fichero.

  El uso del patrón command, introducido en la práctica anterior, facilita la integración de estos nuevos comandos. El manejo de ficheros crea la necesidad de manejar excepciones. Todo el tratamiento de errores asociado a los ficheros, así como al resto de métodos susceptibles de fallar, lo haremos utilizando *excepciones*.

2. Manejo de excepciones

2.1. Description

Aquí describimos las principales excepciones que deben ser definidas, lanzadas y capturadas en nuestra aplicación (excluyendo de momento las asociadas al tratamiento de ficheros). Por supuesto, es posible definir adicionalmente otras excepciones si el alumno lo considera apropiado.

En primer lugar, se debe capturar cualquier excepción lanzada por el sistema (es decir, sin que el programador declare y lance explícitamente estas excepciones), tanto al leer de la consola como de los argumentos de entrada. Estas excepciones han de capturarse y actuar de forma conveniente. Por ejemplo, Game2048 debería poder capturar excepciones de tipo em NumberFormatException.

En segundo lugar, los errores que se produzcan en los siguiente métodos, han de gestionarse mediante el lanzamiento de excepciones:

- Los métodos de clase usados para almacenar el estado de la partida (comandos `undo` y `redo`).
- Métodos de la clase ArrayAsList. Por ejemplo, el método *pop()* de la clase *GameStateStack* debería poder lanzar una excepción del tipo *EmptyStackException*. Para lanzar excepciones desde los métodos de la clase ArrayAsList revisar la funcionalidad equivalente implementada en la clase ArrayList.

Las excepciones definidas y lanzadas por el programador deben poder gestionar las siguientes tipos de error:

- errores producidos en el método *parse* de los comandos.
- errores producidos en el método *execute* de los comandos.

También gestionaremos mediante excepciones el echo de que la partida esté finalizada. En este caso, distinguiremos dos tipos de situaciones dependiendo de si la partida ha sido ganada o perdida. Aunque esto no es en sí un error, podemos hacer uso del mecanismo de excepciones, de forma que el controlador capture estas excepciones e imprima un mensaje u otro dependiendo de si la partida ha terminado debido a una pérdida o a una victoria. Se recomienda implementar una clase *GameOverException* que contemple ambas situaciones.

2.2. Implementation

Usar el mecanismo de excepciones para gestionar los mensajes que imprime el método *run* del controlador. Con esto conseguimos evitar la comunicación entre el controlador y los comandos. Por un lado, los comandos no necesitan decirle al controlador que no imprima el estado de la partida en caso de que se haya producido un error en la ejecución del comando. Por otro lado, el controlador no necesita imprimir el mensaje de error "comando desconocido".

Modificar el método execute de la clase Command para que devuelva un valor de tipo boolean en lugar de void. Si el valor devuelto es *true*, el controlador imprimirá el estado de la partida. En caso contrario, el controlador no imprimirá dicho estado.

Con los dos cambios anteriores, los métodos parse() y execute() de la clase Command no necesitan al controlador como parámetro. Lo mismo le ocurre al método parseCommand() de la clase CommandParser.

Además, desaparece el método setNoPrintGameState(), su correspondiente atributo de tipo boolean attribute y todo lo relacionado con él. Las excepciones lanzadas por los métodos parse() y execute() de la clase Command han de ser capturadas por el controlador y actuar en consecuencia.

Es útil pasar el scanner como argumento del método parse() de la clase Command para ser utilizado en las clases PlayCommand y SaveCommand. Esto evita declarar un método getScanner() en la clase Controller. Con estos cambios, los métodos de la clase Command tienen la siguiente forma (a excepción de posible cláusula throws cuando sea necesario):

```
// execute returns a boolean to tell the controller whether or not to print the game
public abstract boolean execute(Game game);
public abstract Command parse(String[] commandWords, Scanner in);
public String helpText(){return "     " + commandText + ": " + helpText;}
```

2.3. Ejemplos de ejecución

En el siguiente ejemplo, los argumentos de ejecución de la aplicación (run arguments) no son números, por lo que la aplicación termina sin imprimir el tablero:

```
$> java tp.pr3.Game2048 four two

The command-line arguments must be numbers
```

En el siguiente ejemplo, se introducen valores no permitidos para el tamaño del tablero, número de celdas o para la semilla del comando `play`:

```
$> java tp.pr3.Game2048 4 2

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |   2   |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
     best value: 0          score: 0

Command > play fib
*** You have chosen to play the game: 2048, Fibonacci version ***
Please enter the size of the board: big
  The size of the board must be a number
Please enter the size of the board: 6
Please enter the number of initial cells: lots
  The number of initial cells must be a number
Please enter the number of initial cells: 4
Please enter the seed for the pseudo-random number generator: what's that?
  Please provide a single positive integer or press return
Please enter the seed for the pseudo-random number generator: ok
  The seed for the pseudo-random number generator must be a number
Please enter the seed for the pseudo-random number generator:
  Using the default seed for the pseudo-random number generator: 30

     ------------------------------------------------
    |       |       |       |       |       |       |
     ------------------------------------------------
    |       |   1   |       |       |       |   1   |
     ------------------------------------------------
    |       |       |       |       |       |       |
     ------------------------------------------------
    |       |       |   1   |       |       |       |
     ------------------------------------------------
    |   1   |       |       |       |       |       |
     ------------------------------------------------
    |       |       |       |       |       |       |
     ------------------------------------------------
     best value: 0          score: 0

Command > exit
Game over......
```

El parse del los comandos `move` and `play` lanza excepciones, pero las acciones realizadas en su captura no modifican la salida con respecto a la práctica anterior, por lo que no añadimos aquí nungún ejemplo de ejecución.

3. Ficheros

3.1. Descripción

Añadimos una nueva funcionalidad: almacenar el estado actual de la partida en un fichero para poder continuar dicha partida en un futuro. Añadimos dos nuevos comandos:

- `save <filename>` : sirve para guardar en un fichero el estado completo de una partida.
- `load <filename>` : sirve para cargar el estado completo de una partida almacenada previamente en un fichero.

Observaciones acerca de los comandos:

- El argumento `filename` es sensible a minúsculas/mayúsculas.
- La partida cargada con el comando `load` se puede corresponder a un tipo de juego diferente al juego que se esté jugando actualmente. Lo mismo ocurre con el tamaño del tablero y resto de características asociadas a la partida actual.

3.2. Implementación

Añadir dos nuevos comandos

Seguir el procedimiento empleado para la implementación de los comandos `undo` y `redo`.

Parse de los nuevos comandos

El método parse() de la clase SaveCommand debe comprobar que el nombre del fichero proporcionado es válido, es decir, que no contiene caracteres no válidos (esto depende del sistema operativo) y que el fichero exista.

Si el fichero no es válido se lanzará una excepción. Si el fichero existe, se preguntará al usuario si quiere sobreescribir el fichero. Si la respuesta es negativa, se pedirá de nuevo un nombre de fichero y se procederá a validarlo como se ha descrito anteriormente (ver apéndices).

El método parse() de la clase LoadCommand comprobará que el fichero proporcionado es válido y existe (ver apéndices).

Formato del fichero usado para el almacenamiento del estado de las partidas

La ejecución del comando `save` implica la necesidad de almacenar en un fichero de texto información acerca del tipo de juego asociado a la partida actual. Para ello vamos a añadir un nuevo atributo de tipo GameType a la clase Game. Hacemos esto porque es más sencillo almacenar la representación en forma de string de los objetos del tipo GameType que la representación en forma de string de los objetos del tipo GameRules. Este cambio puede provocar cambios en más clases.

En la ejecución del comando `load`, el tipo de juego debe ser leído del fichero (su representación en forma de string) y posteriormente debe crearse un objeto de alguna de las clases que implementan GameRules. Ver el código proporcionado de la clase GameType en el apéndice.

El estado de la partida debe ser almacenado con el siguiente formato:

```
This file stores a saved 2048 game

4 0 0 0
0 0 2 0
0 0 0 0
2 0 0 0

2 4 original
```

La primera línea es información acerca de la aplicación 2028. En la operación de lectura del fichero, se lanzará una excepción si el fichero no contiene esta primera línea. **Usar exactamente esta primera línea para los ficheros**.

Después aparece una línea en blanco seguida de la representación del tablero. Cada valor del tablero está separado por un tabulador. Por último, y después de una linea en blanco, aparecen dos valores numéricos y un valor de tipo string. Estos valores también están separados por tabuladores:

- Número inicial de celdas usadas en el juego (numCells),
- Número actual de puntos acumulado en la partida (score),
- Tipo de juego.

Revisar el código proporcionado en el apéndice de la clase GameType.

Ejecución de los dos nuevos comandos

Existen muchos mecanismos para abrir ficheros en Java. Aquí usaremos flujos de caracteres en lugar de flujos de bytes. En particular, recomendamos el uso de BufferedWriter y FileWriter par escribir en un fichero, y BufferedReader y FileReader para leer de un fichero. Utilizar bloques *try-with-resources* capturando IOException en cada uno de los métodos execute() de las clases LoadCommand y SaveCommand.

El método execute() de la clase SaveCommand invoca a un método llamado store() de la clase Game, que a su vez invoca a un método llamado store() de la clase Board. El método execute() de la clase LoadCommand invoca a un método llamado load() de la clase Game, que a su vez invoca a un método llamado load() de la clase Board. Cada uno de estos métodos devuelve void, excepto el método load() de la clase Game, que devuelve el tipo de juego para que el método execute() de la clase LoadCommand pueda incluir el tipo de juego almacenado en el mensaje que debe imprimirse antes de devolver el control al método run() del controller.

Observaciones:

- Como el tablero actual no tiene por qué tener el mismo tamaño que el tablero almacenado en el fichero, en la ejecución de `load` se debe deducir el tamaño del tablero almacenado a partir de los datos leídos del fichero.
- La última línea de la representación del tablero almacenada en el fichero no es la última línea del fichero. Esto es necesario tenerlo en cuenta a la hora de implementar el método load() de la clase Board.

3.3. Ejemplos de ejecución

```
$> java tp.pr3.Game2048 4 2
```

```
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
    |       |   2   |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
     best value: 0          score: 0

Command > move up

     ---------------------------------
    |       |   2   |       |   2   |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
     best value: 2          score: 0

Command > move left

     ---------------------------------
    |   4   |       |       |       |
     ---------------------------------
    |       |       |   2   |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |   2   |       |       |       |
     ---------------------------------
     best value: 4          score: 4

Command > save myFile.txt
Game successfully saved to file; use load command to reload it.

Command > move down

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
    |   4   |       |       |       |
     ---------------------------------
    |   2   |       |   2   |       |
     ---------------------------------
     best value: 4          score: 4

Command > save myFile.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: myfile.txt
Game successfully saved to file; use load command to reload it.

Command > move right

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
    |       |       |   2   |   4   |
     ---------------------------------
    |       |       |       |   4   |
     ---------------------------------
```

```
     best value: 4          score: 8

Command > load someFile.txt
File not found

Command > load myfile.txt
Game successfully loaded from file: 2048, original version

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
    |   4   |       |       |       |
     ---------------------------------
    |   2   |       |   2   |       |
     ---------------------------------
     best value: 4          score: 4

Command > play fib
*** You have chosen to play the game: 2048, Fibonacci version ***
Please enter the size of the board:
  Using the default size of the board: 4
Please enter the number of initial cells:
  Using the default number of initial cells: 2
Please enter the seed for the pseudo-random number generator:
  Using the default seed for the pseudo-random number generator: 379

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |   1   |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
     best value: 0          score: 0

Command > save myfile.txt
The file already exists; do you want to overwrite it? (Y/N): y
Game successfully saved to file; use load command to reload it.

Command > load myFile.txt
Game successfully loaded from file: 2048, original version

     ---------------------------------
    |   4   |       |       |       |
     ---------------------------------
    |       |       |   2   |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |   2   |       |       |       |
     ---------------------------------
     best value: 4          score: 4

Command > load myfile.txt
Game successfully loaded from file: 2048, Fibonacci version

     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |   1   |       |       |
     ---------------------------------
    |       |       |       |   2   |
     ---------------------------------
     best value: 2          score: 0
```

```
Command > exit
Game over......
```

Algunos errores de ejecución implican el lanzamiento de excepciones en una clase y la captura en otra. Otros errores se producen al cargar un fichero en el cual su primera línea no tiene la estructura esperada.

```
$> java tp.pr3.Game2048 4 2

     ---------------------------------
    |   2   |   2   |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
    |       |       |       |       |
     ---------------------------------
     best value: 0          score: 0

Command > load Test.txt
File not found

Command > load test.txt
Load failed: invalid file format

Command > save
Save must be followed by a filename

Command > load
Load must be followed by a filename

Command > save an interesting file
Invalid filename: the filename contains spaces

Command > load another interesting file
Invalid filename: the filename contains spaces

Command > save //-
Invalid filename: the filename contains invalid characters

Command > load //k
Invalid filename: the filename contains invalid characters

Command > save myfile.txt
The file already exists; do you want to overwrite it? (Y/N): ok
Please answer 'Y' or 'N'
The file already exists; do you want to overwrite it? (Y/N): y
Game successfully saved to file; use load command to reload it.

Command > save hello.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: my file
Invalid filename: the filename contains spaces

Command > save hello.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: //@%
Invalid filename: the filename contains invalid characters

Command > save hello.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: how_about_this.txt
Game successfully saved to file; use load command to reload it.

Command > save hello.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: hello1.txt
```

```
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: hello2.txt
The file already exists; do you want to overwrite it? (Y/N): n
Please enter another filename: this one
Invalid filename: the filename contains spaces

Command > exit
Game over......
```

4. Refactoring

Si no lo has implementado así en la práctica anterior, añadir el método por defecto siguiente en la interface GameRules:

```
boolean canMergeNeighbours(Cell cell1, Cell cell2)
```

Este método comprueba si dos celdas pueden fusionarse. Observa que dos de los juegos implementan dicho método de la misma forma, por eso sugerimos que dicho método sea implementado en la propia interface. Nótese que la implementación del método merge() de la interface GameRules en cada una de las clases que implementan esta interfaz ahora también pueden usar una llamada al nuevo método canMergeNeighbours().

5. Apéndice

En esta sección incluimos código de utilidad para la implementación de algunas de las tareas de la práctica. Observa la implementación de la clase GameType ya que puede servirte para mejorar la clase Direction.

```
package tp.pr3.logic.multigames;

public enum GameType {

    ORIG("2048, original version", "original", new Rules2048()),
    FIB("2048, Fibonacci version", "fib", new RulesFib()),
    INV("2048, inverse version", "inverse", new RulesInverse());

    private String userFriendlyName;
    private String parameterName;
    private GameRules correspondingRules;

    private GameType(String friendly, String param, GameRules rules){
        userFriendlyName = friendly;
        parameterName = param;
        correspondingRules = rules;
    }

    // precondition : param string contains only lower-case characters
    // used in PlayCommand and Game, in parse method and load method, respectively
    public static GameType parse(String param) {
        for (GameType gameType : GameType.values()) {
            if (gameType.parameterName.equals(param))
                return gameType;
        }
        return null;
    }

    // used in PlayCommand to build help message, and in parse method exception msg
    public static String externaliseAll () {
        String s = "";
        for (GameType type : GameType.values())
            s = s + " " + type.parameterName + ",";
        return s.substring(1, s.length()-1);
    }

    // used in Game when constructing object and when executing play command
    public GameRules getRules() {
        return correspondingRules;
    }

    // used in Game in store method
    public String externalise () {
        return parameterName;
    }

    // used PlayCommand and LoadCommand, in parse methods
    // in ack message and success message, respectively
    public String toString() {
```

```
        return userFriendlyName;
    }

}
```

```
    /* This code supposes the following attribute declarations :
     *     private boolean filename_confirmed;
     *     public static final filenameInUseMsg
     *         = "The file already exists ; do you want to overwrite it ? (Y/N)";
     * You may also need to add a throws clause to the declarations .
     */
    private String confirmFileNameStringForWrite(String filenameString, Scanner in) {
        String loadName = filenameString;
        filename_confirmed = false;
        while (!filename_confirmed) {
            if (MyStringUtils.validFileName(loadName)) {
                File file = new File(loadName);
                if (!file.exists())
                    filename_confirmed = true;
                else {
                    loadName = getLoadName(filenameString, in);
                }
            } else {
                // ADD SOME CODE HERE
            }
        }
        return loadName;
    }

    public String getLoadName(String filenameString, Scanner in) {
        String newFilename = null;
        boolean yesOrNo = false;
        while (!yesOrNo) {
            System.out.print(filenameInUseMsg + ": ");
            String[] responseYorN = in.nextLine().toLowerCase().trim().split("\\s+");
            if (responseYorN.length == 1) {
                switch (responseYorN[0]) {
                case "y":
                    yesOrNo = true;
                    // ADD SOME CODE HERE
                case "n":
                    yesOrNo = true;
                    // ADD SOME CODE HERE
                default:
                    // ADD SOME CODE HERE
                }
            } else {
                // ADD SOME CODE HERE
            }
        }
        return newFilename;
    }
```

```
package tp.pr3.util;
```

```
import java.io.File;
import java.io.IOException;
import java.io.FileOutputStream;

public class MyStringUtils {

    // used in text representation of Game and Board objects
    public static final int MARGIN_SIZE = 4;

    public static String repeat(String elmnt, int numOfTimes) {
        String result = "";
        for (int i = 0; i < numOfTimes; i++) {
            result += elmnt;
        }
        return result;
    }

    public static String centre(String text, int len){
        String out = String.format("%"+len+"s%s%"+len+"s", "",text,"");
        float mid = (out.length()/2);
        float start = mid - (len/2);
        float end = start + len;
        return out.substring((int)start, (int)end);
    }

    // Used to exist method: org.eclipse.core.internal.resources.OS.isNameValid(filename).
    // This method is not completely reliable since exception could also be thrown due to:
    // incorrect permissions, no space on disk, problem accessing the device,...
    public static boolean validFileName(String filename) {
        File file = new File(filename);
        if (file.exists()) {
            return canWriteLocal(file);
        } else {
            try {
                file.createNewFile();
                file.delete();
                return true;
            } catch (Exception e) {
                return false;
            }
        }
    }

    public static boolean canWriteLocal(File file) {
        // works OK on Linux but not on Windows (apparently!)
        if (!file.canWrite()) {
            return false;
        }
        // works on Windows
            try {
                new FileOutputStream(file, true).close();
            } catch (IOException e) {
                return false;
            }
        return true;
    }
```

```
}
```