Práctica 3: Excepciones y ficheros

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

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

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.

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.

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.

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.

Ahora todos los mensajes se imprimen desde el método *run* del controlador[1] cuyo cuerpo se asemejará al código siguiente:

```
while (!game.isFinished()){
    System.out.print(prompt);
    String[]  words = in.nextLine().trim().split ("\\s+");
    try {
        Command command = CommandGenerator.parse(words);
            if (command != null) {
                if (command.execute(game)) printGame();
            } else
                System.out.println(unknownCommandMsg);
     } catch (CommandParseException | CommandExecuteException ex) {
        System.out.format(ex.getMessage() + "%n%n");
     }
}
```

Ejemplos de ejecución

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

La ejecución con un número incorrecto de parámetros del programa (0 o más que 2) produce el mensaje siguiente:

```
Usage: plantsVsZombies <EASY|HARD|INSANE> [seed]
```

La ejecución con parámetros del programa *difficult* produce el mensaje siguiente:

```
Usage: plantsVsZombies <EASY|HARD|INSANE> [seed]: level must be one of: EASY, HARD, INSANE
```

La ejecución con parámetros del programa *easy XL* produce el mensaje siguiente (después de capturar una *RuntimeException*):

```
Usage: plantsVsZombies <EASY|HARD|INSANE> [seed]: the seed must be a number
```

Ahora mostramos una ejecución con parámetros de programa válidos (tales como *insane* o *easy -1* o *hard 0*):

```
Welcome to plantsVsZombies v3.0
Random seed used: 578

Current cycle: 0
Sun coins: 50
Remaining zombies: 5

   -------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   -------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   -------------------------------------------------------------
```

[1] salvo los mensajes que indican que se ha guardado / cargado el juego, ver la siguiente sección

```
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------

Command > add torchwood 2 3
Unknown plant name: torchwood

Command > add sun 2 3
Unknown plant name: sun

Command > add plant sunflower 2 3
Incorrect number of arguments for add command: [A]dd <plant> <x> <y>

Command > add s b 3
Invalid argument for add command, number expected: [A]dd <plant> <x> <y>

Command > add s 7 9
Failed to add Sunflower: (7, 9) is an invalid position

Command > add s 0 0
Current cycle: 1
Sun coins: 30
Remaining zombies: 5

    -----------------------------------------------------------------
   | S[1]  |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------

Command > add s 0 0
Failed to add Sunflower: position (0, 0) is already occupied

Command > add s 1 0
Current cycle: 2
Sun coins: 10
Remaining zombies: 5

    -----------------------------------------------------------------
   | S[1]  |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   | S[1]  |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    -----------------------------------------------------------------

Command > add p 2 0
Failed to add Peashooter: not enough suncoins to buy it

Command > help me
Help command has no arguments

Command > None now
None command has no arguments

Command > printMode fast&furious
Unknown print mode: fast&furious

Command > printmode debug it
Incorrect number of arguments for printmode command: [P]rintMode release|debug

Command > PrintMode
Incorrect number of arguments for printmode command: [P]rintMode release|debug
```

```
Command > print
Unknown command. Use 'help' to see the available commands

Command > quit
Unknown command. Use 'help' to see the available commands

Command > help
The available commands are:
     [A]dd <plant> <x> <y>: Add a plant in position (x, y).
     [H]elp: Show this help message.
     [R]eset: Start a new game.
     [E]xit: Terminate the game.
     [L]ist: Show the list of available plants.
     [N]one: Update the game without any user action.
     [P]rintMode <mode>: Change the print mode [Release|Debug].
     [Z]ombielist: Show the available Zombies.
     addzombie <zombie> <x> <y>: Add a zombie in position (x,y)

Command > exit

      ****** Game over!: User exit ******
```

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.

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.

Implementación

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

```
cycle: yy
sunCoins: ss
level: ll
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, *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:x:y:t
```

donde *symbol* es el símbolo del elemento, *lr* es la vida restante, 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.

El manjeo de ficheros implica el uso de nuevos tipos de excepciones, tanto del sistema como definidas por el usuario. A este respecto, se deberá crear una nueva excepción FileContentsException y lanzarla en caso de detectar un problema en el contenido del fichero del tipo que no puede ser detectado por el sistema, p.ej. un *level* desconocido.

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* para el código donde se abre el fichero, capturando `IOException` en cada uno de los métodos execute() de las clases LoadCommand y SaveCommand.

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

- El método *execute()* de la clase SaveCommand debe
  - Comprobar que el nombre del fichero proporcionado por el usuario es un nombre válido de fichero, donde esta noción depende del sistema operativo.
  - Intentar abrir el fichero con este nombre en escritura. Para simplificar la tarea, si el fichero ya existe, se sobreescribirá.
  - Escribir la cabecera, que consta del mensaje "Plants Vs Zombies v3.0", en el fichero seguido por una línea en blanco. La extensión del fichero, que será siempre `.dat`, la añadirá automáticamente el programa. Así, el usuario únicamente deberá proporcionar el nombre sin extensión.
- A continuación, el método *execute()* de la clase SaveCommand invoca a un método llamado *store()* de la clase *Game*, que a su vez llamará a un método *store()* de la clase *Board*, si existe esta última clase. Respecto al método *store()* de *Game* (o de *Board* en su caso),
  - o bien puede llamar a un método *store()* de cada una de las dos listas, que a su vez llaman a un método *store()* de cada elemento de cada lista, de modo que cada clase se ocupe de escribir sus propios datos en fichero,
  - o bien puede llamar a un método *externalise()* de cada una de las dos listas, que a su vez llaman a un método *externalise()* de cada elemento de cada lista, donde los métodos *externalise()* devuelven un objeto *String*, de manera parecida al funcionamiento del método *toString()*.
- En caso de haber podido guardar el estado del juego en fichero con éxito, se debe imprimir el mensaje:

"Game successfully saved in file <*nombre_proporcionado_por_el_usuario*>.dat. Use the load command to reload it".

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

- El método *execute()* de la clase LoadCommand debe
  - Comprobar que el nombre del fichero proporcionado por el usuario es un nombre válido de fichero y que existe en el sistema de ficheros.
  - Intentar abrir el fichero con este nombre en lectura
  - Leer la cabecera, que consta del mensaje "Plants Vs Zombies v3.0", seguido por una línea en blanco. En un contexto real, probablemente también aceptaría cabeceras de algunas versiones anteriores.
- A continuación, el método execute() de la clase LoadCommand invocará a un método llamado *load()* de la clase *Game*, que a su vez llamará a un método *load()* de la clase *Board*, si existe esta última clase. No hace falta que el método *load()* de *Game* (o de *Board* en su caso) invoque a un método *load()* de cada una de las dos listas, que a su vez llamen a un método *load()* de cada elemento de cada lista, ya que la lectura de ficheros se hace más facilmente por líneas enteras. En vez de ello, el método *load()* de *Game* (o de *Board*) en su caso) puede crear una nueva lista de plantas y una nueva lista de zombies para que las nuevas listas puedan poblarse con los datos leídos de fichero.
- En caso de ocurrir un problema a la hora de cargar un fichero, el método *execute()* de la clase LoadCommand debe recoger la excepción lanzada y reempaquetarla en un *CommandExcecuteException*. Hay circunstancias en las que es buena práctica recoger excepciones de bajo nivel para a continuación lanzar una excepción de alto nivel[2]. En particular:
- El método *load()* de *Game*, así como los métodos llamados desde este método, pueden comprobar la coherencia de los datos leídos y, en caso de haber un problema, lanzar un *FileContentsException* que será recogida por el método *execute()* de la clase LoadCommand.
- En caso de haber podido cargar el estado del juego de fichero con éxito, se debe imprimir el mensaje:
  "Game successfully loaded from file <*nombre_proporcionado_por_el_usuario*>".
  Tras cargar una partida, se utiliza el pintado del tablero en modo `Release`.

En el apendice proporcionamos el código de un método para comprobar que una cadena de caracteres constituye un nombre de fichero válido, así como un método para comprobar que un fichero con este nombre existe en el sistema de ficheros y un método para comprobar que un fichero con este nombre puede abrirse en lectura. También proporcionamos código para el enum *Level* y para un método *loadLine* que puede llamarse desde el método *load* de la clase *Game*.

[2]Tal vez habrás visto ejemplos de mala práctica a este respecto, en forma de aplicaciones de comercio electrónico en web que imprimen para el usuario un mensaje de tipo "SQL error..."; al usuario no le importa este nivel de detalle de implementación.

Mejorar la calidad del código

Para mejorar la calidad del código, debe tomar en cuenta las siguientes observaciones:

Robustez Al cargar un fichero con datos inválidos, la aplicación no debería caerse y tampoco debería quedarse en un estado incoherente. Para evitar que se produzcan estados incoherentes, el método `load` de `Game` debería almacenar los datos del `game` que se quieren cargar en una especie de copia de seguridad (en memoria) antes de empezar a cargarlos. De este modo, cuando se produzca una excepción debido a que el contenido del fichero sea incorrecto, en el método `load` de `Game`, se podrá recoger esta excepción, reponer el estado anterior del `game` a partir de la copia de seguridad y a continuación relanzar la misma excepción (para que se recoja más arriba en la pila de llamadas).

Más refactorización La aplicación sería más mantenible y flexible si las clases `PlantFactory` y `ZombieFactory` utilizaran el mismo mecanismo para la creación de plantas que la clase `CommandParser` utiliza para la creación de comandos. Es decir, que el funcionamiento de los métodos `getPlant` y `getZombie` de las primeras fuera el del método `parse` de la segunda, lo que implicaría tener un método `parse` en los objetos del juego. Ya que estamos, se podría también crear una clase `PrinterManager` para hacer el papel de la clase `CommandParser` y utilizar este mismo mecanismo para gestionar los distintos tipos de printer, lo que implicaría tener un método `parse` en los printers.

Ejemplos de ejecución

A continuación se muestra la traza de una ejecución realizada en Windows.

```
Welcome to plantsVsZombies v3.0

Current cycle: 0
Sun coins: 50
Remaining zombies: 5

   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > help
Available commands:
     [A]dd <plant> <x> <y>: Add a plant in position (x, y).
     [H]elp: Show this help message.
     [R]eset: Start a new game.
     [E]xit: Terminate the game.
     [L]ist: Show the list of available plants.
     [N]one: Update the game without any user action.
     [P]rintMode <mode>: Change the print mode [Release|Debug].
     [S]ave <filename>: Save the state of the game to a file.
     [Lo]ad <filename>: Load the state of the game from a file.
     [Z]ombielist: Show the available Zombies.
     addzombie <zombie> <x> <y>: Adds a zombie in position (x,y).

Command > save /*
Invalid filename: the filename contains invalid characters
```

```
Command > save this
Game successfully saved to file this.dat; use the load command to reload it.

Command > load that
File not found

Command > load this
File not found

Command > load this.dat
Game successfully loaded from file this.dat.

Current cycle: 0
Sun coins: 50
Remaining zombies: 5

   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > add s 0 0
Current cycle: 1
Sun coins: 30
Remaining zombies: 5

   ------------------------------------------------------------
  | S[1]  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > add p 1 0
Current cycle: 2
Sun coins: 10
Remaining zombies: 5

   ------------------------------------------------------------
  | S[1]  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  | P[3]  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > save this
Game successfully saved to file this.dat; use the load command to reload it.

Command > reset
Current cycle: 0
Sun coins: 50
Remaining zombies: 5

   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------------------------------------------------------------
```

```
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------

Command > load this.dat
Game successfully loaded from file this.dat.

Current cycle: 2
Sun coins: 10
Remaining zombies: 5

    ---------------------------------------------------------------
   | S[1]  |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   | P[3]  |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------

Command >
Current cycle: 3
Sun coins: 10
Remaining zombies: 5

    ---------------------------------------------------------------
   | S[1]  |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   | P[3]  |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------

Command >
Current cycle: 4
Sun coins: 20
Remaining zombies: 4

    ---------------------------------------------------------------
   | S[1]  |       |       |       |       |       | W[8]  |       |
    ---------------------------------------------------------------
   | P[3]  |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------

Command > save this
Game successfully saved to file this.dat; use the load command to reload it.

Command > reset
Current cycle: 0
Sun coins: 50
Remaining zombies: 5

    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
   |       |       |       |       |       |       |       |       |
    ---------------------------------------------------------------
```

```
Command > load this.dat
Game successfully loaded from file this.dat.

Current cycle: 4
Sun coins: 20
Remaining zombies: 4

   ------------------------------------------------------------
  | S[1]  |       |       |       |       |       | W[8]  |
   ------------------------------------------------------------
  | P[3]  |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > load defective_file.dat
Load failed: invalid file contents

Command >
Current cycle: 5
Sun coins: 20
Remaining zombies: 4

   ------------------------------------------------------------
  | S[1]  |       |       |       |       |       | B[8]  |
   ------------------------------------------------------------
  | P[3]  |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command >
Current cycle: 6
Sun coins: 30
Remaining zombies: 4

   ------------------------------------------------------------
  | S[1]  |       |       |       |       |       | B[8]  |
   ------------------------------------------------------------
  | P[3]  |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------------------------------------------------------------

Command > exit

      ****** Game over!: User exit ******
```

Referencias

The Java Tutorials: Exceptions
https://docs.oracle.com/javase/tutorial/essential/exceptions/

The Java Tutorials: Input-output Streams
https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Anexo a la Práctica 3

```
package tp.p3.util;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;
import java.nio.file.InvalidPathException;

public class MyStringUtils {

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

    // returns true if string argument is a valid filename
    public static boolean isValidFilename(String filename) {
        try {
            Paths.get(filename);
            return true;
        } catch (InvalidPathException ipe) {
            return false;
        }
    }
    // returns true if file with that name exists (in which case, it may not be accessible)
    public static boolean fileExists(String filename) {
        try {
            Path path = Paths.get(filename);
            return Files.exists(path) && Files.isRegularFile(path);
        } catch (InvalidPathException ipe) {
            return false; // filename invalid => file cannot exist
        }
    }
    // returns true if file wth that name exists and is readable
    public static boolean isReadable(String filename) {
        try {
            Path path = Paths.get(filename);
            return Files.exists(path) && Files.isRegularFile(path) && Files.isReadable(path);
        } catch (InvalidPathException ipe) {
            return false; // filename invalid => file cannot exist, never mind be readable
        }
    }

}
```

```
package tp.p3.game;

public enum Level {

    EASY(3, 0.1), HARD(5, 0.2), INSANE(10, 0.3);

    private int numberOfZombies;
    private double zombieFrequency;

    private Level(int zombieNum, double zombieFreq){
        numberOfZombies = zombieNum;
        zombieFrequency = zombieFreq;
    }

    public int getNumberOfZombies() {
        return numberOfZombies;
    }

    public double getZombieFrequency() {
        return zombieFrequency;
    }

    public static Level parse(String inputString) {
        for (Level level : Level.values())
            if (level.name().equalsIgnoreCase(inputString)) return level;
        return null;
    }

    public static String all(String separator) {
        StringBuilder sb = new StringBuilder();
        for (Level level : Level.values())
            sb.append(level.name() + separator);
        String allLevels = sb.toString();
        return allLevels.substring(0, allLevels.length()-separator.length());
    }

}
```

```
public static final String wrongPrefixMsg = "unknown game attribute: "
public static final String lineTooLongMsg = "too many words on line commencing: "
public static final String lineTooShortMsg = "missing data on line commencing: "

public String[] loadLine(BufferedReader inStream, String prefix, boolean isList)
                        throws IOException, FileContentsException{
   String line = inStream.readLine().trim();
   // absence of the prefix is invalid
   if ( !line.startsWith(prefix + ":") )
      throw new FileContentsException(wrongPrefixMsg + prefix);
   // cut the prefix and the following colon off the line
   // then trim it to get the attribute contents
   String contentString = line.substring(prefix.length()+1).trim();
   String[] words;
   // the attribute contents are not empty
   if (!contentString.equals("")) {
      if (!isList ) {
         // split non-list attribute contents into words
         // using 1-or-more-white-spaces as separator
         words = contentString.split("\\s+");
         // a non-list attribute with contents of more than one word is invalid
         if (words.length != 1)
            throw new FileContentsException(lineTooLongMsg + prefix);
      } else
         // split list attribute contents into words
         // using comma+0-or-more-white-spaces as separator
         words = contentString.split(",\\s*");
   // the attribute contents are empty
   } else {
      // a non-list attribute with empty contents is invalid
         if (!isList )
            throw new FileContentsException(lineTooShortMsg + prefix);
      // a list attibute with empty contents is valid;
      // use a zero-length array to store its words
         words = new String[0];
   }
   return words;
}
```