Programación para Ingeniería Telemática

Curso 2018-19

Práctica 3: Entrada/Salida de texto.

1. Objetivos de aprendizaje. Estructura de la práctica.

Objetivos

- Definir nuevos elementos para el juego.
- Ser capaces de serializar objetos, guardarlos en un fichero de texto y recuperarlos.
- Ser capaces de utilizar el mecanismo de lanzamiento y captura de excepciones de Java.

En esta práctica implementaremos nuevas características de los personajes y veremos cómo guardarlos en un fichero en formato texto y posteriormente recuperarlos. La entrada y salida en fichero nos proporcionará un primer contacto con el paquete `java.io` y nos permitirá trabajar con excepciones. Lo que aprendamos en esta práctica nos servirá posteriormente para guardar y recuperar partidas e incluso para implementar una versión distribuida del juego en la última parte del curso.

Contenidos de la memoria

Esta memoria se estructura en cinco secciones con los siguientes contenidos:

Sección 1: Objetivos y contenido de la práctica.

Sección 2: Los paquetes `common` y `game`. En esta sección se presenta el código de partida que tendrá que usarse para realizar esta práctica en concreto. Este código se irá ampliando en sucesivas prácticas.

Sección 3: Enunciado de los ejercicios a realizar. Estos ejercicios están pensados para facilitar la realización de la práctica entregable correspondiente al bloque de la asignatura dedicado a la programación orientada a objetos. Además, los ejemplos y ejercicios que se vayan mostrando en las prácticas y o bien son semejantes a los que pueden pedirse en los exámenes de la signatura o bien constituyen un paso previo para ser capaces de realizarlos.

Antes de hacer los ejercicios es aconsejable tener una visión de conjunto, por ello **se recomienda leer detenidamente el boletín de prácticas y entender el código suministrado antes de abordar la resolución de los ejercicios.**

2. Los paquetes common y game

2.1. El paquete common

El paquete `common` contiene una nueva clase, `FileUtilities`, que contiene métodos estáticos para leer y escribir en ficheros de texto.

Esta clase se proporciona parcialmente implementada. Algunos de los métodos tendrán que ser modificados en los ejercicios y otros tendrán que completarse. En la Tabla 1 se muestra una breve descripción de los métodos de esta clase.

Tabla 1: Métodos estáticos de `FileUtilities`

Método	**Descripción**
`static void` **`createDirectory`** `(String pathname)`	Crea un directorio en la ruta inidicada.
`static void` **`writeToFile`** `(String s, String fichero) throws FileNotFoundException`	Escribe una cadena de caracteres en un fichero. Lanza excepción si no existe el fichero, no puede crearse o no puede escribirse en él.
`static void` **`writeJsonsToFile`** `( JSONObject [] jsons, String fileName)`	Escribe un array de jsons en un fichero de texto. Cada objeto json se escribe en una línea diferente.
`static JSONArray` **`readJsonsFromFile`** `(String fileName)`	Lee objetos json de un fichero de texto en donde han sido previamente guardados usando `writeJsonsToFile`.

2.2. El paquete game

En esta práctica se va a ampliar este paquete con un nuevo elemento del juego, cuya implementación se deja como ejercicio: la clase `RidingHood_1` (figura 1) y una clase `TestRidingHood` (tabla 2) para probar su funcionamiento. La clase `TestRidingHood` dará errores de compilación hasta que implemente los constructores pedidos en los ejercicios 1 y 2.

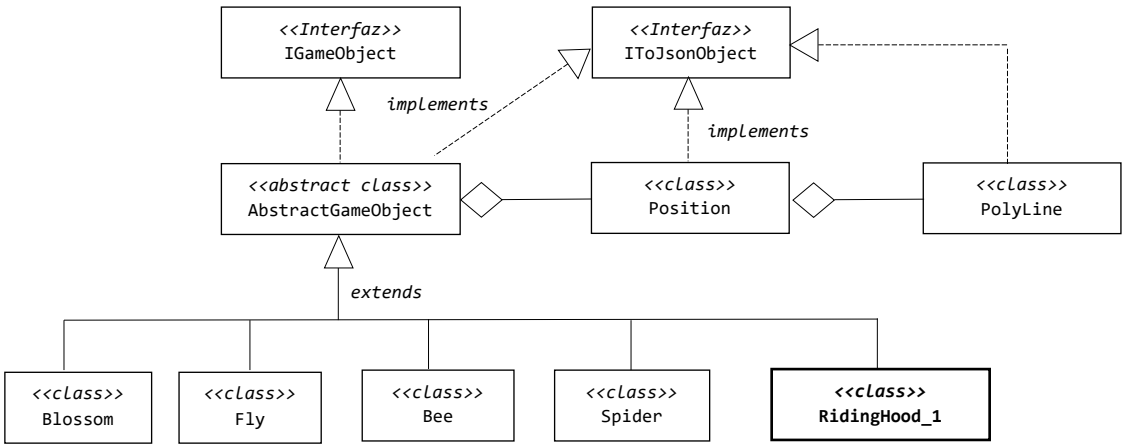

Figura 1: Tipos de datos

Tabla 2: Métodos estáticos de `TestRidingHood`	
Método	**Descripción**
`static void` **`testConstructorAndToJson`** `()`	Crea dos objetos `RidingHoog`. Comprueba su paso a JSON y el constructor con objeto JSON.
`static Position [] testMoveToNextPosition ( IGameObject gObj, Blossom targets[], int n)`	Invoca n veces el método `moveToNextPosition` del objeto `gObj` y devuelve un array con las posiciones por las que ha pasado.
`static void testSaveAndLoad ( JSONObject [] jsons, String fileName)`	Guarda un array de posiciones en un fichero, las lee del fichero y después las muestra por consola.

3. Ejercicios a realizar

Ejercicio 1:

1. Modifique el método `createDirectory` para que no lance excepciones y pruebe su funcionamiento ejecutando el método `main` de la clase `FileUtilities`, que crea un directorio en `Other Sources` (`"src/main/resources/jsons"`).
2. Complete la Implementación del método `readJsonsFromFile` y pruebe su funcionamiento. Para ello, elimine los comentarios de las líneas de prueba de lectura del método `main` de la clase `FileUtilities`.

   Para implementar este método tendrá que manejar un objeto del tipo `JSONArray` definido en la librería `org.json`. Puede encontrar ejemplos de uso en la clase `Polyline` y una documentación bastante completa en https://stleary.github.io/JSON-java/org/json/JSONArray.html

Ejercicio 2:

Implemente la clase `RidingHood_1`, de forma que:

1. Aporte su propia versión de todos los constructores definidos en `AbstractGameObject`. Una vez que haya definido estos constructores, los errores de compilación del método `testConstructorAndToJson` desaparecerán. Comente las líneas de `TestRidingHood` que sigan dando errores de compilación o causen problemas durante la ejecución del método `main` y podrá utilizar esta clase para probar el funcionamiento de los constructores pedidos.

2. Defina un nuevo constructor que tome como argumento un array de objetos `Blossom`:

   ```
   RidingHood_1(Position position, int value, int life, Blossom [] blossoms)
   ```

   Ni el método `toJSONObject` ni el constructor que toma como argumento un objeto json tienen que tomar en consideración los objetos almacenados en este array.

   Una vez implementado este constructor no deben aparecer errores de compilación en `TestRidingHood`. Descomente las líneas que hubiera comentado anteriormente en esta clase.

3. Añada un método privado sin argumentos `moveDiagonal` que incremente las posiciones x e y del objeto en una unidad (`void moveDiagonal()`).

4. Añada un método privado que tome como argumento una posición y modifique la posición actual del objeto `RidingHood_1` de forma que se acerque a dicha posición en una unidad tanto en x como en y (`void approachTo(Position p)`).

5. Implemente el método `moveToNextPosition()` de forma que:

   5.1. Cuando invoque el método el objeto se aproxime al *blossom* más cercano moviéndose una posición en x y otra en y.
   5.2. Cuando llegue a un *blossom* se dirija al siguiente más cercano hasta recorrerlos todos.
   5.3. Cuando ha pasado por todos los *blossoms* avanza en diagonal hacia abajo a la derecha.

6. Implemente el método `testSaveAndLoad` de la clase `TestRidingHood` y pruebe su funcionamiento.