Práctica 1: El juego 2048

Fecha de entrega: 13 de Noviembre de 2017, 09:00

Objetivo: Iniciación a la orientación a objetos y a Java; uso de arrays y enumerados; manipulación de cadenas con la clase `String`; entrada y salida por consola.

1. Introducción

El 2048 es un juego de tablero cuyo objetivo es deslizar las baldosas del tablero para combinarlas y crear una baldosa con el número 2048. Se realizan movimientos de dirección izquierda, derecha, arriba y abajo para mover las baldosas, las cuales se deslizan en su totalidad por el tablero. Si no estás familiarizado con el juego, es recomendable que inviertas algo de tiempo (¡no mucho!) en jugar. Existen versiones gráficas de dicho juego en la web. Por ejemplo en la URL:
`http://gabrielecirulli.github.io/2048/`

La configuración estándar del juego consta de un tablero de tamaño 4×4 con dos baldosas ocupadas y un marcador de puntos, que inicialmente tiene valor 0.

En esta práctica tenemos como objetivo implementar una generalización del juego 2048 mediante interfaz de consola.

En el juego 2048, un movimiento implica las siguientes acciones:

1. ***Desplazar***. Se trata de mover todas las baldosas en la dirección que indique el movimiento (arriba, abajo, derecha, izquierda) sin dejar espacios en blanco entre baldosas.

2. ***Fusionar***. Si dos baldosas con el mismo valor colisionan durante un movimiento, se combinarán en la baldosa más cercana al borde del tablero (desapareciendo la otra baldosa), cuyo valor será el equivalente a la suma de los valores de las dos baldosas originales (es decir, si dos baldosas con el valor 4 colisionan, se combinarán en una baldosa con el valor 8). Sin embargo, la baldosa resultante no podrá combinarse con otra baldosa nuevamente en un mismo movimiento.

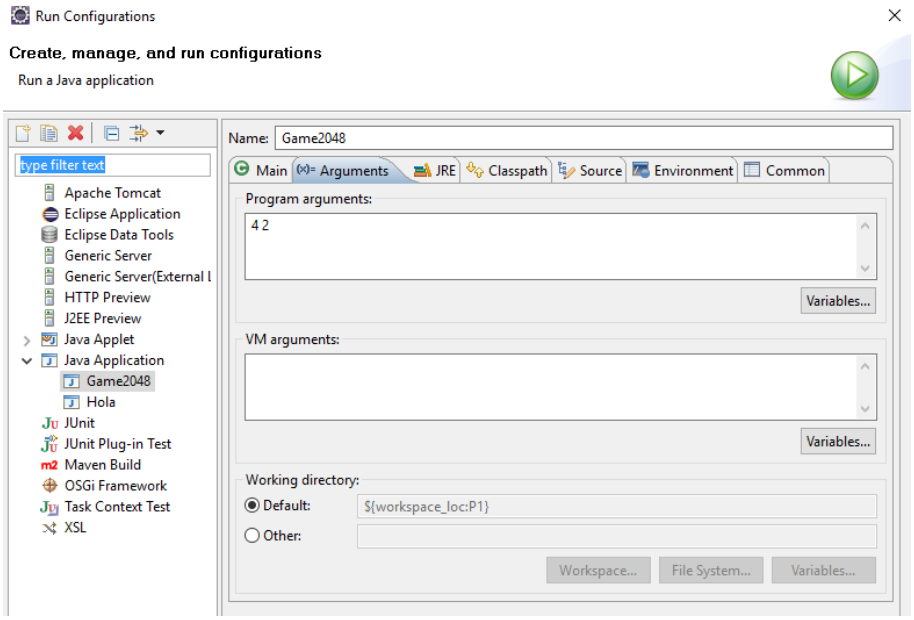

Figura 1: Opciones de ejecución

3. ***Nuevo valor.*** Por último, aparecerá una baldosa nueva en un lugar vacío del tablero y elegido de forma aleatoria, cuyo valor podrá ser 2 (con una probabilidad del 90 % ó 4 (con una probabilidad del 10 %).

Después de cada movimiento se recalcula el marcador de puntos. El marcador comienza en cero y, cuando dos baldosas se combinan, este se incrementa por el valor de la baldosa resultante.

El juego termina cuando no quedan movimientos por hacer o cuando se logra obtener una baldosa con valor 2048 (de ahí el nombre del juego), en cuyo caso se dice que el juego acaba en victoria.

Observaciones:

- Si una baldosa es resultado de una combinación durante la fase ***Fusionar*** de un movimiento, no se puede volver a combinar en el mismo movimiento.
- El marcador de puntos solo se actualiza si dos baldosas se han fusionado.
- Nótese que solo se podrá realizar movimientos mientras haya baldosas vacías o cuando no habiendo vacías, haya baldosas adyacentes con el mismo valor.

2. Descripción de la práctica

2.1. Parámetros de la aplicación

Aunque la configuración estándar del juego consta de un tablero de tamaño 4×4 con dos baldosas ocupadas, nuestra implementación permitirá seleccionar el tamaño del tablero y el número de baldosas ocupadas (ver Figura 1).

Opcionalmente, se podrá especificar la semilla para la generación de números aleatorios.

Por tanto, el programa debe aceptar 2 parámetros obligatorios y uno opcional por línea de comandos.

- El primero, llamado size representa la dimensión del tablero (tableros cuadrados).
- El segundo, llamado numInitial contiene el número de baldosas no vacías en el tablero en la configuración inicial.
- El tercero, llamado seed, contiene el valor de la semilla usada para el comportamiento seudoaleatorio del juego. Este parámetro proporciona un control del comportamiento del programa, por ejemplo, para repetir exactamente una misma ejecución. Ten en cuenta que si no se dispone de este parámetro, cada ejecución será diferente, dificultando así el proceso de depuración del código.

Inicio del juego

Al lanzar la aplicación, se mostrará la configuración inicial del juego en la consola (salida estándar). Esto es el tablero de tamaño size × size con las numInitial baldosas con valor 2 ó 4 en posiciones aleatorias. El valor 2 ó 4 también se ha elegido de forma aleatoria, dando más peso al valor 2 (90 %) frente a un 10 % para el valor 4.

Por otro lado, se mostrará el valor del marcador al que llamaremos **Score**, cuyo valor es 0 en la configuración inicial.

Finalmente, se mostrará también el valor máximo de las baldosas del tablero **Highest**, que puede ser 2 ó 4.

Modo de juego

A continuación, se le preguntará al usuario qué es lo que quiere hacer, a lo que podrá contestar una de las siguientes alternativas:

- move <direction>: Este es un comando de realización de un movimiento en la dirección indicada, que puede ser up, down, left o right. Cada movimiento tiene el comportamiento descrito en la sección anterior (*desplazar*, *fusionar*, *nuevo valor*).
- reset: Este comando permite reiniciar la partida, llevando al juego a la configuración inicial como se describe en la Sección 2.1.
- help: Este comando solicita a la aplicación que muestre la ayuda sobre cómo utilizar los comandos. Se mostrará una línea por cada comando. Cada línea tiene el nombre del comando seguida por un punto y una breve descripción de lo que hace el comando.
- exit: Este comando permite salir de la aplicación, mostrando previamente el mensaje "Game Over".

Observaciones al modo de juego:

- La aplicación debe permitir comandos escritos en minúscula y mayúscula o mezcla de ambos.
- Después de la ejecución de los comandos que modifican el estado del juego (move <direction> y reset) se debe mostrar la nueva configuración del tablero en la consola.

- Cada vez que se muestre el estado del tablero, también se debe mostrar la baldosa de mayor valor en el tablero **Highest** y el valor del marcador **Score**.
- Después de cada movimiento, la aplicación debe comprobar si aún quedan movimientos válidos, esto es, mientras haya baldosas vacías o cuando no habiendo vacías, haya baldosas adyacentes con el mismo valor. Si no quedan movimientos válidos, la partida se da por finalizada e imprimirá el mensaje "Game Over" por consola.
- La partida también termina si se consigue una baldosa con valor 2048, en cuyo caso se imprimirá el mensaje "Well done!" por consola.

3. Implementación

Para lanzar la aplicación se ejecutará la clase `tp.pr1.Game2048`, por lo que se aconseja que todas las clases desarrolladas en la práctica estén en el paquete `tp.pr1` (o subpaquetes suyos). Para implementar la práctica necesitarás, al menos, las siguientes clases:

- Direction: enumerado que representa la dirección e un movimiento. Contiene las constantes de enumeración UP, DOWN, LEFT y RIGHT.
- Position: clase que permite representar posiciones del tablero.
- MoveResults: clase que permite representar los resultados de la ejecución de un movimiento.
- Cell: clase que permite representar cada una de las baldosas del tablero. Esta clase ofrece (al menos) los siguientes métodos:

  1. Método accedente y mutador para consultar y modificar el valor de una baldosa.
  2. Método que permita comprobar si una baldosa está vacía:

     ```
     public boolean isEmpty()
     ```

  3. Método que implementa la acción *Fusionar* de dos baldosas (descrita en la Sección 1). Este método comprueba si se puede realizar la combinación de una baldosa y la baldosa vecina neighbour pasada como argumento. Si es posible, implementa la fusión y devuelve como resultado el valor true. En otro caso, el método devuelve el valor false.

     ```
     public boolean doMerge(Cell neighbour)
     ```

- Board: clase que almacena el estado de un tablero y proporciona métodos necesarios para la gestión de dicho estado. La implementación será genérica, pues permitirá tableros de cualquier dimensión. Como atributos privados se necesitarán al menos:

  ```
  private Cell [ ][ ] _board;
  private int _boardSize;
  ```

  Estos atributos permitirán almacenar el estado del tablero y su tamaño.

  La clase contendrá una constructora con un único argumento que fija la dimensión del tablero _boardSize, e instancia el atributo _board. Adicionalmente, la clase Board ofrece (al menos) los siguientes métodos:

1. public void setCell(Position pos, int value), método usado para modificar el valor de la baldosa de la posición pos con el valor value.
2. public MoveResults executeMove(Direction dir), método que ejecuta las dos primeras acciones de un movimiento (***desplazar*** y ***fusionar***) en la dirección dir del tablero. Devuelve un objeto con los resultados del movimiento.

Recuerda implementar el método público public String toString() para imprimir el estado del tablero _board. Recomendamos el uso de las siguientes variables locales al método:

```
int cellSize = 7;
String space = ;
String vDelimiter = "|";
String hDelimiter = ";
```

En el **Anexo a la Práctica 1** proporcionamos la implementación de la clase MyStringUtils. Dicha clase proporciona 2 métodos para formatear strings y facilitar el "pretty-print" del tablero. Su uso no es obligatorio.

- Game: para representar el estado de una partida (tablero, dimensión del tablero, número de baldosas en la configuración inicial y un objeto de la clase java.util.Ramdom para gestionar el comportamiento aleatorio del juego). Entre sus atributos privados se encuentran:

```
private Board _board;
private int _size;
private int _initCells;
private Ramdom _myRandom;
```

La clase contendrá una constructora de tres argumentos (dimensión, baldosas iniciales y un objeto de la clase Ramdom) que fija cada uno de los atributos privados, e instancia el atributo _board.

Adicionalmente, la clase Game ofrece (al menos) los siguientes métodos:

1. public void move(Direction dir), método que ejecuta un movimiento en la dirección dir sobre el tablero, actualizando el marcador de puntos **Score** y el valor más alto de las baldosas del tablero **Highest**.
2. public void reset(), método responsable de inicializar la partida actual.

- Controller: clase para controlar la ejecución del juego, preguntando al usuario qué quiere hacer y actualizando la partida de acuerdo a lo que éste indique. La clase Controller necesita al menos dos atributos privados:

```
private Game game;
private Scanner in;
```

El objeto in sirve para leer de la consola las órdenes del usuario. Esta clase implementa el método público public void run() que realiza la simulación del juego. Concretamente, mientras la partida no esté finalizada, solicita una orden al usuario (move, reset, exit) y la ejecuta invocando a algún método de la clase Game a través de su atributo game, excepto para el comando help que imprime la ayuda del juego.

- Game2048: Es la clase que contiene el método main de la aplicación. En este caso el método main lee los valores de los parámetros de la aplicación (2, quizá 3), crea una nueva partida (objeto de la clase Game), crea un controlador (objeto de la clase Controller) con dicha partida, e invoca al método run del controlador.

Observaciones a la implementación:

- Durante la ejecución de la aplicación solo se creará un objeto de la clase Controller. Lo mismo ocurre para las clases Game (que representa la partida en curso y solo puede haber una activa) y Board (que representa un tabero asociado a la partida).
- En el **Anexo a la Práctica 1** se proporciona parte de la implementación de la clase ArrayAsList basada en arrays. Se puede utilizar por ejemplo para almacenar las posiciones libres del tablero, facilitando la elección aleatoria de una baldosa vacía en el tablero.

El resto de información concreta para implementar la práctica será explicada por el profesor durante las distintas clases de teoría y laboratorio. En esas clases se indicará qué aspectos de la implementación se consideran obligatorios para poder aceptar la práctica como correcta y qué aspectos se dejan a la voluntad de los alumnos.

4. Ejemplo de ejecución

A continuación mostramos algunos ejemplos de ejecución para un tablero 6×6 y 3 baldosas iniciales.

```
$ java tp.pr1.Game2048 6 3

 -------------------------------------------------
|       |       |       |       |   2   |   2   |
 -------------------------------------------------
|       |       |       |       |       |       |
 -------------------------------------------------
|       |       |       |       |       |       |
 -------------------------------------------------
|       |       |       |       |       |       |
 -------------------------------------------------
|       |       |       |       |       |       |
 -------------------------------------------------
|       |       |       |   2   |       |       |
 -------------------------------------------------
 highest: 2          score: 0

Command > move right

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

```
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |   2   |
 ---------------------------------------------
 highest: 4          score: 4

Command > move LEFT

 ---------------------------------------------
|   4   |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |   2   |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|   2   |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|   2   |       |       |       |       |       |
 ---------------------------------------------
 highest: 4          score: 4

Command > MOve DOwn

 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |   2   |
 ---------------------------------------------
|   4   |       |       |       |       |       |
 ---------------------------------------------
|   4   |       |       |       |       |   2   |
 ---------------------------------------------
 highest: 4          score: 8

Command > MOVE down

 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|       |       |       |       |       |       |
 ---------------------------------------------
|   8   |       |       |   2   |       |   4   |
 ---------------------------------------------
```

```
     highest: 8          score: 20

Command > help
Move <direction>: execute a move in one of the four directions, up, down, left, right
Reset: start a new game
Help: print this help message
Exit: terminate the program

Command > reset

     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |   2   |       |       |       |   2   |       |
     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |   2   |       |
     ---------------------------------------------
     highest: 2          score: 0

Command > MoVe rIgHt

     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |   4   |
     ---------------------------------------------
    |       |   2   |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |       |
     ---------------------------------------------
    |       |       |       |       |       |   2   |
     ---------------------------------------------
     highest: 4          score: 4

Command > ExiT
Game over
```

La siguiente ejecución muestra algunas situaciones erróneas.

```
$ java tp.pr1.Game2048 6 3

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

```
     |       |       |       |       |   4   |       |
      -----------------------------------------------
     |       |       |       |       |       |       |
      -----------------------------------------------
     |       |       |       |       |       |       |
      -----------------------------------------------
      highest: 4          score: 0

Command > move right

      -----------------------------------------------
     |       |       |       |       |       |   2   |
      -----------------------------------------------
     |       |   2   |       |       |       |   2   |
      -----------------------------------------------
     |       |       |       |       |       |       |
      -----------------------------------------------
     |       |       |       |       |       |   4   |
      -----------------------------------------------
     |       |       |       |       |       |       |
      -----------------------------------------------
     |       |       |       |       |       |       |
      -----------------------------------------------
      highest: 4          score: 0

Command > move
Move must be followed by a direction: up, down, left or right

Command > move out
Unknown direction for move command

Command > move on out
Unknown command

Command > mv up
Unknown command

Command > what is going on?
Unknown command

Command >
Unknown command

Command > quit
Unknown command

Command > exit
Game over
```

5. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica. El fichero debe

tener al menos el siguiente contenido[1]:

- Directorio `src` con el código de todas las clases de la práctica.
- Fichero `alumnos.txt` donde se indicará el nombre de los componentes del grupo.

[1]Puedes incluir también opcionalmente los ficheros de información del proyecto de Eclipse

6. Anexo a la Práctica 1

```
package tp.pr1.util;

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);
  }
}
```

```
package tp.pr1.util;

import java.util.Random;

public class ArrayAsList {

  // The rest of the code for the ArrayAsList class is to be added here.
  // The other methods of this class will not be static .


  // This method is static in order to be similar to the "shuffle ()"
  // method of the standard library class "Collections ".
  public static void shuffle(ArrayAsList list, Random random) {
    for (int i = list.size(); i > 1; i--) {
      swap(list.arrayAsList, i - 1, random.nextInt(i));
    }
  }

  // This method is static in order to be similar to the "shuffle ()" method.
  public static Object choice(ArrayAsList list, Random random) {
    return list.get(random.nextInt(list.size()));
  }

  private static void swap(Object[] anArray, int i, int j) {
      Object temp = anArray[i];
      anArray[i] = anArray[j];
      anArray[j] = temp;
  }
}
```