Tarea UD 4. Polimorfismo. Clases abstractas, interfaces.

Contenido

INTRODUCCIÓN

WordleSolver

Es importante repasar y afianzar los conceptos trabajados durante el primer trimestre y añadir nuevas funcionalidades para avanzar en los conocimientos de programación en Java. Así, la **base de la presente tarea es el examen de la primera evaluación, incorporando la creación de interfaces/clases abstractas y mejorando la funcionalidad de dicha aplicación** (restringida por el tiempo limitado durante el examen).

En la aplicación se intentará dar ayuda para resolver el juego del **Wordle** que tiene el New York Times y que se ha realizado durante las tres últimas tareas (ved solución planteada a cada una de ellas):

https://www.nytimes.com/games/wordle/index.html (implantación del juego en el New York Times).

En el anexo I puedes consultar una descripción del juego, que ya hemos realizado en tareas anteriores.

Cada **letra del juego tiene 4 posibles estados**:

- Correcta (***CORRECT***), que aparecerá en verde: cuando la letra de la palabra a adivinar está en la posición indicada.
- Válida (***VALID***), que aparecerá en amarillo: cuando la letra está en la palabra, pero en otra posición.
- Mala (***BAD***), que aparece en gris: cuando la letra no aparece en la palabra.
- Deseleccionada (***DESELECT***): cuando la letra todavía no ha sido seleccionada.

Para resolver el juego **se dispondrá de un diccionario de palabras en un archivo de texto** (***english.txt***), proporcionado en fichero anexo (también podéis hacer uso de diccionarios descargados de Internet, siempre que cada palabra aparezca en una línea independiente).

Se **evaluará** la tarea en dos partes: por un lado, la parte relacionada con la **implantación/ejecución del programa** (que debe comprobarse su funcionamiento mediante capturas de pantalla) **y el código relacionado con declaración de las interfaces/clases abstractas y métodos de mejora**.

LetterStatus (enumeración)

LetterStatus es una enumeración con los posibles estados válidos de las letras del Wordle: *CORRECT('c')*, *VALID('v')*, *BAD('b')*, *DESELECT('d')*.

La enumeración contiene:

- Un atributo privado denominado ***caracter*** de tipo *char*, que indica la letra/carácter con la que se codifica el estado (cuando se teclee una *b*, por ejemplo, se indicará que el estado es BAD).
- Un **constructor (no público) que recoge el carácter** y lo asigna al atributo.
- Métodos de ***tipo get* y *set* para el carácter**.
- Método ***public static** LetterStatus **getLetterStatus(**char **c)***: devuelve el *LetterStatus* asociado al carácter que recoge como argumento.
    - *Por ejemplo: LetterStatus,getLetterStatus('c'); debe devolver CORRECT.*

Letter implementa Comparable<Letter> (clase)

Letter es una clase que identifica la letra del juego, con su ***carácter***, su ***estado*** (*CORRECT*, *VALID*, *BAD*, *DESELECT*) y su ***posición***. Si no está seleccionada la por defecto (DEFAULT_POSITION).

Debe **implementar la interface *Comparable<Letter>***, comparando con otro objeto de tipo *Letter*.

La interface Comparable tiene un único método abstracto que devuelve 0 cuando son iguales, >0 cuando es mayor y <0 cuando es menor:

```
public int compareTo(Letter n);
```

CONSTANTES

- *DEFAULT_POSITION*: con el valor -1, representa la posición por defecto que se usa cuando la letra no ha sido seleccionada (*DESELECT*). *Ojo: cuando es incorrecta (BAD) sí importa la posición, pues cuando se indica una palabra con una letra en posición correcta, podría marcarse como mala en otra posición y no como válida.*

ATRIBUTOS

- ***letter***: **carácter** que representa la letra de la palabra.
- ***estado***: de tipo ***LetterStatus***, indica el estado que tiene la letra.
- ***posicion***: de tipo **entero**, representa la posición de la letra dentro de la palabra. La posición de la letra será la posición por defecto si el estado es DESELECT.

CONSTRUCTORES

- ***Constructor que recoge la letra*** (carácter)***, la posición y el estado***:

  Crea una letra asignando el carácter, la posición y el estado.

  Si el carácter no es una letra (¡Character.isLetter(letra)) le asigna el (carácter) 0 (sin comillas); si es una **letra, la guarda en mayúscula** por medio del método*: Character.toUpperCase().*

  Si el estado es DESELECT o la posición es un número negativo le asigna la posición por defecto, *DEFAULT_POSITION*.

- ***Constructor*** que recoge el carácter de la ***letra***:

  Crea una *Letter* recogiendo el carácter, asignando el estado a *LetterStatus.DESELECT* y la posición a *DEFAULT_POSITION*. Al igual que en el constructor anterior, guarda la letra en mayúscula si es una letra correcta.

MÉTODOS

- Métodos de tipo ***get*** para cada atributo.
- Método de tipo ***set*** para la **posición** y **estado**. El método set de posición comprueba que no sea menor que cero. Si es menor que cero le pone la posición por defecto (*DEFAULT_POSITION*).
- ***equals(Object objeto)***: método que sobrescribe el método de *Object* y devuelve verdadero cuando tienen el **mismo carácter y la misma posición.**
- *public int* ***compareTo(Letter*** *letra)*: implantación del método de la interface. Compara las letras por el carácter que guardan; en el caso de tener el mismo carácter lo hace por la posición. *Ayuda: simplemente devuelve la resta de los caracteres de los dos objetos si no son iguales y la resta de las posiciones en el caso de tener el mismo carácter.*
- ***toString()***: devuelve la cadena que representa al objeto con la letra en mayúscula, la posición entre corchetes y el estado entre paréntesis, de acuerdo con el siguiente formato de cadena: "*Letra [posición] (estado)". Ejemplos (cada línea es un objeto):*

```
A (BAD)
D (BAD)
O[2] (CORRECT)
C[1] (VALID)
C[3] (CORRECT)
K[4] (CORRECT)
W (BAD)
S (BAD)
```

Solution (Clase abstracta) *implementa Comparable<Solution>*

Esta clase abstracta representa la solución a algún juego. Como **no tiene sentido crear una *Solution*** (una solución) **sin indicar el juego concreto (por ejemplo, para el Wordle será una palabra**, de tipo *String*), dicha clase debe ser **declarada como abstracta**.

Únicamente dispone de un atributo denominado ***valor***, de tipo **entero**, que contiene el valor de la solución para poder ordenarla con respecto al resto de soluciones y que será empleado por la implementación del método *compareTo* (este método es el que emplean los métodos de ordenación para saber ordenar los *arrays* o las listas de objetos).

ATRIBUTO

- ***valor***: *int* con el valor de la solución.

CONSTRUCTOR

Sin constructor explícito; sólo el constructor por defecto.

MÉTODOS

- Métodos ***getValor()*** y ***setValor(int valor)*** para el atributo valor. El método *setValor* no debe ser "public" y debe declararse como "*protected*" (accesible por clases del mismo paquete o subclases de otros paquetes).
- ***setValor()***: **método abstracto** que no recoge ningún argumento y le asigna un valor a la solución.
- ***public int compareTo(Solution*** *solucion)*: implantación del método de la interface *Comparable*. Devuelve la comparación de los valores de las soluciones (0 si tienen el mismo valor, >0 si el valor es mayor y <0 si el valor es menor).

WordleSolution extends *Solution*

Clase que hereda de ***Solution*** *y* representa **una posible solución al juego de Wordle**. Una posible solución al juego de Wordle es una cadena con la **palabra**, de tipo *String*.

Por ello, además del atributo heredado (al que no tiene acceso de modo directo) la cadena de texto que representa a la palabra que es una posible solución, **word**. Recuerda que debe implantar el método abstracto *setValor()*, que le da un valor (peso) a la posible solución.

ATRIBUTO

- ***word***: *String* con la palabra de la solución.

CONSTRUCTOR

Sin constructor explícito; sólo el constructor por defecto.

MÉTODOS

- Métodos ***getWord()*** y ***setWord(****String word****)*** para el atributo word.
- ***setValor()***: **implantación del método abstracto** que no recoge ningún argumento y le asigna un valor a la solución. **Implementa la estrategia que consideres más adecuada para darle un valor (peso) a la solución**. Puedes darle un valor que consideres, por ejemplo:
    - Un valor aleatorio: la mejor solución se obtendrá de modo aleatorio.
    - Un valor que se corresponda con el número de letras diferentes: las seleccionará antes la palabra cuyas letras sean todas distintas y, así, probar con más variedad de letras.
    - …
- ***public String toString()***: sobreescribe el método *toString* para que devuelva la cadena con la palabra en mayúsculas si no es nula o la cadena vacía en caso contrario.

GameSolver (Clase abstracta)

Esta clase abstracta representa al tipo de objeto que permite resolver juegos. Como **no tiene sentido crear un *GameSolver* sin indicar que resolución concreta**, dicha clase debe ser **declarada como abstracta**. Únicamente contiene la lista de ***soluciones*** del juego (de tipo *ArrayList<Solution>*) y el ***nombre*** del juego.

Nota: por supuesto, se podrían guardar las soluciones en un arraye implementar así la clase, pero al hacerlo con un array, como tiene tamaño fijo indicado en la creación, si añades una solución y no hay espacio en el array se debería crear una copia de mayor tamaño. Además, habría que controlar los espacios nulos. Más lioso.

ATRIBUTOS

- ***soluciones***: *ArrayList* de tipo *<Solution>* con las soluciones del juego.
- ***nombre***: cadena con el nombre del juego.

CONSTRUCTOR

Un único constructor que **recoge (sólo) el nombre del juego**, asignándolo al atributo y creando un objeto de tipo *ArrayList* para almacenar las soluciones.

MÉTODOS

- Métodos ***get*** para cada atributo (soluciones y nombre).
- ***addSolucion***: recoge una *Solution* y la añade a la lista. Puedes hacer que devuelva un booleano si lo deseas (el método *add* de ArrayList devuelve un boolean).
- ***clearSolucions()***: limpia la lista de soluciones.
- ***solve()***: método **abstracto** (no se implanta) que será implantado en las subclases para que resuelvan el juego.
- ***toString()***: sobrescribe el método de Object, devolviendo una cadena con las lista de soluciones ordenadas, una por línea. *IMPORTANTE: por temas de eficiencia, debe **emplearse StringBuilder** para concatenar cadenas. Para ordenar listas se hace con el método Corrections.sort(solucions).*

WordleSolver extends GameSolver (clase)

Clase que representa el tipo de objeto para resolver *Wordle*. Contiene el array de Letter, ***letras***, que se han escrito para resolver el juego. Además, el nombre del archivo con el diccionario, ***wordsFile***, y un ArrayList<String> con las palabras del diccionario (cargadas desde el archivo), ***diccionario***.

CONSTANTES

- ***SOLVER_NAME*** = "*Wordle Solver*". Nombre del *GameSolver*.
- ***DEFAULT_SIZE*** = 26: tamaño del *array* de letras por defecto (en realidad, en el peor de los casos, precisamos uno de mayor tamaño, pero es más que suficiente).
- ***DEFAULT_WORDS_FILE***: nombre de archivo en el que están guardadas las palabras. Su valor es *"english.txt"*.

ATRIBUTOS

- ***letras***: array de *Letter* con las letras de las que conocemos su estado para resolver.
- ***wordsFile***: nombre del archivo en el que está el diccionario.
- ***diccionario***: de tipo *ArrayList<String>* con la lista de palabras del diccionario.

CONSTRUCTOR

Un constructor por defecto que crea un *GameSolver* con el **nombre**

SOLVER_NAME (recuerda cómo se invoca al constructor de la clase padre con super), **crea un array de letras** de tamaño *DEFAULT_SIZE*, **crea la lista del diccionario** y **carga las palabras del archivo** en el diccionario. Recuerda, para escanear el contenido de un archivo:

```
Scanner sc = new Scanner(new File("ruta_al_archivo"));
```

Puedes poner el archivo en la ruta que desees.

MÉTODOS

- ***listLetters()***: muestra por pantalla la lista de letras, pero **sólo aquellas que no son nulas**.

- *public boolean* ***addLetter****(Letter letra)*: recoge una letra y la añade a al array de letras. La añade a la primera posición nula, si la puede añadir y no está dentro.

- *public boolean* ***addLetters****(String palabra, String estado)*: recoge una palabra y el estado (del mismo tamaño que la palabra) y añade las letras al objeto.

  El estado está codificado mediante letras, por ejemplo: *vbbcc,* en el que se indica que la primera letra es válida, la segunda es mala y las dos últimas correctas. Debe recorrer los caracteres de la palabra y a añadir las letras con su correspondiente estado a la lista de letras (haz uso del método ***addLetter*** anterior).

- *public boolean* ***checkPalabra****(****String*** *palabra)****:*** recoge una palabra y comprueba si se ajusta al patrón definido por las letras:
  - Recorre el array de letras.
  - Si la letra no es nula y el estado no es DESELECT obtiene el estado:
    - Si el estado es CORRECT comprueba si la palabra tiene la letra en esa posición.
    - Si el estado es VALID comprueba qua la palabra contiene la letra pero que no está en la posición de la palabra.
    - Si el estado es BAD comprueba que la palabra no tenga esa letra.
  - Si no se cumple alguna de las anteriores condiciones devuelve *false*. Si se cumplen, sigue comprobando la siguiente letra del array.

- ***solve()***: implementación del método abstracto de la clase padre. Limpia la lista de soluciones (*clearSolucions()*), recorre cada palabra del diccionario y comprueba la palabra (*checkPalabra*). Si es una posible solución la añade a las soluciones (*addSolucion*).

JWordleSolver (aplicación)

Crea una aplicación sencilla que:

- Cree un *WordleSolver* con el constructor por defecto.

- Vaya pidiendo palabras y el estado de cada una de ellas, añadiéndolas al WordleSolver por medio del método ***addLetters***, hasta que se teclee una opción para salir.

  Después de introducir cada palabra/estado debe resolver el WordleSolver y mostrar la lista de soluciones posibles, ordenadas de menor a mayor (ya lo debe hacer el método *solver* o el método *toString()*).

Entrega

Entrega en un archivo comprimido el código fuente de cada una de las clases/enumeraciones (puede ser el proyecto de Netbeans). Además, debe incorporar un PDF con las capturas de pantalla y ejecución del programa, resolviendo el juego con dos días/palabras diferentes.

Entréguese un archivo comprimido de extensión ZIP o 7Z con:

- Código fuente del proyecto completo, comprimido (no se evalúa pero se emplea para comprobar autoría y control).
- **PDF con las capturas de pantalla de la ejecución del proyecto** (resolviendo dos días el juego del New York Times). WordleSolver.pdf
- **PDF con el código de las clases abstractas** y la implantación de la *Solution* del juego Wordle: ***GameSolver.java***, ***Solution.java*** y ***WordleSolution.java*** (implantación de la clase abstracta *Solution.java*).

El formato de documento debe ser:

*Apellido1Apellido2NombreTarefa04PROG.zip (o *.7z)*

Por ejemplo:

WittgensteinLudwigJosef1Tarefa04PROG.zip (sólo tiene un apellido)

Evaluación

GameSolver**.java*, ***Solution**.java* y ***WordleSolution**.java*: ***7,5 puntos.

Implantación completa del juego con especial interés en el método setValue(): ***2,5 puntos.***

Como puede comprobar, la realización de las tres clases es más que suficiente para aprobar la tarea. A mayores, se recomienda la realización completa del programa para practicar todo lo visto hasta este momento.

Anexo I. Descripción del juego

El juego consiste en adivinar una palabra de 5 letras, en la que el jugador tiene 6 intentos para realizarlo.

Cada jugador debe introducir una palabra de 5 letras válida en cada intento.

El color de cada letra en los sucesivos intentos indica cuán aproximado está el jugador de la palabra a adivinar:

- *Si la letra introducida coincide con la posición de la palabra a adivinar aparecerá en VERDE.*
- *So la letra introducida aparece en la palabra a adivinar, pero en otra posición aparecerá en AMARILLO.*
- *Si la letra introducida no aparece en la palabra a adivinar aparecerá en GRIS.*

W is in the word and in the correct spot.

I is in the word but in the wrong spot.

U is not in the word in any spot.

Para resolver el juego, se trata de añadir letras indicando el estado de cada una de ellas y seleccionar las palabras que se ajustan a esas reglas.