

Tarea UD 3. Clases y objetos. POO. Herencia. Arrays (II).

INTRODUCCIÓN

WordLe: **WordleWord** y **WordleWordSizeException**

En esta parte de la tarea necesitaremos la enumeración y la clase creadas anteriormente (**WordleColor** y **WordleLetter**) (que será proporcionada una vez finalice el período de entrega de la tarea anterior), pues crearemos una clase llamada **WordleWord** que precisa de la clase **WordleLetter**, que a su vez precisa de la enumeración.

Además, crearemos una excepción, **WordleWordSizeException**, que hereda de **Exception** y lanzaremos cuando el tamaño de la palabra no se ajuste un tamaño válido de las palabras de nuestro juego.

Crear una excepción propia es algo tan sencillo como **heredar de *Exception* y declarar los métodos que consideremos necesario**. También podemos añadirle atributos informativos, como el tamaño que ha lanzado la excepción o algún método que consideremos necesario.

En la clase **WordleWord** practicaremos con **arrays** de objetos de tipo **WordleLetter**, además de declarar constantes (**static final**). Aunque no lo precisemos en la implementación final juego de manera directa, haremos que la clase **WordleWord** **implemente la interface *Comparable<WordleWord>***, que tiene un único método y nos permitirá ordenar las palabras de tipo **WordleWord** en orden alfabético por medio del **método estático *sort*** de la clase **Arrays**, entre otras posibilidades.

Haremos una pequeña aplicación de ejemplo para comprobar el correcto funcionamiento de la clase, creando varias palabras y ordenándolas.

Nota: sobre excepciones, podéis ver un ejemplo en el apartado 7.2 de “Utilización de objetos” (primera parte de esta unidad). Además, disponéis del formato en el apartado 4.2 de “Desarrollo de clases” (segunda parte de la unidad).

WordleWordSizeException

WordleWordSizeException es una clase que **hereda de Exception**, una excepción de Java. Las excepciones son **errores en tiempo de ejecución** que se **lanzan** (en forma de objetos) cuando se **produce un error durante la ejecución de un programa**. Si no se capturan hará que el programa termine de forma errónea.

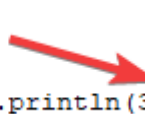
Las **excepciones** lanzadas (por programación o por la máquina virtual) son objetos que **heredan de Exception** (veremos que algunas excepciones no precisan ser tratadas y otras son de obligada captura).

En la siguiente imagen puede verse un ejemplo de una excepción producida por una división entre 0, en la que el programa termina de manera abrupta por producirse esa excepción:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.pepinho.programacion.tarea03.Probas.main(Probas.java:28)
Command execution failed.
org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit value: 1)
    at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor.java:404)
    at org.apache.commons.exec.DefaultExecutor.execute (DefaultExecutor.java:166)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:982)
    at org.codehaus.mojo.exec.ExecMojo.executeCommandLine (ExecMojo.java:929)
```

Puede verse el tipo de excepción (*java.lang.ArithmeticException*) y la línea en la que se ha producido (línea 28). Código que ha producido la excepción:

```
26 |         int n = 0;
27 |
28 |         System.out.println(3/n);
```



En este ejemplo, el error podría tratarse de dos modos: (a) evitando que el divisor sea cero o (b) capturando/tratando la excepción.

(a) Evitando mediante código que se produzca la excepción:

```
int n = 0;
if (n != 0) {
    System.out.println(3 / n);
} else {
    System.out.println("El divisor es cero");
}
```

Salida:

```
El divisor es cero
```

(b) Capturando la excepción:

```
int n = 0;
try {
    System.out.println(3 / n);
} catch (ArithmeticException e) {
    System.out.println("El divisor es cero");
}
```

Salida:

```
El divisor es cero
```

En ambos casos la salida es la misma, aunque en este caso, cuando se puede controlar la excepción desde el programa es más eficiente controlarlo por código. Ejemplos:

(1) Objeto nulo, ***NullPointerException***:

```
String nome = null;
if (nome != null) {
    System.out.println(nome.toUpperCase());
}
```

Este tipo de comprobaciones son **muy importantes** y ya las hemos hecho a lo largo del curso, pues **si el objeto no ha sido creado terminaría el programa al invocar a un método de un objeto nulo**.

Por ejemplo, en la salida de un método *get* que devuelva el nombre en mayúsculas:

```
return (nome != null) ? nome.toUpperCase() : " ";
```

(2) Aritméticas: ***ArithmeticException***, por ejemplo, al dividir entre 0.

(3) Índice excede el tamaño de una cadena: ***StringIndexOutOfBoundsException***

```
String nome = "Ludwig Wittgenstein";
System.out.println(nome.charAt(100));
```

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 100 out of bounds for length 19
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:55)
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:52)
```

Podría haberse evitado con código adecuado, comprobando que no sea nulo y que el tamaño sea mayor que 100:

```
String nome = "Ludwig Wittgenstein";  
if (nome != null && nome.length() > 100) {  
    System.out.println(nome.charAt(100));  
}
```

- (4) Índice fuera del rango de un array, ***IndexOutOfBoundsException***. De modo similar al anterior, pero evitando que el índice del array exceda su tamaño:

```
String[] nomes = {"Ludwig Wittgenstein", "Murnau"};  
if (nomes != null && nomes.length > 3) {  
    System.out.println(nomes[2]); // Tercer elemento del array  
}
```

LANZAR UNA EXCEPCIÓN

Las excepciones las puede generar o *lanzar* la máquina virtual cuando se produce un error en tiempo de ejecución (división entre 0,...) o lanzadas por un método/constructor, diseñado por el programador, creando un objeto de ese tipo y por medio de la cláusula ***throw***. Por ejemplo:

```
if(se cumple una condición de error){  
    throw new WordleWordSizeException(...);  
}
```

En este caso se trata de **crear un tipo (de dato) de *Exception***, de error, que será lanzado por las clases de la aplicación cuando la palabra de Wordle no cumple con los posibles valores permitidos en su tamaño. Excepción que se lanza la palabra no tiene un tamaño adecuado. Dicha excepción tiene un único atributo, ***tamanho***, de tipo *int*, que representa el tamaño que ha lanzado la excepción y no se ajusta a los valores admitidos (máximo o mínimo). Ejemplo:

```
public void setNota(int nota) throws NonNotaException {  
    if (nota >= 1 && nota<=10) {  
        this.nota = nota;  
    } else {  
        throw new NonNotaException (nota);  
    }  
}
```

Atributos

size: tamaño que ha lanzado la excepción. Se trata de un tamaño no válido (por ejemplo, si no tiene un valor mínimo de `WordleWord`, `MIN_SIZE`).

@see `WordleWord#MIN_SIZE`

Constantes de la enumeración

serialVersionUID: atributo privado que **no se pide**, pero si se desea implantar podéis echarle un vistazo a: <https://www.baeldung.com/java-serial-version-uid> y así vamos aprendiendo algo nuevo.

Constructores

Un constructor **por defecto** que llama al constructor de la clase padre, de *Exception*, con la cadena *"El tamaño de la palabra no es correcto."*.

Un constructor que **recoge el tamaño** y se lo asigna la atributo *size*. Previamente debe llamar al constructor de la clase padre pasándole la cadena anterior, *"El tamaño de la palabra no es correcto."*.

Métodos de instancia

getSize(): devuelve el tamaño que ha lanzado la excepción. Se trata de un tamaño no válido para una palabra del *Wordle*.

WordleWord implements Comparable<WordleWord>

WordleWord es la clase que representa **la palabra que se muestra en el juego** de Wordle. Está formada por un **array de letras**, de tipo *WordleLetter* con las letras de la palabra. Además, para poder utilizar métodos de ordenación de *WordleWord*, debe **implantar la interface Comparable**, que tiene un único método **compareTo**.

Téngase en cuenta que la palabra tendrá un tamaño fijo una vez creada, por ese se guardan las letras como *array* de *WordleLetter*.

Atributo

letras: *array* de tipo *WordleLetter* con las letras de la palabra.

Constantes

MIN_SIZE: con valor **4**, representa el tamaño mínimo de la *WordleWord*.

DEFAULT_SIZE: con el valor **5**, representa el tamaño por defecto de una *WordleWord*.

Constructores

a) Constructor que recoge el tamaño (lanza una excepción de tipo *WordleWordSizeException*)

Recoge el tamaño de la palabra y crea una **el array de letras (de tipo WordleLetter)** de ese **tamaño**. Recuerda que los *arrays* se inicializan a los valores por defecto, *null* en este caso, por lo que una vez creado el *array* hay que crear las letras con su constructor por defecto. Algoritmo:

- Si el **tamaño** recogido es **menor que MIN_SIZE** debe **crear y lanzar** una excepción de tipo *WordleWordSizeException* empleando el constructor que recoge el tamaño.
- En caso contrario (el tamaño es al menos igual a **MIN_SIZE**) debe crear el *array* de *WordleLetter* de ese tamaño y crear cada una de las letras, de tipo *WordleLetter*, con el constructor por defecto de *WordleLetter*.

b) Constructor por defecto (lanza una excepción de tipo *WordleWordSizeException*)

Crea una **el array de letras de tamaño DEFAULT_SIZE**.

Simplemente invoca al constructor anterior.

c) **Constructor que recoge una palabra** (lanza una excepción de tipo *WordleWordSizeException*)

Invoca al método **setLetters** (ved métodos) para que asigne la cadena con la palabra recogida al *WordleLetter*.

Métodos

setLetters: método declarado como *final* que recoge una cadena (*String*) de una **palabra** e inicia cada una de las letras **WordleWord** con los caracteres de esa palabra.

Para ello, debes **crear el array letras**, de tipo **WordleLetter**, con el tamaño de la cadena recogida (*palabra.length()*), recorrer los caracteres de la palabra recogida como argumento y creando una *WordleLetter* con cada uno de esos caracteres, poniéndola en el índice correspondiente. Emplea el constructor de *WordleLetter* que recoge un carácter.

Lanza una excepción de tipo *WordleWordSizeException* cuando el tamaño de la palabra recogida es menor que el tamaño mínimo.

addLetra: recoge una letra de tipo **LetraWordle** a la añade a la *WordleWord*.

Debe recorrer el *array* de **letras** de tipo *WordleLetter* en busca de una letra con valor **null** o con color **WordleColor.VOID**. Si existe, la pone en esa posición y devuelve verdadero. Si no hay ningún espacio devuelve false (ha llegado al final del bucle).

addLetra: método sobrecargado que recoge una letra de tipo **char** y la añade a la *WordleWord*. Debe recorrer el *array* de **letras** de tipo *WordleLetter* en busca de una letra con valor **null** o con color **WordleColor.VOID**. Si existe, la pone en esa posición y devuelve verdadero. Si no hay ningún espacio devuelve false (ha llegado al final del bucle).

Obviamente, debes crear la *WordleLetter* por medio del constructor que recoge un carácter si encuentra el null para guardar la letra. Si está vacía (*isVoid()*) debe asignarle la letra (*setLetter(letra)*).

checkWord: recoge una palabra (la que se busca) como cadena y realiza una comprobación de cada carácter de dicha palabra:

- Si el carácter aparece en la posición indicada, pone la *WordLetter* de esa posición a **WordleColor.GREEN**.
- Si el carácter aparece en otra posición pone la *WordLetter* de esa posición a **WordleColor.YELLOW**.
- Si el carácter NO aparece en la posición indicada pone la *WordLetter* a **WordleColor.GRAY**.

Algoritmo:

1. Si la palabra recogida es nula sale del método.
2. Convierte la palabra recogida a mayúscula.
3. Recorre cada *WordleLetter* de la *WordleWord*.
 - 3.1 Si la letra de la *WordleLetter* coincide en posición con la de la palabra recogida la pone en *WordleColor.GREEN*.
 - 3.2 Else Si letra de la *WordleLetter* aparece en la palabra (*word.indexOf(c) != -1*) ponemos la *WordLetter* a *WordleColor.YELLOW*
 - 3.3 En el resto de los casos ponemos la *WordleLetter* a *WordleColor.GRAY* (no ha aparecido).

isVoid(): devuelve **true** cuando todas las letras están vacías, cuando tienen el carácter 0. Llámese al método *isVoid()* de *WordleLetter*.

getWord(): devuelve la palabra (como *String*) que está formada por la concatenación de todas las letras. Emplea *StringBuilder* por temas de eficiencia para concatenar los caracteres de la *WordleWord*.

compareTo: implementación del método *compareTo* de la interface *Comparable<WordleWord>*, comparando por la palabra (cadena) que contiene (obtenida por el método *getWord()* con la del objeto recogido como parámetro.

Haz uso del método *compareToIgnoreCase* de *String*.

```
@Override
public int compareTo(WordleWord ww) {
    // ... vuestro código va aquí.
}
```

toString(): representación de la palabra, *WordleWord*, como cadena. Debe emplearse *StringBuilder* para concatenar las letras de la palabra y devolver su representación como cadena (*sb.toString()*).

AppWordleWord (aplicación)

Clase de la aplicación que pida el número de palabras, cree un array de *WordleWord* de ese tamaño y vaya pidiendo palabras para crearlas por medio del constructor que recoge una cadena. Después debe mostrar las palabras, ordenarlas y volverlas a mostrar.

Además, crea un archivo *jwordleword.jar* que permita ejecutarlo.

ENTREGA

Genérese un único documento **PDF con el código del programa y las capturas de compilación y ejecución**. Además, debe proporcionar el **código fuente**, ***WordleWord.java***, ***WordleWordSizeException.java***, ***AppWordleWord.java*** y el archivo ***jwordleword.jar*** para poder ser ejecutado.

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

- PDF con formato *Apellido1Apellido2NombreTarea0302PROG.pdf*
- Código fuente: ***WordleWord.java***, ***WordleWordSizeException.java***, ***AppWordleWord.java***
- Archivo comprimido para ejecutar: ***jwordleword.jar***.

El formato de documento debe ser:

Apellido1Apellido2NombreTarea0302PROG.zip (o *.7z)

Por ejemplo:

WittgensteinLudwigJosefTarea0302PROG.zip (sólo tiene un apellido)