Object-Oriented Programming. Exercise 4.2

Unit 4. Pre-Defined Java Classes

Exercise 1. (project prSimpleFilesWordCounting)

In this exercise we will create an application to count the number of times each of the words appears in a given text. For this, the classes `WordInText`,`WordCounter` and `WordCounterSig` will be created.

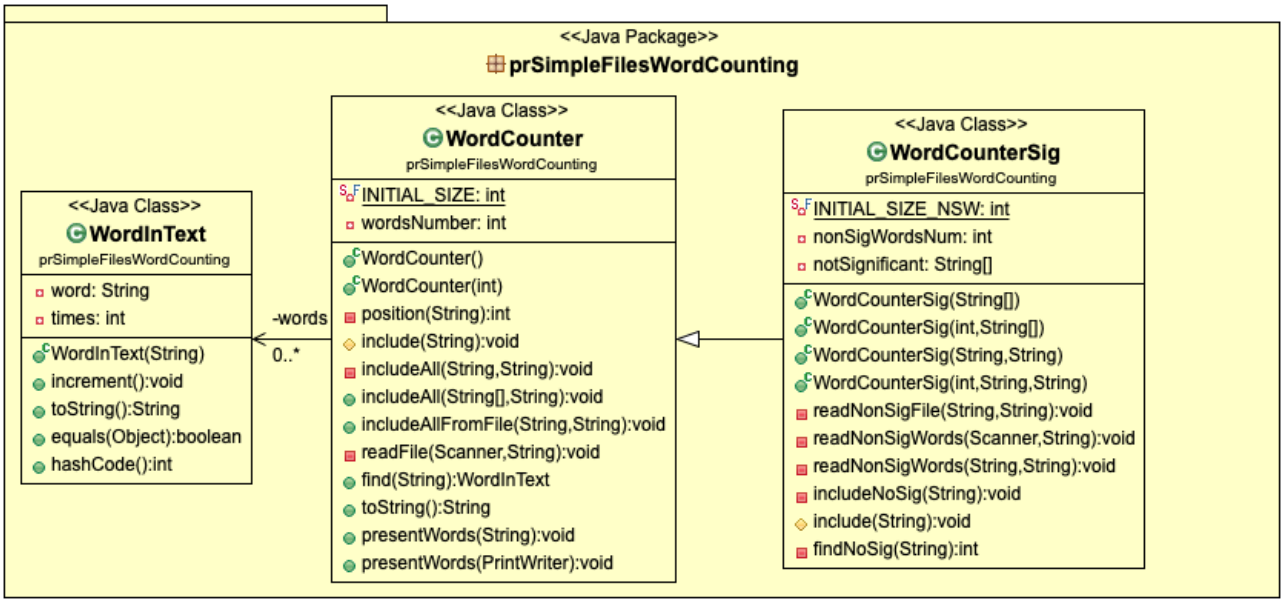

Figura 1: UML class diagram

Class `WordInText`

Create the `WordInText` class to hold information about a word (`String`) and the number of times it appears in a given text (`int`).

1. The class will have a constructor that takes a word as an argument. When a new instance is crated, the number of times the word appears is considered 1. To facilitate its later use, the word will be stored in capital letters.
2. Two objects of the `WordInText` class are equal if the words they contain match. Since they are stored in upper case, the comparison is not case sensitive. The number of occurrences of the words is not taken into account.
3. The string representation of a `WordInText` object should show the word it contains and the number of times it appears, for example,

   ```
   GORRA: 2
   ```

4. The `increment()` method increments the number of times the word appears by one.

App `MainWordInText`

Create a `MainWordInText` application to test the previous class. Two `WordInText` objects will be created in this application with the words `gorra` and `Gorra`. Then the number of occurrences of the first will be increased once, and both objects will be shown on the screen. Finally, it will be checked

if both words are the equal, indicating it on the screen. Running the application will produce the following output:

```
Word 1 = GORRA: 2
Word 2 = GORRA: 1
The words are equal
```

Class `WordCounter`

Create the `WordCounter` class that stores the words that appear in a text in an array of `WordInText`. It will also keep the number of words (`int`) stored in it at any given time.

If, in the course of operations with objects of this class, the array becomes full, it must grow so that the words provided are always fit.

1. The class will have two constructors:
   - One with no arguments, that creates the array with a default size (a constant `INITIAL_SIZE` with value 10);
   - Another with an argument of type `int` that indicates the initial size with which to create the array.

   In both cases, the initial number of words stored will be 0.
2. The private method `position(String)` returns the position where the word corresponding to the string it receives as an argument is found, or `-1` if it is not found.
3. The protected method `include(String)` increases the number of occurrences in the word count of the word that corresponds to the string it receives as an argument, if it already existed, or it includes a new word with that string otherwise.
4. The private method `includeAll(String line, String del)` includes in the word counter each of the words obtained from the string `line` using the delimiters included in `del`. For example, given the line `"this is  an example"` and with blank space being the only delimiter (`"[ ]"`), we would enter the words corresponding to the strings `"this"`, `"is"`, `"a"` and `"example"`. Note that empty strings must not be included.
5. The public method `includeAll(String[] text, String del)` includes each of the words in each of the strings in the `text` array according to the separators in the `del` string. That is, each element of the array is considered a line of a text, and each of the words in that text is included in the counter.
6. The public method `includeAllFromFile(String filename, String del)` includes in the word count all the words found in the file named `filename`. The lines in the file are considered one by one and broken into words taking into account the delimiters provided in the `del` string. This method creates an input stream (`Scanner`) and invokes the private method `readFile(Scanner sc, String del)` which reads the file line by line. The method throws a `FileNotFoundException` exception if the file is missing or cannot be opened for reading.
7. The public method `find(String)` searches the word array for the instance of `WordInText` that corresponds to the string it receives as an argument. If the word is not found it throws a `NoSuchElementException` exception.
8. The string representation of the class instances is as shown below. Use `StringJoiner` or `StringBuilder` to create the representation.

   ```
   [GUERRA: 5, TENÍA: 2, JARRA: 3, PARRA: 7]
   ```

9. Finally, the class has public methods `presentWords(String)` and `presentWords(PrintWriter)` that generate a presentation of the word counter in the format below. While the first one stores the information in the file whose name it receives as an argument, the second one receives as

parameter the output stream (of type `PrintWriter`) where to carry out the action. The method will throw a `FileNotFoundException` exception if the file is read-only and cannot be opened for writing.

```
GUERRA: 5
TENÍA: 2
UNA: 2
JARRA: 3
Y: 1
...
```

App `MainWordCounter`

Create a `MainWordCounter` application to test the previous class. In this application, an object of the `WordCounter` class is created with an array of size 5. Subsequently, it is sent a message `includeAll()` with the following parameters:

- As the first parameter, the following array:

```
String [] datos = {
    "Esta es la primera frase del ejemplo",
    "y esta es la segunda frase"
};
```

- As second parameter, the following string (white space is the only delimiter):

```
"[ ]"
```

Finally, the application will display the contents of the counter on the screen.

Running the application should produce the following output:

```
[ESTA: 2, ES: 2, LA: 2, PRIMERA: 1, FRASE: 2, DEL: 1, EJEMPLO: 1, Y: 1, SEGUNDA: 1]
```

App `MainWordCounter2`

Create a `MainWordCounter2` application that builds a word counter with the contents of the file `"quijote.txt"` and separators `"[ .,;:]+"`, and writes the counter to a file `"salida-quijote.txt"`.[1] Given the file `"quijote.txt"` provided, the output file should be like the file `"salida-quijote.txt"` also provided. If any of the two files cannot be opened or is not found, a message will be displayed on the screen indicating this.

Class `WordCounterSig`

Create the `WordCounterSig` class that represents word-counting objects that, in the inclusion procedures, discard the words considered **not significant**. The idea is to leave out of the counter words such as "and", "or", "a", "the", etc. For this, the class has, in addition to the stored words and their counters, an array of non-significant words (`notSignificant`) and the number of these non-significant words (`nonSigWordsNum`).

1. The class will have a `WordCounterSig(int, String[])` constructor that receives the initial size of the `WordInText` array and an array of `String` with the non-significant words. A second `WordCounterSig(String[])` constructor only receives the non-significant words array, and initializes the words array with the default size (`INITIAL_SIZE`). The `nonSigWordsNum` variable is initialized in both cases with the size of the array received as an argument. Non-significant words

[1] The file `quijote.txt` should be placed in the root folder of the project in the student's workspace. The output file will be created by your program on the same site. The text of the file includes accented characters, punctuation, and others that may be displayed incorrectly depending on the encoding used.

that come in that parameter array will be stored in uppercase in the `notSignificant` array to facilitate their use.

2. The class will have two additional constructors

   ```
   public WordCounterSig(String filNoSig, String del);
   public WordCounterSig(int n, String filNoSig, String del);
   ```

   to allow the non-significant words to be obtained from a file. These constructors receive a `filNoSig` parameter of type `String` with the name of the input file from which non-significant words are read, and a `del` parameter, also of type `String`, that contains the string with the delimiter characters to be used to break those words in the file. In both cases, the `notSignificant` variable will be instantiated with a `String` array of size `INITIAL_SIZE_NSW`, and the `nonSigWordsNum` variable will be suitably initialized.

   Both constructors will call the private method

   ```
   private void readNonSigFile(String filNoSig, String del)
   ```

   This will create an input stream (`Scanner`) and in turn call the private method

   ```
   private void readNonSigWords(Scanner sc, String del)
   ```

   which will carry out the reading of the file word by word, using the specified delimiters, and filling the array `notSignificant`. Like previous constructors, non-significant words will be capitalized.

3. Instances of the `WordCounterSig` class must behave like those of `WordCounter`, except that word inclusion methods must discard non-significant words.

4. In addition, the class will provide `includeNoSig(String)` and `findNoSig(String)` methods to add and search for a non-significant word.

App `Main`

Copy the provided files `datos.txt` and `fichNoSig.txt` in the project root folder in your workspace. Running the `Main` program should produce a file `output-data.txt` similar to the one provided and the following output in the terminal:[2]

```
With no main's argument...
[GUERRA: 5, TENÍA: 2, UNA: 2, JARRA: 3, Y: 1, PARRA: 7, PERRA: 6, PERO: 1, LA: 10, DE: 8, ROMPIÓ: 1,
 PEGÓ: 1, CON: 3, PORRA: 3, A: 3, OIGA: 1, USTED: 1, BUEN: 1, HOMBRE: 1, POR: 1, QUÉ: 1, HA: 1,
 PEGADO: 2, PORQUE: 1, SI: 1, NO: 2, HUBIERA: 2, ROTO: 1]

[GUERRA: 5, TENÍA: 2, JARRA: 3, PARRA: 7, PERRA: 6, PERO: 1, ROMPIÓ: 1, PEGÓ: 1, PORRA: 3, OIGA: 1,
 USTED: 1, BUEN: 1, HOMBRE: 1, POR: 1, QUÉ: 1, HA: 1, PEGADO: 2, PORQUE: 1, HUBIERA: 2, ROTO: 1]

PARRA: 7
Not found word Gorra
We repeat the execution taking reading/writing from/to file
With no main's argument...
[GUERRA: 5, TENÍA: 2, UNA: 2, JARRA: 3, Y: 1, PARRA: 7, PERRA: 6, PERO: 1, LA: 10, DE: 8, ROMPIÓ: 1,
 PEGÓ: 1, CON: 3, PORRA: 3, A: 3, OIGA: 1, USTED: 1, BUEN: 1, HOMBRE: 1, POR: 1, QUÉ: 1, HA: 1,
 PEGADO: 2, PORQUE: 1, SI: 1, NO: 2, HUBIERA: 2, ROTO: 1]

[GUERRA: 5, TENÍA: 2, JARRA: 3, PARRA: 7, PERRA: 6, PERO: 1, ROMPIÓ: 1, PEGÓ: 1, PORRA: 3, OIGA: 1,
 USTED: 1, BUEN: 1, HOMBRE: 1, POR: 1, QUÉ: 1, HA: 1, PEGADO: 2, PORQUE: 1, HUBIERA: 2, ROTO: 1]

GUERRA: 5
TENÍA: 2
UNA: 2
JARRA: 3
Y: 1
```

[2] Some carriage returns have been entered to display the output.

PARRA: 7
PERRA: 6
PERO: 1
LA: 10
DE: 8
ROMPIÓ: 1
PEGÓ: 1
CON: 3
PORRA: 3
A: 3
OIGA: 1
USTED: 1
BUEN: 1
HOMBRE: 1
POR: 1
QUÉ: 1
HA: 1
PEGADO: 2
PORQUE: 1
SI: 1
NO: 2
HUBIERA: 2
ROTO: 1

GUERRA: 5
TENÍA: 2
JARRA: 3
PARRA: 7
PERRA: 6
PERO: 1
ROMPIÓ: 1
PEGÓ: 1
PORRA: 3
OIGA: 1
USTED: 1
BUEN: 1
HOMBRE: 1
POR: 1
QUÉ: 1
HA: 1
PEGADO: 2
PORQUE: 1
HUBIERA: 2
ROTO: 1