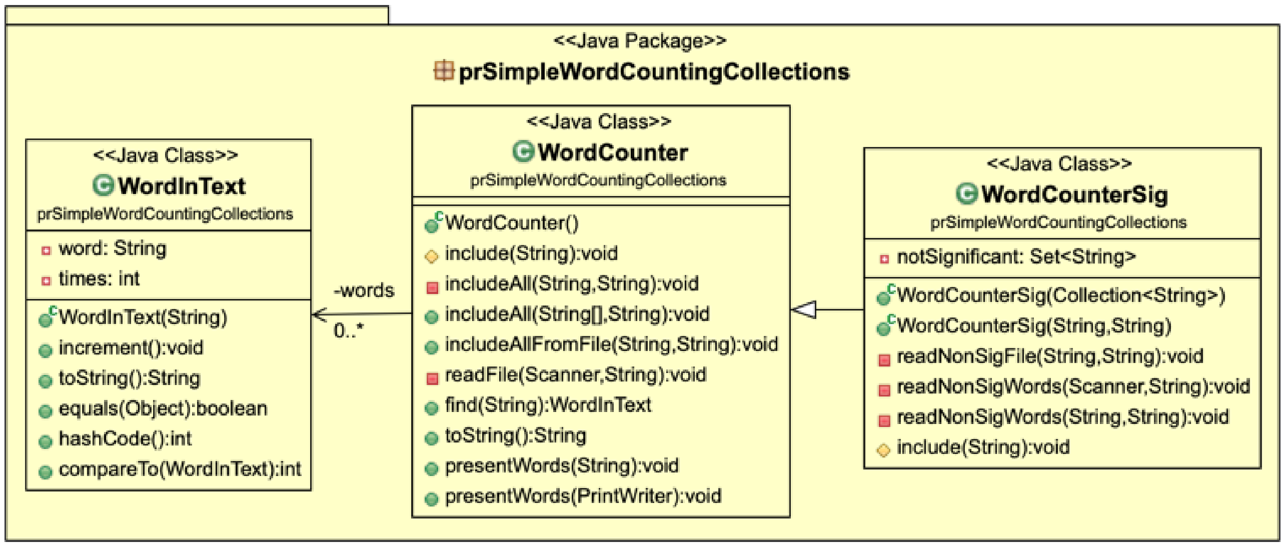
Object-Oriented Programming. Exercise 5.1

Unit 5. Collections

Exercise 1. (project `prSimpleWordCountingCollections`)

In this exercise we will create an application to count the number of times each of the words appears in a given text, as in Exercise 4.2, but with some changes. The classes `WordInText`, `WordCounter`, `WordCounterSig` and `Main` will be redefined with the following changes:



<<Java Package>>
prSimpleWordCountingCollections
<<Java Class>>
WordInText
prSimpleWordCountingCollections
word: String
times: int
WordInText(String)
increment():void
toString():String
equals(Object):boolean
hashCode():int
compareTo(WordInText):int
-words
0..*
<<Java Class>>
WordCounter
prSimpleWordCountingCollections
WordCounter()
include(String):void
includeAll(String,String):void
includeAll(String[],String):void
includeAllFromFile(String,String):void
readFile(Scanner,String):void
find(String):WordInText
toString():String
presentWords(String):void
presentWords(PrintWriter):void
<<Java Class>>
WordCounterSig
prSimpleWordCountingCollections
notSignificant: Set<String>
WordCounterSig(Collection<String>)
WordCounterSig(String,String)
readNonSigFile(String,String):void
readNonSigWords(Scanner,String):void
readNonSigWords(String,String):void
include(String):void

Figura 1: Diagrama de clases UML

Class `WordInText`

The `WordInText` class in Practice 4.2 will be extended with a natural order based on the natural order of the character string it stores (without distinguishing between upper or lower case letters).

App `MainWordInText`

Implement an application that creates two `WordInText` objects, with two different strings (`"uno"` and `"dos"`). Then make the counter of one of them increase three times and that of the other twice, and show the result of comparing them. Try two other objects that contain two equal strings, one uppercase and one lowercase (for example, `"uno"` and `"UNO"`).

Class `WordCounter`

The `WordCounter` class will now store the words that appear in a text in a **collection** (collection of objects of type `WordInText`) and will have:

1. A constructor with no argument that creates the empty word collection (specifically a set of type `TreeSet<WordInText>` will be created).
2. The `include(String)` method increments in the word counter the number of occurrences of the word corresponding to the string it receives as an argument. If the word is not on the counter, a new word will be included.

3. The private method `includeAll(String line, String del)` includes in the word counter each of the words in `line` using the delimiters included in `del`. Each one of the obtained words is included in the word counter, increasing its counter if it already exists, or creating a new one otherwise.
4. The public method `includeAll(String[] text, String del)` includes all the words found in the `text` array. Each element of the array will be considered a line of text and, on each line, the words must be separated using the delimiters included in `del`.
5. The public method `includeAllFromFile(String filename, String del)` includes all the words found in the file `filename`. The file will be read line by line, and in each line, the words will be separated by one of the delimiters included in `del`. This method creates an input stream (`Scanner`) and invokes the helper method `readFile(Scanner sc, String del)` which performs the line-by-line file reading.
6. The `find(String)` method returns the instance of `WordInText` in the word collection that matches the string passed as argument. If the word is not found the method throws a `NoSuchElementException` exception.
7. A method for the representation of objects of the class as character strings as in Exercise 4.2 will be provided. See output in the examples below.
8. The class will have methods `presentWords` such as those of practice 4.2 to generate a presentation of the file index.

(Adapt the classes `MainWordCounter` and `MainWordCounter2` from Exercise 4.2 to test the operation of the previous classes.)

Class `WordCounterSig`

The `WordCounterSig` class, whose objects discard the words considered **non-significant** in the inclusion procedures, now uses a collection of `String` to store these non-significant words (kept in **uppercase**). The class provides:

1. A `WordCounterSig(Collection<String>)` constructor that receives a collection of non-significant words — which should be capitalized — and creates an initially empty word counter.
2. A second `WordCounterSig(String nonSigFile, String del)` constructor allows you to construct a counter with the non-significant words in the `nonSigFile` file with the delimiters specified in the `del` string. This constructor will call the private method `readNonSigFile(String nonSigFile, String del)` which creates an input stream (`Scanner`) on the `nonSigFile` file with which it invokes the method `readNonSigWords(Scanner sc, String del)` to perform the word-by-word reading of the file and store them in your collection of non-significant words (as always, words will be stored in uppercase).
3. Redefine the necessary methods so that the word inclusion methods do not include non significant words in the counter.

App `Main`

Given the supplied files `data.txt` and `fichNoSig.txt` files, running the `Main` program should produce files `output-data.txt` and `outputSig.txt` similar to those provided and the following output in the terminal:[1]

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

```
We create a word counter
[A: 3, BUEN: 1, CON: 3, DE: 8, GUERRA: 5, HA: 1, HOMBRE: 1, HUBIERA: 2, JARRA: 3, LA: 10, NO: 2,
 OIGA: 1, PARRA: 7, PEGADO: 2, PEGÓ: 1, PERO: 1, PERRA: 6, POR: 1, PORQUE: 1, PORRA: 3, QUÉ: 1,
 ROMPIÓ: 1, ROTO: 1, SI: 1, TENÍA: 2, UNA: 2, USTED: 1, Y: 1]

PARRA: 7
The word Gorra does not exist

We repeat the execution taking the input from file
[A: 3, BUEN: 1, CON: 3, DE: 8, GUERRA: 5, HA: 1, HOMBRE: 1, HUBIERA: 2, JARRA: 3, LA: 10, NO: 2,
 OIGA: 1, PARRA: 7, PEGADO: 2, PEGÓ: 1, PERO: 1, PERRA: 6, POR: 1, PORQUE: 1, PORRA: 3, QUÉ: 1,
 ROMPIÓ: 1, ROTO: 1, SI: 1, TENÍA: 2, UNA: 2, USTED: 1, Y: 1]

Output to terminal:
A: 3
BUEN: 1
CON: 3
DE: 8
GUERRA: 5
HA: 1
HOMBRE: 1
HUBIERA: 2
JARRA: 3
LA: 10
NO: 2
OIGA: 1
PARRA: 7
PEGADO: 2
PEGÓ: 1
PERO: 1
PERRA: 6
POR: 1
PORQUE: 1
PORRA: 3
QUÉ: 1
ROMPIÓ: 1
ROTO: 1
SI: 1
TENÍA: 2
UNA: 2
USTED: 1
Y: 1

Output to file: salida-datos.txt

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

We repeat the execution taking the input from file
[BUEN: 1, GUERRA: 5, HA: 1, HOMBRE: 1, HUBIERA: 2, JARRA: 3, OIGA: 1, PARRA: 7, PEGADO: 2, PEGÓ: 1,
 PERO: 1, PERRA: 6, POR: 1, PORQUE: 1, PORRA: 3, QUÉ: 1, ROMPIÓ: 1, ROTO: 1, TENÍA: 2, USTED: 1]

Output to terminal:
BUEN: 1
GUERRA: 5
HA: 1
HOMBRE: 1
HUBIERA: 2
JARRA: 3
OIGA: 1
PARRA: 7
PEGADO: 2
PEGÓ: 1
PERO: 1
PERRA: 6
```

```
POR: 1
PORQUE: 1
PORRA: 3
QUÉ: 1
ROMPIÓ: 1
ROTO: 1
TENÍA: 2
USTED: 1

Output to file: salidaSig.txt
```