10 Using Java Collections; JUnit; More Sorting

In this assignment you should try to write as little code as possible - using the `Java Collections Framework` classes for getting the work done. Also, you should use `JUnit` for all tests.

Portfolio Programs: The Java Collections Framework

Finish all the work for Lab 10. (See below)

10.1 HashMap, JUnit

Finish all parts of Lab 10 and hand in the completed work with your partner.

10.2 Quicksort

In this part you will learn a bit more about the QuickSort sorting algorithm and do some preliminary timing measurements for a couple of sorting algorithms.

Start with a new project **SortingAlgorithms** and import all files in **Sorting.zip** file.

1. Save the file **citydb.txt** in the same directory where *Eclipse* has your `src` and `bin` folder.

   Set up the *Configuration* as usual and run the project. It will come up with a file dialog asking you to select the input file. Choose the file **citydb.txt**. Now look at the results. Besides the tests, the output includes the timing results for the insertion sort included in the code.

   Change the number of data items to be read to 30000 and run the tests again. Be patient, it may take a while.

2. Add the code for the two variants of *QuickSort* posted on the course wiki. Add the code that invokes each algorithm with the data from the **citydb.txt** file. Add the necessary test cases. Finally add the code that measures the time needed to complete each algorithm.

   Run the program and observe the timing results.

3. The **Algortihms.java** files includes the definition of a list of items of the type `T`. Add the method `quicksort` to these classes and the interface - using the technique similar to what we did in class and what you did in Fundies 1 last semester.

4. Add tests, the code that invokes the algorithm with the full database of cities, and the code that measures the timing.

   Run the program and observe the timing results.

5. The last part is a *pencil and paper* exercise.

   For the following starting data show how each of the three versions of the *QuickSort* proceeds. We have shown you the beginning of the *pencil and paper* analysis of the given *Insertion sort* algorithm.

```
+---+---+---+---+---+---+---+---+---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 | 1 | 6 |
+---+---+---+---+---+---+---+---+---+

inserting 1:
                                  sorted
+---+---+---+---+---+---+---+---+---++---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 | 1 || 6 |
+---+---+---+---+---+---+---+---+---++---+
                                    ^    ^
                            compare - no swap

inserting 3:
                                sorted
+---+---+---+---+---+---+---+---++---+---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 || 1 | 6 |
+---+---+---+---+---+---+---+---++---+---+
                              ^    ^
                            swap

                           sorted
+---+---+---+---+---+---+---++---+---+---+
| 7 | 2 | 9 | 5 | 4 | 8 || 1 | 3 | 6 |
+---+---+---+---+---+---+---++---+---+---+
                                ^    ^
                            compare - no swap

inserting 8:
                           sorted
+---+---+---+---+---+---+---++---+---+---+
| 7 | 2 | 9 | 5 | 4 | 8 || 1 | 3 | 6 |
+---+---+---+---+---+---+---++---+---+---+
                          ^    ^
                        swap

                       sorted
+---+---+---+---+---+---++---+---+---+---+
| 7 | 2 | 9 | 5 | 4 || 1 | 8 | 3 | 6 |
+---+---+---+---+---+---++---+---+---+---+
                            ^    ^
                          swap

                       sorted
+---+---+---+---+---+---++---+---+---+---+
| 7 | 2 | 9 | 5 | 4 || 1 | 3 | 8 | 6 |
+---+---+---+---+---+---++---+---+---+---+
                              ^    ^
                            swap
```

```
                      sorted
+---+---+---+---+---++---+---+---+---+
| 7 | 2 | 9 | 5 | 4 || 1 | 3 | 6 | 8 |
+---+---+---+---+---++---+---+---+---+
                                ^   ^
                               swap
inserting 4:

...
```

10.3 William Shakespeare

The Application

Have you ever wondered about the size of Shakespeare's vocabulary? For this assignment you will write a program that reads its input from a text file and lists the words that occur most frequently, together with a count of how many different words occur in the file. If this program were to run on a file that contains all of Shakespeare's works, it would tell you the approximate size of his vocabulary, and how often he uses the most common words.

Hamlet, for example, contains about 4542 distinct words, and the word "king" occurs 202 times.

The Problem

Start by downloading the file `Assignment10.zip` and making an Eclipse project that contains these files. Run the project, to make sure you have all pieces in place. The `Examples` class uses the `tester` package as we have done before.

You are given the file `Hamlet.txt` that contains the entire text of *Hamlet* and a file `InFileReader.java` that contains the code that generates the words from the file `Hamlet.txt` one at a time, via an iterator. Save the file `Hamlet.txt` in the `Eclipse` project directory (where you find the subdirectories `src` and `bin`).

Note: Here you will use the imperative Iterator interface that is a part of Java Standard Library. Make sure to look up the documentation for this interface and understand how it works.

Your tasks are the following:

1. Design the class `Word` to represent one word of Shakespeare's vocabulary, together with its frequency counter. The constructor takes only one `String` (for example the word "king") and starts the counter at one. We consider one `Word` instance to be equal to another, if they represent the same word, regardless of the value of the frequency counter. That means that you have to override the method `equals()` as well as the method `hashCode()`.

2. Design the class that implements the `Comparator` interface, so that the words can be sorted by frequencies. (Be careful!) When you are done, place this class definition as the last part of the class definition of the class `Word`. This is called an *inner class*.

   Note: In this program there will be two ways of comparing the instances of the Word class - by the String that it represents and by the counter for the word that this instance represents.

3. Include in the class `Word` the method that allows you to increment the counter (using mutation), and a method `toString` that prints one line with the word and its frequency.

4. Design the class `WordCounter` that keeps track of all the words we have seen so far. It should include the following methods:

```
// records the Word objects generated by the given Iterator
// for each word record the number of ocurrences
void countWords (Iterator it) { ... }

// How many different Words has this WordCounter recorded?
int words() { ... }

// Prints the n most common words and their frequencies.
void printWords (int n) { ... }
```

   Here are additional details:

5. `countWords` consumes an iterator that generates the words and builds the collection of the appropriate `Word` instances, with the correct frequencies. This collection is then used by the next two method to show the results of our text analysis.

6. `words` produces the total count of different words that have been consumed.

7. `printWords` consumes an integer `n` and prints the top `n` words with the highest frequencies (using the `toString` method defined in the class `Word`).

Note: The given code expects that you implement the classes as given, with the same names and methods. It will then check whether your program works correctly. That does not mean you do not need to design tests.

Testing of the Shakespeare Project

Of course, you need to test all methods as you are designing them. Design the tests in two stages:

1. For the class `Word` and the the class `WordCounter` use a technique similar to what was done in the past assignments, i.e. design a class `Examples` with the necessary sample data and all tests.

2. Convert all tests into `JUnit` tests. Hand in both versions.

10.4 Documentation

The projects should contain complete `Javadoc` documentation that should produce the documentation pages without warnings. You do not need to submit the documentation pages.