1DV507, Programming and Data Structures, Spring 2017

Assignment 3: Algorithms, Hashing and BSTs, and GUI (Part 2)

Problems?
Do not hesitate to ask your teaching assistant at the practical meetings (or Jonas at the lectures) if you have any problems. You can also post a question in the assignment forum in Moodle.

Prepare Eclipse for course 1DV507 and Assignment 3
Inside your Java project named 1DV507, create a new *package* with the name `YourLnuUserName_assign3` and save all program files for this assignment inside that package. Later on, when submitting your assignment, you should submit a zipped version of this folder/package.

General Assignment Rules

- Use English! All documentation, names of variables, methods, classes, and user instructions, should be in English.
- Each exercise that involves more than one class should be in a separate package with a suitable (English!) name. For example, in Exercise 2, create a new sub package named `sort` inside your package `YourLnuUserName_assign3` and save all .java files related to this exercise inside this package.
- All programs asking the user to provide some input should check that the user input is correct and take appropriate actions if it is not.

Lecture 7 - Algorithms (1)

- **Exercise 1**
  Every Computer Science course involving algorithms should include the *Euclidean algorithm*! The Euclidean algorithm is an algorithm to decide the greatest common divisor of two positive integers. The greatest common divisor of N and M, in short GCD(M,N), is the largest integer X such as M and N are evenly dividable with X. Some examples:

  ```
  GCD(18,12) = 6
  GCD(42,56) = 14
  GCD(9,28) = 1
  ```

  Write a program `EuclideanMain` that takes two integers as input and calculates (and presents) their GCD using this algorithm.

  More information about the Euclidean algorithm can be found on the Internet. In Swedish: *Euklides algoritm*

- **Exercise 2 (50% VG Exercise!)**

  This exercise is about sorting arrays of integer and strings. Create one class `SortingAlgorithms` containing four static methods:

  ```
  public int[] insertionSort(int[] in)
  public int[] mergeSort(int[] in)         // VG Exercise
  public String[] insertionSort(String[] in, Comparator<String> c)
  public String[] mergeSort(String[] in, Comparator<String> c)   // VG Exercise
  ```

All methods return a new sorted array where all elements are sorted in ascending order (lowest first) in the first two integer array methods, as defined by the Comparator in the final two String array methods. The input arrays `in` should not be changed. The methods should of course use the *Insertion Sort* and *Merge Sort* algorithms.

You should also provide a JUnit test class `SortTest` that tests each method.

Notice:

- Our suggestion is that you start to implement (and test) the integer versions.
- Both algorithms are described in the lecture slides and in the textbook by Horstmann. You can also find plenty of information on the Internet.
- The textbook (and Internet) also describes how to implement these algorithms (integer version) in Java. Feel free to take any such implementation as your starting point. However, in this case you must clearly state in your assignments submission which implementation you have used. Provide a web site if you have taken it from the Internet.

Lecture 8 - Hashing and Binary Search Trees

The following five exercises are actually one large exercise named *Count Words*. We have divided Count Words into smaller steps, Exercises 3 - 7, for simplicity. What we want you to do is to count the number of different words in the text HistoryOfProgramming.txt by adding all "words" to a set. We will use four different set implementations: two predefined from the Java library and two that you will implement by yourselves.

Notice: All files related to *Count Words* should be saved in a package named `count_words`.

- **Exercise 3**
  Write a program `IdentyfyWordsMain` that reads a text file(like HistoryOfProgramming) and divide the text into a sequence of words (word = sequence of letters). All non-letters (except whitespace) should be removed. Save the result in a new file (words.txt). Example:

```
Text
====
Computer programming, History of programming
From Wikipedia, the free encyclopedia (081110)

The earliest known programmable machine (that is a machine whose
behavior can be controlled by changes to a
"program") was Al-Jazari's programmable humanoid robot in 1206.

Sequence of words
=================
Computer programming History of programming
From Wikipedia the free encyclopedia
The earliest known programmable machine that is a machine whose
behavior can be controlled by changes to a
program was Al Jazaris programmable humanoid robot in
```

  All exceptions related to file handling shall be handled within the program.

- **Exercise 4**

  Create a class Word, representing a word. Two words should be considered equal if they consist of the same sequence of letters and we consider upper case and lower case as equal. For example hello, Hello and HELLO are considered to be equal. The methods equals and hashCode define the meaning of "equality". Thus, the class Word should look like the following.

```
public class Word implements Comparable<Word> {
   private String word;
```

```
    public Word(String str) { ... }
    public String toString() { return word; }

    /* Override Object methods */
    public int hashCode() { ... compute a hash value for word }
    public boolean equals(Object other) { ... true if two words are equal }

    /* Implement Comparable */
    public int compareTo(Word w) { ... compares two words lexicographically }
}
```

Note:

- If you want, you can add more methods. The methods mentioned above are the minimum requirement.
- Exercise 5 and onward is based on Exercise 4. Thus, carefully test all methods before proceeding.

- **Exercise 5**

  Create a program WordCount1Main doing the following:

  For each word in the file word.txt

  1. Create an object of the class Word
  2. Add the object to a set of the type java.util.HashSet
  3. Add the object to a set of the type java.util.TreeSet

  Note:

  1. The size of the sets should correspond to the number of different words in the files. (Our tests gave 350 words for the file HistoryOfProgramming)
  2. An iteration over the words in the TreeSet should give the words in alphabetical order.
  3. Since our defintion of a word is not very precise (similar to the WarAndPeace exercise in Assignment 2), we do not expect all of you to end up with exactly 350 words. But it should be rather close.

- **Exercise 6**

  Given the following interface

```
public interface WordSet extends Iterable {
    public void add(Word word); // Add word if not already added
    public boolean contains(Word word); // Return true if word contained
    public int size(); // Return current set size
    public String toString(); // Print contained words
}
```

  Implement the interface using a) Hashing, b) Binary Search Tree. In the case of hashing, a rehash shall be performed when the number of inserted elements equals the number of buckets. For the binary search tree, the elements shall be sorted using the method compareTo. The names of the two implementations shall be `HashWordSet` and `TreeWordSet`.

  Note: You are not allowed to use any predefined collection classes from the Java library. However, you are allowed to use arrays.

- **Exercise 7**

  Repeat Exercise 5 with the new implementations HashWordSet and TreeWordSet. The program shall be called `WordCount2Main`. The two notes of Exercise 5 should still be valid.

Lecture 9 - JavaFX (Part 2)

Important: You are not allowed to use any GUI builder tools in these assignments. All your code should be written by you, not generated by a tool.

- **Exercise 8**
  Design and implement a class `UpDownPane` where the pane contains a pattern of 7x7 smaller panes. The small pane in the middle of the big pane will, once the program is started, contain a small icon (or image). When an arrow key is pressed, the icon will move a step up/down or right/left, to a neighboring small pane. If the icon is moved out of the bigger pane, then the icon will appear on the other side of the bigger pane (i.e. if the icon is moved upwards when in an uppermost small pane, then it will appear on the corresponding small panel in the bottom row).
  Also write a test program `UpDownMain.java`, starting a window containing an `UpDownPanel`

- **Exercise 9**
  Design and implement an application that plays the game *Catch-the-Creature*. Use an image or icon to represent the creature. Have the creature appear at random locations, then disappear and reappear somewhere else. The goal for the player is to "catch" the creature by pressing the mouse button while the mouse pointer is on the creature image.

  Create a separate class to represent the creature, and include in it a method that determines if the location of the mouse click corresponds to the current location of the creature. Display a count of the number of times the creature is caught.

- **Exercise 10**
  In Assignment 1 you used an external library XChart to present a bar and pie chart for a set of numbers read from a file. Revisit this task but create a graphical user interface that presents the numbers with one or more suitable charts using JavaFX. The file should be opened using a file selector and a suitable error message should be displayed if an error occurs.

- **Exercise 11 (VG Task)**
  Create a program with JavaFX that displays a bouncing graphics (an image of a ball or the head of Darth Vader or anything else). The object should bounce around in the window. There should also be a button in the win- dow that releases another object (unlimited number should be possible to have in the window, but only one new object should be released for each press of the button). The objects must also bounce on each other when the program is running.

Submission

We are only interested in your .java files. Hence, zip the directory named `YourLnuUserName_assign3` (inside directory named `src`) and submit it using the Moodle submission system.