Assignment 3: Recursion, Hashing and BSTs, GUI 2

Problems?
Do not hesitate to ask your teaching assistant at the practical meetings (or Jonas or Tobias 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

- 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 name.
- 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.

Submission

Please note that all exercises apart from the VG exercises (the merge sort part of Exercise 2, and Exercise 10) are mandatory to pass. Also, 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.
Important: Use zip (not rar, 7z, gz or some other compression format). Also, make sure to also include any images or icons that you might have used in the JavaFX exercises.

Lecture 7 - Recursion and External Packages

Exercises 1-3 can be handled by a single class respectively. Hence, there is no need for any additional classes apart from the one containing the main method. However, feel free to divide your programs into a number of methods.

- **Exercise 1**
  Write a program `PrintRecursive` that includes two recursive methods `print(String str, int pos)` and `printReverse(String str, int pos)` that prints the input string (one character at the time) as it is (`print`) and in reverse order `printReverse`. Add also a main method that looks like this

```
public static void main(String[] args) {
   String str = "Hello Everyone!";

   print(str, 0);
   System.out.println(); // Line break
   printReverse(str, 0);
}
```

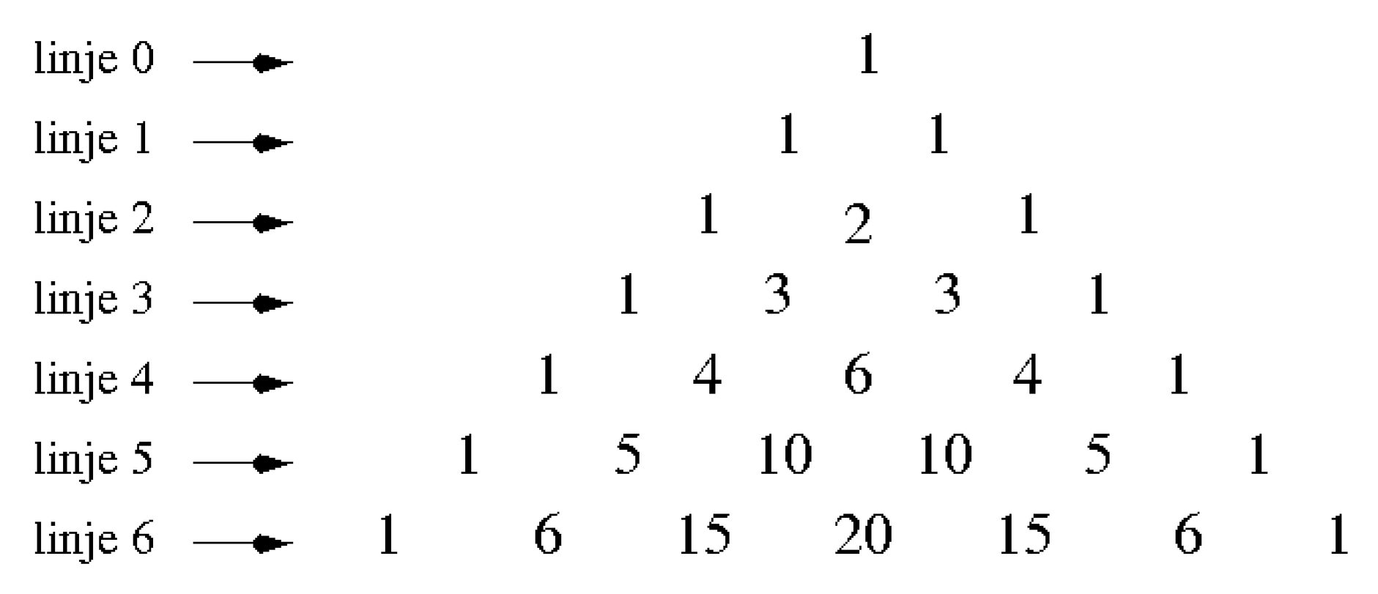
  and when executed generates the following output:

```
Hello Everyone!
!enoyrevE olleH
```

  Notice: Both methods should be recursive and you are not allowed to use any iterative statements.

- **Exercise 2**
  Write a program `PrintJavaMain` that includes a recursive method `printAllJavaFiles(File directory)` that prints all `.java` files in the directory and all its sub directories. Both the name of the file and the size, given as the number of rows, should be printed. All exceptions should be handled in the program.

- **Exercise 3**
  Write a program `PascalMain` that prints the n:th row of Pascal's Triangle. The program should include a recursive method `int[] pascalRow(int n)` computing the n:th row of the triangle. Notice, your program only needs to print line n, not necessarily the whole triangle.



- **Exercise 4**
  XChart (`http://knowm.org/open-source/xchart/`) is an open source project making it possible to, in a Java program, plot different charts and diagrams. You can also, for example, make simple xy-plots, pie charts, bar charts and a lot of other things.

  Note: This task is about downloading, installing and learning to use an unknown Java package, available on the Internet. Therefore, we will not provide much instructions.

  In the following tasks you are supposed to use XChart to produce different types of curves and diagrams.

  - *Warm up, not to be submitted.*: Download and install XChart. Test your installation by creating a program `ScatterPlot.java` containing the following main method:

```
public static void main(String[] args) {
    // Create and Customize Chart
    XYChart chart = new XYChartBuilder().width(800).height(600).build();
    chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesRenderStyle.Scatter);
    chart.getStyler().setChartTitleVisible(false);
    chart.getStyler().setLegendPosition(LegendPosition.InsideSW);
    chart.getStyler().setMarkerSize(5);

    // Generate data
    List xData = new ArrayList();
    List yData = new ArrayList();
    Random random = new Random();
    int size = 1000;
    for (int i = 0; i < size; i++) {
        xData.add(random.nextGaussian() );
        yData.add(random.nextGaussian());
    }

    // Display scatter plot
    chart.addSeries("Gaussian Blob", xData, yData);
    new SwingWrapper(chart).displayChart();
}
```

  - *To be submitted:* Write a program `SinMain.java` plotting the curve `y = (1 + x/pi)*cos(x)*cos(40*x)` in the range `0 <= x <= 2*pi`.
  - *To be submitted:* In the previous course you wrote the program `Histogram.java` showing a very primitive bar chart of a number of integers. Change the program to use XChart to present a bar chart and a pie chart of the same set of data.

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 5 - 8, 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 5**
  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 6**

  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 6 and onward is based on Exercise 5. Thus, carefully test all methods before proceeding.

- **Exercise 6**

  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 7**

  Given the following interface

```
public interface WordSet extends Iterable<Word> {
   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 8**

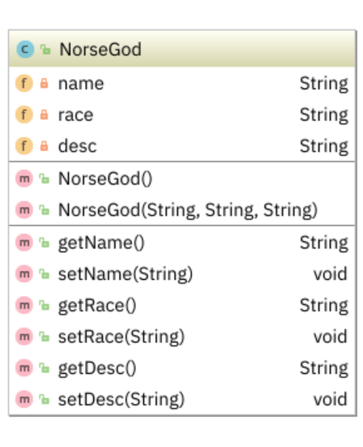
  Repeat Exercise 6 with the new implementations HashWordSet and TreeWordSet. The program shall be called `WordCount2Main`. The two notes of Exercise 6 should still be valid.

Lecture 9 - JavaFX (Part 2)

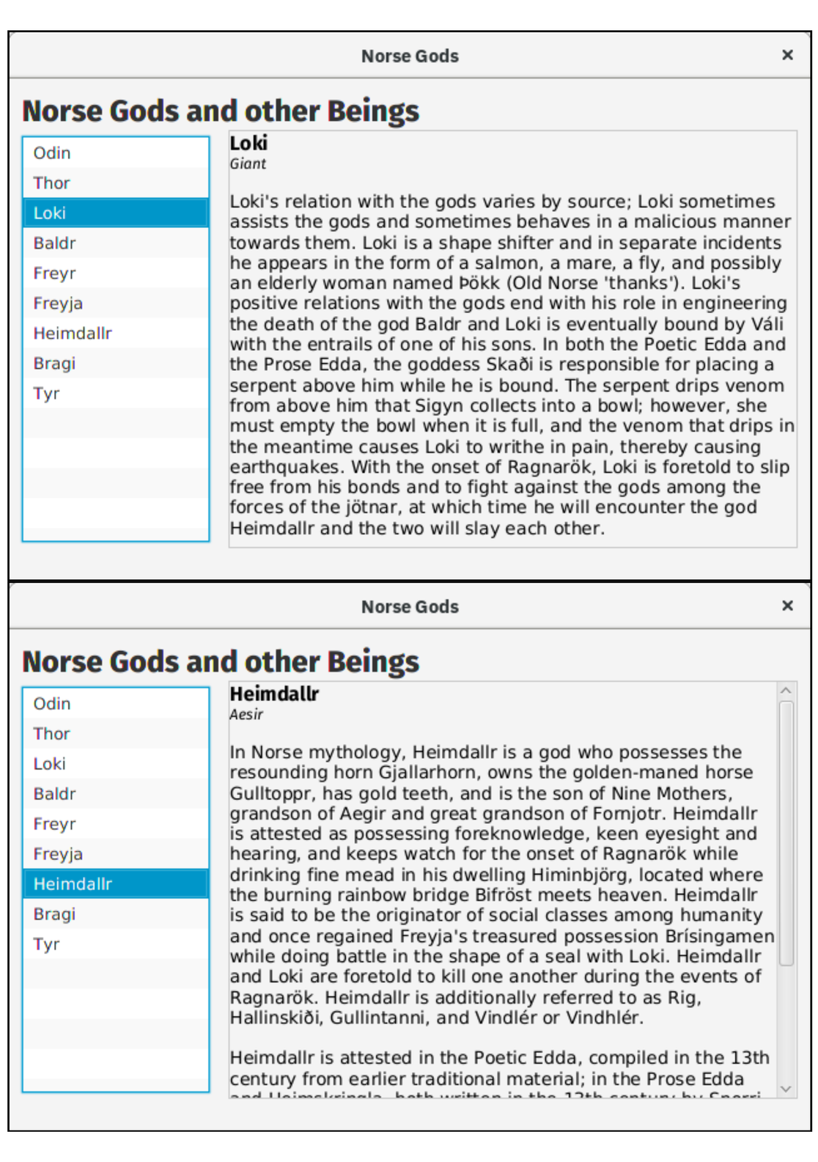
Important: You are not allowed to use any GUI builder tools in these assignments. All your code should be written by you, not generated by a tool.

- **Exercise 8**
  In the picture below there is a class in UML for **NorseGod**s. In this task you are going to create this class as well as a graphical user interface for showing information on the different gods by selecting them from a list.

  When the program starts an **ArrayList** of these gods should be contructed and populated with *at least* eight Norse gods (or other beings from the Norse mythology). The information stored is the name of the god, its race (aesir, vanir, giant or other) as well as a description of the god (or being). The information on the gods and beings themselves can be copied and pasted from Wikipedia.



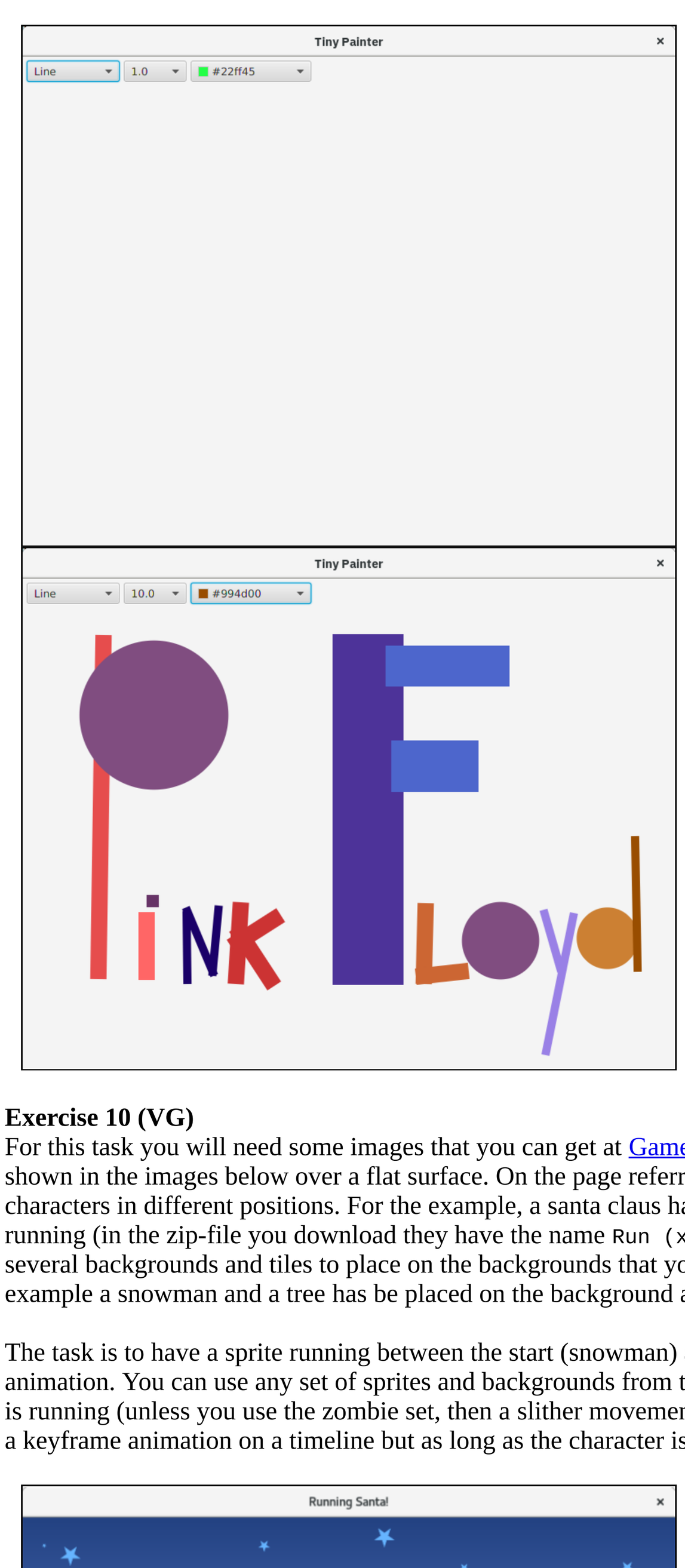
  The final program should look something like in the pictures below, but you are free to change the GUI as long as the same functionality exists. The GUI shown has been created using a **BorderPane** for layout and for the text the class **TextFlow** has been used (this is not a requirement, several **Text** or **Label** classes are possible to use as well). A requirement is that the text should be scrollable if too much is shown in the window and therefore you need to look at the class **ScrollPane**.



- **Exercise 9**
  Create a JavaFX program called `TinyPainter` that will be a simple paint application. At the top of the application there should be a row of drop down menus for (in order) shape, size and colour. The shapes are limitied to Line, Dot, Rectangle and Circle and for Line and Dot a size is important, which is the second menu. The sizes should be ranging from 1 to 40 in suitable sizes (not all need to be included). The shapes Rectangle and Circle need no size but the menu can be available anyway.The last menu is for colour. Use the component `ColorPicker` to select a colour.

  When a shape, size and colour is selected, the user should be able to draw the shape in the window. Drawing is done by pressing a mouse button and then drag the mouse to the suitable position in the window. As long as the mouse key is pressed the shape should follow the mouse, but as soon as it is released, the shape should be finalised in the window. For a Line, the start position is where the mouse key is first pressed and end position where it is released. A rectangle is drawn from upper left corner to lower right corner while a Dot simply is a square in the size selected. A circle is drawn with the center at the position where the mouse key is pressed and with a radius to the mouse pointer.

  To simplify somewhat, it is acceptable if the shapes can overwrite the buttons at the top. See images below for examples of what the finished program can look like.



- **Exercise 10 (VG)**
  For this task you will need some images that you can get at Game Art 2D. The task is to animate a character as shown in the images below over a flat surface. On the page referred to you will find so called *sprites*, characters in different positions. For the example, a santa claus has been used which has eleven characters for running (in the zip-file you download they have the name `Run (x).png`, where x is 1 to 11). There are also several backgrounds and tiles to place on the backgrounds that you can combine using a paint tool. In the example a snowman and a tree has be placed on the background at the start and end of the image.

  The task is to have a sprite running between the start (snowman) and end (tree) markers with JavaFX animation. You can use any set of sprites and backgrounds from the Game Art 2D site as long as the character is running (unless you use the zombie set, then a slither movement is quite okay). We recommend that you use a keyframe animation on a timeline but as long as the character is nicely moving over the surface, it is okay.