Assignment 1: Inheritance, Recursion and GUI (1)

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 1
Start by creating a new Java project named 1DV507. Then create a new *package* with the name `YourLnuUserName_assign1` inside the Java project `1DV507` and save all program files for this assignment inside that package.

General Assignment Rules

- Use English! All documentation, names of variables, methods, classes, and user instrutions, should be in English.
- Each execise involving more than one class should be in a separate package with a suitable (English!) name. For example, in Exercise 1, create a new sub package named `intCollection` inside your package `YourLnuUserName_assign1` 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 1 - Inheritance

- **Exercise 1**
  The zipped directory int_collection.zip contains an abstract class `AbstractIntCollection` and two interfaces `IntList` and `IntStack`. The abtract class contains support for developing array-based data structures. The two interfaces define the functionality of an integer list and an integer stack. Your task is to implement the two interfaces by inheriting the support provided by the abstract class and by adding the code required for each individual data structure. That is, provide two classes `ArrayIntList` and `ArrayIntStack` with the following signatures.

  ```
  public class ArrayIntList extends AbstractIntCollection implements IntList

  public class ArrayIntStack extends AbstractIntCollection implements IntStack
  ```

  Additionally, write a program `CollectionMain` that demonstrates how the two classes can be used.
  Notice: The two classes must make use of the abstract class and you are not allowed to make any changes (not a single character) in either the abstract class or the two interfaces apart from changing the package name.

- **Exercise 2**
  In the following exercise you should create a number of classes to solve a problem. The exercise description is rather vague, more of a sketchy scenario than a concrete problem specification. Your task is to create the necessary classes to simulate this scenario. All classes should be properly documented and encapsulated.

  Your task is to create a programming system for a ferry. The ferry transports passengers and vehicles (cars, busses, lorries and bicycles). The ferry has space for 200 passengers and 40 cars. A lorry needs as much space as two busses or 8 cars. A car needs as much space as 5 bicycles. There are different fees for different vehicles and an extra fee might be added for passengers. Use the following fees:

  1. Passenger without vehicle, 20 kr.
  2. Bicycle 40 kr (passenger included).
  3. Car 100 kr + 15 kr/passenger (maximum 4 passengers).

4. Bus 200 kr + 10 kr/ passenger (maximum 20 passengers).
5. Lorry 300 kr + 15 kr/ passenger (maximum 2 passengers).

Each type of vehicle (car, bus, lorry, bicycle) will inherit from the class Vehicle. The functionality of the ferry is given by the interface Ferry :

```
public interface Ferry  {
   int countPassengers();              // Number of passengers on board
   int countVehicleSpace();            // Used vehicle space. One car is 1.
   int countMoney();                   // Earned money
   Iterator<Vehicle> iterator();       // Vehicle iterator
   void embark(Vehicle v);             // Embark vehicle, warning if not enough space
   void embark(Passenger p);           // Embark passenger, warning if not enough room
   void disembark();                   // Clear (empty) ferry. The money earned remains,
                                       // i.e., is not reset to zero
   boolean hasSpaceFor(Vehicle v);     // true if we can embark vehicle v
   boolean hasRoomFor(Passenger p);    // true if we can embark passenger p
   String toString();                  // Nice looking ferry status print out

}
```

A vehicle cannot leave the ferry until the ferry has been disembarked and the same vehicle cannot embark twice. The ferry iterator should iterate over all vehicles embarked (not the passengers). Also write a program FerryMain.java, embarking a number of vehicles and passengers, showing the functionality of the methods.

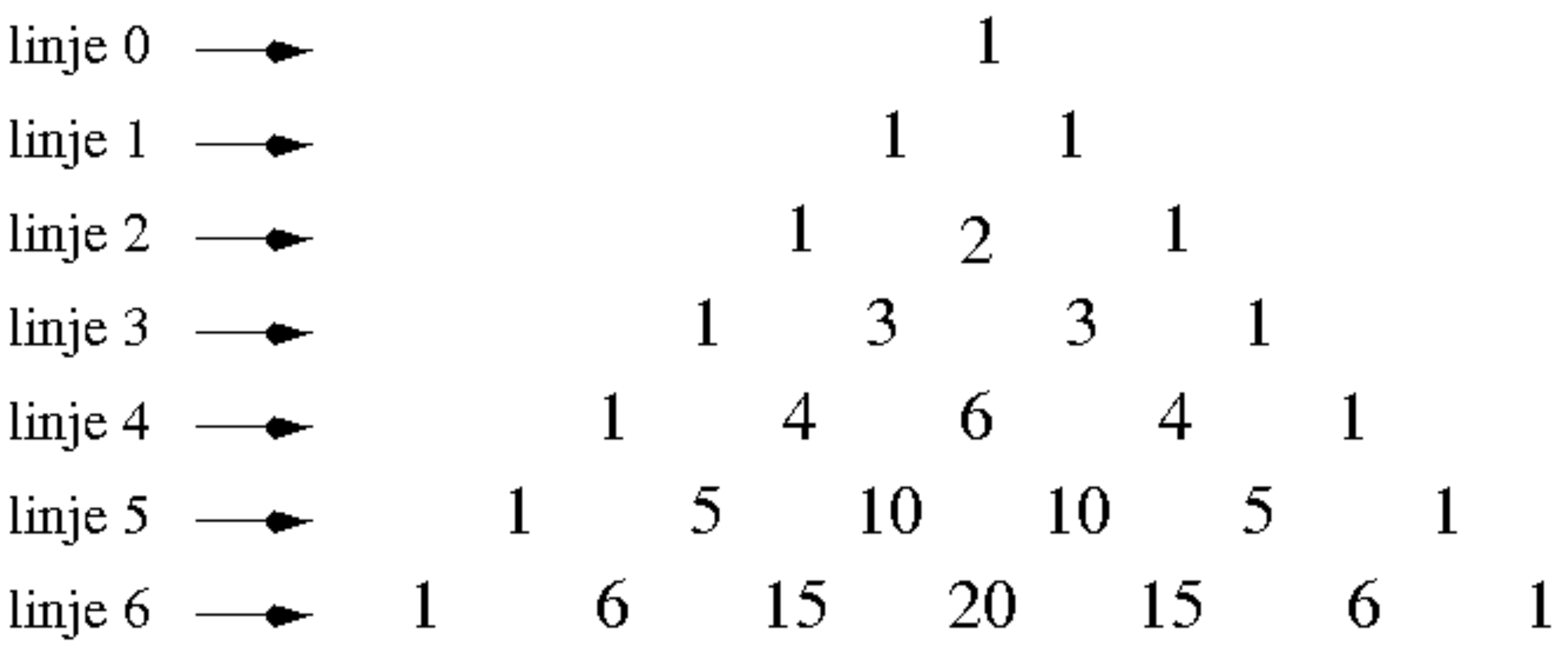
Lecture 2 - Recursion and External Packages

Exercises 3-5 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 3**
  Write a program `SumMain`, that includes a recursive method computing the sum of the integers 1, 2, 3, ..., N. The computation should be based on the following principle: the sum of the integers from 1 to N is equal to the sum of the integers from 1 to N/2 added with the sum of the integers from N/2+1 to N.
  Is this a good solution strategy? Motivate your answer!

- **Exercise 4**
  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 5**
  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.



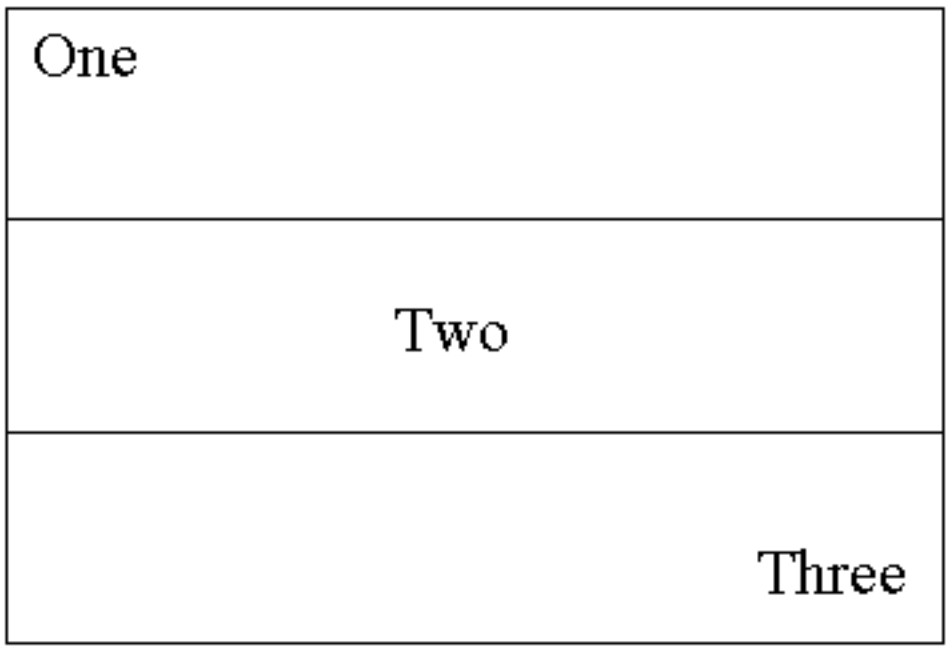
- **Exercise 6**
  The main purpose of this exercise is for you to try to download, install, and use an external package provided as a jar file.

  Create a console based program using the external package Language Tool (`https://languagetool.org/`) as described in the lectures (and on the homepage). The required jar file can be found here. When executed, the program should have the path of a text file as one of the parameters and it should present the errors it might find in a nice way (not just outputting the result from Language Tool). Make the program as object oriented as possible with well defined classes.

Lecture 3 - JavaFX

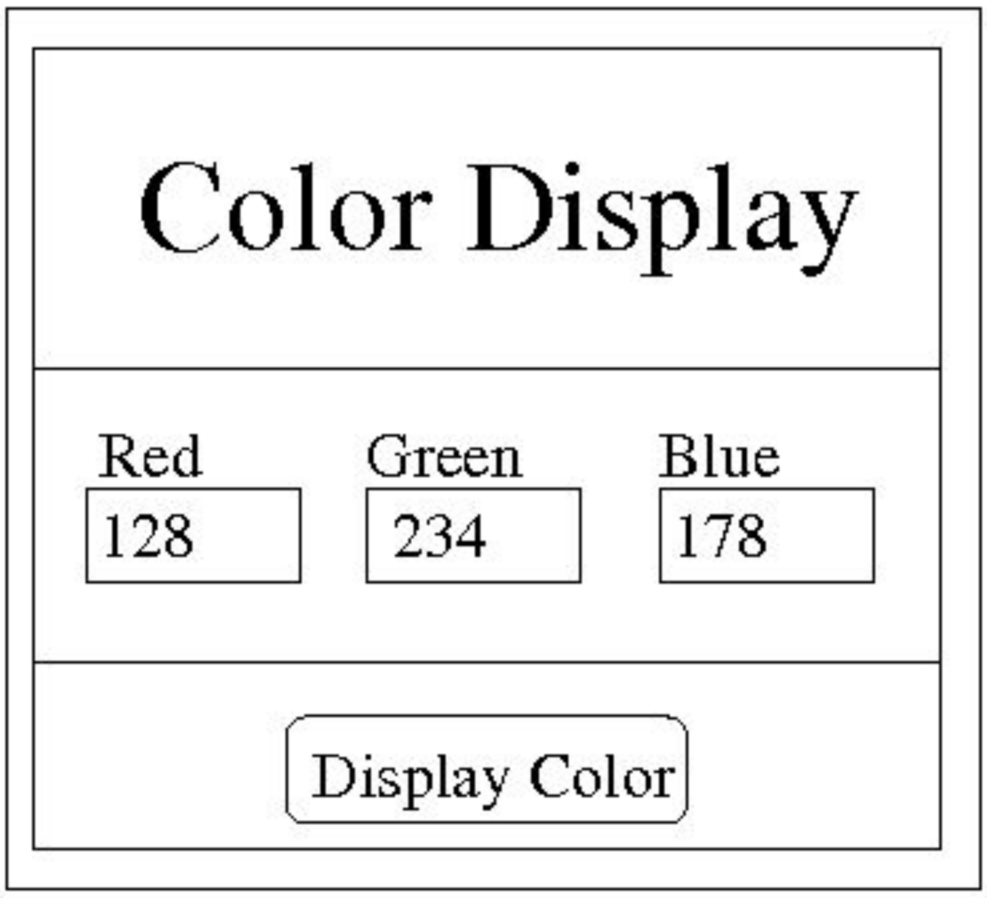
- **Exercise 7**
  Write a program `OneTwoThree.java` which creates a JavaFX Application window containing the following.
  1. Three panes with different colors.
  2. Each pane should contain a word: `One`, `Two` or `Three`.
  3. `One` should be placed in the upper left corner, `Two` should be placed in the middle of the panel and `Three` in the lower right corner.

  Choose size and color of the panes to make it look nice.



- **Exercise 8**
  Create a class `RandomPanel` (extends BorderPane) containing two panes. One of the panes should contain a button with the text `New Random`. When the button is pressed a random number in the interval 1-100 is generated and shown in the other pane. Try to change the size of the text so that the number 100 fills the pane. Also, write a test program `RandomMain.java` starting an Application containing a `RandomPanel`.

- **Exercise 9**
  Write a GUI program `ColorDisplay.java`. When the button ”Display Color” is pressed, three integers (red, green, blue) are read and the color of the upper pane is changed according to the values of the integers. Use three components of the type TextField to read the integers. The picture below shows how the execution of the program can look.



If erroneous values of the integers are read, an error message should be printed and the upper pane should not be updated. The most important thing is that the program is working properly. It is also important for the program to have a good structure to make it easy to follow and to understand. Divide big methods and big classes into smaller ones, if necessary.

- **Exercise 10**
  Re-use as much as possible from Exercise 6 but create a graphical user interface for it using JavaFX. In addition to the native JavaFX, also use Controls- FX (`http://fxexperience.com/controlsfx/`). The user should start the program and be able to open a file using a file selector. The file should be presented in the user interface in one text field and the errors in another. Use ControlsFX to make your application better, we like you to experiment with it and use the controls you see fit to make your application better or nicer to look at.

Submission

All exercises should be handed in and we are only interested in your .java files. Hence, zip the directory named `YourLnuUserName_assign1` (inside directory named `src`) and submit it using the Moodle submission system.