1DV507, Programming and Data Structures, Spring 2017

Assignment 1: Inheritance, Recursion and Java 1.8

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**
  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(XYSeriesRenderSty
le.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 3 - New Java 1.8 Features

- **Exercise 7**
  The file CountrySort.java contains a skeleton of a Java program where we want you to sort a string array containing the names of all countries in the world (a few years ago) in three different ways. The sorting should be done using the library method `Arrays.sort(String[] s, Comparator<String> c)` that requires a comparator function `String, String --> int` to define the sorting order to be used.

- Task 1: Sort country names based on their length (longest first) and print a list of the ten countries with longest names
- Task 2: Sort country names in reverse alphabetic order and print a list of the ten first countries in that list
- Task 3: Sort country names based on the number of As (A or a) they contain print a list of the ten countries with most As

Notice: There is not much coding needed in this exercise. You only need to define three comparator functions. Also, we expect you to download CountrySort.java, make a few changes, and than submit that file.

- **Exercise 8**
  The file FunctionPointers.java contains a skeleton of a Java program where we want you to add support for (a) apply predicates on a list of integers, and (b) apply functions on a list of doubles. Recommended steps:
  1. Provide an implementation to the method `selectAndPrint` that prints all elements in the list where a given predicate evaluates to true.
  2. Update the two predicates `odd` and `aboveTen` to make them recognize odd numbers and numbers that are greater than 10
  3. Provide an implementation to the method `applyFunction` that returns a new list containing the numbers resulting from applying function fx on the numbers in the input list.
  4. Provide a function powerOfTwo that computes the square of any input double

  Notice: Once again, there is not much coding needed in this exercise. And once again,, we expect you to download FunctionPointersSort.java, make a few changes, and than submit that file. An execution of the program once you have completed all tasks should look like:

```
Part 1: Apply predicates
Print all numbers: 45 3 24 16 1 1 3 8 7 6 10 12 17 22 30
Print all odd numbers: 45 3 1 1 3 7 17
Print all numbers greater than 10: 45 24 16 12 17 22 30

Part 2: Apply functions
Original: [1.0, 16.0, 25.0, 81.0]
Square root: [1.0, 4.0, 5.0, 9.0]
Power of two: [1.0, 256.0, 625.0, 6561.0]
```

- **Exercise 9 (VG Task)**
  The (very large) file WarAndPeace.txt contains the entire text of the book War and Peace by the russian author Leo Tolstoy. Your task is to write a single-class program `WarAndPeace.java` that, once you have split the text into separate words, use a *single* Java 1.8 stream to compute the number of unique words used in the text. Words are substrings made of ordinary letters and the characters `'` (as in `don't`) and `-` (as in `well-known`). Other type of

characters (e.g. digits) should be removed from each word as well as entire words only containing such characters. Also, we do not care about upper and lower case when looking for unique words. Hence, `HELLO` and `Hello` are considered to be the same word.

The first lines in your code should look something like:

```
String text = readText("WarAndPeace.txt");   // My own method
String[] words = text.split(" ");  // Simple split, will be cleaned
up later on
System.out.println("Initial word count: "+words.length);   // We
found 577091

Stream stream = Arrays.stream(words);
... continue here
```

Submission

We are only interested in your .java files and please notice that the VG exercises 9 is not mandatory. In the end, zip the directory named `YourLnuUserName_assign1` (inside directory named `src`) and submit it using the Moodle submission system.