1DV506, Problem Solving and Programming, Autumn 2017

Assignment 4: Exceptions, IO, and Interfaces

Problems?
Do not hesitate to ask your teaching assistants 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 Assignment 4

Create a new *package* with the name `YourLnuUserName_assign4` inside the Java project `1DV506` and save all program files for this assignment inside that package.

Lecture 9 - IO and Exceptions

All exceptions related to exercises 1 and 2 should be handled within the programs. Also, exercises 1 and 2 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**
  Create a program `Histogram.java`, reading any number of integers from a file and then printing a histogram bar-chart for all integers between 1 and 100. Note: not all integers in the file are neccesarily in the interval [1-100]. An example of an execution:

```
Reading integers from file: C:\Temp\integers.dat
Number of integers in the interval [1,100]: 46
Others: 16
Histogram
  1  - 10  | ******
  11 - 20  | ****
  21 - 30  | **
  31 - 40  | ***
  41 - 50  | *******
  51 - 60  | ****
  61 - 70  | ***
  71 - 80  | *********
  81 - 90  | *****
  91 - 100 | ***
```

  Note 1: You will have to create your own data file. We expect it to be an ordinary text file with one integer on each row. **Note 2:** The absolute path to the data file should be provided as an argument to the main method.

- **Exercise 2**
  Write a program `CountChar.java`, counting characters of different types in a text read from a file. Give the number of characters of the following types:
  - Upper case letters
  - Lower case letters
  - Digits
  - "Whitespace" (i.e. space, tab, return)
  - Other characters

The path to the text file to read can be coded into the program. (We will test with some other file.) An example of a text file is `HistoryOfProgramming`. It is a part of an article from Wikipedia about the history of programming. However, we strongly recommend you to start testing your program with a smaller file where you can manually check the result.

An execution with the file `HistoryOfProgramming` as input should give the following result:

```
Number of upper case letters: 86
Number of lower case letters: 3715
Number of digits: 41
Number of "whitespaces": 728
Number of others: 111
```

If your result does not agree completely with the example above, you have to add a written explanation why you think this happens, to your submission.

Lecture 10 - Static Members and Interfaces

- **Exercise 3**
  Start by creating a new sub package named `stack` inside your package `YourLnuUserName_assign4` and save all .java files related to this exercise inside this package.
  A stack is a LiFo (Last-in, first-out) data structure with three basic operations: push, pop and peek. push is putting an element on the top of the stack, pop removes (and returns) the top element, and peek returns (without removing) the top element. Think of a stack as a pile of plates that can be found in certain restaurants. You can only add and remove the top-most plate. You can not remove any plates in the middle of the pile (without first removing all plates above it). Your task is to implement the following stack interface:

```
public interface Stack {
   int size();                  // Current stack size
   boolean isEmpty();           // true if stack is empty
   void push(Object element);   // Add element at top of stack
   Object pop();                // Return and remove top element,
                                // exception if stack is empty
   Object peek();               // Return (without removing) top element,
                                // exception if stack is empty.
   Iterator<Object> iterator(); // Element iterator
}
```

  The iterator traverses all elements currently in the stack, starting with the top element. Illegal operations on an empty stack (e.g., pop() and peek()) should generate an exception. You should also present a test program StackMain.java that demonstrates how each method can be used. Notice: You are not allowed to use any of the data structures in the Java library. However, you can use arrays.

- **Exercise 4**
  Start by creating a new subpackage named `sort_cities` inside your package `YourLnuUserName_assign4` and save all .java files related to this exercise inside this package.
  Implement a program SortCities that reads an arbitrary number of city names and their zip codes from a text file. You can assume one city in each line and that each city name (String) and zip code (integer) is separated by a semi-colon(;). Create a class City that represents a city and create a city object for each city you read from the file. The City class should also implement the interface Comparable. Once you have read (and constructed) one City object for each line in the file you should print the cities in a sorted order based on their zip codes. An execution might look like this:

```
Reading cities from file: C:\Temp\cities.dat
Number of cities found: 7

23642 Höllviken
```

```
35243 Växjö
51000 Jönköping
72211 Västerås
75242 Uppsala
90325 Umeå
96133 Boden
```

General Java

In the following exercises you should create a number of classes to solve a problem.

- **Exercise 5** (The Drunken Walker)
  Create a class `RandomWalk.java`, simulating a random walk. A random walk is basically a sequence of steps in some enclosed plane, where the direction of each step is random. The walk terminates when a maximal number of steps have been taken or when a step goes outside the given boundary of the plane.

  For this task, assume a plane given by a grid, with the point (0, 0) at the center. The size of the plane is given by an integer; if the given integer is k, then the values of the x and y coordinates can vary from -k to k. Each step will be one unit up, one unit down, one unit to the right or one unit to the left (no diagonal movements).

  The class `RandomWalk` will have the following instance data :

  - X coordinate of the current position
  - Y coordinate of the current position
  - The maximum number of steps in a walk
  - The number of steps taken so far in the walk
  - The size of the plane (according to the description above)

  Other members

  - `RandomWalk(int max, int size)`: the maximum number of steps is `max` and `size` is the size of the plane. The start position is set to (0, 0).
  - `String toString()`: returns a string containing the number of steps taken so far and the current position.
  - `void takeStep()`: simulates taking a single step. Generate a random number, taking on four different values, and let them correspond to a movement up, down, to the right and to the left, respectively. The method should also increase the number of steps taken.
  - `boolean moreSteps()`: returns `true` if the number of steps taken is less than the maximal number of steps, otherwise `false` is returned.
  - `boolean inBounds()`: returns `true` if the current position is inside the boundary of the plane, otherwise `false` is returned.
  - `void walk()`: simulates a random walk, i.e. steps are taken until the maximum number of steps has been taken or until a step goes outside the boundary of the plane.

  Simulation
  Create another class `DrunkenWalk`, simulating walks of drunken people on a platform in a lake. The program should read the boundary, the maximum number of steps, and the number of drunks to simulate. One drunk at a time should be put on the platform and perform its walk. Your program should count how many drunks fall into the water. Test your program for some different values of size and number of steps. Example of an execution:

```
Enter the size: 10
Enter the number of steps: 50
Enter the number of walks: 150
Out of 150 drunk people, 14 (9.34%) fell into the water.
```

- **Exercise 6** (VG Exercise)
  The following 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.

  Start by creating a new sub package named `newsagency` inside your package `YourLnuUserName_assign4` and save all .java files related to this exercise inside this package. Newspapers exchange news by using news agencies (such as Reuters and ITAR- TASS). A newspaper registers at a news agency and sends all its news to the agency. The news agency collects the news and broadcasts it to all registered newspapers, except from to the one who delivered it. Create the classes needed to simulate this scenario. Also create a main class to show a simulation where a couple of newpapers generate news and receive news from others.

Submission

All exercises should be handed in and we are only interested in your .java files. (Notice that the VG exercise 6 is not mandatory.) Hence, zip the directory named `YourLnuUserName_assign4` (inside directory named `src`) and submit it using the Moodle submission system.