Assignment 4: Generic Classes and More Algorithms

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 4
Inside your Java project named 1DV507, create a new *package* with the name `YourLnuUserName_assign4` 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.
- 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.

Generic Classes

- **Exercise 1**

  In Assignment 2 you were asked to provide a linked implementation of the following queue interface

```
public interface Queue {
   public int size();                    // current queue size
   public boolean isEmpty();             // true if queue is empty
   public void enqueue(Object element);  // add element at end of queue
   public Object dequeue();              // return and remove first element.
   public Object first();                // return (without removing) first element
   public Object last();                 // return (without removing) last element
   public String toString();             // return a string representation of the queue content
   public Iterator<Object> iterator();   // element iterator
}
```

  Tasks:
  - Provide a similar *generic* queue interface named `Queue<T>`.
  - Provide a linked generic queue named `LinkedQueue<T>` that implements the interface `Queue<T>` using the *head-and-tail* approach.
  - Write a JUnit test program `QueueTest` for your generic queue implementation. Make sure that you are using at least two parameter types (e.g. Integer and String) in your tests.

  You are of course allowed to reuse your own non-generic queue implementation/test from Assignment 2. However, make sure to take into account all comments given to you by your teaching assistant when he/she corrected it.

Time Measurements

Exercises 2-4 forms a unit related to time measurements. Please create a new subpackage named `time` to handle these exercises.

- **Exercise 2**

  Repeated string concatenations can be done in two ways: 1) Using the plus operator you can construct a long string by constantly increasing the length as: `str = str + "..."`, 2) Using the StringBuilder class and repeated use of method `append("...")`. Your task is to find the fastest approach by measuring how many concatenations, and the length of the final string, each of them can compute *in 1 second* when:

  1. Adding short strings containg only one character
  2. Adding long strings representing a row with 80 characters

  Notice 1: We are interested in eight different numbers. The number of concatenations, and the final string length when: 1) Concatenating short strings, 2) Concatenating long string, 3) Appending short strings, and 4) Appending long strings. Make sure to include the final toString() call when measuring the time for the StringBuilder approach.

  Notice 2: Trustworthy experiments using computers is not based on a single run of the program. Use repeated runs (say 5-10) having an average of about 1 second.

  Notice 3: You will be asked to write a report presenting your result. Hence, it might be a good idea to glance through Exercise 4 before you start implementing.

- **Exercise 3 (50% VG Exercise)**

  In Assignment 3 you implemented 4 different sorting algorithms: Insertion Sort (for both strings and integers) and Merge Sort (for both strings and integers). How many strings and integers can be sorted *in 1 second* using these four algorithms?

  For integers: Sort arrays with random generated integers. The range used by the random generator should be larger than the array size in order to reduce the number of duplicate elements.
  For strings: Sort arrays in alphabetic order using arrays of random generated strings where each string contains 10 randomly generated characters.

  Notice: Handling Merge Sort is a VG Exercise (since implementing it was a VG Exercise in Assignment 3)

- **Exercise 4**

  Write a short report about your experiments in Exercises 2 and 3. For each experiment:

  1. Describe how you did your experiment.
  2. Show a table and/or figure of your results.
  3. An example of what a report might look like.
  4. We expect a good looking report in pdf format produced using a tool like Word or Latex.

  Also, in the report, try to explain why StringBuilder is much faster than string concatenation using the `+` operator.

Priority Queues

Exercises 5-6 are related to Priority Queues and Binary Heaps. Please create a new subpackage named `binheap` to handle these exercises.

- **Exercise 5**

  A very simple version of an array based binary heap can be considered as an integer data structure with only four methods (plus one constructor):

```
public BinaryIntHeap()   // Constructs an empty heap
public void insert(int n) // Add n to heap
public int pullHighest()    // Return and remove element with highest priority
public int size()          // Current heap size
public boolean isEmpty() // True if heap is empty
```

  Task: Implement a class BinaryIntHeap containing the methods above following the standard rules for how to implement a binary heap. Write also a JUnit test case to check the correctness of your heap implementation.

  Notice: In this very simple approach the element and the priority are the same. More general tasks will be handled in the next exercise.

  More information about binary heaps can be found:

  1. In the lecture slides
  2. In the textbook by Horstmann
  3. On the Internet. Just Google on "Binary Heap".

  As usual, please give proper references to all "resources" found on the Internet that you have been using.

- **Exercise 6 (VG Exercise)**

  The binary heap is one implementation technique for a Priority Queue. A Priority Queue is a data structure that allows the processing of a number of Tasks based on some priority. It supports two major operations: `void insert(Task t)` and `Task pullHighest()`. However, we also expect it to support standard operations like contains(), size(), isEmpty(), peekHighest(), etc.

  1. Your first task is to design two interfaces (or abstract classes) named `PriorityQueue` and `Task` that together describes a priority queue in general.
  2. Your second task is to provide a concrete priority queue implementation named `BinaryHeapQueue` (based on binary heaps) and a concrete task implementation named `WorkTask`. In addition to a priority (positive integer), a WorkTask also comes with a work description (a string).

  We also expect you to write a small program `WorkMain` that demonstrates how to use your priority queue.

  Notice: Good Design implies flexibility and extendability. Hence, try to make it easy to replace the `WorkTask` with another type of task that also implements the `Task` interface. Also, it should be easy to switch from one `PriorityQueue` implementation (e.g `BinaryHeapQueue`) to another.

Java in General

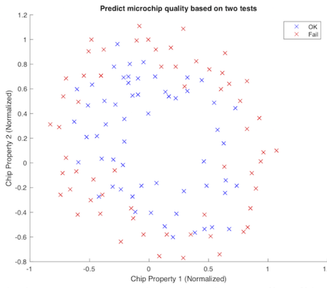
Exercises 7 and are not related to any particular lecture or chapter. They are however typical examples of Java exercises that you will face if you continue to study Computer Science.

- **Exercise 7**

  Dropped! Not part of Assignment 4 any longer.

- **Exercise 8 (VG Exercise)**

  The file microchips.csv contains a dataset related to an attempt to identify flaws in a microchip based on two numerical chip properties. Each row in the file represents one microchip and contains three comma separated values, the two properties and an integer (1 or 0) indicating if the microchip turned out to be OK (1) or Failed (0). A plot of the dataset is shown below.



  Your task is to implement a well-known machine learning algorithm called k-Nearest Neighbors (kNN) and to predict whether three unknown microchips are likely to be OK or Fail. The properties associated with the three unknown microchips are (-0.3, 1.0), (-0.5, -0.1), and (0.6, 0.0), and you should repeat the experiments for k = 1, 3, 5. Hence, a total of nine predictions.

  Your program should start by presenting the dataset in a figure (like the figure above) and then print the nine prediction result on the console. For example, for k = 5 the print out might look like:

```
k = 5
        chip1: [-0.3, 1.0] ==> Fail
        chip2: [-0.5, -0.1] ==> OK
        chip3: [0.6, 0.0] ==> OK
```

  You can find plenty of information about k-Nearest Neighbors on the Internet. Also, feel free to use any kNN Java library/code you can find but make sure to give a proper reference, and to include in your submission everything we need to run your program.

Submission

All exercises should be handed in and we are only interested in your .java files. (Notice that the VG exercise 6 and 8 are not mandatory.) Hence, zip the directory named `YourLnuUserName_assign4` (inside directory named `src`) and submit it using the Moodle submission system.