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.

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.

Lecture 9 - Generic Classes

- **Exercise 1**

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

  ```java
  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 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.

- **Exercise 2 (VG Exercise)**

  Provide also an array based generic queue implementation of the interface `Queue<T>` based on the circular array approach that was presented in Exercise 2, Assignment 2.

Lecture 10 - Time Measurements and Priority Queues

Exercises 3-5 forms a unit related to time measurements. Exercises 6-7 are related to Priority Queues and Binary Heaps.

Notice: In exercises 3 and 4 we have changed the time from 10 to 1 second to avoid certain memory problems.

- **Exercise 3**

  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 containing 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) having an average of about 1 second.

- **Exercise 4**

  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 of random generated strings where each string contains 10 randomly generated characters.

- **Exercise 5**

  Write a short report about your experiments in Exercises 3 and 4. For each experiment:

  1. Describe how you did your experiment.
  2. Show a table of your results.

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

- **Exercise 6**

  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 7 (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.

Submission

All exercises should be handed in and we are only interested in your .java files. (Notice that the VG exercises 2, and 7 are not mandatory.) Hence, zip the directory named `YourLnuUserName_assign4` (inside directory named `src`) and submit it using the Moodle submission system.