Assignment 2: Data Structures, JUnit, and JavaFX (2)

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 2
Inside your Java project named 1DV507, create a new *package* with the name `YourLnuUserName_assign2` 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 exercise that involves 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 `queue` inside your package `YourLnuUserName_assign2` 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 4 - Simple Data Structures

- **Exercise 1**

  A Queue is a FIFO (first in, first out) data structure. Consider 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 iterator();            // element iterator
}
```

  The iterator iterates over all elements of the queue. Operations not allowed on an empty queue shall generate an unchecked exception.

  Tasks:
  - Create a *linked* implementation `LinkedQueue.java` of the interface Queue. Use the *head-and-tail* approach.
  - Write also a program `QueueMain.java` showing how all methods work.
  - Create Javadoc comments in the code and generate good-looking and extensive HTML documentation for the interface and the class. All public class members shall be documented.

  Notice:
  - The implementation shall be linked, i.e. a sequence of linked nodes where each node represents an element.
  - You are not allowed to use any of the predefined classes in the Java library.
  - In the report, the HTML pages generated by the classes `Queue` and `LinkedQueue` shall be attached. Attach no other HTML pages!

- **Exercise 2 (VG Task)**
  A straight forward arrray based implementation of the Queue interface above would use an Object array (that grows on demand) and two indices `first` and `last` to keep track of the array positions where to remove an element on `dequeue` (return and increase position `first`), and where to add an element on `enqueue` (insert at

and increase position `last`). The problem with this approach is that after (say) 100 `dequeue` we will have that `first = 100` and 100 non-used elements (positions 1 to 99) that never will be used again. That is, a waste of memory.

Your task is to provide an array based Queue implementation (`ArrayQueue`) that avoids this problem by treating the array like a circular array in which array indices larger than the array size ``wrap around" to the beginning of the array. That is,

- When, after a number of enqueues, index `last` reaches `array.length`, you should move `last` to position 0 and start to reuse the first part of the array.
- Later on, after an even larger number of dequeues, you will reach the point where index `first` reaches `array.length`. Then, move `first` to position 0 (and you have returned to the initial configuration where all the queue elements are stored between `first` and `last`).
- Finally, the array is full when `last`, after one or more ``wrap arounds", reaches `first`. In that case you should resize the array and restore the order such that `first` equals 0.

This approach is called a queue implementation based on *circular arrays*. Look it up on the Internet to get all details.

Lecture 5 - JUnit Testing

- **Exercise 3**
  Write a JUnit test for the class `LinkedQueue` in Exercise 1.

- **Exercise 4** (VG Task)
  Modify the Queue test in Exercise 3 so that it can also handle the `ArrayQueue` class from Exercise 2. We do not want two separate tests. We want one test that can be used for any implementation of the `Queue` interface *with just a minimum of modifications*.

Lecture 6 - JavaFX

Important: You are not allowed to use any GUI builder tools in these assignments. All your code should be written by you, not generated by a tool.

- **Exercise 5**
  Design and implement a class `UpDownPane` where the pane contains a pattern of 7x7 smaller panes. The small pane in the middle of the big pane will, once the program is started, contain a small icon (or image). When an arrow key is pressed, the icon will move a step up/down or right/left, to a neighboring small pane. If the icon is moved out of the bigger pane, then the icon will appear on the other side of the bigger pane (i.e. if the icon is moved upwards when in an uppermost small pane, then it will appear on the corresponding small panel in the bottom row).
  Also write a test program `UpDownMain.java`, starting a window containing an `UpDownPanel`

- **Exercise 6**
  Design and implement an application that plays the game *Catch-the-Creature*. Use an image or icon to represent the creature. Have the creature appear at random locations, then disappear and reappear somewhere else. The goal for the player is to "catch" the creature by pressing the mouse button while the mouse pointer is on the creature image.

  Create a separate class to represent the creature, and include in it a method that determines if the location of the mouse click corresponds to the current location of the creature. Display a count of the number of times the creature is caught.

- **Exercise 7**
  In the previous course (1DV506) you had an assignment where you read numbers from a file and showed a simple histogram in text (Assignment 4, Exercise 1). Revisit this task but create a graphical user interface that shows the numbers with one or more suitable charts using JavaFX. The file should be opened using a file selector and a suitable error message should be displayed if an error occurs.

- **Exercise 8 (VG Task)**
  Create a program with JavaFX that displays a bouncing graphics (an image of a ball or the head of Darth Vader or anything else). The object should bounce around in the window. There should also be a button in the window that releases another object (unlimited number should be possible to have in the window, but only one new object should be released for each press of the button). The objects must also bounce on each other when the

program is running.

Submission

All exercises should be handed in and we are only interested in your .java files. (Notice that the VG exercises 2, 4, and 8 are not mandatory.) Hence, zip the directory named `YourLnuUserName_assign2` (inside directory named `src`) and submit it using the Moodle submission system. Make sure to also include any images or icons that ypou might have used in the JavaFX exercises.