Exercise No. 3

27.11.05-11.12.05

The main purpose of this assignment is to practice *Design By Contract* and unit testing. You will implement and define the contract and invariant of a class representing a FIFO (*first-in, first-out*) queue of a fixed capacity. Then, you will write a class for black-box tests.

The Queue Class:

The name of the class is `FixedCapacityQueue<T>`. It is a generic class that supports the following commands and queries:

- **`public FixedCapacityQueue(int capacity)`** - Constructs a FIFO queue of a fixed capacity as specified.
- **`public void enqueue(T t)`** - Inserts the given element `t` at the tail of the queue if the queue is not full.
- **`public void dequeue()`** - Deletes the element at the head of the queue if the queue is not empty.
- **`public T getHead()`** - Returns the element at the head of the queue if the queue is not empty
- **`public boolean isEmpty()`** - Returns true if the queue is empty. Otherwise, returns false.
- **`public boolean isFull()`** - Returns true if the queue is full. Otherwise, returns false.

Your implementation of the queue will be based on an array of length *n*, where *n* is the specified capacity in the constructor. Note in order to construct a generic array of length n, you should use the following syntax: `(T[])new Object[n]` (this syntax will be explained in class).

Implement the class efficiently using a cyclic implementation. The simplest approach is to hold two accessory variables: `start` and `length`. Where `start` holds the index of the current top of the queue in the array and `length` is the number of elements. When dequeuing, `start` grows by one and `length` is reduced by one, modulo the capacity of the array.

Carefully define the contract in Javadoc comments using the `@pre` and `@post` and `@inv` tags for the pre-condition, post-condition and representation invariant, respectively. Also, define the implementation contract using the `@imp-inv` and `@imp-post` for non-exported clauses in the invariant and post-condition, as explained in recitation 5.

Use Javadoc for generating documentation in HTML format with the API and contract of the class. Note that for this purpose you will need to install the "taglets.jar" jar (available on the course's web-site) according to the instructions given in recitation 5.

The Unit Test Case Class:

Implement a class named `TestFixedCapacityQueue` that will act as a unit test case with black-box tests for the `FixedCapacityQueue` class. Design at least three test cases (scenarios) in which `FixedCapacityQueue` is used. Define three methods in TestFixedCapacityQueue, named `test1`,

`test2`, `test3` that will conduct the test cases. Create a `main` method to call these methods. The test cases should not be trivial and should demonstrate testing corner cases. Note also that your black-box test cases should be general and pass any implementation that satisfies the specification of the `FixedCapacityQueue` class and not only your implementation. In the context of the exercise, this means that the exercise-checker may check your test class by running it on different implementations including buggy implementations to see whether it reports errors/failures.

Use Javadoc for generating documentation in HTML format with the API of the class (contract description is not required).

What to hand in?

1. **Softcopy**: A jar file named `ex3.jar`, with the `FixedCapacityQueue` class, its unit test class `TestFixedCapacityQueue`, and the HTML Javadoc documentation, should be placed under your `~/software1` directory as explained in the submission guidelines.
2. **Hardcopy:** Printouts of the `FixedCapacityQueue`.java, `TestFixedCapacityQueue`.java files and the Javadoc HTML files for these classes.