# OBJECT ORIENTED PROGRAMMING - ADTS

## Unassessed Tutorial 2: *List variations*, *Stacks* and *Queues*

The aims of this tutorial are to practice with:

- ADTs that are variations of lists, stacks and queues by implementing some of their access procedures.
- to practice using lists, stacks and queues to solve small problems.

1. Give the implementation of the access procedures `add` and `delete` of the interface `List<T>`, as specified in Slides 2 and 3 of Unit 2, but in the case when this interface is implemented by a `doublyLinked` class that uses an inner class `Node`.

2. Assume the existence of an ADT `Stack<T>` as defined in Unit 3. Implement in Java the following methods of a client program that makes uses of a dynamic implementations of `Stack<T>`.

   (a) `public void displayReverse(Stack<Integer> stack)`
   //post: Displays the content of `stack` from bottom to top.

   (b) `public int count(Stack<Integer> stack)`
   //post: Counts the number of items in `stack`, leaving `stack` unchanged;

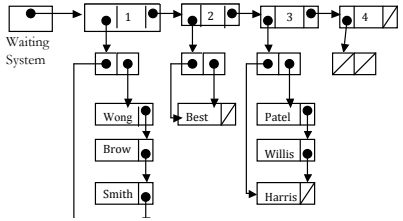
   (c) `public void deleteElement(Stack<Integer> stack, int elem)`
   //post: Deletes every occurrence of `elem` in `stack`, leaving the order of the
   //post: remaining elements in `stack` unchanged.

3. Consider a calculator that evaluates expressions written in postfix form. For instance the expression `2*(3+4)` is entered in the calculator in its postfix form "`2 3 4 + *`". You can assume that the expression entered is correct and that the calculator accepts only the binary operators "*", "/", "+", and "−". Implements the class Calculator that takes in input an expression in postfix form and returns its value. Each postfix expression given in input ended with newline and its tokens are separated by spaces. It throws an exception for the cases of too many or too few operands. Your implementation should reflect the UML diagram in Figure 1.

4. Implement the ADT `Queue<T>` given in Unit 3 using a `CircularQueue<T>` data structure.

5. Consider the interface `DEQueue<T>` given in Unit 3, for a double ended queue ADT. Assume the existence of an ADT `DENode<T>`. Give the dynamic implementation of `DEQueue<T>`. Include also the implementation of an inner class `QueueIterator<T>` for iterating over the queue.

6. Consider a system for handling the waiting list of patients at a hospital. The hospital has four different levels of priority for patients. Each level of priority has an associated FIFO queue of patients. Only the patient name is stored in the queue. The diagram below shows an example of complete waiting list system with sample data. Figure 2 gives the UML design of the system.




Given an implementation of the following access procedures of `LinkedBasedWaitingList`:

(a) `public void addPatient(String name, int priority)`
//pre: the queue with given `priority` already exists
//post:patient `name` is added to waiting queue with given `priority`

(b) `public String getNextPatient(int priority)`
//pre: the queue with given `priority` already exists
//post: returns and removes patient next patient from the waiting queue of given
//post: priority. Message is returns if waiting queue is empty.

**Figure 1**

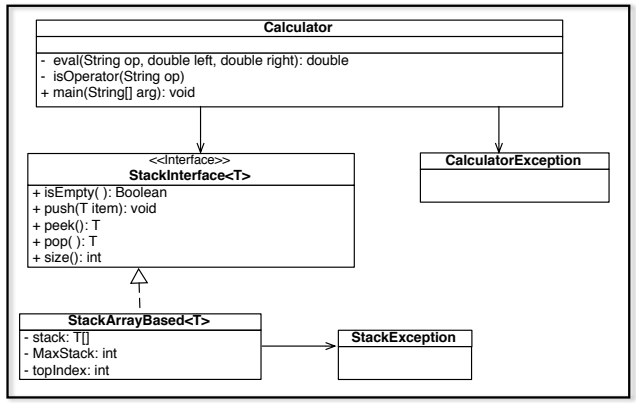



**Figure 2**