**Student Name: Weight: 15%**

**Student ID:** **Marks:** **/100**

# Assignment: Creating ADTs and Using an XML Parser

## Description

In this assignment, you’ll work in groups to write your own ADTs for a stack and a queue, and then create working versions of an array list, a double linked list, a stack and a queue. You’ll then use these utility classes to implement your XML document parser, which will read and confirm that the XML files are properly formatted.

## Equipment and Materials

For this assignment, you will need:

* Java IDE

## Instructions

This assignment consists of three parts, to be completed outside of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

#### Part A: Create and Implement Stack and Queue Interface Classes (95 marks)

1. Using the requirements and hints in the *Specifications* section below:
   1. Create your own stack and queue interface classes.
   2. Implement your utility classes for an array list, double linked list, a stack and a queue.
   3. Use the utility classes to implement an XML document parser.
2. There are **two** submission folders for this assignment. The requirements for each submission are outlined in the *Submission Deliverables* section of this document. Each of the two submissions has a separate due date. See Brightspace for exact due dates.

#### Part B: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

#### Part C: Complete an Evaluation as a Group

After completing the work outlined in Part A, check your work against the provided marking criteria. Your instructor will refer to your group’s self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group’s final grade.

1. Open the Marking Criteria document (MarkingCriteria\_Assignment2.docx) and save a copy with your group’s name.
2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.
3. Save this file for submission to Brightspace along with your completed code.

## Part A: Specifications

**Important:** Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

1. Write your own **StackADT.java** and **QueueADT.java** interface with all the required functionalities as method stubs with appropriate pre-conditions, post-conditions, return values and expected exceptions using proper Javadoc notations.
2. Create a complete set of JUnit tests in a class called **MyArrayListTests.java**.

**Note:** Ensure all tests fail before moving to the next step.

1. Write an implementation for the utility class **MyArrayList.java** using the supplied **ListADT.java** interface.
2. Completely test the implementation of **MyArrayList.java** for correct functionality using the set of JUnit test from the previous step.
3. Create a complete set of JUnit test functions in a class called **MyStackTests.java**.

**Note:** Ensure all tests fail before moving to the next step.

1. Write an implementation for the utility class **MyStack.java** using the **StackADT.java** interface provided by your instructor. Use your **MyArrayList.java** implementation as the underlying data structure.
2. Repeat steps 2–4 for the utility class **MyDLL.java** using the **ListADT.java** interface provided by your instructor.
   1. Your implementation should also include a **MyDLLNode.java** class.
   2. Name the complete set of JUnit test functions **MyDLLTests.java**.
3. Write an implementation for the utility class **MyQueue.java** using the **QueueADT.java** interface provided by your instructor. Use your **MyDLL.java** implementation as the underlying data structure.
   1. Include a complete set of JUnit test functions in a class called **MyQueueTests.java**, again using the TDD approach.

**Important:** Move to the next step only when you are satisfied that the data structures above perform properly. Then implement your XML document parser.

1. Write an XML parser that will accomplish the following:
   1. Read supplied XML documents.
   2. Parse for errors in the XML construction.
   3. When parsing is complete, print all lines that are not properly constructed in the order in which the errors occur.

**Important:** Do not create ANY additional files.

1. Ensure that your program can be supplied the XML document via the command line and show all results of the parsing on the console.

### Hints

An XML document is syntactically correct if it meets the following requirements:

* An opening tag has the format <tag> and a closing tag has the format </tag>.

**Note:** Although, XML tags may contain attributes in the format of name=“value” pairs, **ignore** these attributes for this assignment.

* For every closing tag, there is an earlier, matching opening tag.
* An exception to the above is a self-closing tag, which has the format <tag/>. Self-closing tags require no closing tag.
* The sub-phrase between a pair of matching tags is well-constructed.
* All tags are case-sensitive.
* Every XML document must have one and only one root tag.
* Tags in the format: <?xml somedata=”data”?> are processing instructions and can be **ignored** for this assignment.
* If nested, the tag pairs cannot intercross. For example, the following is not syntactically correct:

<b>This is to be bold and <i>italic</b></i>

### Warning

Implement this project using ONLY the libraries that you developed for the MyStack, MyQueue, MyArrayList and MyDLL.

You will be penalized **50% of your final mark for the assignment** if you use the java classes from the java.util.\*, javax.xml.\*, org.xml.\* or similar packages.

## Submission Deliverables

**Submission 1: Stack and Queue ADTs** **(14 marks)**

* 1. Ensure that your assignment .zip file includes the following items:
* Your StackADT.java
* Your QueueADT.java
* Your group’s completed Marking Scheme document for Submission 1.
  1. Name your group’s assignment submission folder as your group name followed by the assignment number (e.g., Group5Assign2-1.zip).
  2. Upload your group’s .zip file to Brightspace by the specified due date and time.

**Submission 2: Data Structures and XML Parser(81 marks)**

1. Ensure that your assignment .zip file includes the following items:

* An executable Java Archive file (.jar) for your sort application called **Parser.jar**.
* A **readMe.txt** file with instructions on how to install and use the XML Parser program.
* Your completed javadoc using the “-private” option when generated, and the output placed in the **doc** directory of the project.
* A folder containing the complete Eclipse project directory. At the root of the project directory, include a **readMe.txt** describing the completeness of the assignment (as a percentage) and a list of known deficiencies and/or missing functionalities.
* Your group’s completed Marking Scheme document for Submission 2.

1. Name your group’s assignment submission folder as your group name followed by the assignment number (e.g., Group5Assign2-2.zip).
2. Upload your group’s .zip file to Brightspace by the specified due date and time.