ASSIGNMENT TWO: ADT and TDD - XML Parser.

Description

First you will be required to write your own ADTs for a stack and a queue. Then you will create working versions of an array list, a doubly linked list, a stack and a queue. These will be the required utility classes used implement your XML Parser that will read and confirm that XML files are properly formatted.

Notes

- This is your second assignment and is worth **8.6%** of your final grade in this course.
- You are required to complete this assignment with your assigned group.
- No late assignments will be accepted.
- This assignment will be done in four phases:
    1. The first phase will be to create your own stack and queue interface classes.
    2. The second phase is to implement your utility classes for an array list and doubly linked list.
    3. The third phase is to implement your utility classes for a stack and a queue.
    4. The fourth phase will be to use the utility classes to implement a XML document parser.
- Below you will find the specifications for your application, along with some hints on how to proceed.
- Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

Specifications

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 test in a class called **MyArrayListTests.java**. Ensure all tests fails prior to the next step.
3. Write an implementation for the utility class **MyArrayList.java** using the supplied **ListADT.java** and **Iterator.java** interfaces supplied. The implementation of MyArrayList.java should be tested completely for correct functionality using the set of JUnit test from the previous step.
4. Create a complete set of JUnit test functions in a class called **MyStackTests.java**, ensure all tests fails prior to the next step.

5. Write an implementation for the utility class **MyStack.java** using the instructor-provided **StackADT.java** and **Iterator.java** interfaces. Your stack implementation should use your MyArrayList.java implementation as the underlying data structure and the EmptyStackException from java.util.
6. Repeat steps 2-3 for the utility class **MyDLL.java** using the same supplied **ListADT.java** and **Iterator.java** interfaces. Your implementation should also include a **MyDLLNode.java** class. The complete set of JUnit test functions should be called **MyDLLTests.java**.
7. Write an implementation for the utility class **MyQueue.java** using the instructor-provided **QueueADT.java** and **Iterator.java** interfaces. Your queue implementation should use your MyDLL.java implementation as the underlying data structure. Include a complete set of JUnit test functions in a class called **MyQueueTests.java**, again using the TDD approach. You must created your own EmptyQueueException class.

Once you are satisfied the above data structures perform properly you can then implement your XML document parser.

8. Write an XML parser that will accomplish the following:
    a. Read supplied XML documents.
    b. Parse for errors in the XML construction.
    c. At completion of parsing, print all lines that are not properly constructed in the order in which the errors occur. NO EXTRA files may be created.

    An XML document is syntactically correct if:

    i. An opening tag has the format <tag>, a closing tag has the format </tag>.
    ii. XML tags may contain attributes in the format of name="value" pairs, these attributes are to be ignored for this assignment.
    iii. For every closing tag, there is an earlier matching opening tag.
    iv. An exception to the above is a self-closing tag, a self-closing tag has the format <tag/>. Self-closing tags require no closing tag.
    v. The sub phrase between a pair of matching tags is itself well-constructed.
    vi. All tags are case-sensitive.
    vii. Every XML document must have one and only one root tag.
    viii. Tags that are in the following format <?xml somedata="data"?> are processing instructions and can be ignored for this assignment.
    ix. If nested, the tag pairs cannot intercross. For example, the following not syntactically correct: (ill-constructed)

        This is to be bold and italic

9. Your program will be supplied the XML document via the command line and show all results of the parsing on the console.

Warning

You are required to implement this project using ONLY the libraries that you developed for the MyStack, MyQueue, MyArrayList and MyDLL. With the exception of exception classes, if you resort to using the java classes from the java.util.*, javax.xml.*, org.xml.* or similar packages you will be **penalized 50%** of you final mark for the assignment.

Submission Deliverables

There are **two separate submission dates** for this assignment:

1. Part 1 – Stack and Queue ADTs:
   - Assignment zip file will be uploaded into **D2L** by the specified due date and time, also named as the assignment number and your group number – i.e. A1Group3.zip.

   The assignment zip file will include the following:
   a. Your StackADT.java
   b. Your QueueADT.java

2. Part 2 – Data structures and XML parser:
   - Assignment zip file will be uploaded into **D2L** by the specified due date and time, also, named as the assignment number and your group number – i.e. A1Group3.zip.

   The assignment zip file will include the following:
   a. An executable Java Archive file (.jar) for your sort application called **Parser.jar**.
   b. A readMe.txt file with instructions on how to install and use the XML parser program.
   c. The project should have completed javadoc using **"-private"** option when generated, with output placed in the doc directory of the project.
   d. A folder containing the complete Eclipse project directory.
      - At the root of the project directory, include a readMe.txt to describe the completeness of the assignment (as a percentage) and a list of known deficiencies and/or missing functionalities.

Submission Criteria

Your program must execute. Code that doesn't run/compile will not be accepted.

No late Assignments will be accepted and will be assigned a mark of 0%.

You are expected not merely to solve the problems, but to make intelligent decisions about how to approach the problems. Remember, it is not enough to submit adequately coded programs that merely produce the expected output, but should be elegant, designed legibility, have maintainability, and other such important factors.

Please note that proficiency in using official Java API Doc can answer a lot of questions you may have about using some standard Java classes and methods, and many other types of programming which you will encounter, both at SAIT, and beyond.

Criteria for Marking Assignment

Logic for parsing tags		/	**2**
Program produces the correct output		/	**1**
StackADT Interface contains proper functionality and documentations		/	**2**
Stack implemented properly using MyArrayList data structure		/	**3**
Implementation of MyArrayList		/	**3**
QueueADT Interface contains proper functionality and documentations		/	**2**
Queue implemented properly using MyDLL data structure		/	**3**
Implementation of MyDLL		/	**3**
Complete sets of JUnit tests			
- ArrayList		/	**3**
- DLL		/	**2**
- Stack		/	**2**
- Queue		/	**2**
Readmes/Instructions and javadoc documentation		/	**2**
Total:		**/**	**30**

Final Individual Grade:

Team Total (30)	*	Peer Evaluation Multiplier (as a percentage)	=	Subtotal (30)	Final Score
	*		=		
					+
Peer Evaluations Completed (3)					
					=
				Final Grade (33)	