Extra Credit: Stacks

**Objectives:**

The focus of this assignment is understanding of the Stack structure and implementing one of its uses. Use the Stack class that is defined in the java API, in the java.util package.

**Program Background:**

A **molecule** is a grouping of two or more atoms. A molecule can be written as a sequence of the symbols of its atoms. For example, H denotes an atom of hydrogen, C denotes an atom of carbon, and O denotes an atom of oxygen. **Formic Acid** is defined by the sequence HCOOH, which is a hydrogen, carbon, oxygen, oxygen, hydrogen. If a digit is placed after a symbol it means there are that many of those atoms in the molecule, such as **water**, H2O is two hydrogen atoms and an oxygen. Digits can also be placed after groupings such as (CH)3 which is three carbon/hydrogen groupings, (CH)(CH)(CH). The weight of a molecule is simply the weight of all the atoms in it. Given the weight of hydrogen is 1, carbon is 12, and oxygen is 16, Formic Acid has a weight of 46, Water has a weight of 18, and (CH)3 has a weight of 39. More complex molecules sequences can be created by nesting groupings such as H((OH)2C3H)2, which has a weight of 143.

Parenthesis and other types of braces must follow rules in order for an expression to be valid. Stacks are often used for processing of parenthesis in text, such as compilers making sure braces match up correctly. A Stack can also be used to evaluate parenthesized expressions like we have here.

Every opening and closing parenthesis must have a matching of the other. It is also important to note that in valid text the matching opening parenthesis to a closing one is **last one** encountered. This is what makes the Stack the perfect structure for such problems. Open parenthesis signify the beginning of a grouping, nested groupings can be added to the stack. Once a closing parenthesis is reached it closes the group at the top of the stack, which can then be evaluated.

**Program Description:**

The program will accept a number of molecule sequences and report their weights.

The following classes are required:

* InvalidAtomException
* InvalidSequenceException
* Molecule
* MoleculeDriver

**UML DIAGRAM FOR AND DISCUSSION FOR InvalidAtomException**

|  |
| --- |
| InvalidAtomException extends RuntimeException |
|  |
| <<constructor>> InvalidAtomException(atom : char) |

Methods:

InvalidAtomException(atom : char) – Class constructor. Calls upon super class constructor with the String, “Invalid atom value: “, followed by the character that is passed.

**UML DIAGRAM FOR AND DISCUSSION FOR InvalidSequenceException**

|  |
| --- |
| InvalidSequenceException extends RuntimeException |
|  |
| <<constructor>> InvalidSequenceException() |

Methods:

InvalidSequenceException( ) – Class constructor. Calls upon super class constructor with the String “Invalid molecule sequence“.

**UML DIAGRAM FOR AND DISCUSSION FOR Molecule**

|  |
| --- |
| Molecule |
| * sequence : String * weight : int |
| <<constructor>> Molecule(seq : String)  + toString() : String |
| + atomWeight(atom : char) : int  + calculateWeight() |

Data Members:

sequence – String representation of the Molecule

weight – Weight of the Molecule

Methods:

Molecule(seq : String) – Class constructor. Sets data member, sequence, to the String passed. Calls upon calculateWeight method.

toString( ) : String – Returns a String with the Molecule sequence formatted with width of 25, left justified, followed by the molecule weight, separated by a space padded colon. Example:

H((OH)2C3H)2 : 143

atomWeight(atom : char) : int – Accepts a char representing an atom symbol, returns the weight of that atom. We are only using hydrogen, carbon, and oxygen, with the respective weights of 1, 12, and 16. If any other character is passed this method with throw an InvalidAtomException, giving the character passed to this method to its constructor.

calculateWeight( ) – Sets the weight data member using the Molecule’s sequence. The algorithm is described below. For this problem we need to have a Stack that holds both Integers and something to indicate an open parenthesis. Since all values will be positive a -1 will be used to represent an open parenthesis.

1. Create a Stack of Integers

2. Create a new sequence String with ( ) surrounding it (if original sequence is H20, create new local String (H20), DO NOT change the original sequence data member

3. For each character in the new sequence String from left to right do the following:

If the character is an open parenthesis push it onto the Stack

If the character is an atom push it onto the Stack

If the character is a digit pop the top value off the Stack and multiply this by the digit, push the result onto the

Stack

If the character is a close parenthesis pop all values off the Stack, summing them up as you go until an open

parenthesis is reached, pop the open parenthesis off the Stack, push the sum onto the Stack

4. The final weight will be the only item left on the Stack

Invalid Sequences

This algorithm assumes the sequence String is constructed correctly. I suggest writing it this way first, then altering it to handle invalid sequences. The following are examples of invalid sequences:

)(HOC)2

(OH

(((H))))

You do not have to test for the case of multiple digits in a row, something like (OH)23 will not be tested.

**DISCUSSION FOR MoleculeDriver**

See expected output below discussion, if your output is different in any way you are subject to losing points.

The user is asked for the number of Molecule sequences they would like to enter, check for InputMismatchException and ensure value is greater than 0, reprompt until a valid int is entered.

Create an array of Molecules of the given size.

Ask the user for sequences **stating which sequence number** they are entering as shown in the example output.

Create an instance of Molecule and place it in the array. If the Molecule constructor throws an exception output what was wrong by outputting the exception’s message and reprompt them for another String for *the same* Molecule. That is, don’t skip it and go to the next. Do not move on to the next Molecule until a valid one has been created.

Output all of the Molecules.

The example output below contains examples of both valid and invalid inputs and how they should resolve. These cases are a good place to start your testing, however you should make your own test cases as well.

**Helpful methods:**

You may find the following methods in the java API helpful

Character class methods:

public static int getNumericValue(char c) - Returns the int value that the specified Unicode character represents. For example, the character '\u216C' (the roman numeral fifty) will return an int with a value of 50. The character ‘6’ will return an int with the value of 6. If the character does not have a numeric value, then -1 is returned.

public static boolean isDigit(char ch) – Determines if the specified character is a digit.

**Example Run (User input underlined)**

Number of molecules: -5

Number of molecules: 0

Number of molecules: 6

Enter sequence 1: H2O

Enter sequence 2: hcooh

Enter sequence 3: H((OH)2C3H)2

Enter sequence 4: ((CH)2(OH2H)(C(H))O)3

Enter sequence 5: ((((O2))))()

Enter sequence 6: )(

Invalid molecule sequence

Please try again

Enter sequence 6: (COOH

Invalid molecule sequence

Please try again

Enter sequence 6: (OOH))

Invalid molecule sequence

Please try again

Enter sequence 6: DOH

Invalid atom value: D

Please try again

Enter sequence 6: ((CO)2H)3

H2O : 18

hcooh : 46

H((OH)2C3H)2 : 143

((CH)2(OH2H)(C(H))O)3 : 222

((((O2))))() : 32

((CO)2H)3 : 171

**Assignment Questions**

Provide the answers to these questions in your submission directory within a file called AssignmentECQuestions.

1. As is, this program only works for molecules made of hydrogen, carbon, and oxygen. Which class(es) and method(s) would need to be changed and how in order for it to work with all stable atoms on the Periodic Table of Elements?

2. Why did you have to put parenthesis around the original expression in order for the given algorithm to work?

3. Create six of your own test case sequences, some valid, some invalid. What were your test cases and their results?