COMP 250 - Homework #3 Mathieu Blanchette

Due on March 7th 2012, 23:59, via MyCourses.
The java code file for question is to be submitted electronically on MyCourses.
Solutions to questions 2-5 must also be submitted on MyCourses as a PDF.

IMPORTANT NOTES:

- For questions 1 and 2:
    - Submit only your Expression.java and CardDeck.java files
    - Do not change the class name, the method names, or the methods' arguments.
    - Do not include packages
    - You will get zero for these two questions if you violate any of the above, because this will break our automated marking program
- For questions 3 and 4: Submit a single PDF document.

Question 1 (50 points)

The problem is to write a Java program that reads from the keyboard a line of text describing an arithmetic expression and then evaluates it. An expressions is made of any of the following elements: non-negative numbers, operators "+", "-", "*", and "%", as well as parentheses "(" and ")". The meaning of the "%" operator is integer division, not modulo like in Java. Usually, operators "*" and "%" have a higher priority than "+" and "-", but for this homework we will assume that all operators have the same priority and that an expression can thus be evaluated from left to right.
Here are some examples:

Input: 5
Output: 5

Input: 2+3*5
Output: 25 (because 2+3 is calculated first)

Input: 2+(3*5)
Output: 17

Input: (2+4)*(3+(19%3))
Output: 54

Input: 2*(((((1+1)))))
Output: 4

Input: 2+3*
Output: Malformed expression

Input: 2+((3+4)*5
Output: Malformed expression

Start from the class skeleton available at http://www.cs.mcgill.ca/~blanchem/250/hw3 and implement the public method "int evaluate(String s)" that returns the value of expression s. If the expression s is malformed, the method should throw an exception that will be caught and handled properly inside your main method, printing the message

"Malformed expression". You can assume that the input strings will only contains characters 0,1,2,3,4,5,6,7,8,9,+,-,*,%,(,). Notice to make our life easier, we will assume that operators +, -, *, and % are all binary operators, so the expression "-5" would be a malformed expression. Negative numbers would have to be written as "0-5".

To write this program, it will help if you use two different stacks: one for the numbers and one for the operators. Think carefully about your algorithm before writing your code; the code is fairly simple (probably less than 70 lines), but if you do it the wrong way it will be much more complicated.

Some Java classes that will be useful:
The Java class Stack is a predefined class implementing a stack. It is described clearly in the Java API: https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html. Only objects of non-primitive types can be stored in a Stack. Thus, one cannot have a stack of int, but can have a stack of String or of Integer. To define a Stack of Strings, write:
Stack<String> myStack = new Stack<String> ();

A few words on the class Integer: an object of type Integer is actually an int put inside a class. It has a number of methods that will be useful for you. See http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html?is-external=true. Pay particular attention to the methods "int intValue()" and "static Integer valueOf(String s)". You will probably want your numbers to be stored in a Stack of Integer and your operators to be store in a Stack of String (or Character).

Finally, you will find useful to use the StringTokenizer class to take the expression String and break into several parts, each being a single operator or number. Again, see the Java API for the description of the StringTokenizer class. An example is also provided in the skeleton given to you.

Question 2 (15 points)

Consider the following "magic" trick. You have a deck of *n* cards, labeled 1,2,...,*n* (but not necessarily in that order).
You then repeat the following process until no cards are left: (i) Show to the public the card on the top of the deck, and remove it from the deck, and (ii) Take the next card from the top of the deck and place it at the bottom of the deck, without showing it. Your goal is to have previously ordered the cards in the deck so that the cards shown to the public are in increasing order: 1, 2, ..., *n*. For example, if *n*=5, then starting from the arrangement 1,5,2,4,3 would work: 1,5,2,4,3 -> 2,4,3,5 -> 3,5,4 -> 4,5 -> 5

Question: Download the CardDeck class:
http://www.cs.mcgill.ca/~blanchem/250/hw3/CardDeck.java

Implement the setupDeck(n) method in the CardDeck class so that it returns a new CardDeck with n cards arranged in such a way that when the runTrick method is executed, cards are shown in order.

Question 3. (15 points)

Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 * T(n - 1) + n + 1 & \text{if } n > 1 \end{cases}$$

a) (10 points) Obtain an explicit formula for the following recurrence using one of the techniques seen in class. In the process of doing so, you will possibly come across the summation $\Sigma_{i=1..n} (i * 2^i)$, which can be simplified as $2 * (1 + 2^n * (n-1))$.

b) (5 points) Using induction, prove that your explicit formula always takes the same values as the recurrence, for all values of $n >= 1$.

Question 4. (10 points)

Prove that $\log(n!) \in \Theta(n \log(n))$ (that big Theta, not big O)