University of New Brunswick
CS1083 – Open Access Version

Module 10 Assignment

The purpose of this assignment is to give you practice:

- Writing a Stack class; and
- Using the Stack class to solve a mathematical problem.

For all assignments:

- Follow the instructions in the document "Java Coding Guidelines.pdf" available within the "Start Here" module, under the "Assignments" topic.
- For each .java file you create, include a javadoc comment (one that begins with /**) at the beginning of the file that describes that Java class. This comment block should include a line that begins with @author followed by your name and student number on the same line. (Note: inclusion of this comment is part of the instructions given in "Java Coding Guidelines.pdf")
- Include comments throughout your programs to explain any non-obvious portions of your code.
- It is recommended that you create a separate folder on your computer for each assignment. You might even wish to create separate sub-folders within an assignment folder if the assignment has multiple parts. Keeping your work organized makes it easier to find things later when you want to review what you have done.
- Few things in life are more frustrating than losing your work while working on an assignment. Get in the habit of saving frequently when working on an assignment. Also, regularly make backup copies of any files you create as part of your work for this course.
- Each module has a link to the discussion forum where you can ask questions and seek help in completing your assignments.
- Submitting your assignment involves creating a pdf file and uploading that single file to the assignment drop box on D2L. In general, that file will tend to include all Java code you wrote for the assignment, plus any output from running your programs that the assignment instructions specify you should capture. Specific submission instructions are included at the end of each assignment.
- To create the pdf file for each assignment, begin by opening a new document using the word processing program of your choice. Save the document using the name “Your Name CS1083 Module x Assignment Submission.docx” Replace x

with the correct module number. “docx” may be different depending on which word processing software you choose to use.

- At the beginning of your submission document enter your name, your student number, CS1083, the assignment name (this one is “Module 2 Assignment”), and the date. It doesn’t matter if the date is when you started working on the assignment or when you submit it – either will do.
- You can add content to your submission document as you work on the various questions in the assignment. Clearly label each part of the assignment (“Part A” etc.) in your document. When your document is complete, save / export as a pdf file. Always make sure your pdf file opens properly before uploading it.
- To include Java code in your submission document, copy all the text in your .java file and then paste that text into your submission document. Use a monospaced font (e.g.: Consolas, Courier) for your code to maintain proper indentation.
- To include output from running your program in your submission document, you have two choices. You can either (a) copy and paste the text from your command prompt window, or (b) capture a screen shot of the output and include that as a picture / image in your submission document. Either approach is fine.
- To copy text from your command prompt window, try selecting the desired text and then pressing either command-c (Mac) or control-c (Windows or Linux). If you have issues, you can always use Google to see how to do this on your specific type of computer.
- To capture a screen shot of a selected portion of your command prompt window, try command-shift-4 (Mac), WindowsKey-shift-s (Windows), or shift-PrtScrn (Linux).
- Once you have pasted your output text into your submission document, make sure it is in a monospaced font (e.g.: Consolas, Courier) so you retain the alignment of output.

Part A – A Stack Class

For the Module 9 Assignment you created a doubly linked list solution that includes the following classes:

- **Node.java** – Contains a person's name as a String.
- **LinkedList.java** – Manages a doubly linked list of Node objects.

Use these two classes (with no changes) for this Module 10 Assignment.

Write a **Stack** class that extends your LinkedList class to represent a stack of Node objects. Assume the tail of the list is the top of your stack.

The Node class uses an instance variable called "name" for the String value inside each node, but for this assignment we're going to treat it as a stack of strings … which could be any type of String values.

The Stack constructor simply calls the superclass constructor to create an empty stack (which is actually an empty LinkedList object, but our Stack class makes it behave like a stack).

The push() method accepts a string as a parameter and calls the insertAtTail() method from the superclass to place this string on top of the stack.

The pop() method:

1) Removes the node from the top of the stack; and
2) Returns the string from that removed node.

If pop() is called for an empty stack, then there is no node to be removed and the method returns null.

The peek() method returns the string from the node on top of the stack (or null if the stack is empty). The contents of the stack are unchanged.

Write a simple **StackTest** class with a main() method that does the following. Label all output so it is understandable:

- Creates an empty Stack instance;
- Pushes any three strings of your own choosing onto the stack;
- Calls displayInReverse() to display the contents of the stack (this method is inherited from the LinkedList class);
- Uses getSize() to display the number of items on the stack (another inherited method);
- Pops one string off the top of the stack and displays the popped string; and
- Repeats displayInReverse() and displays the result of getSize() again, which should confirm that the pop() method worked.

Part B – Prefix to Infix

The most commonly used form of mathematical expression is called infix, where the operator appears between two operands. For example, 4+7 represents the addition of 4 and 7.

One of the downsides of infix notation is that parentheses are sometimes necessary to clarify order of operation. For example, 5+5/5 is by default interpreted by Java as 5+(5/5), which evaluates to 6. With parentheses, however, the same operators and operands can be changed to (5+5)/5, which evaluates to 2.

As an alternative way to state mathematical expressions, prefix notation places each operator *before* its two operands. For example, the infix expression 4+7 is represented in prefix as +47. One advantage is that neither parentheses nor precedence rules are required; it is always clear which operator to evaluate first.

Here are some additional examples, using parentheses around every infix expression to ensure the meaning is clear. Each individual digit is treated as an operand:

```
prefix: +47
infix:  (4+7)

prefix: *+23-85
infix:  ((2+3)*(8-5))

prefix: *-9/41-/763
infix:  ((9-(4/1))*((7/6)-3))

prefix: +-*1234
infix:  (((1*2)-3)+4)

prefix: +3*-/59+821
infix:  (3+(((5/9)-(8+2))*1))
```

This shows more than just mathematical examples – this is sample output from the program you are to write for Part B of this assignment.

Converting prefix expressions to infix is made possible by using a stack as follows:

- Represent expressions as strings. For example, "4", "+47", and "(4+7)" are all String values that represent expressions;
- Process each character in the given prefix expression, one at a time, in reverse order. For example, in processing "+47", loop to process the "7" first, then the "4", and finally the "+";
- If the character is a digit (which represents an operand), push it onto the stack;

- If the character is an operator (+, -, *, or /), then pop two strings from the stack. These strings are operands. An operand might be a single digit, or it could be a more complex expression that has previously been pushed onto the stack. Form a string with these components in this order: "(" followed by the first operand from the stack, then the operator, the second operand from the stack, and finally ")". Push this string onto the stack.
- Repeat those steps until all characters in the prefix expression have been processed, at which point the only string on the stack should be the result – an infix expression that is equivalent to the given prefix expression.

Write a **PrefixToInfix** class that includes a method called **convert**. This method accepts a prefix expression (a String value) of the form shown in the examples on the previous page. Use the algorithm above to return a String with an equivalent infix expression. Use the Stack class you wrote for Part A.

Include in the PrefixToInfix class a main method to test your convert method. Create test expressions as follows:

```
String[] testExpression = {"+47",
                           "*+23-85",
                           "*-9/41-/763",
                           "+-*1234",
                           "+3*-/59+821"};
```

These are the examples provided on the previous page. Loop through such an array of String values to test your convert method with these exact expressions, plus at least two more prefix expressions of your own. Produce output of the format shown on the previous page.

Useful methods for this program are likely to include charAt() from the String class, and isDigit() from the Character class.

Assuming you extract each character from the prefix expression as a value of type char, you will face the challenge of how to convert that char value to a String so you can push it onto the stack. The following might offer a helpful hint about one possible way to accomplish this conversion:

```
char c = '1';
String s = ""+c;  // Results in the String value "1"
```

(Submission instructions are provided on the next page)

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 10 Assignment, and the date
- Part A:
  - Complete code for your Stack and StackTest classes
    (The Node and LinkedList classes are unchanged, so these do not need to be included in your submission document)
  - Output from executing StackTest
- Part B:
  - Complete code for the PrefixToInfix class
  - Output from executing PrefixToInfix

D2L DROPBOX SUBMISSION INSTRUCTIONS

Remember to save / export your submission document as a pdf file, and then upload that one pdf file to D2L as follows:

1. In the top-navigation bar on the course screen, select 'Assessments' and then 'Assignments'.
2. Select the assignment title and follow the instructions to upload your submission document.