Assignment 6 – on linked lists and stacks

Due: Thursday April 15, 12pm (noon)

IMPORTANT: individual work please!

What/how to submit:

- For Part 1 (linked lists), you should submit both a PDF file in D2L and through GitHub. Use the following link to create your GitHub repository, and make sure you push your code to it before the deadline: https://classroom.github.com/a/Ef_U5_ik

- For Part 2 (stacks), you should only submit your code as a single PDF file (separate from the one above for linked lists) through D2L.

Part 1 – linked lists

Take the example DoubleNode.java and DoublyLinkedList.java that we did in class. Add a method "void sort()" in the DoublyLinkedList class, that implements the selection sort algorithm on the linked list. Note: you are not allowed to convert the linked list into an array, and then do the sorting on an array. All the work should be done on pointers. Also, you are not allowed to use the insert and remove methods already implemented, or have code that is similar to doing such operations in sequence: the swapping has to handle the both the removal and inserting at the same time (i.e., no element should be completely unconnected from any other element of the list, at any time while performing the swap). A driver class (SortDoublyLinkedList.java) is provided to you for testing purposes.

Hints:

- Print your list (forward and reversed) at each iteration to see exactly where/when mistakes are inserted.
- Finding the minimum of the rest of the list is just a matter of list traversal from the point where you are at. The difficulty is with the swapping of elements.
- If the list is empty, there is no sorting necessary.
- If the minimum value is already in the correct position, there is no swapping necessary.
- Swapping 2 consecutive elements is different than swapping 2 elements that have other nodes in between.
- Be careful when you swap with first or last element.
- When swapping, I recommend that you first save (in separate variables, like the "prev" and "next" nodes in the "remove" method) all nodes that will be affected: the 2 nodes to swap, and their prev/next nodes. Then I recommend

that you fix the entire sequence on next pointers, followed by fixing the entire sequence of prev pointers.

Part 2 – stacks

For this part, you have to write a program that reads a formula, evaluates it, and prints the result. The formula contains integers and operators, each separated by a single space. The possible operators are: + - * /. However, we will use a template for the formula that is different than what we are used to…this will actually allow us to do the calculation in a much simpler way. We are used to formulas where we get two operands separated by an operator (e.g., "5 + 2"). In the format used here, we will instead have the 2 operands first, and then only the operator (e.g., "5 2 +", which still means that you have to add 5 and 2 together). The advantage of this notation is that you no longer need to worry about the order of the operations (i.e., for example that you should first do the * and / before doing the + and -). There is no need for parentheses anymore either. Here are more complex examples:

"6 + 10 / 2" → "10 2 / 6 +"
"(6 + 10) / 2" → "6 10 + 2 /"
"2 + 3 * 1 – 9" → "2 3 1 * + 9 -"
"((1 + 2) * 3 + 6) / (2 + 3)" → " 1 2 + 3 * 6 + 2 3 + /"

The way to do the calculation is as follows: when you see a number, push it on a stack. When you see an operator, pop 2 numbers from the stack (second operand, and then first operand), apply the operator on those 2 operands, and then push the result you got on the stack. At the end, your stack will contain only one number, and that is the final result of this calculation. Write a program that reads such a formula (as one string), and prints the result on the screen. Use the LinkedList class of the Java API (in java.util) for your stack. Note: you can check if a string str is actually an integer using the following code: "if (str. matches("\\d+"))" → returns true if the string contains only a sequence of digits.