Worksheet 1: Array-based Stacks

The goal of this worksheet is get some experience with Abstract Data Types and, in particular, Stacks. In the lectures, we went through the code for an array-based integer stack, which was implemented in a class called IntegerStack.

In this worksheet, you will develop alternative implementations of array-based stacks that cater for different types of data (e.g. doubles, Strings, ...). For each problem, you will be required to first adapt the Stack code and then create a simple test program that demonstrates that your answer is correct.

If you are unsure that the answer is correct, I recommend that you sketch out the answer with a pen and paper (working out the state of the stack after each operation) and use the provided toString() method to compare the state you have written on paper with the state of the stack program (this is a good general debugging technique).

As with all practicals, some questions are for practice, but you should submit: Q1, Q3. Each question is worth 50% and there is 30% for the updated stack and 20% for the main method (you should write this in the same class as the stack implementation).

1. Adapt the stack solution for **double** values by developing a class called **DoubleStack** and write a main method that performs the following operations:

   Push(3.2), Pop(), Push(4.2), Push(3.0), Push(2.6), Pop(), Push(1.2), Pop(), Pop()

2. Adapt the stack solution for **String** values by developing a class called **StringStack** and write a main method that performs the following operations:

   Push("mat"), Push("the"), Push("the"), Pop(), Push("on"), Push("sat"), Push("pat"), Pop(), Push("cat"), Push("the"), Pop(), Push("The")

   Once you have completed this list of operations, pop all the values off the stack until it is empty and print each value out in the order they are popped. What is the sentence that is created?

   NOTE: Because this stack deals with objects, you must modify the pop() method slightly. This is because when you use objects, you have to make sure that any reference to the object is removed. The danger of not doing this is that the link forces the JVM to keep the object in memory (even though it may no longer be used). This can result in a memory leak that will cause your program to crash after a period of time.

   A sketch of the pop method in pseudo code is provided on the next page.

Algorithm pop():
    top ← top - 1
    temp ← array[top]
    array[top] ← null
    return temp

3. Adapt the stack solution for **char** values by developing a class called **CharStack** and write a main method that performs the following operations:

   Push('Y'), Push('P'), Push('P'), Push('A'), Push('H'), Pop(), Pop(), Pop(), Pop(), Pop()

   Print out each value that is popped using System.out.print(...). What does this code do? HINT: Think of the input as the characters from a word. Write a comment at the start of the main() method containing your answer.

4. Adapt the stack solution for **float** values by developing a class called **FloatStack** and write a main method that performs the following operations:

   Push(13.2), Push(4.2), Push(3.0), Push(2.6), Pop(), Pop(), Push(1.2), Pop(), Pop(), Push(5.4), Push(6.9)

   After you have performed all these operations add some code that pops the remaining values from the stack and prints out the total.

5. Adapt the stack solution for **long** values by developing a class called **LongStack** and write a main method that performs the following operations:

   Push(2), Pop(), Push(4), Push(3), Pop(), Push(6), Push(12), Pop(), Push(5), Push(9), Pop(), Push(3)

   After you have performed all these operations add some code that pops the remaining values from the stack and prints out the average of the values that were in the stack.