CSCM12J Lab 3

Manipulating objects and stacks

Before you start

**Practical notes:** 20 marks will be associated to a lab. 10 marks will be given if you are genuinely attending, and 10 other marks will be given by judging the progress made. Any genuine issue (e.g. medical ones) preventing attendance must be mentioned to us. We do not expect you to solve all the tasks but solving at least 2 of them would be a good achievement. We will go around and monitor progress being made after 90 minutes but please raise your hand before that if you have questions. Please download the java file from the course webpage before starting the coding phase.

You can use your own computer and/or compile code online at <https://www.onlinegdb.com/> (choose java) or <https://repl.it/>.

For compiling with a command line: javac Lab1.java

For running the compiled program: java Lab1

Lab Class Description:

We will first add standard functions to a stack class. This class may help with the notion of objects, interfaces, and stack data structures. Our stack will be made of items, with the class *Item* already given to you. Everything should be implemented using your own code and without using external functions for handling stacks

# Task 1: Complete the standard stack features

*Stack* objects have some usual standard methods as part of their interface. You will have to implement the following ones:

**Stack**

**Input**: max\_items, the number of items in the stack

**Output**: The same stack object will all the internal variable setups. No value returned as we have a constructor function here.

**push**

**Input**: the item one wants to push to the stack

**Output**: no return value but item added to the top of the stack.

**pop**

**Input**: nothing

**Output**: no return value but the top item is removed from the top of the stack.

**top**

**Input** nothing

**Output**: the item on top of the stack.

**isFull**

**Input** nothing

**Output**: a Boolean stating whether the stack is full.

**isEmpty**

**Input** nothing

**Output**: a Boolean stating whether the stack is empty.

Optionally, you can write a function that prints the stack as well if it helps.

# Task 2: Use case of a stack

We would like to test code and see if brackets are correctly nested. This will be coded inside the *testBracket* function. For instance,

*testBrackets*("public static void main(String[] args){}") will return true

and

*testBrackets*("public static void main(String[] args)}") will return false.

The algorithm to solve the problem is linear and will process each character of the string one by one. If one of the three types of brackets is encountered, the corresponding bracket must be pushed to the stack if it is an opening one; and compared to the top of the stack if it is a closing one. The figure below (not related to examples above) shows more or less how it works.

