CC/CS21120: Workshop 2
– Stacks and Matching Brackets –

(Program Design, Data Structures and Algorithms)

Assessed	2.5% of module marks. Signed off during timetabled workshops; submission to Blackboard
Submission	Blackboard → Assignments → Worksheets → Worksheet 2
Deadline for sign-off	During one of the **timetabled** workshops in the **week starting 15th October 2018** (or by the end of the semester if your work was submitted to Blackboard by the below submission deadline).
Deadline for submission to Blackboard	**4pm on 19th October 2018**

A note on gaining marks:

- To gain marks, each worksheet must be signed off during one of the timetabled workshops. A worksheet will only be signed off if you can explain what the code does. If we are not convinced we will not sign it off.
- If you are unable to get the sign-off in the corresponding workshops, please submit your solution to Blackboard by the above submission deadline and get it signed off as soon as possible. You will not get any marks for sign-offs after the given deadline if you do not submit your solution to Blackboard!
- We strongly recommend that you submit your solution to Blackboard even if you get signed off by the given deadline as this will represent a proof that you have done the work.

1 Recommended Preparation

In this worksheet you will implement your own array-based character stack and a program that checks a given text for correct usage of brackets using this stack. You should particularly revise basic data structures (Workshop 1) and stacks (Lecture 2). Please do not hesitate to ask any of the demonstrators or the teaching staff if you have any questions.

2 Tasks

1. Download the code provided on Blackboard and create a new project in your preferred IDE.
2. The existing program contains the following files:
    - The file `Stack.java` provides you with an interface for a stack. You need to implement this interface for Task 1.

- The file `BracketChecker.java` provides you with an interface to perform the bracket checking. You need to implement this interface for Task 2.
- The file `BracketDemo.java` provides you with a framework to run and test your implementation. It lets you input a text to be checked. There is no need to modify this class for this task.

2.1 Task 1: Implementing a Stack (1%)

You are first asked to implement the `Stack` interface provided on Blackboard using the array-based implementation discussed in Lecture 2.

1. Create a new class called `ArrayStack` that implements the `Stack` interface, i. e., `public class ArrayStack implements Stack`
2. You need to implement all methods defined in the interface. You can use the examples on the lecture slides and the below hints as a starting point. You can create the methods stubs automatically in Intellij via Code->Generate...->Implement Methods...
   - Add two private fields: the data array (`Character[] data`) and an integer (`int top`) that keeps track of the top of the stack. You can also use `top` to determine if the stack is empty or full.
   - Create a constructor that instantiates a new data array of a given capacity and initialises `top`. The demo provided currently sets the capacity to 1000.
   - `push`: You need to check if the stack is full. If not, add the element to the next available cell and increment `top`. Otherwise throw an exception.
     For the latter you should change the signature of the method to `public void push(Character element) throws IllegalStateException`. You can throw a new exception by using `throw new IllegalStateException();`
   - `pop`: You need to check if the stack is empty. If not, decrement `top`, return the top element of the stack and make sure the cell content is set to null to facilitate garbage collection. Otherwise, return `null`.
   - `peek`: Return the top element of the stack (or `null` if the stack is empty).
   - `isEmpty`: Check if the stack is empty.
   - `size`: The number of elements currently on the stack. You can derive this from the value of `top`.
3. Don't forget to test your implementation.

2.2 Task 2: Checking Correctness of Brackets (1.5%)

The correct use of matching brackets is crucial when writing code or stating mathematical formulas. The goal of this task is to write a program that checks a given text for correct usage of different brackets. This means:

- Each opening bracket of a specific type must have a matching closing bracket of the same type.

- For each pair the opening bracket appears before the closing bracket.
- For any two pairs of brackets, either one is nested inside the other or they do not intersect.

Apart from the brackets, the given text can contain big and small latin letters, digits, spaces and punctuation marks.

Examples

Correct usage of brackets:	()	{}[]	{()}	a[i]		
Incorrect usage of brackets:	)	(	(]	{[]	{(})	)[]

Guidelines

1. We assume that the given text contains a combination of round, curly and square brackets that need to be checked.
2. Create a new class called `StackBracketChecker` that implements the `BracketChecker` interface. The function `public boolean check(String text)` needs to provide the required functionality and you need to use your stack from Task 1.
3. Add a private field `Stack unmatchedBrackets` to your class and create a constructor that instantiates a new `ArrayStack` with a given capacity.
4. **Hints:**
   - You need to iterate over the input String. You can access the i-th character of a String by using `text.charAt(i)`. Ignore all characters that are neither opening nor closing round, curly or square brackets.
   - When you see an opening bracket push it onto a stack to keep track of the type of opening brackets you have seen.
   - When you see a closing bracket, check if its type matches the last seen opening bracket (if there is one). If it does, make sure you have removed the bracket from the stack and continue. Otherwise, you have found an error in the given input (incorrect pair of brackets) and should return `false`. If there is no opening bracket on the stack at the time, you have also found an error in the given input (the closing bracket is missing an opening bracket) and need to return `false`.
   - When you have processed the input, check if there are opening brackets left on the stack. If this is the case, you have found an error in the given input (the remaining brackets are missing a closing bracket), otherwise the input is correct. Return `true` or `false` as appropriate.
5. Test your implementation with some correct and incorrect examples.
6. Assume you know that your text only contains round brackets. Can you think of an easier solution to implement the required functionality that does not require a stack? (Note: You do not have to implement this solution.)
7. When finished, have the work signed off and upload to Blackboard. You should particularly discuss your ideas for 6.

Task 3 (Advanced)

Note This task does not carry any marks and will not be assessed in any part of the module. It is an invitation for students who want to challenge themselves further by tackling more advanced topics.

Expand the program so that it does not only return if there is an error but also where the error occurs (if there is one). Assuming that the given text is a single line, it would for example be sufficient to additionally return the position in the given string. Alternatively, you could expand your program to read a given file line by line and return both, line number and position within the line.

Can you expand your program so that it also returns what kind of error you have encountered, e.g., an opening (closing) bracket that is missing a closing (opening) bracket or has a closing (opening) bracket of the wrong type?