# **Lab 4 Using a Stack to check for balanced parentheses**

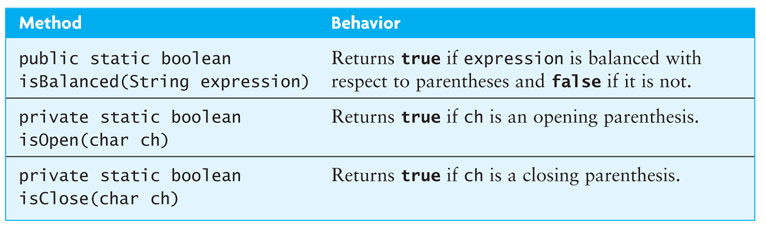
# **Week beginning 5th October 2015**

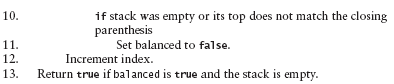
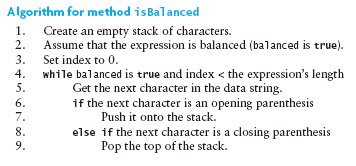
1. When analyzing arithmetic expressions, it is important to determine whether an expression is balanced with respect to parentheses

( a + b \* ( c / ( d – e ) ) ) + ( d / e )

The problem is further complicated if braces or brackets are used in conjunction with parentheses

The solution is to use stacks!

Likewise, Java compilers use a stack in parsing the syntax of programs to check that (, {, [ and < match up. 



**Change to algorithm above:**

**Replace lines 9 to 11 with the following:**

**if stack is empty**

**set balanced to false**

**else**

**remove item from top of stack**

**if removed item does not match closing parenthesis**

**set balanced to false**

Write the code for isOpen, isClose and isBalanced methods. The algorithm for isBalanced method is given above.

If you are checking for the different types, include another method

private static boolean areMatching(char c1, char c2)

This method returns true if c1 and c2 are matching open and close parentheses e.g. ‘(‘ and ‘)’ or ‘<’ and ‘>’ etc.

To test your code (use JUnit):

Provide a variety of input expressions displaying the result true or false

Try several levels of nested parentheses

Try nested parentheses where corresponding parentheses are not of the same type – i.e. use (, { , [ and < as Java allows.

e.g. try (2+[5+7]+1)

Try unbalanced parentheses

No parentheses at all!

Stack implementation in Java:

Note (from the slides):

* java.util provides a class *Stack*
* But the java documentation states:
  + A more complete and consistent set of LIFO stack operations is provided by the [Deque](file:///C:\Program%20Files\Java\jdk1.6.0_21\docs\api\java\util\Deque.html) interface and its implementations, which should be used in preference to this class.
* Use LinkedList as the implementation of Deque interface

2. Extra work – in your own time. See Backtracking.pptx