COMPX201/Y05335

Stack & Testing

Due**:** Friday 6th of June 11:59pm

Part One (50%):

In this assignment you will create a linked list that processes commands in LIFO order (i.e. a Stack). You will need to define Java classes to implement the stack. Your stack should use a linked list (i.e. not an array) following the specification given below.

1. **Stack:** define a class called `Stack` in a file called `Stack.java`. This class is to implement a dynamic linked list as a stack that supports the following public methods where applicable.
    - `push(String x)` - add item `x` to the **start** of the list.
    - `pop()` - remove the item from the **start** of the list and return its value.
    - `peek()` - look at the item at the **start** of the list and return its value.
    - `isEmpty()` - returns boolean true if head node is null; false otherwise.
    - `length()`: returns as an int the number of items in the stack.
    - `dump()`: prints the contents of the stack to standard output.
2. **The Node:** define a class called `Node` for the nodes in your `Stack`. It can either be an external class in a separate file called `Node.java` or an inner class of `Stack`. It should have the following:
    - A member variable to hold the string value.
    - A member variable to hold a link to another `Node`.
    - A constructor that takes a value as a string argument and copies that value into the Node's member variable.

You may have additional member variables and methods if they are useful to you, but they should be private.

Part Two (50%):

Using JUnit as described in class, create a test file that will test the operations of your Stack. For each test consider the testing strategies discussed in class and ensure your test suite has adequate coverage of the functionality. Consider "edge" cases, like testing an empty stack or a stack with only one item.

Add appropriate documentation to each of your tests to detail your reasons for adding the test to your test suite. It is useful for readability to group tests for the same function together in the document.

Note that if you have implemented Part One correctly your tests for Part Two should pass. However, test quality is also important here, adding duplicate tests which cover the same functionality or code paths are meaningless and you will only receive marks for one of the duplicate tests. It may be useful to create a test plan to ensure that you cover the different parts of the functionality.

Assessment:

Your solution will be marked on the basis of how well it satisfies the specification, the level of coverage that your tests provide, the quality of your tests, how well-formatted and easy to read your code is, the use of JavaDocs, and whether each class and public method has at least some comment explaining what it does, what it's for, and what any of its arguments are (i.e. documentation) .

Your code should compile and run as a console program from the command-line (i.e. no GUI). Students are encouraged to test their code prior to submitting their solutions.

Submission:

Create an empty directory (i.e. folder) using your student ID number as the directory name. Place copies of your source code in this directory. If you wish to communicate with the marker any additional information then you may include a plain text README file, but nothing else (e.g. no compiled code). Upload this directory through the Canvas submission page for this assignment.