Question 2 - Binary Search Tree [20 marks]

Your Q2 directory contains code that implements a binary search tree with a set of integer numbers. The implementation has had the code for “*find*”, “*delete*”, and “*sumEvenNodes*” removed.

You are required to complete the implementation replacing the missing code. Your answer must be placed in your Q2 directory.

Tasks:

1) [7 marks] Implement the “*find*” method. The method should return “`true`” if a tree contains a key, otherwise return “`false`”.
2) [7 marks] Implement the “*delete*” method. Use successor to replace the target node if the target node has two children.
3) [6 marks] Implement the “*sumEvenNodes*” method to print the sum of the nodes that have an even number of direct children (zero is an even number).

Check the provided comments and test classes (`Task1Test.java, Task2Test.java, Test3Test.java`) for more details on implementation details, i.e. return type, input arguments, and return type.

You may create more methods if you need. Make sure that you do not move the files in Q2 directory into another directory.

Handy tips

Three possible cases in deletion and required actions:

1. If the target node has no children
    a. Delete the target node
2. If the target node has one child
    a. Replace the target node with the child node
3. If the target has two children (subtrees)
    a. Replace the target node with its successor
    b. Delete successor in subtree

Question 3 - Testing [20 marks]

Your Q3 directory contains code that implements two useful utilities. `MyUtil.java` file contains *`parseDouble`* method to extract the first number in an input string. `MyStringUtil.java` file contains *`isMixedCase`* to check whether an input string contains both uppercase and lowercase characters.

Tasks:

1) [10 marks] Your first task for this question is to implement a **minimum number** of JUnit test cases for *`parseDouble`* that is **code complete**. Write your test case(s) in *`test()`* method in `MyUtilTest.java`. Use *`assertEquals`* to check the correctness of the implementation. All test cases should pass the JUnit test to get the full marks.

2) [10 marks] Your second task for this question is to implement a **minimum number** of JUnit test cases for *`isMixedCase`* that is **branch complete**. Write your test case(s) in *`test()`* method in `MyStringUtilTest.java`. Use *`assertEquals`* to check the correctness of the implementation. All test cases should pass the JUnit test to get the full marks.

Handy tips
Code complete: with a code complete test, all statements need to be executed at least once during the test. **Branch complete**: with a branch complete test, all possible branch condition statements need to be executed during the test. Branch complete is different from path complete, which needs to take into account all possible execution paths of a program.

Question 4 - Tokenizer, Parser [30 marks]

The theme of this question is developing a simple parser for LOGO programming language. LOGO controls the commands for movement and drawing of a pointer on the screen.

Assume that we have a grid with 11 x 21 cells, where 11 is the number of rows and 21 is the number of columns. A pointer is represented by one of the following characters:

- “^”: The pointer is facing the *NORTH* direction
- “>”: The pointer is facing the *EAST* direction
- “<”: The pointer is facing the *WEST* direction
- “v”: The pointer is facing the *SOUTH* direction

We can control the movement and drawing of the pointer by the following commands:

- LEFT: Turn the direction of the pointer by 90 degrees to the left
- RIGHT: Turn the direction of the pointer by 90 degrees to the right
- PENUP: Set the status of the pointer to be leaving no trail, when it moves
- PENDOWN: Set the status of the pointer to be leaving a trail, when it moves
- FORWARD(*n*): Move the pointer along the direction it is pointing by *n* cells
- BACK(*n*): Move the pointer in the reverse direction it is pointing by *n* cells
- FORWARD_TO_END: Move the pointer along the direction until it reaches the boundary of the grid
- BACK_TO_END: Move the pointer in the reverse direction until it reaches the boundary of the grid

The grammar of simplified LOGO language is given by:

```
<Command> := LEFT | RIGHT | PENUP | PENDOWN | FORWARD(<num>) | BACK(<num>)
| FORWARD_TO_END | BACK_TO_END
<Exp> := <Command>;  <Exp>  |  <Command>;
```

where <num> is an integer literal.

(Check the example 1 on the next page)

Example 1:

- Initial screen: (in the example, the pointer is initially positioned with *NORTH* direction at the center of the grid with PENUP status)

```
#####################
#####################
#####################
#####################
#####################
##########^##########
#####################
#####################
#####################
#####################
#####################
```

- Input:

  PENUP; **LEFT**; **FORWARD**(10); **PENDOWN**; **RIGHT**; **BACK**(3);

- Output:

```
#####################
#####################
#####################
#####################
#####################
.####################
.####################
.####################
^####################
#####################
#####################
```

(Check the example 2 on the next page)

Example 2:

- Initial screen: (in the example, the pointer is initially positioned with *SOUTH* direction at the center of the grid with PENUP status)

```
#####################
#####################
#####################
#####################
#####################
##########v##########
#####################
#####################
#####################
#####################
#####################
```

- Input:

  PENDOWN; BACK_TO_END;

- Output:

```
##########v##########
##########.##########
##########.##########
##########.##########
##########.##########
##########.##########
#####################
#####################
#####################
#####################
#####################
```

where an empty cell is represented by character "#", a cell with the trail of the pointer is represented by character "`.`", and the pointer is initially with PENUP status. Note that you do not need to consider cases where FORWARD(n) and BACK(n) can go beyond the boundary of the grid.

(Check your tasks on the next page)

Tasks:

1) [10 Marks] Complete "*next*()" methods in "`Tokenizer.java`" to extract an input expression into tokens.
2) [10 Marks] Complete "*parse*()" method in "`Parser.java`" to parse an input expression by computing the final position of the pointer and marking the pointer movement on the screen.
3) [10 Marks] Complete "*trace*()" method in "`Screen.java`" to return a string showing the trail of the pointer, its current position and direction.

Please check the expected results of these methods from the JUnit test files: `TokenizerTest.java, ParserTest.java, ScreenTest.java`.

For 1) - 3), you should modify the required methods within Q4 directory. You can make any additional method if you need while completing the tasks. Make sure that you do not move the files in Q4 directory into another directory.