UMGC CMSC 315
Project 3 Indications

I suggest an incremental approach of program development by first defining a minimal functional BinaryTree class consisting of class constructor and the inorder tree traversal method that generates the fully parenthesized inorder traversal of the tree.

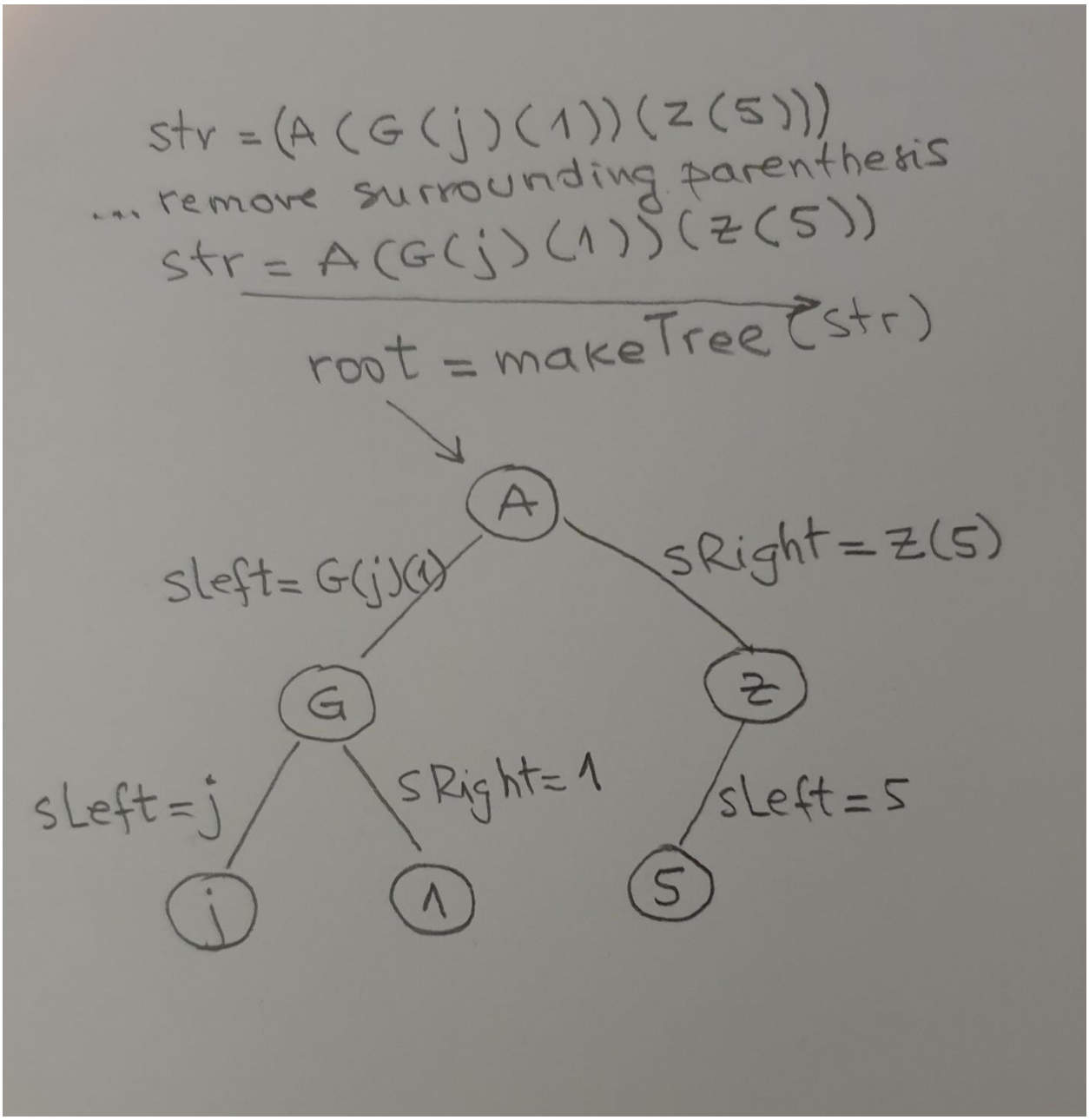
The rest of the required methods should be defined and tested, one by one, in an incremental approach.

Below is the skeleton of the BinaryTree class that illustrates the tree construction process. Note the recursive implementation of makeTree method which is invoked from the BinaryTree constructor.

The comments should be considered as actions that should be implemented in Java.

```
public class BinaryTree {

  private Node root = null;  // tree root

  static class Node {
     // define node data of character type
     // define left and right references
     // define Node constructor
  }

  // BinaryTree constructor takes the String of prefix tree
  // representation as parameter and throws InvalidTreeSyntax exception
  public BinaryTree(String prefixFormat) throws InvalidTreeSyntax {

    // check conditions for throwing the InvalidTreeSyntax exception
    // remove the first and last parenthesis
    root = makeTree(prefixString);
  }

  // makeTree method recursively builds the binary tree
  private Node makeTree(String str) {

     if(str == null) return null;
     Node n = new Node(str.charAt(0), null, null);

     // get left substring (without the surrounding parenthesis) in sLeft

     // get right substring (without the surrounding parenthesis) in sRight

     if(sLeft != null && sRight == null) n.left = makeTree(sLeft);
     else {n.left = makeTree(sLeft); n.right = makeTree(sRight);}

     return n;
  }

  // the rest of the required methods ...
}
```

Below is an execution trace of how makeTree method works for the prefix representation of the tree that is given in the instructions: (A(G(j)(1))(z(5))).



Important notes

1. After building the tree, I suggest traversing it (in pre-order or post-order because no parenthesis are necessary) and display the traversal result to make sure the tree is correctly built.

2. To better understand how the tree is built, I suggest manually tracing, using paper and pencil, the step by step execution of the method makeTree with the prefix expression (A(G(j)(1))(z(5))) as parameter.