CS 361L **Project 2**

Assigned: 04/17/2023. Due: 05/05/2023, 11 pm (on Canvas).

Instructor: Tamisra H. Sanyal

In this project, we are going to study the use of **singleton pattern** which is an important design pattern in software engineering. This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

We are going to implement inductively defined binary trees. Binary trees of number nodes can be defined inductively using the following grammar:

R1:

R2:

such that the first rule R1 indicates that a binary tree can be an empty tree Nil, while the second rule R2 says that a binary tree can be defined as a node consisting of a number (the node value) and two subtrees.

We are going to implement the above grammar in Java using **inheritance** and **polymorphism**. Firstly, we define the following abstract class for binary trees:

abstract class BT

{

// abstract method for getting the tree height

public abstract int height();

}

This class is the implementation for the grammar symbol BT which is the general binary tree class. Since every binary tree can only be either empty (i.e., Nil) or node, this class will never have any instances.

Then we implement the Java class for Nil, and it should be a class which has a singleton instance, since all empty trees are same.

class Nil extends BT

{

private static Nil nil; // reference to the singleton object

// dummy constructor

private Nil()

{

}

// obtaining reference to the singleton object

public static Nil getNil()

{

if(nil == null)

nil = new Nil();

return nil;

}

// overriding method for computing the height of Nil

public int height()

{

return -1;

}

public String toString()

{

return "Nil";

}

}

Then we define the class for nodes as follows:

class Node extends BT

{

int num; // value of the node

BT left; // reference to the left subtree

BT right; // reference to the right subtree

// constructor for combining two existing binary trees

Node(int n, BT l, BT r)

{

num = n;

left = l;

right = r;

}

// computing the height of the tree

public int height()

{

int lh = left.height();

int rh = right.height();

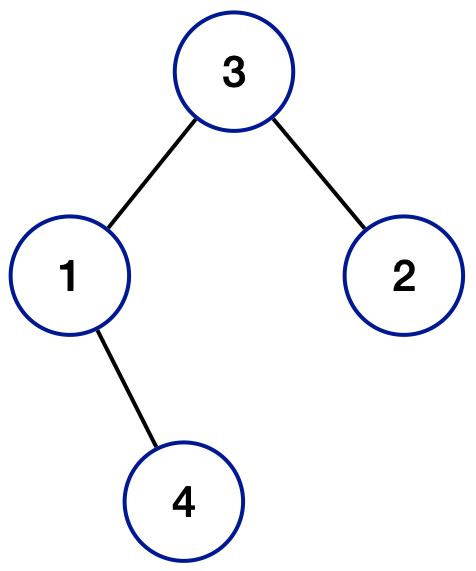
return (lh < rh ? rh : lh) + 1;

}

}

By using the singleton pattern for Nil, the method height only requires 3 statements. The following binary tree can be created using the Java expression

BT tree = new Node(3, new Node(1, nil, new Node(4, nil, nil)), new Node(2, nil, nil));



**Programming Tasks:**

1. (5 points) Override the method toString in the Node class, such that the structure of the binary tree can be translated to the following string format:

Node(num, string\_of\_left\_subtree, string\_of\_right\_subtree).

For example, the tree showed above is translated to the string

Node(3, Node(1, Nil, Node(4, Nil, Nil)), Node(2, Nil, Nil))

1. (5 points) Add an abstract method isBST to the BT class and override it in the subclasses, it verifies if the binary tree is a binary search tree. **Restrictions:** You are not allowed to use loops or checking if a reference is null or not.
2. (10 points) Write a static method in the BT class. It takes an array of integer values and creates a binary search tree by consecutively inserting these values to an empty tree (Nil). The method is required to return a reference of a Node object. You need to implement a helper function for inserting a new node to a binary search tree.
3. (10 points) Write a method to delete the node which has the given key. We assume that the binary tree is a binary search tree.

In the project, we assume that all nodes have distinct keys. Please write a project report which includes an explanation of your implementations and their running times. You do not need to give pseudocode.