FLIP ROBO

FULL STACK DEVELOPMENT
WORKSHEET – 6

Ques 1. Write a java program that inserts a node into its proper sorted position in a sorted linked list.

Ans.
```
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class SortedLinkedList {
    Node head;

    public SortedLinkedList() {
        this.head = null;
    }

    // Method to insert a node into its proper sorted position
    public void insert(int newData) {
        Node newNode = new Node(newData);

        if (head == null || head.data >= newData) {
            newNode.next = head;
```

```
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null && current.next.data < newData) {
                current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
    }

    // Method to display the linked list
    public void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        SortedLinkedList sortedList = new SortedLinkedList();
```

```
        // Inserting elements into the sorted linked list
        sortedList.insert(5);
        sortedList.insert(10);
        sortedList.insert(2);
        sortedList.insert(8);

        // Displaying the sorted linked list
        System.out.println("Sorted Linked List:");
        sortedList.display();
    }
}
```

The Insert method of the SortedLinkedList class in this programme accepts a new data value as an argument and adds a new node holding that data into the linked list at the correct sorted position. The sorted linked list is printed using the display technique. An illustration of adding entries to the sorted linked list is provided in the main procedure.

Ques 2. Write a java program to compute the height of the binary tree.

Ans.

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
```

```
    }
}

class BinaryTree {
    Node root;

    public BinaryTree() {
        root = null;
    }

    // Method to compute the height of a binary tree
    public int height(Node root) {
        if (root == null) {
            return 0;
        } else {
            // Calculate the height of the left and right subtrees
            int leftHeight = height(root.left);
            int rightHeight = height(root.right);

            // Return the maximum of left and right subtree heights, plus 1 for the current level
            return Math.max(leftHeight, rightHeight) + 1;
        }
    }
}
public class Main {
```

```
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Constructing a sample binary tree
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        // Calculating the height of the binary tree
        int treeHeight = tree.height(tree.root);

        System.out.println("Height of the Binary Tree is: " + treeHeight);
    }
}
```

The height of the binary tree rooted at a given node is determined by the program's BinaryTree class's height function, which accepts a node as an input. The primary method builds an example binary tree, computes its height, and outputs the result. The longest path from a root node to a leaf node is the height of a binary tree.

Ques 3. Write a java program to determine whether a given binary tree is a BST or not.

Ans.

```
class Node {
    int data;
    Node left, right;
```

```
    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    // Helper method to check if a binary tree is a BST
    private boolean isBSTUtil(Node node, int min, int max) {
        // An empty tree is a BST
        if (node == null) {
            return true;
        }

        // Check if the current node's data is within the valid range
        if (node.data < min || node.data > max) {
            return false;
        }

        // Recursively check the left and right subtrees
        return isBSTUtil(node.left, min, node.data - 1) && isBSTUtil(node.right, node.data + 1, max);
```

```
    }

    // Method to check if the binary tree is a BST
    public boolean isBST() {
        return isBSTUtil(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }
}
public class Main {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Constructing a sample binary tree
        tree.root = new Node(2);
        tree.root.left = new Node(1);
        tree.root.right = new Node(3);

        // Checking if the binary tree is a BST
        boolean isBST = tree.isBST();
            if (isBST) {
            System.out.println("The binary tree is a BST.");
        } else {
            System.out.println("The binary tree is not a BST.");
        }
    }
}
```

The BinaryTree class in this programme has a method called isBST that uses the helper function isBSTUtil to determine if the binary tree is a BST. Three parameters are required by the helper method: the current node, the lowest value that can be assigned to it, and the highest value that may be assigned to it. It repeatedly determines if the data of the current node is into the acceptable range and whether the BST property is also satisfied by the left and right subtrees. The main method builds an example binary tree, determines if it is a BST, and outputs the result.

Ques 4. Write a java code to Check the given below expression is balanced or not . (using stack)

{ { [[(( ))]) } }

Ans. import java.util.Stack;

```
public class BalancedExpressionChecker {


    // Method to check if the given expression is balanced
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();


        // Iterate through each character in the expression
        for (char ch : expression.toCharArray()) {
            // If the character is an opening bracket, push it onto the stack
            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            }
            // If the character is a closing bracket, pop from the stack and check if it matches the corresponding opening bracket
            else if (ch == ')' || ch == ']' || ch == '}') {
                if (stack.isEmpty()) {
```

```
                return false; // Unmatched closing bracket
            }
            char top = stack.pop();
            if ((ch == ')' && top != '(') || (ch == ']' && top != '[') || (ch == '}' && top != '{')) {
                return false; // Mismatched brackets
            }
        }
    }

    // If the stack is empty, the expression is balanced
    return stack.isEmpty();
  }

  public static void main(String[] args) {
    String expression = "{ { [ [ ( ( ) ) ] ) } }";

    if (isBalanced(expression)) {
        System.out.println("The expression is balanced.");
    } else {
        System.out.println("The expression is not balanced.");
    }
  }
}
```

The isBalanced function in this programme employs a stack to record the opening brackets that are encountered. It determines if there is an opening bracket at the top of the stack that corresponds to each closing bracket. It proceeds if the brackets match; if not, it returns false. In order to confirm that every bracket was matched, the last check makes sure the stack is empty. The primary method checks if the expression is balanced and outputs the result.

Ques 5. Write a java program to Print left view of a binary tree using queue.

Ans. import java.util.LinkedList;

import java.util.Queue;

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    // Method to print the left view of the binary tree
    public void leftView() {
        if (root == null) {
```

```
    return;
}

Queue<Node> queue = new LinkedList<>();
queue.add(root);

while (!queue.isEmpty()) {
    int size = queue.size();

    // Traverse all nodes at the current level and print the leftmost node
    for (int i = 0; i < size; i++) {
        Node current = queue.poll();

        if (i == 0) {
            System.out.print(current.data + " ");
        }

        if (current.left != null) {
            queue.add(current.left);
        }
        if (current.right != null) {
            queue.add(current.right);
        }
    }
}
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Constructing a sample binary tree
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.right = new Node(6);
        tree.root.left.left.left = new Node(7);

        System.out.println("Left View of Binary Tree:");
        tree.leftView();
    }
}
```

The left view of the binary tree is printed using a level order traversal in this programme by the leftView function of the BinaryTree class. After building an example binary tree, the main function uses the leftView method to output the tree's left view. The set of nodes that are visible while looking at a binary tree from the left side is known as the left view.