1. Binary Search Tree Implementation

Java Code:

```java
import java.util.*;

class node {
    int data;
    node left, right;

    node(int value) {
        data = value;
        left = right = null;
    }
}

public class BST {
    node r;

    public node insert(node r, int val) {
        if (r == null) return new node(val);
        if (val < r.data) r.left = insert(r.left, val);
        else if (val > r.data) r.right = insert(r.right, val);
        return r;
    }

    public void inorder(node r) {
        if (r != null) {
            inorder(r.left);
            System.out.print(r.data + " ");
            inorder(r.right);
        }
    }

    public node ios(node r) {
        while (r.left != null) r = r.left;
        return r;
    }

    public void bfs(node r){
        Queue<node> q = new LinkedList<>();
        q.add(r);
        while(!q.isEmpty()){
            node root = q.poll();
            System.out.print(root.data+" ");
            if (root.left != null) q.add(root.left);
            if (root.right != null) q.add(root.right);
        }
        System.out.println();
    }

    public node delete(node r, int val) {
        if (r == null) return null;
```

```
        if (val < r.data) r.left = delete(r.left, val);
        else if (val > r.data) r.right = delete(r.right, val);
        else {
            if (r.left == null && r.right == null) return null;
            else if (r.left == null) return r.right;
            else if (r.right == null) return r.left;

            node tc = ios(r.right);
            r.data = tc.data;
            r.right = delete(r.right, tc.data);
        }
        return r;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST t = new BST();
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int val = sc.nextInt();
            t.r = t.insert(t.r, val);
        }
        t.bfs(t.r);
    }
}
```

2. Binary Tree Implementation

Java Code:

```
import java.util.*;

public class bt {
    static Scanner sc = new Scanner(System.in);

    static class Node {
        int data;
        Node left, right;
        Node(int value) {
            data = value;
            left = right = null;
        }
    }

    public static Node buildTree() {
        int val = sc.nextInt();
        if (val == -1)
            return null;
        Node node = new Node(val);
        node.left = buildTree();
        node.right = buildTree();
        return node;
    }

```

```
    public static int depth(Node root) {
        if (root == null)
            return 0;
        int ld = depth(root.left);
        int rd = depth(root.right);
        return Math.max(ld,rd) + 1;
    }

    public static void inorder(Node root, int[] minMax) {
        if (root == null)
            return;
        inorder(root.left, minMax);
        System.out.print(root.data + "␣");
        minMax[0] = Math.min(minMax[0], root.data);
        minMax[1] = Math.max(minMax[1], root.data);
        inorder(root.right, minMax);
    }

    public static void preorder(Node root) {
        if (root == null)
            return;
        System.out.print(root.data + "␣");
        preorder(root.left);
        preorder(root.right);
    }

    public static void postorder(Node root) {
        if (root == null)
            return;
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.data + "␣");
    }

    public static void main(String[] args) {
        Node root = buildTree();
        int[] minMax = {Integer.MAX_VALUE, Integer.MIN_VALUE};
        inorder(root, minMax);
        System.out.println();
        preorder(root);
        System.out.println();
        postorder(root);
        System.out.println();
        System.out.println(depth(root));
    }
}
```

3. Sorted Priority Queue Implementation

Java Code:

```
import java.util.*;

public class SortedPriorityQueue {
```

```java
    static int[] arr = new int[100];
    static int size = 0;

    static void push(int val) {
        int low = 0, high = size - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] < val) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        for (int i = size; i > low; i--) {
            arr[i] = arr[i - 1];
        }
        arr[low] = val;
        size++;
    }

    static int pop() {
        if (size == 0) return -1;
        return arr[--size];
    }

    static int top() {
        if (size == 0) return -1;
        return arr[size - 1];
    }

    public static void main(String[] args) {
        push(10);
        push(5);
        push(20);
        push(15);
        System.out.println("Max element: " + top());
        System.out.println("Removed: " + pop());
        System.out.println("New max: " + top());
    }
}
```

Student Name: Pulkit
UID: 23BCS11733
Branch: CSE
Section/Group: KRG-3B
Semester: 5th
Date of Performance: 31 JULY
Subject Name: Design and Analysis of Algorithm (23CSH-301)
Experiment No: 3

AIM: Stack and Queue Implementation in Java

1. Stack Implementation using Array

Java Code:

```java
class StackArray {
    int[] arr = new int[10];
    int top;

    StackArray() {
        top = -1;
    }

    void push(int x) {
        if (top >= 9) {
            System.out.println("OVERFLOW!");
            return;
        }
        arr[++top] = x;
    }

    void pop() {
        if (top < 0) {
            System.out.println("UNDERFLOW!!");
            return;
        }
        top--;
    }

    int peek() {
        if (top < 0) {
            System.out.println("UNDERFLOW!!");
            return -1;
        }
        return arr[top];
    }

    public static void main(String[] args) {
        StackArray s = new StackArray();
        s.push(96);
        s.push(100);
        s.pop();
        System.out.println("Top␣Element␣:␣" + s.peek());
    }
}
```

2. Stack Implementation using Linked List

Java Code:

```
class Node {
    int data;
    Node next;
    Node(int x) {
        data = x;
        next = null;
    }
}

class StackLL {
    Node top;

    StackLL() {
        top = null;
    }

    void push(int x) {
        Node temp = new Node(x);
        temp.next = top;
        top = temp;
    }

    void pop() {
        if (top == null) {
            System.out.println("UNDERFLOW!!");
            return;
        }
        top = top.next;
    }

    int peek() {
        if (top == null) {
            System.out.println("UNDERFLOW!!");
            return -1;
        }
        return top.data;
    }

    public static void main(String[] args) {
        StackLL s = new StackLL();
        s.push(10);
        s.push(20);
        s.push(30);
        s.pop();
        s.pop();
        System.out.println("TOP␣ELEMENT␣:␣" + s.peek());
    }
}
```

3. Queue Implementation using Array

Java Code:

```
class QueueArray {
    int[] arr = new int[10];
    int front, rear;

    QueueArray() {
        front = rear = -1;
    }

    void enqueue(int x) {
        if (rear >= 9) {
            System.out.println("Overflow!!");
            return;
        }
        if (front == -1) front = 0;
        arr[++rear] = x;
    }

    void dequeue() {
        if (front == -1 || front > rear) {
            System.out.println("Queue␣Underflow");
            return;
        }
        front++;
    }

    int peek() {
        if (front == -1 || front > rear) {
            System.out.println("Queue␣is␣Empty");
            return -1;
        }
        return arr[front];
    }

    public static void main(String[] args) {
        QueueArray q = new QueueArray();
        q.enqueue(10);
        q.enqueue(20);
        q.dequeue();
        System.out.println("Front␣element:␣" + q.peek());
    }
}
```

4. Queue Implementation using Linked List

Java Code:

```
class NodeQ {
    int data;
    NodeQ next;
    NodeQ(int x) {
```

```
        data = x;
        next = null;
    }
}

class QueueLL {
    NodeQ front, rear;

    QueueLL() {
        front = rear = null;
    }

    void enqueue(int x) {
        NodeQ temp = new NodeQ(x);
        if (rear == null) {
            front = rear = temp;
            return;
        }
        rear.next = temp;
        rear = temp;
    }

    void dequeue() {
        if (front == null) {
            System.out.println("UNDERFLOW!!");
            return;
        }
        front = front.next;
        if (front == null) rear = null;
    }

    int peek() {
        if (front == null) {
            System.out.println("UNDERFLOW!!");
            return -1;
        }
        return front.data;
    }

    public static void main(String[] args) {
        QueueLL q = new QueueLL();
        q.enqueue(10);
        q.enqueue(20);
        q.dequeue();
        System.out.println("Front element : " + q.peek());
    }
}
```

5. Parentheses Checker using Stack

Java Code:

```
import java.util.*;
```

```
class StackChar {
    char[] arr = new char[10];
    int top;

    StackChar() {
        top = -1;
    }

    void push(char x) {
        if (top >= 9) {
            System.out.println("OVERFLOW!!");
            return;
        }
        arr[++top] = x;
    }

    void pop() {
        if (top == -1) {
            System.out.println("UNDERFLOW!!");
            return;
        }
        top--;
    }

    char peek() {
        if (top == -1) {
            System.out.println("UNDERFLOW!!");
            return '\0';
        }
        return arr[top];
    }

    boolean isEmpty() {
        return top == -1;
    }
}

public class ParenthesesChecker {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the expression: ");
        String e = sc.next();
        StackChar s = new StackChar();

        for (int i = 0; i < e.length(); i++) {
            char ch = e.charAt(i);
            if (ch == '(' || ch == '{' || ch == '[') {
                s.push(ch);
            } else if (ch == ')' || ch == '}' || ch == ']') {
                if (s.isEmpty()) {
                    System.out.println("Unbalanced Parentheses");
                    return;
                }
                char topChar = s.peek();
                if ((ch == ')' && topChar != '(') ||
                    (ch == '}' && topChar != '{') ||
```

```
                    (ch == ']' && topChar != '[')) {
                    System.out.println("Unbalanced Parentheses");
                    return;
                }
                s.pop();
            }
        }

        if (s.isEmpty()) {
            System.out.println("Balanced Parentheses");
        } else {
            System.out.println("Unbalanced Parentheses");
        }
    }
}
```