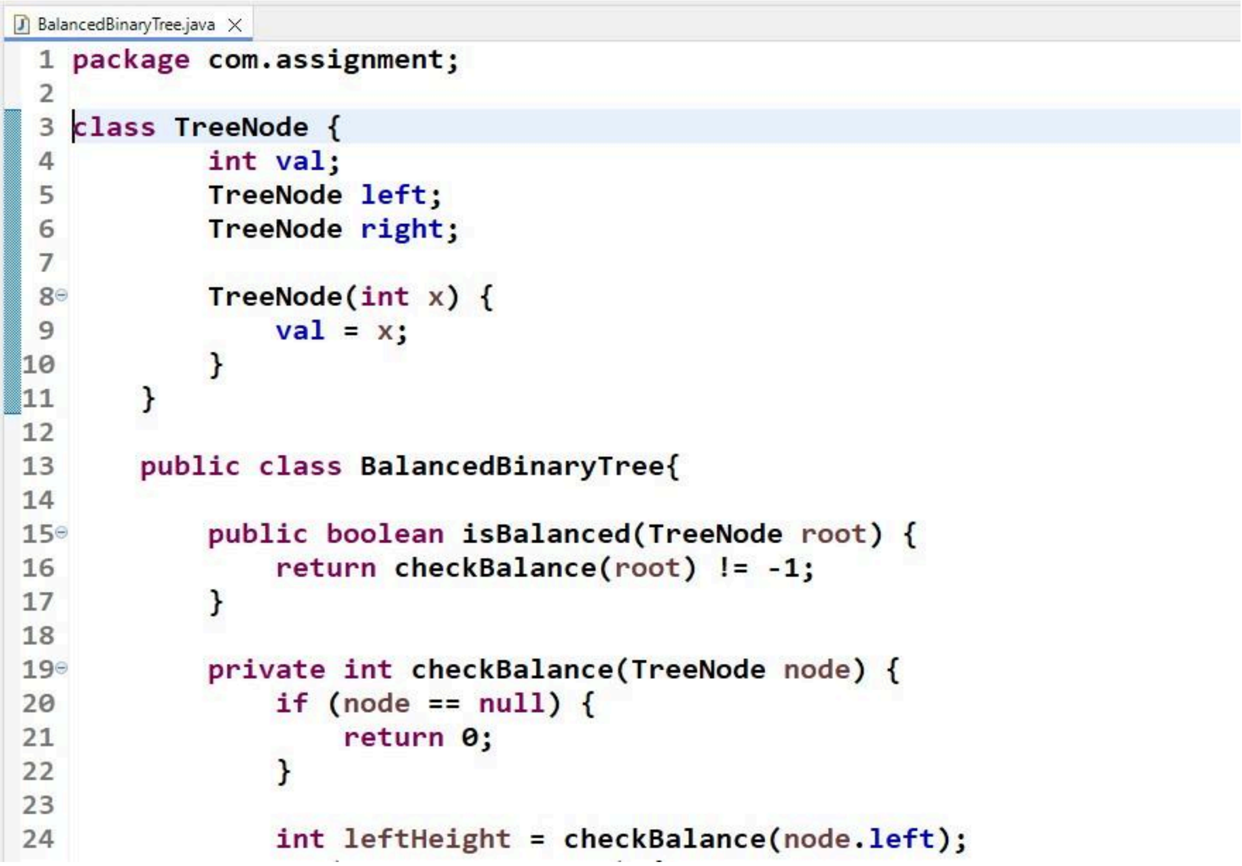
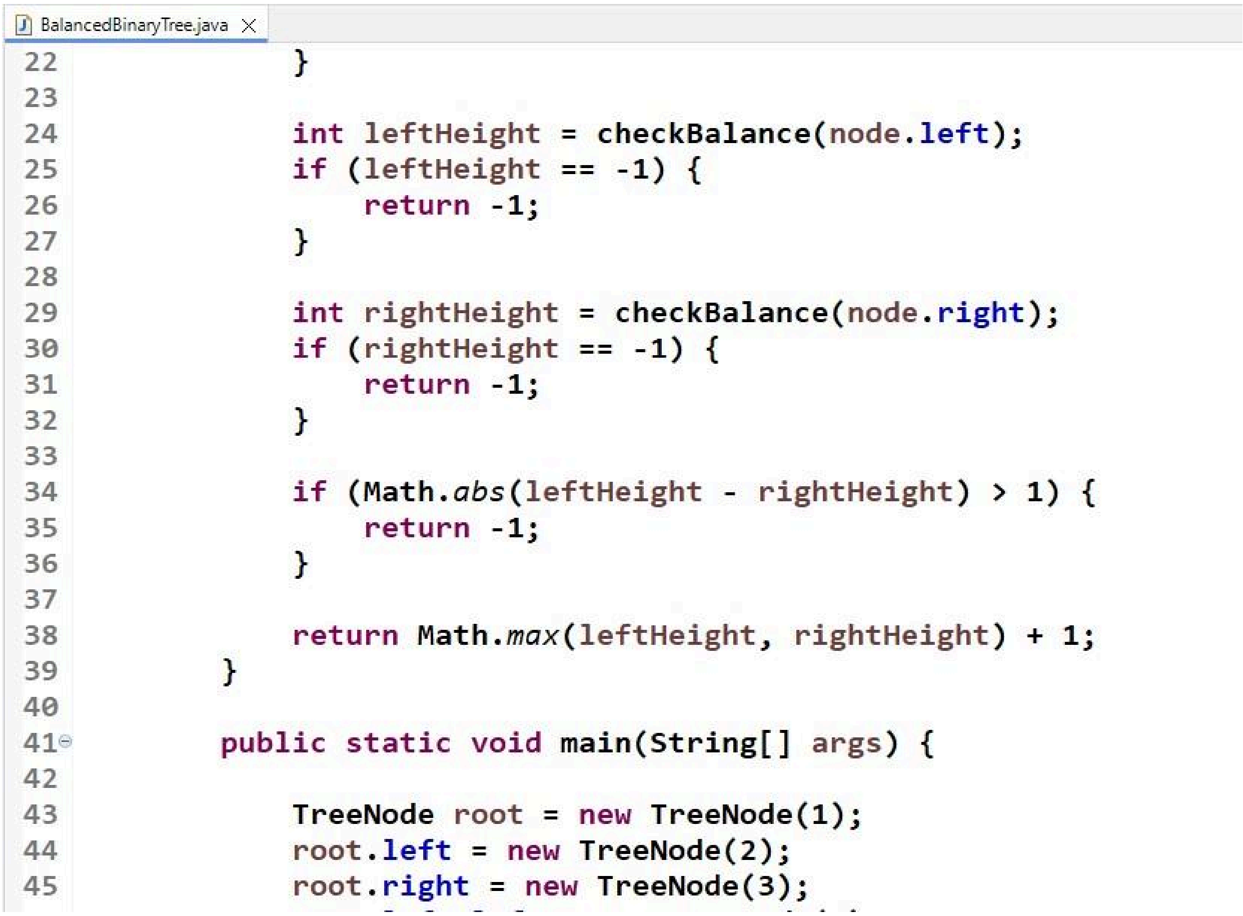
Day 7 and 8

1] Task 1: Balanced Binary Tree Check
Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

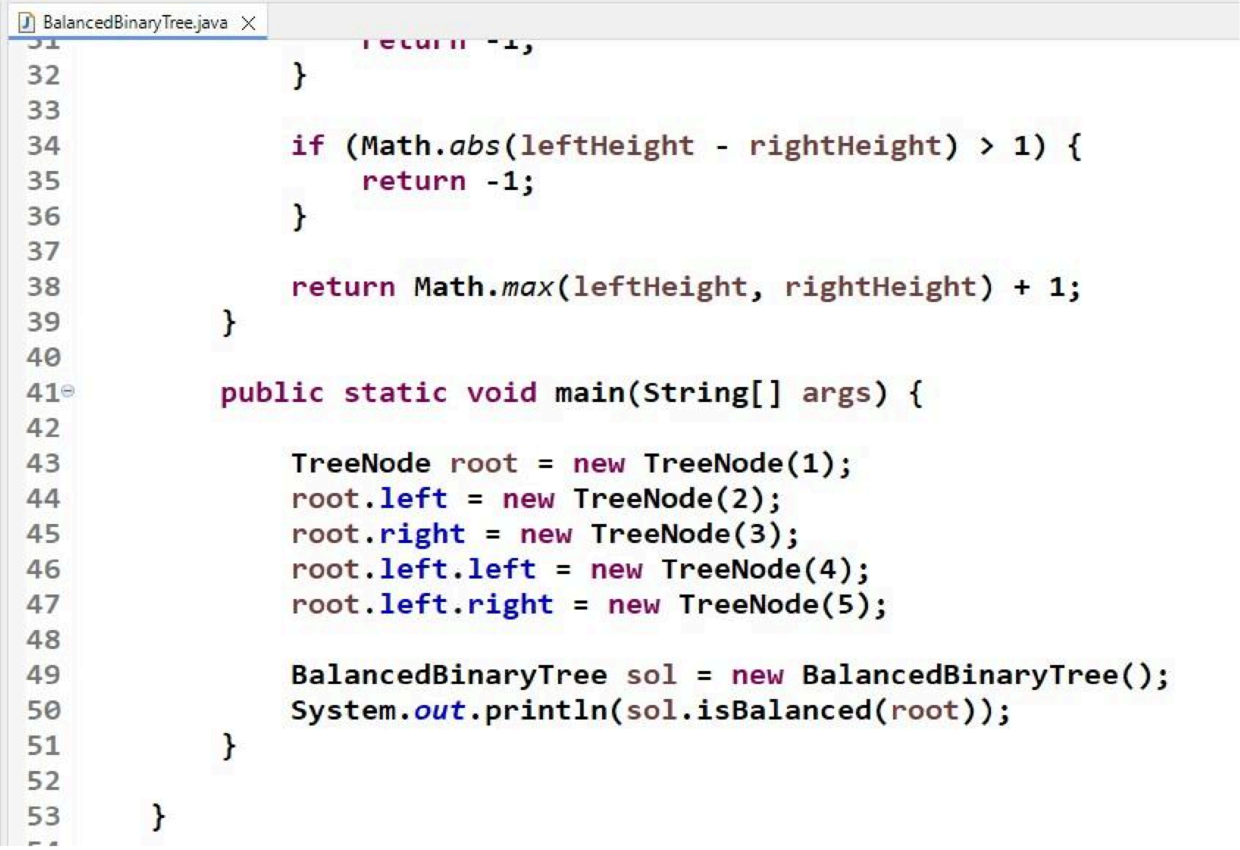
Solution:-
Code -



```java
package com.assignment;

class TreeNode {
        int val;
        TreeNode left;
        TreeNode right;

        TreeNode(int x) {
            val = x;
        }
    }

    public class BalancedBinaryTree{

        public boolean isBalanced(TreeNode root) {
            return checkBalance(root) != -1;
        }

        private int checkBalance(TreeNode node) {
            if (node == null) {
                return 0;
            }

            int leftHeight = checkBalance(node.left);
```

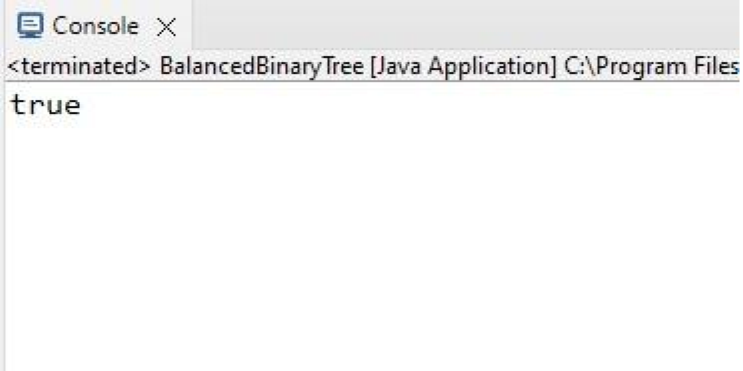


```java
        }

        int leftHeight = checkBalance(node.left);
        if (leftHeight == -1) {
            return -1;
        }

        int rightHeight = checkBalance(node.right);
        if (rightHeight == -1) {
            return -1;
        }

        if (Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
        }

        return Math.max(leftHeight, rightHeight) + 1;
    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
```



```
                return -1;
            }

            if (Math.abs(leftHeight - rightHeight) > 1) {
                return -1;
            }

            return Math.max(leftHeight, rightHeight) + 1;
        }

        public static void main(String[] args) {

            TreeNode root = new TreeNode(1);
            root.left = new TreeNode(2);
            root.right = new TreeNode(3);
            root.left.left = new TreeNode(4);
            root.left.right = new TreeNode(5);

            BalancedBinaryTree sol = new BalancedBinaryTree();
            System.out.println(sol.isBalanced(root));
        }

    }
```

Output -

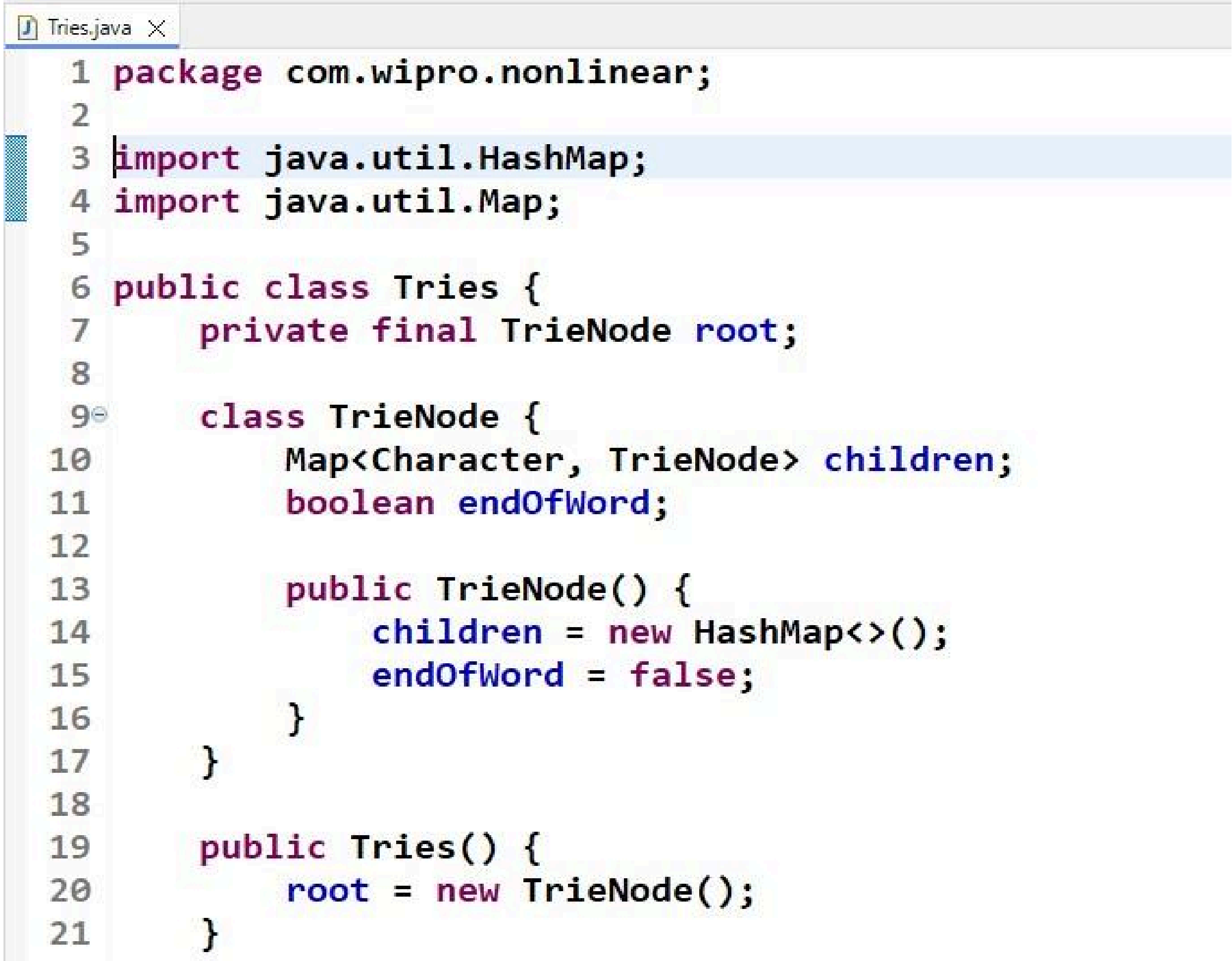


```
<terminated> BalancedBinaryTree [Java Application] C:\Program Files
true
```

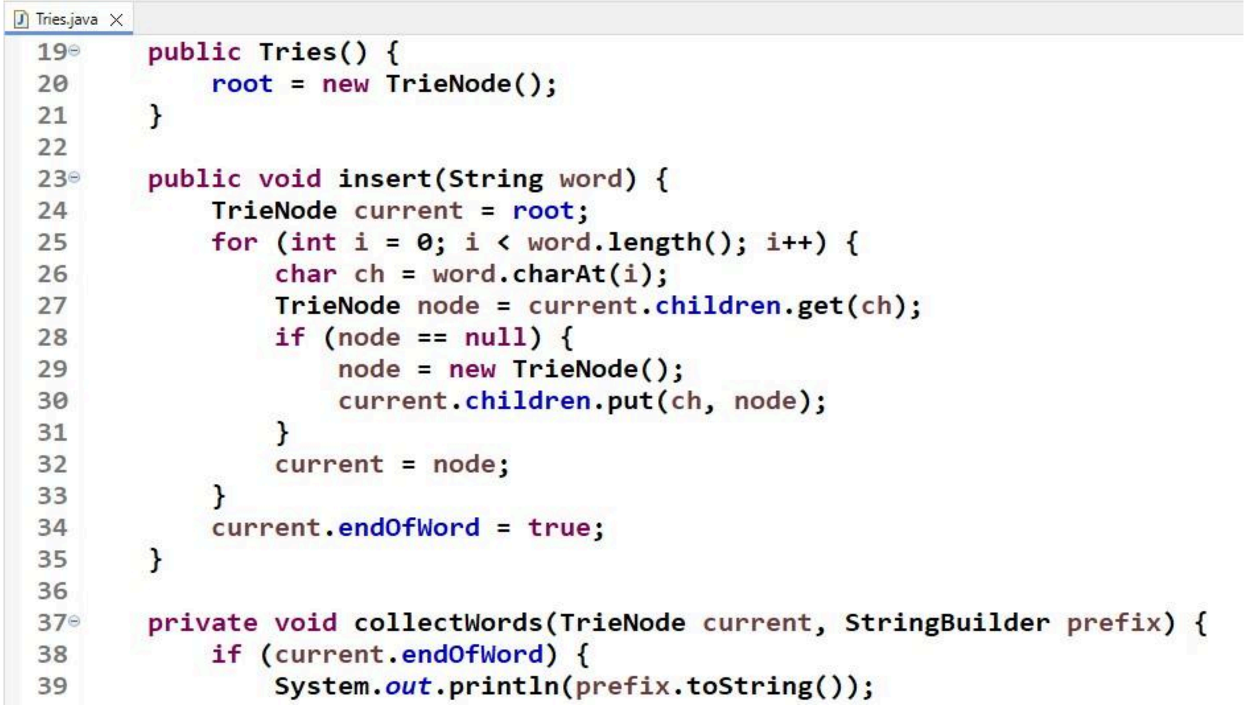
2] Task 2: Trie for Prefix Checking

Implement a trie data structure in java that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

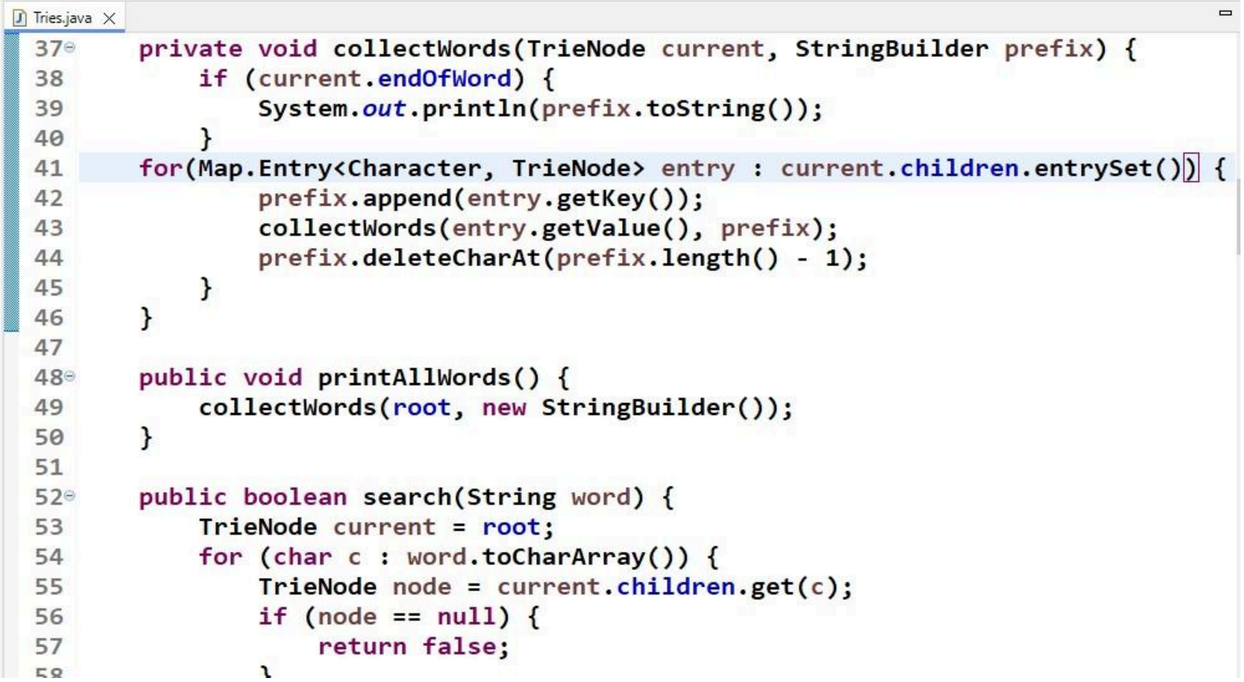
Solution:-
Code -



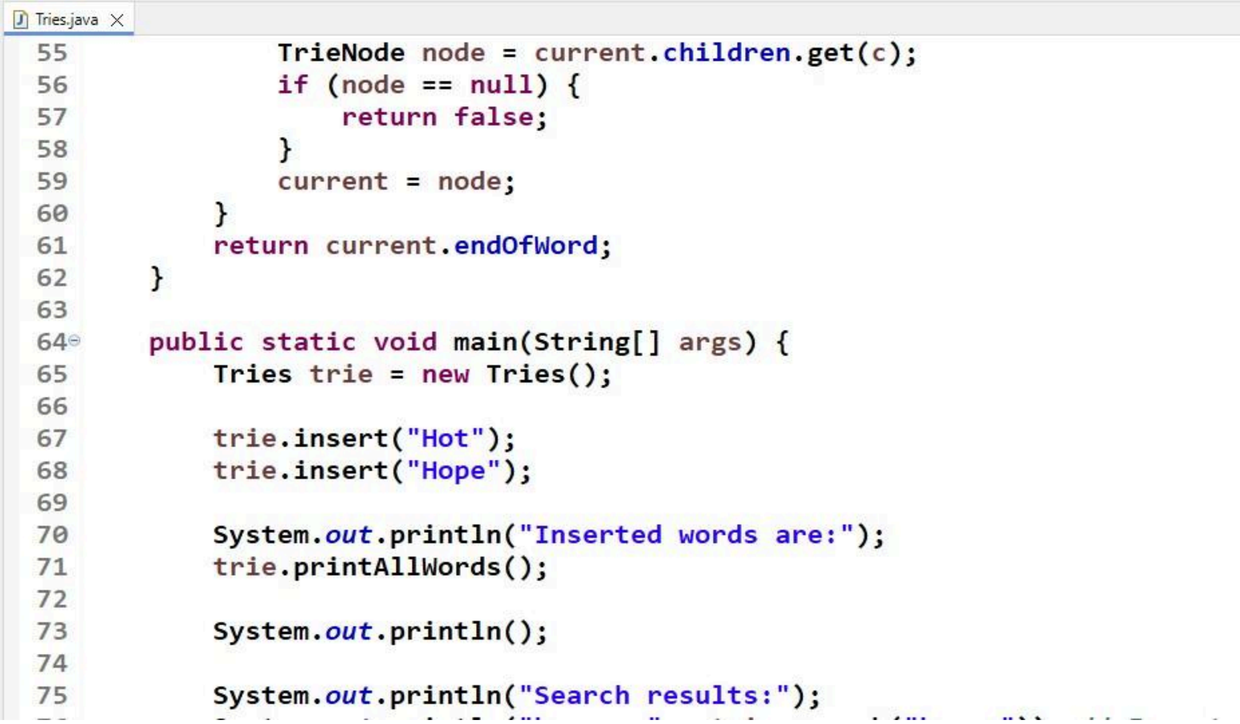
```java
package com.wipro.nonlinear;

import java.util.HashMap;
import java.util.Map;

public class Tries {
    private final TrieNode root;

    class TrieNode {
        Map<Character, TrieNode> children;
        boolean endOfWord;

        public TrieNode() {
            children = new HashMap<>();
            endOfWord = false;
        }
    }

    public Tries() {
        root = new TrieNode();
    }
```



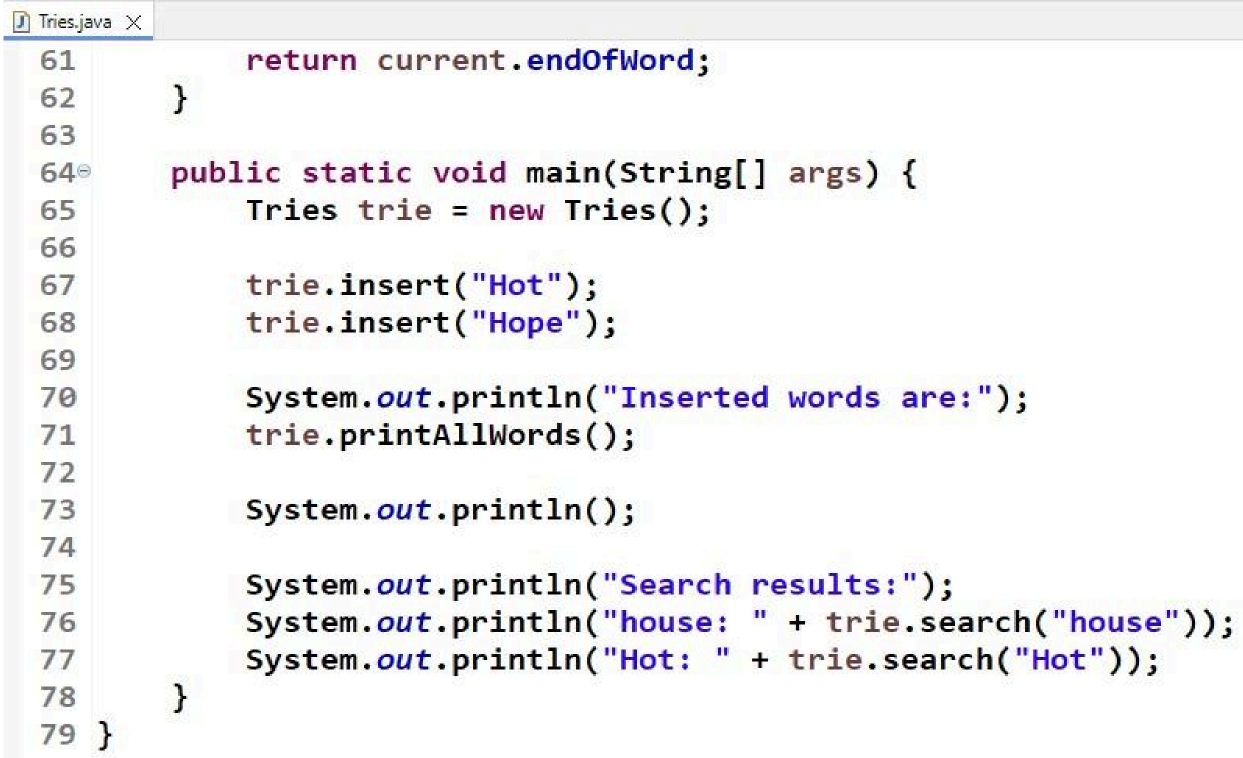
```java
    public Tries() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode current = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            TrieNode node = current.children.get(ch);
            if (node == null) {
                node = new TrieNode();
                current.children.put(ch, node);
            }
            current = node;
        }
        current.endOfWord = true;
    }

    private void collectWords(TrieNode current, StringBuilder prefix) {
        if (current.endOfWord) {
            System.out.println(prefix.toString());
```



```java
    private void collectWords(TrieNode current, StringBuilder prefix) {
        if (current.endOfWord) {
            System.out.println(prefix.toString());
        }
    for(Map.Entry<Character, TrieNode> entry : current.children.entrySet()) {
            prefix.append(entry.getKey());
            collectWords(entry.getValue(), prefix);
            prefix.deleteCharAt(prefix.length() - 1);
        }
    }

    public void printAllWords() {
        collectWords(root, new StringBuilder());
    }

    public boolean search(String word) {
        TrieNode current = root;
        for (char c : word.toCharArray()) {
            TrieNode node = current.children.get(c);
            if (node == null) {
                return false;
            }
```

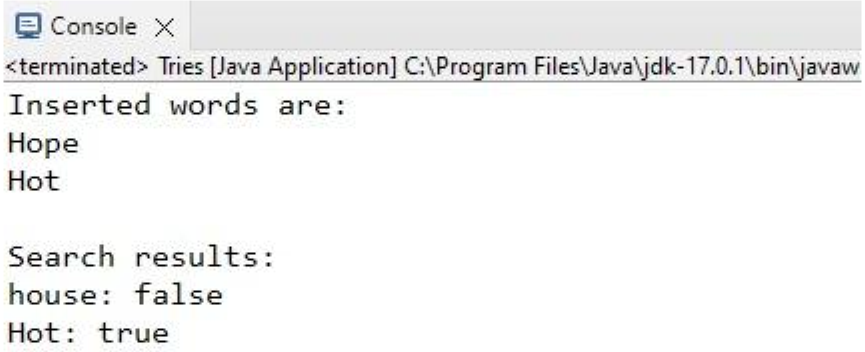


```java
            TrieNode node = current.children.get(c);
            if (node == null) {
                return false;
            }
            current = node;
        }
        return current.endOfWord;
    }

    public static void main(String[] args) {
        Tries trie = new Tries();

        trie.insert("Hot");
        trie.insert("Hope");

        System.out.println("Inserted words are:");
        trie.printAllWords();

        System.out.println();

        System.out.println("Search results:");
```



```java
        return current.endOfWord;
    }

    public static void main(String[] args) {
        Tries trie = new Tries();

        trie.insert("Hot");
        trie.insert("Hope");

        System.out.println("Inserted words are:");
        trie.printAllWords();

        System.out.println();

        System.out.println("Search results:");
        System.out.println("house: " + trie.search("house"));
        System.out.println("Hot: " + trie.search("Hot"));
    }
}
```

Output -


Console
<terminated> Tries [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw

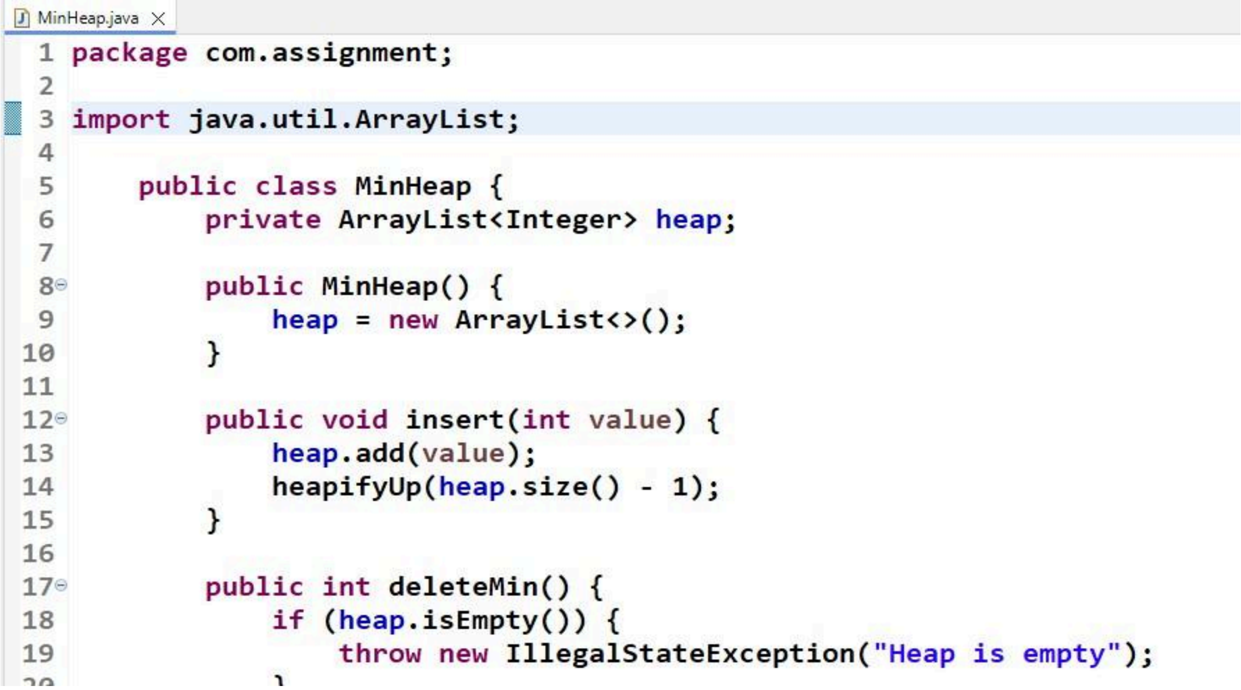
```
Inserted words are:
Hope
Hot

Search results:
house: false
Hot: true
```

3] Task 3: Implementing Heap Operations

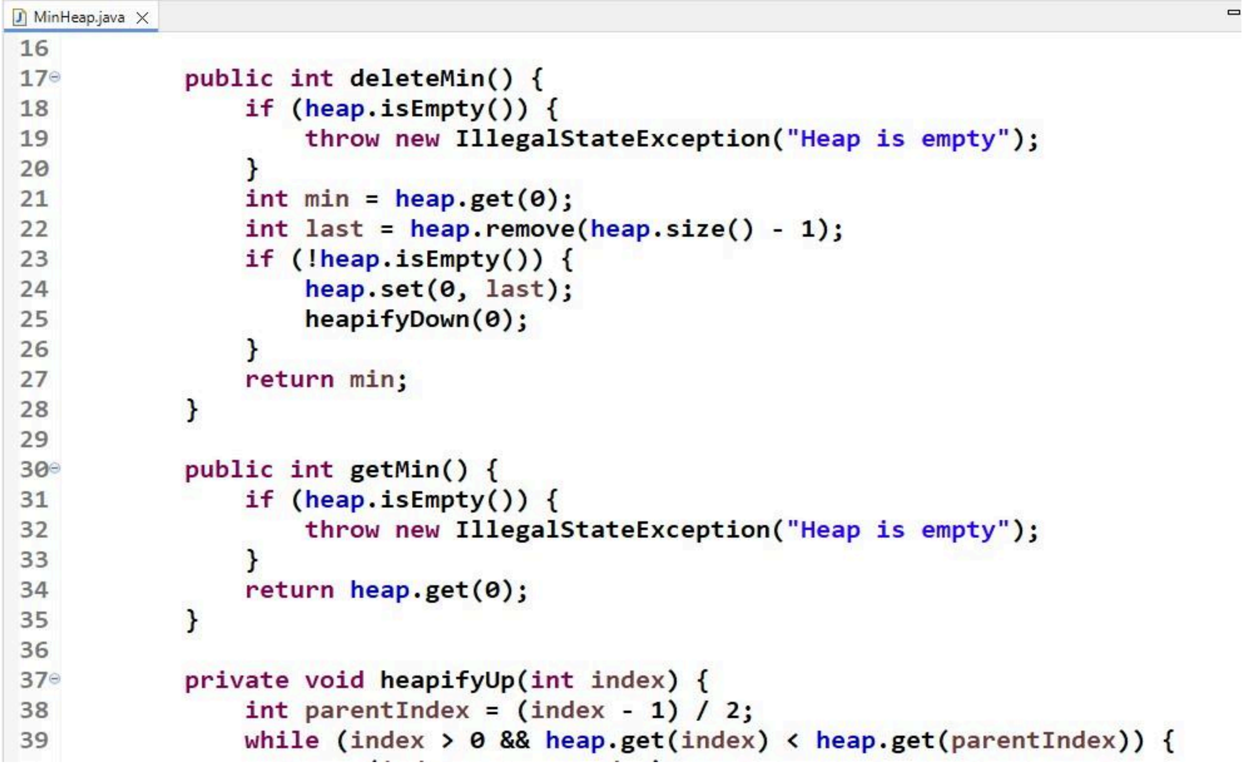
Code a min-heap in java with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

Solution:-

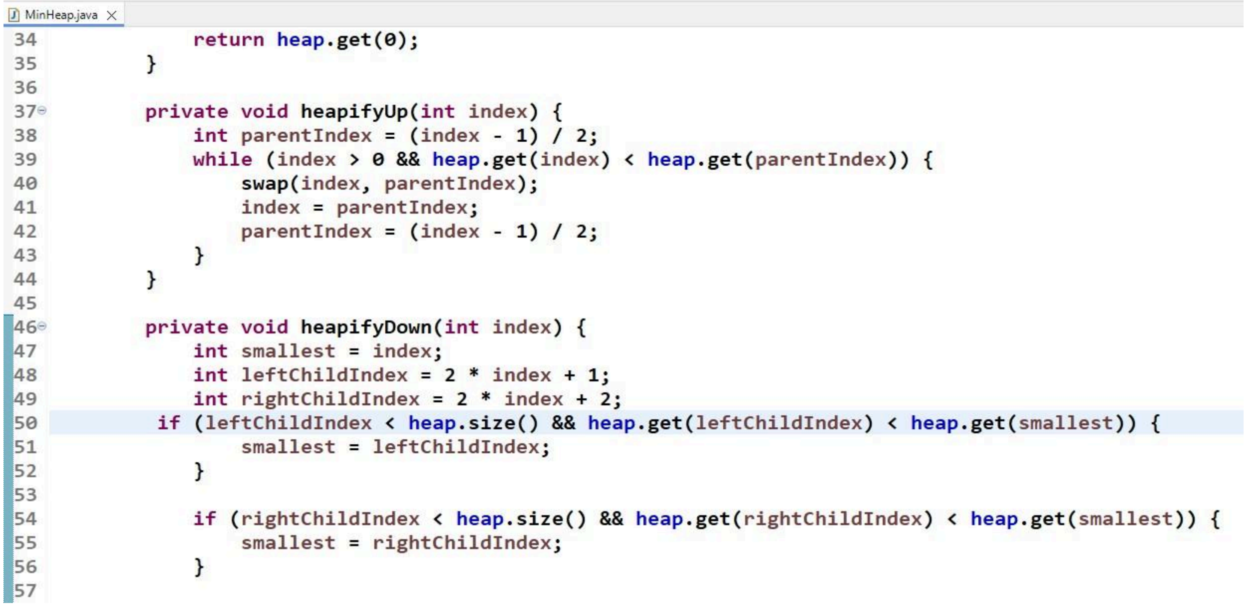
Code -



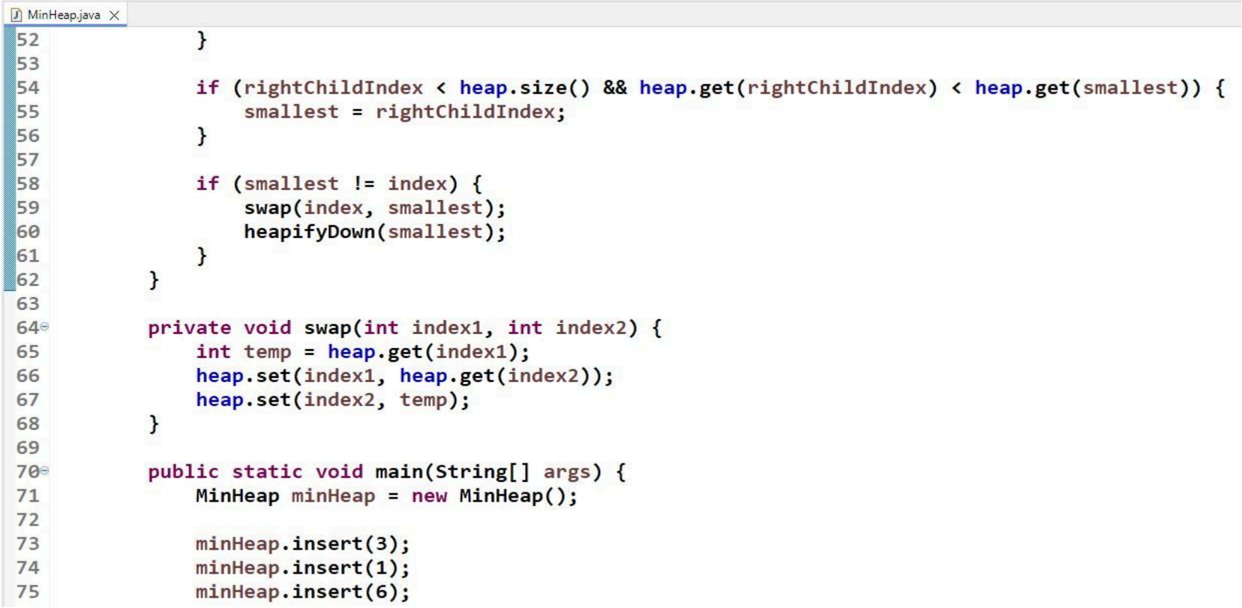
```java
package com.assignment;

import java.util.ArrayList;

    public class MinHeap {
        private ArrayList<Integer> heap;

        public MinHeap() {
            heap = new ArrayList<>();
        }

        public void insert(int value) {
            heap.add(value);
            heapifyUp(heap.size() - 1);
        }

        public int deleteMin() {
            if (heap.isEmpty()) {
                throw new IllegalStateException("Heap is empty");
```



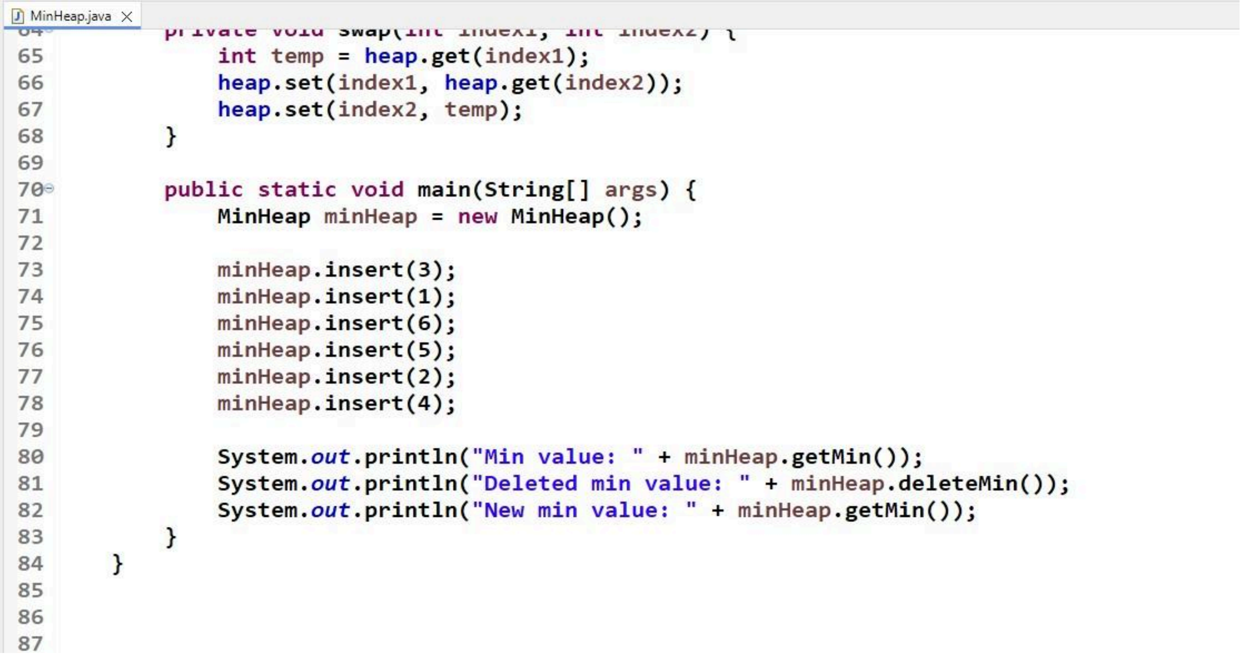
```java

        public int deleteMin() {
            if (heap.isEmpty()) {
                throw new IllegalStateException("Heap is empty");
            }
            int min = heap.get(0);
            int last = heap.remove(heap.size() - 1);
            if (!heap.isEmpty()) {
                heap.set(0, last);
                heapifyDown(0);
            }
            return min;
        }

        public int getMin() {
            if (heap.isEmpty()) {
                throw new IllegalStateException("Heap is empty");
            }
            return heap.get(0);
        }

        private void heapifyUp(int index) {
            int parentIndex = (index - 1) / 2;
            while (index > 0 && heap.get(index) < heap.get(parentIndex)) {
```



```java
            return heap.get(0);
        }

        private void heapifyUp(int index) {
            int parentIndex = (index - 1) / 2;
            while (index > 0 && heap.get(index) < heap.get(parentIndex)) {
                swap(index, parentIndex);
                index = parentIndex;
                parentIndex = (index - 1) / 2;
            }
        }

        private void heapifyDown(int index) {
            int smallest = index;
            int leftChildIndex = 2 * index + 1;
            int rightChildIndex = 2 * index + 2;
         if (leftChildIndex < heap.size() && heap.get(leftChildIndex) < heap.get(smallest)) {
                smallest = leftChildIndex;
            }

            if (rightChildIndex < heap.size() && heap.get(rightChildIndex) < heap.get(smallest)) {
                smallest = rightChildIndex;
            }

```

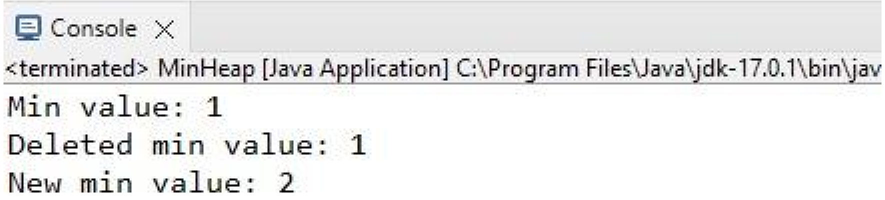


```java
            }

            if (rightChildIndex < heap.size() && heap.get(rightChildIndex) < heap.get(smallest)) {
                smallest = rightChildIndex;
            }

            if (smallest != index) {
                swap(index, smallest);
                heapifyDown(smallest);
            }
        }

        private void swap(int index1, int index2) {
            int temp = heap.get(index1);
            heap.set(index1, heap.get(index2));
            heap.set(index2, temp);
        }

        public static void main(String[] args) {
            MinHeap minHeap = new MinHeap();

            minHeap.insert(3);
            minHeap.insert(1);
            minHeap.insert(6);
```



```java
    private void swap(int index1, int index2) {
        int temp = heap.get(index1);
        heap.set(index1, heap.get(index2));
        heap.set(index2, temp);
    }

    public static void main(String[] args) {
        MinHeap minHeap = new MinHeap();

        minHeap.insert(3);
        minHeap.insert(1);
        minHeap.insert(6);
        minHeap.insert(5);
        minHeap.insert(2);
        minHeap.insert(4);

        System.out.println("Min value: " + minHeap.getMin());
        System.out.println("Deleted min value: " + minHeap.deleteMin());
        System.out.println("New min value: " + minHeap.getMin());
    }
}
```

Output -



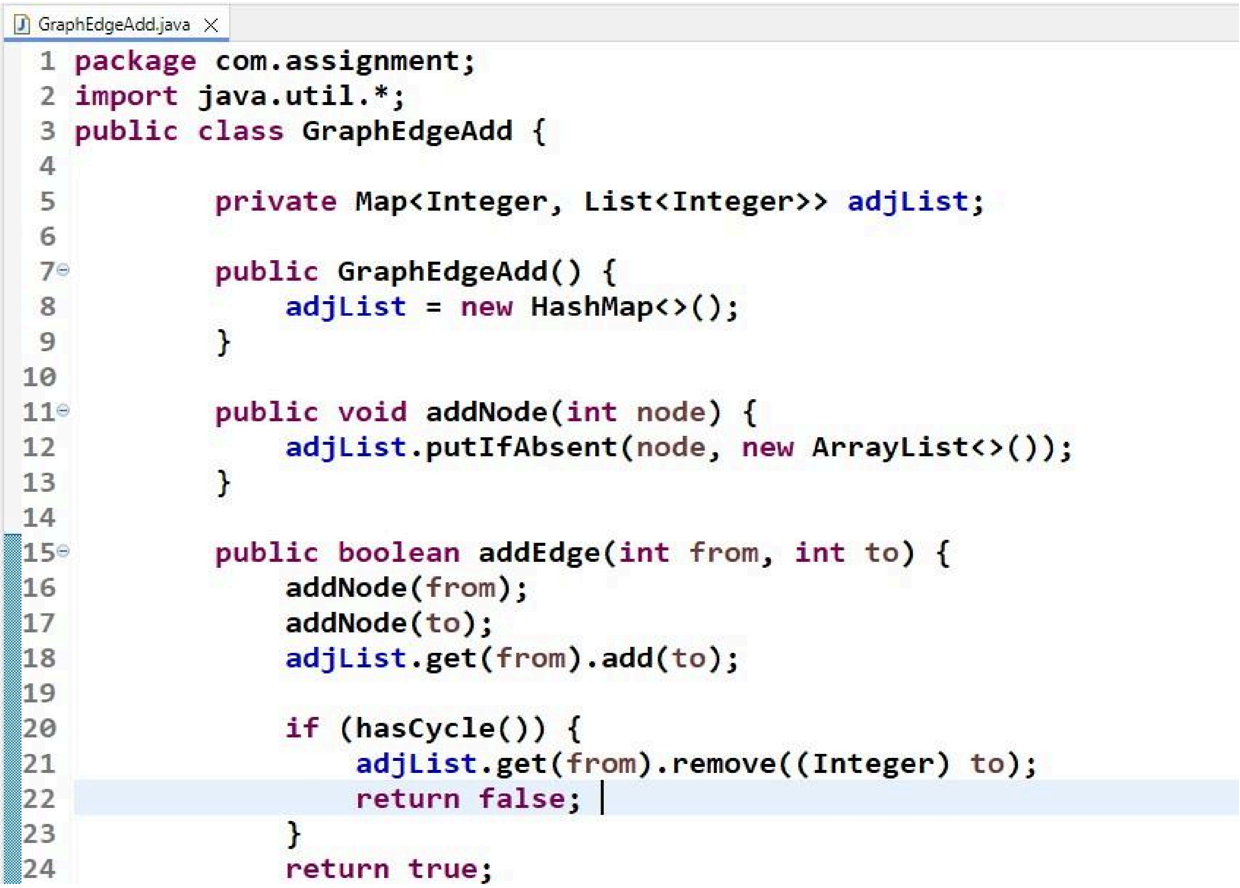
```
<terminated> MinHeap [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\jav
Min value: 1
Deleted min value: 1
New min value: 2
```

4] Task 4: Graph Edge Addition Validation

Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.

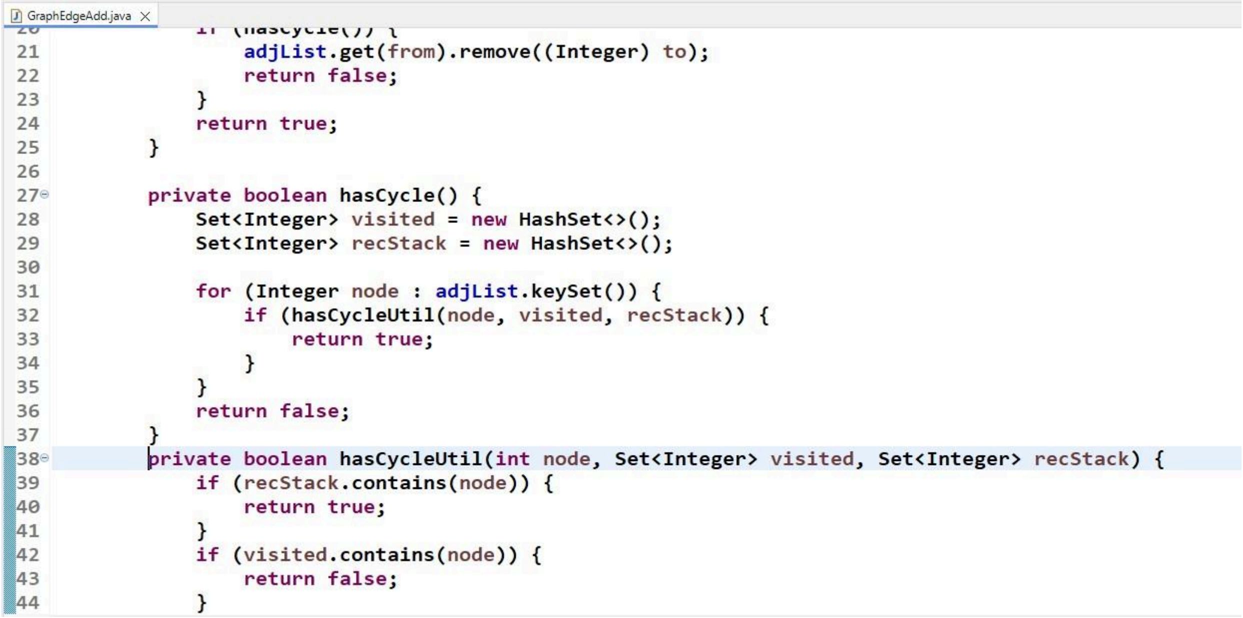
Solution:-

Code -



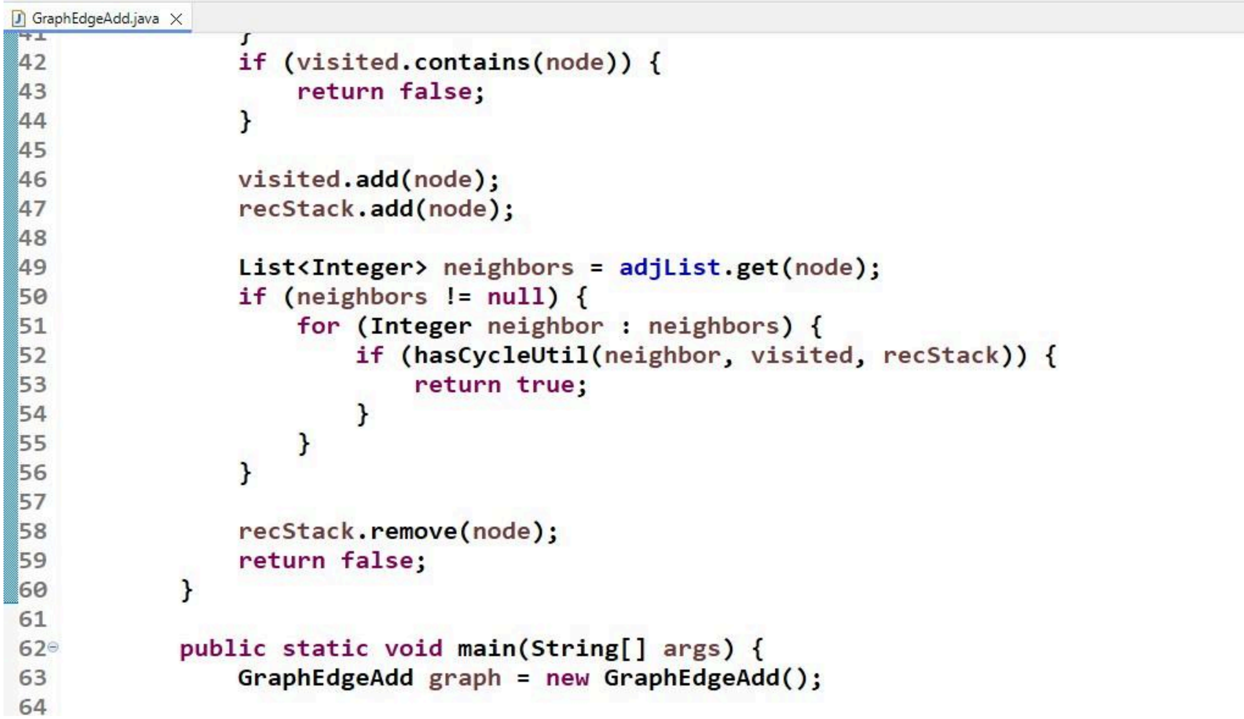
GraphEdgeAdd.java

```java
package com.assignment;
import java.util.*;
public class GraphEdgeAdd {

        private Map<Integer, List<Integer>> adjList;

        public GraphEdgeAdd() {
            adjList = new HashMap<>();
        }

        public void addNode(int node) {
            adjList.putIfAbsent(node, new ArrayList<>());
        }

        public boolean addEdge(int from, int to) {
            addNode(from);
            addNode(to);
            adjList.get(from).add(to);

            if (hasCycle()) {
                adjList.get(from).remove((Integer) to);
                return false;
            }
            return true;
```

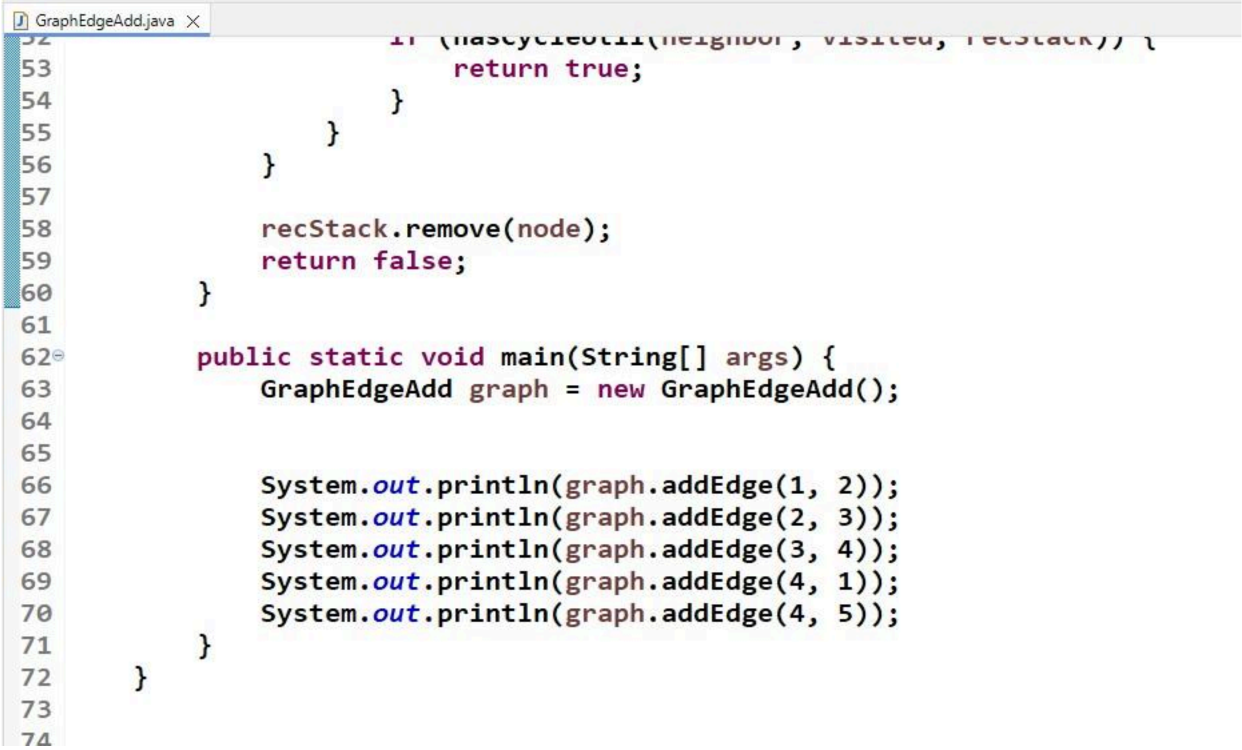


GraphEdgeAdd.java

```java
            if (hasCycle()) {
                adjList.get(from).remove((Integer) to);
                return false;
            }
            return true;
        }

        private boolean hasCycle() {
            Set<Integer> visited = new HashSet<>();
            Set<Integer> recStack = new HashSet<>();

            for (Integer node : adjList.keySet()) {
                if (hasCycleUtil(node, visited, recStack)) {
                    return true;
                }
            }
            return false;
        }
        private boolean hasCycleUtil(int node, Set<Integer> visited, Set<Integer> recStack) {
            if (recStack.contains(node)) {
                return true;
            }
            if (visited.contains(node)) {
                return false;
            }
```



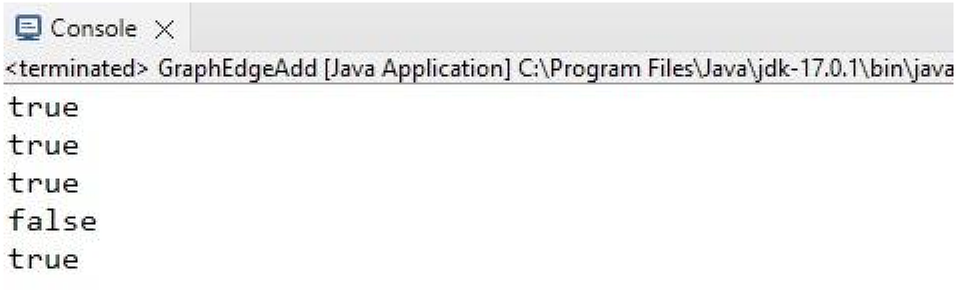
```
        }
        if (visited.contains(node)) {
            return false;
        }

        visited.add(node);
        recStack.add(node);

        List<Integer> neighbors = adjList.get(node);
        if (neighbors != null) {
            for (Integer neighbor : neighbors) {
                if (hasCycleUtil(neighbor, visited, recStack)) {
                    return true;
                }
            }
        }

        recStack.remove(node);
        return false;
    }

    public static void main(String[] args) {
        GraphEdgeAdd graph = new GraphEdgeAdd();

```



```
                if (hasCycleUtil(neighbor, visited, recStack)) {
                    return true;
                }
            }
        }

        recStack.remove(node);
        return false;
    }

    public static void main(String[] args) {
        GraphEdgeAdd graph = new GraphEdgeAdd();
```

```


        System.out.println(graph.addEdge(1, 2));
        System.out.println(graph.addEdge(2, 3));
        System.out.println(graph.addEdge(3, 4));
        System.out.println(graph.addEdge(4, 1));
        System.out.println(graph.addEdge(4, 5));
    }
}

```

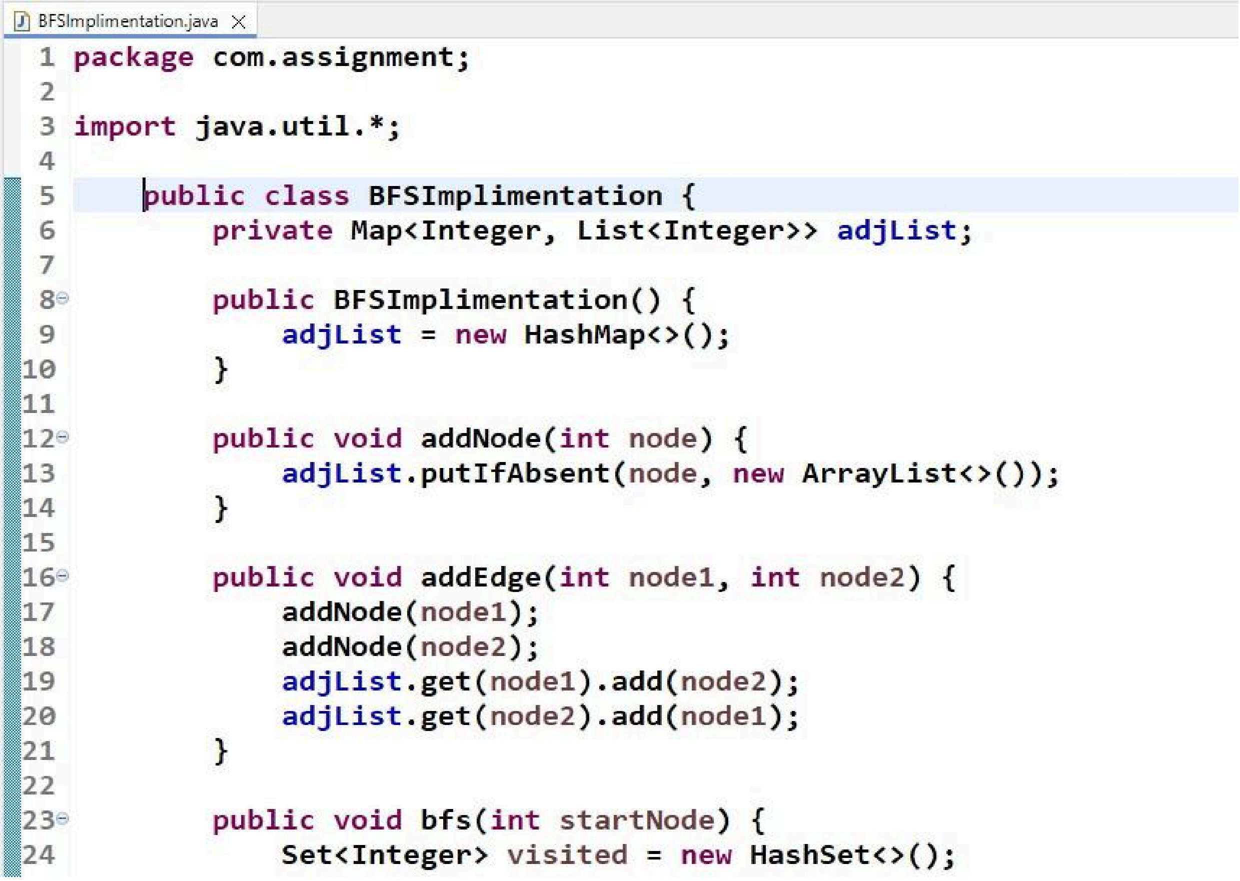
Output -



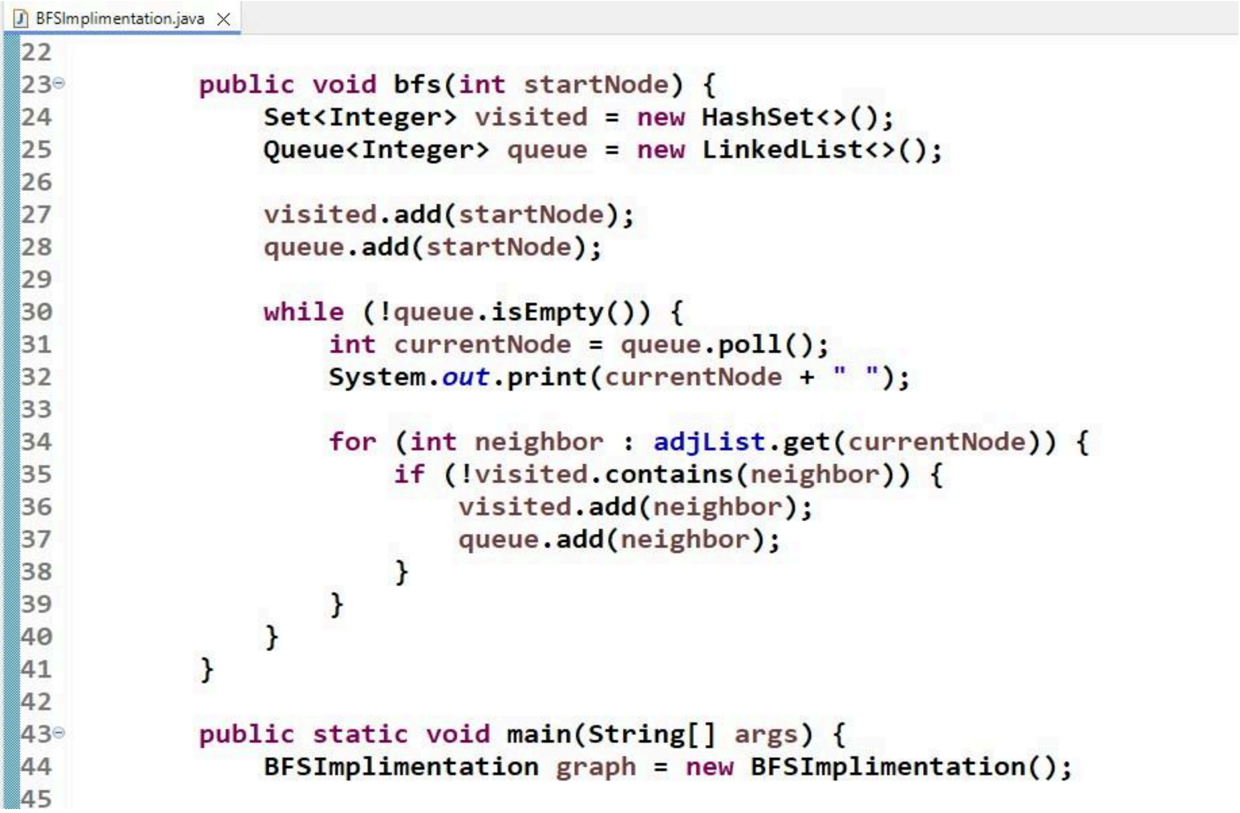
```
<terminated> GraphEdgeAdd [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\java
true
true
true
false
true
```

5] Task 5: Breadth-First Search (BFS) Implementation
For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.

Solution:-
Code -

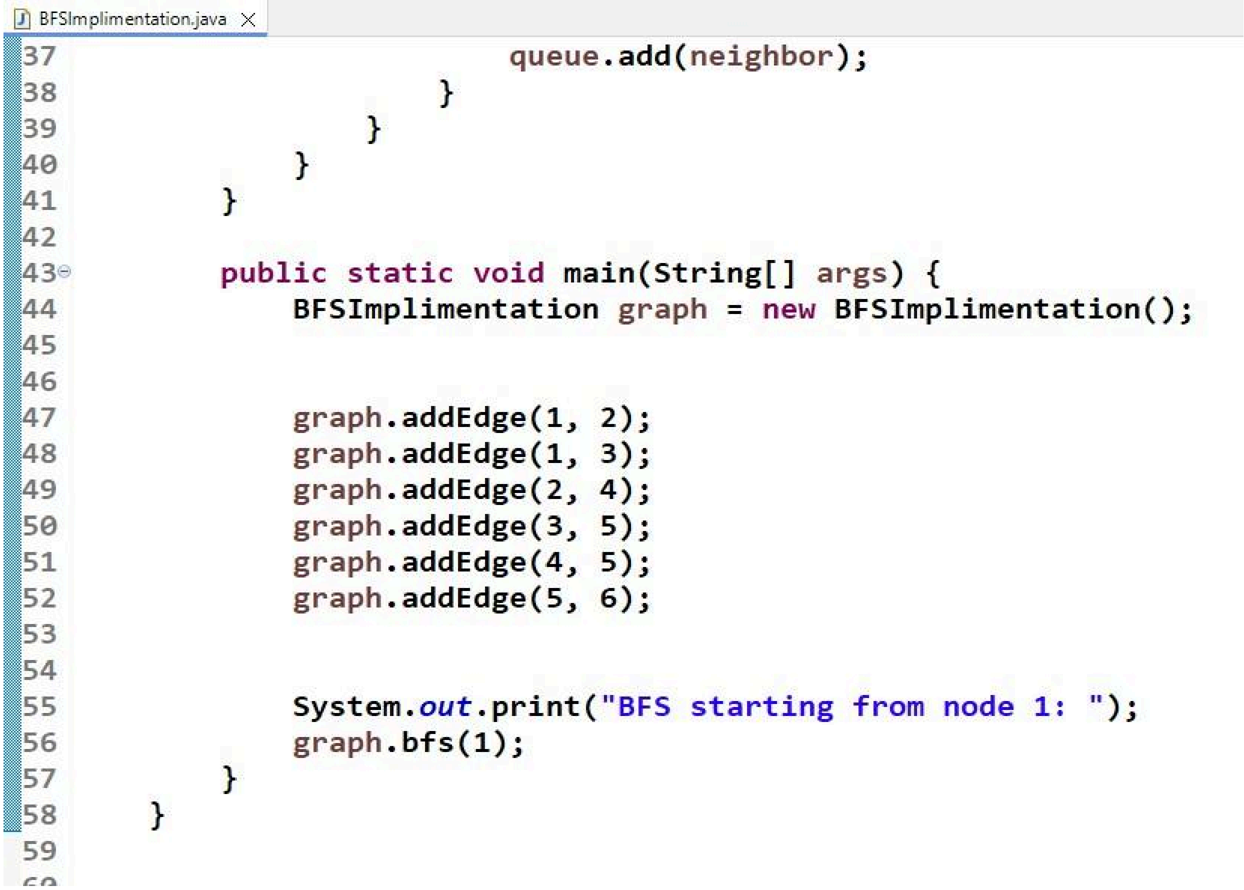


```java
package com.assignment;

import java.util.*;

    public class BFSImplimentation {
        private Map<Integer, List<Integer>> adjList;

        public BFSImplimentation() {
            adjList = new HashMap<>();
        }

        public void addNode(int node) {
            adjList.putIfAbsent(node, new ArrayList<>());
        }

        public void addEdge(int node1, int node2) {
            addNode(node1);
            addNode(node2);
            adjList.get(node1).add(node2);
            adjList.get(node2).add(node1);
        }

        public void bfs(int startNode) {
            Set<Integer> visited = new HashSet<>();
```



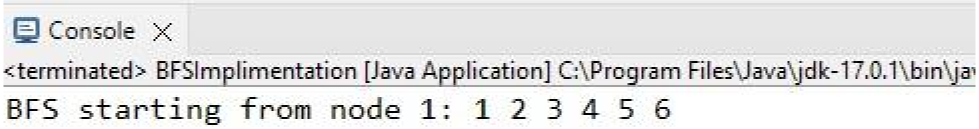
```java

        public void bfs(int startNode) {
            Set<Integer> visited = new HashSet<>();
            Queue<Integer> queue = new LinkedList<>();

            visited.add(startNode);
            queue.add(startNode);

            while (!queue.isEmpty()) {
                int currentNode = queue.poll();
                System.out.print(currentNode + " ");

                for (int neighbor : adjList.get(currentNode)) {
                    if (!visited.contains(neighbor)) {
                        visited.add(neighbor);
                        queue.add(neighbor);
                    }
                }
            }
        }

        public static void main(String[] args) {
            BFSImplimentation graph = new BFSImplimentation();

```

6] Task 6: Depth-First Search (DFS) Recursive



```
                    queue.add(neighbor);
                }
            }
        }
    }

    public static void main(String[] args) {
        BFSImplimentation graph = new BFSImplimentation();


        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 5);
        graph.addEdge(4, 5);
        graph.addEdge(5, 6);


        System.out.print("BFS starting from node 1: ");
        graph.bfs(1);
    }
}
```

Output -



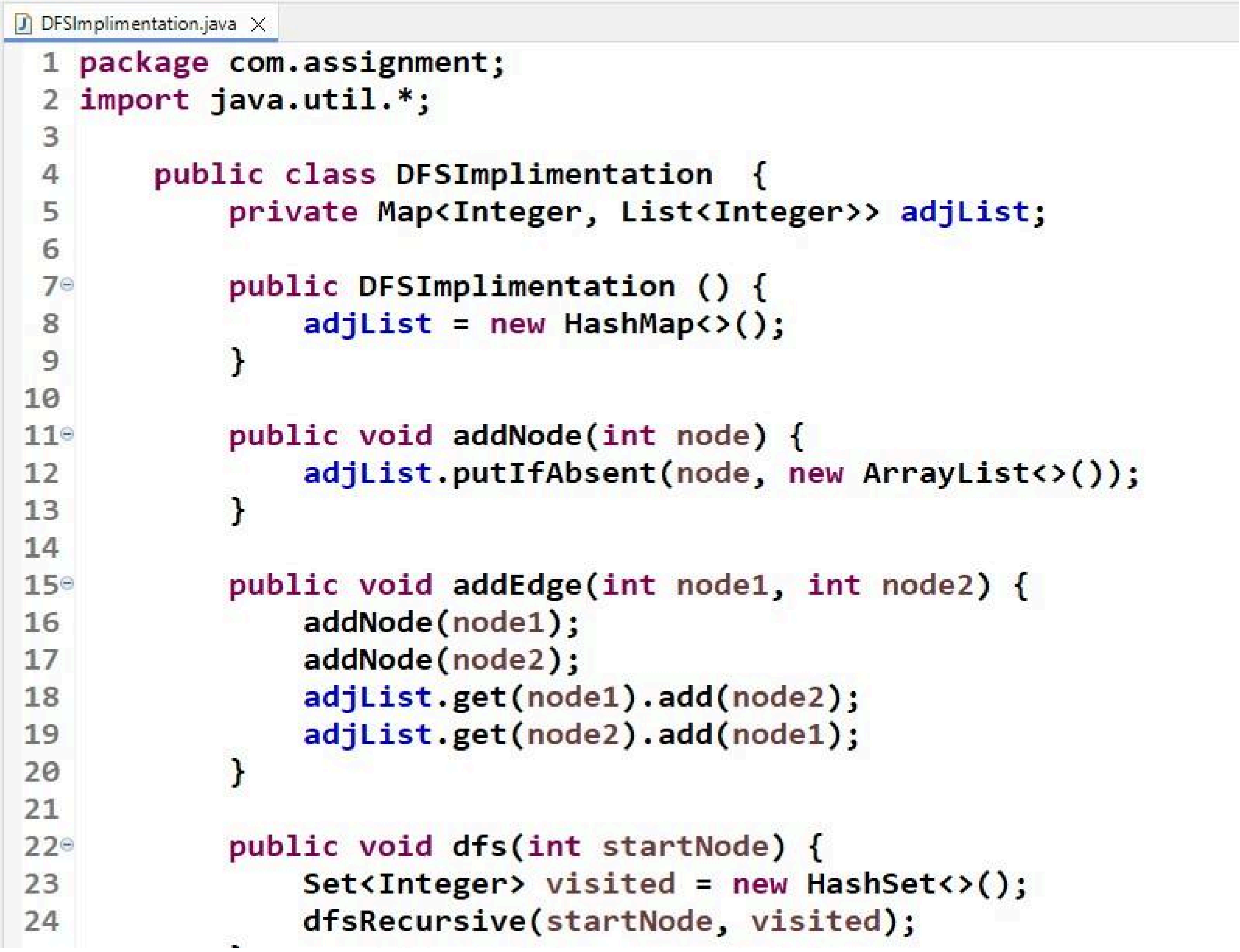
```
<terminated> BFSImplimentation [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\ja
BFS starting from node 1: 1 2 3 4 5 6
```

6] Task 6: Depth-First Search (DFS) Recursive

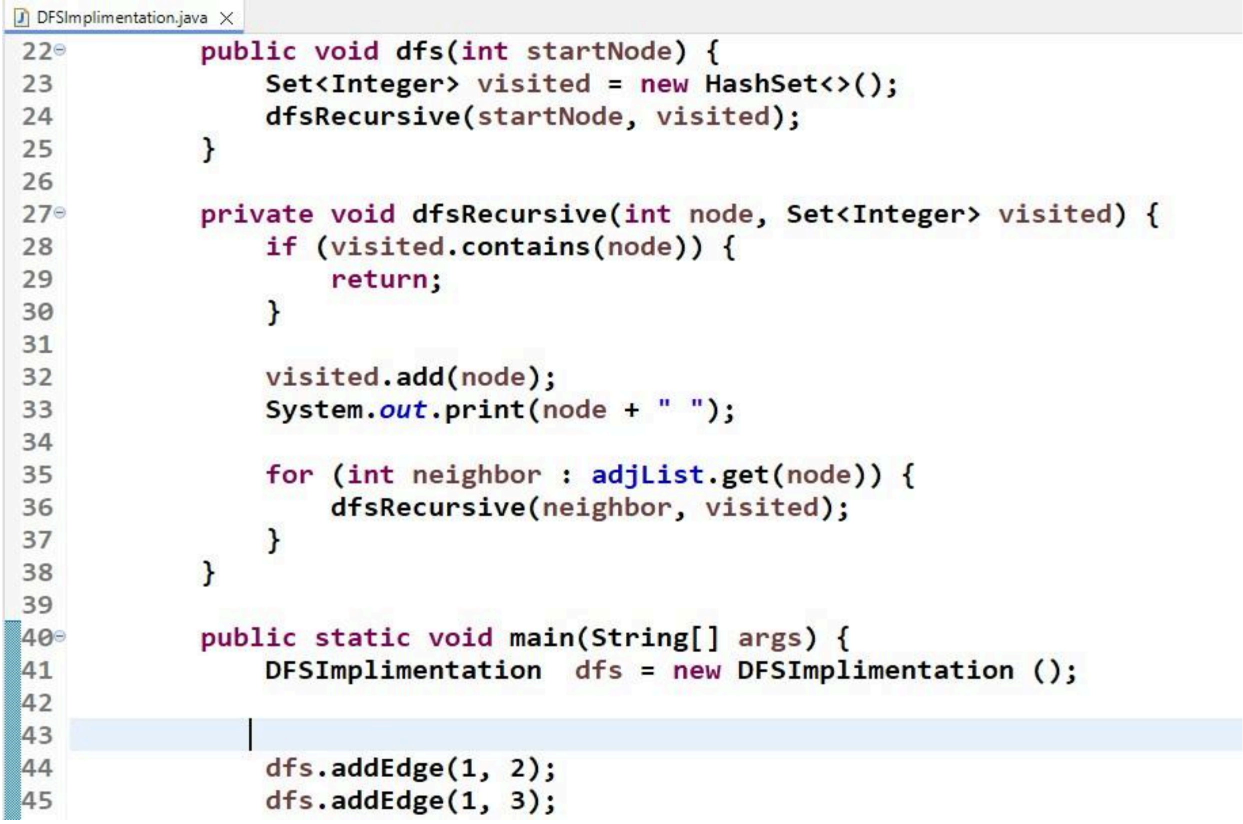
Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.

Solution:-

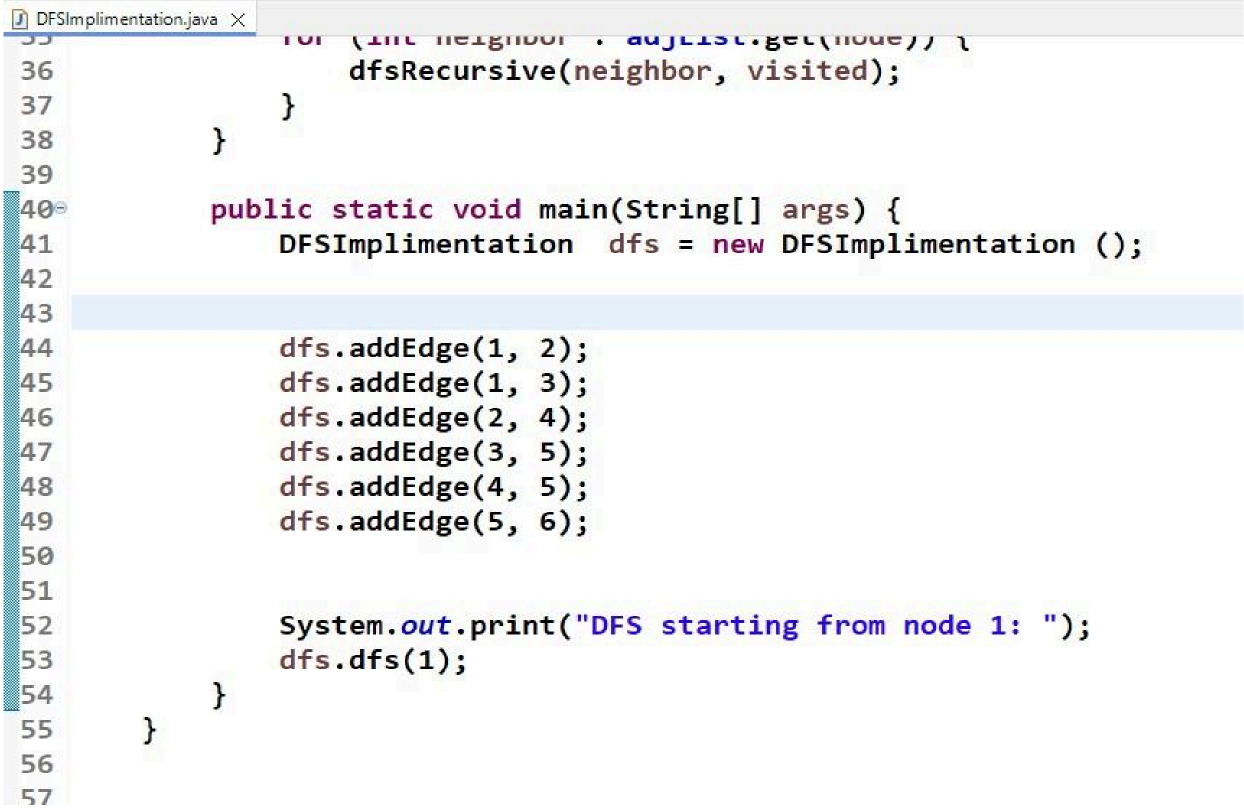
Code -



```java
package com.assignment;
import java.util.*;

    public class DFSImplimentation  {
        private Map<Integer, List<Integer>> adjList;

        public DFSImplimentation () {
            adjList = new HashMap<>();
        }

        public void addNode(int node) {
            adjList.putIfAbsent(node, new ArrayList<>());
        }

        public void addEdge(int node1, int node2) {
            addNode(node1);
            addNode(node2);
            adjList.get(node1).add(node2);
            adjList.get(node2).add(node1);
        }

        public void dfs(int startNode) {
            Set<Integer> visited = new HashSet<>();
            dfsRecursive(startNode, visited);
```

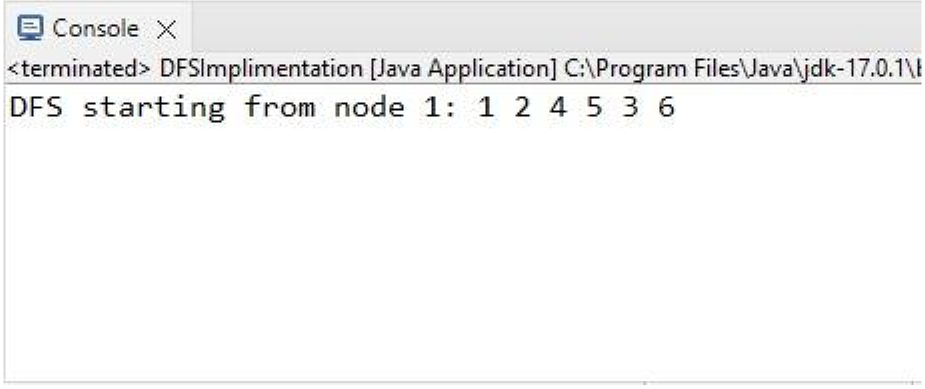


```
        public void dfs(int startNode) {
            Set<Integer> visited = new HashSet<>();
            dfsRecursive(startNode, visited);
        }

        private void dfsRecursive(int node, Set<Integer> visited) {
            if (visited.contains(node)) {
                return;
            }

            visited.add(node);
            System.out.print(node + " ");

            for (int neighbor : adjList.get(node)) {
                dfsRecursive(neighbor, visited);
            }
        }

        public static void main(String[] args) {
            DFSImplimentation  dfs = new DFSImplimentation ();


            dfs.addEdge(1, 2);
            dfs.addEdge(1, 3);
```



```
            for (int neighbor : adjList.get(node)) {
                dfsRecursive(neighbor, visited);
            }
        }

        public static void main(String[] args) {
            DFSImplimentation  dfs = new DFSImplimentation ();


            dfs.addEdge(1, 2);
            dfs.addEdge(1, 3);
            dfs.addEdge(2, 4);
            dfs.addEdge(3, 5);
            dfs.addEdge(4, 5);
            dfs.addEdge(5, 6);


            System.out.print("DFS starting from node 1: ");
            dfs.dfs(1);
        }
    }


```

Output -



```
<terminated> DFSImplimentation [Java Application] C:\Program Files\Java\jdk-17.0.1\
DFS starting from node 1: 1 2 4 5 3 6
```