Assignment-13

Implement an undirected graph in Java with the following functionalities:

Graph Class: Create a Graph class that supports:
Adding vertices and edges.
Removing vertices and edges.
Checking if vertices and edges exist.
Performing graph traversal algorithms like Depth-First Search (DFS) or Breadth-First Search (BFS).

```
import java.util.*;

class Graph {
   private Map<Integer, Set<Integer>> adjacencyList;


   public Graph() {
      adjacencyList = new HashMap<>();
   }

   public void addVertex(int v) {
      adjacencyList.putIfAbsent(v, new HashSet<>());
   }


   public void addEdge(int v1, int v2) {
      adjacencyList.get(v1).add(v2);
      adjacencyList.get(v2).add(v1);
   }

   public void removeEdge(int v1, int v2) {
      if (adjacencyList.containsKey(v1) && adjacencyList.containsKey(v2)) {
         adjacencyList.get(v1).remove(v2);
         adjacencyList.get(v2).remove(v1);
      }
```

```
    }

    public void removeVertex(int v) {
        if (adjacencyList.containsKey(v)) {
            // Remove vertex from all adjacency lists
            for (int neighbor : adjacencyList.get(v)) {
                adjacencyList.get(neighbor).remove(v);
            }
            adjacencyList.remove(v);
        }
    }

    public boolean hasEdge(int v1, int v2) {
        return adjacencyList.containsKey(v1) &&
adjacencyList.get(v1).contains(v2);
    }

    public boolean hasVertex(int v) {
        return adjacencyList.containsKey(v);
    }
    public List<Integer> dfs(int v) {
        List<Integer> visited = new ArrayList<>();
        Set<Integer> visitedSet = new HashSet<>();
        dfsHelper(v, visited, visitedSet);
        return visited;
    }

    private void dfsHelper(int v, List<Integer> visited, Set<Integer> visitedSet) {
        visited.add(v);
        visitedSet.add(v);
        for (int neighbor : adjacencyList.get(v)) {
            if (!visitedSet.contains(neighbor)) {
                dfsHelper(neighbor, visited, visitedSet);
            }
        }
    }

    public List<Integer> bfs(int v) {
        List<Integer> visited = new ArrayList<>();
        Queue<Integer> queue = new LinkedList<>();
```

```
        Set<Integer> visitedSet = new HashSet<>();

        queue.offer(v);
        visitedSet.add(v);

        while (!queue.isEmpty()) {
            int vertex = queue.poll();
            visited.add(vertex);
            for (int neighbor : adjacencyList.get(vertex)) {
                if (!visitedSet.contains(neighbor)) {
                    queue.offer(neighbor);
                    visitedSet.add(neighbor);
                }
            }
        }

        return visited;
    }

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

        // Adding vertices
        graph.addVertex(1);
        graph.addVertex(2);
        graph.addVertex(3);
        graph.addVertex(4);

        // Adding edges
        graph.addEdge(1, 2);
        graph.addEdge(2, 3);
        graph.addEdge(3, 4);
        graph.addEdge(4, 1);


        System.out.println("Adjacency List:");
        for (Map.Entry<Integer, Set<Integer>> entry :
graph.adjacencyList.entrySet()) {
            System.out.println(entry.getKey() + " -> " + entry.getValue());
        }
```

```
        System.out.println("\nDFS from vertex 1:");
        List<Integer> dfsResult = graph.dfs(1);
        System.out.println(dfsResult);

        System.out.println("\nBFS from vertex 1:");
        List<Integer> bfsResult = graph.bfs(1);
        System.out.println(bfsResult);
    }
}
```