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.

```
import java.util.*;
public class Graph {
    private int V; // Number of vertices
    private LinkedList<Integer>[] adj;
    public Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList<Integer>();
    }
    public void addEdge(int v, int w) {
        adj[v].add(w);
    }
    private boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack) {
        if (!visited[v]) {
            visited[v] = true;
            recStack[v] = true;
            for (Integer neighbor : adj[v]) {
                if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack)) {
                    return true;
                } else if (recStack[neighbor]) {
                    return true;                }           }       }
        recStack[v] = false;
        return false;
    }
    private boolean isCyclic() {
        boolean[] visited = new boolean[V];
```

```
    boolean[] recStack = new boolean[V];
    for (int i = 0; i < V; i++) {
        if (isCyclicUtil(i, visited, recStack)) {
            return true;            }        }
    return false;
}
public boolean addEdgeAndCheckCycle(int v, int w) {
    addEdge(v, w);
    boolean hasCycle = isCyclic();
    if (hasCycle) {
        adj[v].removeLast();
    }
    return !hasCycle;
}
public static void main(String[] args) {
    Graph graph = new Graph(4);
    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 2);
    graph.addEdge(2, 0);
    graph.addEdge(2, 3);
    graph.addEdge(3, 3);
    boolean isEdgeAdded = graph.addEdgeAndCheckCycle(1, 3);
    if (isEdgeAdded) {
        System.out.println("Edge added successfully.");
    } else {
        System.out.println("Edge not added due to cycle creation.");
    }   }   }
```

Explanation:

- **Graph Class:** Represents the directed graph and contains methods for adding edges, performing DFS traversal to check for cycles, and adding edges while checking for cycles.
- **addEdge(v, w):** Adds an edge from vertex v to vertex w.
- **isCyclicUtil(v, visited, recStack):** Recursive utility function to perform DFS traversal and check for cycles.
- **isCyclic():** Checks if the graph has a cycle using DFS traversal.
- **addEdgeAndCheckCycle(v, w):** Adds an edge between vertices v and w, and checks if the graph has a cycle. If a cycle is found, the edge is removed.