DFS CODE IN JAVA

```
import java.util.*;
public class Graph {
    private int V; // number of vertices
    private LinkedList<Integer>[] adj; // adjacency list
    // constructor
    public Graph(int v) {
        V = v;
        adj = new LinkedList[V];
        for (int i = 0; i < V; i++) {
            adj[i] = new LinkedList<>();
        }
    }
    // add an edge to the graph
    public void addEdge(int v, int w) {
        adj[v].add(w);
    }
    // DFS traversal
    public void DFS(int v, boolean[] visited) {
        visited[v] = true;
        System.out.print(v + " ");
        Iterator<Integer> it = adj[v].listIterator();
        while (it.hasNext()) {
            int n = it.next();
            if (!visited[n]) {
                DFS(n, visited);
            }
        }
    }
    // DFS traversal from a given source vertex
    public void DFS(int v) {
        boolean[] visited = new boolean[V];
        DFS(v, visited);
    }
```

```
    // main method to test DFS
    public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is the Depth First Traversal "+
                           "(starting from vertex 2)");

        g.DFS(2);
    }
}
```

Code Explanation :

Here is the algorithm for Depth First Search (DFS) for a graph with V vertices and E edges:

1. Create a boolean array `visited` of size V, and initialize all elements to false.
2. Create a stack `stack` to store the vertices to be visited.
3. Push the starting vertex `s` onto the stack `stack`, and mark it as visited by setting `visited[s]` to true.
4. While the stack `stack` is not empty, do the following:
    - Pop a vertex `v` from the stack `stack`.
    - Visit the vertex `v`, and do whatever operation needs to be done on it.
    - For each unvisited neighbor `n` of `v`, mark `n` as visited by setting `visited[n]` to true, and push it onto the stack `stack`.
5. Repeat step 4 until the stack `stack` is empty.

This algorithm ensures that all vertices in the graph are visited, and all connected components are explored. It also avoids revisiting any vertex that has already been visited.

Note that the implementation of the algorithm may differ depending on the data structure used to represent the graph and the specific requirements of the problem being solved.