```
public class Graph {

 /**
  * edges[i][j] is the weight of the edge from i to j, or 0 if
  * there is no such edge.
  */
 private double[][] edges;

 /** The argument is the number of vertices in this Graph. */
 public Graph(int vertices) {
   edges = new double[vertices][vertices]; // All zero by default
 }

 /** Add an edge of weight 1 from i to j. */
 public void addEdge(int i, int j) { edges[i][j] = 1; }

 /** Add edges of weight 1 from i to j and from j to i. */
 public void addUndirectedEdge(int i, int j) {
   edges[i][j] = 1;
   edges[j][i] = 1;
 }
```

```
/**
  * Return a list of the vertices reachable from source, in depth-
  * first order.
  */
 public List<Integer> depthFirstTraverse(int source) {
   List<Integer> result = new ArrayList<Integer>();
   boolean[] visited = new boolean[size()];
   depthFirstTraverse(source, result, visited);
   return result;
 }

 /**
  * Visit the vertices reachable from vertex, in depth-first order.
  * Add vertices to result as they are visited.
  */
 protected void depthFirstTraverse(int vertex,
                                   List<Integer> result,
                                   boolean[] visited) {
   visited[vertex] = true;
   result.add(vertex);
   for (Integer i : neighbors(vertex)) {
     if (!visited[i]) {
       depthFirstTraverse(i, result, visited);
     }
   }
 }
```