```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Stack;

/***
    Homework Assignment 5:
    Topological Sort for Class precedence list

    Group name: X-WORD
    Group members:
        * Kaan Kıranbay
        * Utku Mert Topçuoğlu
        * Çağatay Sel
        * Can Özgürel
        * Fuad Aghazada

    @date: 23.04.2019
*/
class Main
{
    /**
        Class for Graph data structure (Adjacency list implentation)
        + Topological sort algorithm
    */
    static class Graph
    {
        // Properties
        int vertices;
        LinkedList<String>[] adjList;
        ArrayList<String> vertice;

        /**
            Constructs the graph object with
            the given number of vertices

            @param vertices: number of vertices
        */
        Graph(int vertices)
        {
            this.vertices = vertices;
            adjList = new LinkedList[vertices];
            vertice = new ArrayList<>();
            for (int i = 0; i < vertices; i++)
            {
                adjList[i] = new LinkedList();
            }
        }

        /**
            Method addEdge: Adds an edge to the graph structure
            given the source and destination names

            @param source: source node name
            @param destination: destination node name
        */
        private void addEdge(String source, String destination)
        {
            if(vertice.size() == 0)
            {
                vertice.add(source);
```

```
        }
        boolean foundS = false;
        int i;
        for(i = 0; i < vertice.size(); i++)
        {
            if (vertice.get(i).equals(source))
            {
                foundS = true;
                break;
            }
        }

        if(!foundS)
        {
            vertice.add(source);
            System.out.println("Source " + i + ": " + source);
        }

        adjList[i].add(destination);

        boolean foundD = false;
        for(i = 0; i < vertice.size(); i++)
        {
            if (vertice.get(i).equals(destination))
            {
                foundD = true;
                break;
            }
        }

        if(!foundD)
        {
            vertice.add(destination);
        }
    }

    /**
        Topological sort algorithm
    */
    private void topologicalSorting()
    {
        boolean[] visited = new boolean[vertices];
        for (int i = 0; i < vertices; i++)
            visited[i] = false;

        Stack stack = new Stack<>();

        //visit from each node if not already visited
        for (int i = 0; i < vertices; i++)
        {
            if (!visited[i])
            {
                topologicalSortUtil(i, visited, stack);
            }
        }

        System.out.println("---Topological Sort--- (from highest to lowest)\n");

        // Printing the output
        int size = stack.size();
        int popped;
```

```
            for (int i = 0; i < size ; i++)
            {
                popped = (int) stack.pop();
                System.out.println((i+1) + ") " + vertice.get(popped));
            }
            System.out.println("\n-------------------------------------\n\n");

        }

        /**
            Helper method for implementing Topoogical sort algorithm

            @param start: start index for the sort
            @param visited: the array of visited nodes
            @param stack: stack for managing the class precedence
        */
        private void topologicalSortUtil(int start, boolean[] visited, Stack stack)
        {
            visited[start] = true;

            String i;
            int j;
            Iterator<String> it = adjList[start].iterator();
            while(it.hasNext())
            {
                i = it.next();
                for(j = 0; j < vertice.size(); j++)
                    if(vertice.get(j).equals(i))
                        if (!visited[j])
                            topologicalSortUtil(j, visited, stack);
            }

            stack.push(start);
        }
    }

    // Main for executing
    public static void main(String args[])
    {
        // Test 1
         Graph  test1 = new Graph(13);
         test1.addEdge("Jacque", "Athletes");
        test1.addEdge("Jacque", "Wheightlifters");
         test1.addEdge("Jacque", "Shotputters");
         test1.addEdge("Wheightlifters", "Athletes");
         test1.addEdge("Wheightlifters", "Endomorphs");
         test1.addEdge("Shotputters", "Athletes");
        test1.addEdge("Shotputters", "Endomorphs");
        test1.addEdge("Athletes", "Dwarfs");
        test1.addEdge("Endomorphs", "Dwarfs");
        test1.addEdge("Dwarfs", "Everything");
         test1.addEdge("Crazy", "Professors");
         test1.addEdge("Crazy", "Hackers");
         test1.addEdge("Professors", "Eccentrics");
         test1.addEdge("Hackers", "Eccentrics");
         test1.addEdge("Hackers", "Programmers");
        test1.addEdge("Professors", "Teachers");
        test1.addEdge("Eccentrics", "Dwarfs");
        test1.addEdge("Teachers", "Dwarfs");
        test1.addEdge("Programmers", "Dwarfs");
```

```
        // Test 2
        Graph  test2 = new Graph(9);
        test2.addEdge("Ord", "Real");
        test2.addEdge("Num", "Real");
        test2.addEdge("Num", "Fractional");
        test2.addEdge("Fractional", "Floating");
        test2.addEdge("Fractional", "RealFrac");
        test2.addEdge("Real", "RealFrac");
        test2.addEdge("Real", "Integral");
        test2.addEdge("Enum", "Integral");
        test2.addEdge("RealFrac", "RealFloat");
        test2.addEdge("Floating", "RealFloat");

        // Test 3
        Graph test3 = new Graph(7);
        test3.addEdge("Dwarfs", "Everything");
        test3.addEdge("Competitors", "Dwarfs");
        test3.addEdge("Gourmands", "Dwarfs");
        test3.addEdge("Diarists", "Dwarfs");
        test3.addEdge("Managers", "Competitors");
        test3.addEdge("Blimpy", "Managers");
        test3.addEdge("Blimpy", "Diarists");
        test3.addEdge("Blimpy", "Gourmands");

        // Output
        System.out.println("\n\nOUTPUT");
        System.out.println("\nTest data 1");
        test1.topologicalSorting();
        System.out.println("Test data 2");
        test2.topologicalSorting();
        System.out.println("Test data 3");
        test3.topologicalSorting();
    }
}
```

OUTPUT:

Source 7: Crazy
Source 2: Num
Source 7: Enum
Source 2: Competitors
Source 3: Gourmands
Source 4: Diarists
Source 5: Managers
Source 6: Blimpy

OUTPUT

Test data 1
---Topological Sort--- (from highest to lowest)

1) Crazy

2) Hackers
3) Programmers
4) Professors
5) Teachers
6) Eccentrics
7) Jacque
8) Shotputters
9) Wheightlifters
10) Endomorphs
11) Athletes
12) Dwarfs
13) Everything

Test data 2
---Topological Sort--- (from highest to lowest)

1) Enum
2) Num
3) Fractional
4) Floating
5) Ord
6) Real
7) Integral
8) RealFrac
9) RealFloat

Test data 3
---Topological Sort--- (from highest to lowest)

1) Blimpy
2) Managers
3) Diarists
4) Gourmands
5) Competitors
6) Dwarfs
7) Everything
