**Project Documentation**

**Name: Dan Beck**

**Assignment: Project 4**

**Date: October 12, 2020**

**Problem Statement**: A program that accepts information contained in a file about the class dependencies in a Java program and creates a directed graph from that information. From the directed graph, it produces two different kinds of displays of those dependency relationships.

**Analysis:** File used:

ClassA ClassC ClassE ClassJ

ClassB ClassD ClassG

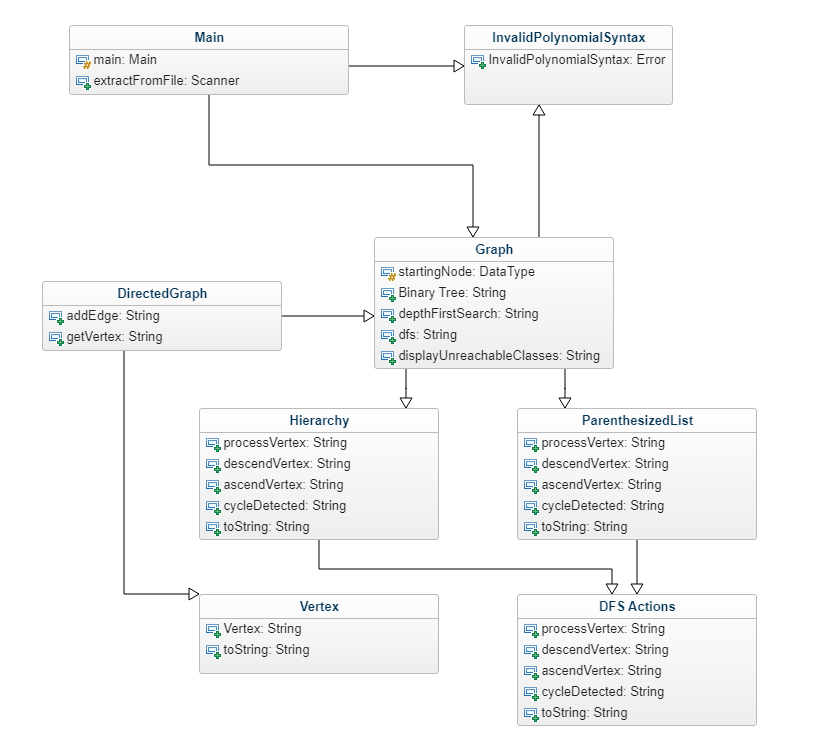
ClassC ClassA

ClassE ClassB ClassF ClassH

ClassJ ClassB

ClassI ClassC

**Design (for project assignments only):**



**Code:**

**package BeckProject4;**

**/\* File: Project 4 - Main Class**

**\* Author: Dan Beck**

**\* Date: October 10, 2020**

**\* Purpose: Executes the program. Scans a selected file and generates the output**

**\*/**

**import java.io.File;**

**import java.io.FileNotFoundException;**

**import java.util.NoSuchElementException;**

**import java.util.Scanner;**

**import javax.swing.JFileChooser;**

**import javax.swing.JOptionPane;**

**public class Main**

**{**

**static DirectedGraph graph = new DirectedGraph();**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: extract from file**

**\* Allows user to select file**

**\* Evaluates the lines from file**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public void extractFromFile()**

**{**

**//Allows user to select file and reads lines from the file**

**JFileChooser fileChooser = new JFileChooser();**

**fileChooser.setFileSelectionMode(JFileChooser.FILES\_AND\_DIRECTORIES);**

**fileChooser.setCurrentDirectory(new File(System.getProperty("user.dir")));**

**int status = fileChooser.showOpenDialog(null);**

**if (status == JFileChooser.APPROVE\_OPTION)**

**{**

**File file = fileChooser.getSelectedFile();**

**try**

**{**

**//scans each line. Creates one expression from each line**

**Scanner scan = new Scanner(file);**

**if (file.isFile())**

**{**

**//loop to create the list**

**while (scan.hasNextLine())**

**{**

**String edgeString = scan.nextLine();**

**String[] edge = edgeString.split(" ");**

**// Marks the first node of the graph**

**// DFS starts from this node**

**if (graph.startingNode == null)**

**{**

**graph.startingNode = graph.getVertex(edge[0]);**

**}**

**// add edges to the Directed graph**

**// First node of the Line - All other nodes**

**for (int i = 1; i < edge.length; i++)**

**{**

**graph.addEdge(edge[0], edge[i]);**

**}**

**}//end while (scan.hasNextLine())**

**}// if (file.isFile())**

**scan.close();**

**}//end try**

**catch (NoSuchElementException nse)**

**{**

**JOptionPane.showMessageDialog(JOptionPane.getRootFrame(), "The selected file is empty!");**

**}//end catch (NoSuchElementException nse)**

**catch (FileNotFoundException fnf)**

**{**

**JOptionPane.showMessageDialog(JOptionPane.getRootFrame(), "File can not be found!");**

**}//end catch (FileNotFoundException fnf)**

**}//end if (status == JFileChooser.APPROVE\_OPTION)**

**}//end public static void extractFromFile()**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: Main**

**\* Initializes main**

**\* Starts Depth First Search**

**\* Allows results to be displayed**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public static void main(String[] args)**

**{**

**// Initializing Main Class**

**new Main().extractFromFile();**

**// Starting Depth First Search Utility to complete the DFS**

**graph.depthFirstSearch();**

**// Display Parenthesized List after processing the vertices**

**System.out.print("Hierarchy: ");**

**System.out.println(graph.parenthesizedList.toString());**

**// Display Hierarchy after processing the vertices**

**System.out.println("Parenthesized List: ");**

**System.out.println(graph.hierarchy.toString());**

**// Display all the nodes that remained unreachable in the searching process**

**graph.displayUnreachableClasses();**

**}//end main**

**}//end public class Main**

**package BeckProject4;**

**/\* File: Project 4 - Graph**

**\* Author: Dan Beck**

**\* Date: October 11, 2020**

**\* Generates a graph based on the file selected**

**\*/**

**import java.util.\*;**

**public class Graph<V>**

**{**

**//Starting point of the graph**

**public V startingNode = null;**

**//Maps the vertex name (String) to a corresponding Vertex**

**Map<String, V> vertices = new HashMap<>();**

**//Adjacency representation of the graph**

**Map<V, ArrayList<V>> adjacencyList = new HashMap<>();**

**//Track if a node/vertex is visited in the searching process**

**Set<V> visited = new HashSet<>();**

**//Representation utility**

**ParenthesizedList hierarchy = new ParenthesizedList();**

**Hierarchy parenthesizedList = new Hierarchy();**

**//Tracks if the graph contains a circle**

**boolean cycle;**

**Set<V> discovered = new HashSet<>();**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: Depth First Search**

**\* initializes the DFS with all other related attributes**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public void depthFirstSearch()**

**{**

**// Marking cycle flag as false**

**cycle = false;**

**// Starting DFS from the first node of the input data**

**dfs(startingNode);**

**}//end public void depthFirstSearch()**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: DFS**

**\* Search in the adjacency list in Depth-First-Order**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**private void dfs(V node)**

**{**

**// check if the node is already visited in and not completed discovering it's child yet**

**// If so, a cycle has been detected**

**if (discovered.contains(node))**

**{**

**cycle = true;**

**// Perform DFS Actions Cycle Detected operation**

**hierarchy.cycleDetected();**

**parenthesizedList.cycleDetected();**

**return;**

**}//end if (discovered.contains(node))**

**//Perform DFS Actions Vertex Add operation**

**hierarchy.processVertex((Vertex) node);**

**parenthesizedList.processVertex((Vertex) node);**

**//Perform DFS Actions Descend Vertex operation**

**hierarchy.descendVertex((Vertex) node);**

**parenthesizedList.descendVertex((Vertex) node);**

**//add the node to the discovery list**

**discovered.add(node);**

**//mark the node as visited**

**visited.add(node);**

**//discover all of it's child**

**ArrayList<V> list = adjacencyList.get(node);**

**if (list != null)**

**{**

**for (V u : list)**

**dfs(u);**

**}//end if (list != null)**

**// Perform DFS Actions Ascend Vertex operation**

**hierarchy.ascendVertex((Vertex) node);**

**parenthesizedList.ascendVertex((Vertex) node);**

**// this node has discovered completely and remove it from the discovered list**

**discovered.remove(node);**

**}//end private void dfs(V node)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: Display Unreachable Classes**

**\* Prints all the unvisited nodes/classes**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public void displayUnreachableClasses()**

**{**

**// Loop all over the adjacency list**

**for (Map.Entry<V, ArrayList<V>> entry : adjacencyList.entrySet())**

**{**

**// for each entry check if there is any unvisited/undiscovered node/class**

**if(entry.getValue().size()>0)**

**{**

**// if found one print it and mark it as visited to avoid double printing**

**// check the node itself**

**if(!visited.contains(entry.getKey()))**

**{**

**System.out.println("Unreachable: " + entry.getKey());**

**visited.add(entry.getKey());**

**}//end if(!visited.contains(entry.getKey()))**

**// check all of it's adjacent nodes**

**for (V vertex : entry.getValue())**

**{**

**if(!visited.contains(vertex))**

**{**

**System.out.println("Unreachable: " + vertex);**

**visited.add(vertex);**

**}//end if(!visited.contains(vertex))**

**}//end for (V vertex : entry.getValue())**

**}//end if(entry.getValue().size()>0)**

**}//end for (Map.Entry<V, ArrayList<V>> entry : adjacencyList.entrySet())**

**}//end public void displayUnreachableClasses()**

**}//end public class Graph<V>**

**package BeckProject4;**

**/\* File: Project 4 - Hierarchy**

**\* Author: Dan Beck**

**\* Date: October 11, 2020**

**\* Generates the hierarchy string**

**\*/**

**import java.util.LinkedList;**

**import java.util.Queue;**

**public class Hierarchy implements DFSActions<Vertex>**

**{**

**Queue<String> res = new LinkedList<>();**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: processVertex**

**\* adds vertex to the string**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**@Override**

**public void processVertex(Vertex vertex)**

**{**

**res.add(vertex.toString());**

**}//end public void processVertex(Vertex vertex)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: descendVertex**

**\* adds opening parentheses**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**@Override**

**public void descendVertex(Vertex vertex)**

**{**

**res.add("(");**

**}//end public void descendVertex(Vertex vertex)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: ascendVertex**

**\* adds closing parentheses**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**@Override**

**public void ascendVertex(Vertex vertex)**

**{**

**res.add(")");**

**}//end public void ascendVertex(Vertex vertex)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: cycleDetected**

**\* adds asterisk when cycle is detected**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**@Override**

**public void cycleDetected()**

**{**

**res.add("\*");**

**}//end public void cycleDetected()**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: toString**

**\* generates the hierarchy list**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**@Override**

**public String toString()**

**{**

**String build = "";**

**int size = 0;**

**while (res.size() > 0)**

**{**

**String makeString = res.peek();**

**res.remove();**

**if (makeString == "(")**

**{**

**if (res.peek() == ")")**

**{**

**res.remove();**

**continue;**

**}//end if (res.peek() == ")")**

**else if (res.peek() == "\*")**

**{**

**build += res.peek() + " ";**

**res.remove();**

**res.remove();**

**continue;**

**}//end else if (res.peek() == "\*")**

**}//end if (makeString == "(")**

**if(makeString=="(")**

**{**

**size++;**

**}//end if(makeString=="(")**

**else if(makeString==")")**

**{**

**--size;**

**}//end else if(makeString==")")**

**if(makeString=="(" || makeString==")")**

**{**

**continue;**

**}//end if(makeString=="(" || makeString==")")**

**if(makeString!="\*")**

**{**

**build += "\n";**

**}//end if(makeString!="\*")**

**for (int i = 0; i < size; i++)**

**{**

**build += "\t";**

**}//end for (int i = 0; i < size; i++)**

**build += makeString + " " ;**

**}//end while (res.size() > 0)**

**build += "\n";**

**return build;**

**}//end public String toString()**

**}//end public class Hierarchy implements DFSActions<Vertex>**

**package BeckProject4;**

**/\* File: Project 4 - Vertex**

**\* Author: Dan Beck**

**\* Date: October 11, 2020**

**\* Sets the vertex**

**\*/**

**public class Vertex**

**{**

**private String name;**

**public Vertex(String name)**

**{**

**this.name = name;**

**}//end public Vertex(String name)**

**@Override**

**public String toString()**

**{**

**return name;**

**}//end public String toString()**

**}//end public class Vertex**

**package BeckProject4;**

**/\* File: Project 4 - DFS Actions**

**\* Author: Dan Beck**

**\* Date: October 11, 2020**

**\* creates the interface for DFS actions**

**\*/**

**public interface DFSActions<V>**

**{**

**public void processVertex(V vertex);**

**public void descendVertex(V vertex);**

**public void ascendVertex(V vertex);**

**public void cycleDetected();**

**}//end public interface DFSActions<V>**

**package BeckProject4;**

**/\* File: Project 4 - Directed Graph**

**\* Author: Dan Beck**

**\* Date: October 11, 2020**

**\* builds the directed graph from the graph onformation**

**\*/**

**import java.util.ArrayList;**

**public class DirectedGraph extends Graph<Vertex>**

**{**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: Add Edge**

**\* creates a directed edge and add it to the graph**

**\* u Node have a edge from (source node)**

**\* v Node have a edge to (destination node)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public void addEdge(String u, String v)**

**{**

**// Check if th source node already has some connected edges**

**ArrayList<Vertex> list = adjacencyList.get(getVertex(u));**

**// if already not in the Adjacency list**

**// Map it to a new Vertex and initialize**

**if (list == null)**

**{**

**list = new ArrayList<>();**

**}//end if (list == null)**

**// add a edge to source to destination**

**list.add(getVertex(v));**

**// update the adjacency list**

**adjacencyList.put(getVertex(u), list);**

**}//end public void addEdge(String u, String v)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DESCRIPTION: getVertex**

**\* checks if a node is already mapped to a vertex**

**\* u node(String) to be mapped**

**\* returns the mapped correspond vertex of the node**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**public Vertex getVertex(String u)**

**{**

**// if this node(String) showed up for the first time**

**// map it to a correspond vertex**

**if (!vertices.containsKey(u))**

**{**

**vertices.put(u, new Vertex(u));**

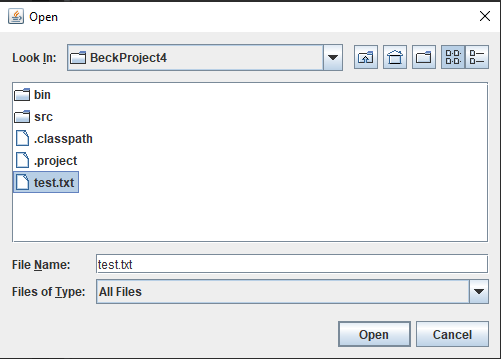
**}//end if (!vertices.containsKey(u))**

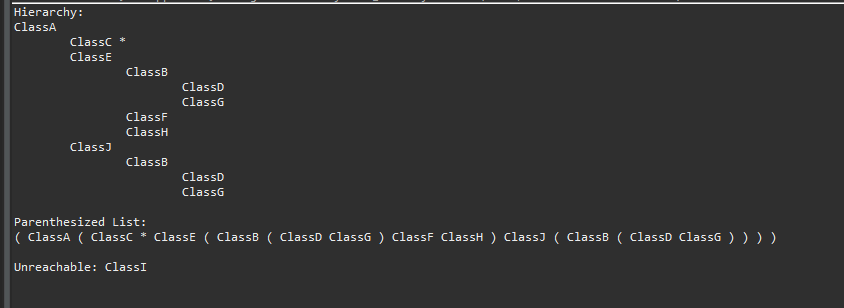
**return vertices.get(u);**

**}//end public Vertex getVertex(String u)**

**}//end public class DirectedGraph extends Graph<Vertex>**

**Output:**





**Reflection:** Although this project seemed to tie a lot of the other items from previous projects, this one was the most difficult of them. It took a while to build out how the graph would be laid out but the knowledge of this seems valuable.