Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	12-04
Receipt ID:	67f1988bc67997a71f025c7a4473726f

Turnin Successful!

The following file(s) were received:

TJGraphAdjList.java (4863 bytes)

```
1. //name:    date:
2. //resource classes and interfaces
3. //for use with Graphs3: EdgeList
4. //             Graphs4: DFS-BFS
5. //             Graphs5: EdgeListCities
6.
7. import java.io.*;
8. import java.util.*;
9. /********************  Graphs 3:  EdgeList ******************************/
10. interface VertexInterface
11. {
12.    public String toString();     //just return the name
13.    public String getName();
14.    public ArrayList<Vertex> getAdjacencies();
15.    public void addEdge(Vertex v);
16. }
17.
18. interface TJGraphAdjListInterface
19. {
20.    public List<Vertex> getVertices();
21.    public Vertex getVertex(int i) ;
22.    public Vertex getVertex(String vertexName);
23.    public Map<String, Integer> getVertexMap();
24.    public void addVertex(String v);
25.    public void addEdge(String source, String target);
26.    public String toString();
27.
28. }
29.
30.
31.    /*********************Graphs 4:  DFS and BFS ***************************/
32. interface DFSAndBFS
33. {
34.    public List<Vertex> depthFirstSearch(String name);
35.    public  List<Vertex> breadthFirstSearch(String name);
36.    public  List<Vertex> depthFirstRecur(String name);
37. }
38.
39.    /*****************Graphs 5:  EdgeList with Cities  *********/
40. interface EdgeListWithCities
41. {
42.    public void graphFromEdgeListData(String fileName) throws FileNotFoundException;
43.    public int edgeCount();
44.    public boolean isReachable(String source, String target);
45.    public boolean isConnected();
46. }
47. /*********************************************************/
48. class Vertex implements VertexInterface
49. {
50.    private final String name;
```

```
51.    private ArrayList<Vertex> adjacencies;
52.
53.   /* enter your code here  */
54.    public Vertex(String s)
55.    {
56.       name = s;
57.       adjacencies = new ArrayList<Vertex>();
58.    }
59.    public String toString()     //just return the name
60.    {
61.       return name;
62.    }
63.    public String getName()
64.    {
65.       return name;
66.    }
67.    public ArrayList<Vertex> getAdjacencies()
68.    {
69.       return adjacencies;
70.    }
71.    public void addEdge(Vertex v)
72.    {
73.       if(!adjacencies.contains(v))
74.          adjacencies.add(v);
75.    }
76. }
77. /*****************************************************/
78. public class TJGraphAdjList implements TJGraphAdjListInterface, DFSAndBFS //EdgeListWithCities
79. {
80.    private ArrayList<Vertex> vertices = new ArrayList<Vertex>();
81.    private Map<String, Integer> nameToIndex = new HashMap<String, Integer>();
82.
83.  /* enter your code here  */
84.    public List<Vertex> getVertices()
85.    {
86.       return vertices;
87.    }
88.    public Vertex getVertex(int i)
89.    {
90.       return vertices.get(i);
91.    }
92.    public Vertex getVertex(String vertexName)
93.    {
94.       return vertices.get(nameToIndex.get(vertexName));
95.    }
96.    public Map<String, Integer> getVertexMap()
97.    {
98.       return nameToIndex;
99.    }
100.   public void addVertex(String v)
101.   {
102.      vertices.add(new Vertex(v));
103.      nameToIndex.put(v, new Integer(vertices.size()-1));
104.   }
105.   public void addEdge(String source, String target)
106.   {
107.      if(nameToIndex.containsKey(source))
108.      {
109.         int index = nameToIndex.get(source);
110.         vertices.get(index).addEdge(new Vertex(target));
111.      }
112.   }
113.   public String toString()
114.   {
115.      String str = "";
116.      for(Vertex v: vertices)
117.      {
118.         str += v.getName() + " " +  v.getAdjacencies() +"\n";
119.      }
120.      return str;
121.   }
122.
123.   public List<Vertex> depthFirstSearch(String name)
124.   {
125.      int index = nameToIndex.get(name);
126.      return depthFirstSearch(vertices.get(index));
127.   }
128.   private List<Vertex> depthFirstSearch(Vertex v)
129.   {
130.      List<Vertex> list = new ArrayList<Vertex>();
131.      Stack<Vertex> stack = new Stack<Vertex>();
```

```
132.
133.        stack.push(v);
134.
135.        while(!stack.isEmpty())
136.        {
137.           Vertex temp = stack.pop();
138.           if(!list.contains(temp))
139.              list.add(temp);
140.           ArrayList<Vertex> edges = temp.getAdjacencies();
141.           for(Vertex x: edges)
142.           {
143.              x = getVertex(x.getName());
144.              if(!list.contains(x))
145.              {
146.                 stack.push(x);
147.              }
148.           }
149.        }
150.        return list;
151.     }
152.
153.     public  List<Vertex> breadthFirstSearch(String name)
154.     {
155.        int index = nameToIndex.get(name);
156.        return breadthFirstSearch(vertices.get(index));
157.     }
158.     private List<Vertex> breadthFirstSearch(Vertex v)
159.     {
160.        List<Vertex> list = new ArrayList<Vertex>();
161.        Queue<Vertex> queue = new LinkedList<Vertex>();
162.
163.        queue.add(v);
164.
165.        while(!queue.isEmpty())
166.        {
167.           Vertex temp = queue.remove();
168.           if(!list.contains(temp))
169.              list.add(temp);
170.           ArrayList<Vertex> edges = temp.getAdjacencies();
171.           for(Vertex x: edges)
172.           {
173.              x = getVertex(x.getName());
174.              if(!list.contains(x))
175.              {
176.                 queue.add(x);
177.              }
178.           }
179.        }
180.        return list;
181.
182.     }
183.
184.     public  List<Vertex> depthFirstRecur(String name)
185.     {
186.        return null;
187.     }
188.
189. }
190.
191.
192.
```