Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	12-05
Receipt ID:	224fad1bf4bf949eb9f033d7db18eac3

Turnin Successful!

The following file(s) were received:

TJGraphAdjList.java (6005 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.      if(!nameToIndex.keySet().contains(v))
103.      {
104.         vertices.add(new Vertex(v));
105.         nameToIndex.put(v, new Integer(vertices.size()-1));
106.      }
107.   }
108.   public void addEdge(String source, String target)
109.   {
110.      if(nameToIndex.containsKey(source))
111.      {
112.         int index = nameToIndex.get(source);
113.         vertices.get(index).addEdge(new Vertex(target));
114.      }
115.   }
116.   public String toString()
117.   {
118.      String str = "";
119.      for(Vertex v: vertices)
120.      {
121.         str += v.getName() + " " +  v.getAdjacencies() +"\n";
122.      }
123.      return str;
124.   }
125.
126.   public List<Vertex> depthFirstSearch(String name)
127.   {
128.      int index = nameToIndex.get(name);
129.      return depthFirstSearch(vertices.get(index));
130.   }
131.   private List<Vertex> depthFirstSearch(Vertex v)
```

```
132.    {
133.       List<Vertex> list = new ArrayList<Vertex>();
134.       Stack<Vertex> stack = new Stack<Vertex>();
135.
136.       stack.push(v);
137.
138.       while(!stack.isEmpty())
139.       {
140.          Vertex temp = stack.pop();
141.          if(!list.contains(temp))
142.             list.add(temp);
143.          ArrayList<Vertex> edges = temp.getAdjacencies();
144.          for(Vertex x: edges)
145.          {
146.             x = getVertex(x.getName());
147.             if(!list.contains(x))
148.             {
149.                stack.push(x);
150.             }
151.          }
152.       }
153.       return list;
154.    }
155.
156.    public  List<Vertex> breadthFirstSearch(String name)
157.    {
158.       int index = nameToIndex.get(name);
159.       return breadthFirstSearch(vertices.get(index));
160.    }
161.    private List<Vertex> breadthFirstSearch(Vertex v)
162.    {
163.       List<Vertex> list = new ArrayList<Vertex>();
164.       Queue<Vertex> queue = new LinkedList<Vertex>();
165.
166.       queue.add(v);
167.
168.       while(!queue.isEmpty())
169.       {
170.          Vertex temp = queue.remove();
171.          if(!list.contains(temp))
172.             list.add(temp);
173.          ArrayList<Vertex> edges = temp.getAdjacencies();
174.          for(Vertex x: edges)
175.          {
176.             x = getVertex(x.getName());
177.             if(!list.contains(x))
178.             {
179.                queue.add(x);
180.             }
181.          }
182.       }
183.       return list;
184.
185.    }
186.
187.    public  List<Vertex> depthFirstRecur(String name)
188.    {
189.       return null;
190.    }
191.
192.    public void graphFromEdgeListData(String fileName) throws FileNotFoundException
193.    {
194.       Scanner infile = new Scanner(new File(fileName));
195.       while(infile.hasNext())
196.       {
197.          String source = infile.next();
198.          String target = infile.next();
199.          addVertex(source);
200.          addEdge(source, target);
201.       }
202.    }
203.
204.    public int edgeCount()
205.    {
206.       int count = 0;
207.       for(Vertex v: vertices)
208.          for(Vertex x: v.getAdjacencies())
209.             count++;
210.       return count;
211.    }
212.
```

```
213.    public boolean isReachable(String source, String target)
214.    {
215.       List<Vertex> list = depthFirstSearch(source);
216.       boolean bool = false;
217.       for(Vertex v: list)
218.          if(v.getName().equals(target))
219.             bool = true;
220.       return bool;
221.    }
222.    public boolean isConnected()
223.    {
224.       Set<String> set = nameToIndex.keySet();
225.       boolean bool = true;
226.       for(String s: set)
227.       {
228.          List<Vertex> list = depthFirstSearch(s);
229.          if(list.size()<vertices.size()-1)
230.             bool = false;
231.       }
232.       return bool;
233.    }
234.
235. }
236.
237.
238.
```