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)

```java
//name:    date:
//resource classes and interfaces
//for use with Graphs3: EdgeList
//             Graphs4: DFS-BFS
//             Graphs5: EdgeListCities

import java.io.*;
import java.util.*;
/*********************  Graphs 3:  EdgeList ******************************/
interface VertexInterface
{
   public String toString();     //just return the name
   public String getName();
   public ArrayList<Vertex> getAdjacencies();
   public void addEdge(Vertex v);
}

interface TJGraphAdjListInterface
{
   public List<Vertex> getVertices();
   public Vertex getVertex(int i) ;
   public Vertex getVertex(String vertexName);
   public Map<String, Integer> getVertexMap();
   public void addVertex(String v);
   public void addEdge(String source, String target);
   public String toString();

}


   /*********************Graphs 4:  DFS and BFS ***************************/
interface DFSAndBFS
{
   public List<Vertex> depthFirstSearch(String name);
   public  List<Vertex> breadthFirstSearch(String name);
   public  List<Vertex> depthFirstRecur(String name);
}

   /***************Graphs 5:  EdgeList with Cities  *********/
interface EdgeListWithCities
{
   public void graphFromEdgeListData(String fileName) throws FileNotFoundException;
   public int edgeCount();
   public boolean isReachable(String source, String target);
   public boolean isConnected();
}
/************************************************************/
class Vertex implements VertexInterface
{
   private final String name;
   private ArrayList<Vertex> adjacencies;

  /* enter your code here  */
   public Vertex(String s)
   {
      name = s;
      adjacencies = new ArrayList<Vertex>();
   }
   public String toString()     //just return the name
   {
      return name;
   }
   public String getName()
   {
      return name;
   }
   public ArrayList<Vertex> getAdjacencies()
   {
      return adjacencies;
   }
   public void addEdge(Vertex v)
   {
      if(!adjacencies.contains(v))
         adjacencies.add(v);
   }
}
/*****************************************************/
public class TJGraphAdjList implements TJGraphAdjListInterface, DFSAndBFS //EdgeListWithCities
{
   private ArrayList<Vertex> vertices = new ArrayList<Vertex>();
   private Map<String, Integer> nameToIndex = new HashMap<String, Integer>();

 /* enter your code here  */
   public List<Vertex> getVertices()
   {
      return vertices;
   }
   public Vertex getVertex(int i)
   {
      return vertices.get(i);
   }
   public Vertex getVertex(String vertexName)
   {
      return vertices.get(nameToIndex.get(vertexName));
   }
   public Map<String, Integer> getVertexMap()
   {
      return nameToIndex;
   }
   public void addVertex(String v)
   {
      vertices.add(new Vertex(v));
      nameToIndex.put(v, new Integer(vertices.size()-1));
   }
   public void addEdge(String source, String target)
   {
      if(nameToIndex.containsKey(source))
      {
         int index = nameToIndex.get(source);
         vertices.get(index).addEdge(new Vertex(target));
      }
   }
   public String toString()
   {
      String str = "";
      for(Vertex v: vertices)
      {
         str += v.getName() + " " +  v.getAdjacencies() +"\n";
      }
      return str;
   }

   public List<Vertex> depthFirstSearch(String name)
   {
      int index = nameToIndex.get(name);
      return depthFirstSearch(vertices.get(index));
   }
   private List<Vertex> depthFirstSearch(Vertex v)
   {
      List<Vertex> list = new ArrayList<Vertex>();
      Stack<Vertex> stack = new Stack<Vertex>();

      stack.push(v);

      while(!stack.isEmpty())
      {
         Vertex temp = stack.pop();
         if(!list.contains(temp))
            list.add(temp);
         ArrayList<Vertex> edges = temp.getAdjacencies();
         for(Vertex x: edges)
         {
            x = getVertex(x.getName());
            if(!list.contains(x))
            {
               stack.push(x);
            }
         }
      }
      return list;
   }

   public  List<Vertex> breadthFirstSearch(String name)
   {
      int index = nameToIndex.get(name);
      return breadthFirstSearch(vertices.get(index));
   }
   private List<Vertex> breadthFirstSearch(Vertex v)
   {
      List<Vertex> list = new ArrayList<Vertex>();
      Queue<Vertex> queue = new LinkedList<Vertex>();

      queue.add(v);

      while(!queue.isEmpty())
      {
         Vertex temp = queue.remove();
         if(!list.contains(temp))
            list.add(temp);
         ArrayList<Vertex> edges = temp.getAdjacencies();
         for(Vertex x: edges)
         {
            x = getVertex(x.getName());
            if(!list.contains(x))
            {
               queue.add(x);
            }
         }
      }
      return list;

   }

   public  List<Vertex> depthFirstRecur(String name)
   {
      return null;
   }

}


```