Class Town implements Comparable<Town>

+ name: String
+ towns: Set<Town>
+ weight: Integer
+ previous: null

+ Town(String name): constructor that sets this.name = name
+Town(Town templateTown): constructor that sets this.name to templateTown.name, this.weight to templateTown.weight, this.towns to templateTown.weight, and this.previous to templateTown.previous
+compareTo(Town o): int
return this.name.compareToname(o.name)
+equals(Object o): run an equals method based on lower case conversion of this.name to o.name
+getName():String
return this.name
+hashCode(): int
return this.name.hashCode();
+toString():String
return the name and weight as a String delimited by a " "

Class Road implements Comparable<Road>

+ source: Town
+ source: Destination
+ name: String
+ weight: int

+ Town(Town source, Town destination, int degrees, String name)
Constructor with no weight preset
this.source = source, this.destination = destination, weight = degrees, this.name = name;
+ Town(Town source, Town destination, String name)
Constructor wigh default weight 1
this.source = source;
this.destination = destination;
this.weight =1;
this.name = name;
+ compareTo(Road o): int
return this.weight minus o.weight
+contains(Town town): boolean
return this.source.equals(town) or this.destination.equals(town)
+equals(Object r)
return this.source == r.source && this.destination == r.destination
+getDestination():Town
return this.destination
+getName(): String
return this.name
+getSource(): Town
return this.source
+getWeight(): int
return this.weight
+toString(): String
return name and weight delimited by a " "

Class Graph implements GraphInterface <Town, Road>

+ towns = Set<Town>
+ roads = Set<Road>
+ reviewed = Set<Town>
+ unreviewed = Set<Town>

+getEdge(Town sourceVertex, Town destinationVertex)
result = null
for(Road r: this.roads)
if r.contains(sourceVertex) or r.contains(destinationVertex) continue;
result = r;
break;
return result
+addEdge(Town sourceVertex, Town destinationVertex, int weight, String description)
s.towns.add(destinationVertex)
d.towns.add(sourceVertex)
Road result = new Road(sourceVertex, destinationVertex, weight, String)
add the result to the set of roads
return result
+addVertex(Town v): boolean
if this.towns has the town already return false
else this.towns.add(v)
return true
+containsEdge(Town sourceVertex, Town destinationVertex): boolean
return this.getedge(sourceVertex,destinationVertex) != null ? true: false;
+containsVertex(Town v): boolean
boolean result = false;
for(Town t: this.towns)
if v.equals(t) return true; break;
return result;
+edgeSet(): Set<Road>
return this.roads
+edgesOf(Town vertex): Set<Road>
Set<Road> result = new HashSet<Road>();
for (Road r: this.roads
if r.contains(v)
result.add(r) return result
+removeEdge(Town sourceVertex, Town destinationVertex, int weight, String description): Road
Road remove = null;
for(Road r: this.roads){
if (r.contains(s) == false || r.contains(d) == false) continue;
if r.weight is not equal to weight continue;
if name is null or r.name is not equal to name continue;
s.towns.remove(d)
d.towns.remove(s)
this.roads.remove(r)
result = r;
}
return result;
+removeVertex(Town v)
if (v is null or this.towns.contains(v) == false) return false
for(Road r: this.edgesOf(v)){
this.removeEdge(r.source,r.destination,r.weight, r.name);}
return this.towns.remove(v);
+vertexSet(): Set<Town>
return this.towns;
+shortestPath(Town sourceVertex, Town destinationVertex): ArrayList<String>
ArrayList<String> p = new ArrayList<String>();
this.reviewed = new HashSet<Town>();
this.unreviewed = new HashSet<Town>
this.reviewed.add(sourceVertex)
this.unreviewed.remove(sourceVertex)
for(Town t: this.towns){
t.reset();
}
sourceVertex = 0;
this.dijkstrasShortestPath(s);
this.buildPaths(p,sourceVertex, destinationVertex);
reverse the p
reurn p
+ dijkstrasShortestPath(Town sourceVertex): Void
{boolean found = false
while (found == false and this.unchecked != empty){
found = true;
Town shortestTown = null;
int shortestDistanne = Integer.MAX_VALUE;
for(Towns visited: this.checked){
Set<Town> r = visited.towns;
Set<Town> ur = new HashSet<Town>();
for(Town t: r) if unreviewed.contains(t) == false continue;
for(Town t: ur) int weight = this.calculateWeights(t, reviewed, sourceVertex)
if(weight<shortestDistance) shortestDistance = weight; shortestTown = t; t.previous = visited;
if(shortestTown!= null) found = false; shortestTown.weight = shortestDistance;
reviewed.add(shortestTown); unreviewed.remove(shortestTown);
+ calculateWeights(Town unreviewed, Town reviewed, Town source): int
return u.quals(s)? 0 : v.weight+this.getEdge(visited,unvisited).weight;
+buildPaths(ArrayList<String> p, Town sourceVertex, Town destinationVertex): Void
StringBuilder path = new StringBuilder();
Road r = this.getEdge(destinationVertex.previous,destinationVertex);