Elizabeth Perez

CMSC 204

Professor Monshi

12/14/2020

Design Document

|  |
| --- |
| Town  implements Comparable<Town> |
| String : townName |
| Town(String name)  Set townName to name  Town(Town templateTown)  Set townName to templateTown’s getName()  compareTo(Town o)  If o’s getName() equals townName  Return 0  Else  Return 1  equals(Object obj)  If obj’s compareTo(this) equals 0  Return true  Else  Return false  getName()  Return townName  hashCode()  Return townName’s hashCode  toString()  Return townName |

|  |
| --- |
| Road  implements Comparable<Road> |
| String : roadName  Town : roadSource, roadDestination  Int : weight |
| Road(Town source, Town destination, int degrees, String name)  Set roadName to name  Set roadSource to source  Set roadDestination to destination  Set weight to degrees  Road(Town source, Town destination, String name)  Set roadName to name  Set roadSource to source  Set roadDestination to destination  Set weight to 1  compareTo(Road o)  If o’s getName equals roadName  Return 0  Else  Return 1  equals(Object r)  If r’s getSource equals roadSource and r’s getDestination equals roadDestination  Return true  Else  Return false  getDestination()  Return roadDestination  getName()  Return roadName  getSource()  Return roadSource  getWeight()  Return weight  toString()  Return roadName |

|  |
| --- |
| Graph  implements GraphInterface<Town,Road> |
| LinkedList<Road>[] : adjList  Set<Town> : towns  Set<Road> : roads |
| getEdge(Town sourceVertex, Town destinationVertex)  For i less than adjList[sourceVertex’s hashCode]’s size  If adjList[sourceVertex’s hashCode]’s get(i)’s getDestination() equals destinationVertex  Return adjList[sourceVertex’s hashCode]’s get(i)  Return null  addEdge(Town sourceVertex, Town destinationVertex, int weight, String description)  Check if sourceVertex and destinationVertex are contained already  Declare a Road called edge and set to new Road(sourceVertex, destinationVertex, weight, description)  Add edge to adjList[sourceVertex’s hashCode]  Add edge to roads  Return edge  addVertex(Town v)  Add v to towns  containsEdge(Town sourceVertex, Town destinationVertex)  For i less than adjList[sourceVertex’s hashCode]’s size  If adjList[sourceVertex’s hashCode]’s get(i)’s getDestination() equals destinationVertex  Return true  Return false  containsVertex(Town v)  If towns contains(v)  Return true  Else  Return false  edgeSet()  Return roads  edgesOf(Town vertex)  Declare and initialize a Set<Roads> called edges  For i less than adjList[vertex’s hashCode]’s size  Add ajdList[vertex’s hashCode]’s get(i) to edges  Return edges  removeEdge(Town sourceVertex, Town destinationVertex, int weight, String description)  Declare a Road called edge and set to new Road(sourceVertex, destinationVertex, weight, description)  If roads contains edge  For i less than adjList[sourceVertex’s hashCode]’s size  If adjList[sourceVertex’s hashCode]’s get(i) equals edge  Remove adjList[sourceVertex’s hashCode]’s get(i)  Return edge  Return null  removeVertex(Town v)  If towns contains v  Set adjList[v’s hashCode] to null  Return true  Else  Return false  vertexSet()  Return towns  shortestPath(Town sourceVertex, Town destinationVertex)  Find the shortest path from the sourceVertex to the destinationVertex  Call dijktraShortestPath(sourceVertex)  dijkstraShortestPath(Town sourceVertex)  Return shortest paths to every other town |

|  |
| --- |
| TownGraphManager  implements TownGraphManagerInterface |
| Graph : graph |
| AddRoad(String town1, String town2, int weight, String roadName)  Declare Road called road and set to new Road(town1, town2, weight, roadName)  Declare Road called added and set to graph’s addEdge(road)  If added is null  Return false  Else  Return true  GetRoad(String town1, String town2)  Declare a Road called got and set to graph’s getEdge(town1, town2)  If got is null  Return null  Else  Return got’s getName()  AddTown(String v)  Declare a Town named town and set to new Town(v)  Call graph’s addVertex(town)  Return graph’s containsVertex(town)  GetTown(String name)  If containsTown(name)  Declare a Town called town and set to new Town(name)  Return town  Else  Return null  ContainsTown(String v)  Declare a Town named town and set to new Town(name)  Return graph’s containsVertex(town)  ContainsRoadConnection(String town1, String town2)  Declare a Town called source and set to a new Town(town1)  Declare a Town called destination and set to a new Town(town2)  Return graph’s containsEdge(source, destination)  AllRoads()  Return graph’s edgeSet()  DeleteRoadConnection(String town1, String town2, String road)  Declare Road called edge and set graph’s getEdge(town1, town2)  Declare a Town called source and set to graph’s getVertex(town1)  Declare a Town called destination and set to graph’s getVertex(town2)  Declare Road called removed and set to graph’s removeEdge(source, destination, edge’s getWeight, road)  If removed is null  Return false  Else  Return true  DeleteTown(String v)  Declare a Twon named town and set to new Town(v)  Call graph’s removeVertex(town)  AllTowns()  Return graph’s vertexSet()  GetPath(String town1, String town2)  Return graph’s shortestPath(town1, town2) |