Hunegnaw Chanie

Pro. Alexander Robert

CMSC204 design #6

Pseudocode

**Create class Town implements comparable**

Private string name, int weight, set <town > town, town backPath.

Constructor for the town class

Public town(String name, town templateTown)

{this.name =name;

This.name=templateTown.name;  
}

Public String getName(){return Name;}

Public int getWeight(){return weight;}

Public void setWeight(int weight){this.weigth=weight;}

Public void setWeight(int weight){this.weigth=weight;}

Public town getBackpath(){return Backpath;}

Public void setBackpath(town backpath){this.Backpath=backpath;}

addTown: public void addTowns(town T){adjTown.add(T);}

removeTown: public void removeTowns(town T){adjTown.remove(T);}

get adjacent town method: public set<Town> getAdjecentTowns(){return adjTowns;}

set adjacent town method: public void setAdjacentTowns(Set<Town> adjacentTowns) {this.AdjTowns = adjacentTowns;

}

Resetlocation: public void resetLocation() {

Weight = Integer.MAX\_VALUE;

Backpath= null;}

, hash code: public int hashCode() {

Final int p = 31;int result = 1;

result = p \* result + ((Name ==null) ? 0 : Name.hashCode());

return result;}

equals: public boolean equals(Object obj) {

If(this == obj)

Return true;

…

}, compareTo method.

**Create road class that implements comparable**

Private string name, int weight, town destination, town source, set <town > town

Constructor for the town class

Public Road (String name, int weight, town destinationTown, town sourceTown)

Public String getName() {

return name;

}

Public int getWeight() { return weight}

Public void setName(String name) {this.name = name;}

Public void setWeight(int weight) {this.weight = weight;}

Public Set<Town> getTowns() {return Towns;}

Public void setTowns(Set<Town> towns) {Towns = towns;}

Public Boolean contains(Town town) {

If(this.source.equals(town) ||this.destination.equals(town)) This true;

Return false }

Public int compareTo(Road other) {return this.getWeight() - other.getWeight();}

conatains method

public int hashCode(){…; }.

Public Boolean equals(object obj){ if statements here}

Public String toString (){return name;}

**create graph class that implements graphInterface**

**public classTownGraph implements GraphInterface< town, road>{ … }**

getEdge:public Road getEdge(town sourceVertex, town destinationVertex) { iterator go here, and add source and destination vertex, use while loop here for iterator}

public Road addEdge(town sourceVertex, town destinationVertex,int weight, string destination )

public Boolean addVertex(Town v){return setTown.add(v);}

contains edge: public Boolean containsEdge(Town sourceVertex, town destinationVertex)

{ use iterator loop for it}

Public Boolean containsVertex(town v){ return setTown.contains(v);}

set<road> edgeset {return setRoad}

set<road> edgeOf (town vertex){iterator and while loop here}

public Road removeEdge(town sourceVertex, town destinationVertex,int weight, string destination )

public Boolean removeVertex(Town v){return setTown.add(v);}

public set<town> vertexSet (){return setTown;}

set<town> vertexOf method

arraylist string shortestPath method: public arraylist<string> shortestpath(town sourceVertex, town destinationVertex) { put stuff here}. to find the shortest path.

Private void getShortestpath(town sourceVertex, town destinationVertex) { put stuff here}.

GetadijstrashortestPath: public dijkstraShortestpath(town sourceVertex){ while and got loop go here)

Private int getTotalweight(town unvisitedtown, town visitedtown, town sourceVertex){ use if statement }

**Create TownGraphManager class that Implement** the TownGraphManagerInterface. There are methods to populate the graph (reading from a text file),

Public class TownGraphManager implements TownGraphManagerInterface{

Private town Graph m=new town Graph();

Private ArrayList<string> road=new Arraylist<string>();}

Private ArrayList<string> town=new Arraylist<string>();

Public boolean populateTownGraph(File file) throws exception{ while ;loop go here}

Public town getTownfromMap(string name ){ }

Public int getRoadfromMapweight( string name, town a, town b){for loop go here}

public boolean addRoad(string sourcetown, string destinationtown, string destinationtown,int weight, int roadname ){}

public string getRoad(string sourcetown, string destinationtown){ if clause go here}

public boolean addtown(string townName){ if clause go here}

public boolean containstown(string townName(){}

public boolean containsRoadconnection(string sourcetown, string destinationtown)

public arraylist<string> allRoad(){}

public boolean deleteRoadconnection(string sourcetown, string destinationtown){ if statement here}

public boolean deletetown(string townName){ if statement go here}

Public ArrayList<string> allTown(){…}

public arrayList<string> getpath(string sourcetown, string destinationtown)

{ if clause go here}

public Town getTown(string town1)

{

Town source =new town(town1);

Return source ;}

}

**Exception Classes**

FileNotFoundException – created and thrown when the selected input file is not found.

IOException – created and thrown when user selects an input file that cannot be read

**GUI Driver (provided for you)**

Town section where you can add towns.

Road section where you add roads.

Find Connection section

administration section.

It has a Read File button, which allows the text files provided to be read and populate the graph.

**Testing**

1. Create a JUnit Test - TownGraphManager\_STUDENT\_Test. Test all the methods of the TownGraphManager with a different set of data than the TownGraphManagerTest provided for you.
2. Create a JUnit Test – Graph\_STUDENT\_Test. Test all the methods of the Graph with a different set of data than the GraphTest provided for you.
3. Create a Junit Test – Road\_STUDENT\_Test. Test all the methods of your Road class.
4. Create a Junit test – Town\_STUDENT\_Test. Test all the methods of your Town class.