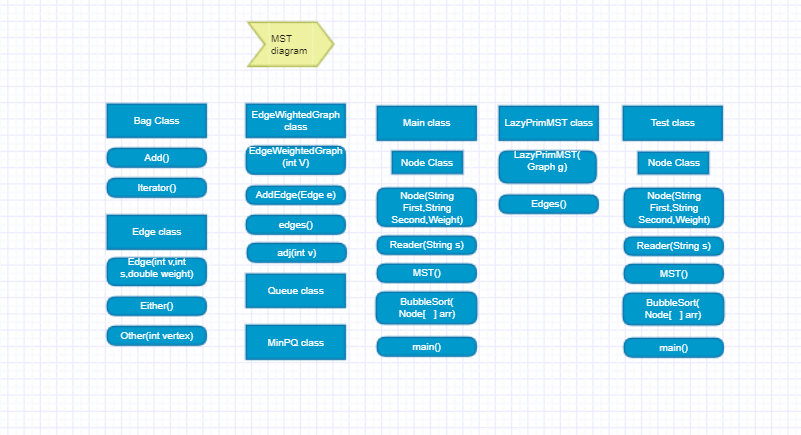
## Problem Statement and Code Design:

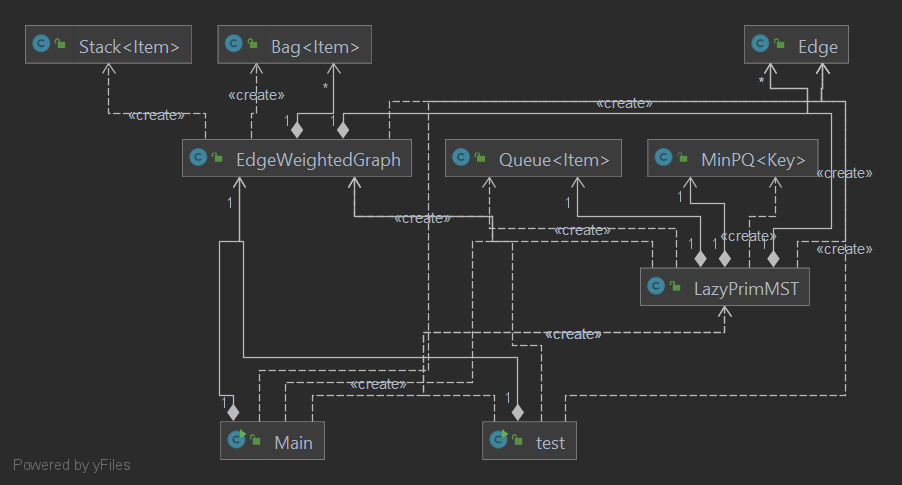
## Part 1:

In the first task the problem is creating a high-speed railroad network in the USA. This railroad will decrease the traveling time between cities. We need to plan a railroad and calculate the time between the cities. For this purpose, we need to use MST algorithms because MST algorithms calculates minimum spanning trees in the graphs. We need to think the cities as edges and length between cities as weight for creating graph. We need to create a layer between the MST algorithm and the program to convert cities to indexes otherwise, we will need to create a complete Edge, graph, and prim algorithm class. After converting an unknown problem to a known problem, we can easily calculate the MST and see the result and calculate times. In the program this layer will be provided by a String array. First, we need to read the cities from the file and keep them in the Cities array. Then we will search for cities in the array and return their indexes to create edges for instance NYC will get index 0 and Chicago will get index 1 and so on. After creating edges will add the edges to the graph. After creating the graph, we are done with Creating the Cities and their connections and the distance between cities. We created the map of the USA. To calculate minimum spanning tree in this graph we need MST algorithm, I choose Prim algorithm. After creating the graph wee need to call the MST algorithm, the algorithm returns the minimum spanning tree and weights as a list. For these results we need another class to be able to compare two strings because if two or more cities connects a city [local root] and if the distances between two or more cities and local root is equal, we need to check their lexicographical order. The node class exactly does that job. It compares the name of the two cities and help to sort them according to their order.

### Structure chart:



### Dependencies chart:



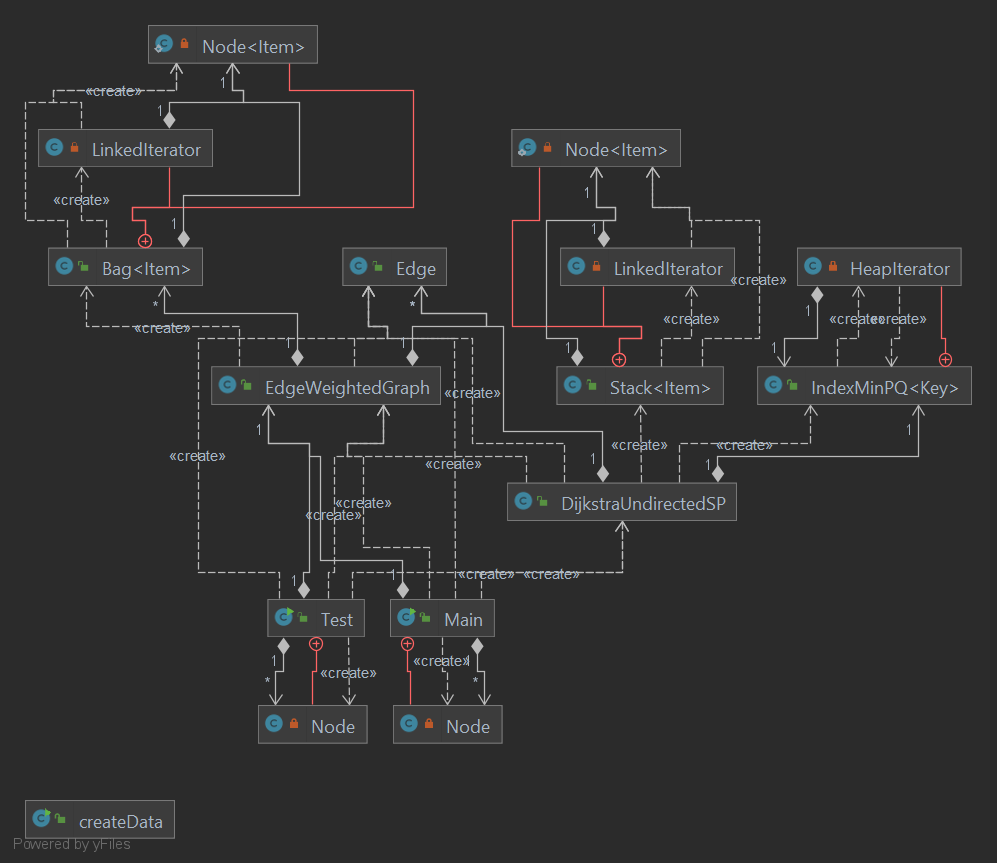
## Part 2:

In this part of the homework, we are going create a plane with x and y coordinates. This plane will include cities as points. This plane will be our map for the program. All cities in the graph will have X and Y values. With the X and Y values we need calculate the distances between the cities in the plane. The calculation will be made with Euclidean distances. The travelers will have to pass only cities during the travel for instance travelers cannot go Trabzon directly from Istanbul, travelers must first come to Ankara then go Trabzon. This is a shortest path problem. The cities will be our edges and the distance between cities will be the edges’ weight. The plane will be our map and the map will be our graph. We will apply shortest path to the graph for requested cities and find and print the shortest path. Firstly, I created a main class to able to run the program. Then I read the file and create a node object for each city. The Node class keeps the Name of the city, X coordinate and Y coordinate and index of the city. In this way it will be easier to move between cities while creating the graph and it will be easier to calculate Euclidean distances. After I created the Node object for every city, I started to create edges, the edges take indexes and distances between cities. After creating the edges, I add edges to the graph. Finally, I wrote a method that takes two strings as a input and returns the shortest path between those cities with printing middle cities as well (If a path exist).

### Structure chart:

### 

### Dependencies chart:



## Implementation and Functionality:

### Part 1:

Bag class: The bag implementation to help the graph and alghoritm. It is very simple,It just has a constructor and a method that satisfy the requirement of adding items.

Add (Item item): this method takes an Item as an input and chain it to old item.

PseudoCode : public void add(Item item) {

Node<Item> oldfirst = first;

first = new Node<Item>();

first.item = item;

first.next = oldfirst;

n++;

}

Edge class: the class that help to create edges for the graph. It takes three argument that need for a weighted edge and creates an Edge object.

Edge (int v, int w, double weight): this is the constructor method of the Edge class.

Either (): It returns left vertex of the Edge (either-----25----other).

Other(int vertex): It returns right vertex of the Edge.

EdgeWeightedGraph class: the graph class that required to create an Edge weighted graph. It uses Bag class as a helper class to keep the edges.

EdgeWEightedGraph (int V) : takes one integer to create an array of bags to keep edges. The V also represent the size of the graph.

PseudoCode : Creation of the array of the bags that keep edges.

adj = (Bag<Edge>[]) new Bag[V];

for (int v = 0; v < V; v++) {

adj[v] = new Bag<Edge>();

AddEdge(Edge e): takes an edge object and adds it to the graph.

LazyPrimMST class: this class calculates the Minimum Spanning Tree of a weighted graph and returns the edges and their weight with respect to their order.

LazyPrimMST(EdgeWeightedGraph G) : the constructor of the class. It takes a graph as an input and call the prim() method to calculate the MST.

Prim(EdgeWeightedGraph G,int s) : calculate the MST according to prim alghoritm. Uses priority queue and stack as helper data structures.

Main class: This class operates all instructors for the program. It reads file, Creates Edges, calls MST algorithm and prints the result of the MST algorithm.

Reader (String path) : It takes a string as input which must be the path of the file that is going be read and parsed. It reads file line by line and create required objects for the program such as Edges, Graph and City array.

Pseudo Code: Creating an Edge and adding it to the graph.

String input2 = myReader.nextLine(); //reading nextline

String[] arr2 = input2.split(" "); //splitting the line in two.

int first = FindCity(arr2[0]); // search for index of the cities

int second = FindCity(arr2[1]);

Edge e = new Edge(first, second, Integer.parseInt(arr2[2])); // edge that represent cities and distances.

graph.addEdge(e); //adding edge to the graph.

Public void MST(): this method creates a LazyPrimMST object and sends the graph to the calculate the MST. After the calculation takes Edges that LazyPrimMS returned and sends it to bubblesort method to control whether there is need to change the edges order according to their lexicographical order or not. after calling bubble sort prints the edges according to their order.

bubbleSort(Node [ ] arr): Classical bubble sorting algorithm designed to check for lexicographical order and sort the array according to lexicographical order in case of need.

Node class: This class helps to calculate the lexicographical order between edges in case of need.

Public Node(String first, String second, int weight): Constructor for node class takes two vertices of the Edge and its weight.

Public int CompareTo(Node o) : compares two Node objects Other Edges in lexicographical order.

### Part 2:

Bag class: The bag implementation to help the graph and alghoritm. It is very simple,It just has a constructor and a method that satisfy the requirement of adding items.

Add (Item item): this method takes an Item as an input and chain it to old item.

PseudoCode : public void add(Item item) {

Node<Item> oldfirst = first;

first = new Node<Item>();

first.item = item;

first.next = oldfirst;

n++;

}

Edge class: the class that help to create edges for the graph. It takes three argument that need for a weighted edge and creates an Edge object.

Edge (int v, int w, double weight): this is the constructor method of the Edge class.

Either (): It returns left vertex of the Edge (either-----25----other).

Other(int vertex): It returns right vertex of the Edge.

EdgeWeightedGraph class: the graph class that required to create an Edge weighted graph. It uses Bag class as a helper class to keep the edges.

EdgeWEightedGraph (int V) : takes one integer to create an array of bags to keep edges. The V also represent the size of the graph.

Pseudo Code: Creation of the array of the bags that keep edges.

adj = (Bag<Edge> []) new Bag[V];

for (int v = 0; v < V; v++) {

adj[v] = new Bag<Edge>();

AddEdge(Edge e): takes an edge object and adds it to the graph.

DijkstraUndirectedSP class: This class represents the implementation of the shortest path alghoritm. It takes a graph and an index as input and calculated the shortest from that edge to another edges. It returns the result as a list with respect to their order.

DijkstraUndirectedSP (EdgeWeightedGraph G, int s): the constructor method of the class. It takes a graph and an Integer to be able to call the alghoritm. It initializes required objects and structures for the algorithm such as IndexMinPQ and edgeTo array. Also it starts picking edges and calls the relax method while needed to choose best option.

DistTo(int v) : return distance between source vertex and target vertex.

Pseudo Code: returning distance between source and target vertex.

public double distTo(int v) {

validateVertex(v);

return distTo[v];

}

pathTo(int v): Returns the path between source vertex and target vertex.

Main class: This class operates all instructors for the program. It reads file, Creates Edges, calls SP algorithm and prints the result of the SP algorithm.

Reader (String path) : It takes a string as input which must be the path of the file that is going be read and parsed. It reads file line by line and create required objects for the program such as Edges, Graph and Node objects.

Pseudo Code : adding a Edge to the graph

data = myReader.nextLine();

String[] arr = data.split(" ");

int first = Integer.parseInt(arr[0]); //edge either

int second = Integer.parseInt(arr[1]); // edge other

g.addEdge(new Edge(first, second ,Euclidean (City[first].xC, City[first].yC, City[second].xC, City[second].yC))); // edge object been created and added to the graph.

Sp(String sour,String tar): The method to calculate the shortest path algorithm. It creates SP object and sends correct indexes to calculate the result. Then it prints result with cities names and distance between source and target cities.

Euclidean (int x1, int Y1, int x2, int y2) : calculates Euclidean distances and returns the result.

Pseudo Code: calculating the Euclidean distance:

return (int) Math.sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));

Node class: A helper to class to help easily calculate distances and return city names. It keeps the coordinates, Name, and index of the city.

## Testing:

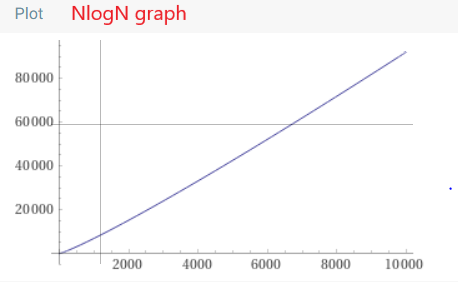
Part 1:

I created a test class that tests the program for 4 different graphs. The class runs 4 different graphs in order and returns corresponding result. The graph sizes are different and measures different inputs for the graph. The program works as expected.

### Part 2:

I created a test class for the program. The test class takes 5 different graphs and calculates the memory and time usage of the graph. I wrote a Create class to create graphs in different sizes. I create 4 graph that sizes as following 1000, 1000,10000,20000 edge per graph. Then I created their connections as well I created the graph. After Creating the graph, I calculated the results of the memory usage and run time to calculate the complexity. After calculating the result of the graph, I created an Excel to be able to illustrate the relation between Time -Memory, Time-Edge number, and Memory -Edge number. I created charts to show the correlation between them and the expected

complexity for the algorithm. The program works nearly as expected as given in the book in huge scale of data. When the data becomes bigger and bigger the program creates thousands of objects therefore, memory and time complexity is a bit higher than expected. To speed up the program we can reduce the object creation for the second part, we can write a better algorithm to find cities from only their name. this will decrease time and memory complexity for the program.



## Final assessment:

The trouble point in this assignment was in the first part. The algorithms that given in the books was not giving the result that I expected. Then I tried lazy prim (also from book), It gave me expected results.

The most challenging part was the first part. The Edges was not following each other, and I had to solve it. I solved by checking whether each parameter is following each other or not then if not following make them follow.

I like the second part of the assignment because it was smooth, no challenge required.

I learned shortest path and MST algorithms better.