# model

**«interface»**
**Comparable<City>**

+int compareTo(City o)

**City**

-name: String
-x:int
-y:int

+City(String,int,int)
+compareTo(City):int
+getName():String
+getX():int
+getY():int
+setName(String):void
+setX(int):void
+setY(int):void
+toString(): String

**TravelingProgram**

+CITIES: int

+TravelingProgram()
+getCities(): City[]
+getCitiesK():City[]
+getGraph(): GraphByLists<City, Highway>
+getGraphK():GraphByLists<City,Highway>
+getPath(Vertex<City>, Vertex<City>): ArrayList<Vertex<City>>
+getPathPrim():int[]
+refillCities():void
+refillCitiesK():void
+refillGraph():void
+refillGraphK():void
+setCities(City[]):void
+setCitiesK(City[]):void
+setGraph(GraphByLists<City, Highway>):void
+setGraphK(GraphByLists<City,Highwat>):void
+stringPath(Vertex<City>, Vertex<City>): String

**«interface»**
**Comparable<Highway>**

+int compareTo( Highway o)

**Highway**

-name: String

+Highway(String)
+compareTo(Highway): int
+getName(): String
+setName(String):void
+toString(): String

cities  32

graph

# Interface

**JFrame**

**MainWindow**

+MainWindow()
+getTravel():TravelinProgram()
+setTravel(TravelinProgram):void
+main(String[]):void

travel

initial

**InitialPanel**

+BACKGROUND: Image

+InitialPanel(MainWindow)
+getInfo(): InfoPanel
+getMap():MapPanel
+getWindow():MainWindow
+paintComponent(Graphics):void
+setInfo(InfoPanel):void
+setMap(MapPanel):void
+setWindow(MainWindow):void

info

initial

**JPanel**

**InfoPanel**

+ROUTE: String
+TRAVEL: String
-from: JLabel
-to:JLabel
-comboFrom:JComboBox
-comboTo:JComboBox
-distance:JLabel
-txtDistance:JLabel
-kms:JLabel
-route:JButton
-travel:JButton
-cities: String[32]

+InfoPanel(InitialPanel)
+actionPerformed(ActionEvent):void
+combox():void
+components():void
+getComboFrom():JComboBox
+getComboTo():JComboBox
+getTxtDistance():JLabel
+paintComponent(Graphics):void
+setComboFrom(JComboBox):void
+setComboTo(JComboBox):void
+setMap(MapPanel):void
+setTxtDistance(JLabel):void

**MapPanel**

+BRACKGROUND:Image

+MapPanel(InitialPanel)
+paintComponent(Graphics):void
+tooltip():void

map

initial

# dataStructures

**«interface»**
**Comparable<Node<T>>**

+int compareTo(Node<T> n)

**«interface»**
**Comparable<T>**

+int compareTo(T o)

**«interface»**
**Comparable<E>**

+int compareTo(E o)

**«interface»**
**IGraph**

+addEdge(T,T,boolean,double,E):void
+removeEdge(T,T,boolean,double):void
+addVertex(T):void
+removeVertex(T):void
+isAdjacent(T,T):void
+getNumVertex():void

**Node**

-cost: Double

+Node(Vertex<T>,Double)
+compareTo(Node<T>):int
+getCost():Double
+getVertex():Vertex<T>
+setCost(Double):void
+setVertex(Vertex<T>):void

**Vertex**

-value:T

+Vertex(T)
+getValue():T
+setValue(T):void
+compareTo(T):void
+toString():String

from

**Edge**

-value:E
-cost: double
-directed: boolean

+Edge(boolean,double,E)
+getValue():E
+getName():String
+isDirected():boolean
+setDirected(boolean):void
+getCost():double
+setCost(double):void
+compareTo(E):int
+toString():String

0..*

destination

**GraphByLists**

-adjList:Map

+GraphByLists()
+addEdge(T,T,boolean,double,E):void
+removeEdge(T,T,boolean,double):void
+addVertex(T):void
+removeVertex(T):void
+isAdjacent(T,T):boolean
+getEdgesOfVertex(T):ArrayList<Edge<E>>
+numEdgesOfVertex(T):int
+edgesBetween(T,T):ArrayList<Edge<E>>
+graphToString():String
+getNumvertex():int
+getAdjMatrix():ArrayList<Edge<E>>
+setAdjMatrix(ArrayList<Edge<E>>):void
+getvertices():ArrayList<Vertex<T>>
+setVertices(ArrayList<Vertex<T>>):void

**GraphByMatrix**

-adjMatrix:ArrayList<Edge<E>>
-vertices: ArrayList<Vertex<T>>
-numVertex:int

+GraphByMatrix(int)
+initMatrix():void
+getIndexVertex(T):int
+addEdge(T,T,boolean,double,E):void
+removeEdge(T,T,boolean,double):void
+addVertex(T):void
+removeVertex(T):void
+isAdjacent(T,T):boolean
+getEdgesOfVertex(T):ArrayList<Edge<E>>

**MethodsGraphs**

+MethodsGraphs()
+BFS(GraphByLists<T,E>,Vertex<T>):ArrayList<Vertex<T>>
-BFS(GraphByLists<T,E>, Vertex<T>, boolean[], Queue<Vertex<T>>, ArrayList<Vertex<T>>): ArrayList<Vertex<T>>
+BFS(GraphByMatrix<T,E>,Vertex<T>):ArrayList<Vertex<T>>
-BFS(GraphByLists<T,E>,Vertex<T>,boolean[], Queue<Vertex<T>>, ArrayList<Vertex<T>>): ArrayList<Vertex<T>>
+DFS(GraphByLists<T,E>,Vertex<T>):ArrayList<Vertex<T>>
-DFS(GraphByLists<T,E>,Vertex<T>,boolean[], Stack<Vertex<T>>, ArrayList<Vertex<T>>): ArrayList<Vertex<T>>
+DFS(GraphByMatrix<T,E>,Vertex<T>):ArrayList<Vertex<T>>
-DFS(GraphByLists<T,E>, Vertex<T>,boolean[], Stack<Vertex<T>>, ArrayList<Vertex<T>>): ArrayList<Vertex<T>>
+Dijkstra(GraphByLists<T,E>, Vertex<T>):double[]
-Dijkstra(GraphByLists<T,E>, Vertex<T>,boolean[],double[],ArrayList<Double>,PriorityQueue<Node<T>>):double[]
+Dijkstra(GraphByMatrix<T,E>, Vertex<T>):double[]
-Dijkstra(GraphByMatrix<T,E>, Vertex<T>,boolean[],double[],ArrayList<Double>, PriorityQueue<Node<T>>):double[]
+floydWarshall(GraphByLists<T,E>):double[][]
-floydWarshall(GraphByLists<T,E>,double[][]):double[][]
+floydWarshall(GraphByMatrix<T,E>):double[][]
-floydWarshall(GraphByMatrix<T,E>,double[][]):double[][]
+kruskal(GraphByLists<T,E>):double
-kruskal(GraphByLists<T,E>, int[], ArrayList<Edge<E>>):double
+kruskal(GraphByMatrix<T,E>):double
-kruskal(GraphByMatrix<T,E>, int[], ArrayList<Edge<E>>):double
+prim(GraphByLists<T,E>):double
-prim(GraphByLists<T,E>, Vertex<T>, ArrayList<Vertex<T>>, boolean[], double[], int[], int, int double, int):double
+prim(GraphByMatrix<T,E>, Vertex<T>):double
-prim(GraphByMatrix<T,E>, Vertex<T>, ArrayList<Vertex<T>>, boolean[], double[], int[], int, int double, int):double
+dijkstra(GraphByLists<T,E>, Vertex<T>): Path<T,E>
+dijkstra(GraphByMatrix<T,E>, Vertex<T>):Path<T,E>
-dijkstra(GraphByLists<T,E>, Vertex<T>, ArrayList<Vertex<T>>,boolean[],double[], int[], Path<T,E>, int, int): Path<T,E>
-dijkstra(GraphByMatrix<T,E>, Vertex<T>, ArrayList<Vertex<T>>,boolean[],double[], int[], Path<T,E>, int, int): Path<T,E>
+primP(GraphByLists<T,E>, Vertex<T>):Path<T,E>
-primP(GraphByLists<T,E>, Vertex<T>, ArrayList<Vertex<T>>, boolean[], double[], int[], int, int double, int, Path<T,E>):Path<T,E>
+primP(GraphByMatrix<T,E>, Vertex<T>):Path<T,E>
-primP(GraphByMatrix<T,E>, Vertex<T>, ArrayList<Vertex<T>>, boolean[], double[], int[], int, int double, int, Path<T,E>): Path<T,E>

# Tests

**TestCase**

**GraphByListsTest**

-graph:GraphByLists<String,String>

+stageOne():void
+stageThree():void
+stageTwo():void
+testAddEdge():void
+testAddVertex():void
+testGetEdgesOfVertex():void
+testIsAdjacent():void

**GraphByMatrixTest**

-graph:GraphByMatrix<String,String>

+stageOne():void
+stageThree():void
+stageTwo():void
+testAddEdge():void
+testAddVertex():void
+testGetEdges():void
+testInitMatrix():void
+testIsAdjacent():void
+testRemoveEdge():void

**MethodsGraphsTest**

-graphList: GraphByLists<Integer,Integer>
-graphMatrix: GraphByMatrix<Integer,Integer>
-mG:MethodsGraphs<Integer,Integer>

+stageOne():void
+stageTwo():void
+stageThree():void
+stageFour():void
+stageFive():void
+stageSix():void
+stageSeven():void
+stageEight():void
+testBFS():void
+testBFSMatrix():void
+testDFS():void
+testDFSMatrix():void
+testDijkstra():void
+testDijkstraMatrix():void
+testFloydWarshall():void
+testFloydWarshallMatrix():void
+testBFSTwo():void
+testBFSMatrixTwo():void
+testDFSTwo():void
+testDFSMatrixTwo():void
+testDijkstraTwo():void
+testDijkstraMatrixTwo():void
+testFloydWarshallTwo():void
+testFloydWarshallMatrixTwo():void
+testPrim()
+testKruskal()