<<Java Class>>
Graph
project

adjacents: Hashtable<String,List<String>>

Graph()
getEdges():List<Edge>
getAdjacents():Hashtable<String,List<String>>
addVertex(String):void
addEdge(String,String,int,int):void
nullCheck():void

<<Java Class>>
Edge
project

startCity: String
endCity: String
miles: int
minutes: int

Edge(String,String,int,int)
getStartCity():String
setStartCity(String):void
getEndCity():String
setEndCity(String):void
getMiles():int
setMiles(int):void
getminutes():int
setminutes(int):void

~edges
0..*

~graph 0..1

<<Java Class>>
RoadTrip
project

alHash: Hashtable<String,String>
check: Hashtable<String,Boolean>
temp: Hashtable<String,String>
distance: Hashtable<String,Integer>
cities: ArrayList<String>
attractCheck: ArrayList<String>
miles: int
minutes: int

RoadTrip()
parseRoads(String):void
parseAttraction(String):void
milesEdge(String,String):int
validAttraction(String):boolean
minutesEdge(String,String):int
toString(List<String>):void
minVertex():String
route(String,String,List<String>):List<String>

<<Java Class>>
Driver
project

Driver()
main(String[]):void

The Graph Implementation I chose is both directed and Weighted, considering the fact that this project involves having to use the weight to determine the amount of distance and time that the route will have, and direction as to make sure that the order of the nodes is in place.

The Route() Function in the RoadTrip Class uses Dijkstra's algorithm to take in to account the amount of minutes and time each route takes and finds the shortest possible path, while also taking in to account various attractions that the user would want to visit, in order to get find the most optimum route. I chose Dijkstra's specifically because of its running time of O(ElogV). In addition, I also chose to use Dijkstra's over Prim's because in terms of efficiency, Dijkstra's is superior when it comes to graphs with direction AND weight.