Explanation of Code

Class- Main Runner: Read the input list and act as the user input interface for the entire operation

Method: *Main(String[])*-
Read the input data and initiate the program

Class- Location: Translates user inputed locations into a format readable by the program

Class- Locations: Reads the imputed locations of users as found by Location, determines who the drivers and passengers are, and houses the main formula for calculations based off of the Haversine formula

Method: *addLoc*-
It reads the locations on the user inputed list, determines who are the drivers and who are the passengers, and identifies where the final location will be.

Method: *calculateDistanceMatrix*-
Uses the Haversine formula to find the distance in between all the locations. Used for all distance calculations

Method: *readLocString*-
Assigns values to each aspect of the user inputted values

Class- Utils: It calculates the base distance from passenger to passenger, and illustrates the route in which passengers are picked up for display in the output.

Method: *calculateTotalDistance*-
Use the Haversine formula to calculate the distance each driver needs to travel

Method: *getBaseTravelDistance*-
Display the base travel distance to compare against the optimized travel distance

Method: *printPath*-
Display the resulting order in which passengers are picked up

Class- GRASP Heuristic: Creates initial path based off of the next closest passenger, then adds it as the best route (so far), afterwards it improves the path by doing random search of swapping and shuffling.

Method: *addToBestRoutes*-
Adds the newly created path as the best result (so far). If a better route is created it then replaces the old one

Method: *createClosestDistPath*-
Generate an initial path by having the driver travel from closest passenger to closest passenger

Method: *search*-
Improve the path by doing random search (swapping and shuffling

Class- Visualization: Uses longitude and latitude to print out a map of all passengers, print out who they are clustered too, and finally print out the route each driver will take to bring their passengers to the final destination as well as statistics such as overall best distance and the base distance

Method:

Class- Clustering: Assign each passenger to the closest driver

Method: *Assign to Closest Driver*-

- Clusters passengers by assigning them to the closest available driver using their latitudes and longitudes

Haversine Formula

The haversine formula is an equation important in navigation, giving great-circle distances between two points on a sphere from their longitudes and latitudes. It is a special case of a more general formula in spherical trigonometry, the law of haversines, relating the sides and angles of spherical triangle

The haversine formula is used to calculate the distance between two points on the Earth's surface specified in longitude and latitude. It is a special case of a more general formula in spherical trigonometry, the law of haversines, relating the sides and angles of spherical "triangles".

$$d = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right)$$

d is the distance between two points with longitude and latitude (ψ,φ) and *r* is the radius of the Earth.

```
public class HaversineDistance {

public static double haversineDistance(double lat1,double lat2,double
lon1,double lon2) {
        double deltaLat = Math.toRadians(lat1 - lat2);
        double deltaLong = Math.toRadians(lon1 - lon2);
        double lat1R = Math.toRadians(lat1);
        double lat2R = Math.toRadians(lat2);
```

```
        double a = (Math.sin(deltaLat/2.0) * Math.sin(deltaLat/2.0)) +
                (Math.sin(deltaLong/2.0) * Math.sin(deltaLong/2.0) *
                   Math.cos(lat1R) * Math.cos(lat2R));
        double c = 2.0*Math.atan2(Math.sqrt(a), Math.sqrt(1.0 - a));
        double d = 3959.0*c;
        return d;
      }

    public static void main(String[] args) {

System.out.println(haversineDistance(21.7679,40.4230,78.8718,98.7372));
      }

}
```

URL: http://reflectvicky.blogspot.com/2013/04/calculate-haversine-distance-in-java.html

Future Work and Clustering

1. **Develop a fully automated system that uses Google Maps or GPS system in order to create more accurate results**
    a. Requires only that participants put in their information and the program will solve it and email results automatically
        i. Maybe an app?
    b. Google Maps is easier with conversion to Python

2. **Our New Method of Clustering:**
    a. In order to make sure our clustering is accurate, we clustered passengers along the path from each driver to the final destination based off of closest proximity to the path

i. To do this, we needed to use a formula that found the distance from each point (passenger) to the line (direct path from drivers to final destination

b. **Formula for finding the distance from a point to a line:**

(Point is: (X_0, Y_0) , line is: $Ax + By = C$ or $y=\frac{(-A)x+C}{(B)}$):

d *(distance)* = $\frac{|A(X_0)+ B(Y_0) - C|}{\sqrt{(A^2 + B^2)}}$

Data

Case #	Base Case	ETRAM (travel distance)	Improvement %	Saved Mileage	Saved Gas	Saved Money
1	25	13	48 %	12	.6 gal	$1.2
2	65	26	60 %	39	~2 gal	$4
3	81	26	68 %	55	2.75 gal	$5.5
4	54	16	70 %	38	~2 gal	$4
5	156	52	~67 %	104	~5 gal	$10
6	117	34	~71 %	83	~4 gal	$8
7	96	34	65 %	62	~3 gal	$6
8	74	47	36 %	27	1.35 gal	$2.7
9	128	54	58 %	74	3.7 gal	$7.4
10	273	55	~80 %	218	~11 gal	$22
11	205	72	~65 %	133	6.65 gal	$14
12	59	34	~65 %	34	6.65 gal	$14
13	257	80	~69 %	177	~9 gal	~$18
14	117	44	~62 %	73	3.65 gal	$7.3
15	32	23	~28 %	9	0.45 gal	$1
16	185	45	~76%	140	7 gal	$14
17	87	27	~69 %	60	3 gal	$6
18	74	16	~78 %	58	~3 gal	$6
19	72	35	~51%	37	~2 gal	$4
20	152	62	~59 %	90	4.5 gal	$9