Extenions to CITS1001 Project 1

Brandon Barker

2019

Added More Outputs

Having more useful analytics is never a bad thing. The displacement and average speed over the trip were added as outputs to the Trip class. The displacement was calculated using the Haversine formula. The Haversine formula takes the initial and final coordinates (in latitude and longitude) and uses the radius of the earth to find the distance between them

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

$$\lambda = \text{longitude}$$

$$\phi = \text{latitude}$$

The Haversine equation's implementation in the Trip class to return the displacement trip can be seen below

```
public int tripDisplacement()
    {
        int earthRadius = 6372797;

        double latChange=(getFinish().getLat()-getStart().getLat())/2.0;
        double lonChange=(getFinish().getLon()-getStart().getLon())/2.0;

        double cos1 = Math.cos( getStart().getLat()*Math.PI/180);
        double cos2 = Math.cos( getFinish().getLat()*Math.PI/180);

        double sin1 = Math.sin(latChange*Math.PI/180);
        double sin2 = Math.sin(lonChange*Math.PI/180);

        double intriumValue = sin1*sin1 + cos1*cos2 * sin2*sin2;

        return (int)(2.0*earthRadius*Math.asin(Math.sqrt(intriumValue)));
    }
```

```
void square(double *x) {
  *x = *x * *x;
}
```

Test the `square()` function:

```
.C('square', 9)
```

```
## [[1]]
## [1] 81
```

```
.C('square', 123)
```

```
## [[1]]
## [1] 15129
```

Over larger distances the approximation used in the project overview becomes inaccurate. This is fine for calculating the total distance travelled, as the distance between successive pings is small. Thus, to allow an accurate calculation for the total displacement from the initial ping to the final ping, the Haversine formula was used. The average speed was much simpler to calculate

```
public double averageSpeed()
    {
        return (double)tripLength()/(double)tripDuration();
    }
```

This simply divides the total distance by the total duration of the trip. Giving the average speed over the entire trip. Testing both of these functions was quite easy, as both of these outputs can be easily verified by evaluating the correct values manually and checking them against the values the Trip class outputs.

A makeTable() method was added

Although there is a toString() method, returning data in a string is very limiting to the way it can be presented. A table is a universal and simple way of displaying data. It allows for easy reading and extraction of the data. The makeTable() method uses some classes from the java.awt and java.swing libraries. The table displays values in their raw form, as well as the more convenient form used in the toString() method. This allows for easy data extraction, as there is no need to remove the formatting used to display the in the string and it allows for easy and informative reading of the data

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;

    public void makeTable()
    {
        JFrame frame = new JFrame();

        Object rowData[][] = {
            {"Results", "", ""},
            {"Trip Duration (s)", tripDuration(),tripDuration()/86400+" Days "
                +(tripDuration()%86400/3600)+" Hours "
                +(tripDuration()%3600/60)+" Minutes "
                +(tripDuration()%60)+" Seconds"},
            {"Trip Length (m)",tripLength(),
                String.format("%.4f km",(float)tripLength()/1000)},
            {"Trip Displacement (m)", tripDisplacement(),
                String.format("%.4f km",(float)tripDisplacement()/1000)},
            {"Average Speed (m/s)", String.format("%.4f",averageSpeed())
                ,String.format("%.4f",averageSpeed()*3.6)+" km/h"},
            {"Max Latitude (\u00b0)", NEbound().getLat(), ""},
            {"Max Longitude (\u00b0)",NEbound().getLon(), ""},
            {"Min Latitude (\u00b0)", SWbound().getLat(), ""},
            {"Min Longitude (\u00b0)",SWbound().getLon(), ""},
            {"", "", ""},
            {"", "", ""},
            {"Inputs", "Initial", "Final"},
            {"Latitude (\u00b0)", getStart().getLat(), getFinish().getLat()},
            {"Longitude (\u00b0)", getStart().getLon(), getFinish().getLat()},
            {"Unix Time (s)", getStart().getTime(), getFinish().getTime()},
        };
        Object columnNames[] = { "", "", "" };
        JTable table = new JTable(rowData, columnNames);
        JScrollPane scrollPane = new JScrollPane(table);
        frame.add(scrollPane, BorderLayout.CENTER);
        frame.setSize(800, 400);
        frame.setVisible(true);
    }
```

Testing this function was also quite easy. It consisted of checking if the values matched the output from their respective methods and if the formatting was adequate.