Lab 7 A Distance Calculator[1]

Due: Tuesday, October 25, 2011 at 23:00

"We will go to the moon. We will go to the moon and do other things, not because they are easy but because they are hard." - John F. Kennedy, Jr.

1. Outcomes

1. Translate a UML class Diagram into Java Source Code
2. Translate UML sequence diagrams into source code
3. Develop software using multiple interacting classes
4. Apply formatted printing statements to outputs
5. Range check user input values

2. Introduction

Before one starts out on a journey, it is always helpful to have an idea of how long the journey is going to take. If one is driving between Milwaukee and Chicago, for example, it is good to know that the distance is around 90 miles. That means you won't get there is 15 minutes (unless you really have a lead foot, in which case it might be good to refer back a couple of labs to the insurance calculator), but it probably won't be an all day trip either (unless the construction on I-94 is really bad).

Since the earth can be approximated as a sphere, we can approximate the distance between two cities as the distance around a sphere. For most distances, this is a reasonable approximation. We do this using the Spherical law of cosines. This law states that the distance between two points can be calculated as

$$d = \arccos(\sin(\lambda 1) \times \sin(\lambda 2) + \cos(\lambda 1) \times \cos(\lambda 2) \times \cos(\varphi 1 - \varphi 2)) \times R \quad (1)$$

where

- $\lambda 1$, $\lambda 2$ represent the latitude of locations 1 and 2 in radians,
- $\varphi 1$, $\varphi 2$ represent the longitudes of locations 1 and 2 in radians, and
- R represents the radius of the earth.

3. Lab Details

For this lab, you will be constructing a program that calculates the distance between two cities using the Spherical law of Cosines. You will prompt the user to enter the name of the first city, the abbreviation for the state, and the latitude and longitude coordinates of the city in degrees. Once this is created, the appropriate instances of the EarthLocation and City classes will be instantiated. You will then prompt the user to enter data for the second city before calculating the distance between the two cities.

In order to facilitate the completion of this lab, you are being provided with three things. First, you are being provided with a class diagram for the classes you will need to develop. This includes the definition of the methods for each of the four classes within the distance calculator, namely the EarthLocation, the City, the DistanceCalculator, and the DistanceCalculatorDriver.

[1] Or GPS in Java 101…

Second, you are being provided with a sequence diagram showing the interactions between the classes. This diagram shows the instantiation of new objects using the constructors, the messages passed between objects, and the instances of classes involved with this program.

Third, you are being provided with a partially complete main program. The main program in the driver is partially complete. You will need to develop the rest of the source code to match what is present on the sequence diagram.

4. Sample Output

```
Enter the first cities name.
Milwaukee
Enter the first cities state abbreviation.
WI
Enter the latitude of the first city.
43.052222
Enter the longitude of the first city.
-87.955833
Enter the second cities name.
Chicago
Enter the second cities state abbreviation.
IL
Enter the latitude of the second city.
41.881944
Enter the longitude of the second city.
-87.627778
This distance between Milwaukee, WI and Chicago, IL is 82.52 miles.
================================================================================
Enter the first cities name.
Milwaukee
Enter the first cities state abbreviation.
Wisconsin
Enter the latitude of the first city.
43.052222
Enter the longitude of the first city.
-87.955833
Enter the second cities name.
Los Angeles
Enter the second cities state abbreviation.
California
Enter the latitude of the second city.
34.05
Enter the longitude of the second city.
-118.25
This distance between Milwaukee, Unknown and Los Angeles, Unknown is 1738.44 miles.
```

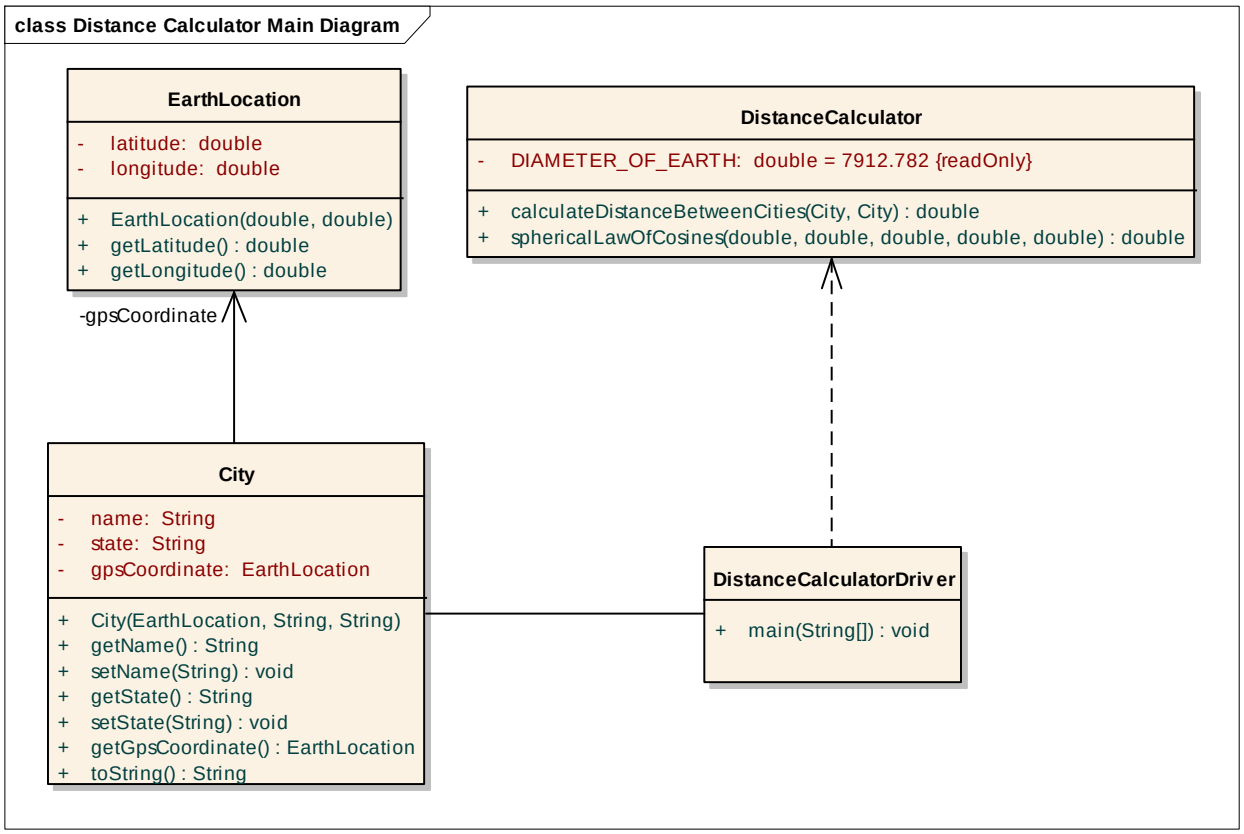

Figure 1: UML diagram showing design of baseline source code.

4.1. UML Diagram

Figure 1 is a UML class diagram for this program. While there are many annotations that you have not learned about as of yet, the important part that you have been taught is how to translate a class representation into source code.

5. Mistakes which I think you will make

1. Failure to declare the distance calculator methods and variables static. The underlined text on the UML class diagram indicates these are static variables and methods. If you fail to declare these properly as statics, things may not work properly.
2. Failure to call the static methods properly. Calls to the static methods in the DistanceCalculator need to be made in a static fashion. Just as you would call Math.sqrt(15), these methods need to be called in the same fashion.
3. Failure to convert degrees to radians. The Java math library functions for sine, cosine, arctan, etc. assume parameters are given in Radians. The GPS coordinates are given in degrees. If you do not convert them properly, you may have erroneous results.
4. Forgetting to read the Javadoc. Some of the parameters for the City require input validation. If you do not do this, the output probably will not match that provided in the examples.
5. Missing parenthesis in the law of cosines calculation. If you miss a set of parenthesis in the Spherical Law of Cosine calculation, you may have erroneous results.

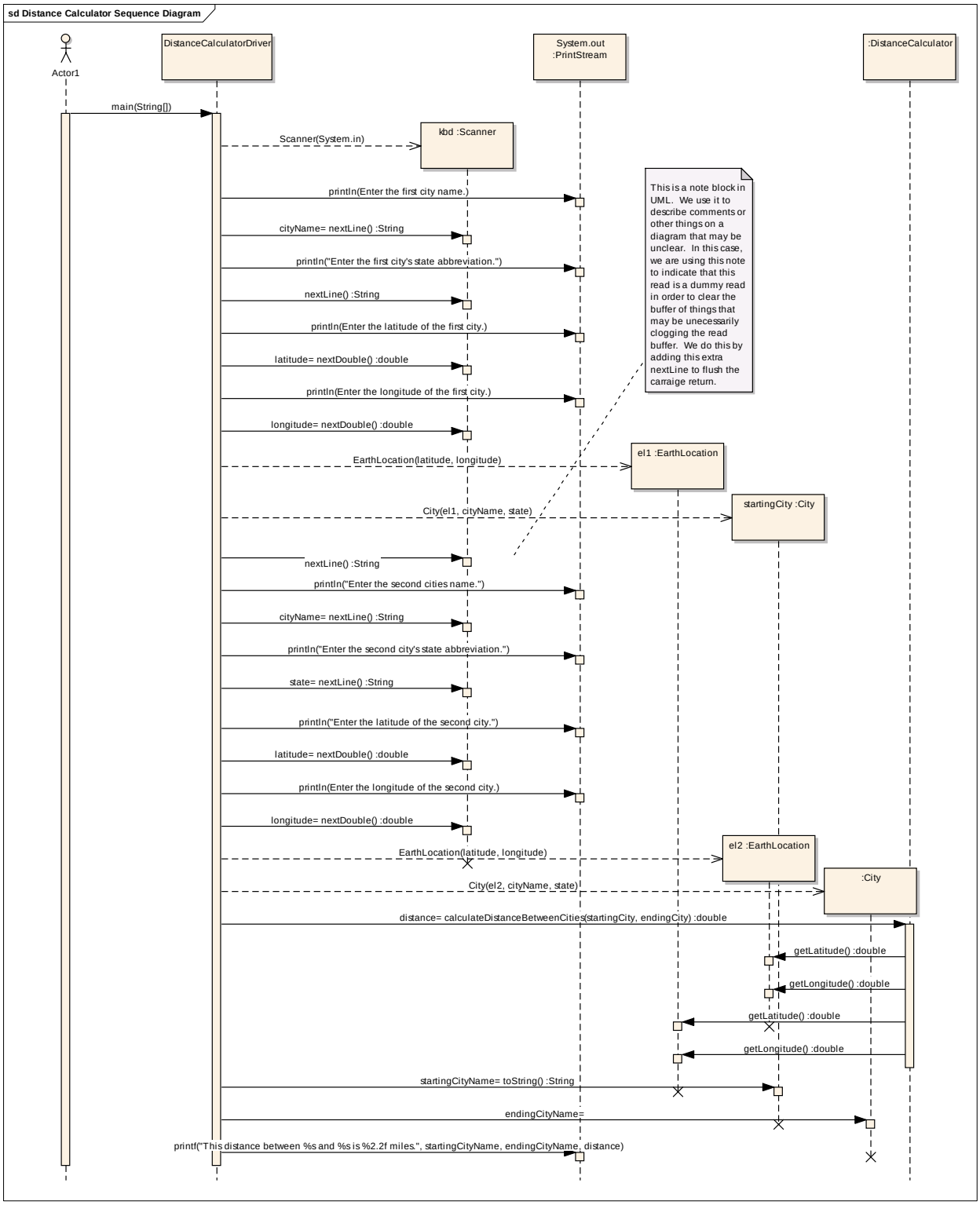

Figure 2: UML sequence diagram showing the interaction between classes.

6. Deliverables

1. Lab report, submitted in pdf format through the course website. This report should include:
   (a) Name, date, title, and course information.
   (b) A short description of what you did in this lab.
   (c) Samples of the program executing showing the output written to the console. This output should show correct operation for all test cases given in this document.
   (d) A short description of what went wrong and what went right during the lab.
   (e) A description of what you learned from this lab.
   (f) The Java source code you wrote as an appendix.
2. Your source code for all java classes.

All deliverables are to be uploaded through the course website.

7. JavaDoc for the project

7.1. Class City

```
java.lang.Object
  └ City
```

```
public class City
extends java.lang.Object
```

Constructor Summary

City(EarthLocation gpsCoordinate, java.lang.String name, java.lang.String state) This is the constructor for the City class.	

Method Summary

EarthLocation	**getGpsCoordinate**() This method will return the GPS coordinate for the given city.
java.lang.String	**getName**() Obtain the name of the city.
java.lang.String	**getState**() Obtain the name of the state.
void	**setName**(java.lang.String name) Set the name of the city.
void	**setState**(java.lang.String state) Set the name of the state.
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

7.1.1. City

```
public City(EarthLocation gpsCoordinate,
            java.lang.String name,
            java.lang.String state)
```

This is the constructor for the City class. It will take a GPS coordinate, a name of the city, and a state.

Parameters:

`gpsCoordinate` - This is an initialized EarthLocation class which stores information about the location of the city.

`name` - This is the name of the city. It can be any length greater than 0. If the length is 0, then it is initialized to "UNKNOWN".

`state` - This is the name of the state. It must be a two letter abbreviation. If more than 2 characters are provided in the string, it will be initialized to "UK" for unknown.

Method Detail

7.1.2. getName

`public java.lang.String` **`getName`**`()`

Obtain the name of the city.

Returns:

the name of the city

7.1.3. setName

`public void` **`setName`**`(java.lang.String name)`

Set the name of the city.

Parameters:

`name` - the name to set. Must be greater than 0 characters in length. Otherwise, unknown will be used as the city name.

7.1.4. getState

`public java.lang.String` **`getState`**`()`

Obtain the name of the state.

Returns:

the state for the city. This will be a two letter abbreviation

---

7.1.5. setState

`public void` **`setState`**`(java.lang.String state)`

Set the name of the state.

Parameters:

`state` - the state to set Must be a 2 letter abbreviation. If not equal to 2 characters in length, set to unknown.

7.1.6. getGpsCoordinate

`public EarthLocation` **`getGpsCoordinate`**`()`

This method will return the GPS coordinate for the given city.

Returns:

the gpsCoordinate This is the GPS coordinate for the location.

7.1.7. toString

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `java.lang.Object`

7.2. Class DistanceCalculator

```
java.lang.Object
  └ DistanceCalculator
```

```
public class DistanceCalculator
extends java.lang.Object
```

This class will calculate the distance between two locations on earth. It uses the law of cosines to do this.

Author:
schilling

Method Summary

`static double`	**`calculateDistanceBetweenCities`**`(City location1, City location2)` This method will calculate the distance between two cities using the spherical law of cosines.
`static double`	**`sphericalLawOfCosines`**`(double radLat1, double radLon1, double radLat2, double radLon2, double radius)` This method will calculate the distance between two objects using the spherical law of cosines.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

7.2.1. calculateDistanceBetweenCities

```
public static double calculateDistanceBetweenCities(City location1,
                                                     City location2)
```

This method will calculate the distance between two cities using the spherical law of cosines.

Parameters:

`location1` - Location 1 is the first city.

`location2` - Location 2 is the second city.

Returns:

The return value will be the distance in miles between two cities.

7.2.2. sphericalLawOfCosines

```
public static double sphericalLawOfCosines(double radLat1,
                                           double radLon1,
                                           double radLat2,
                                           double radLon2,
                                           double radius)
```

This method will calculate the distance between two objects using the spherical law of cosines.

Parameters:
`radLat1` - This is the latitude of position 1.
`radLon1` - This si the longitude of position 2.
`radLat2` - This is the latitude of position 1.
`radLon2` - This is the longitude of position 2.
`radius` - This is the radius of the sphere.
Returns:
The distance between the two locations across the face of the sphere will be returned.

7.3. Class EarthLocation

```
java.lang.Object
  └ EarthLocation
```

```
public class EarthLocation
extends java.lang.Object
```

Author:

schilling This class will store the information about a location on earth. The location is described by a latitude and longitude.

Constructor Summary

EarthLocation(double latitude, double longitude)	

Method Summary

double	**getLatitude**() This method will obtain the latitude of the location.
double	**getLongitude**() The longitude of the location will be returned.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

7.3.1. EarthLocation

```
public EarthLocation(double latitude,
                     double longitude)
```

Parameters:

`latitude` - This is the latitude on earth. Can range between -90 and 90. If ottside of this range, a value of 0 is used.

`longitude` - This is the longitude. Can range between -180 and 180. If outside of this range, a value of 0 is used.

Method Detail

7.3.2. getLatitude

```
public double getLatitude()
```

This method will obtain the latitude of the location.

Returns:

The latitude of the coordinate is returned. This value is in degrees.

7.3.3. getLongitude

`public double` **`getLongitude`**`()`

The longitude of the location will be returned.

Returns:

The longitude of the coordinate is returned. This value is in degrees.

7.4. Class DistanceCalculatorDriver

```
java.lang.Object
  └ DistanceCalculatorDriver
```

```
public class DistanceCalculatorDriver
extends java.lang.Object
```

Constructor Summary

DistanceCalculatorDriver()

Method Summary

static void	**main**(java.lang.String[] args)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

7.4.1. DistanceCalculatorDriver

```
public DistanceCalculatorDriver()
```

Method Detail

7.4.2. main

```
public static void main(java.lang.String[] args)
```