CMSC 113 - Computer Science 1
Fall 2020
Homework Assignment #1

NOTE: Before starting this Assignment make sure you have (1) Read Section 1.2 from S&W, and (2) Started and completed Lab#2.

Overview

Websites such as Google Maps, Mapquest, Bing Maps, and others are used by millions of people each day to get directions from one place to another and to get an idea of how long their travel will take.

In this assignment, you will implement three small programs that perform calculations related to time and distance, similar to what would be used in a map/directions application.

For each program, first read the problem, then devise a solution, if not already provided. Clearly identify all the inputs and outputs. Write down the steps and calculations that will need to be performed. Code the steps into a Java program. Create, compile and run the program and test it on several inputs. Double-check the program's answers with the ones we have provided to confirm that the results are correct.

Learning Outcomes

In completing this assignment, you will learn to:

- Write a Java program using a text editor
- Compile a Java program using the "javac-introcs" command
- Execute a Java program using the "java-introcs" command
- Read runtime arguments into a program's main method
- Declare, initialize, and update variables in a Java program
- Perform mathematical operations in a Java program
- Write to the terminal in a Java program

Part 1: Converting minutes into hours and minutes:

In determining how long a journey will take, a map/directions application such as Google Maps may calculate the time only in minutes, but then display it to the user in hours and minutes so that it's easier to understand.

Write a Java program called **Hours** that takes as input a number (representing minutes) and outputs the equivalent in hours and minutes. Three sample runs are shown below:

```
[xena@codewarrior ~]$ java-introcs Hours 67

1 hours and 7 minutes


[xena@codewarrior ~]$ java-introcs Hours 453

7 hours and 33 minutes

[xena@codewarrior ~]$ java-introcs Hours 360

6 hours and 0 minutes
```

You may assume that the input to the program is a positive integer, i.e. a whole number greater than 0. You do not have to worry about the case in which the input is missing, a negative number, a fraction, something that's not a number, etc.

After completing Part 1, take a break!

Part 2: Estimated Time of Arrival

Once a map/directions application has determined a route from a starting point to the destination, it can tell the user the approximate time that they'll arrive, based on knowing the current time and how long it will take to travel.

Write a Java program called **ArrivalTime** that takes as input the current time, the distance to travel, and the expected average speed, and prints the estimated arrival time. The four runtime arguments should be as follows, in this order:

- *h* = the hour portion of the current time, ranging from 0-23
- *m* = the minutes portion the current time, ranging from 0-59
- *d* = the distance to be traveled, in miles
- *s* = the expected average speed, in miles per hour

Here are a few sample runs of the program:

```
[xena@codewarrior ~]$ java-introcs ArrivalTime 11 10 15 60

Arrival Time is 11:25

[xena@codewarrior ~]$ java-introcs ArrivalTime 15 50 125 50

Arrival Time is 18:20

[xena@codewarrior ~]$ java-introcs ArrivalTime 22 15 110 60

Arrival Time is 0:5
```

Note that in the last example, the time would ordinarily be displayed as "00:05" but it is okay to omit the leading "0" for the purposes of this assignment.

After reading the four runtime arguments, your program should be implemented according to these steps:

1. Calculate the travel time *t* = *d* / *s* * 60; note that you need to multiply by 60 since *s* is miles per hour but we want *t* in minutes. *Hint!* Think about which datatypes you should use to store *t*, *d*, and *s*, keeping in mind what you have learned about integer division.
2. Update the minutes portion of the arrival time: *m* = *m* + *t*
3. Since this number of minutes may exceed one hour, update the hours portion of the arrival time: *h* = *h* + *m* / 60
4. Since the minutes portion of the arrival time may exceed 59, calculate the correct value to display: *m* = *m* % 60
5. Since the hours portion of the arrival time may exceed 23, calculate the correct value to display: *h* = *h* % 24
6. Display the results using "*h*:*m*"

You may assume that the four inputs to the program exist and are positive integers, and that *h* is between 0-23 and that *m* is between 0-59.

After completing Part 2, take a break!

Part 3: Great Circle

For very long journeys, for instance between cities on two different continents, a map/directions application will estimate the distance using the cities' geographical latitude and longitude coordinates.

Since the earth is a sphere and not a plane, though, the application cannot use simple Euclidean distance but rather must use the "great-circle distance," which measures the distance between two points on a sphere.

Write a program called **GreatCircleDistance** that takes four runtime arguments as doubles — x1, y1, x2, y2, representing the latitude and longitude, in degrees, of two points x and y on the earth -- and prints the great-circle distance between them.

The great-circle distance, *d* (in nautical miles) is given by the following equation:

$$d = E * \arccos(\sin(x1) * \sin(x2) + \cos(x1) * \cos(x2) * \cos(y1 - y2))$$

where you can use the earth's radius E = 3986 miles.

Notes:

- You will need to use the **Math.sin, Math.cos, and Math.acos** functions to perform the trigonometric functions.
- The latitude and longitude of the two points are in **degrees**, whereas Java's trigonometric functions use **radians**. Use **Math.toRadians(d)** and **Math.toDegrees(r)** to convert between the two as needed.

The following example shows the output for the distance between Philadelphia (39.9526 N and 75.1652 W) and Honolulu (21.3069 N and 157.8583 W):

```
[~]$ java-introcs GreatCircleDistance 39.9526 75.1652 21.3069 157.8583

4945.290632589314 miles
```

This next example shows the distance between Paris (48.87 N and -2.33 W) and San Francisco (37.8 N and 122.4 W):

```
[~]$ java-introcs GreatCircleDistance 48.87 -2.33 37.8 122.4

5598.250822014901 miles
```

You may assume that the four inputs to the program exist and are all valid latitudes and longitudes, i.e. are numbers between -180 and 180, inclusive.

Getting Help

Learning how to program can be quite challenging at first and it is completely normal to struggle and run into issues on your first assignment. Don't get discouraged, though! You learn by overcoming that struggle and finding ways to address the issues that you face.

If you find yourself stuck on this assignment and not making progress for, say, 30 minutes or so, then the best thing to do may be to walk away from it for a while and try to think about something else. It often is the case that your brain just needs a rest and that the solution will seem a bit clearer if you look at it with a refreshed mind.

If you're still stuck, though, then don't worry! The CMSC B113 instruction staff is here to support you and to help you do well on this assignment.

The course Teaching Assistants (TAs) are available for drop-in office hours, and the schedule is posted on the course website. You do not need an appointment for these office hours: you can just show up and the TA will help you as soon as they can.

You don't need to wait for office hours, though. You can post a question in Piazza so that the community of students, TAs, and instructors can try to help you, as well. Please be sure not to accidentally post your solution publicly, though! You should at most only be posting 3-4 lines of code or so.

Last, it is also okay to email your instructor, especially with private concerns, though posting in Piazza is likely to be the fastest way to get help, especially if no office hours are being held.

Submitting Your Solutions

To submit your work, copy/paste the source code of your three Java programs into a *single* PDF, and submit the PDF in the "Assignment 1" subfolder of the Dropbox folder that your instructor created for you.

To create the PDF, you can put the code into Microsoft Word and export/print it as a PDF, or put it into a Google Doc and then do File → Download → PDF.

Please be sure to put all of your code into a *single* document and please only submit a PDF, not a Microsoft Word .docx file or the .java source files.

It's important to get the semester off to a good start, so if you know that you are unlikely to be able to submit this assignment by the due date, please notify your instructor *immediately*. Assignment #2 is likely to come out right after this one is due, and we don't want you to fall behind!