CMSC B113 Computer Science 1 – Spring 2025
Programming Assignment #1

Overview

Websites such as Google Maps, Mapquest, Apple Maps, Tomtom, 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 some small programs that perform calculations related to time and distance, similar to what would be used in a map/directions application.

In completing this assignment, you will learn to:

- Understand and address Java compiler and runtime errors
- Declare, initialize, and update variables in a Java program
- Perform mathematical operations in a Java program
- Write programs that get input from the command line
- Write to the terminal in a Java program

Even if you have prior programming experience, it is strongly recommended that you get an early start on this assignment. There are four parts, and you don't necessarily need to do them all in one sitting, but you don't want to wait too long to get started! See the "Getting Help" section toward the bottom of this document if you need support in completing this assignment.

Part 1: Debugging

Debugging is a huge part of the process of programming, and you may often find yourself spending more time debugging your code than you did writing it in the first place!

There are typically three types of errors that you may run into while programming:

- **Compiler error:** this is where the code will not compile when you run "javac-introcs" because the code is not valid Java. Also called **syntax error**.
- **Runtime error:** this is where the code compiles, but then performs an illegal operation when you run "java-introcs", which then causes the program to crash
- **Logical error:** this is where the code compiles and the program runs to completion, but produces the wrong output

In this part of the assignment, you will get a little bit of experience in understanding the types of error messages that Java may display when it runs into a problem.

Open VS Code, and create a new file by selecting File → New File from the menu, then select File → Save to save the file in your cs113 directory and name it **`Time.java`**.

Then copy/paste the following code so that the file contains only these lines:

```
public class Time {
    public static void main(String[] args) {
        int minutes = Integer.parseInt(args[0]); // convert String on int
        System.out.println(minutes + " minutes")
    } // main()
} // class Time
```

Be sure to save the file after you have added the code!

Before you edit the program to address any errors, you need to first compile it. If the Terminal window is not already open on the bottom of the screen, select Terminal → New Terminal from the menu; the terminal that opens should read "bash" in the upper right corner.

First, compile the code with this command: `javac Time.java`

You should see the following error message:

```
Time.java:4: error: ';' expected
        System.out.pritln(minutes + " minutes")
                                                ^
1 error
```

Based on the error message provided by the compiler, modify the code to fix the problem. Once you have fixed it, save the file and recompile it, and continue to fix any compiler errors that you receive (hint: there may be more than one!).

Once you have fixed all the errors, you will no longer see any error messages. Now you can run the program with the command: `java Time 10`

Depending on the changes you made, you may receive an error here, too! If so, fix the error in the code, save the file, recompile it, and re-run it; keep in mind that you must recompile it before you re-run it!

Once the program displays "10 minutes", then you're ready to move on to the next part!

Part 2: Converting Time

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.

Modify the **Time** program from Part 1 so that, instead of just displaying the number of minutes, it converts it to the equivalent in hours and minutes and displays that instead. Here are some sample program executions and expected outputs:

```
java-introcs Time 127
2 hours and 7 minutes

java-introcs Time 453
7 hours and 33 minutes

java-introcs Time 360
6 hours and 0 minutes

java-introcs Time 77
1 hours and 17 minutes
```

Note that the last example is not grammatically correct but it's okay for now! You'll fix this in a later assignment. 😉

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.

Part 3: 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 **Arrival** that takes as input the current time, the distance to travel, and the expected average speed, and prints the estimated arrival time. The four program inputs (or "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:

```
java-introcs Arrival 11 10 15 60
Arrival Time is 11:25

java-introcs Arrival 15 50 125 50
Arrival Time is 18:20

java-introcs Arrival 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.

Part 4: Calculating Great Circle Distance

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 **Distance** 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* 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.

Since x1, y1, x2, and y2 are meant to be doubles, you will need to use the **Double.parseDouble** function to convert the program inputs from Strings to doubles.

To calculate the distance, you will need to use the **Math.sin, Math.cos, and Math.acos** functions to perform the sine, cosine, and arccosine trigonometric functions, respectively. For instance:

```
double s = Math.sin(angle);
```

returns the sine of **angle** and stores it in the variable **s**.

However, the latitude and longitude of the two points are in **degrees**, whereas Java's trigonometric functions use **radians**. Use **Math.toRadians(a)** to convert the angle **a** from degrees to radians before using the sine and cosine functions.

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 Distance 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 Distance 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 floating-point numbers between -180.0 and 180.0, 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 in Park 230 and 231 in the evenings from Sunday through Thursday (see hours posted on course website). You do not need an appointment for these TA Hours: you can just show up and the TA will help you as soon as they can.

Academic Honesty

Unless otherwise noted, all Programming Assignments in this course are subject to the College Honor Code and to the course policy on academic honesty as presented in the syllabus in Moodle.

We trust that you will not discuss the details of your solutions or share code with any of your classmates. Although it is okay to discuss the intent of the assignment with classmates and to ask for help with clarification, the work that you submit should be your own. After all, the purpose of these assignments is for you to learn how to program in Java, and you won't learn as much if someone else is doing the work for you!

Please consult with the instructor if you have any questions about what is and is not permitted.

What to Submit

The due date for the assignment is posted on the course website. All assignments are due at the start of class on the due date. Your submission should contain a printout of all your programs, plus a printout of the sample runs (use the examples in this handout for sample inputs and confirm the results shown) to show that your program produces correct results. You can cut and paste everything into a well formatted Word document and print that out. Write up a short note on your reflections from doing this assignment. **Staple all the sheets together and leave your submission at the front desk in class as you enter**.