Lecture 04

User Input

21 Prairial, Year CCXXX

Song of the day: ***This Hell*** *by Rina Sawayama (2022).*

Sections

Part 1: Review

The problem

One of the most obvious signs that you have entered another country once you cross the US border with Canada is the fact that all speed limits are now posted using metric units; if you have been living in the United States for a while and are used to cruising at, say, 60 miles—per—hour, suddenly seeing a speed limit sign with a huge `100` (without any units written on it) can be pretty confusing.

Figure 1: Graphic design is their passion, apparently. ***Source***

Anyway, using all the conventions that we learned in class on Monday, write a public class that will convert 60 miles-per-hour from imperial units to metric units and print it out. Then, your program must repeat the process but in reverse: convert 100 kilometres-per-hour from metric units to imperial. You may name this class, along with all its variables and constants, however you feel appropriate.

These conversion factors may come in handy:

- **1 mile**: 1609.34 metres.
- **1 kilometre**: 0.621371 miles.

Recall that we learned naming and casing conventions for both variables and constants. Please apply them accordingly.

As a challenge, try to convert the acceleration of gravity (approximately 9.81 metres per second^2) to miles per hour-squared! (You can do this in the same class, if you'd like)

Sample output:

```
60.0 miles per hour is equivalent to 96.5606698735538 kilometres per hour.
100.0 kilometres per hour is equivalent to 62.137100000000004 metres per hour.
```

Challenge sample output:

```
9.81 metres per second-squared is equivalent to 78999.6176496 miles per hour-squared.
```

The solution

As you'll see later in the semester, Java encourages you to divide your programs depending on the type of data and/or functionality they represent. So I'll be splitting the main method of my class, which I named `SpeedLimitConverter`, as follows:

```
public class SpeedLimitConverter {
    public static void main(String[] args) {
        // Constants

        // Variables

        // Calculations

        // Display

    }
}
```

Code Block 1: The skeleton of our program.

Cool, so let's tackle them one-by-one. You could have used either (or both) of the conversion that I provided. I chose the second one, so I threw that into our constants section. Recall that constants in Java take the `final` keyword to avoid being modified, and ARE_WRITTEN_LIKE_THIS:

```
// Constants
final double MILES_PER_KILOMETRE = 0.621371;
```

Next up, I put our speed limits in the variables section. Recall that these have to have clear names, and follow camelCase:

```
// Variables
double imperialSpeedLimit = 60.0;
double metricSpeedLimit = 100.0;
```

The calculations that follow are simple using these values. I chose to save their results in variables for ease of printing later:

```
// Calculations
double imperialToMetricSpeedLimit = imperialSpeedLimit / MILES_PER_KILOMETRE;
```

```
double metricToImperialSpeedLimit = metricSpeedLimit * MILES_PER_KILOMETRE;
```

Finally, we can match the sample output in the prompt by doing the following:

```
// Display
System.out.println(imperialSpeedLimit + " miles per hour is equivalent to " + imperialToMetric
        " kilometres per hour.");
System.out.println(metricSpeedLimit + " kilometres per hour is equivalent to " + metricToImper
        " metres per hour.");
```

If you chose to do the challenge, I separated it visually from the first part by adding the following comment:

```
/* CHALLENGE */
```

This is another way of writing comments, and are usually used for denoting more important messages and/or official documentation. Here's the code that got me the correct result (note that, in this case, there were no variables except for the result of the conversion):

```
/* CHALLENGE */
// Constants
final double ACCELERATION_OF_GRAVITY_SI = 9.81; // m/s^2
final double MILES_IN_METRE = 0.000621371;
final double SECONDS_IN_HOUR = 3600.0;

// Calculations
double imperialAccelerationOfGravity = ACCELERATION_OF_GRAVITY_SI * MILES_IN_METRE * SECONDS_I
        SECONDS_IN_HOUR;


// Display
System.out.println(ACCELERATION_OF_GRAVITY_SI + " metres per second-squared is equivalent to '
        imperialAccelerationOfGravity + " miles per hour-squared.");
```

You can find my full solution **here**.

Part 2: *Numeric Operators and Converting Types*

The following numeric operators are available to us in Java:

Symbol	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000

Symbol	Meaning	Example	Result
`/`	Division	`1.0 / 2.0`	`0.5`
`%`	Remainder	`20 % 3`	`2`

Figure 2: Numeric operators in Java.

A couple of notes on these:

- Watch out when using the `/` operator—specially if your operands happen to both be `int` types. Java will try its best to preserve the original type of the operands. If they are both integers, it will round _down to the integer closest to the result:

```
int firstOperand = 3;
int secondOperand = 4;

/**
 * While we would expect something like 0.75 to be printed, this will actually print 0.
 */
System.out.println(firstOperand / secondOperand);
```

- The `%` operator, often called the **modulus** (or mod for short) operator, is often a source of great confusion to students. If you picture a long division, it is basically the number that you get after finishing the process:

```
     1
   ____
13 | 20
     13
  ______
     7   <---- this is the value that 20 % 13 would give you.
```

In order to avoid the first issue, a good tip is to **convert** one or both of your operands to a `double` before or during your division. Conversion between types is called ***casting***, and it would be done as follows:

```
int firstOperand = 3;
int secondOperand = 4;

double firstOperandDouble = (double) firstOperand;      // 3.0
double secondOperandDouble = (double) secondOperand;    // 4.0

System.out.println(firstOperand / secondOperand);       // prints 0.75, a double
```

Code Block 2: Casting integers into `double` types. Note that Java will always pick the most precise type for the result of a mathematical operation. Therefore, if we only casted `firstOperand` as a double and tried `3.0 / 4` , we would get our desired result of `0.75` (since `double` is more precise than `int`).**

Be careful when typecasting from more precise types (like a `double`) to a less precise type (such as a `float`), since you will be losing a good amount of precision and thus could potentially lead to inaccurate results depending on the use case.

One last thing you may have noticed (specially if you are coming from a language like Python) is that Java does not have an exponentiation operator. We, thus, have to multiply values several times in order to get exponentiation-like results:

```
int twoToTheEighthPower = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2;
```

It's not hard to see how this is not only inefficient, but is also super prone to human error. Favour instead the `Math.pow()` method, which accepts 2 numbers as parameters (the base, and the power) returns a `double`:

```
int twoToTheEighthPower = (int) Math.pow(2, 8);  // 256
```

Code Block 3: Note that we have to cast down to an integer in order for the result to be allowed to be stored inside our integer parameter.

Let's put some of these concepts to use by writing a class that will print the current time. There is this cool little curiosity in computer science called the **UNIX timestamp**:

> It is the number of seconds that have elapsed since the Unix epoch...The Unix epoch is 00:00:00 UTC on 1 January 1970 (an arbitrary date).

At the time of writing, this **timestamp** clocks at 1,654,790,322 seconds. How can we use this number to print out the current time? Well, we can get the UNIX timestamp by calling the following method:

```
System.currentTimeMillis();
```

So, using this value, let's follow this algorithm in order to get each of the elements necessary to print the current time:

1. Calculate the total amount of seconds by **dividing** `System.currentTimeMillis()` (an int) by `1000` (i.e the amount of seconds we can form with our total amount of milliseconds).
2. Calculate the current second by **modding (`%`)** the total number of seconds (an int) by `60` (i.e. the amount of seconds leftover after grouping as many of them as we can into minutes).
3. Calculate the total amount of minutes by **dividing** `System.currentTimeMillis()` (an int) by `60` (i.e the amount of minutes we can form with our total amount of seconds).

4. Calculate the current minute by **modding** the total amount of minutes (an int) by `60` (i.e. the amount of minutes leftover after grouping as many of them as we can into hours).
5. Calculate the total amount of hours by **dividing** the total amount of minutes (an int) by `60` (i.e the amount of hours we can form with our total amount of minutes).
6. Calculate the current hour by **modding** the total amount of minutes (an int) by `24` (i.e. the amount of hours leftover after grouping as many of them as we can into days).
7. Print the values calculated.

In **code**, these would look like this:

```
package userInput;

import java.util.Scanner;

public class ShowCurrentTime {
    public static void main(String[] args) {
        /* Constants */
        final int MILLISECONDS_IN_SECOND = 1000;
        final int SECONDS_IN_MINUTE = 60, MINUTES_IN_HOUR = 60;
        final int HOURS_IN_DAY = 24;

        /**
         * Current Time
         * */
        /* Calculations */
        long totalMilliseconds = System.currentTimeMillis();            // Step 1
        long totalSeconds = totalMilliseconds / MILLISECONDS_IN_SECOND; // Step 2
        long currentSecond = totalSeconds % SECONDS_IN_MINUTE;          // Step 3
        long totalMinutes = totalSeconds / MINUTES_IN_HOUR;             // Step 4
        long currentMinute = totalMinutes % MINUTES_IN_HOUR;            // Step 5
        long totalHours = totalMinutes / HOURS_IN_DAY;                  // Step 6
        long currentHour = totalHours % HOURS_IN_DAY;                   // Step 7

        /* Display */
        System.out.println("Current time is " + currentHour + ":" + currentMinute + ":" + curren

        /**
         * User-Entered Time
         * */
        /* Get user input */
        Scanner scanner = new Scanner(System.in);

        /* Calculations */

        totalMilliseconds = scanner.nextLong();                     // Step 1; ask the user for L
        totalSeconds = totalMilliseconds / MILLISECONDS_IN_SECOND; // Step 2
        currentSecond = totalSeconds % SECONDS_IN_MINUTE;          // Step 3
        totalMinutes = totalSeconds / MINUTES_IN_HOUR;             // Step 4
        currentMinute = totalMinutes % MINUTES_IN_HOUR;            // Step 5
        totalHours = totalMinutes / HOURS_IN_DAY;                  // Step 6
        currentHour = totalHours % HOURS_IN_DAY;                   // Step 7

        /* Display */
        System.out.println("Current time is " + currentHour + ":" + currentMinute + ":" + curren
    }
```

```
}
```