CSCI165 Computer Science II
Lab Assignment

Objectives:

- Research and consult the Java API
- Work with Java primitive data types
- Work with the String class

Complete the following problems 1 - 6 in a file called: **Primitives.java**

1. Define and initialize variables of each of the Java primitive types. Use appropriate sample data that is not the default values. Print each value with a descriptive method using ***printf***.
   https://www.baeldung.com/java-printstream-printf

   Demonstrate both character and numeric literals for the **char** type. Refer to ***Data_and_Variables.pdf*** or Oracle's Java Tutorials:
   https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

2. Perform a series of ***implicit widening casts*** and ***explicit narrowing casts***. Be sure to include narrowing casts that result in value manipulation. Print the values. Include comments that describe what is happening with the values when they are cast from type to type, especially values that are changed by a narrowing cast.

3. Create two variables of type **int**. Assign these variables the maximum and minimum values of this data type. Use the **MIN_VALUE** and **MAX_VALUE** defined constants in the Integer class. Each primitive type has a corresponding class type. Research the **Integer API** for this.
   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Integer.html

   Print these values with a meaningful message.

4. Repeat the process described in step 3, except with the type **long**. Also show the difference between ***Integer.MAX_VALUE*** and ***Long.MAX_VALUE***.
   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Long.html

5. Using a ***command line argument*** enter an integer when you run the program. You will need to convert the value to an int. Because Strings are objects, you cannot use type casting; you have to call a method. Check out the ***parseInt*** method in the Integer class.
   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Integer.html

   Display the square, cube, and fourth power. Display with descriptive messages. Research the **Math** class and use the **pow** method for each calculation. Use a loop if you'd like.
   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Math.html

6. Add the ability to process two more command line arguments. Enter an integer dividend and divisor. Compute floor division and floor modulus. Use the operators (/ and %) and the **floor** methods from the Math class. Look this up in the API. Print the result with a descriptive

message using **printf**
https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Math.html

7. Create a new source file called **Initials.java**. Write Java code that uses command line arguments for the users first and last name. Concatenate these into a **single String variable.** Extract the first character of the first name into a variable of type **char**. Extract the first character of the last name into a variable of type **char.** Use the String method **indexOf** to locate the index of the space. Use this location to get the initial. Research the String API spec.
   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/String.html

   Print

   1. The characters individually
   2. The numeric values of the characters (Unicode value). Perform a cast to int to accomplish this.
   3. The sum of the numeric values
   4. The characters concatenated together as a String.

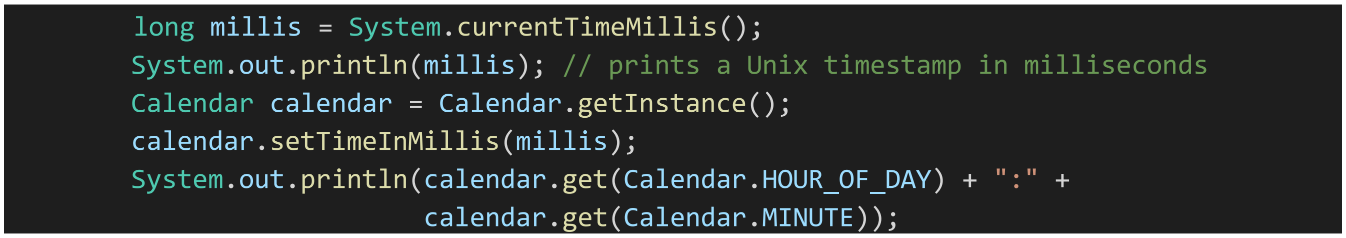
8. **Greenwich Mean Time (UTC), the UNIX Timestamp:** Dates, times and time zones are a constant source of frustration for programmers. It's best to begin wrapping your head around this concept now. Begin here
   https://currentmillis.com/tutorials/system-currentTimeMillis.html

   https://upload.wikimedia.org/wikipedia/commons/e/e8/Timezones2008_UTC%2B2_gray.png

   Create a ***new source file*** called **UTC.java**. In this file you will experiment with two ways to calculate the time in HH:MM:SS format.

   Format One: This format is easy, you let the language do all the hard work for you. Use the example presented in the article above. Using Java date/time classes will allow you to work with ***time zone adjusted*** dates. Be sure to import ***java.util.Calendar***. Import statements appear in your source code ***above your class definition***. Importing classes gives your program the ability to use things from the Java library. The ***getInstance()*** method below illustrates one way to create a workable object from a class. In this case you are asking the Calendar class to give you an ***instance***. An instance is another word for a workable object. You can now send messages to this object by calling methods. The methods you can call, their parameters and return types are listed in the API.
   https://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html

   Don't worry about the Java version here. Java versions are backwards compatible and contain everything from previous versions. While there may be newer Calendar/Date/Time classes, it is perfectly fine to use an older one. You will need to consult the API to figure out to include the seconds



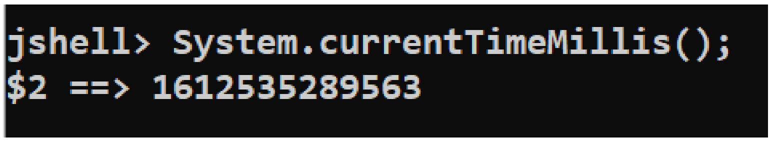
```java
long millis = System.currentTimeMillis();
System.out.println(millis); // prints a Unix timestamp in milliseconds
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(millis);
System.out.println(calendar.get(Calendar.HOUR_OF_DAY) + ":" +
                   calendar.get(Calendar.MINUTE));
```

Format Two: Raw time calculation using math. Big Brain Time. This exercise is to allow you to get familiar with Java mathematical expressions and primitive types.

1. Use the **currentTimeMillis** method from the System class. Remember, this returns a ***UNIX timestamp;*** the number of milliseconds since the **UNIX Epoch**. This gives you a very large number. This number is not ***time zone adjusted***. It is straight UTC, and if every human from every time zone had perfectly calibrated system clocks, we would all get the exact same value if we ran this at the exact same time. (practically impossible)

   https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/System.html#currentTimeMillis()

2. Here is the result of this call on **Friday 2/5/2021 at 9:28 AM**



```
jshell> System.currentTimeMillis();
$2 ==> 1612535289563
```

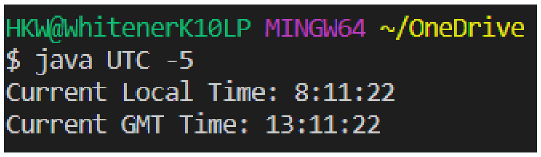
3. Using a command line argument have the program accept the time zone offset to GMT (-5 would be New York). This value is the number of hours we are from GMT (either east, negative offset, or west, positive offset). To calculate local time you'll need to add these hours to the timestamp. Consult the map above for a more thorough picture of this concept. **Don't worry about the 15 minute increments.**

4. Using math (addition, division, modulus division) convert the millisecond UNIX time stamp to HH:MM:SS. Military time is fine. Things to think about:

   1. How many years has this been?
   2. How many months has this been?
   3. How many days?
   4. Only then can you think about smaller quantities like hours, minutes and seconds.

5. ***<u>You are not allowed to use if statements</u>***. Construct an appropriate mathematical expression using things like division, modulus division, multiplication etc. This is essentially a math problem.

6. **Sample Run at 8:11:22 AM:**



```
HKW@WhitenerK10LP MINGW64 ~/OneDrive
$ java UTC -5
Current Local Time: 8:11:22
Current GMT Time: 13:11:22
```

Submission:

Push to the ***week2*** folder in your repository. You should have three files to push for this lab: ***Primitives.java, Initials.java, UTC.java*** Please do not push **.class files**