The objective of this project is to apply the concepts covered in Modules 3 and 4.

For this activity, you will practice creating methods that accomplish a specified behavior.

A Non-conventional Time Conversion

Your task for this project is to complete the five methods outlined in the starter code provided for this project.

Download the files *TimeConversion.java* and *TimeConversionDriver.java* and open them in your IDE. To start you will notice that TimeConversionDriver.java shows errors. This is to be expected as our program is not yet complete. We need to add enough logic to the TimeConversionDriver and TimeConversion classes so that when the TimeConversionDriver class is executed (run) it asks the user to enter a number that represents the number of seconds to be converted then outputs information about converting this value to some non-conventional time units.

The majority of code you need to add will be in the TimeConversion class. The starter code includes comments for what you need to do.

Part A - 5 points

Complete the TimeConversionDriver class to ask the user to enter a time in seconds.

Tip: Refer to Tackling a programming task section below.

Part B - 15 points

Complete the TimeConversion class by adding the following methods:

- Make a method named ***showDecaseconds***, which accepts the number of seconds as an argument. The method should display a description of Decaseconds (a decasecond is 10 secs) and the argument converted to decaseconds,
- Make a method method named ***showJiffies***, which accepts the number of seconds as an argument. The method should display a description of a Jiffy (A jiffy is a unit of time used in computer operating systems. It is 10 milliseconds.) and the argument converted to jiffies,
- Make a method method named ***showNewYorkMinutes***, which accepts the number of seconds as an argument. The method should display a description of a New York Minute (A new york minute is the period of time between the traffic

lights turning green and the cab behind you honking. It is 1/20th of 1 second.) and the argument converted to new york minutes,
- Make a method method named ***showNanoCenturies***, which accepts the number of seconds as an argument. The method should display a description of a Nanocentury (A nanocentury is a computing measurement coined from the expression - "never to let the user wait more than a few nanocenturies for a response". It is 3.156 seconds.) and the argument converted to nanocenturies,
- Make a the method method named ***showScaramuccis***, which accepts the number of seconds as an argument. The method should display a description of a Scaramucci (A scaramucci is the tenure of former White House Communications Director Anthony Scaramucci. It is 11 days.) and the argument converted to scaramuccis.

In summary, a working program will

- Allow the user to enter a whole number
- Output:
    - A description of a decasecond and the number converted to decaseconds
    - A description of a jiffy and the number converted to jiffies
    - A description of a new york minute and the number converted to new york minutes
    - A description of a nanocentury and the number converted to nanocenturies
    - A description of a scaramucci and the number converted to scaramuccis

Tackling a programming task

Rather than trying to code everything at once and then trying to run and fix errors you encounter it might be better if you divide up the work. For example, after completing Part A you can comment out the statements that come after the first print statement so that when you try and run the driver class the only thing that the main method will do is ask the user for input and print that out. So if you do this and verify that it works as you expect it to (i.e., you enter a value and you see it printed) you can move on to the the next part knowing that this step has no errors.

When correcting errors, start by correcting the code that caused the uppermost error in the error list then recompile. Remember that a malformed line of code can actually cause

errors in subsequent lines of code that are would otherwise be correct. Often, fixing one error results in other "errors" disappearing.

Similarly with part B, you can complete one method in the TimeConversion class and comment out enough lines in the driver class so it only asks the user for input and the TimeConversion object only calls this method. As you continue adding methods you can comment/uncomment lines to make your testing and debugging more focused and efficient. Remember to uncomment all code before submitting this assignment.

Happy coding!

Coding Style – 3 points

As we mention in class, formatting your code make a difference when someone else has to look at it and review it. It is important to practice good formatting habits. Here are some guidelines to follow:

- Appropriate variable names. In general, do not use single letters like X. Make your variables names descriptive (e.g., hrlyWage, rent,...). Use camel case.
- Proper indentation
- Good commenting (explains what code is doing)
- Well-organized, elegant solution

Submission Requirements - 2 points

1. Submit the code file (**TimeConversion.java**).
2. Submit the code file **(TimeConversionDriver.java**).
3. Submit a PDF document with the name **project2Info.pdf** that includes the following assignment information:
    1. List of references used to acquire information
    2. Explanation of any special or additional features you added, if any.
    3. Explanation of status, stopping point, and issues if incomplete.
    4. Discuss the easy and challenging parts of the assignment. How did you overcome all or some of the challenges?