You can access to the class material from the following link:

https://drive.google.com/drive/folders/1yo4pvaTngQ1L2TkHuWPCzbfapLDyvwUt?usp=sharing

You can check your lab attendance and quiz scores from the following link:

https://docs.google.com/spreadsheets/d/1xKQEqMSGIj3RBo9kLh97UzbTO1fyCvaT/edit?usp=sharing&ouid=104851561367296854968&rtpof=true&sd=true

Coding Challenges

This document contains coding challenges. This document does not necessarily contain questions similar to the final exam. This document helps you practice the concepts. Final exam questions will be similar to the labs, lecture coding examples and programming projects.

1) Class Creations:

Create a class called Measurement that has a single (instance) double value called quantity. Everything using your class should be able to change and access the quantity value. Make it so that when you create an instance of the class with new, you must specify the value for quantity. Write the class using the proper object-oriented style for Java.

2) Conditional Statements:

Create a class (you can call it anything) with a single static method **truncate** that takes a single **int** input **x**. Use a conditional statement so that **truncate** returns 0 if **x** is negative, **x** if x is even, and **x-1** if **x** is odd.

3) toSting and equals methods:

Take the **Measurement** class you created before and change how the toString and equals methods work.

- the **toString** method should return the value of the quantity as a string
- two measurements should be equal if their quantities are equal

4) Constructor

Create a class **Temperature** that extends the **Measurement** class you previously wrote. A **Temperature** should have a single value *quantity*.

5) While Loop 1

Write a static method **greatestOddFactor** that takes an integer **n** and uses a while loop to repeatedly divide 2 from **n** (storing the result back into **n**) as long as **n** is even. Then return the result.

6) While Loop 2

Create a method **cuberoot** that takes a double value as input and (using Newton's method) returns the cube root of that input.

7) For Loop 1

Write a static method **palindromeError** that takes a String as input and it returns the first *index* where that character does not match its opposite character. If the input is a palindrome, then the return value should be -1. (You code this exactly the same as the *isPalindrome* example, but when the character at the front does not match the character at the back, you return the front index value rather than false.) Write a static method **palindromeError** that takes a String as input and it returns the first *index* where that character does not match its opposite character. If the input is a palindrome, then the return value should be -1. (You code this exactly the same as the *isPalindrome* example, but when the character at the front does not match the character at the back, you return the front index value rather than false.)

8) For Loop 2

This challenge has two parts.

1) Write a method **toLowercase** that takes a **String** as input and returns a **String** that matches the input except that every upper case letter is converted to lower case.

2) Write a method **shout** that takes a **String** and an **int** as input. It returns a **String** that is the input string plus the **int** number of exclamation points added to the end. So **shout("Hi", 10)** outputs **"Hi!!!!!!!!!!"**.

9) Array 1

Write a static method **squareAll** that takes an **int array** as input and replaces every element x of the array with its square, x*x. As a hint to help you test your code, the required reading shows a fast way to place elements into an array called an "array initializer".

10) Array 2

Write a static method **append** that takes two **int arrays** as input and returns an array that appends the second array onto the first array. So you should have a single array that contains all the contents of both input arrays, in order.

11) Array 3

Create a static method **boundSearch** that takes three inputs: an int that is a **low** bound, an int that is a **high** bound, and an **array of int**. The method should return the index of an element whose value is at least as large as the low bound and no larger than the high bound. If there is no such element, return -1. If there are multiple such elements, return the index of any one of them.

12) Multi-dimensional Array

1) Create a method **sum** that takes a 2-dimensional array of double as input and returns the sum of all the entries in the array. If there are no entries in the array, the method should return 0.
2) Create a method **sum** that takes a 3-dimensional array of double as input and returns the sum of all the entries in the array. If there are no entries in the array, the method should return 0.

13) Main method

Create a program called Echo that when you run it, it repeats the String typed in. It does not have to match the spaces exactly:

```
> java Echo Hello Everyone!  I hope you are having fun.

Hello Everyone! I hope you are having fun.
```

14) Class Hierarchy

Create two types: one to model an "ellipse" and one to model a "circle". An ellipse is defined by two values: the major axis and the minor axis, and a circle is defined by its radius. If an ellipse has its major axis equal to its minor axis, then it is a circle. Use good object oriented coding to place these into the shape hierarchy we already created.

15) Interface

Create an **interface** called **Friendly** that has a single instance method **greeting** that takes no inputs and returns a **String**.

Have our **Employee**, **HourlyEmployee**, and **GeometricFrame** classes all implement the **Friendly** interface, and their **greeting** method shoud return a friendly welcome. The employee's should give their names and the geometric frame its size.

Create a class with a static method **printGreeting** that can take and **Friendly** class as an input and prints the greeting of that class using **System.out.println**.

16) Linked List Node

Using the LLNode class from the module, attempt the following in order:

1) Create an LLNode storing the int value 3.

2) Create an LLNode storing the int value 1, and place that node *in front* of the node containing 3.

3) Create an LLNode storing the int value 2, and place that node *after* the node containing 1, but *before* the node containing 3.

Take your time to code it. You can if the values are stored correctly by using node.getElement(), node.getNext().getElement(), and node.getNext().getNext().getElement().

17) Linked List

Inside the LinkedList.java class of the module, create a method:

- swapFirstTwo that swaps the first two nodes (not elements) of the linked list. If the list is empty or has only one node, the method should do nothing.

18) Linked List Loops

Add the following method to the body of the LinkedList.java class from the module.

- insertSecondToEnd takes an element as input and inserts the element into the linked list so that the element becomes the second to last element of the linked list. If the linked list is empty, add the element as the only element of the list.

19) Collections

Write a static method that will work for both an ArrayList containing Double or a LinkedList containing Double (both from the Java Collections). Have the method return the sum of all the values in the ArrayList or LinkedList.

20) Generic Array

Create a static method that takes an array storing non-primitive values and two elements as input. The method should replace every occurrence of the first element in the array with the second element.