This problem set is due at 8:00pm on Wednesday, October 10, 2018, KST. Don’t go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

- To solve each problem below, you will be implementing a Java class.
- Please carefully read and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
- You should create your programs using emacs, not Eclipse.
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what is a readable style, see the examples from my lecture and textbook as a starting point for a reasonable coding style.
- This problem set assumes you have installed Java and emacs on your computer.
- You must use the command-line interface to run your programs. That is, you must use the javac and java commands to do this homework. Do not use Eclipse yet.
- Be sure to include a comment at the top of each file submitted that gives your name and email address.

What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lecture 1 through Lecture 9 (Chapters 1 through 7 of our textbook). Note that first five problems deal with materials from Chapter 6 and last two problems deal with materials from Chapter 7.

What to Submit

Submit the following files as a single zip or tar file on Blackboard. (A zip file if you are using Windows and a tar or rar file if you are using a Mac. Multiple submissions are allowed before the due date. Please do not submit .class files or any I did not ask for.

Cylinder.java
Digits.java
TwinPrimes.java
StringPalin.java
Password1.java
Frequency.java
Password2.java
Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Problem 1 (10 points)

Read the FAQ next to this handout for hints (and possibly corrections) before you try each of the problems below.

Write a program (Cylinder) that includes the following methods in addition to a main.

- a method called sphereVolume that accepts a radius parameter and returns the volume of a sphere with that radius. For example:
  - sphereVolume(2.0) should return 33.510321638291124 or a number close to it.
  - sphereVolume(5.0) should return 523.5987755982989 or a number close to it.

- a method called cylinderSurfaceArea that accepts a radius and height as arguments and returns the surface area of a cylinder with those dimensions. For example:
  - cylinderSurfaceArea(3.0, 4.5) should return 141.3716694115407 or a number close to it.
  - cylinderSurfaceArea(4.5, 6.0) should return 296.8805057642354 or a number close to it.

For PI, use the Java-supplied static constant: Math.PI and your program should read the radius, height values from the user. Here is one sample run:

Enter a sphere radius: 2.0
Sphere volume: 33.510321638291124

Enter another sphere radius: 5.0
Sphere volume: 523.5987755982989

Enter a cylinder radius and height: 3.0 4.5
Cylinder surface area: 141.3716694115407

Enter another cylinder radius and height: 4.5 6.0
Cylinder surface area: 296.8805057642354

My sample solution for this problem did not use any of the following: conditionals, loops. It is a simple problem dealing with methods (Chapter 4 material). Hand in Cylinder.java.

Problem 2 (10 points)

Write a program (Digits.java) with a main method that prompts the user to enter an integer and sums the digits in the integer. You must define a method of the following header and use it:

    public static int sumDigits(int n)

Here are three sample runs:
Enter an integer: 0
Sum of digits of 0 is 0

Enter an integer: 9
Sum of digits of 9 is 9

Enter an integer: 34521
Sum of digits of 34521 is 15

Hand in Digits.java.

Problem 3 (10 points)
Write a program (TwinPrimes.java) with a main method that prompts the user to read an integer and prints all twin primes that are less than the integer. Twin primes are a pair of prime numbers that differ by 2. For example, 3 and 5 are twin primes. So are 11 and 13. Here are two sample runs:

Enter an integer: 13
(3, 5)
(5, 7)

Enter an integer: 14
(3, 5)
(5, 7)
(11, 13)

You must define two methods of the following headers and use them:

```java
public static boolean isPrime(int n)
public static boolean isDivisibleBy(int i, int j)
```

Hand in TwinPrimes.java.

Problem 4 (10 points)
Write a program (StringPalin.java) with a main method that prompts the user to read a string and determines if it is a palindrome. You must define two methods of the following headers and use them:

```java
public static String reverse(String s)
public static boolean isPalindrome(String s)
```

Here are three sample runs:

Enter a word: Apple
Apple is not a palindrome

Enter a word: Appa
Appa is a palindrome

Enter a word: alula
alula is a palindrome

Hand in StringPalin.java.
Problem 5 (10 points)
Write a program (Password1.java) with a main method that prompts the user to read a password and determines if the password read in is a valid password or not. A password is valid if it passes all of the following rules:

- A password must have at least eight characters.
- A password must consist of only letters and digits.
- A password must contain at least two digits.

You must define three methods of the following headers and use them:

```java
public static boolean isValidPassword(String s)
public static boolean isLettersOrDigits(String s)
public static int countDigits(String s)
```

Here are three sample runs:

Enter a password: ComputerNot
ComputerNot is not valid

Enter a password: Computer2Not3
Computer2Not3 is valid

Enter a password: Computer2NotYet
Computer2NotYet is not valid

Hand in Password1.java.

Problem 6 (80 points)

**Suggestion 1:** Try to solve this problem incrementally. That is, solve it one step at a time and make sure the current step works correctly before you add any code for the next step.

**Suggestion 2:** You will be asked to use random numbers in an array below. Using random numbers are convenient if you don’t want to generate arrays manually. However, testing your program with random numbers generated on the fly is very hard because you don’t know what numbers you will be getting in the array. So, while you are developing your program, I suggest that you use a small array with known elements that you include in the array yourself, hard-coded. After you finish debugging your program, switch it to a random number array with the specified size before you hand it in. Of course, you should test your final version to your satisfaction before you actually hand it in. You will most likely want to write a function that prints the elements of an array and use it as a debugging tool as you develop your program.

Write a program (Frequency.java) with the following methods:

1. (7 pts) Define a method named randomArray with two formal parameters: one integer indicating the size of the array to be created and the other indicating the upper-limit for the range of random numbers to be generated. If the second number is 15, then it would mean that the random numbers will be in the range of 0 and 14 inclusive. Remember the suggestion above? So, initially just have this method return a known hard-coded array (ignoring the parameters for now) for this part and make it work. Then, switch it to a random array later (after you are sure that everything in the rest of this problem works correctly with the known array).

2. (3 pts) In your main call randomArray with two actual arguments: 100 as the size and 15 as the upper-limit, and store the returned array into a local variable of your choice in main.
3. (7 pts) Define a method, named `arrSum` with one formal parameter of type array of integers, which returns the sum of the elements in the array.

4. (2 pts) In your `main` print the average of the numbers in the array that you obtained in step 2 above. To compute the average, you must use `arrSum` that you defined earlier. Print the average to the standard output device, i.e., screen. When you generate output to the screen, add some annotation so that the output will be meaningful.

5. (6 pts) Define a method named `contains` with two formal parameters: one an array of integers and the other an integer. The method returns true if the second argument is contained in the first array argument; returns false otherwise.

6. (2 pts) In your `main` call `contains` with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output.

7. (7 pts) Define a method named `contains2` with two formal parameters: one an array of integers and the other an integer. The method returns the index of the array where the first occurrence of the second argument is found, if found. If the second argument is not contained in the first array argument, it returns −1.

8. (2 pts) In your `main` call `contains2` with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output device.

9. (10 pts) Define a method named `countMultiplesOf` with two parameters: one an array of integers and the other an integer. This method returns the count of the integers in the array that are multiples of the second parameter. For example, 8 is a multiple of 2, but not a multiple of 3, and zero is a multiple of any number. I suggest that you define and use an auxiliary function that tests if a number is a multiple of another and returns a boolean value.

10. (2 pts) In your `main` call `countMultiplesOf` with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output.

11. (20 pts) Define a method named `buildHistogram` with one parameter: an array of integers. It then returns an array of integers, which is a histogram. You will build the histogram as follows. If the array that you obtained in step 2 contains 5 occurrences of 6, then 5 will be stored as the value in the array index 6 of the histogram array. What should be the size of the histogram array? Well, you should find the largest element in the source array and use that information to decide the size of the histogram array. You should consider adding another method, say `largest` that finds the largest element of an array and returns.

12. (2 pts) In your `main` call `buildHistogram` with the array obtained in step 2 above. Store the histogram that is returned into a local variable of your choice in `main`.

13. (10 pts) Define a method named `printHistogram` with one formal parameter of type array of integers. Call this method from the `main` with the histogram obtained in the previous step and let this method print the histogram in the following format:

   \[
   \text{i: <n>: <n of \#’s for the index i of the array. n is the element in the index i>}
   \]

   So, if the histogram array holds the following values:

   +------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+
   |  5  |  1  |  4  |  6  |  0  |  2  | 10  |  6  |            |
   +------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+------------------------+
   0  1  2  3  4  5  6  7

   Then, the histogram printed would look like this:
Note that the distribution in this histogram will show you how good the random number generator was. That is, how random are those numbers in the array.

Hand in your Frequency.java.

**Problem 7 (30 points)**

Write a program (Password2.java) with the following methods:

1. (8 pts) Define a method called `isUnique` that accepts an array of integers as a parameter and returns `true` if all the numbers are unique (i.e., no duplicates) and `false` otherwise. For example, if the array `a1` stores `[3, 8, 12, -48, 46, -3]`, the call `isUnique(a1)` would return `true` and if the array `a2` stores `[4, 7, 3, 9, 12, -47, 3]`, the call `isUnique(a2)` would return `false` since the value `3` appears twice.

2. (2 pts) Add code in `main` to test your implementation of `isUnique` at least with the examples given in the previous step.

3. (8 pts) Define a method named `randomUniqueArray` with two formal parameters: one integer indicating the size of the array to be created and the other indicating the upper-limit for the range of random numbers to be generated. If the second number is 15, then it would mean that the random numbers will be in the range of 0 and 14 inclusive. However, the elements in the resulting array should all be unique. That is, there should be no duplicate elements. You are allowed to create only one array in the process.

4. (2 pts) Add code in `main` to test your implementation of `randomUniqueArray` with two actual arguments: 10 as the size and 15 as the upper-limit, and store the returned array into a local variable of your choice in `main` and print the elements of the array.

5. (8 pts) Define a method named `buildPassword` that takes no parameter and returns a valid password as a `String`. A password is valid if it passes all of the following rules (Note: these rules are different from the ones used in Password1 above):
   - A password must have at least eight characters but no more than 12.
   - A password must alternate between letters and digits starting with a letter.
   - The only letters allowed are vowels.
   - The only digits allowed are odd digits.

   The following examples are valid passwords: "a5a3e903", "e1o5u3a9e5u3". But, the following are not: "a5a3e9", "a5a3e8", "a5a3e603".

6. (2 pts) Add code in `main` to test your implementation of `buildPassword` by printing a password that was generated. Print at least three valid passwords.

Hand in your Password2.java.