WESTERN UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS
WINTER 2022

Lab06 – Control Statements – Conditional and Loop Statements (ET2+KB4)

Instructor:
DR. QUAZI RAHMAN

Prepared by:
HIRA NADEEM

Acknowledgement:
DR. A. OUDA

Lead TA: Hira Nadeem, hnadeem5@uwo.ca

---

1 Goal

Java provides several kinds of loops for repeating a block of code while a certain condition is true. In this Lab, we will learn about three kinds of loops: while, do-while, and For-loop, and we will practice the switch conditional statement along with if-statements.

2 Resources

Lecture notes:	"Unit 5: Control Statements in Java – Conditional and Loop Statements."
Pre-recorded Lectures:	Session 1: if-statements, Session 2: switch, Session 3: while
IntelliJ Tutorials:	If-statements, switch, while

3 Deliverables

One zip file folder containing your project folder. Name it username_Lab3.zip where username is the beginning part of your email (e.g., Dr. Rahman's email address is qrahman3@uwo.ca, and his UWO username is: qrahman3).
Submission deadline: Submit your code by the end of your lab session. Prepare to demonstrate your understanding during the lab session.

4 Good Programming Practices

Below are several programming practices that should be followed to write and maintain quality codes. **Please note that you will be marked on all these components:**

- Include meaningful comments in your program. This will help you remember what each part of your code does, especially after long breaks from your work. Your TA will also appreciate understanding your code by going over your comments.

- Choose meaningful and descriptive names for your variables. There is a balance between descriptive names and code readability but always err on the side of descriptive.

- It is recommended that you follow the naming strategy for variables, methods and class names, as outlined in your course handout (Unit 2). Since the identifiers cannot contain white-space characters (spaces or tabs), words in an identifier should be separated by uppercase letters (myNewFunction(), myNewVariable). For class names, capitalize the first letter of each word in the name (e.g. MyClass, MatrixCalculator). For any constant name, use uppercase letters, and if needed, concatenate two or more words with underscore (e.g., MINIMUM_DRIVING_AGE)

- Initialize variables when declaring them. This means giving them initial values which are easier to track in your program if logical errors are present with your output.

- Indent and properly format your code. You should write your codes so that your teaching assistant can read and understand your code easily.

- **Include a header in each of your java class files**. The header should include your full name, UWO ID number, date the code was written and a brief description of the program in that file. Use any print statement to print the Header information on the screen based on the format below:

```
/*******************************************
* Add your full name here*
* Add your student number*
* Add Date*
* Give a brief description of the task *
*******************************************/

public class MyClass
{
    public static void main(String args[])
    {
      // Your code here
    }
}
```

5 Control Statements – Conditional and Loop Statements

The while loop:

In a **`while`** loop, the condition is tested first. The body is only executed if the condition is **`true`**, and then the process repeats.

The basic syntax of this loop is the following:

```
while (condition) {
    // do something in the body
}
```

The body of the loop may contain any valid Java statements, including conditional statements and even other loops (nested loops).

A loop becomes an infinite loop if the condition is always **`true`**. For example,

```
while (true) {
    // infinite loop
}
```

We will talk more about using infinite loops in later topics.

Example 1. The following loop prints consecutive integer numbers until reaching 5.

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

The output:

```
0
1
2
3
4
```

Example 2. The following code prints the English alphabet in a single line.

```
public class WhileDemo {
    public static void main(String[] args) {
        char letter = 'A';
        while (letter <= 'Z') {
            System.out.print(letter);
            letter++;
        }
    }
}
```

Here's what's happening:

The program takes the first character, 'A', and then repeats. If the letter is less than or equal to 'Z', the program goes to the loop body; inside the body, it prints the current value of letter and moves on to the next.

The output:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The do-while loop:

In a **`do-while`** loop, the body is executed first and only then is the condition statement tested. Therefore, the body will be always executed at least once.

The basic syntax of the loop is the following:

```
do {
    // do something
} while (condition);
```

The program below keeps reading integer numbers from standard input and outputting them. If 0 is entered, the program outputs it and stops.

```
public class DoWhileDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int value;
        do {
            value = scanner.nextInt();
            System.out.println(value);
        } while (value > 0);
    }
}
```

The input:

```
1 2 4 0 3
```

The output:

```
1
2
4
0
```

Note that a do-while loop may become infinite, just like a while loop.

The for-loop:

Sometimes, we need to repeat a block of code a certain number of times. Java provides the for-loop that is very convenient for this purpose. This loop is often used to iterate over a range of values or an array. If the number of iteration or the range borders are known, it is recommended to use the for-loop. If they are unknown the while-loop may be a preferable solution. This loop has the following base syntax:

```
for (initialization; condition; increment/decrement) {
    // do something
}
```

The explanations:

- the initialization statement is executed once when the loop begins, usually, here loop variables are initialized;
- the condition determines the need for the next iteration; if it's false, the loop terminates;
- the increment/decrement expression is invoked after each iteration of the loop; usually, here loop variables are changed.

Inside the loop's body, the program can perform any correct java statements. It can even contain other loops. The order of execution for any for-loop is always the same:

1. the initialization statement;
2. if the condition is false then terminate;
3. the body
4. the increment/decrement
5. go to the stage 2 (condition).

Let's write a loop for printing integer numbers from 0 to 9 in the same line.

```
int n = 9;
for (int i = 0; i <= n; i++) {
    System.out.print(i + " ");
}
```

The code outputs:

```
0 1 2 3 4 5 6 7 8 9
```

The variable declared in the initialization expression is visible only inside the loop including the condition, the body and the increment/decrement expression. The integer loop's variables is often called i, j, k or index.

Another example. Let's calculate the sum of integer numbers from 1 to 10 (inclusive) using the for-loop.

```
int startIncl = 1, endExcl = 11;
int sum = 0;
for (int i = startIncl; i < endExcl; i++) {
    sum += i;
}
System.out.println(sum); // it prints "55"
```

Sometimes it's needed to declare a variable outside the loop. it's possible as well

```
int i;
for (i = 10; i > 0; i--) {
    System.out.print(i + " ");
}
```

Infinite for-loop

The initialization statement, the condition, and the increment/decrement expression are optional, the for loop may not have one or all of them. Moreover, it's possible to write an infinite loop:

```
for (;;) {
    // do something
}
```

Nested loops

It's possible to nest one for-loop into another for-loop. For instance, the following code prints the multiplication table of numbers from 1 to 9 (inclusive).

```
for (int i = 1; i < 10; i++) {
    for (int j = 1; j < 10; j++) {
        System.out.print(i * j + "\t");
    }
    System.out.println();
}
```

It outputs:

```
1   2   3   4   5   6   7   8   9
2   4   6   8   10  12  14  16  18
3   6   9   12  15  18  21  24  27
4   8   12  16  20  24  28  32  36
5   10  15  20  25  30  35  40  45
6   12  18  24  30  36  42  48  54
7   14  21  28  35  42  49  56  63
8   16  24  32  40  48  56  64  72
9   18  27  36  45  54  63  72  81
```

6 Lab Assignment Question (20 marks)

This question asks you to create multiple methods for math operations. See your instructor or TA as soon as you are confused about anything on this question.

You will need to reuse the following methods for this lab:

- **myHeader** -> Outputs a header with your name, lab number and question number. Recycle the **myHeader** method from your previous lab.
- **myFooter** -> Outputs an exit message i.e. `*** Goodbye from YourFullName! ***`. Recycle the **myFooter** method your previous lab. Make sure the method does not take any arguments.

Use IntelliJ IDE to create a project named ***username*_Lab5.**

1. Create a package named **Lab6Q.**
2. Create a public class called **MyMethod** in the **Lab6Q** package.
   You will use this class in future labs and it will serve as your class where all of your public-static methods are defined, just like Math class where you can Math-methods, e.g., Math.pow().)
3. Add and define the following public-static methods including the driver method.
   - **Initially keep the driver method empty**. Define the other methods one at a time, and then check them inside a driver method:

```
public static void main(String[] args){
        //empty
}
```

4. After defining each method below, check each method by calling it from inside main().
   - **public static double myPow(double x, int y)** – this method will accept a real number **x** as base and an integer number **y** as power, and return $\mathbf{x^y}$. Instead of using the Math.pow() method, you will define and use this method (***Hint:*** this method has been solved in the course handout). **Note that your method should work for any positive or negative y values.**
     - Check this method by calling it from the driver method as:

```
double result = myPow(2, -2)
```

     and print the resulting value. The answer should give you 0.25.
     - If it works, add another method call for myPow() using 2 new values to confirm that your method is working. Then go to the next method. See your instructor if you have any question.
   - **public static double myFactorial(int n)** – this method will accept an integer number n (n>= 0), and return the factorial value of that number. The factorial of a number n is defined as:

     n! = 1 for n = 0, and n! = 1 x 2 x 3 x 4 x .....x n, for n>0.

Note: This part has been solved in the Class handout – Unit 5.

- Once done, check this method by calling it from the driver method.

  E.g.
  5! = 5 x 4x 3 x 2 x 1 = 1 x 2 x 3 x4 x 5 = 120,
  3! = 1x 2 x 3 = 6, etc.

- **public static double mySin(double x)** – this method will accept a real number (in radian value) and return the sine of that real number. Instead of using the Math.sin() method, you will define this method using the following **algebraic** expression:

$$\sin x \approx \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{(2n+1)}$$

  Where, n **(n >= 0)** and N **(N > 0)** are integer values**,** and **n! is defined as:**

  n! = 1 for n = **0**, and n! = 1 x 2 x 3 x 4 x .....x n, for n>0

  - Pick N = 20 to solve this problem.
  - For the factorial, you can use your myFactorial() method you defined before.
  - For the exponent, write the code by using Math.pow(). (You will replace it with your myPow() method after you check.)
  - Once you're done, check your method from main() by using a radian argument. You need to use radian arguments, because the above algebraic equation is written for radian values.
    - For example, 30 degree = Math.PI/6 radians; sin(Math.PI/6) = 0.50.
  - Once done checking, carefully replace Math.pow() with your myPow() method, and ensure that you get identical results.
  - ***Hint:*** A similar problem has been solved in the Unit 5 (Brief Version) course handout, slide 78. Also, this slide has been discussed in the pre-recorded lecture and during the class-lecture where it has been said that this kind of questions are asked in the exam.
  - See your instructor if you are not sure how to implement this.

- **public static double myCos(double x)** – this method will accept a real number (in radian value) and return the cos of that real number. Here instead of using Math.cos() method, you will define this method using the following **algebraic** expression:

$$\cos x \approx \sum_{n=0}^{N} \frac{(-1)^n}{(2n)!} x^{(2n)}$$

  - Pick N = 20 to solve this problem.

- Once again, for the factorial, you can use your myFactorial() method you defined before.
- Once again, for the exponent, write the code by using Math.pow(). (You will replace it with your myPow() method after you check.)
- Now, check this method by calling it from the driver method. Cross-check your method with **Math.cos ()** method to ensure that both are showing identical results.
- Now, carefully replace **Math.pow()** with your **myPow()** method call, and ensure that you get the identical result.

- **public static double myDegreeToRadian(double degree)** – this method will accept a value of degrees and convert it to radians before calling the mySin() and myCos() methods. Generally, the users enter the argument in degrees, and so, you need to provide the flexibility to the user by defining this method. **Note:** x degrees = x*Math.PI/180 radians.
  - Check this method in the main() the same way as the other methods you have defined to verify it works as expected.
- **public static void header and footer methods** – copy and paste these methods from the previous lab.

5. Once you are done checking all the methods in main(), cleanup the main() driver method by either commenting out all the test statements you used or by deleting them (it is good to maybe save them in a different file if you ever want to refer back to them later
6. Next, follow the below specifications for the driver method:
   a. Call your static header method using the **myHeader()** call with the required arguments. (You can simply call this question Q)
   b. Declare a character type variable called **choice**. This will be used for accepting the user's choice from the menu.
   c. Create an infinite loop with while: **while**(**true**) {....};
   d. Inside this loop, using any type of print statement, create the menu as shown below.

```
==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================
```

   e. Prompt the user to enter their choice.
   f. Using a **switch** statement, go to the matching menu option/case based on the user input. You will test each case and demo it to the TA.

g. Ask the user to input the required values from the keyboard, and then call the appropriate method (myPow(), myFactorial(), etc.) to do the specific operation. Check the sample output for an example of how it should look.

   - For example, for case 'a', your code should prompt the user to enter x and y values. Then, it should call the **myPow()** method. The returned value will be printed on the screen based on the formatting shown in the sample output.

h. ***Hint:*** In the switch-case statement, keep the **default** case as "Invalid choice". For case 'e', use **break**, and once you are out of the **switch**, use an **if-statement** to check whether the choice was 'e'. If so, break the loop with **break** statement as shown below:

   `if(choice == 'e') break;` It will break the infinite loop.

i. After coming out the loop, print your footer message using your **myFooter()** call.

Note: Work on each case at a time, and then run the code to test, and then go to the next case.

You are highly recommended to see your instructor, ahead of time, if you are stuck on any part of this lab.

SAMPLE OUTPUT

```
**************************************************
Quazi Rahman
Lab 6, Question 1
**************************************************
Testing Mathematical Functions:

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: a

POWER FUNCTION: Calculating x to the power of y.
Enter x: 2.5
Enter y: 2
2.50^2 = 6.25

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: a

POWER FUNCTION: Calculating x to the power of y.
Enter x: 2
Enter y: -2
2.00^-2 = 0.25

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: b

FACTORIAL FUNCTION: Calculating factorial of n.
Enter n: 5
5! = 120.00

==================
a: Power Function
```

```
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: b

FACTORIAL FUNCTION: Calculating factorial of n.
Enter n: 13
13! = 6227020800.00

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: c

SINE FUNCTION: Calculating sin(k) degree.
Enter k in degrees: 60
sin(60.00) degree = 0.87

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: c

SINE FUNCTION: Calculating sin(k) degree.
Enter k in degrees: 30
sin(30.00) degree = 0.50

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: d

COSINE FUNCTION: Calculating cos(m) degree.
Enter m in degrees: 45
cos(45.00) degree = 0.71
```

```
==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: d

COSINE FUNCTION: Calculating cos(m) degree.
Enter m in degrees: 60
cos(60.00) degree = 0.50

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: n

Invalid Choice!

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: 13

Invalid Choice!

==================
a: Power Function
b: Factorial Function
c: Sine Function
d: Cosine Function
e: exit
==================

Enter your choice: e

*** Goodbye from Quazi Rahman! ***
```

7 Submission

Zip the project file, name it ***username_Lab6.zip*** and submit in OWL by the end of your lab session.

If you have any questions about the Lab Handout, please contact the Lead TA, Hira Nadeem at hnadeem5@uwo.ca

8 Practice Problem

This practice problem will be covered in a future lab. Please give it a try.
Note: For any inquiries, please contact your instructor ahead of time. Attend Dr. Quazi Rahman's office hours to ask any questions. Ask your TA's for any help as well.

Part A:
This class will hold on to the characteristics of a Complex class in Rectangular/Cartesian form. You can copy-paste your ComplexNumber class from the previous lab just by copying it from its package to the package you create below in the IntelliJ IDE environment. Then, after right clicking on it, select Refractor →Rename, and rename it as RecComplexNum.

- Create a package and add a public class called **RecComplexNum**.
- Define this class using the following UML diagram, and associated specifications. You need to modify this class by adding the getter/accessor methods, highlighted in the UML diagram below:

RecComplexNum
-real: double -imaginary: double
+RecComplexNum () +RecComplexNum (re: double, im: double) +getReal():double +getImaginary():double +getMagnitude(): double +getAngle(): double +displayRecForm():void

Specifications of the RecComplexNum Class:

- **The constructor without argument** will assign all field values to zero.
- **The constructor with arguments** should accept the real and imaginary values as arguments and assign these values to the fields - real and imaginary.
- The getter/accessor methods **getReal()** and **getImaginary()** will provide access to the private fields by returning those.
- The **getMagnitude()** method is a helper method which will return the magnitude of a complex number. Magnitude of a complex number a+bi = $\sqrt{a^2 + b^2}$ (Use Math's **Math.pow()** and **Math.sqrt()** methods)

- The **getAngle()** method is a helper method which will return the angle of a complex number in degrees. Angle of a complex number a+bi = $\tan^{-1}\frac{b}{a}$ (Here, **i** is a symbol for the imaginary part of the number. Use Math's **Math.atan2(b, a)** that returns the angle in radians, and use **Math.toDegrees(x)** method to convert the radians to degrees. See Unit 2, Part 5: slide 34. All the trigonometric methods in Math class return the results in radians, and to get the result in degrees one must use **Math.toDegree(x)** method)
- The **displayRecForm()** method will display the complex number as shown in the sample output. For example, if the complex number is (a + bi), it should display a + bi with the corresponding a and b values. The displayed result should use up to 2 decimal places.

Part B:

Add a second public class called **PolarComplexNum** in the same package, and define this class using the following UML diagram, and associated specifications. This class will hold on to the characteristics of a Complex class in Polar form.

PolarComplexNum
-magnitude: double **-angle: double**
+PolarComplexNum () **+PolarComplexNum (mag: double, ang: double)** **+getRealValue(): double** **+getImaginaryValue():double**

Specifications of the PolarComplexNum Class:

- The constructor without argument will assign all field values to zero.
- The constructor with parameter should accept the magnitude and angle value (in degree) as arguments and assign these values to the fields - magnitude and angle.
- The **getRealValue ()** helper method will return the real term of a complex number with the help of the magnitude and angle (see the note below).
- The **getImaginaryValue ()** helper method will return the imaginary term of a complex number with the help of the magnitude and angle (see the note below).

Note: If the magnitude is *m* and angle is θ for a complex number in polar form, the real and imaginary terms ***a*** and ***b*** of the corresponding rectangular/cartesian form is **(*a*+ *b*i),** where ***a*** = ***m*** **cos** θ, and ***b*** = ***m*** **sin** θ.

Part C:

Add the driver class called **DemoComplexNumberYourFirstName** in the Lab6Q package, and define this class based on the following specifications. It will contain the **three public static methods and the driver method**.

- Define (it has been defined below for you) a public static method with the header:
  `public static RecComplexNum addComplexNumbers(RecComplexNum x, RecComplexNum y)`

This method will accept two **RecComplexNum** reference variables, and return a **RecComplexNum** type reference variable after adding the objects referred to by x and y. Please note that the complex numbers are added easily in the rectangular/Cartesian domain as shown in the last lab. In this method the real fields are added together, and the imaginary fields are added together, and then a new **RecComplexNum** reference variable is instantiated using these added results. This **RecComplexNum** reference variable is returned in the end. To save your time, this definition has been given below, which you can use in your code:

```
public static RecComplexNum addComplexNumbers(RecComplexNum x, RecComplexNum y)
{

/*Declaring a RecComplexNum reference variable res to hold on to the result of
the addition. This is instantiated with the added values of the fields referred to
by x and y*/

   RecComplexNum res = new RecComplexNum(x.getReal()+y.getReal(),
   x.getImaginary()+y.getImaginary());

   //Return the reference variable res
   return res;
}
```

- Define a public static method with the header:
  `public static RecComplexNum subtractComplexNumbers(RecComplexNum x, RecComplexNum y)`

  This method will accept two **RecComplexNum** reference variables, and return a **RecComplexNum** type reference variable after subtracting the objects referred to by **x** and **y**. Please note that the complex numbers are subtracted easily in the rectangular/Cartesian domain. In this method the real fields are subtracted from each other (**x.getReal() – y.getReal()**) , and the imaginary fields are subtracted from each other (**x.getImaginary() - y. getImaginary()**) and then these results are used to instantiate a new **RecComplexNum** reference variable. This **RecComplexNum** reference variable is returned in the end.

- Define (it has been defined below for you) a public static method with the header:
  `public static RecComplexNum divideComplexNumbers(RecComplexNum x, RecComplexNum y)`

  This method will accept two **RecComplexNum** reference variables, and return a **RecComplexNum** type reference variable after dividing the object referred to by **x** by the object referred to by **y**. Please note that the complex numbers are divided easily in the polar domain. In this method the magnitudes are divided by each other (**x.getMagnitude() / y.getMagnitude()**) , and the angles are subtracted from each other (**x.getAngle() – y. getAngle()**) and then these results are used to instantiate a new **polarComplexNum** type reference variable. Now that the return type is **RecComplexNum,** you need to declare a new **RecComplexNum** type reference variable and with the help of the **polarComplexNum** reference variable's

getRealValue() and **getImaginaryValue()** this is instantiated. This **RecComplexNum** reference variable is returned in the end.

```
public static RecComplexNum divideComplexNumbers(RecComplexNum x, RecComplexNum y){

/*getting the magnitude and angle values after the division operation in division
magnitude are divided from each other while the angles are subtracted from each
other*/

   double finalMag = x.getMagnitude()/y.getMagnitude();//magnitude after division
   double finalAngle =x.getAngle()-y.getAngle();//angle after division

/*declaring a PolarComplexForm type reference variable and instantiating it with
the final values from above*/

   PolarComplexNum pc = new PolarComplexNum(finalMag,finalAngle);

/*Since the return type is a RecComplexNum type reference variable, declare one,
and instantiate it with pc's (see above) real and imaginary value*/

   RecComplexNum res = new RecComplexNum(pc.getRealValue(),
   pc.getImaginaryValue());

   //Return RecComplexNum type reference variable
   return res;
}
```

- Define a public static method with the header:
  `public static RecComplexNum multiplyComplexNumbers(RecComplexNum x, RecComplexNum y)`

  This method will accept two **RecComplexNum** reference variables, and return a **RecComplexNum** type reference variable after multiplying the objects referred to by **x** and **y**. Please note that the complex numbers are multiplied easily in the polar domain. In this method the magnitudes are multiplied with each other (**x.getMagnitude()** * **y.getMagnitude()**) , and the angles are added to each other (**x.getAngle()** + **y. getAngle()**) and then these results are used to instantiate a new **polarComplexNum** type reference variable. Now that the return type is **RecComplexNum,** you need to declare a new **RecComplexNum** type reference variable and with the help of the **polarComplexNum** type reference variable's getRealValue() and getImaginaryValue(), this is instantiated. This **RecComplexNum** reference variable is returned in the end.

Specifications for the driver method:

a. Call your static header method using **MyMethod.myHeader()** call with the required arguments.
b. Declare the following variables / Reference variables
c. Two **RecComplexNum** type reference variables **x** and **y** and instantiate those with the help of the constructors with arguments so that **x** and **y** can be presented as **x = 1 - 2i**, and **y = -3 + 4i.**

d. One other **RecComplexNum** type reference variable for holding on to the results. You can instantiate it with the help of the constructor without argument (optional).
e. One character type variable called **choice** for accepting the user's choice
f. Print the x and y values as shown in the sample output.
g. Crete an infinite loop with while: **while**(**true**) {….};
h. Inside this loop create the menu as shown in the sample output, using any type of print statement.
i. Ask the user to enter their choice.
j. Using a switch statement, go to the matching case, and call the appropriate method (add, subtract, multiply, divide) to do the specific operation. Check the sample output to complete the code.
    - For example, for case 'a', your code should call the **RecComplexNum addComplexNumbers()** method, and then the returned value will be printed with the help of **displayRecForm()** method, and other print statements.
k. ***Hint:*** In the switch-case statement, keep default case as "Invalid choice". For case 'e', just use break, and once out of the switch, use an if-statement to check whether the choice was 'e', and if so, break the loop with break statement:
    if(choice == 'e') break; It will break the infinite loop.
l. After coming out the loop you can print your footer message using **MyMethod.myFooter().**

Note: Work on each case at a time, and then run the code to test, and then go to the next case.

m. To test your code with more data-points use the following website (there might be very slight difference in the decimal points while you are comparing, but most of the cases your result will be exactly the same): https://www.omnicalculator.com/math/cartesian-to-polar
n. Once the code is running successfully, change each of the Math.sin(), Math.cos(), Math.pow() methods with corresponding MyMethod.mySin(), MyMethod.myCos(), MyMethod.myPow() methods one at a time, and check whether your code gives you the identical result or not.

You are highly recommended to see your instructor, ahead of time, if you are stuck on any section of this lab.

```
Sample output:

*****************************************************
   Quazi Rahman
   Lab 6, Question 1
*****************************************************
You have entered the following two complex numbers x and y:
x = 1.00 - 2.00i, Magnitude: 2.24, Angle: -63.43 degrees
y = -3.00 + 4.00i, Magnitude: 5.00, Angle: 126.87 degrees

Complex Number Calculator:
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: t
Invalid Choice!
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: 6
Invalid Choice!
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: w
Invalid Choice!
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: a
x + y = -2.00 + 2.00i, Magnitude: 2.83, Angle: 135.00 degrees
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
```

```
==================
Enter your choice: v
Invalid Choice!
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: b
x - y = 4.00 - 6.00i, Magnitude: 7.21, Angle: -56.31 degrees
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: c
x * y = 5.00 + 10.00i, Magnitude: 11.18, Angle: 63.43 degrees
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: d
x / y = -0.44 + 0.08i, Magnitude: 0.45, Angle: 169.70 degrees
==================
a: Addition
b: Subtraction
c: Multiplication
d: Division
e: exit
==================
Enter your choice: e

*** Goodbye from Quazi Rahman! ***
```