Western Engineering

WESTERN UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS
WINTER 2022

Lab07 – Arrays (ET2+KB4)

Instructor:
DR. QUAZI RAHMAN

Prepared by:
HIRA NADEEM

Lead TA: Hira Nadeem, hnadeem5@uwo.ca

1 Goal

Arrays are a powerful tool in Java which act as containers to store multiple values of the same datatype in a single variable. In this lab, we will begin to practice defining and using arrays as well as continuing to practice loops, conditional statements, and methods.

2 Resources

Lecture Notes:	Unit 6: Arrays and Files, Unit 5: Control Statements Refresh Unit 4: Class Fundamentals in Coding, Unit 3: Methods in Java
Pre-recorded lecture videos:	Session 1: Introducing Array, Session 2: Copying arrays, and parameter passing between methods
IntelliJ Tutorials:	Arrays - Part 1, Arrays - Part 2, Arrays and Methods

3 Deliverables

One zip file folder containing your project folder. Name it username_Lab7.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 Lab Assignment Questions

You will be using methods from your **MyMethod** class.

1. Under the current project's source (src) file, copy-paste your MyMethod Class along with its package.
2. Now, you can import the above package that contains MyMethod Class to the destination package.
3. Import the MyMethod class using the import statement at the top. This statement will look like this: **import thePackageName.MyMethod**; or **import thePackageName.***;
   Here the package name should be the name of the package where you have created **MyMethod** class.

Use IntelliJ IDE to create a project named ***username_Lab7***.

5.1 Question 1 (5 marks)

In this question, we will simulate the Winter-Olympic judges' score by randomly generating five (5) scores between 7 and 10 (<7=score<10), and then get the final score by discarding the maximum and the minimum scores out of the five. This lab will give you a good practice in writing public static methods using arrays.

1. Create a package named **Q1.**
2. Import your **MyMethod** class. (Instructions at the top of page 3 of this handout)
3. Create a public class called **SimulateJudgesScore**.
4. Define a public static method with the header:
   `public static void populateArray(double[] anyName)`
   This method will populate the array referenced by the array-reference variable **anyName**. The values will be generated by the **Math.random()** method with the range: 7<=number<10 (see Unit 2.5, slide 36); we discussed the random number generation idea in the class on the 9th of March.
5. Define a public static method **displayArray()** with the header:
   `public static void displayArray(double[] anyName)`
   This method will display the array-content using the format shown in the sample output. Take a second look at the displayed arrays - the last "," is not there. You can implement it in many ways. One way to accomplish this is to use the backspace escape character (\b) discussed in Unit 2. For example, System.out.println("Hi there!23\b\b"); will remove 3 and 2 with the help of two \b characters while printing "Hi there!".
6. Define a public static method called **finalScore()** with the following header:
   `public static double finalScore(double[] anyName)`
   This method will have the following specifications:
   a. Declare an array reference variable to refer to a double-type array with the same size of the array referred to by **anyName**.
   b. Copy the content of **anyName** to this new array using **System.arraycopy()** method (see Unit 6 [Brief Version] slide 25-26)
   c. Find the maximum and minimum values from the **anyName** array. Once found, overwrite the maximum and minimum values in the other array you created with zero values. ***Hint***: you can use the index values of the maximum and minimum numbers and then assign zero to the corresponding indexed elements in the new array. See your instructor for ideas about it.
   d. Print these maximum and minimum values on the screen (see the sample output)

e. Print the content of the new array that has zeros in place of maximum and minimum values (see the sample output)
f. Now find the average of all the scores from the new array and return it.

7. Define a driver method using the specifications below:
   a. Call your **myHeader()** method you created in **MyMethod** class. The call will be **MyMethod.myHeader(arguments)**.
   b. Declare an array reference variable to refer to five (5) double type variables to store the scores from five (5) judges.
   c. Populate the array my calling he **populateArray()** method.
   d. Display the array content with by calling the **displayArray()** method (see the sample output)
   e. Display the final score of the athlete by calling the **finalScore()** method
   f. Call your **myFooter()** method from your **MyMethod** class. The call will be **MyMethod.myFooter().**

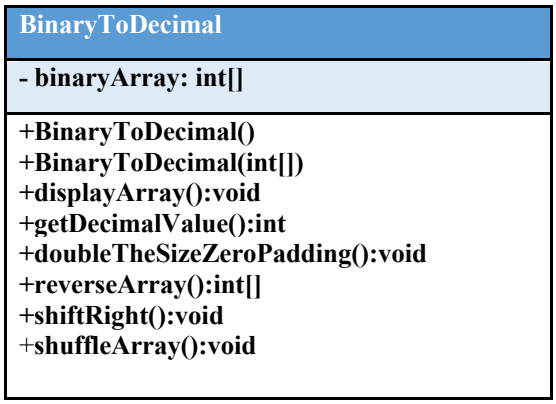
Sample Output

```
Note: Every run will result in different output. Here is the sample output of a
run:
**************************************************
   Quazi Rahman
   Lab 7, Question 1
**************************************************
Here are the scores from 5 Judges:
[8.67, 7.44, 9.88, 7.90, 8.45]
   Highest Score: 9.88
   Lowest Score: 7.44
Here are the scores after discarding the highest and the lowest scores:
[8.67, 0.00, 0.00, 7.90, 8.45]
   The final score is: 8.34

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

5.2 Question 2 (10 marks)

In this exercise, we will randomly generate a 4-bit number and store it in an integer array of size 4. Then we will carry out some simple operations on the 4-bit number. Since you will generate the 4-bit number randomly, it will result in different bit patterns during every run. Here, we will create two classes, **BinaryToDecimal** Class and **DemoBinaryToDecimal** (the driver) Class. Here are the specifications of the classes.



1. Create a package named **Q2.**
2. Import your **MyMethod** class. (Instructions at the top of page 3 of this handout)
3. Define a public class called **BinaryToDecimal** using the specifications below (see the UML diagram):
    a. a private integer type array field
    b. a constructor without argument (leave the definition empty)
    c. a constructor with an integer type array parameter.
    d. a public method called **displayArray()** to display/print the content of the array in the following format: [1 0 1 1]
    e. a public method **getDecimalValue()** will return the decimal equivalent value of the binary number. Here is the expression to find the decimal equivalent value for each bit: $weightOfTheBit\ x\ 2^{bitPosition}$.The bit position for the right most one (the least significant bit) is 0, the next one is 1 and so forth. To find the total decimal value we add all the individual values, e.g. $1010 = 1x2^3 + 0x2^2 + 1x2^1 + 0X2^0 = 10$. Use **MyMethod.myPow()** method while calculating power.
    f. a public method called **doubleTheSizeZeroPadding()** will double the size of the array (see Unit 6 [Brief Version] slide 27) and do zero padding on the left-hand side. You will work on this idea in the Digital Logic (ECE2277) course (ECE, SW, MSE program). e.g., if the array is containing a 4-bit number [1 0 1 1], this method will make it an 8-bit number [0 0 0 0 1 0 1 1] by doubling the size and padding zeros on the left-hand 4 bits - this will not change the decimal equivalent value of the resultant 8-bit number. You can use **System.arrayCopy()** method (see Unit 6 [Brief Version] slide 25-26)
    g. a public method called **shuffleArray()** will shuffle the array when it is called (see Unit 6 [Brief Version] slide 21)
    h. a public method called **reverseArray()** that will reverse the original array (see Unit 6 [Brief Version] slide 18)and return an array reference e.g. if the original array is [0 0 0 0 1 0 1 1],

the reversed array will be [1 1 0 1 0 0 0 0], whose reference will be returned when this member method is called.

i. a public method called **shiftRight()** will shift all the bits to right by one bit by adding the last bit in the array to the first bit position (see Unit 6 [Brief Version] slide 22). e.g. given [1 0 1 1] will result in [1 1 0 1] when **shiftRight()** method is called. As discussed in the class, you will work on this idea in in the Digital Logic (ECE2277) course for a hardware component called shift-register.

4. Define a driver method in a public class called **DemoBinaryToDecimal** using the specifications below:
   a. Call your **myHeader()** method you created in **MyMethod** class. The call will be **MyMethod.myHeader(arguments)**.
   b. Declare an integer type array-reference variable and instantiate with a size 4. Randomly populate this array with 0's and 1's using **Math.random()** method (see Unit 2.5, slide 36)
   c. Declare a **BinaryToDecimal** type reference variable and instantiate it with the array reference variable you prepared above.
   d. Display the 4-bit number as shown in the sample output by using the **displayArray()** method. For any questions on this, see your instructor.
   e. Display the corresponding decimal value as shown in the sample output by using the **getDecimalValue()** method.
   f. Double the size of the array using the **doubleTheSizeZeroPadding()** method, and display the content using **displayArray()** method as shown in the sample output. Also, print the decimal equivalent value which should be same as the previous one
   g. Declare a second **BinaryToDecimal** type reference variable and instantiate it with the array reference variable after reversing the above array. Display this content and then display the decimal equivalent value as shown in the sample output
   h. Shift the bits right by using **shiftRight()** method. Display the content and the corresponding decimal value.
   i. Now shuffle the array using **shuffleArray()** method. Display the content and the corresponding decimal value.
   j. Call your **myFooter()** method from your **MyMethod** class. The call will be **MyMethod.myFooter().**

Sample Output

```
Note: The output will be different for each run. Here is the sample output from a
run:

**************************************************
   Quazi Rahman
   Lab 7, Question 2
**************************************************
The 4 bit number: [1 0 0 1]
The corresponding decimal value is: 9
The 8 bit number, after the size is doubled and zero padded: [0 0 0 0 1 0 0 1]
The corresponding decimal value is: 9
The 8 bit number after reversing the array: [1 0 0 1 0 0 0 0]
The corresponding decimal value is: 144
The 8 bit number after the shift right operation: [0 1 0 0 1 0 0 0]
The corresponding decimal value is: 72
The 8 bit number after the shuffling operation: [0 0 1 0 0 0 0 1]
The corresponding decimal value is: 33

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

5.3 Question 3 (15 marks)

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**.

1. Create a package and add a public class called **RecComplexNum**.
2. Import your **MyMethod** class. (Instructions at the top of page 3 of this handout)
3. 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**

Note: there are two new methods in this RecComplexNum class.

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 the Java Math class methods Math.pow() and Math.sqrt())
- The **getAngle()** method is a helper method which will return the angle of a complex number in degrees. Angle of a complex number $a + bi = \tan^{-1}\frac{b}{a}$, where ***i*** is a symbol for the imaginary part of the number. (Use the Java Math class method 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 the Java 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 $(a + bi)$, where $a = mcos\theta$nd $b = msin\theta$.

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. 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 **myHeader()** method you created in **MyMethod** class. The call will be **MyMethod.myHeader(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. Create 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 by calling your **myFooter()** method from your **MyMethod** class. The call will be **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. **IMPORTANT: 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 on next page)

```
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! ***
```

6 Lab Assignment Questions

Zip the project file, name it ***username_Lab7.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