WESTERN UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS
WINTER 2022

Lab08 – Part 2: Inheritance and Polymorphism (ET2+KB4)

Instructor:
DR. QUAZI RAHMAN

Prepared by:
HIRA NADEEM

Lead TA: Hira Nadeem, hnadeem5@uwo.ca

1 Goal

In this lab, we will explore inheritance (method overriding, abstract classes) and polymorphism (interfaces). Inheritance and polymorphism are powerful concepts in programming that are useful as they allow programmers to save time by making the code reusable.

2 Resources

Lecture Notes:	Unit 7: Inheritance and Polymorphism; refresh on all concepts from course
IntelliJ Tutorials:	Inheritance and Abstract Class, Abstract Class & Casting in Inheritance, Polymorphism, Interface

3 Deliverables

One zip file folder containing your project folder. Name it username_Lab8.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 during the Week of April 4 – 8, 2022. 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_Lab8***. This is Part 2 of 2, you will include Part 1 in this same project.

Note: For this part of the lab, you are highly recommended to work with your lab-partner, and if you have any confusion, see your instructor along you're your lab-partner ASAP.

5.1 Question 3 (10 marks)

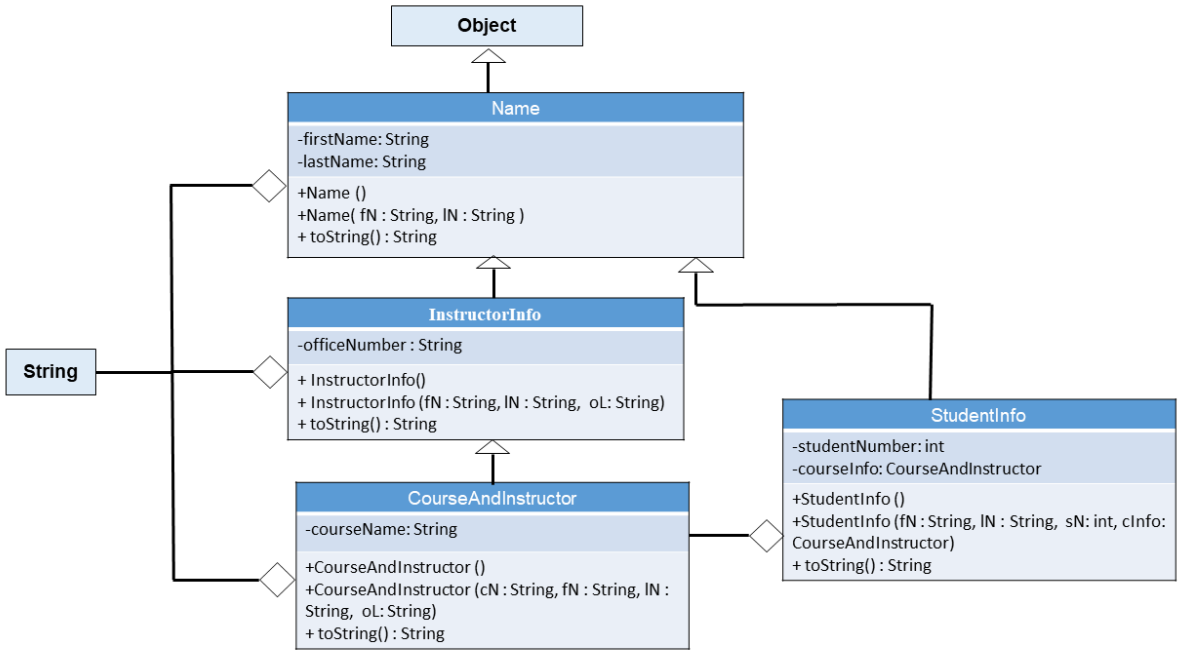
This coding exercise has been prepared to help you practice all the concepts from Unit 7. This is a very preliminary version of a program that will offer course-grade calculation for a student enrolled in the ES1036 course. The calculation involved all five assessments: Lab, Quiz, Bonus Quiz, Midterm and Final. Once this preliminary version is done, it can be generalized, in future, for any number of courses, any number of instructors, and any number of students. To keep this future goal in mind, we plan to accomplish this simple task using many simple classes.

The Exercise

In the ES1036 course, the following evaluations are carried out in assessing our learning outcomes:

- Lab Activity: 8 labs averaged over 15%
- Quiz Activity: 5 best quizzes averaged over 20%
- A midterm exam: 25%
- A final exam: 40%
- Bonus Quiz: 11 assessments averaged over 5%

Here are the UML diagrams of the class hierarchy we will be working on:



Let's prepare a package along with 6 different classes in the same project which will give us the option to calculate our grade for this course.

1. Create a package Called **Q3**. This package should be in the same project folder as the one you made for Lab 8 Part 1.
2. Create a class called **DemoCourseGradeByYourFirstName** with your driver method. You will use this later but also to check your classes as you go.
3. Create a public class called **Name** that has the following specifications:

Name
-firstName: String **-lastName: String**
+Name() **+Name(firstName : String, lastName : String)** **+ toString() : String**

   - Two private fields **firstName** and **lastName** of String Type
   - Two constructors, one *with argument* and one *without argument* (as shown in the UML Diagram). You can keep the body of the *constructor without argument* empty as: **public Name(){}**
   - Overridden **toString()** method that will print the first name followed by the last name on the screen (see the sample output). [This is done in Unit 7]
   - **Check that your Name class is working:** Create a **Name** type reference variable (e.g, anyName) in the driver method in the **DemoCourseGradeByYourFirstName** class, and instantiate it with parameters and then print the name by calling the toString() method as System.out.println(anyName). If your Name class is working fine, you can comment out the code inside the driver method using a block comment (/*- ----*/) and prepare the next class.

4. Create a second public class called **InstructorInfo** in the same package by inheriting the **Name** class (The inheritance chain: Object←Name←InstructorInfo) with the following specs.

InstructorInfo
-officeLocation : String
+InstructorInfo() **+InstructorInfo (fN : String, lN : String, oL: String)** **+toString() : String**

   - A private field called **officeLocation** of type String that contains instructor's office location (e.g., TEB263).
   - Two constructors one *with argument* and one *without argument* (as shown in the UML Diagram). You can keep the body of the *constructor without argument* empty. Remember to use the **super** keyword to access the super class's constructor.
   - Override the **toString()** method that will call super's **toString()** method [This is done in Unit 7] and add the office number to it to print instructor's name and office number together (see the sample output).
   - **Check that your InstructorInfo class is working:** Go back to the driver method in the **DemoCourseGradeByYourFirstName** class and check whether your **InstructorInfo** class is working or not in the same way as you checked your **Name** class. If your **InstructorInfo** class is working fine, you can comment out the code inside the driver method using a block comment (/*- ----*/) and prepare the next class.

5. Create a third public class called **CourseAndInstructor** in the same package by inheriting the **InstrcutorInfo** class (The inheritance chain: Object←Name←InstructorInfo←CourseAndInstructor) with the following specs.

CourseAndInstructor
-courseName: String
+CourseAndInstructor() **+CourseAndInstructor (cN : String, fN : String, lN : String, oL: String)** **+ toString() : String**

   - A private field called **courseName** of type String that contains course name (e.g., Programming Fundamentals).
   - Two constructors one *with argument* and one *without argument* (as shown in the UML Diagram). You can keep the body of the *constructor without argument* empty. Remember to use the **super** keyword to access the super class's constructor.
   - Override the **toString()** method that will call super's **toString()** method and add the course name to it to print instructor's name, office number and the course they are teaching.
   - **Check that your CourseAndInstructor class is working:** Go back to the driver method in the **DemoCourseGradeByYourFirstName** class and check whether your **CourseAndInstructor** class is working or not in the same way as you checked the previous classes. If your **CourseAndInstructor** class is working fine, you can comment out the code inside the driver method using a block comment (/*- ----*/) and prepare the next class.

6. Create a fourth public class called **StudentInfo** by inheriting the **Name** class (The inheritance chain: Object←Name←StudentInfo) with the following specs.

StudentInfo
-studentNumber: int **-courseInfo: CourseAndInstructor**
+StudentInfo () **+StudentInfo (fN : String, lN : String, sN: int, cInfo: CourseAndInstructor)** **+toString() : String**

   - Two private fields: an integer type field called **studentNumber** to store the student number, and a **CourseAndInstructor**-type object-field called **courseInfo**.
     Take away: The use of second object-field demonstrates that **StudentInfo** class 'has a' relationship with **CourseAndInstructor** class, which is called aggregation as discussed in Unit 7.
   - Two constructors one *with argument* and one *without argument* (as shown in the UML Diagram). You can keep the body of the *constructor without argument* empty. While defining the constructor with argument, you can take the help of the given code in Unit 7. Remember to use the **super** keyword to access the super class's constructor.
   - Override the **toString()** method that will call super's **toString()** method to print student's name and add the **studentNumber** and finally add the field **courseInfo** (which will automatically call the **CourseAndInstructor's toString()** method) to print the student's name, followed by the student's number followed by the course name followed by the instructors name and office location. (See Unit 7 to get an idea how the overridden toString() method is written in the Course class).
   - **Check that your StudentInfo class is working:** Go back to the driver method in the **DemoCourseGradeByYourFirstName** class and check whether your **StudentInfo** class is working or not in the same way as you checked the previous classes. If your **StudentInfo** class is working fine, you can comment out the code inside the driver method using a block comment (/*- ----*/) and prepare the next class.

Take away: From the above implemented code, we can see the application of **Polymorphism**:

- If we call **toString()** method for a Name object, it will print firstName and lastName;
- if we call **toString()** method for a **InstructorInfo** object, it will print Instructor's firstName, lastName and officeLocation;
- if we call **toString()** method for a **StudentInfo** object, it will print student's full name, student number, the course-name they are taking, and the associated instructor's name with their office location.
- **Summary:** In this case, method overriding is resulting in **Polymorphism**.

We are almost there ……

7. Download the public class (from OWL) called **GradeActivity**, and add this to the Lab 8 project folder in IntelliJ. You are required to write comments for this given code. This class has the following specs as shown in the UML diagram.

GradeActivity
-score: double **-grade: double[]** **-outOf: int** **-percentScore: double**
+GradeActivity() **+GradeActivity(score: double, out of: int)** **+GradeActivity(grade: double[], out of: int)** **+getScore():double** **+getLetterGrade():String** **+toString():String**

   - A private field called **score** of type double, which stores the score of any assessment.
   - A private field called **outOf** of type int that stores the value out of which the score is earned
   - A double[] type private array-reference field called **grade** to refer to multiple grade components.
   - A private filed called **percentScore** of type double to store the score out of 100.
   - Three constructors as shown in the UML diagram.
   - An accessor method for the field **score**.
   - A helper method **getLetterGrade()** that returns the letter grade.
   - Overridden **toString()** method.
   - **Check that your GradeActivity class is working:** Go back to the driver method in the **DemoCourseGradeByYourFirstName** class and check whether your **GradeActivity** class is working or not in the same way as you checked the previous classes. If your **GradeActivity** class is working fine, you can comment out the code inside the driver method using a block comment (/*- ----*/) and prepare the next class.
8. Now it is time to demonstrate the whole code in the **DemoCourseGradeByYourFirstName** class. Here are the sequential specifications of the driver method in the class. Follow this sequence to complete the code:
   a. (After Import the **MyMethod** class, as you did before) Call your **myHeader()** method with the appropriate arguments.
   b. Declare a reference variable of **CourseAndInstructor** type and instantiate it with the course-name (Programming Fundamentals), Instructor's first name (Quazi), last name (Rahman), and Instructor's office location (TEB361)
   c. Declare a reference variable of **StudentInfo** type and instantiate it with Student's first name (your first name), last name (your last name), student number (your student number), and **CourseAndInstructor** type reference variable you instantiated in the previous step. After this, print the associated values with the help of the **toString()** method (see the sample output).
   d. Create a double-type array-reference (e.g., labG) and populate it with all the 8 lab grades (each out of 15) you received/expected to receive using Initialization list. Now create a **GradeActivity** type reference variable **avgLabG** and instantiate it with the **labG** array

reference variable and 15 (as 'out of' data). Print the Average Lab grade on the screen (see the sample output) with the aid of **toString()** method.

e. Create a double-type array-reference (e.g., quizG) and populate it with the best 5 quiz grades (each out of 20). Now create a **GradeActivity** type reference variable **avgQuizG** and instantiate it with the **quizG** array reference variable and 20 (as 'out of' data). Print the Average Quiz grade on the screen (see the sample output) with the aid of **toString()** method.
f. Now create a **GradeActivity** type reference variable **midtermG** and instantiate it with your midterm grade (you received out of 25) and 25 (as 'out of' data). Print the midtrem grade on the screen (see the sample output) with the aid of **toString()** method.
g. Now create a **GradeActivity** type reference variable **finalG** and instantiate it with your expected final exam grade (you may receive out of 40) and 40 (as 'out of' data). Print the expected final exam grade on the screen (see the sample output) with the aid of **toString()** method.
h. Create a double-type array-reference (e.g., finalScore) and populate it with all the 4 assessment grades from 4, 5, 6 and 7 above with the aid of the getter method of the **GradeActivity** class using Initialization list. Now, create a **GradeActivity** type reference variable **courseG** and instantiate it with the **finalScore** array reference variable and 100 (as 'out of' data). Print the course grade on the screen (see the sample output) with the aid of **toString()** method.
i. Create a double-type array reference variable (bonusG) and populate it with the 11 reading assessment grades (out of 5 each) using Initialization list. Now create a **GradeActivity** type reference variable **avgBonusG** and instantiate with the **bonusG** array reference variable and 5 (as 'out of' data). Print the Average Quiz grade on the screen (see the sample output) with the aid of **toString()** method.
j. Now create a **GradeActivity** type reference variable **reportedFinalG** and instantiate with the 'sum of the scores from 8 and 9' above, and 100 (as 'out of' data). Print the reported final grade on the screen (see the sample output) with the aid of **toString()** method.
k. Call your **myFooter()** method.

Sample Output (on the next page)

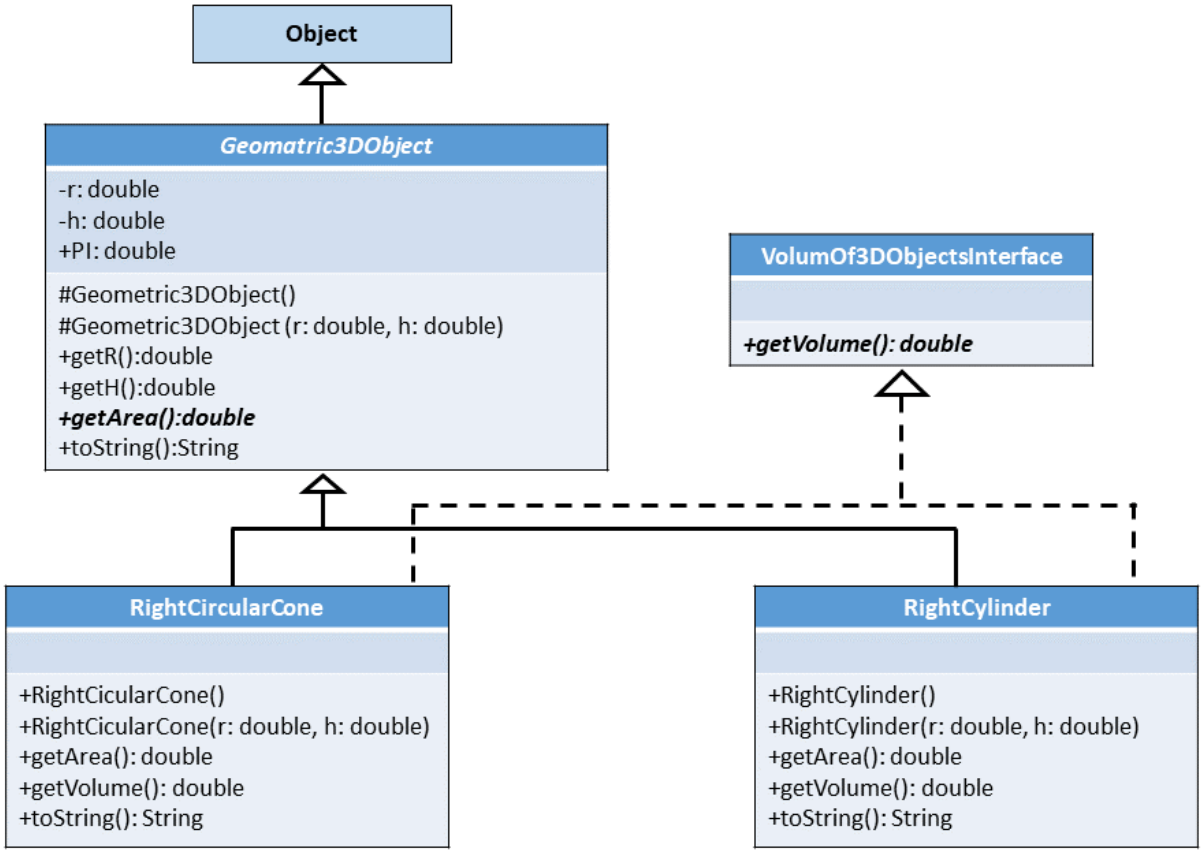
```
*****************************************************
   Quazi Rahman
   Lab 8, Question 3
*****************************************************
Name: Robert Kraichnan
Student Number: 251006789
Course Name: Programming Fundamentals
Instructor's Info:
Name: Quazi Rahman
Office Location: TEB 263
====================
   Score Card
====================
Lab Grade: 12.71 (out of 15), Letter Grade: A [84.76%]
Quiz Grade: 16.00 (out of 20), Letter Grade: A [80.00%]
Midterm Grade: 18.00 (out of 25), Letter Grade: B [72.00%]
Final Exam Grade: 31.00 (out of 40), Letter Grade: B [77.50%]
=================================================================
Course Grade: 77.71 (out of 100), Letter Grade: B [77.71%]
=================================================================
Bonus Quiz Grade: 4.04 (out of 5), Letter Grade: A [80.73%]
Reported Final Grade (with bonus): 81.75 (out of 100), Letter Grade: A [81.75%]
=================================================================

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

5.2 Question 4 (10 marks)

This coding exercise has been prepared to help you finish all the concepts from Unit 7. Here we will demonstrate our understanding of abstract class, interface, inheritance, and polymorphism with a very simple exercise.

The Exercise

In this exercise we will evaluate the volumes and surface areas of couple of 3D objects. See the UML diagram below:



Let's prepare a package along with 3 different classes and one interface in the same project based on the following instructions:

1. Create a package Called **Q4**. This package should be in the same project folder as the one you made for Lab 8 Part 1.
2. Create a public abstract class called **Geometric3DObject** (in IntelliJ IDE, create the class first and then add the modifier **abstract** once it is created; check the video tutorial in OWL) that has the following specifications:
    - Two private fields **r** (for radius) and **h** (for height) of double type.
      Note: Since we will be using an algebraic equation to find the volume and surface area, it is a good idea to use same variable names that represent radius and height in the expression. This will provide clarity when writing the code.

- One public field **PI** of type double. Here, we'll assign **Math.PI** to **PI** (PI = Math.PI) inside the class definition, then use just **PI** in the code. See your instructor if you are confused about this.
- Two protected constructors as shown in the UML Diagram. You can keep the body of the *constructor without argument* empty. (abstract class constructors are generally modified with protected modifier to restrict their access from other packages. Review Unit 7).
- A public abstract method **getArea().**
- Overridden **toString()** method that will print the **r** and **h** values on the screen (see the sample output).

3. Create a public interface called **VolumeOf3DObjectsInterface** (in IntelliJ IDE, right-click on the package and then select new→Java Class. In the pop-up window, jot down the name and select Interface - see the video tutorial in OWL)
   - Add a public abstract method **getVolume()** as outlined in the UML diagram. (Interface does not contain any definition of any method - see Unit 7)
4. Create a concrete (a regular one) public class called **RightCircularCone** by inheriting the abstract class **Geometric3DObject** and implementing the interface **VolumeOf3DObjectsInterface** ***(public class RightCircularCone extends Geometric3DObject implements VolumeOf3DObjectsInterface)***. It has the following specifications:
   - Two public constructors as shown in the UML Diagram. You can keep the body of the *constructor without* argument empty. For the other constructor you need to use the **super** keyword inside the definition (to understand why and how: Read Unit 7 / ask your instructor / check the video tutorial in OWL).
   - Now, implement the **getVolume()** and **getArea()** methods, which have already been declared inside the abstract class and inside the interface.
     Area of a Right Circular Cone: $A = \pi r(r + \sqrt{h^2 + r^2})$ (where **r** = radius and **h** = height)
     Volume of a Right Circular Cone: $V = \frac{\pi r^2 h}{3}$
   - Override the **toString()** method so that it can print the necessary information as shown in the sample output.
5. Create a second concrete (a regular one) public class called **RightCylinder** by inheriting the abstract class **Geometric3DObject** and implementing the interface **VolumeOf3DObjectsInterface**. It has the following specifications:
   - Two public constructors as shown in the UML Diagram. You can keep the body of the constructor without argument empty.
   - Now, implement the **getVolume()** and **getArea()** methods, which have already been declared inside the abstract class and inside the interface.
     Area of a Right Cylinder: $A = 2\pi r(r + h)$ (where **r** = radius and **h** = height)
     Volume of a Right Cylinder: $V = \pi r^2 h$
   - Override the **toString()** method so that it can print the necessary information as shown in the sample output.
6. Now it is time to demonstrate the whole code in the **DriverForAreaAndVolYourFirstName** class. Here you are required to create two public static methods inside the driver class and those are: printResult() and the driver method. The specifications follow:
   - The **printResult()** method:

- The header: public static void printResult(**Geometric3DObject** anyName);
- Inside the method, you need to print the result just by calling the **toString()** method.
- Hint: see Slide #61 and 70, Unit 7. If you are confused, please see your instructor.

7. Here are the sequential specifications of the driver method. Follow this sequence to complete the code:
   a. Call your **myHeader()** method with the appropriate arguments.
   b. Declare a double-type variable **r** and with the aid of the **Math.random()** method assign a real number to r within the range: 2<=r<3
   c. Now declare a double-type variable **h** and with the aid of the **Math.random()** method assign a real number to **h** within the range: 4<=h<7
   d. Declare a Geometric3DObject type reference variable called cone and instantiate it with the aid of the RightCircularCone type object and the variables r and h.
   e. Now, declare a second **Geometric3DObject** type reference variable called **cylinder**, and instantiate it with the aid of the **RightCylinder** type object and the variables **r** and **h**.
   f. Now call the **printResult()** method with appropriate argument to print the area and volume values for the cone and the cylinder. (Hint: Slide #61 and #70, Unit 7.)
   g. Call your **myFooter()** method.

Sample Output (on next page)

```
Note: Since r and h are generated randomly, every run will result in different
output
**************************************************
   Quazi Rahman
   Lab 8, Question 4
**************************************************
Given: Radius = 5.74 cm, Height = 2.29 cm
Cone's Volume: 79.02 cubic cm
Cone's surface area: 249.54 sq. cm

Given: Radius = 5.74 cm, Height = 2.29 cm
Cylinder's Volume: 237.06 cubic cm
Cylinder's surface area: 289.38 sq. cm

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

6 Lab Assignment Questions

Zip the project file, name it ***username_Lab8.zip*** and submit in OWL by the end of your lab session during the week of April 4-8, 2022. You will submit Part 1 and Part 2 in this same project folder.

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