Lab 03 Specification – A practical exercise to understand the use of Inheritance in Object-Oriented Progamming
50 points

Lab Goals

- Do a simple exercise on Inheritance.
- Master Is-a relationship development through a series of practical exercises.
- Practice UML Diagrams.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`

Learning Assignment

If you have not done so already, please read all of the relevant ”GitHub Guides”, available at the following website:
`https://guides.github.com/`
that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- **GT chapter 02, 2.1 and 2.2**

Assignment Details

Now that we have discussed some basics of object-oriented progamming together in the last few lectures, it is your turn. In this lab, you will practice a variety of programs to retain the knowledge of object-oriented programming that had been covered so far. This includes modifying one or more code files to implement a series of functionalities.

The lab has two parts. The first part is required to be completed by the end of the lab session. The second part is required to be completed within a one week time period. Please make sure to push your code file(s) and commit by the deadline provided in each of the following section(s).

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other’s program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc $\cdots$

At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL's to clarify if there is any confusion related to the lab and/or class materials.

Section 1: In-Lab Exercise (25 points)

Attendance is in general a required component in computer science class(es) and laboratory session(s). By attending these sessions regularly, students get an opportunity to interact with the Professor and the Technical Leaders(s). This interaction helps maximize the student's learning experience. Attendance is taken in the form of Mastery Quizze(s) and by checking the commit log of the student's In-Lab exercise submission. No student is allowed to take the Mastery Quizze(s) outside the laboratory room unless prior arrangements are done with the Professor. It is against the class honor code to violate the policy outlined above.

Part 01 - Attendance Mastery Quiz (5 points)

The Mastery Quiz is graded based on a Credit/No-credit basis. Students are required to complete the Mastery Quiz only in the laboratory room, which is in **Alden 101** before 3:30 PM. Submissions after 3:30 PM is not acceptable. Students who had completed the Mastery Quiz according to the policy outlined above will automatically receive (5) points towards their lab grade. The link to the Mastery Quiz is provided below:
`https://forms.gle/vwTpbX4n7ZHW1LCB7`

Part 02 - A Simple Exercise on Inheritance (15 points)

Students are required to complete this part only in the laboratory room, which is in **Alden 101**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is late submission accepted for this part, according to the late policy outlined in the course syllabus.

Write a Java program to show the concept of **Inheritance**.

1. **ADD** the following statement to all your code files including the file named `Room.java`, `RoomDriver.java`, and `BedRoom.java` .

   This work is mine unless otherwise cited - Student Name

2. The starter code is provided inside the lab repository in a file named, `Room.java`. The starter code has a minimal amount of code. Add the following members to the `Room` class.

   - A member variable named `length` of the integer data type. No access modifier should be specified. This means the default modifier.

- A member variable named `breadth` of the integer data type. No access modifier should be specified. This means the default modifier.
- A constructor with two formal parameters (x and y) of the integer data type. The constructor should initialize the value of the variables length and breadth to the value of x and y respectively.
- A member function named `area` with no formal parameters and of integer return type. The method should compute the area by multiplying the length and breadth. The method should return the area as an integer output.

3. The starter code is provided inside the lab repository in a file named, `BedRoom.java`. The starter code has a minimal amount of code. Add the following members to the `BedRoom` class.

   - Create a parent-child relationship between Room and BedRoom class. BedRoom class should be a child to the Room class. So how do you connect two classes and create an edge between them?
   - A member variable named `height` of the integer data type. No access modifier should be specified. This means the default modifier.
   - A constructor with three formal parameters (x, y, and z) of the integer data type. The constructor should invoke the parent constructor in the`Room` class with the actual parameters x and y respectively. So how do you call the parent constructor from the child constructor? Recall the use of **super** keyword discussed in class. Next, the constructor should initialize the value of the variables height to the value of z. Please recall the difference between actual and formal parameters from class discussion. A quick google search may be done to quickly recall our discussion points.
   - A member function named `volume` with no formal parameters and of integer return type. The method should compute the volume by multiplying the length, breadth, and height. The method should return the volume as an integer output.

4. The starter code is provided inside the lab repository in a file named, `RoomDriver.java`. The starter code has a minimal amount of code. Add the following members to the `RoomDriver` class.

   - Prompt the user to type in the room specifications, such as the length, breadth, and height. The user prompt may be done in a similar manner as we did in the previous labs. The user prompt may be implemented by greeting the user with a welcome message, prompting the user to type in the values, and using Scanner to parse the user input and store it in the variables.
   - Instantiate the class BedRoom by creating an object named `br`.
   - Invoke the `area` method using the object `br` and print the return value from the function on the console.
   - Invoke the `volume` method using the object `br` and print the return value from the function on the console.
   - Once all the requirements listed above are set up correctly, the output of the area and volume is expected to be printed as an output of the program. Note: The area and volume should be computed based on the user input for length, breadth, and height.
   - It is worth noting that the `area` method is inside the Room class. But we are accessing the method through the BedRoom class, which is the child of the Room class. That's the magic of Single Inheritance.

Part 03 - Pair Discussion (5 points)

Students are required to complete this part only in the laboratory room, which is in **Alden 101**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is no late submission accepted for this part unless prior arrangements are done with the Professor.

Pair up with at least one of your peers. Talk to your peer and discuss with them at least one idea of how to implement Multilevel Inheritance to the program developed in the earlier part. An example of multi-level inheritance is shown in

Figure 1. In the example, Shape has a child called Round and Round has a child called Oval and Circle. If there is more than one level in the hierarchy, it is said to be multi-level inheritance. Each person in the pair should discuss their own idea and write a short summary to explain their idea. The other person should critic the idea of their peer by providing them with at least one strong and weak point related to the idea. The summary should also include both the original idea and as well as the critic points from the peer. Name the summary file as **pd-summary.pdf**. Here **pd** refers to pair discussion. In order to pass the submission requirements, the file should be generated using a PDF format.

ADD the following statement to all your code files including the file named `pd-summary.pdf`.

This work is mine unless otherwise cited - Student Name

After completing all the parts in Section 01, it is acceptable for students to start working on Section 2 remotely. Although it is strongly recommended to be in the laboratory room till the end of the lab session, and work on the lab along with other colleagues.

Section 2: Take Home Exercise (25 points)

Students are recommended to get started with this part in the laboratory room, by discussing ideas and clarifying with the Professor and the Technical Leader(s). All the part(s) in this section is due in **one week** time. It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. The deadline for submitting the part(s) in this section is **12th Feb 2020, 8:00 AM**. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

There are several classic programming examples used to show inheritance, including an `Animal` hierarchy, a `Vehicle` hierarchy, and a `Clock` hierarchy. In this lab, we will be implementing a fourth classic example, the `Shape` hierarchy.

The hierarchy of classes that you will create is shown at the top of the next page in Figure 1. Detailed descriptions for each of the classes follow, and a final UML class diagram for each class is shown in Figure 2.

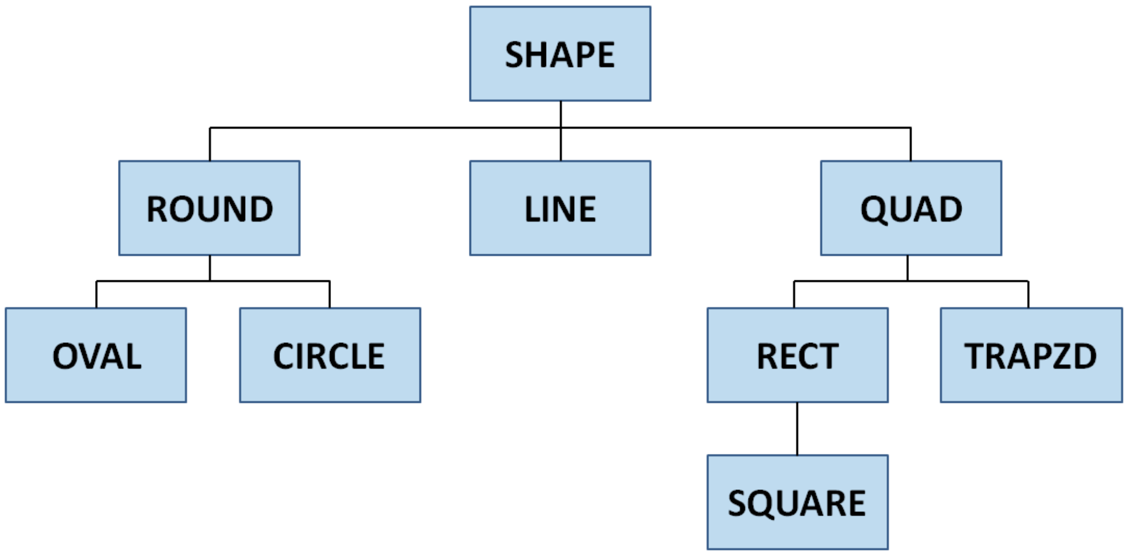

Figure 1: A summary view of the classes involved in this lab and the inheritance relationships between them.

Part 01 - Developing Shape Class (10 points)

To begin, we need to create a `Shape` class. For each of the shapes that we create, we will want to have the ability to calculate their perimeter and area; however, each function will have a different way to calculate perimeter and area, so we can't put the functions that calculate here. But we can create `float` variables for `perimeter` and `area`, as well as accessors and mutators to get (getters) and set (setters) those variables (accessors are optional; you can set the access control on the `perimeter` and `area` variables to `protected` to save a few lines of code). You may initialize `perimeter` and `area` to 0 here. Therefore, the `Shape` parent class should consist of two `private` (or `protected`) variables and four (or two) `public` functions.

Also included in this section is the ability for your classes to work with a `main` function called `ShapeStub` that is provided in the `starter-code` directory. Please look carefully at this code, as it is commented to show the order of the parameters that I expect for the constructors. I recommend starting this lab slowly, implementing one branch of the hierarchy at a time, testing with the `main` function by commenting and uncommenting relevant code lines in the main function. A printout of the output for `ShapeStub` is included after all of the shape descriptions.

One other thing to note is that the UML diagram in Figure 2 includes `private` variables with accessors and mutators to access and modify them. However, you are also permitted to skip the accessors and mutators, and simply set the access control on each variable to `protected` instead. Either implementation is fine, and it is your choice which path to follow. The `ShapeStub.java` code is expecting the `getArea()` and `getPerimeter()` functions to exist, however.

1. **ADD** the following statement to all your code files including the file named `Shape.java` and `ShapeStub.java`.

   This work is mine unless otherwise cited - Student Name

Part 02 - Developing Shape children, grand-children, and great grand-children. (15 points)

Figure 1 shows that we need to create three child classes that will inherit from `Shape`: `Round`, `Line`, and `Quad`. The `Round` and `Quad` classes have children of their own, which will actually implement the `calculateArea()` and `calculatePerimeter()` functions.

1. **ADD** the following statement to all your code files including the file named `Round.java`, `Line.java`, `Quad.java`, `Circle.java`, `Oval.java`, `Rect.java`, `Trapzd.java`, and `Square.java`.

   This work is mine unless otherwise cited - Student Name

Details for each of these implementations follow.

1. **Round Class**

   The `Round` class is an intermediate class in the hierarchy, located above the `Oval` and `Circle` classes but below the `Shape` class. We still calculate the area and perimeter of these shapes differently, so that behavior doesn't belong at this level. We can add one property here though. Both child shapes are round (hence the name of the class), so both will make use of π in calculating the perimeter and area. In this class, we can declare a `private double` for `pi` equal to 3.1415926535 (you can initialize it here), and an accessor to get that value. Therefore, the `Round` class should consist of one private variable and one public function.

2. **Line Class**

   The `Line` class is at the bottom of its hierarchy chain, so it will actually implement the two functions `calculateArea()` and `calculatePerimeter()`. We know that the area of a line is 0, so this one is trivial. We will think of a line as a rectangle with a height of 0, so the perimeter of the line is equal to double its length, or $2 * \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. We will also need to create a `Line` constructor to get those x_1, y_1, x_2, and y_2 variables in that order, as well as instance variables to store them (because these will belong to the `Line` class, they will not need accessors and mutators). Therefore, the `Line` class should consist of four private variables, a constructor, and two public functions.

3. **Quad Class**

   The `Quad` class is another intermediate class located between the `Shape` class and its actual shapes. Much like the `Round` class, we will calculate the area and perimeter differently for each of its children, so that behavior doesn't belong at this level. We can add variables for `width` and `height` at this level though, as well as the accessors and mutators to get and set them. Therefore, the `Quad` class should consist of two private variables and four public functions.

4. **Oval Class**

   The `Oval` class will be similar to what we needed to do for the `Line` class, as well as all subsequent classes in this section. We will need two variables for our oval, representing half of the major axis and minor axis of the oval (think of them as the radius of the oval at both its largest and smallest). We can call these variables `major` and `minor`. We will need a constructor to set these variables whenever a new `Oval` is created. Finally, we will need to implement the `calculateArea()` and `calculatePerimeter()` functions. The area of an oval is $pi * major * minor$, and the perimeter of an oval is approximated by $2 * pi * \sqrt{\frac{major^2 + minor^2}{2}}$. The `calculateArea()` and `calculatePerimeter()` functions should perform these calculations, and then set the values into the `area` and `perimeter` variables at the `Shape` class level. Therefore, the `Oval` class should consist of two private variables, a constructor, and two public functions.

5. **Circle Class**

   The `Circle` class is similar to the `Oval` class, in that we will be calculating the perimeter and area again. Instead of two variables, we only need one: `radius`, as well as a constructor to set it. Finally, the formula to calculate the perimeter of a circle is $2 * pi * radius$, and the formula to calculate to the area is $pi * radius^2$. The `Circle` class should consist of one private variable, a constructor, and two public functions.

6. **Rect Class**

   The `Rect` class is a special case because it is a parent to the `Square` class, but also has its own area and perimeter. We already have the `width` and `height` variables in the `Quad` class, so the `Rect` class only needs a constructor to set them and the functions to calculate perimeter ($2 * (width + height)$) and area ($width * height$). The `Rect` class will have a constructor and two public functions.

7. **Trapzd Class**

   The `Trapzd` class will get its height and base width from the `Quad` class as well, but it requires a second width (for the top) to calculate the area, and two side lengths to calculate the perimeter. Therefore, we need to create three additional private variables at this level, in addition to the constructor and the area and perimeter functions. The area of a trapezoid is $height * (\frac{width + otherBase}{2})$, and the perimeter is $width + otherBase + side1 + side2$. The `Trapzd` class will have three private variables, a constructor, and two public functions.

8. **Square Class**

   Last but not least is the `Square` class, which inherits from the `Rect` class. A square is just a simple case of a rectangle where the height and width are equal. Therefore, you can set up your constructor to use either `height` or `width` for your area and perimeter calculations. Assuming that we use `width`, the formula for the area of a square is $width^2$, and the formula for the perimeter is $4 * width$. The `Square` class will have a constructor and two public functions.

Note because the `Square` class inherits from the `Rect` class, and the `Rect` constructor requires two parameters, the first line in our `Square` constructor needs to convert from one to two parameters, something like `super(w, 0)` or `super(w, w)`.

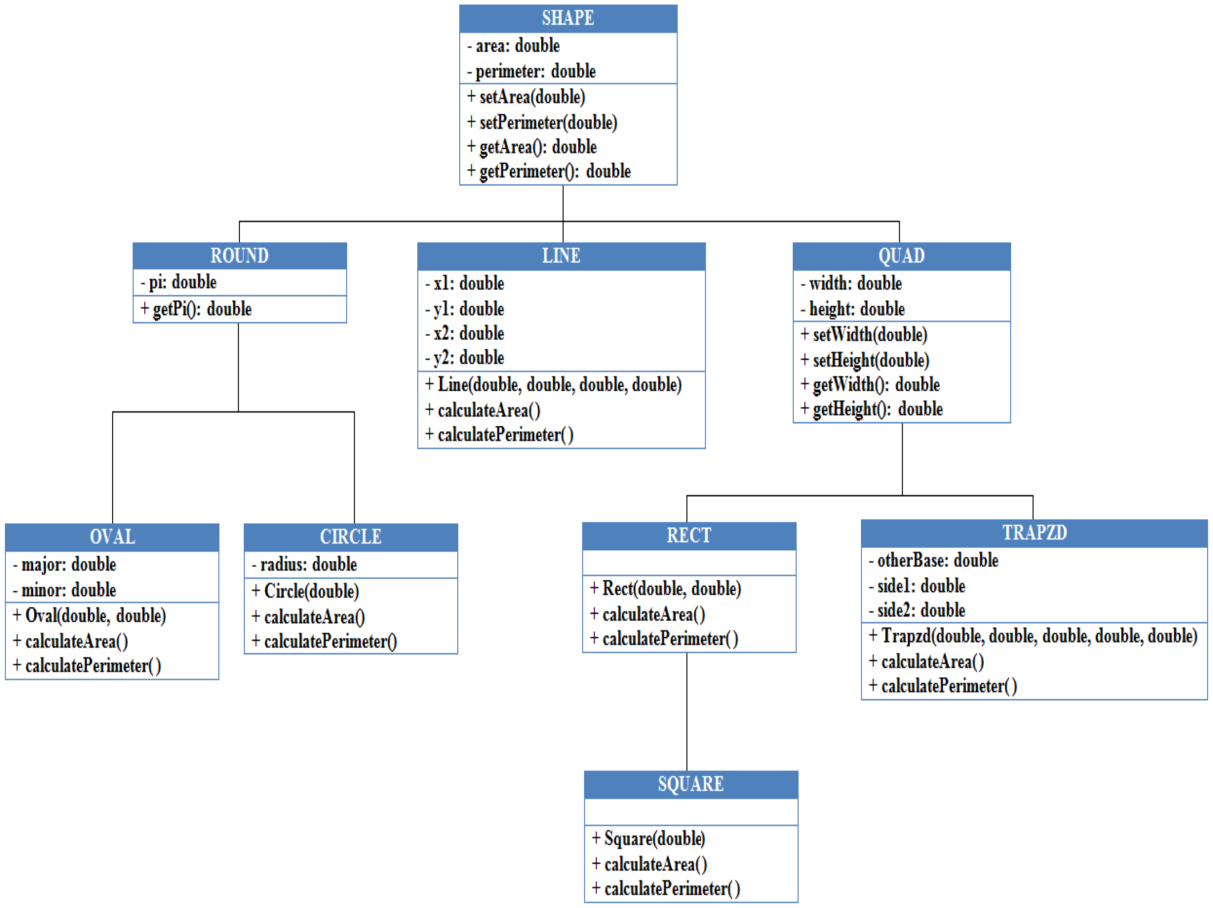

Figure 2: A detailed view of the classes involved in this lab and the inheritance relationships between them.

9. **Sample Output with ShapeStub**

```
TESTING LINE SHAPE
------------------
Line area = 0.0
Line perimeter = 16.1245154965971

TESTING OVAL SHAPE
------------------
Oval area = 226.194671052
Oval perimeter = 59.607529593072904

TESTING CIRCLE SHAPE
--------------------
Circle area = 30.974846926448603
Circle perimeter = 19.729201863980002
```

```
TESTING RECT SHAPE
------------------
Rect area = 36.0
Rect perimeter = 30.0

TESTING TRAPZD SHAPE
--------------------
Trapzd area = 16.0
Trapzd perimeter = 11.0

TESTING SQUARE SHAPE
--------------------
Square area = 2.617924
Square perimeter = 6.472
```

Submission Details

For this assignment, please submit the following to your GitHub repository before the **deadline** by using the link shared to you by the Professor:

1. Commented source code from the “Room”, “BedRoom”, and “RoomDriver” program.
2. Commented source code from the “Shape”, “ShapeStub”, “Round”, “Line”, “Quad”, “Oval”, “Circle”, “Rect”, “Trapzd”, and “Square” program.
3. A document containing the peer discussion points, named `pd-summary.pdf`.
4. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllbus. Make sure to add the statement ”This work is mine, unless otherwise cited.” in all your deliverables such as source code and PDF files. In your peer discussion summary file, please make sure to include your name and your partner’s name.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student’s submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.