

EIE3320: OBJECT ORIENTED DESIGN AND PROGRAMMING

LABORATORY 1 – 21/10/2023

ISLAM, AHNAF AL [20041569D] & DINESH GAUTAM [20040968D]

2.1 Introduction

The program output requests the user to select one of the shape options on the menu interface. If the user chooses one of the shape options, the menu will prompt the user to input the shape's geometric values. Based on the user's input, the program will calculate the shape's area and perimeter and display the shape proportionately to its geometric value. We can summarize our objectives as follows:

- Understand the principles of Object-Oriented design.
- Apply Java in Object-Oriented software development.
- Apply UML in Object-Oriented software modeling.
- Apply Object-Oriented approach to developing computer software.
- Learn independently and be able to search for the information required to solve problems.
- Present ideas and findings effectively.
- Work in a team and collaborate effectively with others.

To achieve our objectives, we must carefully consider the specifications of required classes, the hierarchy relationship between the classes, the program flow, and work distribution.

2.2 Methodology

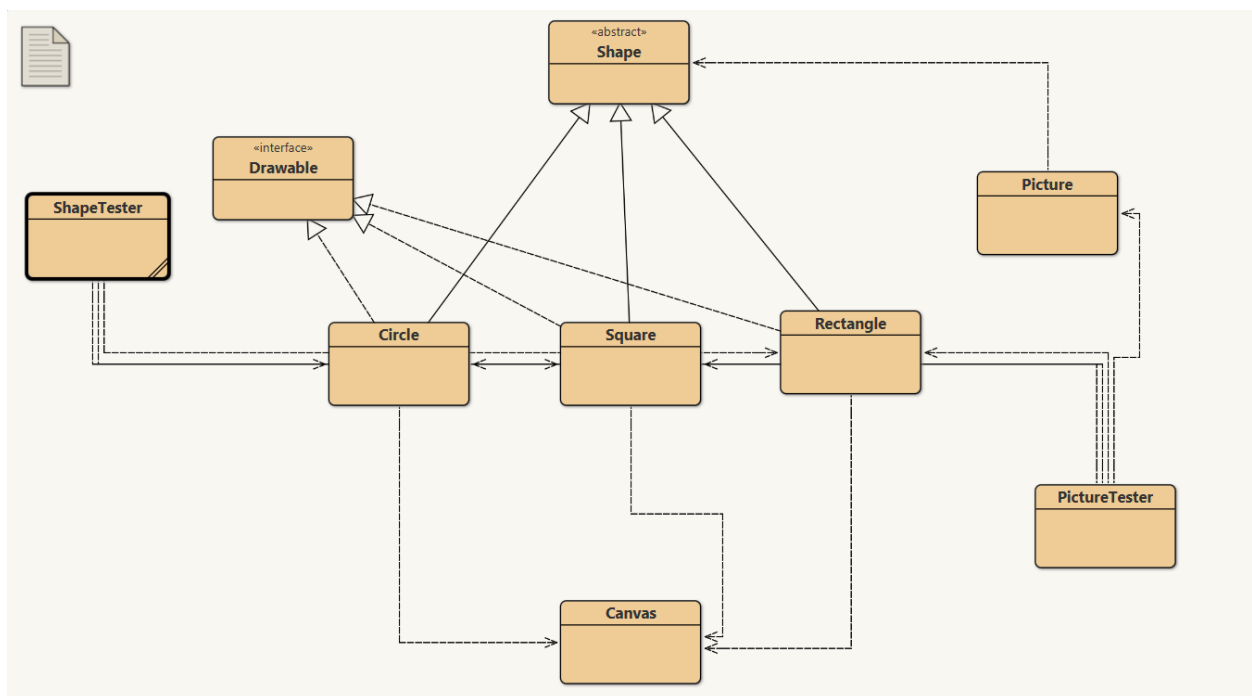


Fig.1. Final UML Diagram of the program

including o The specifications of the classes defined, and the public/private member functions/variables included o The flow of execution such as class diagram or flow chart.

Work distribution and Schedule:

Before we began working, we discussed the work distribution in-depth to ensure that the tasks were evenly distributed and suited to the individual's style. As such, the designed and implementation of the superclass, Shape, and its subclasses, Circle, Rectangle, and Shape, were done by Ahnaf Al Islam, including the ShapeTester class. The interface, Drawable, and the remaining classes, Picture and PictureTester, were designed by Gautam Dinesh, including modification of the provided Canvas class. Both teammates carefully planned and monitored the program flow to achieve our objectives. We set small milestones to reach to make effective progress. We planned to complete the superclass and its subclasses, interface, and two tester classes in the first week. The following week, we modified the Canvas class to integrate its functions within our program, allowing us to display shapes on the interface.

Program Structure:

The program defines one superclass, Shape, with three subclasses: Circle, Square, and Rectangle. In addition to these classes, we define four other types: ShapeTester, PictureTester, Picture, and Canvas, and an interface called Drawable.

The Shape class defines methods called readShape, which is used to read the shape information provided by the user. The method computeArea calculates the area of the shape using the given information. The method computePerimeter calculates the perimeter of the shape using the given information. The method displayShape outputs the calculated area and perimeter to the screen. The function draw works in conjunction with the drawable interface and the Canvas class to output a filled drawing of the shape. As mentioned earlier, these methods are inherited by the subclasses, which are overridden according to the specifications of each subclass. For example, the perimeter of a rectangle using the formula $= 2 * \text{length} * \text{width}$. This relationship is demonstrated in the UML diagram, where each subclass has arrows pointing to the Shape class and the drawable interface.

The class Picture defines a new ArrayList called shapes to store shape data. It also defines the method addShape to add the data to the array. It defines the method computeShape to calculate the area and perimeter of all the shapes stored in the array. The function listAllShapeTypes outputs the area and perimeter of all the shapes stored in the array. The function listSingleShapeType lists the area and perimeter of all the shapes that fit the criteria specified, which, in this case, is the shape type.

The class Canvas provides a simple way to create a canvas, draw shapes, and manipulate the canvas display. It achieves this by utilizing the javax.swing and java.awt packages for GUI components and graphics operations. The getCanvas() method is a static method that returns the instance of the Canvas class. It ensures that only a single instance of Canvas is created. The private constructor of the Canvas class takes the title, width, height, and background color of the canvas as parameters. It initializes the frame, canvas, and other instance variables. The setVisible() method sets the visibility of the canvas and brings it to the front of the screen when made visible. The draw() method adds an object with a shape and color to the canvas. It stores the shape and color in the shape hashmap and calls the redraw() method. The erase() method removes an object from the canvas and the shapes hashmap. It then calls the redraw() method. The setForegroundColor() method sets the foreground color for drawing shapes based on the provided color string. The wait() method introduces a delay in milliseconds, allowing for animations or timed operations. The redraw() method clears the canvas, iterates over the objects in the shapes hashmap, and draws each shape on the canvas. The erase() method clears the entire canvas by filling it with the background color. The CanvasPane class is a nested class that extends JPanel and is responsible for painting the canvas image on the screen. The ShapeDescription class represents a shape with its associated color. It has a draw() method that sets the color and fills the shape on the canvas.

The ShapeTester class provides a simple menu-driven interface for the user to choose different shapes. Based on the user's choice, the corresponding shape object is created. This class contains the main method, which is the program's entry point. The flag variable is declared as a static Boolean variable and set to true. This variable is used to control the execution of the program. The main method starts a while loop that runs as long as the flag variable is true. This loop allows the user to choose different shapes repeatedly until they choose to exit. Within the loop, a menu is displayed to the user using System.out.println(). The menu presents options to choose different shapes or to exit the program. An instance of the Scanner class is created to read the user's input from the console. The input.next().charAt(0) expression reads a single character from the user's input. The switch statement is used to perform different actions based on the user's choice:

- If the user chooses 'c,' a Circle object is created and assigned to the circle1 variable.
- If the user chooses 's,' a Square object is created and assigned to the square1 variable.
- If the user chooses 'r,' a Rectangle object is created and assigned to the rectangle1 variable.
- If the user chooses 'x,' the flag variable is set to false, which will exit the while loop and terminate the program.

- If the user enters any other character, an "Invalid command!" message is displayed.
- The while loop continues until the flag variable is set to false.

The PictureTester class provides the main function, which contains the code that creates a Picture object and adds several shapes (squares, circles, and rectangles). It then performs computations on the shapes, lists all the shape types present, and lists all the shapes of a specific type. Inside the main method, a Picture object called p is created using the default constructor. The Picture class is assumed to be a custom class that handles shapes and computations on those shapes. Several shape objects are made using the Square, Circle, and Rectangle classes and added to the p object using the addShape method. Two Square objects are created with side lengths of 2. Two Circle objects are created with radii of 3 and 4. Two Rectangle objects are created with lengths and widths of 5/6 and 7/8. The computeShape method is called on the p object. This method likely performs some calculations or computations on the shapes added to the p object. The listAllShapeTypes method is called on the p object. This method likely lists all the shapes present in the p object. The listSingleShapeType("Circle") method is called on the p object. This method likely lists all the shapes of a specific type (in this case, circles) present in the p object.

The Drawable interface defines a contract that specifies the draw() method, which must be implemented by any class that implements the Drawable interface. This allows for a consistent way of interacting with objects that can be drawn or rendered. Defining this interface means that any class that implements the Drawable interface must provide an implementation for the draw() method. The draw() method represents a behavior related to drawing or rendering an object. The specific implementation of the draw() method will vary depending on the class that implements the Drawable interface. Classes implementing the Drawable interface must provide their implementation details for the draw() method based on their specific requirements.

2.3 Program Testing

For product testing, we broke down our objectives into two parts:

1. Computing the area and perimeters using ShapeTester
2. Drawing

For objective 1, we broke down our objectives further. Firstly, we tested the menu interface to ensure it would match the program's requirements, making adjustments when necessary. We were able to achieve the output as below:

```
*****
* Please choose one the followings:      *
* Press 'c' - Circle                     *
* Press 's' - Square                     *
* Press 'r' - Rectangle                  *
* Press 'x' - EXIT                       *
*****
```

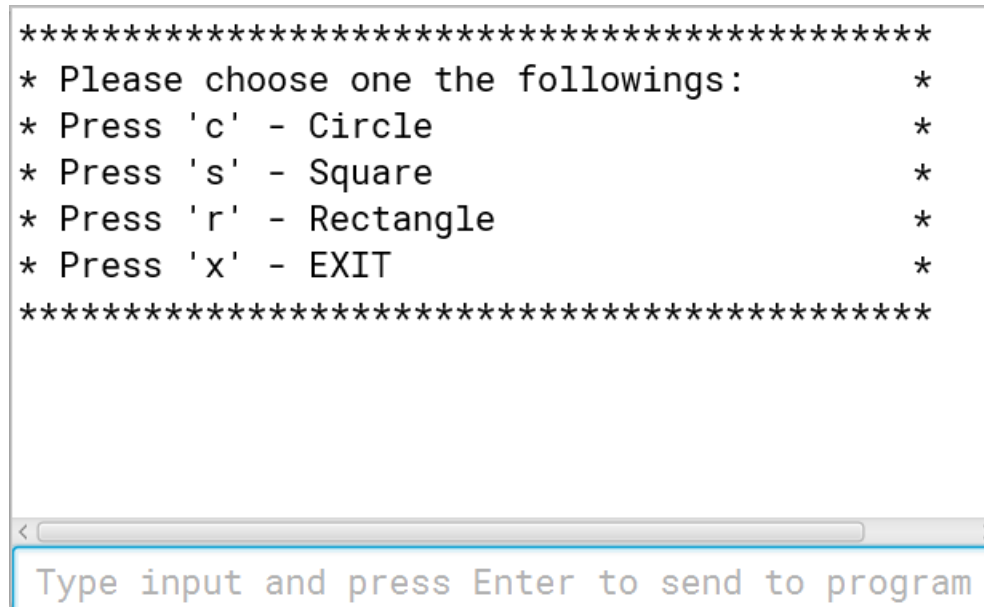
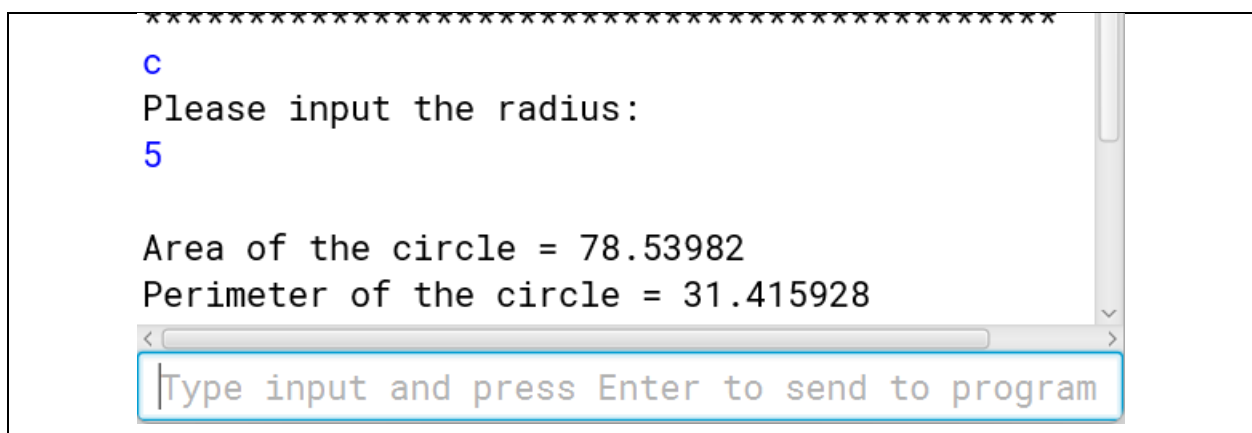


Fig.2.1 The menu interface of our program

Once we ensured the menu was flawless, we worked on implementing the area and perimeter functions for each of the subclasses – circle, rectangle, and square.

```
*****
c
Please input the radius:
5

Area of the circle = 78.53982
Perimeter of the circle = 31.415928
```



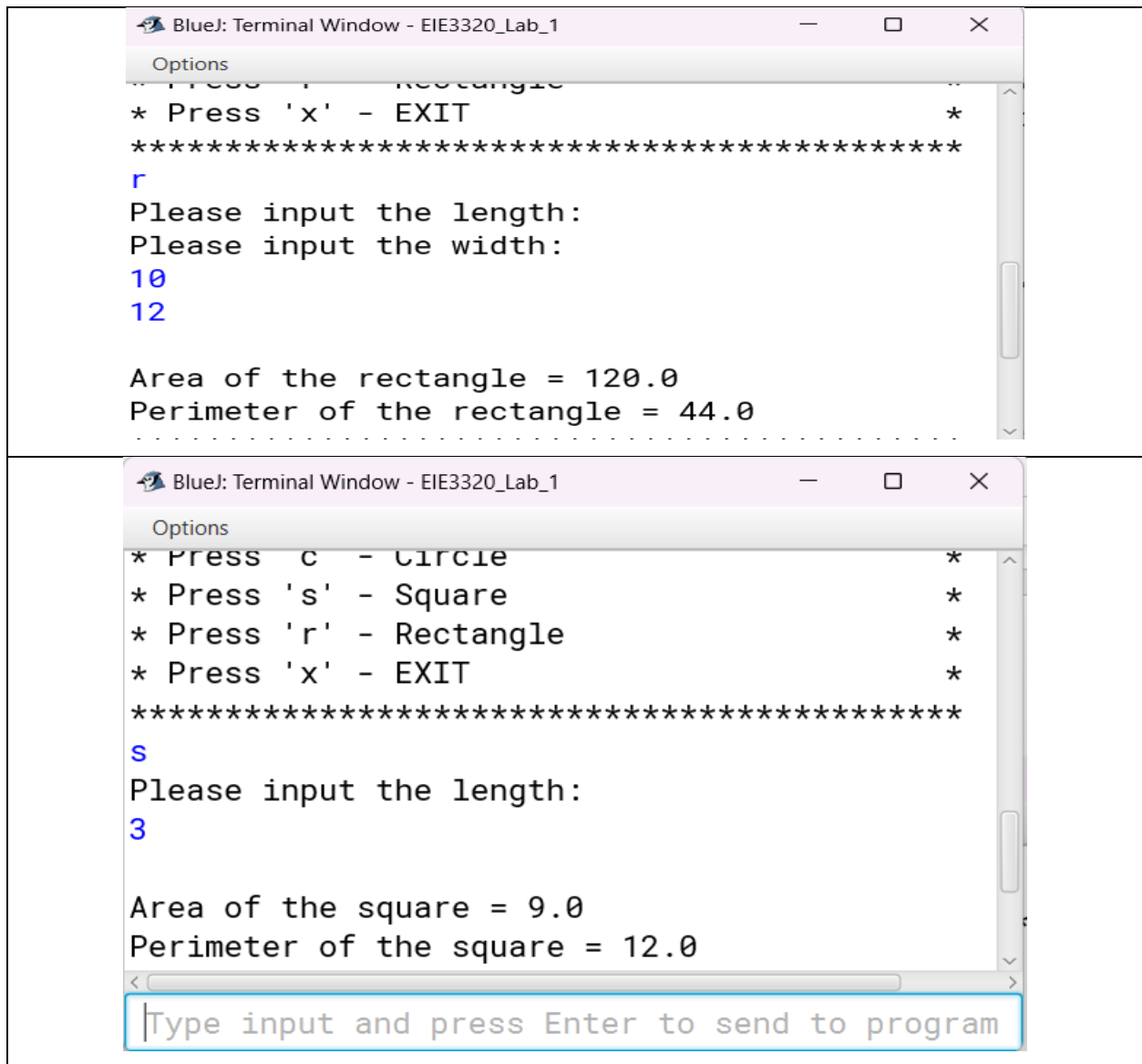


Fig.2.2 Output of area and perimeter functions for Circle, Rectangle, and Square

Lastly, we added a loop so the menu only accepts the desired inputs. After obtaining area and perimeter values, the user is redirected back to the main menu again. This continues until the user inputs 'x' to exit the menu. We obtained the output below:

```

*****
* Please choose one the followings:      *
* Press 'c' - Circle                     *
* Press 's' - Square                     *
* Press 'r' - Rectangle                  *
* Press 'x' - EXIT                       *
*****
a
Invalid command!
*****
* Please choose one the followings:      *
* Press 'c' - Circle                     *
* Press 's' - Square                     *
* Press 'r' - Rectangle                  *
* Press 'x' - EXIT                       *
*****
s
Please input the length:
3

Area of the square = 9.0
Perimeter of the square = 12.0
*****
* Please choose one the followings:      *
* Press 'c' - Circle                     *
* Press 's' - Square                     *
* Press 'r' - Rectangle                  *
* Press 'x' - EXIT                       *
*****
x
Can only enter input while your program is running

```

Fig.2.3 The program accepts desired values and loops until the user closes the menu

2.4 Conclusion

In conclusion, our program represents a successful endeavor in Object-Oriented software development using Java and UML modeling. We have achieved our stated objectives, which include understanding Object-Oriented design principles, applying Java for software development, utilizing UML for software modeling, and adopting an Object-Oriented approach to solving complex problems. Moreover, we have demonstrated our ability to work independently and collaboratively within a team. As reflected in the UML

diagram and the class descriptions, our program's structure shows a clear and well-organized design. We have effectively used inheritance and interfaces to create a hierarchy of classes, allowing for flexibility and extensibility in the future.

Overall, our Object-Oriented software development project has been a valuable learning experience, and it lays the foundation for future enhancements and innovations in the world of geometric shape manipulation and visualization.

2.5 Future Development

There are various intriguing potentials for future development. We can improve our program's adaptability by introducing additional shape options like triangles and pyramids. Furthermore, we could allow users to change the fill color of shapes, giving them greater creative freedom. Finally, users could have the option to choose whether a shape should be filled with color or just drawn, which could improve the program's functionality and user experience.