In Partial Fulfillment of the Requirements for the

CS 223 – Object-Oriented Programming

Geometric Shape Calculator

Prsesnted to:

Dr. Unife O. Cagas
professor

Presented by:
Jerette – Jean B. Baker

BSCS-2A
May 2024

Geometric Shape Calculator

Project Description:

My project is a console based application that can allow users to calculate the area and perimeter or circumference of circles, rectangles, and triangles. The Shape class is an abstract class that serves as a base class for the specific shapes (Circle, Rectangle, and Triangle). The calculate_area and calculate_perimeter methods are defined as abstract methods in the Shape class, and they are implemented in the specific shape classes.

The main function allows the user to enter the type of shape (circle, rectangle, or triangle) and its dimensions. The program then calculates the area and perimeter of the shape and prints the results. The user can quit the program by entering "quit" as the shape.

Objectives:

- To create a program that performs geometric shape calculations.
- To demonstrate understanding of object-oriented programming concepts.
- To provide a user-friendly interface for calculating geometric properties.

Importance and contribution of the project:

This geometric shape calculator program can be a valuable tool for students, teachers, and anyone who needs to perform basic calculations of areas and perimeters of shapes. It provides a user-friendly interface that can simplify these calculations and potentially improve the knowledge of students.

Engineers and designers can also this program to quickly calculate the area and perimeter of shapes in their designs, aiding in tasks such as architectural planning. It can be used as a handy tool for solving geometry related problems in various subjects, including mathematics, physics.

Four Principles of Object-Oriented Programming:

• **Encapsulation:** Bundles data (attributes) and methods that operate on that data together within a single unit called a class. This restricts direct access to the data, promoting data integrity and security.

```
5  #encapsulation
6  class Shape:
7      def __init__(self, units="centimeters"):
8          self.units = units
```

```
15 #encapsulation
16 class Circle(Shape):
17     def __init__(self, radius,
   units="centimeters"):
18         super().__init__(units)
19         self.radius = radius
```

```
26 #encapsulation
27 class Rectangle(Shape):
28     def __init__(self, length, width,
   units="centimeters"):
29         super().__init__(units)
30         self.length = length
31         self.width = width
```

```
38 #encapsulation
39 class Triangle(Shape):
40     def __init__(self, side1, side2, side3,
   units="centimeters"):
41         super().__init__(units)
42         self.side1 = side1
43         self.side2 = side2
44         self.side3 = side3
```

Encapsulation is demonstrated by encapsulating attributes (radius, length, width, side1, side2, side3) within their respective classes (Circle, Rectangle, Triangle). The Shape class encapsulates the concept of a geometric shape with its unit attribute (units) and abstract methods (calculate_area and calculate_perimeter). It restricts direct access to these methods.

• **Abstraction:** Exposes only essential details about an object, hiding the underlying implementation complexities. This allows users to interact with an object without needing to know its inner workings.

```
3  import math
4
5  class Shape:
6      def __init__(self, units="centimeters"):
7          self.units = units
8
9      #abstract
10     def calculate_area(self):
11         pass
12
13     #abstract
14     def calculate_perimeter(self):
15         pass
16
```

The Shape class hides the implementation details of calculating area and perimeter for specific shapes. It provides a simple interface through the abstract methods that

subclasses must implement. This allows users to interact with the program without needing to know the complexities behind of each shape's calculation.

 Inheritance: Enables creating new classes (subclasses) that inherit properties and behaviors from existing classes (super classes). This promotes code reusability and facilitates the creation of specialized objects.

```python
#parent class
class Shape:
    def __init__(self, units="centimeters"):
        self.units = units

    def calculate_area(self):
        pass

    def calculate_perimeter(self):
        pass
```

```python
#sub_class
class Circle(Shape):
    def __init__(self, radius, units="centimeters"):
        super().__init__(units)
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius ** 2

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius
```

```python
#sub_class
class Rectangle(Shape):
    def __init__(self, length, width, units="centimeters"):
        super().__init__(units)
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)
```

```python
#sub_class
class Triangle(Shape):
    def __init__(self, side1, side2, side3, units="centimeters"):
        super().__init__(units)
        self.side1 = side1
        self.side2 = side2
        self.side3 = side3

    def calculate_area(self):
        s = (self.side1 + self.side2 + self.side3) / 2
        return math.sqrt(s * (s - self.side1) * (s - self.side2) * (s - self.side3))

    def calculate_perimeter(self):
        return self.side1 + self.side2 + self.side3
```

The Circle, Rectangle, and Triangle classes inherit from the Shape class. They inherit the units attribute and abstract methods, but provide their own implementations for calculating area and perimeter specific to their shapes. This promotes code reusability and reduces redundancy.

• **Polymorphism:** Allows objects of different classes to respond to the same method call in different ways. This enables flexible and dynamic behavior in programs.

```
5  class Shape:
6      def __init__(self, units="centimeters"):
7          self.units = units
8  #polymorphism
9      def calculate_area(self):
10         pass
11
12     def calculate_perimeter(self):
13         pass
```

```
26 class Rectangle(Shape):
27     def __init__(self, length, width,
   units="centimeters"):
28         super().__init__(units)
29         self.length = length
30         self.width = width
31 #polymorphism
32     def calculate_area(self):
33         return self.length * self.width
34
35     def calculate_perimeter(self):
36         return 2 * (self.length + self.width)
```

```
38 class Triangle(Shape):
39     def __init__(self, side1, side2, side3,
   units="centimeters"):
40         super().__init__(units)
41         self.side1 = side1
42         self.side2 = side2
43         self.side3 = side3
44
45 #polymorphism
46     def calculate_area(self):
47         s = (self.side1 + self.side2 + self.
   side3) / 2
48         return math.sqrt(s * (s - self.side1) *
   (s - self.side2) * (s - self.side3))
49
50     def calculate_perimeter(self):
51         return self.side1 + self.side2 + self.
   side3
```

```
15 class Circle(Shape):
16     def __init__(self, radius,
   units="centimeters"):
17         super().__init__(units)
18         self.radius = radius
19 #polymorphism
20     def calculate_area(self):
21         return math.pi * self.radius ** 2
22
23     def calculate_perimeter(self):
24         return 2 * math.pi * self.radius
```

Polymorphism is demonstrated by treating objects of different subclasses (Circle, Rectangle, Triangle) uniformly as objects of the Shape class. The calculate_area and calculate_perimeter methods are defined as abstract methods in the Shape class. Subclasses (Circle, Rectangle, and Triangle) override these methods with their specific calculations. This allows for polymorphic behavior where the same method call (calculate_area) on different objects (circle, rectangle, triangle) results in different calculations based on the object's type.

Hardware and Software Used:

Hardware: Desktop computer and cellphone

Software: Online GDB compiler , Python programming language, pydroid and VS code.

Output (Screenshots) with Description:

There are three outputs that the code will generate, depending on the shape the user preferred to calculate.

• Circle:

```
Enter the shape (circle, rectangle, triangle): circle
Enter the radius of the circle in centimeters: 24
Area: 1809.5573684677208 square centimeters
Circumference: 150.79644737231007 centimeters

[Program finished]
```

Here, if the user chooses the circle shape, the program will ask the user to input the radius then the program automatically calculate the area and circumference of the circle.

• Rectangle:

```
Enter the shape (circle, rectangle, triangle): rectangle
Enter the length of the rectangle in centimeters: 13
Enter the width of the rectangle in centimeters: 14
Area: 182.0 square centimeters
Perimeter: 54.0 centimeters

[Program finished]
```

If the user input rectangle shape to be calculated, the program will ask the user to input the length and width then the program automatically calculate the area and perimeter of the rectangle.

• Triangle:

```
Enter the shape (circle, rectangle, triangle): triangle
Enter the length of side 1 of the triangle in centimeters: 1
4
Enter the length of side 2 of the triangle in centimeters: 1
8
Enter the length of side 3 of the triangle in centimeters: 2
6
Area: 119.81235328629515 square centimeters
Perimeter: 58.0 centimeters

[Program finished]
```

Here, if the user chooses the triangle shape, the program will also ask for the measure of the three sides of the triangle. It automatically generate the area and perimeter of the triangle.

code:

This is the full code of my program

```
import math

class Shape:
    def __init__(self, units="centimeters"):
        self.units = units

#abstract
    def calculate_area(self):
        pass

#abstract
    def calculate_perimeter(self):
        pass

class Circle(Shape):
    def __init__(self, radius, units="centimeters"):
        super().__init__(units)
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius ** 2

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius

class Rectangle(Shape):
    def __init__(self, length, width, units="centimeters"):
        super().__init__(units)
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

class Triangle(Shape):
    def __init__(self, side1, side2, side3, units="centimeters"):
        super().__init__(units)
        self.side1 = side1
        self.side2 = side2
        self.side3 = side3

    def calculate_area(self):
        s = (self.side1 + self.side2 + self.side3) / 2
        return math.sqrt(s * (s - self.side1) * (s - self.side2) * (s - self.side3))

    def calculate_perimeter(self):
        return self.side1 + self.side2 + self.side3

def main():
    while True:
        shape = input("Enter the shape (circle, rectangle, triangle): ").lower()
```

```
def main():
    while True:
        shape = input("Enter the shape (circle, rectangle, triangle): ").lower()

        if shape == "circle":
            radius = float(input("Enter the radius of the circle in centimeters: "))
            circle = Circle(radius)
            area = circle.calculate_area()
            perimeter = circle.calculate_perimeter()
            print("Area:", area, "square", circle.units)
            print("Circumference:", perimeter, circle.units)
            break

        elif shape == "rectangle":
            length = float(input("Enter the length of the rectangle in centimeters: "))
            width = float(input("Enter the width of the rectangle in centimeters: "))
            rectangle = Rectangle(length, width)
            area = rectangle.calculate_area()
            perimeter = rectangle.calculate_perimeter()
            print("Area:", area, "square", rectangle.units)
            print("Perimeter:", perimeter, rectangle.units)
            break

        elif shape == "triangle":
            side1 = float(input("Enter the length of side 1 of the triangle in centimeters: "))
            side2 = float(input("Enter the length of side 2 of the triangle in centimeters: "))
            side3 = float(input("Enter the length of side 3 of the triangle in centimeters: "))
            triangle = Triangle(side1, side2, side3)
            area = triangle.calculate_area()
            perimeter = triangle.calculate_perimeter()
            print("Area:", area, "square", triangle.units)
            print("Perimeter:", perimeter, triangle.units)
            break

        elif shape == "quit":
            break

        else:
            print("Invalid shape entered. Please enter 'circle', 'rectangle', or 'triangle'.")

            shape = input("Enter the shape (circle, rectangle, triangle): ").lower()

if __name__ == "__main__":
    main()
```

User guide:

Here's a step-by-step user guide for the geometric shape calculator program:

Step 1: Launch the Program

Run the geometric shape calculator program.

Step 2: Choose a Shape

The program will prompt you to enter the shape you want to calculate the area and perimeter (circumference) for (circle, rectangle, or triangle). Enter your selection in lowercase letters.

Step 3: Enter Shape Dimensions (if applicable)

- **Circle:** If you select circle, enter the radius of the circle in centimeters when prompted.
- **Rectangle:** If you select rectangle, enter the length and width of the rectangle in centimeters when prompted.
- **Triangle:** If you select triangle, enter the lengths of all three sides of the triangle in centimeters when prompted.

Step 4: View Results (or Exit)

- The program will calculate and display the area and perimeter of the chosen shape in square centimeters and centimeters, respectively.
- If you entered 'quit' in Step 2, the program will terminate.

Step 5: Repeat or Exit

- You can continue using the program to calculate the area and perimeter of other shapes by entering a new shape in Step 2.
- Enter 'quit' in Step 2 to exit the program.

Note:

Ensure to provide valid numeric inputs for shape dimensions.

References:

Oop tutorial: https://www.datacamp.com/tutorial/python-oop-tutorial?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=157156376311&utm_device=m&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=698229374827&utm_targetid=dsa-2218886984100&utm_loc_interest_ms=&utm_loc_physical_ms=9061352&utm_content=&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-go_9-na_10-na_11-na-may24&gad_source=1&gclid=CjwKCAjwrvyxBhAbEiwAEg_KgmXlqmHg534x8RhOQSnYyZJcMc4JFaZCHkdwF5An9A9BPUXqlEpnoxoCp-wQAvD_BwE

Shape calculator base program: https://github.com/DaniDiazTech/Object-Oriented-Programming-in-Python/blob/f0f20abe7ed0e46f7d9cbef0a5d28b0fbfe74121/shape_calculator/calculator.py

Four principles of Oop: https://dev.to/terrythreatt/the-four-principles-of-object-oriented-programming-in-python-1jbi

Four principles of Oop: https://www.educative.io/answers/what-are-the-four-pillars-of-oops-in-python