Laboratory Activity No. 7

Polymorphism

Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 02/22/2025
Section: 1A	**Date Submitted:** 02/26/2025
Name: Disomnong, Jalilah M.	**Instructor:** Engr. Maria Rizette H. Sayo

1. Objective(s):

This activity aims to familiarize students with the concepts of Polymorphism in Object-Oriented Programming

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the use of Polymorphism in Object-Oriented Programming
2.2 Implement an Object-Oriented Program that applies Polymorphism

3. Discussion:

Polymorphism is a core principle of Object-Oriented that is also called "method overriding". Simply stated the principles says that a method can be redefined to have a different behavior in different derived classees.

For an example, consider a base file reader/writer class then three derived classes Text file reader/writer, CSV file reader/ writer, and JSON file reader/writer. The base file reader/writer class has the methods: read(filepath="") , write(filepath=""). The three derived classes (classes that would inherit from the base class) should have behave differently when their read, write methods are invoked.

Operator Overloading:
Operator overloading is an important concept in object oriented programming. It is a type of polymorphism in which a user defined meaning can be given to an operator in addition to the predefined meaning for the operator.

Operator overloading allow us to redefine the way operator works for user-defined types such as objects. It cannot be used for built-in types such as int, float, char etc., For example, '+' operator can be overloaded to perform addition of two objects of distance class.

Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator on objects, the magic method __add__() is automatically invoked in which the meaning/operation for + operator is defined for user defined objects.

4. Materials and Equipment:

Windows Operating System
Google Colab

5. Procedure:

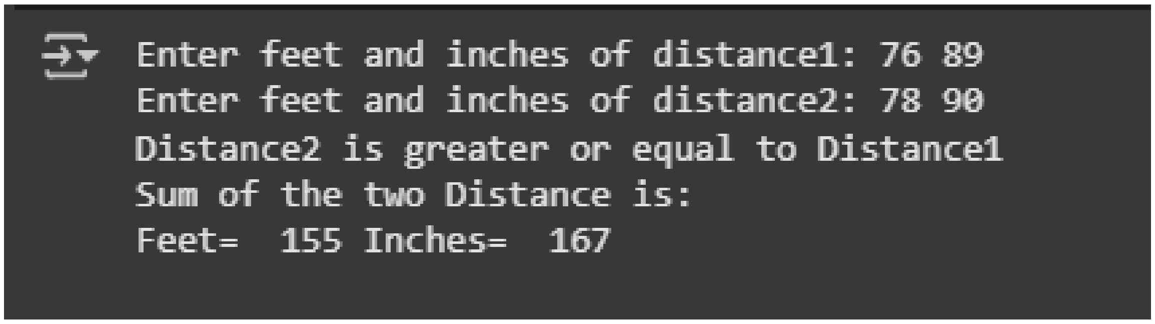
Creating the Classes

1. Create a folder named oopfa1<lastname>_lab8
2. Open your IDE in that folder.
3. Create the base polymorphism_a.ipynb file and Class using the code below:

Coding:

```
# distance is a class. Distance is measured in terms of feet and inches
class distance:
 def __init__(self, f,i):
 self.feet=f
 self.inches=i

# overloading of binary operator > to compare two distances
 def __gt__(self,d):
 if(self.feet>d.feet):
 return(True)
 elif((self.feet==d.feet) and (self.inches>d.inches)):
 return(True)
 else:
 return(False)

# overloading of binary operator + to add two distances
 def __add__(self, d):
 i=self.inches + d.inches
 f=self.feet + d.feet
 if(i>=12):
 i=i-12
 f=f+1
 return distance(f,i)

# displaying the distance
 def show(self):
 print("Feet= ", self.feet, "Inches= ",self.inches)

a,b= (input("Enter feet and inches of distance1: ")).split()
a,b =[int(a),int(b)]
c,d= (input("Enter feet and inches of distance2: ")).split()
c,d =[int(c),int(d)]
d1 = distance(a,b)
d2 = distance(c,d)

if(d1>d2):
 print("Distance1 is greater than Distance2")
else:
 print("Distance2 is greater or equal to Distance1")
d3=d1+d2
print("Sum of the two Distance is:")
d3.show()
```

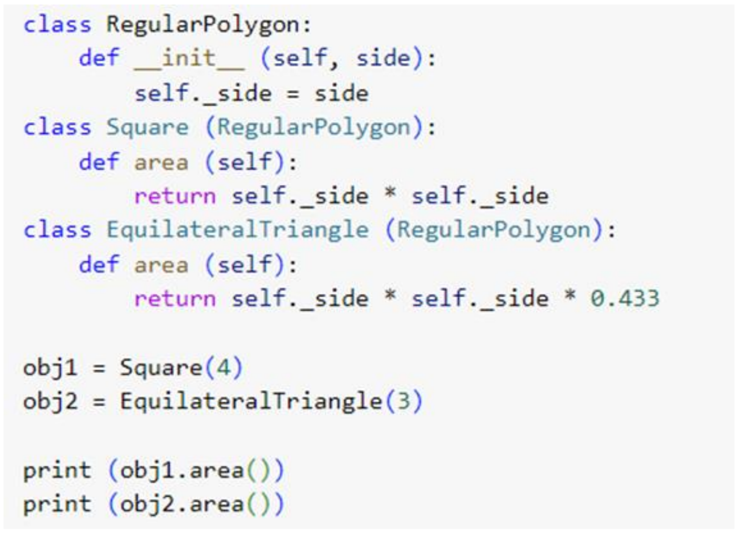
4. Screenshot of the program output:



```
Enter feet and inches of distance1: 76 89
Enter feet and inches of distance2: 78 90
Distance2 is greater or equal to Distance1
Sum of the two Distance is:
Feet=  155 Inches=  167
```

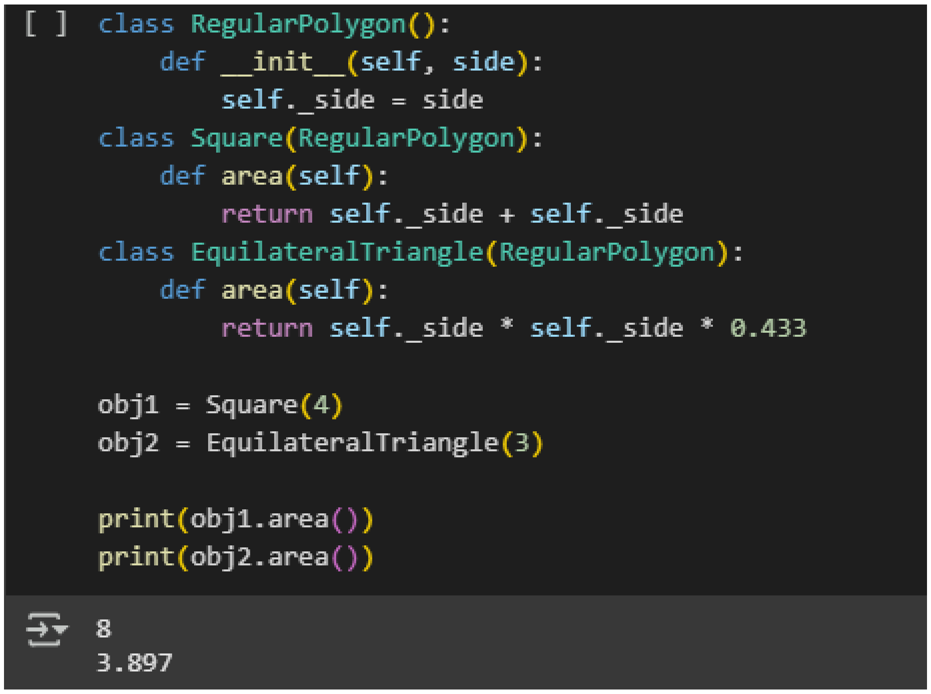
Testing and Observing Polymorphism

1. Create a code that displays the program below:



```
class RegularPolygon:
    def __init__ (self, side):
        self._side = side
class Square (RegularPolygon):
    def area (self):
        return self._side * self._side
class EquilateralTriangle (RegularPolygon):
    def area (self):
        return self._side * self._side * 0.433

obj1 = Square(4)
obj2 = EquilateralTriangle(3)

print (obj1.area())
print (obj2.area())
```

2. Save the program as polymorphism_b.ipynb and paste the screenshot below:



```
[ ] class RegularPolygon():
        def __init__(self, side):
            self._side = side
    class Square(RegularPolygon):
        def area(self):
            return self._side + self._side
    class EquilateralTriangle(RegularPolygon):
        def area(self):
            return self._side * self._side * 0.433

    obj1 = Square(4)
    obj2 = EquilateralTriangle(3)

    print(obj1.area())
    print(obj2.area())
```

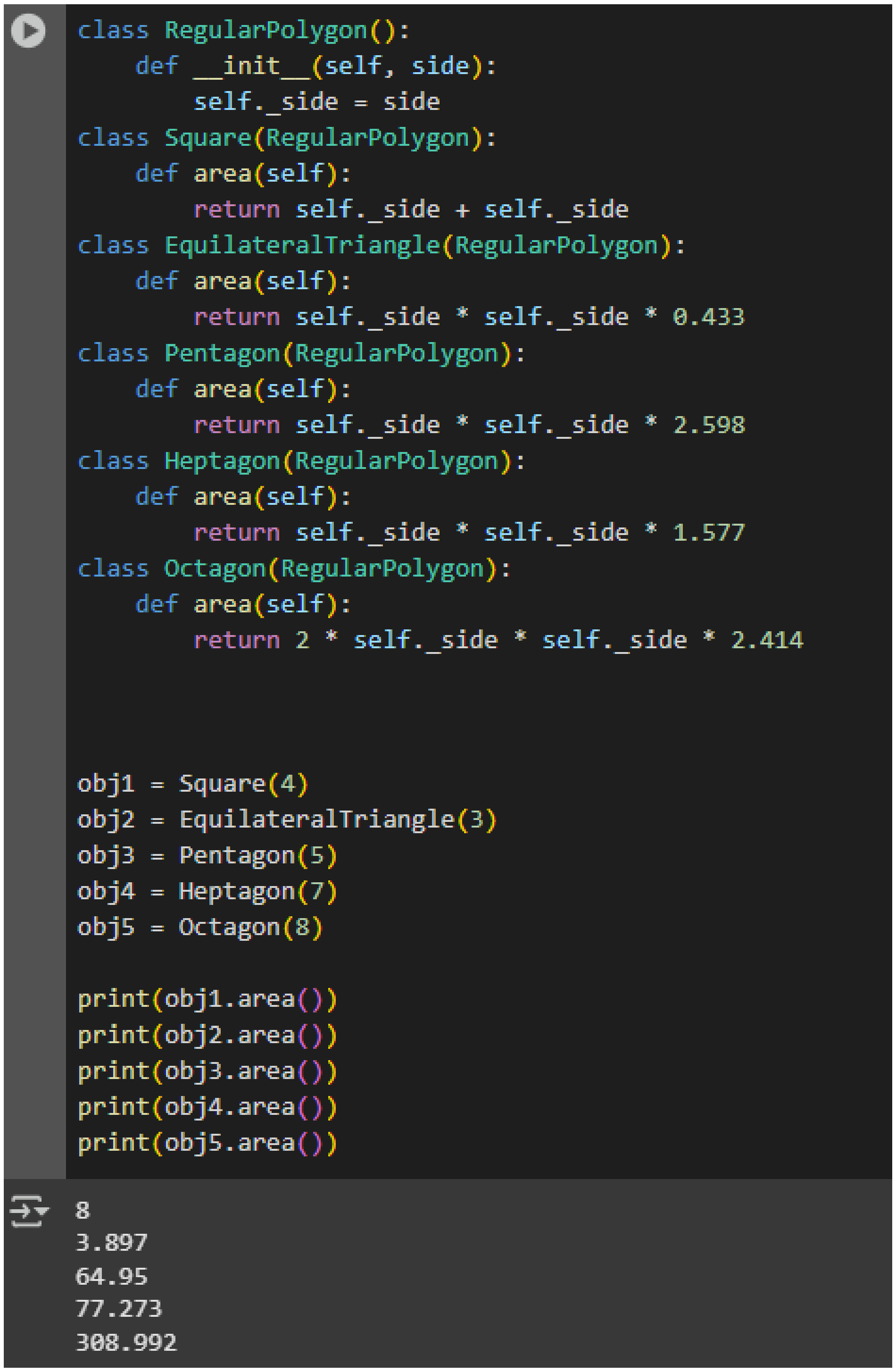
```
8
3.897
```

3. Run the program and observe the output.
4. Observation

   Refer to this link: https://colab.research.google.com/drive/1c8kM-GqYMOEDQWSiz9lF21EUZyz66XDu#scrollTo=u4jWwWYPDFEm

6. Supplementary Activity:

In the above program of a Regular polygon, add three more shapes and solve for their area using each proper formula. Take a screenshot of each output and describe each by typing your proper labeling.



```python
class RegularPolygon():
    def __init__(self, side):
        self._side = side
class Square(RegularPolygon):
    def area(self):
        return self._side + self._side
class EquilateralTriangle(RegularPolygon):
    def area(self):
        return self._side * self._side * 0.433
class Pentagon(RegularPolygon):
    def area(self):
        return self._side * self._side * 2.598
class Heptagon(RegularPolygon):
    def area(self):
        return self._side * self._side * 1.577
class Octagon(RegularPolygon):
    def area(self):
        return 2 * self._side * self._side * 2.414



obj1 = Square(4)
obj2 = EquilateralTriangle(3)
obj3 = Pentagon(5)
obj4 = Heptagon(7)
obj5 = Octagon(8)

print(obj1.area())
print(obj2.area())
print(obj3.area())
print(obj4.area())
print(obj5.area())
```

```
8
3.897
64.95
77.273
308.992
```

Refer to this link: https://colab.research.google.com/drive/1c8kM-GqYMOEDQWSiz9lF21EUZyz66XDu#scrollTo=u4jWwWYPDFEm

Questions

1. Why is Polymorphism important?

   Polymorphism is important because it allows a single interface to be used for different data types. It enables flexibility in code design, making it easier to extend and maintain programs. With polymorphism, objects of different classes can be treated as instances of a common parent class, simplifying the process of integrating new features without modifying existing code.

2. Explain the advantages and disadvantages of using applying Polymorphism in an Object-Oriented Program.

   The advantages of polymorphism include improved code reusability, scalability, and readability. It allows different objects to respond to the same method in unique ways, making the code more modular and adaptable. However, one disadvantage is that it can increase complexity, making debugging and tracing errors more challenging, especially in large programs with multiple inherited classes.

3. What maybe the advantage and disadvantage of the program we wrote to read and write csv and json files?

   When applying polymorphism to programs that read and write CSV and JSON files, an advantage is that it allows a uniform interface for handling different file formats, improving modularity. However, a disadvantage is that improperly structured polymorphic code may lead to compatibility issues, especially if unexpected data structures are encountered.

4. What maybe considered if Polymorphism is to be implemented in an Object-Oriented Program?

   To implement polymorphism effectively, developers must consider the relationships between classes, ensuring that methods are properly overridden or overloaded without causing unnecessary complexity. It is also crucial to follow best practices to prevent conflicts and ensure maintainability.

5. How do you think Polymorphism is used in an actual programs that we use today?

   In todays world, polymorphism is widely used in frameworks, APIs, and libraries. For example, in graphical user interfaces (GUIs), buttons, sliders, and checkboxes inherit from a common UI component class but behave differently when interacted with. Similarly, in database management systems, polymorphism allows different databases to be accessed through a common interface, regardless of their internal structure.

7. Conclusion:

This laboratory activity provided an in-depth exploration of polymorphism and its applications in Object-Oriented Programming. By implementing method overriding and operator overloading, this lab activity we gained a deeper understanding of how polymorphism enhances code flexibility and reusability. We performed a exercises which demonstrated how polymorphism simplifies code maintenance and integration in larger software projects. Understanding these principles is essential for developing scalable and efficient object-oriented applications.

8.Assessment Rubric