Laboratory Activity No. 7	
Polymorphism	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 02-22-25
Section: 1-A	**Date Submitted:** 02-22-25
Name: BRON, JHUSTINE A.	**Instructor:** ENGR. MARIA RIZETTE 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: 12 24
Enter feet and inches of distance2: 13 25
Distance2 is greater or equal to Distance1
Sum of the two Distance is:
Feet =  26 Inches =  37
```

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:

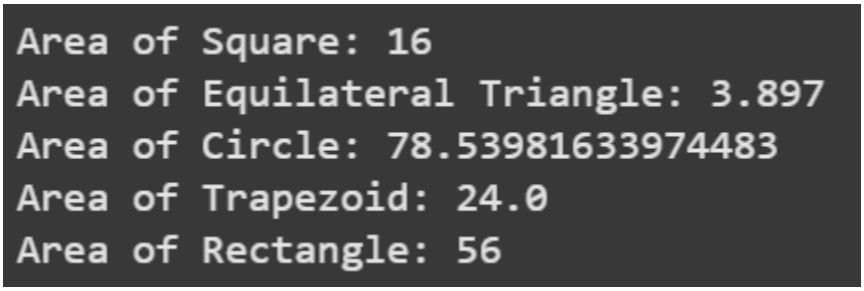
```
16
3.897
```

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

The code demonstrates polymorphism by defining a base class RegularPolygon and two derived classes, Square and EquilateralTriangle. Both derived classes have their own implementation of the area() method. This allows objects of different types (square and triangle) to respond to the same method call (area()) in their own specific way, calculating the area based on their respective shapes.

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.



https://colab.research.google.com/drive/1bT6pmbdzrcn8p-R4TgqL7eEtOTM0qGW8#scrollTo=15bqRNehWLiX

1. **RegularPolygon**: This is like a basic template for any shape with equal sides. It stores the length of one side (referred to as _side). Think of it as the parent of all the shapes.
2. **Square**: This class inherits from RegularPolygon and represents a square. It has a method to calculate the area using the formula: side * side.
3. **EquilateralTriangle**: Also inheriting from RegularPolygon, this represents an equilateral triangle (all sides equal). It calculates the area using an approximation: side * side * 0.433.
4. **Circle**: Although named as a regular polygon which is a misnomer, this class calculates the area of a circle using the formula: pi * (radius * radius). Here, the _side is used as radius.
5. **Trapezoid**: This class represents a trapezoid shape. It requires the length of two bases (base_a and base_b) and its height (height) along with the side for initialization. It calculates area using the formula: 0.5 * (base_a + base_b) * height. Note: although the sides may not be equal, this is included as a child class of RegularPolygon.
6. **Rectangle**: This class represents a rectangle. It requires two sides - _side and _side2, (length and width). It calculates area using the formula: side * side2. Note: although the sides may not be equal, this is included as a child class of RegularPolygon.

Output Descriptions:

The code creates objects of each shape and then prints their areas.
Area of Square:: This will print the area of a square with side length 4.
Area of Equilateral Triangle:: This will print the approximate area of an equilateral triangle with side length 3.
Area of Circle:: This will print the area of a circle with a radius of 5.
Area of Trapezoid:: This will print the area of a trapezoid with bases 5 and 7 and height 4.
Area of Rectangle:: This will print the area of a rectangle with sides of length 7 and 8.

Questions

1. Why is Polymorphism important?
   It enables flexibility and code reuse by allowing objects of different types to be treated as instances of a common interface.

2. Explain the advantages and disadvantages of using applying Polymorphism in an Object-Oriented Program.
   Advantages: Increases code reusability, flexibility, and scalability. Reduces the need for conditionals.
   Disadvantages: Can make debugging harder and add complexity to the code.

3. What maybe the advantage and disadvantage of the program we wrote to read and write csv and json files?
   Advantages: Simplifies data exchange and makes the program more versatile for different formats.
   Disadvantages: Potential performance issues with large datasets; error handling can become complex.
4. What maybe considered if Polymorphism is to be implemented in an Object-Oriented Program?
   When implementing polymorphism, make sure there's a well-defined common interface or base class. It's important to keep the design simple and avoid over-engineering. Also, ensure that the behavior of overridden methods is clear and doesn't introduce bugs. Balancing flexibility with maintainability is key.

5. How do you think Polymorphism is used in an actual programs that we use today?
   Used in GUI frameworks, game engines, and APIs where different objects (like buttons or characters) share a common interface but behave differently.

6. **Conclusion:**

Polymorphism is a powerful tool in object-oriented programming that promotes flexibility and code reuse by allowing objects of different classes to be treated as objects of a common type. While it can introduce some complexity, its benefits in terms of maintainability and scalability often outweigh the drawbacks. The code example demonstrates how polymorphism enables different shapes to calculate their areas through a shared interface, highlighting its practical application. By carefully considering design and implementation, polymorphism can be leveraged to create robust and adaptable software systems.