

Laboratory Activity No. 7

Polymorphism

Course Code: CPE103

Program: BSCPE

Course Title: Object-Oriented Programming

Date Performed: 02/22/2025

Section: 1-A

Date Submitted:

Name: MANONGSONG, KEN R.

Instructor: MA'AM 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 distance 1 (separated by space): 4 5
Enter feet and inches of distance 2 (separated by space): 6 8
Distance 2 is greater or equal to Distance 1
Sum of the two Distance is:
Feet= 11 Inches= 1
```

Testing and Observing Polymorphism

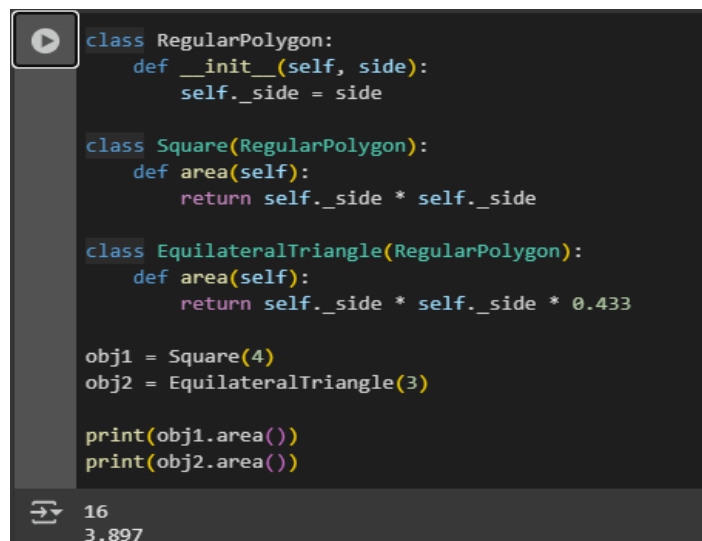
1. Create a code that displays the program below:

```
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())
```

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())
```

16
3.897

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

This Python code shows the inheritance and polymorphism by defining a RegularPolygon base class and specialized Square and EquilateralTriangle classes. Each subclass implements an area method, showing how the same method name can have different behaviors based on the object's type. The output shows the calculated areas of a square with side 4 and an equilateral triangle with side 3

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.

```
class Pentagon(RegularPolygon):  
    def area(self):  
        return 1.72048 * (self._side ** 2)
```

```
obj3 = Pentagon(5)
```

```
print(obj3.area())
```

OUTPUT:

```
43.012
```

In the existing code above I added a third shape which is a pentagon and I just followed the first two Shape's codes and I assigned the return value of 1.72048 the return value calculates the area of a regular pentagon using the formula: $\text{Area} \approx 1.72048 * (\text{side length})^2$

```
class Hexagon(RegularPolygon):  
    def area(self):  
        return (3 * (3**0.5) / 2) * (self._side ** 2)
```

```
obj4 = Hexagon(2)
```

```
print(obj4.area())
```

OUTPUT:

```
10.392304845413264
```

Lastly I added the fourth shape which is a Hexagon and I also followed the same flow for this shape but this time I changed the return value to $(3 * (3^{0.5}) / 2)$ the return value calculates the area of a regular hexagon using the formula: $\text{Area} = (3\sqrt{3} / 2) * (\text{side length})^2$

Questions

1. Why is Polymorphism important?

Polymorphism is crucial because it:

Reduces code duplication: One interface, many implementations. Increases flexibility: Code adapts to different object types. Improves maintainability: Changes are localized. Enhances extensibility: New types easily fit existing code.

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

Pros:

Less code duplication. Adapts to new object types. Easier code updates. Cleaner, simpler code.

Cons:

Harder to trace execution. Slight performance impact. Potential for type-related issues. Identifying root cause can be difficult.

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

Advantages:

Easily exchange data between systems. Convert between CSV (tabular) and JSON (hierarchical) formats. Automates the data processing tasks. Adapt to various data flows

Disadvantages:

Handling different data structures increases code complexity. Potential for data loss during format conversion. Vulnerabilities from untrusted data sources. Potential for data inconsistencies when changing file types back and forth.

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

Identify shared behaviors among classes. Design interfaces/abstract classes for contracts.

Prioritize flexibility and extensibility. Consider performance overhead. Enhance code readability and maintainability.

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

Polymorphism is great in modern applications, enabling them to handle diverse data and actions seamlessly. Polymorphism allows programs to treat different objects uniformly while allowing for specific implementations, leading to more flexible and maintainable code.

7. Conclusion:

The concept of polymorphism, enables objects of various types to be treated equally of a single type, is fundamental to object-oriented programming. This idea improves code reuse, simplifies maintenance, and makes programs more adaptable. We are introduced to polymorphism through practical exercises that shows how to implement and use polymorphic behavior in Python. The hands-on lab exercise effectively helps us understand the practical application, and how to apply these principles in real-world scenarios.

8. Assessment Rubric: