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: Filjohn B. Delinia	**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:

```
In [7]: runfile('C:/Users/ucc_e/OneDrive/Documents/untitled2.py', wdir='C:/Users/ucc_e/OneDrive/Documents')
Enter feet and inches of distance1: 5 9
Enter feet and inches of distance2: 4 11
Distance1 is greater than Distance2
Sum of the two Distance is:
Feet=  10 Inches=  8
```

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:

```
In [8]: runfile('C:/Users/ucc_e/OneDrive/Documents/oopfa1DELINIA_lab8/untitled3.py', wdir='C:/Users/ucc_e/OneDrive/Documents/
oopfa1DELINIA_lab8')
16
3.897
```

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

The code demonstrates inheritance and method overriding in object-oriented programming. The `Square` and `EquilateralTriangle` classes inherit from `RegularPolygon` and each define a custom `area()` method. The `Square` class calculates the area using `side * side`, while the `EquilateralTriangle` uses a different formula. The side length is stored in a protected variable, ensuring encapsulation. This setup allows for reusable and customizable behavior across the different 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.

```
In [9]: runfile('C:/Users/ucc_e/OneDrive/Documents/oopfa1DELINIA_lab8/untitled4.py', wdir='C:/Users/ucc_e/OneDrive/Documents/
oopfa1DELINIA_lab8')
Area of Square: 16
Area of Equilateral Triangle: 3.897
Area of Pentagon: 43.01193501472417
Area of Hexagon: 93.53074360871938
Area of Octagon: 236.5929291125633
```

Output Explanation:

1. Area of Square:
   - Input: Side length = 4
   - Formula: Area = side × side
   - Calculation: 4 × 4 = 16
   - Output: The area of the square is 16.

2. Area of Equilateral Triangle:
   - Input: Side length = 3
   - Formula: Area = side × side × 0.433 (approximately for equilateral triangles)
   - Calculation: 3 × 3 × 0.433 = 3.897
   - Output: The area of the equilateral triangle is 3.897.

3. Area of Pentagon:
   - Input: Side length = 5
   - Formula: Area = $(1/4) \times \sqrt{[5(5 + 2\sqrt{5})]} \times \text{side}^2$
   - Calculation: $(1/4) \times \sqrt{[5(5 + 2\sqrt{5})]} \times 5^2 = 43.0119$ (approximately)
   - Output: The area of the pentagon is 43.0119.

4. Area of Hexagon:
   - Input: Side length = 6
   - Formula: Area = $(3\sqrt{3} / 2) \times \text{side}^2$
   - Calculation: $(3\sqrt{3} / 2) \times 6^2 = 93.5307$ (approximately)
   - Output: The area of the hexagon is 93.5307.

5. Area of Octagon:
   - Input: Side length = 7
   - Formula: Area = $2 \times (1 + \sqrt{2}) \times \text{side}^2$
   - Calculation: $2 \times (1 + \sqrt{2}) \times 7^2 = 236.5929$ (approximately)
   - Output: The area of the octagon is 236.5929.

Questions

1. Why is Polymorphism important?

Polymorphism is important because it allows distinct things to be handled equally through a common interface, even if they behave differently. This makes our code more versatile and reusable. It also makes it easy to extend or modify the software without destroying existing functionality, which is very critical when managing huge systems. Polymorphism simplifies the program's structure, making it more responsive to change.

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

Polymorphism has several advantages, including reduced code duplication, improved maintainability, and increased flexibility. It enables more generalized code, making it easier to change or grow a program without having to rewrite major sections of it. However, it can make the program more complex because it requires careful organization of class hierarchies. Furthermore, utilizing polymorphism might increase performance overhead and make debugging more difficult because the exact method being executed is determined at runtime.

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

The application we created to handle CSV and JSON files has the advantage of being flexible and simple to use, allowing for speedy interaction with commonly used file formats. This facilitates data flow between systems. On the negative, the application may struggle with huge files, and error handling may be inadequate, causing it to miss issues such as corrupted data. Furthermore, dealing with increasingly complicated JSON formats that contain nested items may necessitate additional logic.

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

When implementing polymorphism, we must carefully design the class structure to ensure that it is obvious and extensible. It is critical to utilize abstract classes or interfaces to establish common methods that subclasses can later customize with their own specialized implementations. This ensures that the software stays adaptable and easy to maintain, while eliminating needless complexity that may make the code more difficult to understand.

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

Polymorphism is employed in numerous real-world applications. Polymorphism, for example, allows us to interact with multiple components such as buttons or text boxes in a consistent manner, despite the fact that they all behave differently. Similarly, in game creation, polymorphism allows different character types to share common actions (such as moving or attacking) while allowing each type to have its own distinct behavior. This increases the software's flexibility and allows for easier extension.

7. Conclusion:

In Conclusion, polymorphism is critical in object-oriented programming because it promotes code flexibility, reuse, and maintainability. It permits diverse objects to be treated consistently, making it easier to extend and adapt programs without damaging their fundamental structure. While polymorphism has various benefits, including streamlined code and ease of maintenance, it can also add complexity and performance difficulties. Polymorphism implementation needs careful design, clear class hierarchies, and the right use of abstract classes or interfaces. Polymorphism is commonly employed in real-world applications such as GUI frameworks, game development, and database access, enabling for systems that are more scalable, adaptive, and extendable.

8. Assessment Rubric: