

Polymorphism

Prof. Sindhu R Pai

PCPS Theory Anchor - 2024

Department of Computer Science and Engineering

Polymorphism



Introduction

- Polymorphism refers to having multiple forms.
- refers to the use of the same function name, but with different signatures
- allows objects of different classes to be treated as objects of a common superclass.
- This concept enables a single interface to be used for entities of different types.

Polymorphism



Runtime Polymorphism

- Python supports runtime polymorphism by using techniques like method overloading and method overriding
- Runtime polymorphism is the ability of an object to behave differently based on its actual type during program execution
- It is also known as dynamic polymorphism
- It enables the same method name to behave differently based on the specific class instance at runtime.

Polymorphism



Key Aspects of Runtime Polymorphism:

1.Inheritance:

- Runtime polymorphism is closely associated with inheritance.
- Subclasses inherit methods from their superclass, and they can provide their own implementation for these methods.

2. Method Overriding:

- Subclasses override methods from their superclass to provide their own specialized implementation.
- The method signature remains the same in both the superclass and subclass.

Polymorphism



3. Dynamic Binding:

- During runtime, the appropriate method to execute is dynamically determined
- It is based on the actual type of the object invoking the method

4.Common Interface:

- Different subclasses sharing a common superclass interface
- Exhibits different behaviors based on their specific implementations.

Polymorphism

The following example (Refer next slide) demonstrates all these key features of runtime polymorphism.

In this example 'Rectangle' and 'circle' subclasses override the 'calculate_area' method from the 'shape' superclass to provide their specific area calculation.

The method 'calculate_area' is called on objects of different types('rect' or 'circle'). The appropriate overridden method is executed based on the object's actual type.

This happens at the runtime and method execution is dynamically bound to the object's specific implementation.

Polymorphism

```
PES
UNIVERSITY
```

```
class Shape:
    def calculate_area(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

Polymorphism

```
class Circle(Shape):
  def __init__(self, radius):
    self.radius = radius
  def calculate_area(self):
    return 3.14 * self.radius * self.radius
rect = Rectangle(5, 10)
circle = Circle(4)
print(rect.calculate_area())
print(circle.calculate_area())
```



Polymorphism

PES UNIVERSITY

Practice Program

Build a music player program where 'Audiofile' is the base class. Implement sub classes 'MP3File' and 'WAVFile' inheriting from 'AudioFile'. Both classes should have a 'play' method but provide different functionalities for playing MP3 and WAV files.

Create a base class 'Vehicle' with a method 'calculate_speed'. Derive subclasses 'Car' and 'Bike' from 'Vehicle'. Override the 'calculate_speed' method in each subclass to calculate their specific speed based on different parameters.



THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU Prof. Sindhu R Pai – <u>sindhurpai@pes.edu</u> Prof. C N Rajeswari