

Polymorphism Using Inheritance

In this lesson, we will be implementing polymorphism using the OOP concepts.

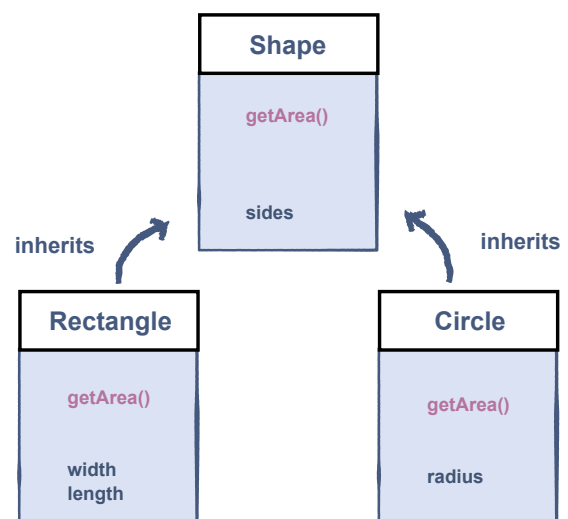
We'll cover the following

- Example
- Implementation
 - Shape Class
 - Rectangle Class
 - Circle Class
- Complete Program
- Explanation

So far, we have learned that we can add new data and methods to a class through inheritance. But what if we want our derived class to inherit a method from the base class and have a different implementation for it? That is when polymorphism, a fundamental concept in the OOP paradigm, comes into play.

Example

Here, we consider the example of a **Shape** class, which is the base class while many shapes like **Rectangle** and **Circle** extending from the base class are derived classes. These derived classes inherit the **getArea()** method, and provide a shape-specific implementation, which calculates its area.



Rectangle and Circle inherit from Shape

Implementation

We will be implementing the **parent class** first, followed by the **child classes**.

Shape Class

The **Shape** class has only one public method called **getArea()**.

Let's look at the implementation of the **Shape** class:

```
1 class Shape:
2     def __init__(self)
3         self.sides = 0
4
5     def getArea(self):
6         pass
7
```

Rectangle Class

Let's look at the implementation of the **Rectangle** class:

```
# Rectangle IS A Shape with a specific width and height
class Rectangle(Shape): # derived form Shape class
    # initializer
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.sides = 4

    # method to calculate Area
    def getArea(self):
        return (self.width * self.height)
```

The **Rectangle** class is extended from the **Shape** class. It inherits the **sides** property from the **Shape** class and defines new properties, **width** and **height**. The *method* **getArea()** returns the area of the rectangle.

Circle Class

Let's look at the implementation of the **Circle** class:

```
# Circle IS A Shape with a specific radius
class Circle(Shape): # derived form Shape class
```

```

class Circle(Shape): # derived form Shape class
    # initializer
    def __init__(self, radius):
        self.radius = radius
        self.sides = 0

    # method to calculate Area
    def getArea(self):
        return (self.radius * self.radius * 3.142)

```

The `Circle` class is extended from the `Shape` class. It inherits the `sides` property from the `Shapes` class and defines only one new *property*, `radius`. The method `getArea()` returns the *area* of the circle.

Complete Program

Now, by merging all the classes and calling the `getArea()` method, see what happens:

```

class Shape:
    def __init__(self): # initializing sides of all shapes to 0
        self.sides = 0

    def getArea(self):
        pass

class Rectangle(Shape): # derived form Shape class
    # initializer
    def __init__(self, width=0, height=0):
        self.width = width
        self.height = height
        self.sides = 4

    # method to calculate Area
    def getArea(self):
        return (self.width * self.height)

class Circle(Shape): # derived form Shape class
    # initializer
    def __init__(self, radius=0):
        self.radius = radius

    # method to calculate Area
    def getArea(self):
        return (self.radius * self.radius * 3.142)

shapes = [Rectangle(6, 10), Circle(7)]
print("Area of rectangle is:", str(shapes[0].getArea()))
print("Area of circle is:", str(shapes[1].getArea()))

```



Explanation

In the main function, we have declared a list that has two objects in it. The first object is a `Rectangle` with `width` 6 and height `10`. The second object is a `Circle` of radius 7.

The `getArea()` method returns the area of the respective shape. This is **Polymorphism**; having specialized implementations of the same methods for each class.

In the next lesson, we'll be learning about the process of **method overriding**.