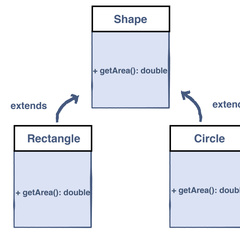
So far, we have learned that we can add new data and methods to a class through inheritance. But what if we want our derived class to inherit a method from the base class and have a different implementation for it? That is when polymorphism, a fundamental concept in the OOP paradigm, comes into play.

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 classes contain the **getArea()** method which calculates the area for the respective shape.



Implementation

We will be implementing the **Base** class and the **Derived** classes respectively.

Shape Class

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

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

```
// A simple Shape which provides a method to get the Shape's area
class Shape {

  public double getArea(){}

}
```

Rectangle Class

Now, consider the **Rectangle** class which is extended from the *Shape* class. It has two data members, i.e., `width` and `height` and it returns the *Area* of the rectangle by using the **getArea()** method.

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

```
// A Rectangle is a Shape with a specific width and height
class Rectangle extends Shape {   // derived form Shape class

  // Private data members
  private double width;
  private double height;

  // Constructor
  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  // Public method to calculate Area
  public double getArea() {
    return width * height;
  }

}
```

Circle Class

Now, consider the **Circle** class which is extended from the *Shape* class. It has only one data member, i.e., *radius* and it returns the *Area* of the circle by using the **getArea()** method.

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

```
// A Circle is a Shape with a specific radius
class Circle extends Shape {

  // Private data member
  private double radius;

  // Constructor
  public Circle(double radius) {
    this.radius = radius;
  }

  // Public method to calculate Area
  public double getArea() {
    return 3.14 * radius * radius;
  }

}
```

Complete Program

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

```

// A Circle is a Shape with a specific radius
class Circle extends Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }
  public double getArea() {
    return 3.14 * radius * radius;
  }

}


class driver {

  public static void main(String args[]) {
    Shape[] shape = new Shape[2]; // Creating shape array of size 2

    shape[0] = new Circle(2); // creating circle object at index 0
    shape[1] = new Rectangle(2, 2); // creating rectangle object at index 1

    System.out.println("Area of the Circle: " + shape[0].getArea());
    System.out.println("Area of the Rectangle: " + shape[1].getArea());
  }

}
```

Run Save Reset

Program Execution

In the main function, we have declared a **Shape** class array of size **2** and declared the **Circle** and the **Rectangle** class objects at index **0** and **1** respectively. Now the `getArea()` method returns the area of the respective shape. This is **Polymorphism.**