

Polymorphism

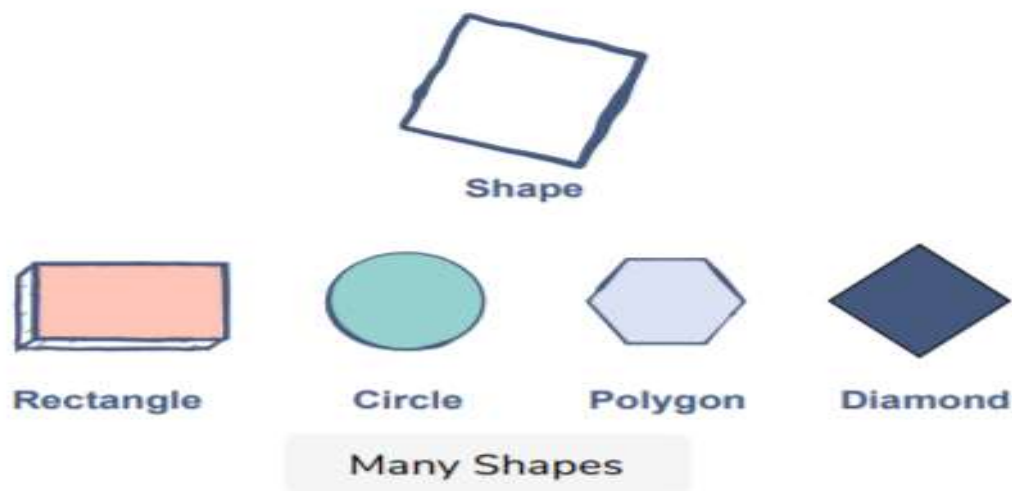
We will cover the following:-

- Definition
- Types of polymorphism
- Compile-time polymorphism
- Method overloading with example
- Run-time polymorphism.
- Method overriding with example

Definition:-

The word Polymorphism is a combination of two Greek words, Poly means many and Morph means forms.

In programming, polymorphism refers to the same object exhibiting different forms and behaviors. For example, take the Shape Class. The exact shape you choose can be anything. It can be a rectangle, a circle, a polygon, or a diamond. So, these are all shapes but their properties are different. This is called Polymorphism



Types of polymorphism

- Compile-time polymorphism
- Runtime polymorphism

Compile-time polymorphism

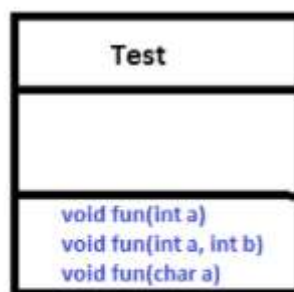
It is also known as static polymorphism. This type of polymorphism is achieved by Method Overloading or operator overloading

Note:- operator overloading is not supported by java.

Method overloading:-

when we have more than one function/method in the same class with the same name and number of arguments. then these functions are known as overloaded functions. Functions can be overloaded by a change in the number of arguments or/and a change in the type of arguments.

Note:-different return type is not considered as overloading.



Overloading

Here, you can see there are three functions in the same class having the same name and the same number of arguments. So these functions are overloaded.

Here we will look at an example of the same with the help of a program written in java. We have overloaded multiply function using different arguments types, and a number of arguments are different in each function.

Main.java

```
Main.java
 2 // Helper class
 3 class Helper {
 4
 5     // Method with 2 integer parameters
 6     static int Multiply(int one, int two)
 7     {
 8         // Returns product of integer numbers
 9         return one * two;
10     }
11
12     // Method 2
13     // With same name but with 2 double parameters
14     static double Multiply(double one, double two)
15     {
16         // Returns product of double numbers
17         return one * two;
18     }
19
20     static int Multiply(int p, int q, int r)
21     {
22         // Return product
23         return p * q * r;
24     }
25 }
26
27 // Class 2
28 // Main class
29 class Main {
30
31     // Main driver method
32     public static void main(String[] args)
33     {
34         // Calling method by passing
35         // input as in arguments
36         System.out.println(Helper.Multiply(5,4));
37         System.out.println(Helper.Multiply(6,4,8));
38         System.out.println(Helper.Multiply(5.5, 6.3));
39     }
40 }
```

```
20
192
34.65
```

```
...Program finished with exit code 0
```

Run-time polymorphism

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. Method Overriding achieves this type of polymorphism. On the other hand, method overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Method overriding

In a simple language, when we have two classes, one is child class, and the other is parent class, and when we write the same function in both the child class and the parent class, the method is said to be overridden. This concept is known as runtime polymorphism because the compiler will decide at runtime to which function it will call during the program's execution.

```
Main.java
1 // parent class
2 class vehicle {
3     // Method of parent class
4     void run()
5     {
6         System.out.println("vehicle is running ");
7     }
8 }
9 // child class
10 class car extends vehicle{
11
12     // Method
13     void run() {
14         System.out.println("car is running");
15     }
16 }
17
18 // Main class
19 class Main {
20
21     // Main driver method
22     public static void main(String[] args)
23     {
24
25         // Creating object
26         vehicle pqr =new car();
27
28         // Now we will be calling print methods
29         // inside main() method
30         pqr.run();
31
32         //when the object behaves as vehicle
33         vehicle abc=new vehicle();
34         abc.run();
35     }
36 }
```

```
car is running
vehicle is running
...Program finished with exit code 0
```

Here, you can easily understand that the run method is called at the runtime, according to whether the vehicle is behaving like a car, or the vehicle is behaving like the vehicle itself.