Difference between Compile-time and Run-time Polymorphism in Java

Difficulty Level : Medium Last Updated : 27 Jan, 2021

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. In this article, we will see the difference between two types of polymorphisms, compile time and run time.

Compile Time Polymorphism: Whenever an object is bound with their functionality at the compile-time, this is known as the compile-time polymorphism. At compile-time, java knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding. Compile-time polymorphism is achieved through method overloading. Method Overloading says you can have more than one function with the same name in one class having a different prototype. Function overloading is one of the ways to achieve polymorphism but it depends on technology that which type of polymorphism we adopt. In java, we achieve function overloading at compile-Time. The following is an example where compile-time polymorphism can be observed.

```
// Java program to demonstrate
// compile-time polymorphism
public class GFG {

    // First addition function
    public static int add(int a, int b)
    {
        return a + b;
    }

    // Second addition function
    public static double add(
        double a, double b)
    {
        return a + b;
    }
```

```
    // Driver code
    public static void main(String args[])
    {
        // Here, the first addition
        // function is called
        System.out.println(add(2, 3));

        // Here, the second addition
        // function is called
        System.out.println(add(2.0, 3.0));
    }
}
```

Output:

```
5
5.0
```

Run-Time Polymorphism: Whenever an object is bound with the functionality at run time, this is known as runtime polymorphism. The runtime polymorphism can be achieved by method overriding. Java virtual machine determines the proper method to call at the runtime, not at the compile time. It is also called dynamic or late binding. Method overriding says child class has the same method as declared in the parent class. It means if child class provides the specific implementation of the method that has been provided by one of its parent class then it is known as method overriding. The following is an example where runtime polymorphism can be observed.

```
// Java program to demonstrate
// runtime polymorphism

// Implementing a class
class Test {

    // Implementing a method
    public void method()
    {
        System.out.println("Method 1");
```

```
    }
}

// Defining a child class
public class GFG extends Test {

    // Overriding the parent method
    public void method()
    {
        System.out.println("Method 2");
    }

    // Driver code
    public static void main(String args[])
    {
        Test test = new GFG();

        test.method();
    }
}
```

Output:

```
Method 2
```

The following table demonstrates the difference between runtime polymorphism and compile-time polymorphism:

Sr.No	Compile Time Polymorphism	Run time Polymorphism
1	In Compile time Polymorphism, the call is resolved by the compiler.	In Run time Polymorphism, the call is not resolved by the compiler.
2	It is also known as Static binding, Early binding and overloading as well.	It is also known as Dynamic binding, Late binding and overriding as well.
3	Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Method overriding is the runtime polymorphism having same method with same parameters or signature, but associated in different classes.

Sr.No	Compile Time Polymorphism	Run time Polymorphism
4	It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
5	It provides fast execution because the method that needs to be executed is known early at the compile time.	It provides slow execution as compare to early binding because the method that needs to be executed is known at the runtime.
6	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.