Polymorphism in Java

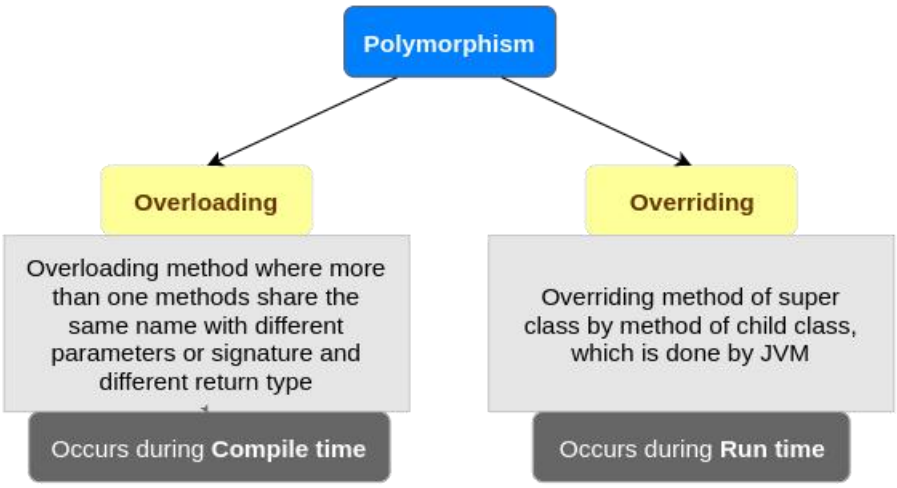
Polymorphism in Java

In Greek language, **Poly** means **many** and **morphs** means **forms**. Therefore **Polymorphism** translates to **many forms**. Precisely, Polymorphism is a condition of occurring in several different forms. Polymorphism in Java can occur in two ways:

- Overloading
- Overriding

In this tutorial, we are going to learn what overloading and overriding mean, with examples.

Following picture gives a basic difference between overloading and overriding in Java.



You may refer Method Overloading in Java to understand what Overloading is in detail.

You may also refer Method Overriding in Java to understand what Overriding is in detail.

Differences between Overloading and Overriding

Following table helps you find the differentiation between Overloading and Overriding mechanisms in Java. The first column contains a parameter on which we differentiate overloading and overriding.

Parameter	Overloading	Overriding
Resolved by	Compiler	JVM (Java Virtual Machine)

Resolved during	Compile time	Run time
Is achieved by	function overloading or parameter overloading	virtual methods and references to object instances
Effect on execution time	Fast ('cause, its done at compile time only once)	Slow ('cause its done at runtime whenever overriding happens)
Other names	Early binding (or) Static binding	Late binding (or) Dynamic binding

Let us see how overloading and overriding works, with the help of following examples.

Example 1 – Overloading

In this example, we have two methods, with san `printer`, but with different set of parameters. First `printer()` method accepts two String arguments while second `printer()` accepts only one String argument. In the main() method, we are calling printer() method with different number of arguments. During compilation, compiler resolves the method to be called, based on the number of arguments we are passing.

Messenger.java

```
package com.tutorialkart.java;

/**
 * @author tutorialkart
 */
public class Messenger {

    public static void main(String[] args)
        { Messenger msngr = new Messenger();
        msngr.printer("Bring me two trays of eggs.", "Room-mate");
        msngr.printer("Good Morning India.");
    }

    public void printer(String message,String
        sender){ System.out.println(sender + " : " + message);
    }

    public void printer(String
        message){ System.out.println("Broadcast message : " +
        message);
    }
}
```

Run the above Java program. Output to the console should be as shown below.

```
Room-mate : Bring me two trays of eggs.
Broadcast message : Good Morning India.
```

Example 2 – Overriding

In this example, we have super class `Car.java` with method run(). And there is a child class `Jaguar.java` with method run(). When an object of type Car is created with reference to Jaguar (**Car c = new Jaguar();**), invoking run method on object "c", would execute run method in Jaguar. All this is done during run time.

OverridingExample.java

```
package com.tutorialkart.java;

/**
 * @author tutorialkart
 */
class Car{
    public void run(){
        System.out.println("Car is running.");
    }
}

class Jaguar extends Car{
    public void run(){
        System.out.println("Jaguar is running.");
    }
}

public class OverridingExample {
    public static void main(String[] args)
        { Car c = new Jaguar();
        c.run();
    }
}
```

Run the above program and you shall get the output to the console as shown below.

```
Jaguar is running.
```