Java Polymorphism

By Umesh Sir

Contact us 7758094241

Polymorphism

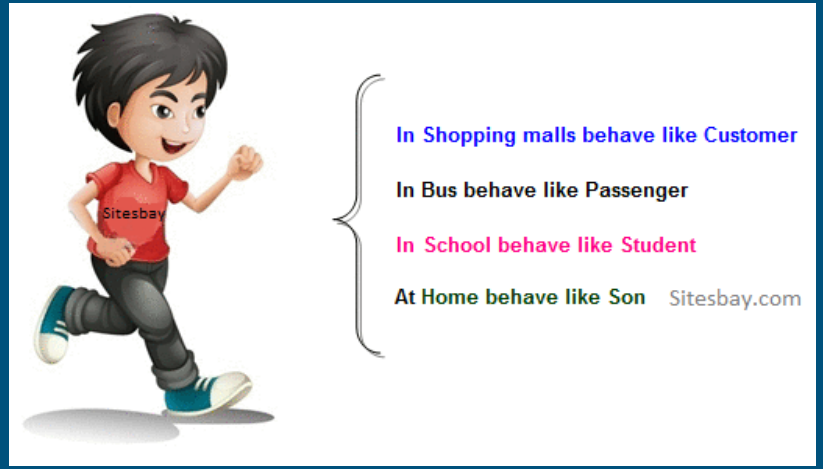
- ★ What is polymorphism?
- ★ Polymorphism Real-World Example.
- ★ Types of Polymorphism in Java.
- ★ Need for Java Polymorphism.
- ★ Compile time polymorphism or method overloading or static binding.
- ★ Example of Method Overloading.
- ★ Runtime polymorphism or method overriding or dynamic binding
- ★ Example of Method Overriding.
- ★ Method Overloading VS Method Overriding.

★ What is polymorphism?

- ★ Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- ★ Polymorphism means one name many forms. OR
- ★ The process of representing one form in multiple forms is known as Polymorphism. OR
- ★ Polymorphism lets us perform a single action in different ways. OR
- ★ Polymorphism allows you to define one interface and have multiple implementations. OR
- ★ We can create functions or reference variables that behave differently in a different programmatic context.

★ Polymorphism Real-World Example.

★ Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.



★ Need for Java Polymorphism.

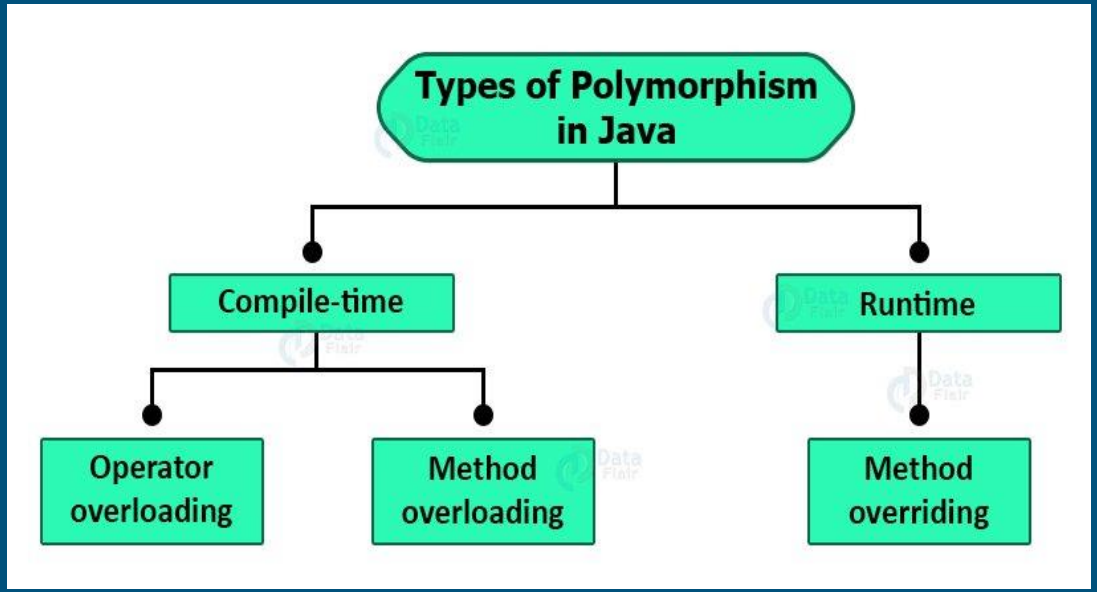
- ★ Polymorphism allows the programmer to write code that is easy to understand. How?
- ★ Let us see the example of a class Car.
- ★ It has values like mileage and functions such as acceleration, However, a subclass, say, for example, Rolls Royce won't be having the default acceleration value.
- ★ For a programmer to correctly define this function, he needs to rewrite the same function again.
- ★ However, polymorphism allows the programmer to use the same function name and redefine it in the child class which has a different implementation of the same method.

★ What is method signature ?

★ In a method declaration, the method name along with it's parameter types is known as method signature. For example in a method declaration public static int add(int a, int b){}, the signature will be add(int, int). Two methods will be of same signature if their signature exactly matches with each other.

★ Types of Polymorphism in Java.

1. Compile time polymorphism or method overloading or static banding
2. Runtime polymorphism or method overriding or dynamic binding



★Compile time polymorphism or method overloading or static banding.

- ★ If the class contains two or more methods having the same name and different arguments then it is method overloading.
- ★ All of the functions have the same method name but they have different arguments which makes them unique.
- ★ The compiler decides which function to call while compiling the program.
- ★ The compiler will resolve the call to a correct method depending on the actual number and/or types of the passed parameters The advantage of method overloading is to increases the readability of the program.

Example of Method Overloading.

In this example, we have created two methods, first, add() method performs the addition of two numbers and a second add method performs addition of three numbers.

```
class Adder {

    int add(int a, int b)   {  return a + b;  }

    int add(int a, int b, int c) {    return a + b + c;  } }

class TestOverloading  {

    public static void main(String[] args) {

                Adder obj = new Adder();

        System.out.println(obj.add(11,  11));

        System.out.println(obj.add(11,  11, 11));

    } }
```

As you can see, the values are different for the same name of the function. The compiler determines the function to call while compiling the program.

This is space-efficient and is easy to understand because the name of the function is the same. Debugging the program becomes easier.

★Runtime polymorphism or method overriding or dynamic binding

- ★ As the name signifies, method overriding is the process of overriding or redefining a method that was already defined in the parent class.
- ★ This is efficient and useful in many cases because a lot of the time, there is a need for redefining the function based on the class it is in.
- ★ Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- ★ In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
- Let's first understand the upcasting before Runtime Polymorphism.
- Upcasting:

- When the reference variable of Parent class refers to the object of Child class, it is known as upcasting.
- There is an important point to note that data members do not override. They stay the same.

★Example of Method Overriding.

Note that the definition of the fuel is altered in the child class. This is method overriding.

```
class Car {
  void fuel() {
    System.out.println("Can have diesel or petrol as the fuel");
  }
}
class Sedan extends Car {
  void fuel() {
    System.out.println("Shiny Car, but runs on diesel. ");
  }
}
class Engine extends Sedan {
  void fuel() {
    System.out.println("Converting diesel to smooth motion!");
  }
  public static void main(String args[]) {
    Car a1,
    a2,
    a3;
    a1 = new Car();
    a2 = new Sedan();
    a3 = new Engine();
    a1.fuel();
    a2.fuel();
    a3.fuel();
  }
}
```

- Method overriding means writing the same method both in parent class and child class,
- When we go for method overriding, when ever we are not satisfied with parent method then we should again write the same method with different logic.
- Example:
- Here there is a class called bank having one method for calculating interest, But Sbi do not want to to give interest at 4% , he want to 8% in that case Sbi overrides that method with modified logic.

```
class Bank
{
        public  double calcInterst(double amount,int duration)
        {
                //Logic for calculationg 4% interst
           return amount*duration*4/100;
        }
}
class Sbi extends Bank
{
        public double calcInterst(double amount,int duration)
        {
         return amount*duration*8/100;
        }
}
class User
{
        public static void main(String [ ] args)
        {
        Bank b=new Sbi();
        System.out.println(b.calcInterst(2000,2));
        }
}
```

★Method Overloading VS Method Overriding.

Method Overloading	Method Overriding
Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
In java, method overloading can't be done by changing only the return type of method. *Return type can be same/different* in overloading, but you must change the parameter.	*Return type must be same or covariant (changing return type to subclass type)* in method overriding.

★ Use of method overriding in Java

The main reason to use method overriding is to change/modify the behavior of parent class method as per the requirement in child class. Now you may think that why not to change the parent class method itself ?, the reason is, that parent class may have been inherited by many other classes as well. If you modify the method of parent class, that will be reflected in all the subclasses whether they need that change or not, which may affect the functionality of your project, so it's not a good practice at all.

Method overriding is used to achieve run time polymorphism in java which is an essential concept of object oriented programming.

When we should use method overriding?

When you find that the parent class method is not full-filling the child class requirement, in other words when you have a need to change/modify the behavior of an existing method of parent class inside child class, you should use method overriding. Many times in real world projects when we create a more specific class from a generic class, we need to use method overriding to change the behavior of generic class method.

For example let's suppose you have an Animal class having a method as sound, now if you create classes like Dog and Cat from Animal class, you would be needed to change/override the behavior of sound method in Dog and Cat classes since a dog barks whereas a cat meows.

★ A real example of Java Method Overriding?

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.

★How does JVM decides which method to call at run time

The JVM checks the actual type of the object which is calling the method, then on that actual type it calls the corresponding method. The actual type of an object is the type which is used with new keyword. For example when you create the object as new Teacher(), it's actual type will be of Teacher type, so it will call the Teacher class method, no matter whether this object was assigned into Teacher class type or Person class type.

```
Teacher teacher = new Teacher();  // Actual type is Teacher

Person person = new Teacher();  // Actual type is Teacher

Person person2 = new Person();  // Actual type is Person
```

★Method overriding rules in Java?

1.There must be parent child relationship which means inheritance must be there.

2.The signature of parent and child class methods must be same.

3.If return type of parent class method is a primitive type, then child class method must also have the same return type.

4.If return type of parent class method is a non primitive type then a covariant return type can also be used in child class method from java 5.

5.The access modifier of child class method should not be more restrictive. It should be same or more accessible modifier.

Thank you