Polymorphism, Method Overloading and Overriding in Java

Polymorphism in Java

Definition:

Polymorphism in Java is the ability of an object to take on many forms. It allows one interface to be used for a general class of actions, meaning a single method can behave differently based on the object that is calling it.

Types of Polymorphism in Java:

1. Compile-time Polymorphism (Static binding / Method Overloading)
2. Runtime Polymorphism (Dynamic binding / Method Overriding)

1. Compile-time Polymorphism (Method Overloading)

This happens when two or more methods in the same class have the same name but different parameters.

Example:

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

   double add(double a, double b) {
      return a + b;
   }
}
```

Syntax:

```
return_type methodName(parameter_list) {
   // method body
}
```

2. Runtime Polymorphism (Method Overriding)

This happens when a subclass provides a specific implementation of a method that is already defined in its superclass.

Example:

```
class Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}

class Dog extends Animal {
   void sound() {
      System.out.println("Dog barks");
   }
}
```

Syntax:

```
@Override
return_type methodName() {
   // overridden method body
}
```

Key Points:

- Polymorphism increases flexibility and reusability of code.
- It supports inheritance and interface implementation.
- Method Overloading is resolved at compile time.
- Method Overriding is resolved at runtime.

Method Overloading and Method Overriding in Java

1. Method Overloading

Definition:

Method overloading means defining multiple methods in the same class with the same name but different parameters (type, number, or order). It is an example of compile-time polymorphism.

Rules:

- Method names must be the same.
- The parameter list must be different.
- It can have a different return type.
- It must be in the same class.

Example:

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

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

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

class Main {
    public static void main(String[] args) {
        Calculator obj = new Calculator();
        System.out.println(obj.add(10, 20));        // Output: 30
        System.out.println(obj.add(2.5, 3.5));      // Output: 6.0
        System.out.println(obj.add(1, 2, 3));       // Output: 6
    }
}
```

Syntax:

```
return_type methodName(parameter_list) {
    // method body
}
```

2. Method Overriding

Definition:

Method overriding means defining a method in the child class that already exists in the parent class with the same name, same return type, and same parameters. It is an example of runtime polymorphism.

Rules:

- The method must have the same name, return type, and parameters.
- It must be in a subclass (inheritance is required).
- The base method must not be private, final, or static.

Example:

```
class Animal {
   void sound() {
      System.out.println("Animal makes sound");
   }
}

class Dog extends Animal {
   @Override
   void sound() {
      System.out.println("Dog barks");
   }
}

class Main {
   public static void main(String[] args) {
      Animal obj = new Dog();
      obj.sound();   // Output: Dog barks
   }
}
```

Syntax:

```
@Override
return_type methodName(parameter_list) {
   // overridden method body
}
```

Difference Between Overloading and Overriding

Feature	Method Overloading	Method Overriding
Polymorphism Type	Compile-time Polymorphism	Runtime Polymorphism
Class	Same class	Parent and child classes
Method Signature	Must be different	Must be the same
Inheritance Requirement	Not required	Required
Return Type	Can be different	Must be the same