Polymorphism in Java

**Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

# Method Overloading in Java

1. [Different ways to overload the method](https://www.javatpoint.com/method-overloading-in-java#monumberofways)
2. [By changing the no. of arguments](https://www.javatpoint.com/method-overloading-in-java#mobynumber)
3. [By changing the datatype](https://www.javatpoint.com/method-overloading-in-java#mobydatatype)
4. [Why method overloading is not possible by changing the return type](https://www.javatpoint.com/method-overloading-in-java#moreturntype)
5. [Can we overload the main method](https://www.javatpoint.com/method-overloading-in-java#momainmethod)
6. [method overloading with Type Promotion](https://www.javatpoint.com/method-overloading-in-java#motypepromotion)

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the [program](https://www.javatpoint.com/java-programs).

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

## **Advantage of method overloading**

Method overloading increases the readability of the program.

### **Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

#### **In java, Method Overloading is not possible by changing the return type of the method only.**

#### **Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters**

# Method Overriding in Java

1. [Understanding the problem without method overriding](https://www.javatpoint.com/method-overriding-in-java#moverproblem)
2. [Can we override the static method](https://www.javatpoint.com/method-overriding-in-java#movercanstatic)
3. [Method overloading vs. method overriding](https://www.javatpoint.com/method-overriding-in-java#moverdiff)

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

### **Usage of Java Method Overriding**

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### **Rules for Java Method Overriding**

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

# Method Overriding in Java

1. [Understanding the problem without method overriding](https://www.javatpoint.com/method-overriding-in-java#moverproblem)
2. [Can we override the static method](https://www.javatpoint.com/method-overriding-in-java#movercanstatic)
3. [Method overloading vs. method overriding](https://www.javatpoint.com/method-overriding-in-java#moverdiff)

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

### **Usage of Java Method Overriding**

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### **Rules for Java Method Overriding**

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

### **Understanding the problem without method overriding**

Let's understand the problem that we may face in the program if we don't use method overriding.

1. //Java Program to demonstrate why we need method overriding
2. //Here, we are calling the method of parent class with child
3. //class object.
4. //Creating a parent class
5. **class** Vehicle{
6. **void** run(){System.out.println("Vehicle is running");}
7. }
8. //Creating a child class
9. **class** Bike **extends** Vehicle{
10. **public** **static** **void** main(String args[]){
11. //creating an instance of child class
12. Bike obj = **new** Bike();
13. //calling the method with child class instance
14. obj.run();
15. }
16. }

### **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.



#### **Java method overriding is mostly used in Runtime Polymorphism which we will learn in next pages.**

1. //Java Program to demonstrate the real scenario of Java Method Overriding
2. //where three classes are overriding the method of a parent class.
3. //Creating a parent class.
4. **class** Bank{
5. **int** getRateOfInterest(){**return** 0;}
6. }
7. //Creating child classes.
8. **class** SBI **extends** Bank{
9. **int** getRateOfInterest(){**return** 8;}
10. }
12. **class** ICICI **extends** Bank{
13. **int** getRateOfInterest(){**return** 7;}
14. }
15. **class** AXIS **extends** Bank{
16. **int** getRateOfInterest(){**return** 9;}
17. }
18. //Test class to create objects and call the methods
19. **class** Test2{
20. **public** **static** **void** main(String args[]){
21. SBI s=**new** SBI();
22. ICICI i=**new** ICICI();
23. AXIS a=**new** AXIS();
24. System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
25. System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
26. System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
27. }
28. }

# Final Keyword In Java

1. [Final variable](https://www.javatpoint.com/final-keyword#finalv)
2. [Final method](https://www.javatpoint.com/final-keyword#finalm)
3. [Final class](https://www.javatpoint.com/final-keyword#finalc)
4. [Is final method inherited ?](https://www.javatpoint.com/final-keyword#finalisinherited)
5. [Blank final variable](https://www.javatpoint.com/final-keyword#finalblank)
6. [Static blank final variable](https://www.javatpoint.com/final-keyword#finalstaticblank)
7. [Final parameter](https://www.javatpoint.com/final-keyword#finalpara)
8. [Can you declare a final constructor](https://www.javatpoint.com/final-keyword#finalcons)

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.



### **Upcasting**

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



1. **class** A{}
2. **class** B **extends** A{}
3. A a=**new** B();//upcasting

For upcasting, we can use the reference variable of class type or an interface type. For Example:

1. **interface** I{}
2. **class** A{}
3. **class** B **extends** A **implements** I{}

Here, the relationship of B class would be:

B IS-A A

B IS-A I

B IS-A Object

Since Object is the root class of all classes in Java, so we can write B IS-A Object.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Typecasting is converting one data type to another.

**Up-casting −** Converting a subclass type to a superclass type is known as up casting.

## Example

class Super {

   void Sample() {

      System.out.println("method of super class");

   }

}

public class Sub extends Super {

   void Sample() {

      System.out.println("method of sub class");

   }

   public static void main(String args[]) {

      Super obj =(Super) new Sub(); obj.Sample();

   }

}

**Down-casting −** Converting a superclass type to a subclass type is known as downcasting.

## Example

class Super {

   void Sample() {

      System.out.println("method of super class");

   }

}

public class Sub extends Super {

   void Sample() {

      System.out.println("method of sub class");

   }

   public static void main(String args[]) {

      Super obj = new Sub();

      Sub sub = (Sub) obj; sub.Sample();

   }

}

# Types of polymorphism in java- Runtime and Compile time polymorphism

n the last tutorial we discussed [Polymorphism in Java](https://beginnersbook.com/2013/03/polymorphism-in-java/). In this guide we will see **types of polymorphism**. There are two types of polymorphism in java:  
1) **Static Polymorphism** also known as compile time polymorphism  
2) **Dynamic Polymorphism** also known as runtime polymorphism

## Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.  
**Method Overloading**: This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters. We have already discussed Method overloading here: If you didn’t read that guide, refer: [Method Overloading in Java](https://beginnersbook.com/2013/05/method-overloading/)

### Example of static Polymorphism

Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

class SimpleCalculator

{

int add(int a, int b)

{

return a+b;

}

int add(int a, int b, int c)

{

return a+b+c;

}

}

public class Demo

{

public static void main(String args[])

{

SimpleCalculator obj = new SimpleCalculator();

System.out.println(obj.add(10, 20));

System.out.println(obj.add(10, 20, 30));

}

}

**Output:**

30

60

## Runtime Polymorphism (or Dynamic polymorphism)

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism. I have already discussed method overriding in detail in a separate tutorial, refer it: [Method Overriding in Java](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/).

**Example**  
In this example we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of parent class. In this example we have child class object assigned to the parent class reference so in order to determine which method would be called, the type of the object would be determined at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

To understand the concept of overriding, you should have the basic knowledge of [inheritance in Java](https://beginnersbook.com/2013/03/inheritance-in-java/).

class ABC{

public void myMethod(){

System.out.println("Overridden Method");

}

}

public class XYZ extends ABC{

public void myMethod(){

System.out.println("Overriding Method");

}

public static void main(String args[]){

ABC obj = new XYZ();

obj.myMethod();

}

}

**Output:**

Overriding Method

When an overridden method is called through a reference of parent class, then type of the object determines which method is to be executed. Thus, this determination is made at run time.  
Since both the classes, child class and parent class have the same method animalSound. Which version of the method(child class or parent class) will be called is determined at runtime by JVM.

**Few more overriding examples:**

ABC obj = new ABC();

obj.myMethod();

// This would call the myMethod() of parent class ABC

XYZ obj = new XYZ();

obj.myMethod();

// This would call the myMethod() of child class XYZ

ABC obj = new XYZ();

obj.myMethod();

// This would call the myMethod() of child class XYZ

In the third case the method of child class is to be executed because which method is to be executed is determined by the type of object and since the object belongs to the child class, the child class version of myMethod() is called.

# Static and dynamic binding in java

Association of method call to the method body is known as binding. There are two types of binding: **Static Binding** that happens at compile time and **Dynamic Binding** that happens at runtime. Before I explain static and dynamic binding in java, lets see few terms that will help you understand this concept better.

What is reference and object?

class Human{

....

}

class Boy extends Human{

public static void main( String args[]) {

/\*This statement simply creates an object of class

\*Boy and assigns a reference of Boy to it\*/

Boy obj1 = new Boy();

/\* Since Boy extends Human class. The object creation

\* can be done in this way. Parent class reference

\* can have child class reference assigned to it

\*/

Human obj2 = new Boy();

}

}

## Static and Dynamic Binding in Java

As mentioned above, association of method definition to the method call is known as binding. There are two types of binding: Static binding and dynamic binding. Lets discuss them.

### Static Binding or Early Binding

The binding which can be resolved at compile time by compiler is known as static or early binding. The binding of static, private and final methods is [compile-time](https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/). **Why?** The reason is that the these method cannot be overridden and the type of the class is determined at the compile time. Lets see an example to understand this:

### Static binding example

Here we have two classes Human and Boy. Both the classes have same method walk() but the method is static, which means it cannot be overriden so even though I have used the object of Boy class while creating object obj, the parent class method is called by it. Because the reference is of Human type (parent class). So whenever a binding of static, private and final methods happen, type of the class is determined by the compiler at compile time and the binding happens then and there.

class Human{

public static void walk()

{

System.out.println("Human walks");

}

}

class Boy extends Human{

public static void walk(){

System.out.println("Boy walks");

}

public static void main( String args[]) {

/\* Reference is of Human type and object is

\* Boy type

\*/

Human obj = new Boy();

/\* Reference is of HUman type and object is

\* of Human type.

\*/

Human obj2 = new Human();

obj.walk();

obj2.walk();

}

}

Output:

Human walks

Human walks

### Dynamic Binding or Late Binding

When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding. [Method Overriding](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/) is a perfect example of dynamic binding as in overriding both parent and child classes have same method and in this case the **type of the object** determines which method is to be executed. The type of object is determined at the run time so this is known as dynamic binding.

### Dynamic binding example

This is the same example that we have seen above. The only difference here is that in this example, overriding is actually happening since these methods are **not** static, private and final. In case of overriding the call to the overriden method is determined at runtime by the type of object thus late binding happens. Lets see an example to understand this:

class Human{

//Overridden Method

public void walk()

{

System.out.println("Human walks");

}

}

class Demo extends Human{

//Overriding Method

public void walk(){

System.out.println("Boy walks");

}

public static void main( String args[]) {

/\* Reference is of Human type and object is

\* Boy type

\*/

Human obj = new Demo();

/\* Reference is of HUman type and object is

\* of Human type.

\*/

Human obj2 = new Human();

obj.walk();

obj2.walk();

}

}

Output:

Boy walks

Human walks

As you can see that the output is different than what we saw in the static binding example, because in this case while creation of object obj the type of the object is determined as a Boy type so method of Boy class is called. Remember the type of the object is determined at the runtime.

## Static Binding vs Dynamic Binding

Lets discuss the **difference between static and dynamic binding in Java**.

1. Static binding happens at compile-time while dynamic binding happens at runtime.
2. Binding of private, static and final methods always happen at compile time since these methods cannot be overridden. When the method overriding is actually happening and the reference of parent type is assigned to the object of child class type then such binding is resolved during runtime.
3. The binding of [overloaded methods](https://beginnersbook.com/2013/05/method-overloading/) is static and the binding of overridden methods is dynamic.