**Polymorphism in java** is a concept by which we can perform a single action by 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.

**Type of Polymorphism:**

1.Compile Time polymorphism(overloading)

1.1.method overloading

1.2.constructor overloading

2.RunTime Polymorphism(overriding)

2.1.method override

**Method Overloading in Java with examples**

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different. constructor overloading that allows a class to have more than one constructors having different argument lists.

**Argument lists could differ in –**  
1. Number of parameters.  
2. Data type of parameters.  
3. Sequence of Data type of parameters.

**Method overloading** is also known as **Static Polymorphism**.

**Points to Note:**  
1. Static Polymorphism is also known as compile time binding or early binding.  
2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

## Runtime Polymorphism in Java

**Runtime polymorphism** or **Dynamic Method Dispatch** 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.

Let's first understand the upcasting before Runtime Polymorphism.

### UpcastingUpcasting in java:- When 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

# Method overriding in java with example

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).

# Declaring a method in ****subclass**** which is already present in ****parent class**** is known as method overriding. Earlier we shared method overloading in java. In this tutorial we will see **method overriding with examples**.

class Human{

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

obj.eat();

}

}

Output:Boy is eating

**Advantage of method overriding**

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class(base class).

#### Rules of method overriding in Java

1. Argument list: The argument list of overriding method must be same as that of the method in parent class. The data types of the arguments and their sequence should be maintained as it is in the overriding method.
2. Access Modifier: The Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class.  For e.g. if the Access Modifier of base class method is public then the overriding method (child class method ) cannot have private, protected and default Access modifier as all of the three are more restrictive than public.  
   For e.g. This is not allowed as child class disp method is more restrictive(protected) than base class(public)
3. private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.
4. Overriding method (method of child class) can throw any unchecked exceptions, regardless of whether the overridden method(method of parent class) throws any exception or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. We will discuss this in detail with example in the upcoming tutorial.
5. Binding of overridden methods happen at runtime which is known as dynamic binding.
6. If a class is extending an abstract class or implementing an interface then it has to override all the abstract methods unless the class itself is a abstract class.

### Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism.

### Why can we not override static method?

### It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

### Can we override java main method?

No, because the main is a static method.

# Difference between method Overloading and Overriding in java

1. Method overloading in java
2. Method overriding in java
3. Overloading vs Overriding in Java
4. Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.
5. Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.(can overload static methos,but can’t override static method).
6. The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
7. Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.
8. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
9. private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
10. Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the overriding method can have more specific return type.
11. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

# Final Keyword

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.

## 1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

## 2) Java final method

If you make any method as final, you cannot override it.

## 3) Java final class

If you make any class as final, you cannot extend it.

### Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

**class** Bike{

**final** **void** run(){System.out.println("running...");}

}

**class** Honda2 **extends** Bike{

**public** **static** **void** main(String args[]){

**new** Honda2().run();

   }

}

### Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

**class** Student{

**int** id;

String name;

**final** String PAN\_CARD\_NUMBER;

...

}

### Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

**class** Bike10{

**final** **int** speedlimit;//blank final variable

  Bike10(){

  speedlimit=70;

  System.out.println(speedlimit);

  }

**public** **static** **void** main(String args[]){

**new** Bike10();

 }

}

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

### Example of static blank final variable

**class** A{

**static** **final** **int** data;//static blank final variable

**static**{ data=50;}

**public** **static** **void** main(String args[]){

    System.out.println(A.data);

 }

}

### Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

**class** Bike11{

**int** cube(**final** **int** n){

   n=n+2;//can't be changed as n is final

   n\*n\*n;

  }

**public** **static** **void** main(String args[]){

    Bike11 b=**new** Bike11();

    b.cube(5);

 }

}

### Q) Can we declare a constructor final?

No, because constructor is never inherited.

# Difference between final, finally and finalize

The final, finally, and finalize are keywords in Java that are used in exception handling. Each of these keywords has a different functionality. The basic difference between final, finally and finalize is that the [**final**](https://www.javatpoint.com/final-keyword) is an access modifier, [**finally**](https://www.javatpoint.com/finally-block-in-exception-handling) is the block in Exception Handling and [**finalize**](https://www.javatpoint.com/java-object-finalize-method) is the method of object class.

Along with this, there are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no.** | **Key** | **final** | **finally** | **finalize** |
| 1. | Definition | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| 2. | Applicable to | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| 3. | Functionality | (1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited. | (1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| 4. | Execution | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.  It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed |

# What is Inheritance in Java Programming?

**Inheritance**is one of the feature of Object-Oriented Programming (OOPs). Inheritance allows a class to use the properties and methods of another class.

In other words, the derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class.

Inheritance is a compile-time mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.

Why use inheritance :

1.for method overriding(so runtime polymorphism can be achieved).

2.for code reusability.

Java supports -single inheritance,multilevel inheritance,hierarchical inheritance

Java doesn’t supports-multiple , hybrid inheritance

Single inheritance- when a class inherits another class,it is known as a single inheritance.

Multi level inheritance-when there is a chain of inheritance, it is known as multilevel inheritance.

Hierarchical inheritance- when two or more classes inherits a single class, it is known as hierarchical inheritance.

# Abstract Classes and Methods in Java

A class that is declared using “abstract” keyword is known as abstract class. It may or may not include abstract methods which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods ( without an implementation, without braces, and followed by a semicolon). An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).

**Abstract class declaration**  
Specifying **abstract keyword** before the class during declaration, makes it abstract. Have a look at below code:

// Declaration using abstract keyword

abstract class AbstractDemo{

// Concrete method: body and braces

public void myMethod(){

//Statements here

}

// Abstract method: without body and braces

abstract public void anotherMethod();

}

Since abstract class allows concrete methods as well, it does not provide 100% abstraction. You can say that it provides partial abstraction(0 to 100%). Interfaces are used for 100% abstraction (full abstraction)

**Remember two rules**:  
1) If the class is having few abstract methods and few concrete methods: declare it as abstract class.  
2) If the class is having only abstract methods: declare it as interface.

**Error!! – Object creation of abstract class is not allowed**  
As discussed above, we cannot instantiate an abstract class. The following code throws an error.

abstract public class AbstractDemo{

public void myMethod(){

System.out.println("Hello");

}

abstract public void anotherMethod();

}

public class ConcreteDemo{

public void anotherMethod() {

System.out.print("Abstract method");

}

public static void main(String args[])

{

//Can't create object of abstract class - error!

AbstractDemo obj = new AbstractDemo();

obj.display();

}

}

Output:

Unresolved compilation problem: Cannot instantiate the type AbstractEx1

Note: The class that extends the abstract class, have to implement all the abstract methods of abstract class, else they can be declared abstract in the class as well.

**Why we need an abstract class?**

Let me explain this with an example. Suppose there is a class Animal and there are few other classes like Cat, Dog and Horse. These classes extends Animal class so basically they are having few common habits(methods in technically) which they are inheriting from Animal class. Now, if you have understood the above example then you would have been able to figure out that **creating object of Animal class has no significance** as you can’t judge that the **new** object of Animal class will represent which animal. Hence for such kind of scenarios we generally creates an **abstract class** and later **concrete classes** extends these classes and overrides their methods accordingly and can have their own methods as well.

#### Abstract vs Concrete

A class which is not abstract is referred as **Concrete class**. In the above example which I explained – Animal is a abstract class and Cat, Dog and Horse are concrete classes.

**Key Points:**

1. An abstract class has no use until unless it is extended by some other class.
2. If you declare an **abstract method** (discussed below) in a class then you must declare the class abstract as well. you can’t have abstract method in a **non-abstract class**. It’s vice versa is not always true: If a class is not having any abstract method then also it can be marked as abstract.
3. Abstract class can have non-abstract method (concrete) as well.

## Abstract methods

Well, we already discussed about abstract methods in the above section. Lets take few examples to understand it better.

**syntax:**

public abstract void display();

**Points to remember about abstract method:**  
1) Abstract method has no body.  
2) Always end the declaration with a **semicolon**(;).  
3) It must be overridden. An abstract class must be extended and in a same way abstract method must be overridden.  
4) Abstract method must be in a abstract class.

**Note:** The class which is extending abstract class must override (or implement) all the abstract methods.**Example of Abstract class and method**

abstract class Demo1{

public void disp1(){

System.out.println("Concrete method of abstract class");

}

abstract public void disp2();

}

class Demo2 extends Demo1{

/\* I have given the body to abstract method of Demo1 class

It is must if you don't declare abstract method of super class

compiler would throw an error\*/

public void disp2()

{

System.out.println("I'm overriding abstract method");

}

public static void main(String args[]){

Demo2 obj = new Demo2();

obj.disp2();

}

}

Output:

I'm overriding abstract method

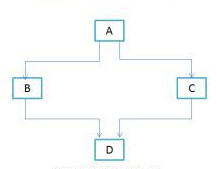
# Does Java support Multiple inheritance?

### Why Java doesn’t support multiple inheritance?

C++ , Common lisp and few other languages supports multiple inheritance while java doesn’t support it. It is just to **remove ambiguity**, because **multiple inheritance** can cause ambiguity in few scenarios. One of the most common scenario is **Diamond problem.**

**In other words,to reduce the complexity,and simplify the language,multiple inheritance is not supported.**

**What is diamond problem?**  
Consider the below diagram which shows multiple inheritance as Class D extends both Class B & C. Now lets assume we have a method in class A and class B & C overrides that method in their own way. **Wait!! here the problem comes**– Because D is extending both B & C so if D wants to use the same method which method would be called (the overridden method of B or the overridden method of C). Ambiguity. That’s the main reason why Java doesn’t support multiple inheritance.



# Interface in java with example programs

### What is an interface?

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default. We will discuss these points in detail, later in this post.

Note:1. **java interface also represent the IS-A relationship.**

**2. When interface will load, then it generate as a .class file.**

**3. since java 8, we can have default and static methods in an interface.**

**4. since 9, we can have private methods in an interface.**

### What is the use of interfaces?

As mentioned above they are used for abstraction. Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not support multiple inheritance, using interfaces we can achieve this as a class can implement more than one interfaces, however it cannot extend more than one classes.

**Declaration**  
Interfaces are declared by specifying a keyword “interface”. E.g.:

interface MyInterface

{

/\* All the methods are public abstract by default

\* Note down that these methods are not having body

\*/

public void method1();

public void method2();

}

### Interface Implementation

This is how a class implements an interface. It has to provide the body of all the methods that are declared in interface.  
Note: class implements interface but an interface extends another interface.

interface MyInterface

{

public void method1();

public void method2();

}

class XYZ implements MyInterface

{

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new XYZ();

//XYZ obj = new XYZ();

obj. method1();

}

}

Output:

implementation of method1

### Interface and Inheritance

As discussed above, an interface can not implement another interface. It has to extend the other interface if required. See the below example where we have two interfaces Inf1 and Inf2. Inf2 extends Inf1 so If class implements the Inf2 it has to provide implementation of all the methods of interfaces Inf1 and Inf2.

public interface Inf1{

public void method1();

}

public interface Inf2 extends Inf1 {

public void method2();

}

public class Demo implements Inf2{

public void method1(){

//Implementation of method1

}

public void method2(){

//Implementation of method2

}

}

**Key points:** Here are the key points to remember about interfaces:  
1) We can’t instantiate an interface in java.(can’t create object)

2) Interface provides complete abstraction(100%) as none of its methods can have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

3) implements keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

6) Interface cannot be declared as private, protected or transient.

7) All the interface methods are by default **abstract and public**.

8) Variables declared in interface are **public, static and final** by default.

interface Try

{

int a=10;

public int a=10;

public static final int a=10;

final int a=10;

static int a=0;

}

All of the above statements are identical.

9) Interface variables must be initialized at the time of declaration otherwise compiler will through an error.

interface Try

{

int x;//Compile-time error

}

Above code will throw a compile time error as the value of the variable x is not initialized at the time of declaration.

10) Inside any implementation class, you cannot change the variables declared in interface because by default, they are public, static and final. Here we are implementing the interface “Try” which has a variable x. When we tried to set the value for variable x we got compilation error as the variable x is public static **final** by default and final variables can not be re-initialized.

class Sample implements Try

{

public static void main(String args[])

{

x=20; //compile time error

}

}

11) Any interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

12) A **class** can implement any **number of interfaces**.

13) If there are **two or more same methods** in two interfaces and a class implements both interfaces, implementation of the method once is enough.

interface A

{

public void aaa();

}

interface B

{

public void aaa();

}

class Central implements A,B

{

public void aaa()

{

//Any Code here

}

public static void main(String args[])

{

//Statements

}

}

14) A class cannot implement two interfaces that have methods with same name but different return type.

interface A

{

public void aaa();

}

interface B

{

public int aaa();

}

class Central implements A,B

{

public void aaa() // error

{

}

public int aaa() // error

{

}

public static void main(String args[])

{

}

}

15) Variable names conflicts can be resolved by interface name e.g:

interface A

{

int x=10;

}

interface B

{

int x=100;

}

class Hello implements A,B

{

public static void Main(String args[])

{

// reference to x is ambiguous both variables are x

System.out.println(x);

System.out.println(A.x);//10

System.out.println(B.x);//100

}

}

#### Benefits of having interfaces:

Following are the advantages of using interfaces:

1. Without bothering about the implementation part, we can achieve the security of implementation
2. In java, **multiple inheritance** is not allowed, However by using interfaces you can achieve the same . A class can extend only one class but can implement any number of interfaces. It saves you from Deadly Diamond of Death(DDD) problem.

# Difference Between Abstract Class and Interface in Java

|  |  |  |
| --- | --- | --- |
|  | **abstract Classes** | **Interfaces** |
| 1 | abstract class can extend only one class or one abstract class at a time | interface can extend any number of interfaces at a time |
| 2 | abstract  class  can extend from a class or from an abstract class | interface can extend only from an interface |
| 3 | abstract  class  can  have  both  abstract and concrete methods | interface can  have only abstract methods |
| 4 | A class can extend only one abstract class | A class can implement any number of interfaces |
| 5 | In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract | In an interface keyword ‘abstract’ is optional to declare a method as an abstract |
| 6 | abstract  class can have  protected , public and public abstract methods | Interface can have only public abstract methods i.e. by default |
| 7 | abstract class can have  static, final  or static final  variable with any access specifier | interface  can  have only static final (constant) variable i.e. by default |

# Java 8 Default method in interface:

# Since java 8,we can have method body in interface,but we need to make it default method.

# Since java 8,we can have static method in interface.

# What is marker or tagged interface?

# An interface which has no member is known as a marker or tagged interface. for example Serializable,cloneable,Remote etc.They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

# Nested Interface in java:

# An interface can have another interface which is known as a nested interface.

# Interface printable{

# void print();

# interface MessagePrintable{

# void msg();

# }

# }

# Polymorphism in Java

**Polymorphism in java** is a concept by which we can perform a single action by 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 static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

## Runtime Polymorphism in Java

**Runtime polymorphism** or **Dynamic Method Dispatch** 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.

Let's first understand the upcasting before Runtime Polymorphism.

### UpcastingUpcasting in java:- When 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

1.is -A relationship

2.same method name

3.same parameter list

# Aggregation in Java

If a class have an entity/class reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

### Why use Aggregation?

* For Code Reusability.

### When use Aggregation?

* Code reuse is also best achieved by aggregation when there is no is-a relationship.
* Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Address ad1=new Address(“kolkata”,”wb”,”india”);

Emplolee emp1=new Employee(101,”prity chatterjee”,30000,ad1);

City=Kolkata

State=”wb”

Country=india

6789

Id=101

Name=priya chatterjee

Salary=30000

Add= Ad1(6789)

|  |
| --- |
|  |
|  |
| Emp1  4567 |
| Ad1  6789 |