# Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%) of data hiding.
2. Interface (100%) of pure data hiding.

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.

### Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

1. **abstract** **void** printStatus();//no method body and abstract

Example of Abstract class that has an abstract method

abstract class Base {

Base()

{

System.out.println("Base Constructor Called");

}

abstract void fun();

}

class Derived extends Base {

Derived()

{

System.out.println("Derived Constructor Called");

}

void fun()

{

System.out.println("Derived fun() called");

}

}

class AbstractClass {

public static void main(String args[])

{

Derived d = new Derived();

d.fun();

}

}

**Example:**

 abstract class Bike{

   Bike(){System.out.println("bike is created");}

   abstract void run();

   void changeGear(){System.out.println("gear changed");}

 }

 class Honda extends Bike{

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

 }

 class TestAbstraction2{

 public static void main(String args[]){

  Bike obj = new Honda();

  obj.run();

  obj.changeGear();

 }

}

**Example:**

import java.io.\*;

abstract class B {

abstract class C {

abstract void myAbstractMethod();

}

}

class D extends B {

class E extends C {

void myAbstractMethod()

{

System.out.println(

"Inside abstract method implementation");

}

}

}

public class Main {

public static void main(String args[])

{

// Instantiating the outer class

D outer = new D();

D.E inner = outer.new E();

inner.myAbstractMethod();

}

}

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

Java Interface also **represents the IS-A relationship**

**An interface allows you to specify whatever a class must do, but not how to do.**

**An interface can implement (but not extends) any number of interfaces, thus supporting multiple inheritances indirectly.**

**Once an “interface” has been defined, one or more classes can implement that interface. To implement an interface, the “implements” keyword is included in the class definition, and then the methods defined by the interface are created.**

### Syntax:

**interface** <interface\_name>{

// declare methods that abstract

}

**Example to implement interface with class:**

Interface interfacename

{

methods(parameters); constants;

}

class classname implements interfacename

{

}

#### **The relationship between classes and interfaces**



## **Java Interface Example**

interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

A6 obj = new A6();

obj.print();

}

}

Output:

Hello

## Java Interface Example: Drawable

interface Drawable{

void draw();

}

//Implementation: by second user

class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class Circle implements Drawable{

public void draw(){System.out.println("drawing circle");}

}

//Using interface: by third user

class TestInterface1{

public static void main(String args[]){

Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()

d.draw();

}}

Output:

drawing circle

## Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



interface Printable{

void print();

}

interface Showable{

void show();

}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

A7 obj = new A7();

obj.print();

obj.show();

}

}

Output:Hello

Welcome

Example:

interface Printable{

void print();

}

interface Showable extends Printable{

void show();

}

class TestInterface4 implements Showable{

public void print(){System.out.println("Hello");}

public void show(){System.out.println("Welcome");}

public static void main(String args[]){

TestInterface4 obj = new TestInterface4();

obj.print();

obj.show();

}

}

Output:

Hello

Welcome

**Example:**

import java.io.\*;

interface Vehicle {

void changeGear(int a);

void speedUp(int a);

void applyBrakes(int a);

}

class Bicycle implements Vehicle{

int speed;

int gear;

public void changeGear(int newGear){

gear = newGear;

}

public void speedUp(int increment){

speed = speed + increment;

}

public void applyBrakes(int decrement){

speed = speed - decrement;

}

public void printStates() {

System.out.println("speed: " + speed

+ " gear: " + gear);

}

}

class Bike implements Vehicle {

int speed;

int gear;

public void changeGear(int newGear){

gear = newGear;

}

public void speedUp(int increment){

speed = speed + increment;

}

public void applyBrakes(int decrement){

speed = speed - decrement;

}

public void printStates() {

System.out.println("speed: " + speed

+ " gear: " + gear);

}

}

class InterfaceExample {

public static void main (String[] args) {

Bicycle bicycle = new Bicycle();

bicycle.changeGear(2);

bicycle.speedUp(3);

bicycle.applyBrakes(1);

System.out.println("Bicycle present state :");

bicycle.printStates();

Bike bike = new Bike();

bike.changeGear(1);

bike.speedUp(4);

bike.applyBrakes(3);

System.out.println("Bike present state :");

bike.printStates();

}

}

static methods in interfaces that can be called independently without an object. **Note**: these methods are not inherited.

**Example:**

interface In1

{

final int a = 10;

static void display()

{

System.out.println("hello");

}

}

class TestClass implements In1

{

public static void main (String[] args)

{

In1.display();

}

}

Inheritance in Java

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java)
* For Code Reusability.

Terms used in Inheritance

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

**class** Subclass-name **extends** Superclass-name

{

   //methods and fields

}

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

**class** Employee{

**float** salary=40000;

}

**class** Programmer **extends** Employee{

**int** bonus=10000;

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

   Programmer p=**new** Programmer();

   System.out.println("Programmer salary is:"+p.salary);

   System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

Programmer salary is:40000.0

Bonus of programmer is:10000

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



Note: Multiple inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void***bark(){*System.out.println("barking...");}

}

**class** TestInheritance{

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

Dog d=**new** Dog();

d.bark();

d.eat();

}}

Output:

barking...

eating...

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");}

}

**class** BabyDog **extends** Dog{

**void** weep(){System.out.println("weeping...");}

}

**class** TestInheritance2{

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

BabyDog d=**new** BabyDog();

d.weep();

d.bark();

d.eat();

}}

Output:

weeping...

barking...

eating...

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");}

}

**class** Cat **extends** Animal{

**void** meow(){System.out.println("meowing...");}

}

**class** TestInheritance3{

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

Cat c=**new** Cat();

Dog d=new Dog();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Output:

meowing...

eating...

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

**class** A{

**void** msg(){System.out.println("Hello");}

}

**class** B{

**void** msg(){System.out.println("Welcome");}

}

**class** C **extends** A,B{//suppose if it were

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

   C obj=**new** B();

   obj.msg();//Now which msg() method would be invoked?

}

}