Inheritance is an important pillar of OOP (Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features (fields and methods) of another

**Important terminology:**

* **Super Class:**The class whose features are inherited is known as superclass (or a base class or a parent class).
* **Sub Class:** The class that inherits the other class is known as subclass (or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:**Inheritance supports the concept of “reusability”, i.e., when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**How to use inheritance in Java**

The keyword used for inheritance is **extends**.

**Syntax:**

class derived-class extends base-class

{

//methods and fields

}

**Example:**In below example of inheritance, class Bicycle is a base class, class MountainBike is a derived class which extends Bicycle class and class Test is a driver class to run program.

//base class

**public** **class** Bicycle {

// the Bicycle class has two fields

**public** **int** gear;

**public** **int** speed;

// the Bicycle class has one constructor

**public** Bicycle(**int** gear, **int** speed) {

**this**.gear = gear;

**this**.speed = speed;

}

// the Bicycle class has three methods

**public** **void** applyBrake(**int** decrement) {

speed -= decrement;

}

**public** **void** speedUp(**int** increment) {

speed += increment;

}

// toString() method to print info of Bicycle

**public** String toString() {

**return** ("No of gears are " + gear + "\n" + "speed of bicycle is " + speed);

}

}

// derived class

**public** **class** MountainBike **extends** Bicycle {

// the MountainBike subclass adds one more field

**public** **int** seatHeight;

// the MountainBike subclass has one constructor

**public** MountainBike(**int** gear, **int** speed, **int** startHeight) {

// invoking base-class(Bicycle) constructor

**super**(gear, speed);

seatHeight = startHeight;

}

// the MountainBike subclass adds one more method

**public** **void** setHeight(**int** newValue) {

seatHeight = newValue;

}

// overriding toString() method

// of Bicycle to print more info

@Override

**public** String toString() {

**return** (**super**.toString() + "\nseat height is " + seatHeight);

}

}

//driver class

**public** **class** Test {

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

MountainBike mb = **new** MountainBike(3, 100, 25);

System.***out***.println(mb.toString());

}

}

**Output**

No of gears are 3

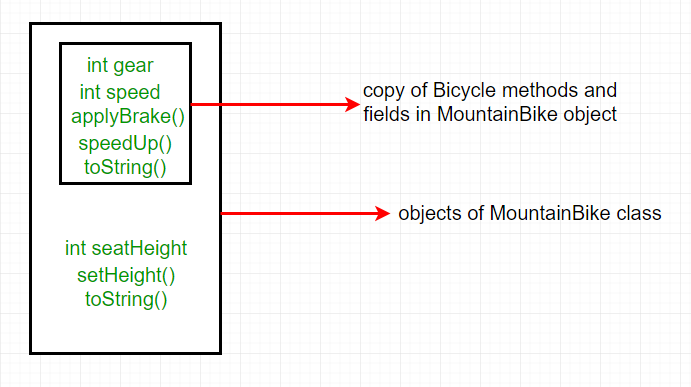
speed of bicycle is 100

seat height is 25

In the above program, when an object of MountainBike class is created, a copy of all methods and fields of the superclass acquire memory in this object. That is why by using the object of the subclass we can also access the members of a superclass.

Please note that during inheritance only object of subclass is created, not the superclass. For more, refer [Java Object Creation of Inherited Class](https://www.geeksforgeeks.org/gfact-52-java-object-creation-of-inherited-classes/).

**Illustrative image of the program:**

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritence1.png)

==========================-=============================-=============================

**Java Object Creation of Inherited Class**

// super class

**public** **class** Fruit {

**public** Fruit() {

System.***out***.println("Super class constructor");

System.***out***.println("Super class object hashcode :" + **this**.hashCode());

System.***out***.println(**this**.getClass().getName());

}

}

//sub class

**class** Apple **extends** Fruit {

**public** Apple() {

System.***out***.println("Subclass constructor invoked");

System.***out***.println("Sub class object hashcode :" + **this**.hashCode());

System.***out***.println(**this**.hashCode() + " " + **super**.hashCode());

System.***out***.println(**this**.getClass().getName() + " " + **super**.getClass().getName());

}

}

//driver class

**public** **class** JavaObjectCreationTest {

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

**new** Apple();

}

}

Outpot

Super class constructor

Super class object hashcode :1829164700

com.mahendra.core.p002.oops.inheritance.Apple

Subclass constructor invoked

Sub class object hashcode :1829164700

1829164700 1829164700

As we can see that both super class (Fruit) object hashcode and subclass(Apple) object hashcode are same, so only one object is created. This object is of class Apple(subclass) as when we try to print name of class which object is created, it is printing Apple which is subclass.

-===========================================================-

In practice, inheritance and [polymorphism](https://www.geeksforgeeks.org/overriding-in-java/) are used together in java to achieve fast performance and readability of code.

**Types of Inheritance in Java**

Below are the different types of inheritance which are supported by Java.

1. **Single Inheritance:**In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.

**package** com.mahendra.core.p002.oops.inheritance.types;

**class** T1One {

**public** **void** m1() {

System.***out***.println("T1One. m1()");

}

}

**class** T1Two **extends** T1One {

**public** **void** m2() {

System.***out***.println("T1Two. m2()");

}

}

// Driver class

**public** **class** T1\_SingleInheritance {

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

T1Two g = **new** T1Two();

g.m1();

g.m2();

g.m1();

}

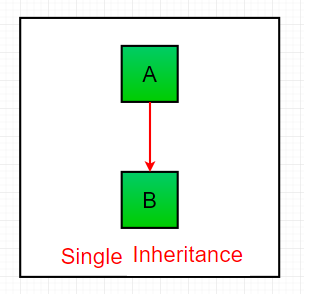
}

**Output**

T1One. m1()

T1Two. m2()

T1One. m1()

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance1.png)

=======================================-=======================================

**Multilevel Inheritance:**In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).

**class** T2One {

**public** **void** m1() {

System.***out***.println("T2One. m1()");

}

}

**class** T2Two **extends** T2One {

**public** **void** m2() {

System.***out***.println("T2Two. m2()");

}

}

**class** T2Three **extends** T2Two {

**public** **void** m3() {

System.***out***.println("T2Three. m3()");

}

}

**public** **class** T2\_MultilevelInheritance {

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

T2Three g = **new** T2Three();

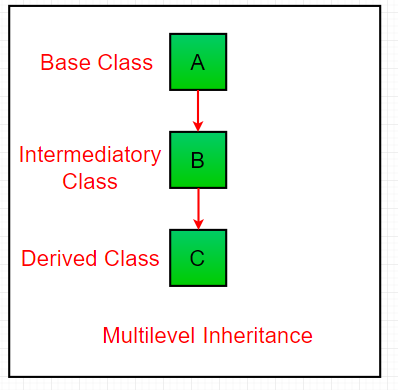
g.m1();

g.m2();

g.m1();

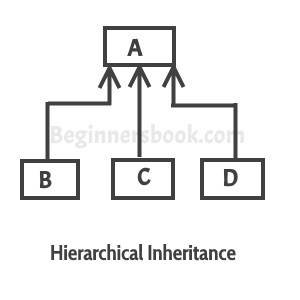
}

}



-========================-============================-=========================

**Hierarchical Inheritance:**In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass



**class** T3One {

**public** **void** m1() {

System.***out***.println("T1One. m1()");

}

}

**class** T3Two **extends** T3One {

**public** **void** m2() {

System.***out***.println("T3Two. m1()");

}

}

**class** T3Three **extends** T3One {

**public** **void** m3() {

System.***out***.println("T3Three. m1()");

}

}

**class** T3Four **extends** T3One {

**public** **void** m4() {

System.***out***.println("T3Four. m1()");

}

}

**public** **class** T3\_HierarchicalInheritance {

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

T3Two t3TwoObj = **new** T3Two();

T3Three t3ThreeObj = **new** T3Three();

T3Four t3FourObj = **new** T3Four();

// All classes can access the method of class A

t3TwoObj.m1();

t3ThreeObj.m1();

t3FourObj.m1();

}

}

------------=====================-=============================-=============

* [**Multiple Inheritance**](https://www.geeksforgeeks.org/java-and-multiple-inheritance/)**(Through Interfaces):**In Multiple inheritances, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. In java, we can achieve multiple inheritances only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/). In the image below, Class C is derived from interface A and B.

**interface** IOne {

**public** **void** m1();

}

**interface** ITtwo {

**public** **void** m1();

}

**interface** IThree **extends** IOne, ITtwo {

**public** **void** m1();

}

**class** Child **implements** IThree {

@Override

**public** **void** m1() {

System.***out***.println("Child.m1()");

}

**public** **void** m2() {

System.***out***.println("Child.m2()");

}

}

**public** **class** T4\_MultipleInheritance {

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

Child c = **new** Child();

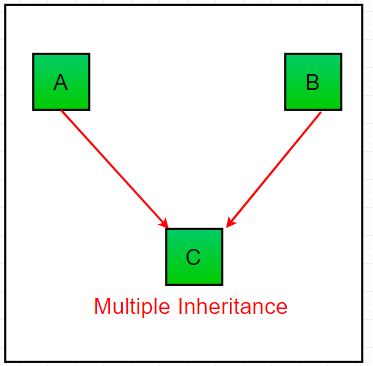
c.m1();

c.m2();

c.m1();

}

}



===--============================================-=================================

**Hybrid Inheritance(Through Interfaces)**

* It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/).

**interface** T5IOne {

**public** **void** m1();

}

**interface** T5ITtwo **extends** T5IOne {

**public** **void** m1();

}

**interface** T5IThree **extends** T5IOne {

**public** **void** m1();

}

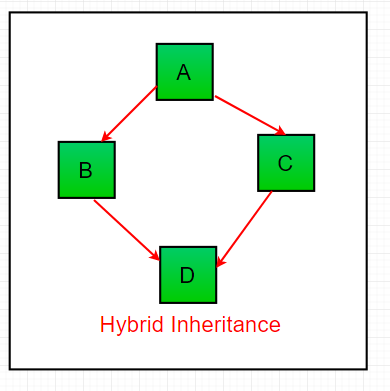
**public** **class** T5\_HybridInheritance **implements** T5ITtwo, T5IThree {

@Override

**public** **void** m1() {

}

}

[](https://media.geeksforgeeks.org/wp-content/uploads/inheritance-1.png)

**Important facts about inheritance in Java**

* **Default superclass**: Except [Object](https://www.geeksforgeeks.org/object-class-in-java/) class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. Although with interfaces, multiple inheritance is supported by java.
* **Inheriting Constructors:**A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

**Java IS-A type of Relationship.**

IS-A is a way of saying: This object is a type of that object. Let us see how the extends keyword is used to achieve inheritance.

**public class** SolarSystem {

}

**public class** Earth **extends** SolarSystem {

}

**public class** Mars **extends** SolarSystem {

}

**public class** Moon **extends** Earth {

}

Now, based on the above example, in Object-Oriented terms, the following are true :-

SolarSystem the superclass of Earth class.

SolarSystem the superclass of Mars class.

Earth and Mars are subclasses of SolarSystem class.

Moon is the subclass of both Earth and SolarSystem classes.

SolarSystem the superclass of Earth class.

SolarSystem the superclass of Mars class.

Earth and Mars are subclasses of SolarSystem class.

Moon is the subclass of both Earth and SolarSystem classes.

**class** SolarSystem {

}

**class** Earth **extends** SolarSystem {

}

**class** Mars **extends** SolarSystem {

}

**public** **class** T5\_HybridInheritance **extends** Earth {

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

SolarSystem s = **new** SolarSystem();

Earth e = **new** Earth();

Mars m = **new** Mars();

System.***out***.println(s **instanceof** SolarSystem);

System.***out***.println(e **instanceof** Earth);

System.***out***.println(m **instanceof** SolarSystem);

}

}

**What all can be done in a Subclass?**

In sub-classes we can inherit members as is, replace them, hide them, or supplement them with new members: 

* The inherited fields can be used directly, just like any other fields.
* We can declare new fields in the subclass that are not in the superclass.
* The inherited methods can be used directly as they are.
* We can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus [overriding](https://www.geeksforgeeks.org/overriding-in-java/) it (as in example above, *toString()* method is overridden).
* We can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus [hiding](https://www.geeksforgeeks.org/g-fact-63/) it.
* We can declare new methods in the subclass that are not in the superclass.
* We can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword [super](http://quiz.geeksforgeeks.org/super-keyword/).