# 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.

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

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

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.

EXAMPLE:

1. **class** Employee{
2. **float** salary=40000;
3. }
4. **class** Programmer **extends** Employee{
5. **int** bonus=10000;
6. **public** **static** **void** main(String args[]){
7. Programmer p=**new** Programmer();
8. System.out.println("Programmer salary is:"+p.salary);
9. System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }

## Types of inheritance in java

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

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



## 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.

*File: TestInheritance.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** TestInheritance{
8. **public** **static** **void** main(String args[]){
9. Dog d=**new** Dog();
10. d.bark();
11. d.eat();
12. }}

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.

*File: TestInheritance2.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** BabyDog **extends** Dog{
8. **void** weep(){System.out.println("weeping...");}
9. }
10. **class** TestInheritance2{
11. **public** **static** **void** main(String args[]){
12. BabyDog d=**new** BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
16. }}

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.

*File: TestInheritance3.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** Cat **extends** Animal{
8. **void** meow(){System.out.println("meowing...");}
9. }
10. **class** TestInheritance3{
11. **public** **static** **void** main(String args[]){
12. Cat c=**new** Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}

Output:

meowing...

eating...

## 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.

*File: TestInheritance.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** TestInheritance{
8. **public** **static** **void** main(String args[]){
9. Dog d=**new** Dog();
10. d.bark();
11. d.eat();
12. }}

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.

*File: TestInheritance2.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** BabyDog **extends** Dog{
8. **void** weep(){System.out.println("weeping...");}
9. }
10. **class** TestInheritance2{
11. **public** **static** **void** main(String args[]){
12. BabyDog d=**new** BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
16. }}

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.

*File: TestInheritance3.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** Cat **extends** Animal{
8. **void** meow(){System.out.println("meowing...");}
9. }
10. **class** TestInheritance3{
11. **public** **static** **void** main(String args[]){
12. Cat c=**new** Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}

Output:

meowing...

eating...

# Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

## Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

## 1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

1. **class** Animal{
2. String color="white";
3. }
4. **class** Dog **extends** Animal{
5. String color="black";
6. **void** printColor(){
7. System.out.println(color);//prints color of Dog class
8. System.out.println(**super**.color);//prints color of Animal class
9. }
10. }
11. **class** TestSuper1{
12. **public** **static** **void** main(String args[]){
13. Dog d=**new** Dog();
14. d.printColor();
15. }}

## 2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

1. **class** Animal{
2. **void** eat()
3. {
4. System.out.println("eating...");}
5. }
6. **class** Dog **extends** Animal
7. {
8. **void** eat()
9. {
10. System.out.println("eating bread...");
11. }
12. **void** bark()
13. {
14. System.out.println("barking...");
15. }
16. **void** work(){
17. **super**.eat();
18. bark();
19. }
20. }
21. **class** TestSuper2{
22. **public** **static** **void** main(String args[]){
23. Dog d=**new** Dog();
24. d.work();
25. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestSuper2)

Output:

eating...

barking...

## 3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

1. **class** Animal{
2. Animal(){System.out.println("animal is created");}
3. }
4. **class** Dog **extends** Animal{
5. Dog(){
6. **//super**();
7. System.out.println("dog is created");
8. }
9. }
10. **class** TestSuper3{
11. **public** **static** **void** main(String args[]){
12. Dog d=**new** Dog();
13. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestSuper3)

Output:

animal is created

dog is created

**Method Overloading in Java**

[Method Overloading](https://www.geeksforgeeks.org/different-ways-method-overloading-java/)is a **Compile time polymorphism**. In method overloading, more than one method shares the same method name with a different signature in the class. In method overloading, the return type can or can not be the same, but we have to change the parameter because, in java, we can not achieve method overloading by changing only the return type of the method.

**Example of Method Overloading:**

Java

*// Java Program to Implement*

*// Method Overloading*

**import** **java.io.\***;

**class** **MethodOverloadingEx** {

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

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

{

**return** a + b + c;

}

*// Main Function*

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

{

System.out.println("add() with 2 parameters");

*// Calling function with 2 parameters*

System.out.println(add(4, 6));

System.out.println("add() with 3 parameters");

*// Calling function with 3 Parameters*

System.out.println(add(4, 6, 7));

}

}

**Output**

add() with 2 parameters

10

add() with 3 parameters

17

**Method Overriding in Java**

[Method Overriding](https://www.geeksforgeeks.org/overriding-in-java/) is a **Run time polymorphism**. In method overriding, the derived class provides the specific implementation of the method that is already provided by the base class or parent class. In method overriding, the return type must be the same or co-variant (return type may vary in the same direction as the derived class).

**Example of Method Overriding:**

Java

*// Java Program to implement*

*// Method Overriding*

**import** **java.io.\***;

*// Base Class*

**class** **Animal** {

void eat()

{

System.out.println("eat() method of base class");

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

}

}

*// Inherited Class*

**class** **Dog** **extends** Animal {

void eat()

{

System.out.println("eat() method of derived class");

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

}

}

*// Driver Class*

**class** **MethodOverridingEx** {

*// Main Function*

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

{

Dog d1 = **new** Dog();

Animal a1 = **new** Animal();

d1.eat();

a1.eat();

*// Polymorphism: Animal reference pointing to Dog object*

Animal animal = **new** Dog();

*// Calls the eat() method of Dog class*

animal.eat();

}

}

**Output**

eat() method of derived class

Dog is eating.

eat() method of base class

eating.

eat() method of derived class

Dog is eating.