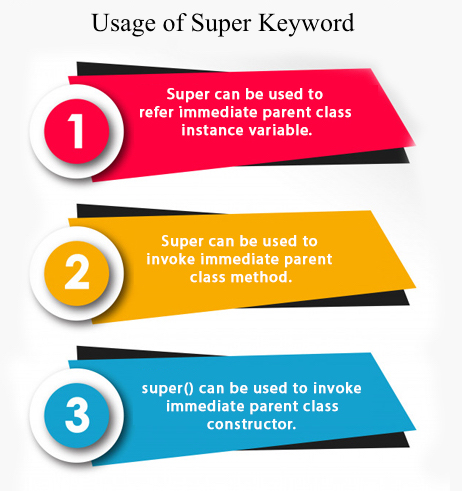
Super Keyword in Java

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. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestSuper1)

Output:

black

white

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

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

Output:

eating...

barking...

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

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. }}

Output:

animal is created

dog is created

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().



As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

**Another example of super keyword where super() is provided by the compiler implicitly.**

1. **class** Animal{
2. Animal(){System.out.println("animal is created");}
3. }
4. **class** Dog **extends** Animal{
5. Dog(){
6. System.out.println("dog is created");
7. }
8. }
9. **class** TestSuper4{
10. **public** **static** **void** main(String args[]){
11. Dog d=**new** Dog();
12. }}

Output:

animal is created

dog is created

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

1. **class** Person{
2. **int** id;
3. String name;
4. Person(**int** id,String name){
5. **this**.id=id;
6. **this**.name=name;
7. }
8. }
9. **class** Emp **extends** Person{
10. **float** salary;
11. Emp(**int** id,String name,**float** salary){
12. **super**(id,name);//reusing parent constructor
13. **this**.salary=salary;
14. }
15. **void** display(){System.out.println(id+" "+name+" "+salary);}
16. }
17. **class** TestSuper5{
18. **public** **static** **void** main(String[] args){
19. Emp e1=**new** Emp(1,"ankit",45000f);
20. e1.display();
21. }}

Output:

1 ankit 45000

# Instance initializer block

|  |
| --- |
| **Instance Initializer block** is used to initialize the instance data member. It run each time when object of the class is created. |
| The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block. |

#### Que) What is the use of instance initializer block while we can directly assign a value in instance data member? For example:

1. **class** Bike{
2. **int** speed=100;
3. }

## Why use instance initializer block?

|  |
| --- |
| Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc. |

### Example of instance initializer block

|  |
| --- |
| Let's see the simple example of instance initializer block that performs initialization. |

1. **class** Bike7{
2. **int** speed;
4. Bike7(){System.out.println("speed is "+speed);}
6. {speed=100;}
8. **public** **static** **void** main(String args[]){
9. Bike7 b1=**new** Bike7();
10. Bike7 b2=**new** Bike7();
11. }
12. }

Output:speed is 100

speed is 100

|  |
| --- |
| There are three places in java where you can perform operations:   1. method 2. constructor 3. block |

## What is invoked first, instance initializer block or constructor?

1. **class** Bike8{
2. **int** speed;
4. Bike8(){System.out.println("constructor is invoked");}
6. {System.out.println("instance initializer block invoked");}
8. **public** **static** **void** main(String args[]){
9. Bike8 b1=**new** Bike8();
10. Bike8 b2=**new** Bike8();
11. }
12. }

Output:instance initializer block invoked

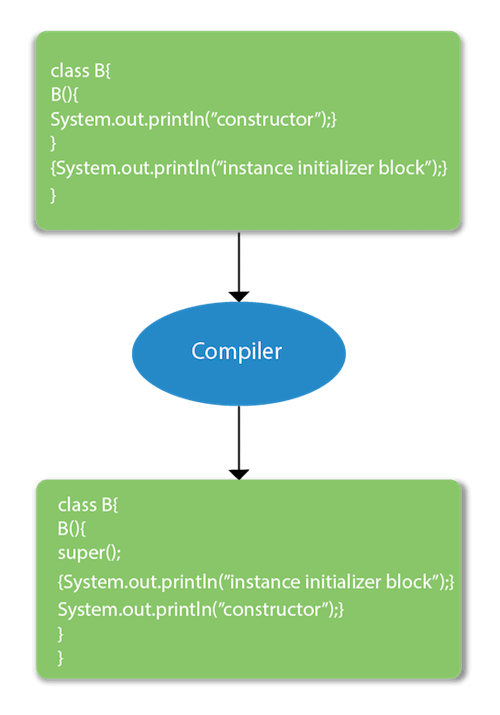
constructor is invoked

instance initializer block invoked

constructor is invoked

|  |
| --- |
| In the above example, it seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked. Let's understand it by the figure given below: |

#### Note: The java compiler copies the code of instance initializer block in every constructor.



## Rules for instance initializer block :

|  |
| --- |
| There are mainly three rules for the instance initializer block. They are as follows: |

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

## Program of instance initializer block that is invoked after super()

**class** A{

A(){

System.out.println("parent class constructor invoked");

}

}

**class** B2 **extends** A{

B2(){

**super**();

System.out.println("child class constructor invoked");

}

{System.out.println("instance initializer block is invoked");}

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

B2 b=**new** B2();

}

}

Output:parent class constructor invoked

instance initializer block is invoked

child class constructor invoked

## Another example of instance block

1. **class** A{
2. A(){
3. System.out.println("parent class constructor invoked");
4. }
5. }
7. **class** B3 **extends** A{
8. B3(){
9. **super**();
10. System.out.println("child class constructor invoked");
11. }
13. B3(**int** a){
14. **super**();
15. System.out.println("child class constructor invoked "+a);
16. }
18. {System.out.println("instance initializer block is invoked");}
20. **public** **static** **void** main(String args[]){
21. B3 b1=**new** B3();
22. B3 b2=**new** B3(10);
23. }
24. }

Output:parent class constructor invoked

instance initializer block is invoked

child class constructor invoked

parent class constructor invoked

instance initializer block is invoked

child class constructor invoked 10