**Java super keyword**

Super keyword refer to parent class instance or object so that we can access instance and static members of the Parent class within the child class.

Writing super keyword is sometimes mandatory, sometimes optional

If the subclass would like to call the super class method also, the subclass can do so using "**super**" keyword. "**super**" keyword is used by the subclass to call that of super class also when the methods or variables are overridden. "**super**" keyword can be used with instance variables and methods but not with classes.

If there is no confusion between instance variables of Parent class and instance variables of Child class then writing super keyword is optional. But In this case compiler will write this keyword and refer to instance variable of Child class.

But if there is a confusion between instance variables of Parent class and instance variables of Child class then writing super keyword is mandatory. Even In this case compiler will write this keyword only and refer to instance variable of Child class.

But if we want to refer to instance variable of Parent class we have to write superkeyword.

**Example:**

**//wap to demo on super keyword**

*class Parent{*

*int a=10;*

*int b=20;*

*void show1(){*

*System.out.println("Parent class show1() method");*

*}*

*void show2(){*

*System.out.println("Parent class show2() method");*

*}*

*}*

*class Child extends Parent{*

*int a=30;*

*int c=40;*

*void show1(){*

*System.out.println("Child class show1() method");*

*}*

*void show3(){*

*System.out.println("Child class show3() method");*

*}*

*void show4(){*

*int a=50;*

*int d=60;*

*System.out.println(super.a);//super is mandatory(Parent class a val)*

*System.out.println(this.a);//this is mandatory(Child class a value)*

*System.out.println(a);//local variable a value*

*System.out.println(b);//super or this is optional(Parent class b value)*

*System.out.println(c); //this is optional (Child class c value)*

*System.out.println(d); // local variable d value*

*super.show1(); //super is mandatory (calls Parent class show1())*

*this.show1();// this is optional(calls Child class show1())*

*show2(); // super or this optional (calls Parent class show2())*

*this.show3(); // this optional (calls Child class show3())*

*}*

*}*

*class SuperDemo1{*

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

*Child c= new Child();*

*c.show4();*

*}*

*}*

**3 ways of Using Super Keyword:**

**Super is used to refer the immediate parent of the class**. Whenever we create an object of the child class then the reference to the parent class will be created automatically.

We can user super keyword by using three ways –

* **Accessing Instance Variable of Parent Class**
* **Accessing Parent Class Method**
* **Accessing Parent Class Class Constructor**

**Way 1: Accessing Instance Variable of Parent Class using Super**

*package com.c4learn.inheritance;*

*public class ParentClass {*

***int*** *ageClass = 100;*

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

*ChildClass c1 = new ChildClass();*

*c1.display();*

*}*

*}*

*class ChildClass extends ParentClass{*

***int*** *ageClass = 200;*

*public* ***void*** *display() {*

*System.out.println("ageClass : " + ageClass);*

*}*

*}*

**Output of the Program**

ageClass : 200

**Consider the above example –**

1. We have same variable declared inside the parent class as well as in child class.
2. Whenever we try to access the variable using the object of child class, **always instance variable of child will be returned**.

Suppose we want to access the the same variable from the parent class then we can modify the display() method as below –

*public* ***void*** *display() {*

*System.out.println("ageClass : " + super.ageClass);*

*}*

**Way 2 : Accessing Parent Class Method using Super**

*package com.c4learn.inheritance;*

*public class ParentClass {*

***int*** *ageClass = 100;*

*public* ***int*** *getValue() {*

***return*** *20;*

*}*

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

*ChildClass c1 = new ChildClass();*

*c1.display();*

*}*

*}*

*class ChildClass extends ParentClass{*

***int*** *ageClass = 200;*

*public* ***int*** *getValue() {*

***return*** *50;*

*}*

*public* ***void*** *display() {*

*System.out.println("result : " + super.getValue());*

*}*

*}*

### Output :

*result : 20*

In this example we have same method declared inside parent and child class.

*public* ***void*** *display() {*

*System.out.println("result : " + super.getValue());*

*}*

In order to execute the parent method from the child class we need to use super keyword.

**Way 3: Accessing Parent Class Constructor using Super**

**Java super keyword & constructors**

**Understanding Constructors :**

*package com.c4learn.inheritance;*

*public class ParentClass {*

*public ParentClass(){*

*System.out.println("Hello Parent");*

*}*

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

*ChildClass c1 = new ChildClass();*

*}*

*}*

*class ChildClass extends ParentClass{*

*public ChildClass(){*

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

*}*

*}*

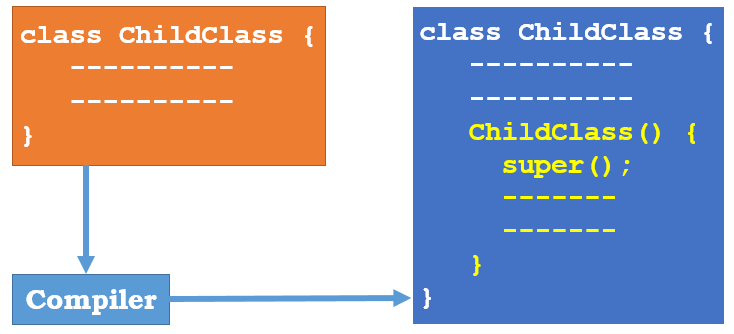
### Output of the Program :

*Hello Parent*

*Hello Child*

**Explanation of above program :**

When you create an instance of a child class, Java automatically calls the default [constructor of the parent class](http://c4learn.com/javaprogramming/constructors-initializing-an-class-object-in-java-programming/) before it executes the child class’s constructor.



**Consider the following code –**

*package com.c4learn.inheritance;*

*public class ParentClass {*

*public ParentClass(){*

*System.out.println("Hello Parent");*

*}*

*public ParentClass(****int*** *val){*

*System.out.println("Hello Parameter");*

*}*

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

*ChildClass c1 = new ChildClass();*

*}*

*}*

*class ChildClass extends ParentClass{*

*public ChildClass(){*

*super(5);*

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

*}*

*}*

Now In the above program, we have written two constructors one with no argument and another with single integer argument.

*public ParentClass(){*

*System.out.println("Hello Parent");*

*}*

*public ParentClass(****int*** *val){*

*System.out.println("Hello Parameter");*

*}*

As we have explicitly called the single argument constructor using the super keyword, **Parent Class’s Parameterized Constructor gets called**.

**Tip #1 : Super Must be First Statement**

If you use super to call the parent class constructor, you must put it as first statement in the constructor.

Following is the valid way of calling parent class constructor –

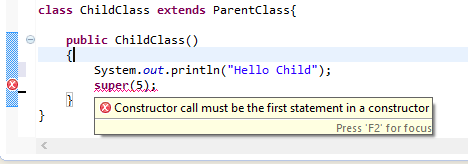
*public ChildClass(){*

*super(5);*

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

*}*

If we write it after other statements then it will throw compile time error –



**Tip #2 : Default constructor**

If you don’t explicitly call super, the compiler inserts a call to the [default constructor](http://c4learn.com/javaprogramming/constructors-initializing-an-class-object-in-java-programming/) of the base class. In that case, the base class must have a default constructor.

If the base class doesn’t have a default constructor, you will get compile time error.

**Note:**

1. A super keyword can access both instance and static members of the Parent class with in child class instance methods, but we can not use super keyword within any static method of child class.
2. We don't write any this keyword or super keyword compiler always write this keyword only but not any super keyword.
3. super keyword in child class always point to the variables or methods of its immediate parent class.

**Eg:**

*class Cone{*

*int a=10;*

*}*

*class Ctwo extends Cone{*

*int a=20;*

*}*

*class Cthree extends Ctwo{*

*int a=30;*

*void show(){*

*System.out.println(super.a); //20*

*Cone c = new Cone();*

*System.out.println(c.a);*

*}*

*}*

*class SuperDemo2{*

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

*Cthree c = new Cthree();*

*c.show();*

*}*

*}*

**1. super with Methods**

In the following program, subclass overrides the eat() method of super class and at the same uses both with "**super**" keyword.

*class Bird{*

*public void eat(){*

*System.out.println("Eats insects.");*

*}*

*}*

*public class Sparrow extends Bird{*

*public void eat(){*

*super.eat();*

*System.out.println("Eats grains.");*

*super.eat();*  ***// again you can call***

*}*

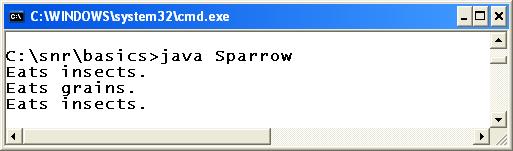
*public static void main(String args[]){*

*Sparrow s1 = new Sparrow();*

*s1.eat();*

*}*

*}*

[](http://way2java.com/wp-content/uploads/2010/11/third33.jpg)

The **eat()** method exists in both **Bird**and **Sparrow**classes. We say, super class method is overridden by subclass. **s1.eat()** calls its own (Sparrow) method. Sparrow uses "**super.eat();**" to call Bird's **eat()** method. The next program illustrates "**static**" with variables.

**Note:**

1. "super" keyword cannot be used from static methods like main().
2. [Static methods cannot be overridden](http://way2java.com/oops-concepts/method-overriding/) (cannot be called with super keyword).

**super with Variables**

You have seen earlier "super" with methods. Let us go for with variables.

*class Packing{*

*int cost = 50;*

*}*

*public class TotalCost extends Packing{*

*int cost = 100;*

*public void estimate(){*

*System.out.println("Cost of articles Rs." + cost);*

*System.out.println("Packing expenses Rs." + super.cost);*

*System.out.println("Total to pay Rs." + (cost + super.cost));*

*}*

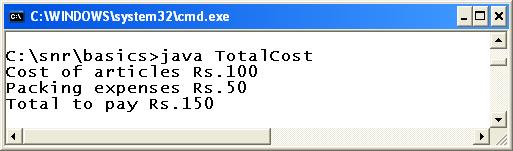
*public static void main(String args[]){*

*TotalCost tc1 = new TotalCost();*

*tc1.estimate();*

*}*

*}*

*[](http://way2java.com/wp-content/uploads/2010/11/third37.jpg)*

In the above code, **cost**variable of **Packing**class is overridden by **TotalCost**. In this case, the subclass object prefers to call its own variable. **super.cost** calls super class **cost**variable.**super.cost**cannot be called from static methods like main() method.

**"super" and "this"**

In java, "super" keyword is used to call super methods and variables (when overridden only, else, not necessary to use) and "[this](http://way2java.com/oops-concepts/using-this-keyword/)" keyword is used to refer the current object.

**Super at constructor level**

The super keyword can also be used to invoke or call the parent class constructor. Constructor are calling from bottom to top and executing from top to bottom.

To establish the connection between base class constructor and derived class constructors JVM provides two implicit methods they are:

* ***Super()***
* ***Super(...)***

**super()**

this super() is used to call the parent class 0 parametrized constructor inside the constructors of Child class.

**Super()** It is used for calling super class default constructor from the context of derived class constructors.

### Super keyword used to call base class constructor

## Syntax

***class*** *Employee{*

*Employee(){*

*System.****out****.println("Employee class Constructor");*

*}*

*}*

***class*** *HR* ***extends*** *Employee{*

*HR(){*

***super****(); //will invoke or call parent class constructor*

*System.****out****.println("HR class Constructor");*

*}*

*}*

***class*** *Supercons{*

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

*{*

*HR obj=****new*** *HR();*

*}*

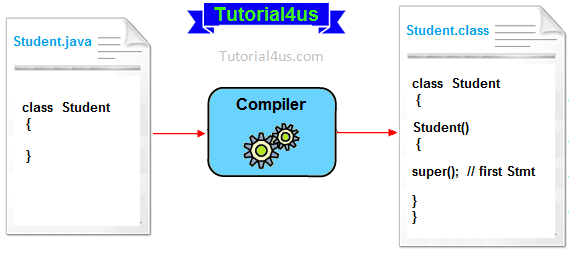
*}*

## Output

Employee class Constructor

HR class Constructor

**Note:**super() is added in each class constructor automatically by compiler.



In constructor, default constructor is provided by compiler automatically but it also adds **super()**before the first statement of constructor.If you are creating your own constructor and you do not have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

**super(...)**

**Super(...)** It is used for calling super class parameterize constructor from the context of derived class constructor.

this super(...) is used to call the parent class parameterized contructor in side the constructors of Child class.

**//wap to demo on super() or super(...)**

*class Parent{*

*Parent(){*

*System.out.println("Parent class 0 parameterized constructor");*

*}*

*Parent(int x){*

*this();*

*System.out.println("Parent class 1 parameterized constructor");*

*}*

*}*

*class Child extends Parent{*

*Child(){*

*this(10,20);*

*System.out.println("Child class 0 parameterized constructor");*

*}*

*Child(int x){*

*this();*

*System.out.println("Child class 1 parameterized constructor");*

*}*

*Child(int x,int y){*

*super(10);*

*System.out.println("Child class 2 parameterized constructor");*

*}*

*}*

*class SuperDemo3{*

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

*Child c = new Child(10);*

*}*

*}*

**Note:**

Inside the constructor if we don't write either this() or super() then compiler will automatically writes super() always. so that every child class constructor will invoke its parent class constructor by default.

**Rules**

1. Calling super() or super(...) or this() or this(...) must be the first statement in the constructor

2. Calling super() or super(...) or this() or this(...) must not be in side the regular method

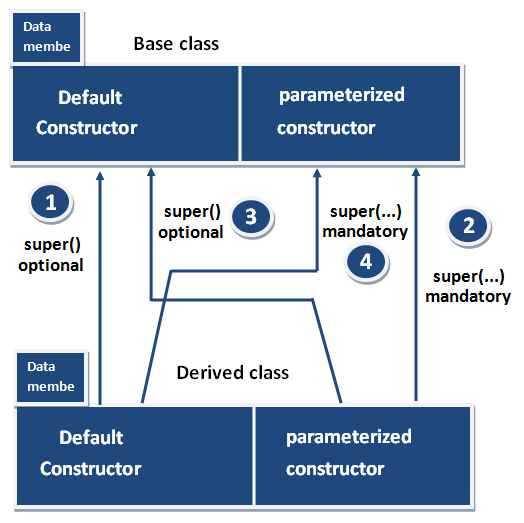
3. We can call super() or super(...) or this() or this(...) at most 1 time

4. Calling super() or super(...) or this() or this(...) must not provide a recursive constructor invocation

**Important rules**

Whenever we are using either super() or super(...) in the derived class constructors the **super**always must be as a first executable statement in the body of derived class constructor otherwise we get a compile time error.

**The following diagram use possibilities of using super() and super(........)**



**Rule 1 and Rule 3**

Whenever the derived class constructor want to call default constructor of base class, in the context of derived class constructors we write super(). Which is optional to write because every base class constructor contains single form of default constructor?

**Rule 2 and Rule 4**

Whenever the derived class constructor wants to call parameterized constructor of base class in the context of derived class constructor we must write super(...). which is mandatory to write because a base class may contain multiple forms of parameterized constructors.

**Ellipsis (...) operator**

Ellipsis (...) operator introduced in java 1.5 version which is used to create method with variable number of arguments

Ellipsis operator internally maintained just like an array.

Ellipsis operator used in only 1 location in entire java that is while writing the parameters of the method.

**// wap to demo on ellipsis (...) operator**

class Demo{

*void sum(int... a){*

*//System.out.println("No.of Parameters="+a.length);*

*int c=0;*

*for(int i=0;i<a.length;i++){*

*c = c + a[i];*

*}*

*System.out.println("sum="+c);*

*}*

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

*Demo d = new Demo();*

*d.sum(10);*

*d.sum(10,20);*

*d.sum(10,20,30);*

*d.sum(10,20,30,450);*

*d.sum(10,13,60,70,70,60,20);*

*d.sum(new int[]{21,34,56});//ellipsis operatator takes array*

*d.sum(new int[]{21,34,56,67,78,78,89});*

*}*

}

**Note:**

In a method/constructor we can also write normal parameters along with parameters which are declared using ellipsis operator but in this case first we have to write normal parameters and followed by parameters which are declared using ellipsis operator.

*void sum(int... ar,int x){ X-invalid*

*}*

*void sum(int x,int... ar){ -valid*

*}*

We can also write our main() method using ellipsis (...) operator

*public static void main(String... args){*

*}*

**What is the diff between (int... args) and (int[] args) ?**

In a method if we use ellipsis (...) operator we can pass any number of arguments but if we use array as parameter then we have to pass only 1 argument that is an array which may contain any number of values.

**Eg:**

*class Demo{*

*void add(int... a){*

*int c=0;*

*for(int i=0;i<a.length;i++){*

*c = c + a[i];*

*}*

*System.out.println("sum="+c);*

*}*

*void mul(int[] a){*

*int c=1;*

*for(int i=0;i<a.length;i++){*

*c = c \* a[i];*

*}*

*System.out.println("mul="+c);*

*}*

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

*Demo d = new Demo();*

*d.add(10);*

*d.add(10,20,30);*

*d.add(new int[]{10,20,30,40});*

*d.mul(new int[]{10,20,30});*

*d.mul(new int[]{10,20,30,40,50,40});*

*//d.mul(10,20,30);* ***-invalid because mul() take only 1 argument***

*}*

*}*

**Note:**

We cannot overload 2 methods which takes array as parameter and second one takes ellipsis operator as parameter.

*void add(int... a){ X-invalid*

*}*

*void add(int[] a){*

*}*

## Important Notes :

You can’t use super and this keywords in a static method and in a static initialization block even though you are referring static members.

|  |
| --- |
| *class SuperClassOne{*  *int i;      //Non-Static member*  *static void methodOne()   {*  *//static method*  *System.out.println("From Super Class");*  *}*  *}*  *class SubClassOne extends SuperClassOne{*  *static{*  *System.out.println(super.i);*  *this.methodTwo();*  ***//Above statements give compile time error***  ***//You can't use super and this keywords inside SIB***  *}*  *static void methodTwo(){*  *super.methodOne();*  *this.methodOne();*  ***//These also give compile time error***  ***//You can't use super and this keywords inside static method***  ***//even though you are accessing static methods***  *}*  *}* |

You should call super() and this() calling statements inside the constructors only and they must be first statement in the constructors.

*class SuperClassOne{*

*void methodOne(){*

*System.out.println("From Super Class");*

*}*

*}*

*class SubClassOne extends SuperClassOne{*

*public SubClassOne(){*

*System.out.println("constructors");*

*super();*

***//compile time error***

***//super() calling statement must be first statement in constructor***

*}*

*void methodTwo(){*

*super();*

*this();*

***//compile time error***

***//you should call super() and this()***

***//calling statements only in constructors.***

*}*

*}*

**Example**

Following program calls super class constructor, field and method from sub class.

|  |
| --- |
| *class SuperClass{*  *int i;    //****Field***  *SuperClass(int j){*  *System.out.println("Super Class Constructor");*  *}*  *void methodOfSuperClass()     //method*  *{*  *System.out.println("From method of super class");*  *}*  *}*  *class SubClass extends SuperClass{*  *SubClass(){*  *super(10);*  *//Calling statement to super class constructor*  *}*  *void methodOfSubClass(){*  *System.out.println(super.i);  //super class field is accessed*  *super.methodOfSuperClass();  // super class method is called*  *System.out.println("From method of sub class");*  *}*  *}* |

super class constructor is called by **super()** calling statement.You can’t use super() calling statement outside the constructor. By default, super() calling statement is the first statement in any constructor. You can go through the constructors rules [here](http://javaconceptoftheday.com/constructors-in-java/).

**When to Use super keyword?**

If you want same implementation as that of super class method in the sub class, but want to add some more extra statements to it, in such cases, super keyword will be very useful. First call the super class method using super keyword and after it add extra statements according to requirements in the sub class method.

*class SuperClass{*

*void methodOfSuperClass(){*

*//Some task*

*}*

*}*

*class SubClass extends SuperClass{*

*void methodOfSubClass() {*

*super.methodOfSuperClass();* ***// super class method is called***

***//add some other extra statements fulfilling the requirements***

*}*

*/****/you can implement same task by overriding super class method also***

*void methodOfSuperClass(){*

***//super class method is overrided.***

*super.methodOfSuperClass();*

***//add some other extra statements fulfilling the requirements***

*}*

*}*

### Program where super is not required

## Example

***class*** *Student{*

***void*** *message(){*

*System.****out****.println("Good Morning Sir");*

*}*

*}*

***class*** *Faculty* ***extends*** *Student{*

***void*** *display(){*

*message();//will invoke or call parent class message() method*

*}*

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

*Student s=****new*** *Student();*

*s.display();*

*}*

*}*

## Output

Good Morning Sir