Super Keyword :

Super keyword is used to represent
1) Call the super class variable.
2) Call the super class methods.
3) Call the super class Constructor.

1) Super with variable

this keyword refers to current class instance variables.
Super keyword refers to base class instance variables.

Super keyword is not required

```
class Parent {
int a=10;
int b=20;
}
class Child extends Parent {
int x=100;
int y=200;

void add(int i,int j) {
System.out.println(i+j);
System.out.println(x+y);
System.out.println(a+b);
}
public static void main(String[] args) {
Child c=new Child();
c.add(1000,2000); }
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
3000
300
30
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Super keyword is required

```
class Parent {
int a=10;
int b=20;
}
class child extends Parent {
int a=100;
int b=200;

void add(int a,int b)
{
System.out.println(a+b);
System.out.println(this.a+this.b);
System.out.println(super.a+super.b);
}
public static void main(String[] args) {
Child t=new Child(); // it calls default cons
t.add(1000,2000);
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
3000
300
30
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

2)super with class methods.

this keyword with method name refers to current class method.
Super keyword with method name refers to base class method.

Super keyword is not required

```
class Parent {
void m1()
{
System.out.println("parent class method"); }

}
class Child extends Parent {
void m2()
{
m1();
System.out.println("child class method");
}
void m3()
{
m1();
System.out.println("child class method");
m2();
}
public static void main(String[] args) {
Child c=new Child();
c.m3();
}

}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
parent class method
child class method
parent class method
child class method
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Super keyword is required

```
class Parent {
void m1()
{
System.out.println("parent class method");
}
}
class Child extends Parent {
void m1()
{
System.out.println("child class method");
}
void m2()
{
this.m1();
System.out.println("child class method");
super.m1();
}
public static void main(String[] args)
{
Child c=new Child();
c.m2();
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
child class method
child class method
parent class method
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

this invokes current class method based on object.

Ex.

```
class Parent {
void m1()
{
System.out.println("m1 - parent class method");
}
void m2()
{
this.m1();
System.out.println("m2 - parent class method");
}
}
class Child extends Parent {
void m1()
{
System.out.println("m1 - child class method");
}

public static void main(String[] args)
{
Child c=new Child();
c.m2();
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
m1 - child class method
m2 - parent class method
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

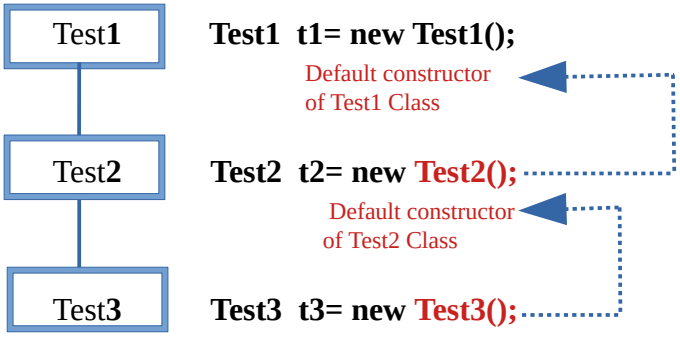
```
class  Parent {
void m1()
{
System.out.println("m1 - parent class method");
}
void m2()
{
this.m1();
System.out.println("m2 - parent class method");
}
}
class Child extends Parent {
void m1()
{
System.out.println("m1 - child class method");
}

public static void main(String[] args)
{
Parent p=new Parent();
p.m2();
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac child.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Child
m1 - parent class method
m2 - parent class method
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

3) Super with Constructor:

1. When we create an object of derived most class , first we get memory space for base class data members and second we get the memory space for intermediate base class and at last we get the memory space for the derived class (Multilevel Inheritance).
2. In whatever order memory space is created in the same order initialization process has to be taken place.
3. We know initialization of data members must be done by constructor.



In the above diagram ,the default constructor of of Test3 is automatically calling default constructor of Test2. And default constructor of Test2 is automatically calling default constructor of Test1. Constructor of Test1 will be executed first , Test2's constructor will be executed second and finally Test3's Constructor will be executed.

In general , in any inheritance applications the way of calling constructors is from bottom to top and they are executing in order of Top to bottom.

In order to establish the communication between the base class and derived class constructor, we have two functions:

i. super ()
ii. super (....)

super() :

1. It is use to call **default constructor** of base class from derived class constructor.
2. The usage of super () in derived class is **Optional**.

super (....)

1. It is use to call **parametrized constructor** of base class constructor from derived class constructor.
2. The usage of super (....) in derived class is **Mandatory.**

Case 1 : Constructor calling

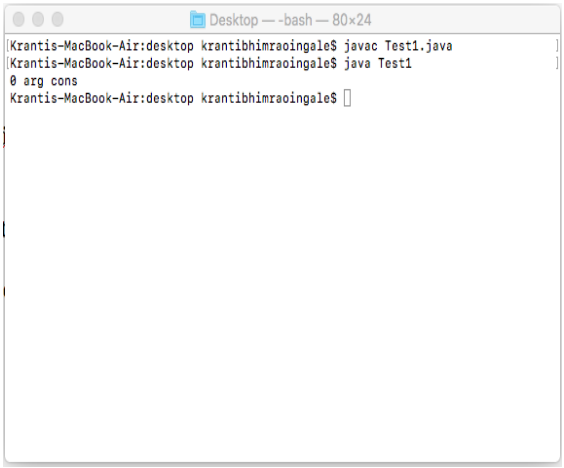
(If we create object of derived class then it automatically calls the constructor of base first then it's own constructor.)

Ex for without super () :

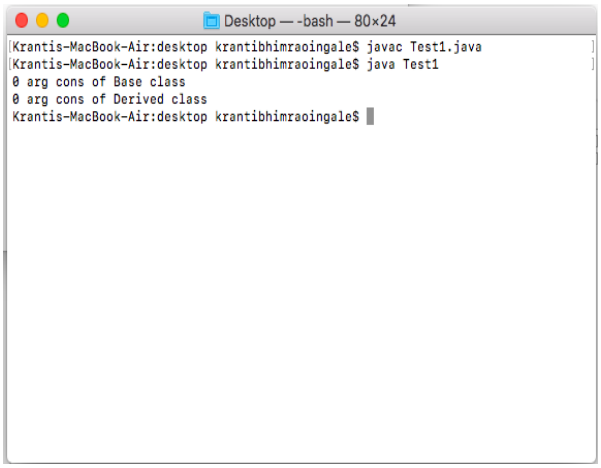
```
class Test
{
Test(){
System.out.println("0 arg cons");
}
}
class Test1 extends Test
{
/* If we don't write any constructor Default
constructor inserted by Compiler.
Test1()
{
}
*/
public static void main(String args[]){
new Test1();
}
}
```

```
class Test
{
Test(){
System.out.println("0 arg cons of Base class");
}
}
class Test1 extends Test
{
Test1(){

System.out.println("0 arg cons of Derived
class");
}
public static void main(String args[]){
new Test1();
}
}
```

Output



Output



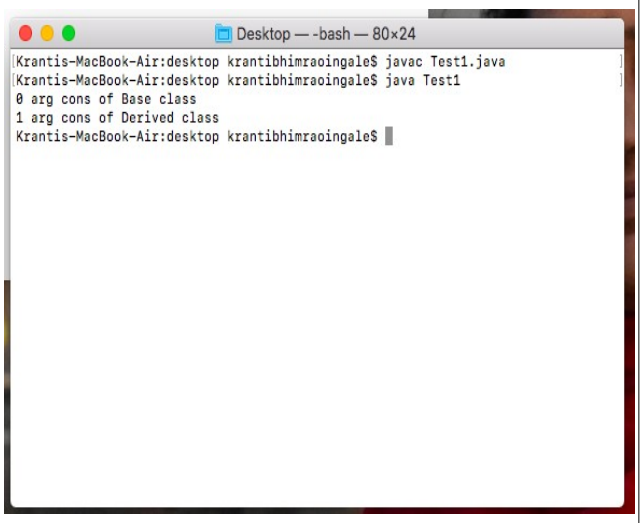
Ex for without super (...)

```
class Test
{
Test(){
System.out.println("0 arg cons of Base class");
}
}
class Test1 extends Test
{
Test1(int b){

System.out.println("1 arg cons of Derived class");
}
public static void main(String args[]){
new Test1(10);
}
}
```

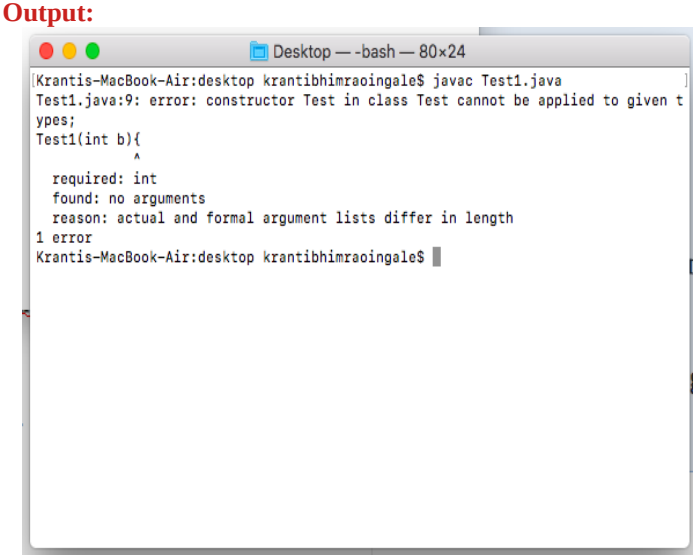
error

```
class Test
{
Test(int a){
System.out.println("0 arg cons of Base class");
}
}
class Test1 extends Test
{
Test1(int b){

System.out.println("1 arg cons of Derived class");
}
public static void main(String args[]){
new Test1(10);
}
}
```

Output:



Output:



by default super() is there in derived class constructor. And super() is searching for default constructor of base class. As we wrote parametrized constructor in base class, we have to use super(...) ,Hence we got C.T Error.

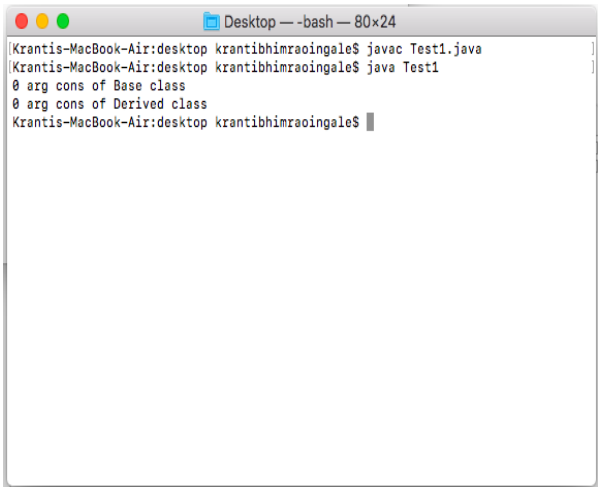
Case 2: The usage of super () in derived class is Optional. We will get the same output

Ex.

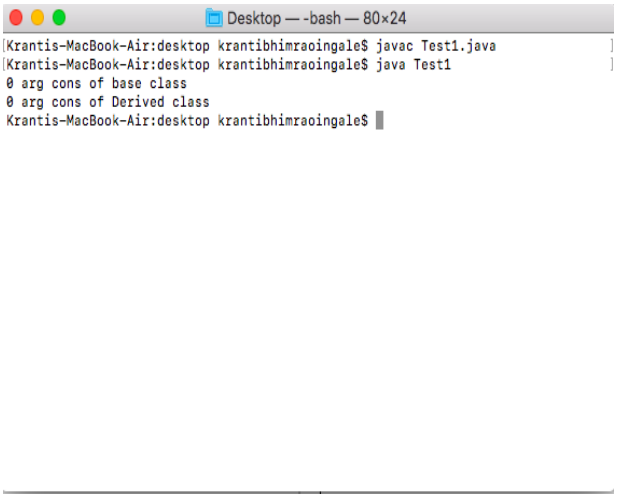
```
//Without super()
class Test
{
Test(){
System.out.println("0 arg cons of Base
class");
}
}
class Test1 extends Test
{
Test1(){
// super() is by default
System.out.println("0 arg cons of Derived
class");
}
public static void main(String args[]){
new Test1();
}
}
```

```
//With super() : Optional to write super()
class Test
{
Test(){
System.out.println("0 arg cons of base class ");
}
}
class Test1 extends Test
{
Test1(){
super();
System.out.println("0 arg cons of Derived class");
}
public static void main(String args[]){
new Test1();
}
}
```

Output:



Output:



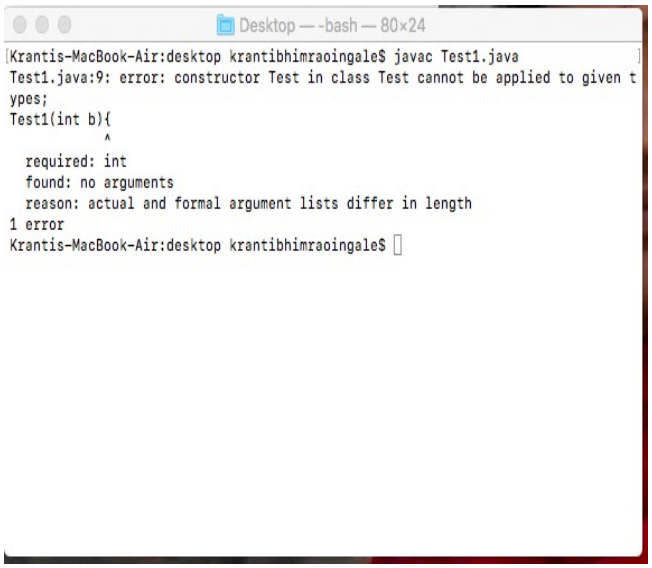
Case 3: The usage of super (..) in derived class is Mandatory. Mandatory to write super(..) otherwise we will get Compile time error.

Ex .

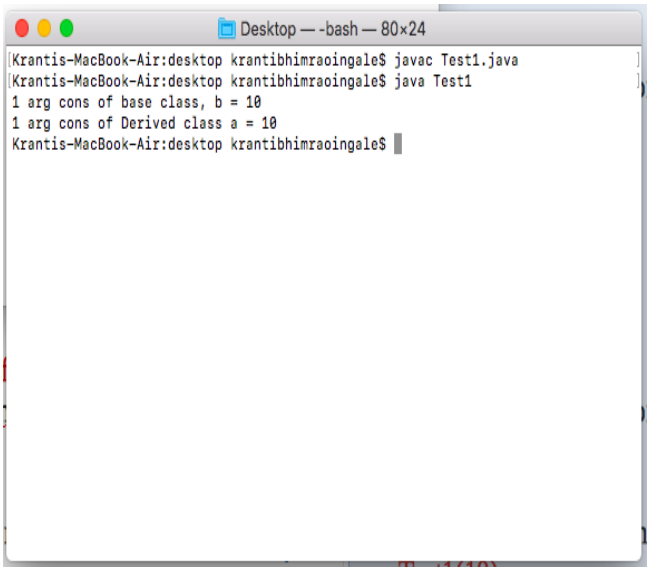
```
//without  super(...)
class Test
{
Test(int a){
System.out.println("1 arg cons of Base
class");
}
}
class Test1 extends Test
 {
Test1(int b){
 // super() is by default
 System.out.println("1 arg cons of Derived
 class");
 }
 public static void main(String args[]){
new Test1(10);
}
}
```

```
//with super(..) : Mandatory
class Test
{
Test(int b){
System.out.println("1 arg cons of base class, b =
"+b);
}
}
class Test1 extends Test
{
Test1(int a){
super(a);
System.out.println("1 arg cons of Derived class a
= "+a);
}
public static void main(String args[]){
new Test1(10);
}
}
```

Output:



Output:



Case 4: whenever we use super() or super(...) in the derived class constructors ,then it must be the first statement otherwise we will get compile time error.

Ex.

```
//Default constructor
class Test
{
Test(){
System.out.println("0 arg cons of base class");
}
}
class Test1 extends Test
{
Test1(){
System.out.println("0 arg cons of
Derived class");
super();
}
public static void main(String args[]){
new Test1();
}
}
```

```
//Parametrized constructor
class Test
{
Test(int b){
System.out.println("1 arg cons of base class, b =
"+b);
}
}
class Test1 extends Test
{
Test1(int a){
System.out.println("1 arg cons of Derived class
a = "+a);
super(a);
}
public static void main(String args[]){
new Test1(10);
}
}
```

Output

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Test1.java
Test1.java:11: error: call to super must be first statement in constructor
super();
     ^
1 error
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Test1.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Test1
1 arg cons of base class, b = 10
1 arg cons of Derived class a = 10
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

Case 5: Calling base class constructors with different ways.

Constructor Overloading

```
class Test
{
Test(){
System.out.println("0 arg cons of base class");
}
Test(int a)
{
System.out.println("1 arg cons of base
class");
}
}
class Test1 extends Test
{
Test1(){
super();
System.out.println("0 arg cons of Derived
class");
}
public static void main(String args[]){
new Test1();
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Test1.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Test1
0 arg cons of base class
0 arg cons of Derived class
Krantis-MacBook-Air:desktop krantibhimraoingale$
```

```
class Test
{
Test(){
System.out.println("0 arg cons of base class");
}
Test(int a)
{
System.out.println("1 arg cons of base class");
}
}
class Test1 extends Test
{
Test1(){
super(10);
System.out.println("1 arg cons of Derived
class");
}
public static void main(String args[]){
new Test1();
}
}
```

Output:

```
Desktop — -bash — 80×24
Krantis-MacBook-Air:desktop krantibhimraoingale$ javac Test1.java
Krantis-MacBook-Air:desktop krantibhimraoingale$ java Test1
1 arg cons of base class
1 arg cons of Derived class
Krantis-MacBook-Air:desktop krantibhimraoingale$
```