

super keyword in java :

* In order to access the member of super class from sub class OR to access the super class memory we should use super keyword.

* super keyword always refers to its immediate super class because java does not support multiple Inheritance.

Example :

```
-----
class A
{
    //Here super keyword will refer to Object class member
}
class B extends A
{
    //Here super keyword will refer to A class member
}
class C extends B
{
    //Here super keyword will refer to B class member
}
```

* Just like this keyword we cannot use super keyword from the static area, It must be used from the non static area only.

We can use super keyword by using the following ways :

- 1) To call the super class variable (Variable Hiding)
- 2) To call the super class method (Method Overriding)
- 3) To call the super class Constructor (Constructor Chaining)

To call super class variable :

Whenever super class variable name and sub class variable name both are same then it is called variable Hiding, Here sub class variable hides super class variable.

In order to access super class variable i.e super class memory, we should use super keyword as shown in the program.

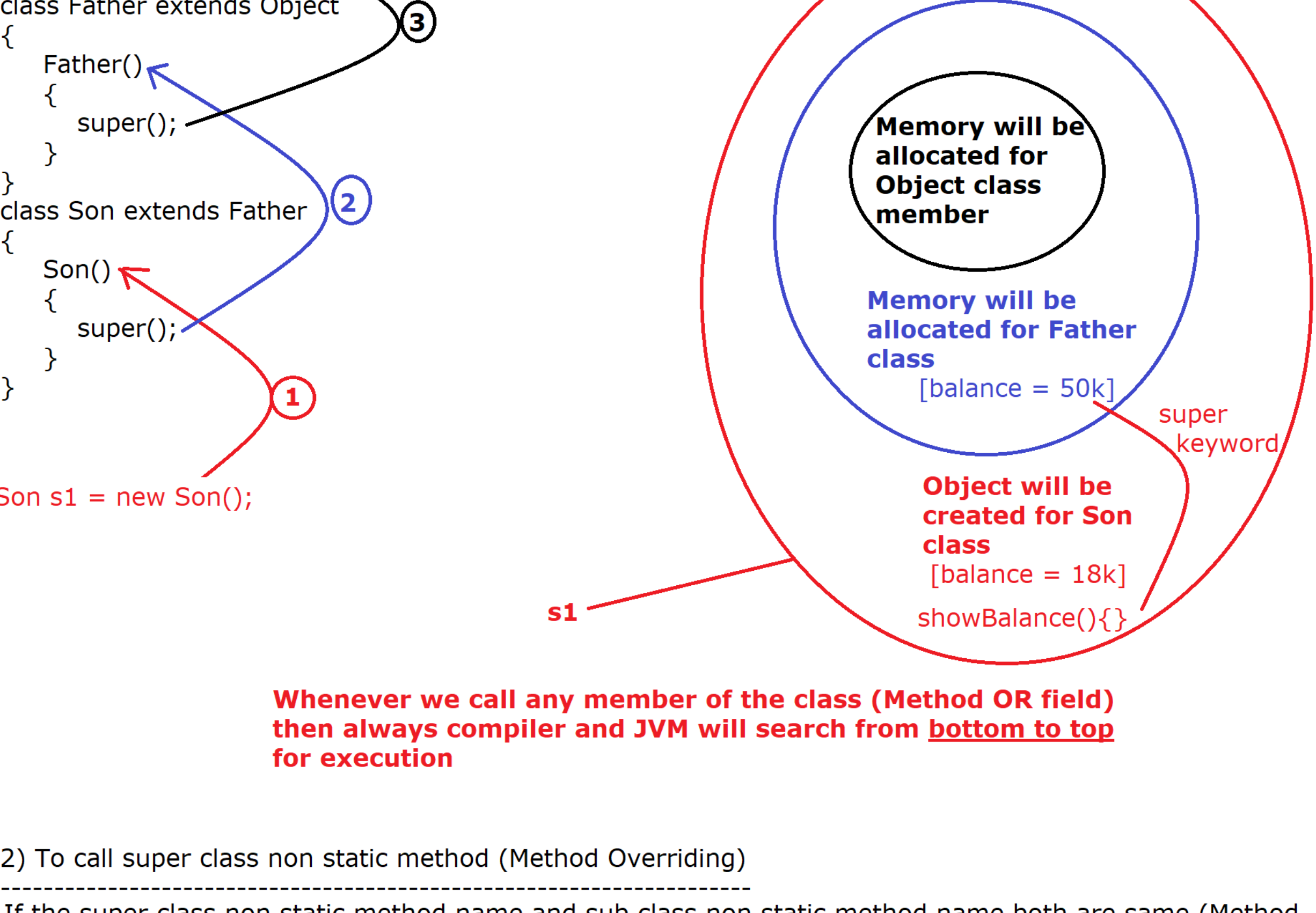
```
package com.ravi.variable_hiding;

class Father
{
    protected double balance = 50000;
}
class Son extends Father
{
    protected double balance = 18000; //Variable Hiding

    public void showBalance()
    {
        System.out.println("Father Balance is :"+super.balance);
        System.out.println("Son balance is :"+this.balance);
    }
}

public class VariableHiding
{
    public static void main(String[] args)
    {
        Son s1 = new Son();
        s1.showBalance();
    }
}
```

Diagram for the above program to find the behavior of compiler and JVM to call the member of the class



2) To call super class non static method (Method Overriding)

If the super class non static method name and sub class non static method name both are same (Method Overriding) and if we create an object for sub class then sub class method will be executed (bottom to top), if we want to call super class method from sub class method body then we we should use super keyword as shown in the program.

```
package com.ravi.method_overriding;

class Alpha
{
    public void display()
    {
        System.out.println("Alpha class Display");
    }
}
class Beta extends Alpha
{
    public void display()
    {
        System.out.println("Beta class Display");
        super.display();
    }
}

public class MethodOverridingDemo
{
    public static void main(String[] args)
    {
        Beta b1 = new Beta();
        b1.display();
    }
}
```

package com.ravi.method_overriding;

```
class A
{
    @Override
    public String toString()
    {
        return "A class";
    }
}
class B extends A
{
    @Override
    public String toString()
    {
        return "B class";
    }
}
class C extends B
{
    @Override
    public String toString()
    {
        return "C class";
    }
}

public class OverridingDemo
{
    public static void main(String[] args)
    {
        C c1 = new C();
        System.out.println(c1.toString());
    }
}
```

3) To call the super class Constructor (Constructor Chaining)

* Whenever we write a class and if we don't write any type of constructor then javac will automatically provide a default constructor in the class.

* The first line if any constructor is reserved for either super() or this() that means the first line of any constructor is meant for calling either the constructor of super class (super()) OR the constructor of current class (this())

* When we don't write super() OR this() explicitly then javac will automatically add super() to the first line of constructor, **javac will only add super()**

```
package com.ravi.constructor_chaining;

class Alpha
{
    public Alpha()
    {
        System.out.println("Alpha class no argument constructor");
    }
}
class Beta extends Alpha
{
    public Beta()
    {
        System.out.println("Beta class no argument constructor");
    }
}
class Gamma extends Beta
{
    public Gamma()
    {
        System.out.println("Gamma class no argument constructor");
    }
}

public class ConstructorChainingDemo {

    public static void main(String[] args)
    {
        new Gamma();
    }
}
```

//Program on Hierarchical Inheritance

```
package com.ravi.constructor_chaining;

class Employee
{
    protected double salary;
    public Employee(double salary)
    {
        super();
        this.salary = salary;
    }
}
class Developer extends Employee
{
    public Developer(double salary)
    {
        super(salary);
    }

    @Override
    public String toString()
    {
        return "Developer [salary=" + salary + "]";
    }
}

class Designer extends Employee
{
    public Designer(double salary)
    {
        super(salary);
    }

    @Override
    public String toString()
    {
        return "Designer [salary=" + salary + "]";
    }
}

public class ConstructorChainingDemo2 {

    public static void main(String[] args)
    {
        Developer dev = new Developer(60000);
        System.out.println(dev);

        Designer des = new Designer(25000);
        System.out.println(des);
    }
}
```

//Program on single level Inheritance

```
package com.ravi.constructor_chaining;

class TempEmp
{
    protected int eid;
    protected String name;
    protected double salary;

    public TempEmp(int eid, String name, double salary)
    {
        super();
        this.eid = eid;
        this.name = name;
        this.salary = salary;
    }
}

class PempEmp extends TempEmp
{
    protected String department;
    protected String designation;

    public PempEmp(int eid, String name, double salary, String department, String designation)
    {
        super(eid, name, salary);
        this.department = department;
        this.designation = designation;
    }

    @Override
    public String toString() {
        return "PempEmp [eid=" + eid + ", name=" + name + ", salary=" + salary + ", department=" + department + ", designation=" + designation + "]";
    }
}

public class SingleLevel
{
    public static void main(String[] args)
    {
        PempEmp p = new PempEmp(1, "Scott", 90000, "IT", "Developer");
        System.out.println(p);
    }
}
```