Inheritance

Table Of Content

Introduction

- Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which **one class is allowed to inherit the features(fields and methods) of another class**.
- Inheritance is a process of a class acquiring the properties(variables) and behavior (methods) from the other class.
- Inheritance can be achieved using **'extends'** keywords.

```
ADVANTAGE OF INHERITANCE:
-------------------------
1. CODE REUSABLITY.
2. less time for developing the s/w.
```

Important Notes

▼ **1. In order to relate two classes we can use extends keywords.**

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**@Note-1
 * In order to relate two classes we can use extends keyword
 * @Note-2
 * Inheritance promotes  IS- A  relationship
 */

class visitor //parent class or super class
{
    String Visitor_Name= "Nandan";
    int Visitor_phone=123456789;
}

class hospital extends visitor //child class or sub class
{
```

```
    void doctor()
    {
        System.out.println("Name: "+Visitor_Name);
        System.out.println("Phone: "+Visitor_phone);
        System.out.println("Tell Me How Can I Help You...!");

    }
}


public class Note_1_2
{
    public static void main(String[] args)
    {
         hospital hospital = new hospital();
         hospital.doctor();
    }
}
//output
Name: Nandan
Phone: 123456789
Tell Me How Can I Help You...!
```

▼ 2. Inheritance promotes IS- A relationship.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**@Note-1
 * In order to relate two classes we can use extends keyword
 * @Note-2
 * Inheritance promotes  IS- A  relationship
 */

class visitor //parent class or super class
{
    String Visitor_Name= "Nandan";
    int Visitor_phone=123456789;
}

class hospital extends visitor //child class or sub class
{
    void doctor()
    {
        System.out.println("Name: "+Visitor_Name);
        System.out.println("Phone: "+Visitor_phone);
        System.out.println("Tell Me How Can I Help You...!");

    }
}


public class Note_1_2
{
    public static void main(String[] args)
    {
         hospital hospital = new hospital();
         hospital.doctor();
    }
}
//output
Name: Nandan
Phone: 123456789
Tell Me How Can I Help You...!
```

▼ 3. Whenever the object of child class is created the memory is allocated for the instance variable of both parent and child class.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-3
 * Whenever the object of child class is created
 * the memory is allocated for the instance variable of both parent and child class.
 */
class a
```

```
{
    String s1 = "I'm in class 'a'"; //parent class 1
}
class b extends a
{
    String s2 = "I'm in class 'b'"; //child class 2
}
public class Note_3
{
    public static void main(String[] args)
    {
        b b_class = new b();
        System.out.println("Accessing the members of parent class 'a': "+b_class.s1);
        System.out.println("Accessing the members of parent class 'b': "+b_class.s2);
    }
}
//output
Accessing the members of parent class 'a': I'm in class 'a'
Accessing the members of parent class 'b': I'm in class 'b'
```

▼ 4. Private members will not participate in the inheritance. This rule is made to promote ENCAPSULATION.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-4
 * Private members will not participate in the inheritance. This rule is made to promote ENCAPSULATION
 */
class class1
{
    private String ns1 = "I'm in class 'a'"; //parent class 1
}
class class2 extends class1
{
    String ns2 = "I'm in class 'b'"; //parent class 2
}

public class Note_4
{
    public static void main(String[] args)
    {
        class2 class2 =  new class2();
        //System.out.println(class2.ns1);//private members can't able to access
        System.out.println(class2.ns2);
    }
}
//output
I'm in class 'b'
```

▼ 5. Child class can call the instance method of parent class directly.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-5
 * Child class can call the instance method of parent class directly.
 */

class parentClass1
{
    void func1()
    {
        System.out.println("Inside parent class 1");
    }
}
class parentClass2 extends  parentClass1
{
    void func2()
    {
        System.out.println("Inside parent class 2");
    }
}

public class Note_5
{
    public static void main(String[] args)
```

```
    {
        //create a instance where ever extends is happening
        parentClass2 p2 = new parentClass2();
        System.out.println("Calling members(functions) from child class");
        p2.func1();
        p2.func2();
    }
}
//output
Calling members(functions) from child class
Inside parent class 1
Inside parent class 2
```

▼ 6. The child class can call the inherited static methods directly.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-6
 * The child class can call the inherited static methods directly.
 */
class parentClass_1
{
    static void func_1()
    {
        System.out.println("Inside static method/function of 'parentClass_1' ");
    }
}
class childClass_1 extends  parentClass_1
{
    void func_11()
    {
        System.out.println("Inside user defined function of 'child class'");
    }
}

public class Note_6
{
    public static void main(String[] args)
    {
        childClass_1 p11 = new childClass_1();
        parentClass_1.func_1(); //calling static method/function of parent class using class name
        p11.func_1(); //calling user defined method/function of parent class using reference variable

        System.out.println();
        p11.func_11();
    }
}
//output
Inside static method/function of 'parentClass_1'
Inside static method/function of 'parentClass_1'

Inside user defined function of 'child class'
```

▼ 7. Unlike the private members constructors are also doesn't participate in the inheritance.

- for 'default constructor'

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-7
 * Unlike the private members constructors are also doesn't participate in the inheritance.
 */

class parentClass22
{
    int i, j; //instance variable declaration
    parentClass22() //constrictor
    {
        i=111; //instance variable initialization
        j=222;
    }
}

class childClass22 extends parentClass22
```

```
{
    childClass22()
    {
        super(); // which calls the constructor of parent class
    }

    void display()
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}

public class Note_7
{
    public static void main(String[] args)
    {
        childClass22 c2 = new childClass22();
        c2.display();
    }
}
//output
'i' is: 111
'j' is: 222
```

- for 'parameterized constructor'

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

class parentClass33
{
    int i,j;
    parentClass33(int k)
    {
        i=111;
        j=222;
        System.out.println("k is: "+k);
    }
}

class childClass33 extends parentClass33
{
    childClass33()
    {
        //pass the parameter
        super(333);
    }

    void display()
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}


public class Note_7_parameterizedConstructor
{
    public static void main(String[] args)
    {
        childClass33 c3 = new childClass33();
        c3.display();
    }
}
//output
k is: 333
i is: 111
j is: 222
```

▼ 8. Even though the parameterized constructor of a sub-class is called yet, the default constructor in the super class is executed first then the parameterized constructor of the sub-class is executed.

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
```

```
 * @Note-8
 * Even though the parameterized constructor of a sub-class is called yet,
 * the default constructor in the super class is executed first
 * then the parameterized constructor of the sub-class is executed.
 */
class parentClass44
{
    parentClass44()
    {
        //calls first
        System.out.println("This is default constructor of parent class");
    }
}

class childClass44 extends parentClass44
{
    childClass44()
    {
        super();
    }
    childClass44(int i, int j)
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}


public class Note_8
{
    public static void main(String[] args)
    {
        childClass44 c4 = new childClass44(111,222);
    }
}
//output
This is default constructor of parent class
i is: 111
j is: 222
```

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

/**
 * @Note-8
 * Even though the parameterized constructor of a sub-class is called yet,
 * the default constructor in the super class is executed first
 * then the parameterized constructor of the sub-class is executed.
 */
class parentClass55
{
    int i,j;
    parentClass55()
    {
        //calls first
        System.out.println("This is default constructor of parent class");
    }
}

class childClass55 extends parentClass55
{

    childClass55(int i, int j)
    {
        super();//to call default constructor of parentClass_3

        // to point the variable we use this (solves shadow problem)
        this.i=i;
        this.j=j;
    }

    void display()
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}
```

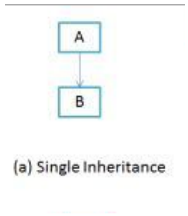
```
public class Note_8_Version2
{
    public static void main(String[] args)
    {
        childClass55 c5 = new childClass55(111,222);//since it is a parameterized constructor in 'childClass55'. So we
 need to pass value
        c5.display();
    }
}
//output
This is default constructor of parent class
i is: 111
j is: 222
```

▼ 9. If the parameterized constructor of the super class has to be executed then parameter super should be used in the sub-class

```
package Object_Oriented_Concepts.inheritance.Points_To_Remember;

class parentClass66
{
    parentClass66()
    {
        //calls first
        System.out.println("This is default constructor of parent class");
    }

    parentClass66(int i, int j)
    {
        this(); //to call the default constructor of same class
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}

class childClass66 extends parentClass66
{
    int k,l;
    childClass66(int k,int l)
    {
        super(111,222);//since it is parameterized constructor in parent class 'parentClass66'
        this.k = k;
        this.l = l;
        //        System.out.println("k is: "+k);
        //        System.out.println("l is: "+l);
    }
    void display()
    {
        System.out.println("k is: "+k);
        System.out.println("l is: "+l);
    }
}

public class NOte_9
{
    public static void main(String[] args)
    {
        childClass66 c6 = new childClass66(333,444);//since it is parameterized constructor in child class 'childClass66'
        c6.display();
    }
}
//output
This is default constructor of parent class
i is: 111
j is: 222
k is: 333
l is: 444
```

Types Of Inheritance

1. Single Level Inheritance



(a) Single Inheritance

- A child class **inherits properties and behavior** from single parent class

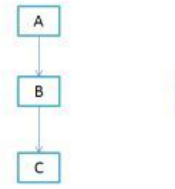
▼ **Single Level Inheritance Example**

```
package Object_Oriented_Concepts.inheritance.Types_Of_Inheritance;

class parentClass
{
    String s;
    void parentMethod()
    {
        s = "From Parent/Super Class";
        System.out.println(s);
        System.out.println("Ready To Inherit Properties(variable) and Behavior(functions) to 'Child Class'");
    }

}
class childClass extends parentClass
{
    String s;
    void childMethod()
    {
        s = "From Child Class";
        System.out.println(s);
        System.out.println("Inherited Properties(variable) and Behavior(functions) from parent class");
    }
}

public class singleLevel_Inheritance
{
    public static void main(String[] args)
    {
        System.out.println("Every Program starts to executes from 'Main Method'");
        System.out.println("Execution starts");
        System.out.println("-----------------------------------");
        childClass cc = new childClass();
        cc.childMethod();
        System.out.println("-----------------------------------");
        System.out.println("Execution ends");
    }
}
//output
Every Program starts to executes from 'Main Method'
Execution starts
-----------------------------------
From Child Class
Inherited Properties(variable) and Behavior(functions) from parent class
-----------------------------------
Execution ends
```

```
package _1Java_Codes_From_Basics._15inheritanceInJava._1TypesOfInheritance;

class parentClass
{
    String s;
    void parentMethod()
    {
        s = "From Parent/Super Class";
        System.out.println(s);
        System.out.println("Ready To Inherit Properties(variable) and Behavior(functions) to 'Child Class'");
    }

}
```

```
class childClass extends parentClass
{
    String s;
    void childMethod()
    {
        s = "From Child Class";
        System.out.println(s);
        System.out.println("Inherited Properties(variable) and Behavior(functions) from parent class");
    }
}


public class _1singleLevelInheritanceEx1
{
    public static void main(String[] args)
    {
        System.out.println("Execution starts");
        System.out.println("-----------------------------------");
        inherted("Every Program starts to executes from 'Main Method'");
        System.out.println("-----------------------------------");
        System.out.println("Execution ends");
    }
    static void inherted(String s)
    {
        System.out.println(s);
        childClass cc = new childClass();
        cc.parentMethod();
        cc.childMethod();
    }
}
//output
Execution starts
-----------------------------------
Every Program starts to executes from 'Main Method'
From Parent/Super Class
Ready To Inherit Properties(variable) and Behavior(functions) to 'Child Class'
From Child Class
Inherited Properties(variable) and Behavior(functions) from parent class
-----------------------------------
Execution ends
```

2. Multi Level Inheritance



(d) Multilevel Inheritance

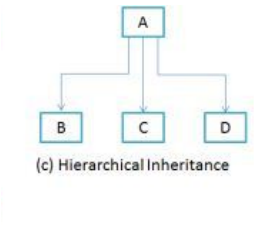
- a child class **inherits properties and behavior** from multiple parent class

▼ Multi Level Inheritance Example

```
package Object_Oriented_Concepts.inheritance.Types_Of_Inheritance;

class A
{
    void func1()
    {
        System.out.println("Parent 1");
        System.out.println("inside the fun1 method");
    }
}
class B extends A
{
    void func2()
    {
        System.out.println("Parent 2");
        System.out.println("inside the fun2 method");
```

```
    }
}
class C extends B
{
    void func3()
    {
        System.out.println("Child");
        System.out.println("inside the fun3 method");
    }
}


public class multiLevel_Inheritance
{
    public static void main(String[] args)
    {
        C c = new C();
        c.func1();
        c.func2();
        c.func3();
    }
}
//output
Execution Started

Parent 1
inside the fun1 method
Parent 2
inside the fun2 method
Child
inside the fun3 method

Execution Ended
```

2. Hierarchical Inheritance



(c) Hierarchical Inheritance

- When more than one child classes inherit **inherits properties and behavior from** a same parent class.

▼ Hierarchical Inheritance Example

```
package Object_Oriented_Concepts.inheritance.Types_Of_Inheritance;

class sameParent
{
    public void methodA()
    {
        System.out.println("method of Parent Class");
    }
}
class child1 extends sameParent
{
    public void methodB()
    {
        System.out.println("method of Child Class-1 of same parent class");
    }
}
class child2 extends sameParent
{
    public void methodC()
    {
        System.out.println("method of Child Class-2 of same parent class");
    }
}
class child3 extends sameParent
{
```

```
    public void methodD()
    {
        System.out.println("method of Child Class-3 of same parent class");
    }
}

public class hierarchical_Inheritance
{
    public static void main(String[] args)
    {
        //here 3 different inherits. So we created separate  object for each inherit
        child1 c1 = new child1();
        child2 c2 = new child2();
        child3 c3 = new child3();
        c1.methodA();
        c1.methodB();
        c2.methodC();
        c3.methodD();
    }
}
//output
method of Parent Class
method of Child Class-1 of same parent class
method of Child Class-2 of same parent class
method of Child Class-3 of same parent class
```

Important keywords & methods required for inheritance

- **this keyword** : it will always points the currently executing object.
- **super keyword** : it is used to access the instance variable of parent class.
- **this()** : it is used to call the constructor within the same class.
- **super()** : it is used to call the constructor of the parent class

super keyword

- During the inheritance if the parent class and child class **having a same variable name** that time it causes name clash & more priority is given to child class instance variable.

  So In order to **access the instance variable of parent class we will use super keyword.**

▼ **Super Keyword Example**

```
package Object_Oriented_Concepts.inheritance._3_Important_Keywords_And_Methods;

/**
 * @Note
 * super keyword   : it is used to access the instance variable  of parent class.
 */
class parent
{
    String name="Nandan";
    int age = 22;
    String email = "gnnandan7@gmail.com";
}

class child extends parent
{
    void access()
    {
        System.out.println("Super keyword is used to access the variables of parent class");
        System.out.println("My name is: "+super.name);
        System.out.println("My age is: "+super.age);
        System.out.println("My email is: "+super.email);

    }
}


public class super_keyword
{
```

```
    public static void main(String[] args)
    {
        child child = new child();
        child.access();
    }
}
//output
Super keyword is used to access the variables of parent class
My name is: Nandan
My age is: 22
My email is: gnnandan7@gmail.com
```

```
package _1Java_Codes_From_Basics._15inheritanceInJava._2importantMethodsAndKeywords;

/*NOTE: During the inheritance if the parent class and child class having a same variable name
 that time it causes name clash & more priority is given to child class instance variable.
 So In order to access the instance variable  of parent class we will use super keyword.*/

class superEx
{
    String s = "ONE";
}

class childEx extends superEx
{
    String s = "TWO";
    void func()
    {
        System.out.println("More Priority Is Given To Child Class Instance Variable, So the value of String 's' is: "+
s);
        System.out.println("Inorder to access the Instance Variable of Parent Class we need to use 'super keyword' like
this: "+super.s);
    }

}
public class _1superKeywordEx1
{
    public static void main(String[] args)
    {
        //create a object where inheritance is actually happening
        childEx cs = new childEx();
        cs.func();
    }
}
//output
More Priority Is Given To Child Class Instance Variable, So the value of String 's' is: TWO
Inorder to access the Instance Variable of Parent Class we need to use 'super keyword' like this: ONE
```

this keyword

- If the Instance variable and local variable are having same variable name that time it causes name clash(Shadow problem) & more priority is given to local variable.
  So In order to access the instance variable we will use **this keyword**.

▼ **this Keyword Example**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._2importantMethodsAndKeywords;

/*NOTE: if the Instance variable and local variable are having same variable name
 that time it causes name clash(Shadow problem) & more priority is given to local variable.
 So In order to access the instance variable we will use this keyword.*/

class A
{
    String s = "ONE";
}

class B extends A
{
    String s = "TWO";
    void func()
    {
        String s = "THREE";
        System.out.println("If the local variable and instance variable are having same name then more priority is give
```

```
n to local variable, So the value of String 's' is: "+s);
        System.out.println("Inorder to access the Instance Variable. We need to use 'this keyword' like this: "+this.
s);
        System.out.println("To access the variable of parent class we use: "+super.s);
    }

}

public class _2thisKeywordEx2
{
    public static void main(String[] args)
    {
        B b = new B();
        b.func();
    }
}
//output
More Priority Is Given To Child Class Instance Variable, So the value of String 's' is: TWO
Inorder to access the Instance Variable of Parent Class we need to use 'super keyword' like this: ONE
```

```
package Object_Oriented_Concepts.inheritance._3_Important_Keywords_And_Methods;

/**
 * @Note
 * this keyword: it will always point the currently executing object.
 */

class parentClass22 //parent class
{
    int i, j; //instance variable declaration
    parentClass22() //constrictor
    {
        System.out.println("This is default constructor");
    }
}

class childClass22 extends parentClass22 //child class
{
    childClass22()
    {
        super(); // which calls the constructor of parent class
    }

    void display(int i, int j)
    {
        //this points the executing object.
        this.i = i;
        this.j = j;
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}


public class this_keyword
{
    public static void main(String[] args)
    {
        childClass22 c2 = new childClass22();
        c2.display(111,222);
    }
}
//output
This is default constructor
i is: 111
j is: 222
```

NOTE: super() and this() can not be placed in the same constructor.

super ()method

- It is used to **call the parent class constructor**.
- we should pass the same **data type value inside super()**, if the parent class constructor is taking any parameter

- **Important 'super()' must be the first statement in child class constructor**

  So In order to **access the instance variable of parent class we will use** **super keyword.**

▼ **Super() Default Constructor Method Example**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._2importantMethodsAndKeywords;

class parent
{
    parent(int i)
    {
        System.out.println(i);
        System.out.println("Inside 'Parent Class' Constructor ");
    }
}

class child extends  parent
{
    child()
    {
        super(10);//super() method must be first statement
        System.out.println("Inside 'Child Class' Constructor");
        System.out.println("Using 'Child Class' Constructor with 'super() method' we are calling 'Parent Class Construc
tor'");
    }
}


public class _3thisMethodEx3
{
    public static void main(String[] args)
    {
        child ch = new child();
    }
}
//output
10
Inside 'Parent Class' Constructor
Inside 'Child Class' Constructor
Using 'Child Class' Constructor with 'super() method' we are calling 'Parent Class Constructor'
```

```
package Object_Oriented_Concepts.inheritance._3_Important_Keywords_And_Methods;

/**
 * @Note
 * super(): it is used to call the constructor of the parent class
 */

class parentClass11 //parent class
{
    int i, j; //instance variable declaration
    parentClass11() //constrictor
    {
        i=111; //instance variable initialization
        j=222;
    }
}

class childClass11 extends parentClass11 //child class
{
    childClass11()
    {
        super(); // which calls the constructor of parent class
    }

    void display()
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}


public class super_method_defaultConstructor
{
    public static void main(String[] args)
```

```
    {
        childClass11 childClass11 = new childClass11();
        childClass11.display();
    }
}
//output
i is: 111
j is: 222
```

▼ **Super() Parameterized Constructor Method Example**

```
package Object_Oriented_Concepts.inheritance._3_Important_Keywords_And_Methods;

class parentClass111
{
    int i,j;
    parentClass111(int k)
    {
        i=111;
        j=222;
        System.out.println("k is: "+k);
    }
}

class childClass111 extends parentClass111
{
    childClass111()
    {
        //pass the parameter
        super(333);
    }

    void display()
    {
        System.out.println("i is: "+i);
        System.out.println("j is: "+j);
    }
}

public class super_method_ParameterizedConstructor
{
    public static void main(String[] args)
    {
        childClass111 c111 = new childClass111();
        c111.display();
    }
}
//output
k is: 333
i is: 111
j is: 222
```

this() method

- It is used to **call the one constructor from another constructor of same class**
- we should pass the same **data type value inside this()**, if the same class constructor is taking any parameter
- **Important 'this()' must be the first statement in same class constructor**

▼ **this() Method Example**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._2importantMethodsAndKeywords;

class aa
{
    aa()
    {
        System.out.println("This is default constructor inside parent class");
    }
    aa(String s)
    {
        this();
        System.out.println("Calling another constructor of same class using 'this() method' inside parent class");
        System.out.println(s);
```

```
    }
}

class bb extends aa
{
    bb()
    {
        super("calling parent class parameterized constructor");
        System.out.println("This is default constructor inside child class");
    }
    bb(String s)
    {
        this();
        System.out.println("Calling another constructor of same class using 'this() method' inside child class");
        System.out.println(s);

    }
}
public class _4thisMethodEx4
{
    public static void main(String[] args)
    {
        bb obj  = new bb("calling child class parameterized constructor");

    }
}
//output
This is default constructor inside parent class
Calling another constructor of same class using 'this() method' inside parent class
calling parent class parameterized constructor
This is default constructor inside child class
Calling another constructor of same class using 'this() method' inside child class
calling child class parameterized constructor
```

NOTE: super() and this() can not be placed in the same constructor.

Execution Steps Of Static, Non Static, & Instance Members In Inheritance

Execution Steps Of Instance Members

```
STEPS FOR EXECUTION OF INTANCE MEMBERS IN IN HERITANCE:
------------------------------------------------------
1. Identify all the instance members of the parent class
2. execute instance variable followed by instance blocks and method and finally execute parent class constrctor
3. Identify all the instance members of the child class
4. execute instance variable followed by instance blocks and method and finally execute child class constrctor
```

▼ **Instance Members Execution Steps**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._4ExecutionProceduresOfStaticAndNonStaticMembersInheritance;


class parentA
{
    String s = "Instance Variable Of Parent Class";
    {
        System.out.println(s);
        System.out.println("---------------------------------------");
        System.out.println("First Instance Block Of Parent Class");
    }
    {
        parentFunc();
        System.out.println("Second Instance Block Of Parent Class");
    }
    parentA()
    {
        System.out.println("User Defined Parent Class Constructor");
        System.out.println("***************************************");
    }
    void parentFunc()
    {
        System.out.println("Parent Class Instance Method");
```

```
    }
}
class childA extends parentA
{
    String ss = "Instance Variable Of Child Class";
    {
        System.out.println(ss);
        System.out.println("---------------------------------------");
        System.out.println("First Instance Block Of Child Class");
    }
    {
        childFunc();
        System.out.println("Second Instance Block Of Child Class");
    }
    childA()
    {
        System.out.println("User Defined Child Class Constructor");
    }
    void childFunc()
    {
        System.out.println("Child Class Instance Method");
    }
}


public class _1instanceMembersInInheritanceEx1
{
    public static void main(String[] args)
    {
        //to access Instance Members Of Parent Class & Child Class Create a Object For Child Class
        childA ca = new childA();
    }
}
//output
Instance Variable Of Parent Class
---------------------------------------
First Instance Block Of Parent Class
Parent Class Instance Method
Second Instance Block Of Parent Class
User Defined Parent Class Constructor
***************************************
Instance Variable Of Child Class
---------------------------------------
First Instance Block Of Child Class
Child Class Instance Method
Second Instance Block Of Child Class
User Defined Child Class Constructor

Process finished with exit code 0
```

Execution Steps Of Static Members

```
STEPS FOR EXECUTION OF STATIC MEMBERS IN INHERITANCE:
-----------------------------------------------------
1. Identify all the static members fo the parent class
2. execute static variable fallowed by static blocks and method.
3. Identify all the static  members fo the child class
4. execute static variable fallowed by static blocks and method.
5. then object is created during object creation constructor will be executed.
```

▼ **Static Members Execution Steps**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._4ExecutionProceduresOfStaticAndNonStaticMembersInheritance;

class parentB
{
    static String bs = "Static Variable Of Parent Class";
    static
    {
        System.out.println(bs);
        System.out.println("---------------------------------------");
        System.out.println("First Static Block Of Parent Class");
    }
    {
        parentFunc1();
```

```
        System.out.println("Second Static Block Of Parent Class");
    }
    parentB()
    {
        System.out.println("User Defined Parent Class Constructor");
        System.out.println("**************************************");
    }
    void parentFunc1()
    {
        System.out.println("Parent Class Static Method");
    }
}
class childB extends parentB
{
    String bss = "Static Variable Of Child Class";
    {
        System.out.println(bss);
        System.out.println("---------------------------------------");
        System.out.println("First Static Block Of Child Class");
    }
    {
        childFunc1();
        System.out.println("Second Static Block Of Child Class");
    }
    childB()
    {
        System.out.println("User Defined Child Class Constructor");
    }
    void childFunc1()
    {
        System.out.println("Child Class Static Method");
    }
}

public class _2staticMembersInInheritanceEx2
{
    public static void main(String[] args)
    {
        childB cb = new childB();
    }
}
//output
Static Variable Of Parent Class
---------------------------------------
First Static Block Of Parent Class
Parent Class Static Method
Second Static Block Of Parent Class
User Defined Parent Class Constructor
**************************************
Static Variable Of Child Class
---------------------------------------
First Static Block Of Child Class
Child Class Static Method
Second Static Block Of Child Class
User Defined Child Class Constructor

Process finished with exit code 0
```

Execution Steps Of Both Static & Instance Members

```
STEPS FOR EXECUTION OF STATIC MEMBERS AND INSTANCE MEMBERS IN INHERITANCE:
-------------------------------------------------------------------------
1. Identify the static members(both parent and child class)
2. execute the sv sb and sm of parent class and execute it in the top-bottom order
3. execute the main method
4. new keyword---> object creation
5. identify the instance members( of both parent and child class)
6. execute the iv ib im and construtor of parent class
7.  execute the iv ib im and construtor of child class
```

▼ **Static & Instance Members Execution Steps**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._4ExecutionProceduresOfStaticAndNonStaticMembersInheritance;
class parentC
{
```

```
    String iV_P = "Instance Variable Of Patent Class";
    static String sV_P = "Static Variable Of Parent Class";
    {
        System.out.println(iV_P);
        System.out.println("Instance Block Of Parent Class");
        instanceFunc_P();
    }
    static
    {
        System.out.println(sV_P);
        System.out.println("Static Block Of Parent Class");
        staticFunc_P();
    }
    parentC()
    {
        System.out.println("Instance Method Of Parent Class Constructor");
    }

    void instanceFunc_P()
    {
        System.out.println("Parent Class Instance Function");
    }
    static void staticFunc_P()
    {
        System.out.println("Parent Class Static Function");
    }
}

class childC extends parentC
{
    String iV_C = "Instance Variable Of Child Class";
    static String sV_C = "Static Variable Of Child Class";
    {
        System.out.println(iV_C);
        System.out.println("Instance Block Of Child Class");
        instanceFunc_C();
    }
    static
    {
        System.out.println(sV_C);
        System.out.println("Static Block Of Child Class");
        staticFunc_C();

    }
    childC()
    {
        System.out.println("Instance Method Of Child Class Constructor");
    }

    void instanceFunc_C()
    {
        System.out.println("Child Class Instance Function");
    }
    static void staticFunc_C()
    {
        System.out.println("Child Class Static Function");
    }
}

public class _3StaticAndInstanceMembersEx3
{
    public static void main(String[] args)
    {
        childC cc = new childC();
    }
}
//output
Static Variable Of Parent Class
Static Block Of Parent Class
Parent Class Static Function
Static Variable Of Child Class
Static Block Of Child Class
Child Class Static Function
Instance Variable Of Patent Class
Instance Block Of Parent Class
Parent Class Instance Function
Instance Method Of Parent Class Constructor
Instance Variable Of Child Class
Instance Block Of Child Class
Child Class Instance Function
Instance Method Of Child Class Constructor
```

```
Process finished with exit code 0
```

Types Of Method With Respect To Inheritance

Inherited Method

- Any method which is **acquired by the child class from the parent class** is called as inherited method.

▼ **Inherited Method**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._3typesOfFunctionsWithRespectToInheritance;

class plane
{
    String name; // instance variable
    void takeOff()
    {
        System.out.println("plane is taking off");
    }
    void fly()
    {
        System.out.println("plane is flying"); // we are inheriting the method fly()
    }
    void land()
    {
        System.out.println("plane is landing");
    }
}

class cargoPlane extends plane
{
    void fly()
    {
        super.name = "Cargo Plane";
        System.out.println(super.name+" Flies at 931 km/h"); // inherited method
    }
    void cargoplane()
    {
        System.out.println(super.name+" Is Used Carry Goods");
    }
}

class passengerPlane extends plane
{
    void fly()// inherited method
    {
        super.name = "Passenger Plane";
        System.out.println(super.name+" Flies At 880 - 926 km/h");
    }
    void passengersPlane()
    {
        System.out.println(super.name+" Is Used Carry Peoples");
    }
}
class fighterJet extends plane
{
    void fly()// inherited method
    {
        super.name = "Fighter Jet";
        System.out.println(super.name+" Flies At 7,200 km/h");
    }
    void fighterPlane()
    {
        System.out.println(super.name+" Is Used Carry Arms");
    }
}

public class _1inheritedMethodEx1
{
    public static void main(String[] args)
    {
        //create a object where we use extends
        cargoPlane type1 = new cargoPlane();
        passengerPlane type2 = new passengerPlane();
        fighterJet type3 = new fighterJet();
```

```
        type1.takeOff();
        type1.fly();
        type1.cargoplane();
        type1.land();

        System.out.println("-------------------------");
        type1.takeOff();
        type2.fly();
        type2.passengersPlane();
        type1.land();

        System.out.println("-------------------------");
        type1.takeOff();
        type3.fly();
        type3.fighterPlane();
        type1.land();
    }
}
//output
plane is taking off
Cargo Plane Flies at 931 km/h
Cargo Plane Is Used Carry Goods
plane is landing
-------------------------
plane is taking off
Passenger Plane Flies At 880 - 926 km/h
Passenger Plane Is Used Carry Peoples
plane is landing
-------------------------
plane is taking off
Fighter Jet Flies At 7,200 km/h
Fighter Jet Is Used Carry Arms
plane is landing
```

Overridden Method

- **any changes performed on inherited method** is called as overridden methods

▼ **Overridden Method**

```
package _1Java_Codes_From_Basics._15inheritanceInJava._3typesOfFunctionsWithRespectToInheritance;
class Parent
{
    void marry()
    {
        System.out.println("marry @ age of 26");               // overriding method happening here in child class
    }
}
class Child extends Parent
{
    void marry()
    {
        System.out.println("marry @ age of 30");                                // overriding method w.r.t child class
    }
    void job()
    {
        System.out.println("Job is Mandatory");
    }
}
public class _2overridingMethodEx2
{
    public static void main(String[] args)
    {
        Child c = new Child();
        c.marry();
        c.job();
    }
}
//output
marry @ age of 30
Job is Mandatory
```

Specialized Method

- methods **which are unique to child class** is called as specialized methods

▼ Specialized Method

```
package _1Java_Codes_From_Basics._15inheritanceInJava._3typesOfFunctionsWithRespectToInheritance;
class parentClass
{
    void marry()
    {
        System.out.println("marry @ age of 26");               // overriding method happening here in child class
    }
}
class childClass extends parentClass
{
    //overridden method
    void marry()
    {
        System.out.println("marry @ age of 30");                              // overriding method w.r.t child class
    }

    //unique method/specialized method in child class
    void job()                                                                //specialized method 1
    {
        System.out.println("Job is Mandatory");
    }
    void decision()                                                          //specialized method 2
    {
        System.out.println("Yes");
    }
}
public class _3specializedMethodEx3
{
    public static void main(String[] args)
    {
        childClass c = new childClass();
        c.marry();
        c.job();
        c.decision();
    }
}
//output
marry @ age of 30
Job is Mandatory
Yes
```