UNIT-II

Inheritance

Inheritance:

It creates new classes from existing classes,so that the new classes will aquire all the fetures of the existing classes is called Inheritance.

Father and Child relationship is called Inheritance.

Definition:

A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

IS-A relationship is also called inheritance.

By using extends keyword we can implement IS-A relation ship

Application Without inheritance:

- code duplication(Redundancy)
- Length of the code is increased.

Example:

```
class A
{ void m1(){}
  void m2(){}
}
class B
{ void m1(){}
  void m2(){}
        void m3(){}
        void m4(){}
}
class C
{ void m1(){}
  void m2(){}
  void m3(){}
  void m4(){}
        void m5(){}
        void m6(){}
}
```

Application With Inheritance/Advantage of Inheritance:

- To reduce the Redundancy
- Reduce the length of the code.

The syntax of inheritance is:

```
class subclass extends superclass
{
}
```

Example:

```
class A              //class which providing  is called Parent class or Super class or base class
{
        void m1(){}
        void m2(){}
}
class B extends A //class which deriving is called Child class or Sub class or derived class
{
        void m3(){}
        void m4(){}
}
class C extends B
{
        void m5(){}
        void m6(){}
}
```

Why super class members are available to sub class?

Because,the sub class object contains a copy of super class object.

Q)What is the advantage of inheritance?

In inheritance,a programmer reuses the super class code without rewriting it,in creation of sub classes.So,developing the classes becomes very easy.Hence,the programmer's productivity is increased.

Note:

- extends is a keyword which is used for inheriting the data members and methods from base class to the derived class and it also improves functionality of derived class.
- Final classes cannot be inherited.
- If the base class contains private data members then that type of data members will not be inherited into derived class.

Whenever we develop any inheritance application, it is always recommended to create an object of bottom most derived class. Since, bottom most derived class contains all the features from its super classes.

• One class can extend only one class at a time. Since, JAVA does not support multiple inheritances.
Whenever we inherit the base class members into derived class, when we creates an object of derived class, JVM always creates the memory space for base class members first and later memory space will be created for derived class members.

Example1:

```
class Parent
{
public void m1()
{
System.out.println("Parent");
}
}
class Child extends Parent
{
public void m2()
{
System.out.println("Child");

}
}

class Test
{
public static void main(String args[])
{
//case 1
Parent p=new Parent();
p.m1();
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example1>javac Test.java
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example1>java Test
Parent
```

Example2:

```
class Parent
{
public void m1()
{
System.out.println("Parent");
}
}
class Child extends Parent
{
public void m2()
{
System.out.println("Child");

}
}

class Test
{
public static void main(String args[])
{
//case 1
Parent p=new Parent();
p.m1();
p.m2();
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example2>javac Test.java
Test.java:24: cannot find symbol
symbol  : method m2()
location: class Parent
p.m2();
 ^
1 error
```

Example 3:

```
class Parent
{
public void m1()
{
System.out.println("Parent");
}
}
class Child extends Parent
{
public void m2()
{
System.out.println("Child");

}
}

class Test
{
public static void main(String args[])
{
//case 2
Child c=new Child();
c.m1();
c.m2();
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example3>javac Test.java
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example3>java Test
Parent
Child
```

Example 4:

```
class Parent
{
public void m1()
{
System.out.println("Parent");
}
}
class Child extends Parent
{
public void m2()
{
System.out.println("Child");

}
}

class Test
{
public static void main(String args[])
{
//case 3
Parent p=new Child();
p.m1();
p.m2();
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example4>javac Test.java
Test.java:24: cannot find symbol
symbol  : method m2()
location: class Parent
p.m2();
 ^
1 error
```

Example 5:

```
class Parent
{
public void m1()
{
System.out.println("Parent");
}
}
class Child extends Parent
{
public void m2()
{
System.out.println("Child");

}
}

class Test
{
public static void main(String args[])
{
//case 4
Child c=new Parent();

}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\Inheritance\Example5>javac Test.java
Test.java:22: incompatible types
found   : Parent
required: Child
Child c=new Parent();
      ^
1 error
```

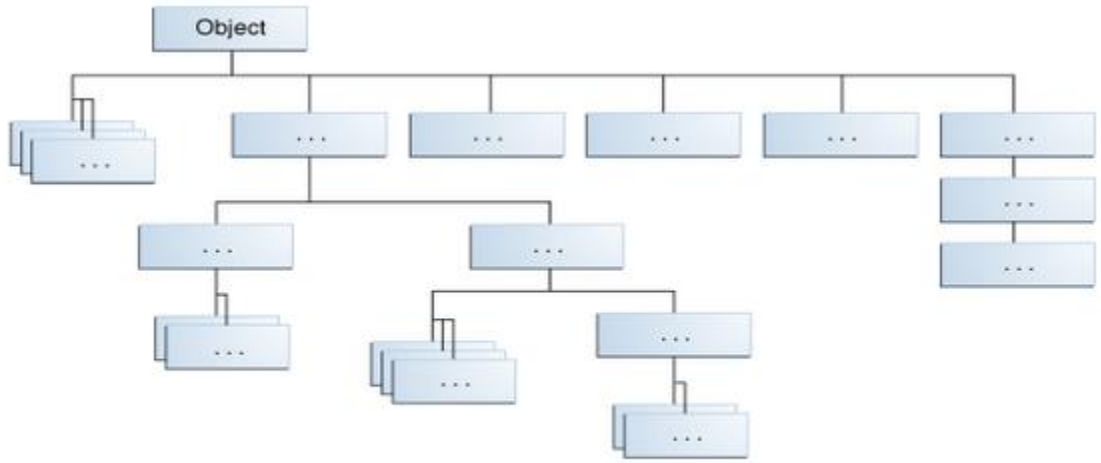
Conclusions:

- what ever methods parent has by default available to the child and hence on the child reference we can call both parent and child class methods.
- what ever methods child has by default not available to the parent and hence,on the parent reference we can not call child specific methods.

- Parent reference can be used to hold child object but using that reference we can't call child specific methods but we can call the methods present in parent class.
- parent reference can be used to hold child object but child reference can't be used to hold parent object.

Total Java API is implemented is based on inheritance concept.

The most common methods which are applicable for any java object are defined in Object class.hence every class in java is the child class of Object either directly or indirectly.so that Object class methods by default available to every java class with out re writing due to this Object class acts as root for all java classes.



All Classes in the Java Platform are Descendants of Object

Types of Inheritance:

There are 5 Types of Inheritance.

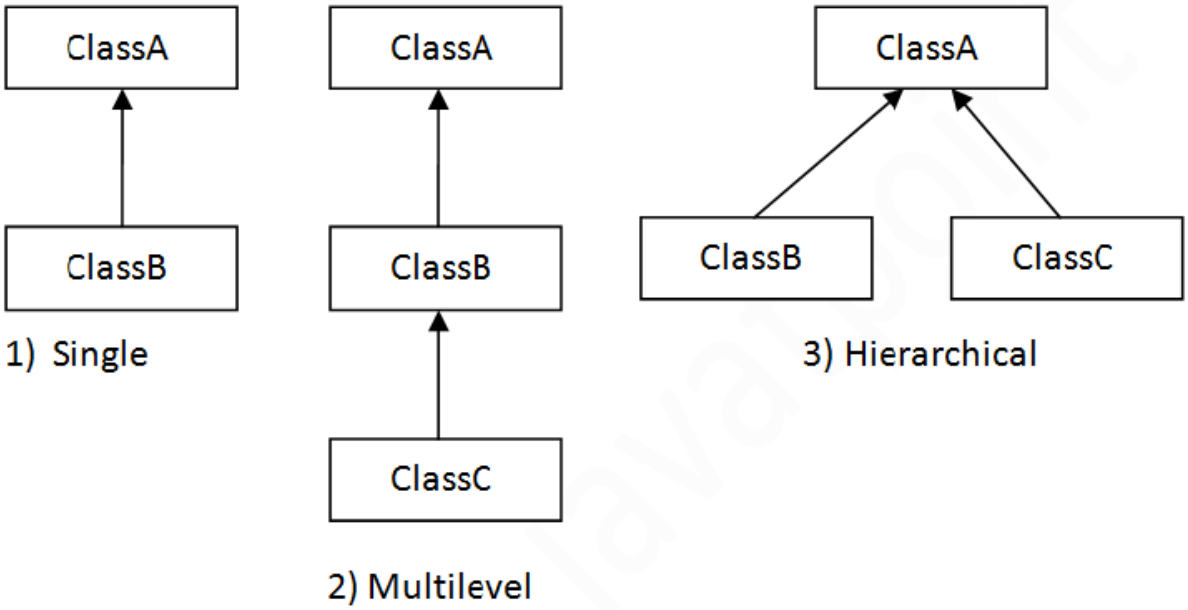
1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hydrid Inheritance

Single Inheritance:
Producing sub classes from a single super class is called single inheritance.

Types of inheritance in java:

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



Example:

```
class A
{
}
class B extends A
{
}
```

2.Multilevel Inheritance

Example:

```
class A
{
}
class B extends A
{
}
```

```
class C extends B
{
}
```

3.Hierarchical Inheritance

Example:

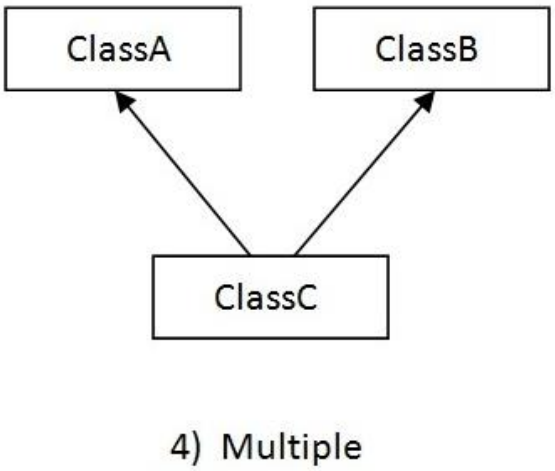
```
class A
{
}
class B extends A
{
}
class C extends A
{
}
class D extends A
{
}
```

4. Multiple Inheritance

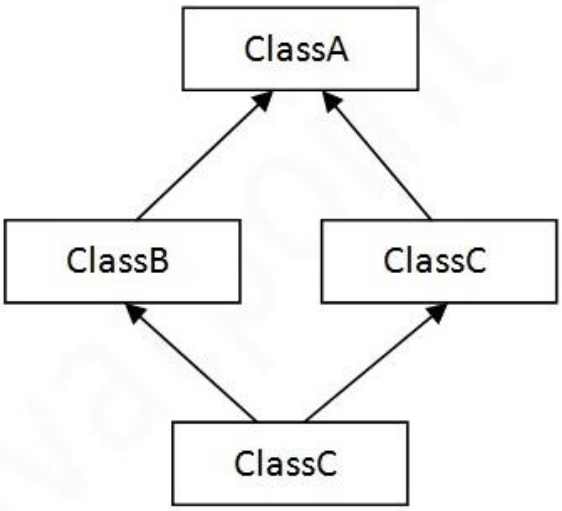
A java class can't extend more than one class at a time hence java would not provide support for multiple inheritance.

Note: Multiple inheritance is not supported in java through class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

Example:

```
class A
{
}
class B
{
}
class C extends A,B
{
}
```

5.Hybrid Inheritance.
combination of Multiple and Hirarchical inheritances is called Hybrid Inheritance.

Java is Not suporting Hybrid Inheritance.
Example:

```
class A
{
}
class B extends A
{
}
class C extends A
{
}
class C extends A,B   //invalid
{
}
```

Note:
1.if our class doesn't extend any other class then only our class is direct child class of Object.

Example:

```
class A
{
}
```

2.if our class extends any other class then our class is indirect child class of Object.
3.Cyclic Inheritance is not allowed in Java ofcoure it is not required.

preventing Inheriatnce

Declare the class Modifier is final then creation of child class object is not possible.

Ex1:

```
final class A
{
	void m1()
	{
		System.out.println("m1 method");
	}

}
class B extends A
{
	public static void main(String[] args)
	{
		new B().m1();
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac B.java
B.java:9: cannot inherit from final A
class B extends A
           ^
1 error

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

Super Keyword:

Representing the current class object use this keyword.

Representing the super class object use super keyword.

```
Class Test
{
	1.parent class variables
	2.parent classMethods
	3.parent classConstructors
	4.parent classinstance blocks
	5.parent classstatic blocks
}
```

1.Parent class variable Representation

```
//parent class variable declaration
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);//local variables 3000
		System.out.println(x+y);//current class variables 300
		System.out.println(a+b);//super class variables 30

	}

	public static void main(String... args)
	{
		new Child().add(1000,2000);
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
3000
300
30

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

```
//parent class variable declaration
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);//local variables 3000
		System.out.println(this.a+this.b);//current class variables 300
		System.out.println(super.a+super.b);//super class variables 30

	}

	public static void main(String... args)
	{
		new Child().add(1000,2000);
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
3000
300
30

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

2.super class method representation

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

```
		System.out.println("Child class m1 method");
	}

	void m2()
	{
		this.m1();
		super.m1();
	}

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

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
Child class m1 method
Parent class m1 method

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

3.Parent class constructors

```
//parent class constructor
class Parent
{
	Parent()
	{
		System.out.println("Parent class 0-arg cons");
	}
}
class Child extends Parent
{
 Child()
	{
	 this(10);
		System.out.println("Child class 0-arg cons");
	}

Child(int a)
	{
```

```
        super();
                System.out.println("Child class 1-arg cons");
        }
        public static void main(String... args)
        {
                new Child();
        }
}
```
output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
Parent class 0-arg cons
Child class 1-arg cons
Child class 0-arg cons

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

```
//parent class constructor
class Parent
{
        Parent()
        {
                System.out.println("Parent class 0-arg cons");
        }
}
class Child extends Parent
{
 Child()
        {
         this(10);
        // super();
                System.out.println("Child class 0-arg cons");
        }

Child(int a)
        {
        // super();
                System.out.println("Child class 1-arg cons");
                super();
        }
```

```
	public static void main(String... args)
	{
		new Child();
	}
}
-----------------------------------
//parent class constructor
class Parent
{
	Parent()
	{
		System.out.println("Parent class 0-arg cons");
	}
}
class Child extends Parent
{
 //compile generated o-arg default cons
 /*

 Child()
	{
		super();
	}
 */


	public static void main(String... args)
	{
		new Child();
	}
}
output:

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
Parent class 0-arg cons

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
-------------------------------------------
4.Parent class instance blocks
//parent class instance block
```

```
class Parent
{

	{
		System.out.println("Parent class instance block");
	}
}
class Child extends Parent
{

	{
		System.out.println("Child class instance block");
	}

	public static void main(String... args)
	{
		new Child();
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
Parent class instance block
Child class instance block

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

```
//parent class instance block
class Parent
{

	{
		System.out.println("Parent class instance block");
	}
	Parent()
	{
		System.out.println("Parent class cons");
	}
}
```

```
class Child extends Parent
{

	{
		System.out.println("Child class instance block");
	}

	Child()
	{
		System.out.println("Child class cons");
	}

	public static void main(String... args)
	{
		new Child();
	}
}
```

5.Parent class Static blocks

```
//parent class instance block
class Parent
{

	{
		System.out.println("Parent class instance block");
	}
	Parent()
	{
		System.out.println("Parent class cons");
	}

	static
	{
		System.out.println("Parent static block");
	}
}
class Child extends Parent
{

	{
		System.out.println("Child class instance block");
```

```
	}

	Child()
	{
		System.out.println("Child class cons");
	}
	Child(int a)
	{
		System.out.println("Child class 1-arg cons");
	}

static
	{
		System.out.println("Child static block");
	}
	public static void main(String... args)
	{
		new Child();
		new Child(10);
	}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>javac Parent.java

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>java Child
Parent static block
Child static block
Parent class instance block
Parent class cons
Child class instance block
Child class cons
Parent class instance block
Parent class cons
Child class instance block
Child class 1-arg cons

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance>
```

Polymorphism:

---

many forms

2 types polymorphism in java
1.Compile Time Poly morphism

static binding
early binding

Example:Method Overloading

2 Runtime Polymormorphism

Dynamic Binding
Late Binding

Ex:Method Overriding

Overriding:
in simple words is process of rewriting.

Overriding requires same signature,return type &equal or more public access modifier.

Rule1:Method Signature is same for overridden and overriding methods.

```
class A
{
void cal(int a)
{
System.out.println("square"+(a*a));
}
}
class B extends A
{
void cal(int a)
{
System.out.println("cube"+(a*a*a));
}
```

```
public static void main(String... args)
{
B obj=new B();
obj.cal(2);
}
}
```

output:

```
D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance\Overriding>java B
cube8

D:\Yellaswamy_ClassNotes\UNIT-2\IICInheritance\Overriding>
```

2.while overriding the return type of overriden and overriding methods must be same.

```
class Animal
{
}
class Monkey extends Animal
{
}
class A
{
Animal m1()
{
System.out.println("A clss m1");
return new Animal();
}
}
class B extends A
{
Monkey m1()
{
System.out.println("B clss m1");

return new Monkey();

}

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

```
{
new B().m1();
}
}
```

Covariant return type:

intriduced in 1.5 version

it is possible to change the return type.

Example:

```
class Animal
{
}
class Dog extends Animal
{
}
class A
{
        Animal m1()
        {
        System.out.println("Animal");
        return new Animal();
        }
}
class B extends A
{
        Dog m1()
        {
        System.out.println("Dog");
        return new Dog();
        }
}
class CovariantReturnTest
{
public static void main(String[] args)
{
        new B().m1();
}
}
```

```
class Test
{
```

```
public static void main(String... args)
{
final int a=10;
a=a+10;
System.out.println(a);
}
}
```

//HAS-A Relation:

```
class A
{
        public void sleep()
        {
        System.out.println("Sleeping");
        }
}
class HasATest
{
public static void main(String[] args)
{
        A obj=new A();
        obj.sleep();
}
}
```

//Dynamic Method Dispatch

```
class A
{
        void calculate(double x)
        {
        System.out.println("Square of A class calculate Method"+x*x);
        }
}
class B extends A
{
        void calculate(double x)
        {
        System.out.println("Cube B class calculate Method"+x*x*x);
        }

}
class C extends A
{
```

```
void calculate(double x)
        {
        System.out.println(" class C calculate Method"+x*x*x*x);
        }
}
class DynamicMethodTest
{
public static void main(String[] args)
{
        C obj1=new C();
        obj1.calculate(3);//c class version
        B obj2=new B();
        obj2.calculate(2);

}
}
```

//Final Demo

```
final class FinaDemoTest
{
        int a=100;
        void m1()
        {
        a=a+10;
        System.out.println(a);
        }
}
class C extends FinaDemoTest
{
        void m1()
        {
        a=a+100;
        System.out.println(a);
        }

public static void main(String[] args)
{
new FinaDemoTest().m1();
}
}
```

//4.Overridden method is declared as final method,it is not possible to override in chilc class

```
 class Parent
{
        final int a=10;

//overridden method
                void m2()
                {
                        a=200;
                System.out.println("Parent m1 method"+a);
                }
}
class child extends Parent
{
        final void m1()
        {
        System.out.println("Child m1 method");
        }



}
class FinalTest
{
public static void main(String[] args)
{
        child c=new child();
        c.m1();
        c.m2();

}
}
```