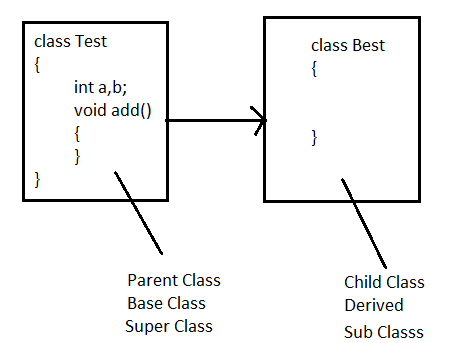
**Inheritance**

* The process of getting variables (properties) and methods (behavior) from one class to another class is called inheritance.
* The main objective of inheritance is code extensibility.
* In inheritance one class giving the properties and behavior (i.e. parent class/ base class/ super class ) and another class is taking the properties and behavior (i.e. child class/ derived class/ sub class).

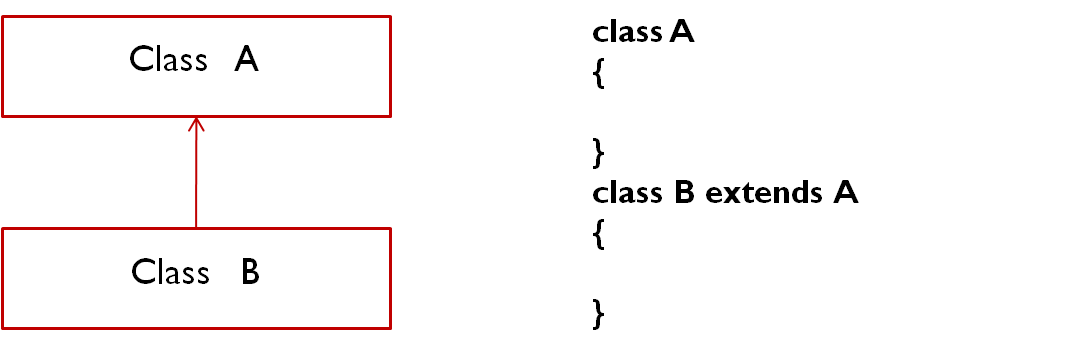


**Types of Inheritance**

* There are five types of inheritance in java
  + Single inheritance
  + Multilevel inheritance
  + Hierarchical inheritance
  + Multiple inheritance
  + Hybrid Inheritance

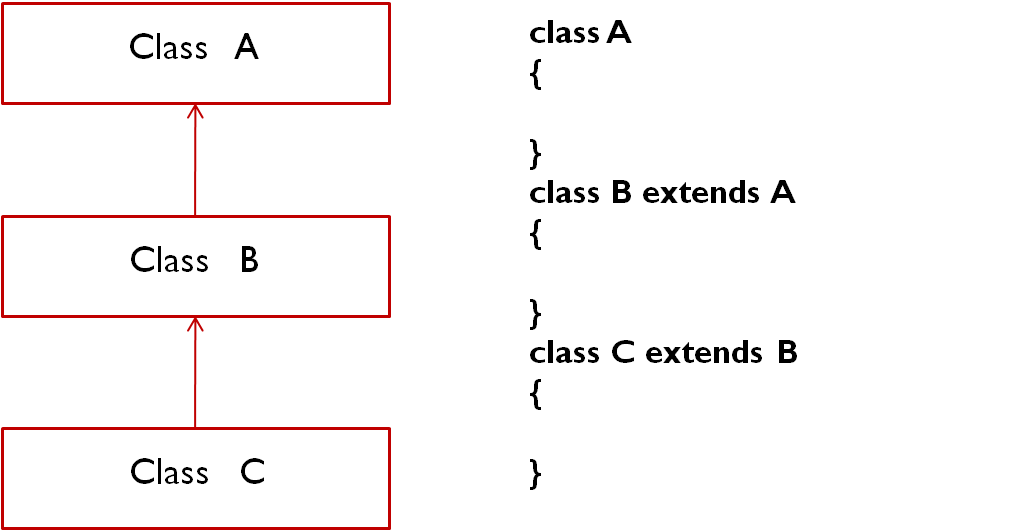
**Single inheritance**

* One class has one direct super class is called single inheritance.



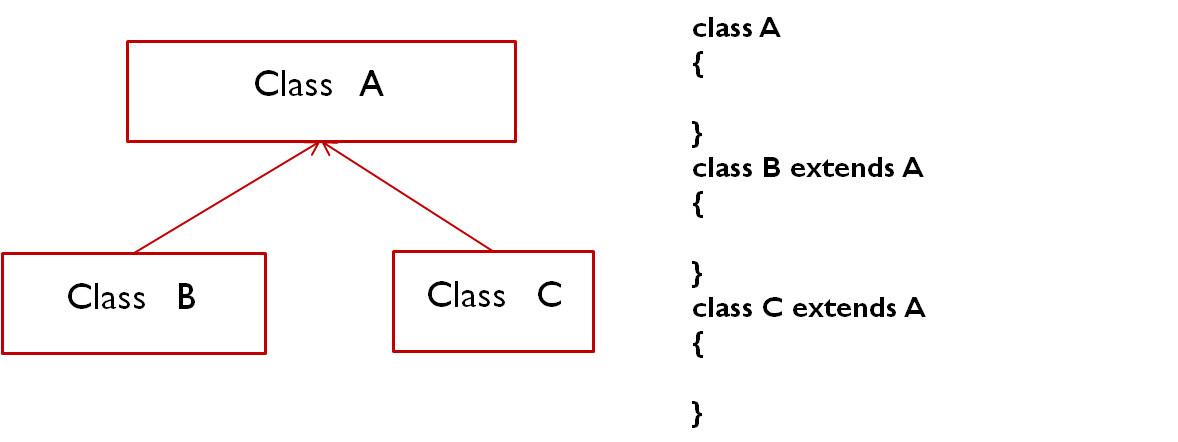
**Multilevel inheritance**

* One Sub class is extending Parent class then that sub class will become Parent class of next extended class this flow is called multilevel inheritance.



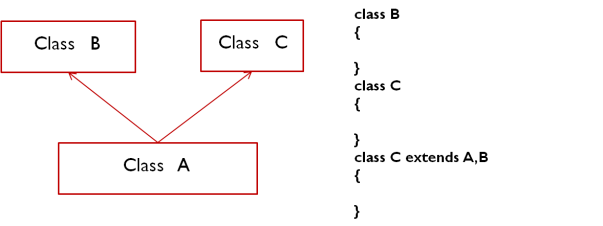
**Hierarchical inheritance**

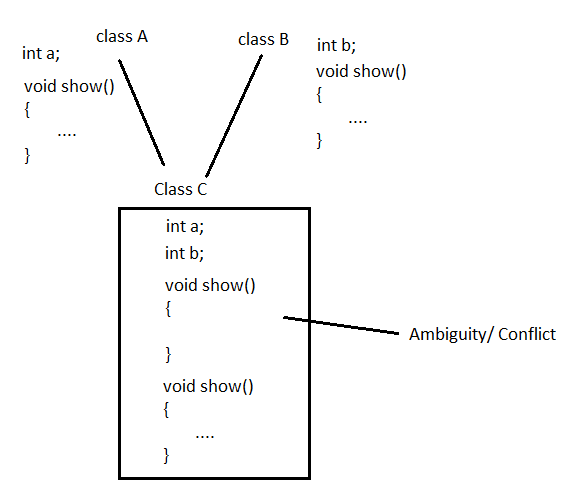
* More than one sub class is extending single Parent is called hierarchical inheritance.



**Multiple Inheritance**

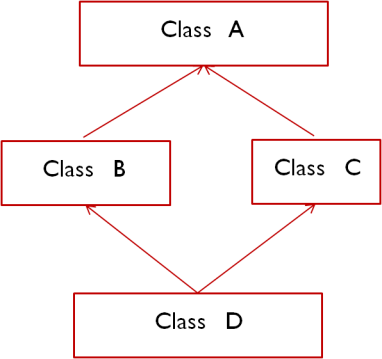
* One sub class is extending more than one super class is called Multiple inheritance and java not supporting multiple inheritance because it is creating ambiguity problems.

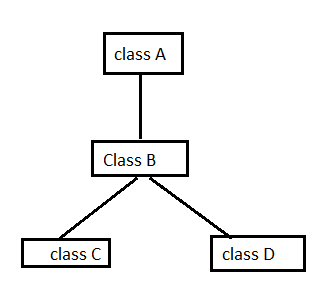




**Hybrid Inheritance**

* Hybrid Inheritance is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance , the hybrid inheritance is also not possible.





**Example#1**

class Test{

int a = 10;

void show(){

System.out.println("Test Class - Show method");

}

}

class Best extends Test{

int b = 20;

void display(){

System.out.println("Best class - display method");

}

}

class SingleInheritanceDemo{

public static void main(String args[]){

Best ob = new Best();

System.out.println("b = "+ob.b);

ob.display();

System.out.println("a = "+ob.a);

ob.show();

}

}

**Example#2**

class Test{

int a = 10;

void show(){

System.out.println("Test Class - show method");

}

}

class Best extends Test{

int b = 20;

void display(){

System.out.println("Best class - display method");

}

}

class Nest extends Best{

int c = 10;

void print(){

System.out.println("Nest Class - print method");

}

}

class MultilevelInheritanceDemo{

public static void main(String args[]){

Nest ob = new Nest();

System.out.println("a = "+ob.a);

System.out.println("b = "+ob.b);

System.out.println("c = "+ob.c);

ob.show();

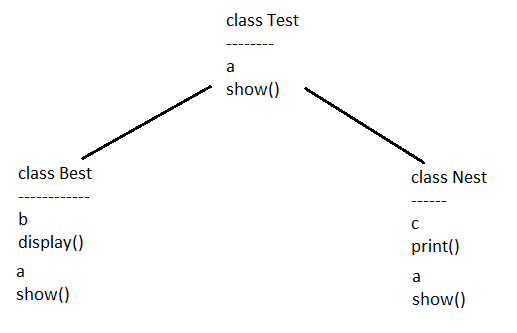
ob.display();

ob.print();

}

}

**Example#3**



class Test{

int a = 10;

void show(){

System.out.println("Test Class - show method");

}

}

class Best extends Test{

int b = 20;

void display(){

System.out.println("Best class - display method");

}

}

class Nest extends Test{

int c = 10;

void print(){

System.out.println("Nest Class - print method");

}

}

class HierarchicalInheritanceDemo{

public static void main(String args[]){

Best ob1 = new Best();

System.out.println("a = "+ob1.a);

System.out.println("b = "+ob1.b);

ob1.show();

ob1.display();

Nest ob2 = new Nest();

System.out.println("a = "+ob2.a);

System.out.println("c = "+ob2.c);

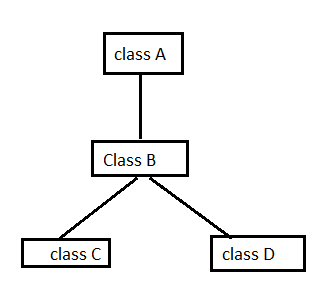
ob2.show();

ob2.print();

}

}

**Example#4**



class A{

int a = 10;

}

class B extends A{

int b = 20;

}

class C extends B{

int c = 30;

}

class D extends B{

int d = 40;

}

class HybridInheritanceDemo{

public static void main(String args[]){

B ob1 = new B();

C ob2 = new C();

D ob3 = new D();

System.out.println("a = "+ob1.a);

System.out.println("b = "+ob1.b);

System.out.println("a = "+ob2.a);

System.out.println("b = "+ob2.b);

System.out.println("c = "+ob2.c);

System.out.println("a = "+ob3.a);

System.out.println("b = "+ob3.b);

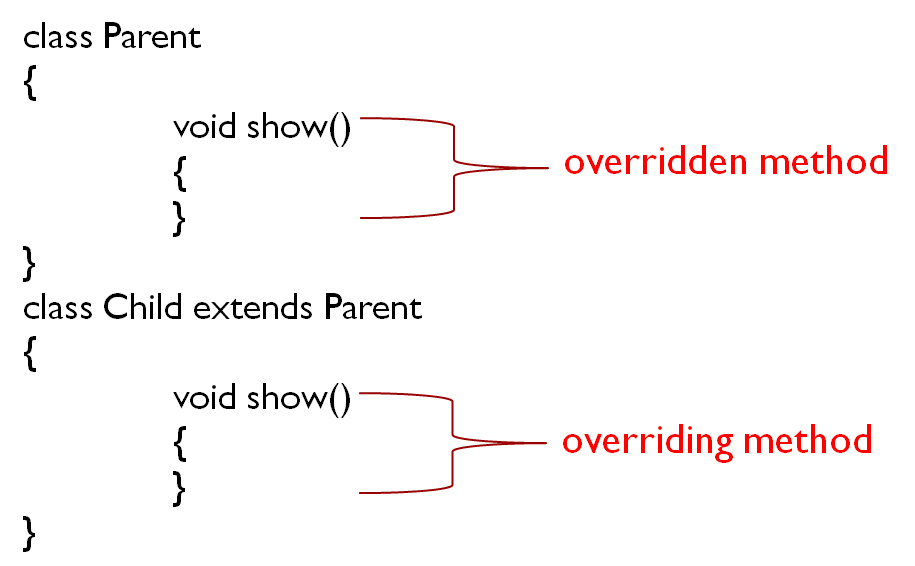
System.out.println("d = "+ob3.d);

}

}

**Method Overriding**

* Redefining or Redesigning parent class method in child class is known as method overriding.
* In method overriding, parent class method signature and child class method signature must be same.



**Example**

class Parent{

void show(){

int a = 10;

int b = 20;

int c = a+b;

System.out.println("sum1 = "+c);

}

}

class Child extends Parent{

void show(){

int a = 10;

int b = 20;

System.out.println("sum2 = "+ (a+b));

}

}

class MethodOverridingDemo{

public static void main(String args[]){

Child ob = new Child();

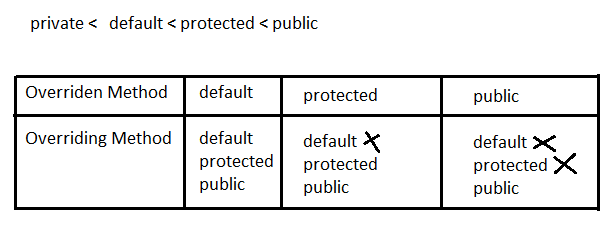
ob.show();

}

}

**Rule#1**

While overriding the methods it is possible to maintain same level permission or increasing order but not decreasing order.



class Parent{

public void show(){

System.out.println("Parent - show method");

}

}

class Child extends Parent{

void show(){

System.out.println("Child - show method");

}

}

class MethodOverridingDemo{

public static void main(String args[]){

Child ob = new Child();

ob.show();

}

}

**Rule#2**

The return types of overridden method & overriding method must be same.

class Parent{

int show(){

return 11;

}

}

class Child extends Parent{

int show(){

return 10;

}

}

class MethodOverridingDemo{

public static void main(String args[]){

Child ob = new Child();

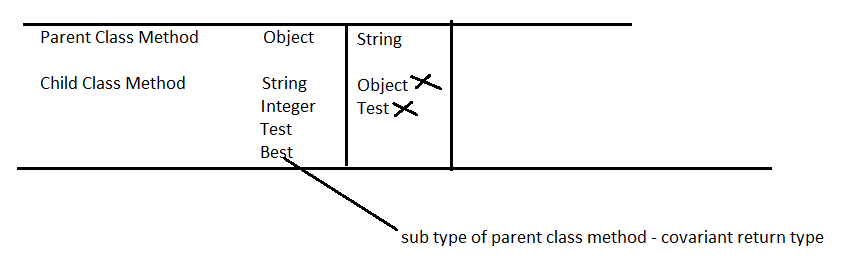
ob.show();

}

}

**Rule#3**

We can use covariant-return types. The return type of overriding method is must be sub-type of overridden method return type this is called covariant return types.



class Parent{

Object show(){

return null;

}

}

class Child extends Parent{

String show(){

return null;

}

}

class MethodOverridingDemo{

public static void main(String args[]){

Child ob = new Child();

ob.show();

}

}

**Note**

Parent class reference variable is able to hold child class object but Child class reference variable is unable to hold parent class object.

class Test{

}

class Best extends Test{

}

class Main{

public static void main(String args[]){

Test ob1 = new Test();

Best ob2 = new Best();

Test ob3 = new Best();

Best ob4 = new Test(); // not allowed

}

}

**Dynamic method dispatch / Runtime Polymorphism**

* Dynamic method dispatch is a technique by which call to a overridden method is resolved at runtime.
* When an overridden method is called by a reference, then which version of overridden method is to be called is decided at runtime according to the type of object it holds.
* Dynamic method dispatch is performed by JVM not compiler.

class Test{

void show(){

System.out.println("Test class - show method");

}

}

class Best extends Test{

void show(){

System.out.println("Best class - show method");

}

}

class Main{

public static void main(String args[]){

Test ob = new Best();

ob.show();

}

}

**Final Modifier**

* Final keyword provides restriction.
* Final is the modifier applicable for classes, methods and variables.

**Final Class**

* If a class is declared as final, then we cannot create child class for that class.
* It prevents inheritance.

final class Test{

void show(){

System.out.println("Test class - show method");

}

}

class Best extends Test{

void show(){

System.out.println("Best class - show method");

}

}

class Main{

public static void main(String args[]){

Test ob = new Best();

ob.show();

}

}

**Final Method**

* If a method declared as a final we cannot override that method in child class.
* It prevents overriding.

class Test{

final void show(){

System.out.println("Test class - show method");

}

}

class Best extends Test{

void show(){

System.out.println("Best class - show method");

}

}

class Main{

public static void main(String args[]){

Test ob = new Best();

ob.show();

}

}

**Final Variable**

* If a variable declared as a final we cannot reassign that variable.

class Main{

public static void main(String args[]){

final int a = 10;

a++;

System.out.println(a);

}

}