Static methods are also called Generic methods

Non-static methods are also called as Specific methods.

Inheritance:

Code Reusability

IS-A: Inheritance

---Parent-Child Relationship

---Base Class-Sub Class Relationship

---Is-A Relationship

---Existing class—Derived class Realtionship

Has-A: Dependency Injection

Aggregation/Composition

One class acquiring properties and behaviour from another class---Inheritance

Un related classes--🡪Linked with the help of extends keyword.

**class** Demo1

{

String name="Chinmay";

**int** age=25;

**public** **void** disp()

{

System.***out***.println("Name: "+name+" Age : "+age);

}

}

**class** Demo2 **extends** Demo1

{

}

//Demo2 is Child/Derived/Sub class

//Demo1 is Parent/base/Existing class

**public** **class** LaunchInh {

**public** **static** **void** main(String[] args) {

Demo2 d1=**new** Demo2();

d1.disp();

}

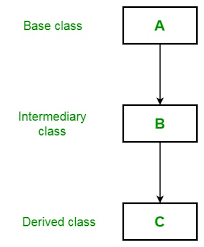
}

Object class is the Parent of all the Classes

Single Inheritance:

* One class can extend another class.

Multilevel Inheritance:



**class** Demo11

{

String name="Chinmay";

**int** age=25;

**public** **void** disp()

{

System.***out***.println("Name: "+name+" Age : "+age);

}

}

**class** Demo21 **extends** Demo11

{

}

**class** Demo3 **extends** Demo21

{

}

**public** **class** LaunchMultiLevel {

**public** **static** **void** main(String[] args) {

Demo3 d1=**new** Demo3();

d1.disp();

}

}

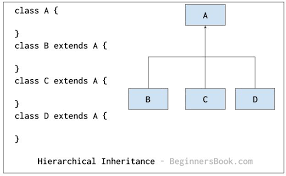
In the above code for Demo11 even we have not extended with any class by default it will extend Object class internally like below. So what ever will be Properties and behaviour of Object class will be included in the Demo11 class. The classes which extends the Demo11 will also these Properties and Behaviours.

class Demo11 extends Object{

}

Hierarchical Inheritance:

One parent can have any number of child classes.



Hybrid Inheritance:

It is a mix of two or more of the above types of inheritance. Since Java doesn’t support multiple inheritances with classes, hybrid inheritance involving multiple inheritance is also not possible with classes. In Java, we can achieve hybrid inheritance only through [Interfaces](https://www.geeksforgeeks.org/interfaces-in-java) if we want to involve multiple inheritance to implement Hybrid inheritance.

**class** Parent

{

String name="Sai";

**int** age=25;

**public** **void** disp()

{

System.***out***.println("Name: "+name+" Age : "+age);

}

}

**class** Child1 **extends** Parent

{

}

**class** Child2 **extends** Parent

{

}

**class** GChild1 **extends** Child1

{

}

**public** **class** LaunchHybrid {

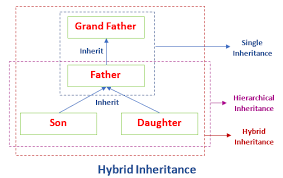
**public** **static** **void** main(String[] args) {

GChild1 d1=**new** GChild1();

d1.disp();

}

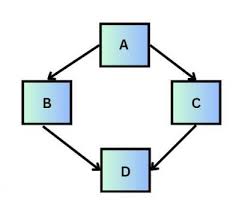
}



Multiple Inheritance:

Diamond Shaped Problem:

Java will fell into Ambiguous problem like which properties/behaviours it needs to consider from class A/class B.

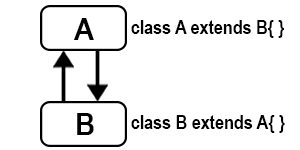


class D extends A,B—this is not allowed

we can’t directly have multiple inheritance in java but we can implement using interfaces.

Cyclic Inheritance:

Cyclic Inheritance is not allowed in Java.



Private members of a class will not participate in Inheritance to preserve Encapsulation.—To avoid direct access to data members.

No, the private member are ***not inherited*** because the scope of a private member is **only limited** to the class in which it is defined. Only the public and protected member are inherited.

Constructors will not participate in Inheritance. But the child class will have super() in the constructor so the Constructor of parent will also gets Executed.

A screen shot of a computer program

Description automatically generated

Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.

class Parent {

public Parent()

{

}

public void print()

{

}

}

public class Child extends Parent {

public Parent()

{

}

public void print()

{

}

public static void main(String[] args)

{

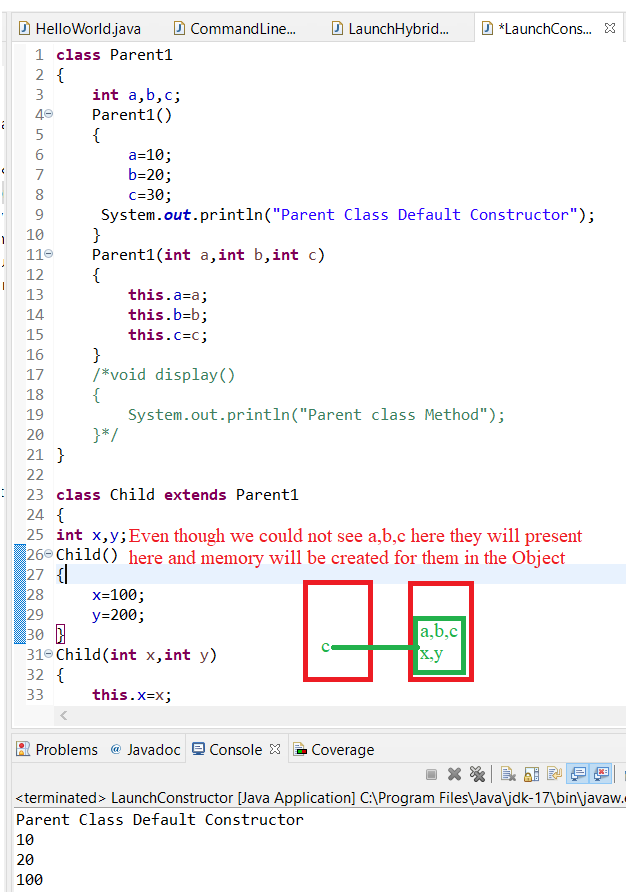
Child c1 = new Child(); // allowed

Child c2 = new Parent(); // not allowed

}

}

If we define Parent class constructor inside Child class it will give compile time error for return type and consider it a method. But for print method it does not give any compile time error and consider it a overriding method.



**class** Parent1

{

**int** a,b,c;

Parent1()

{

a=10;

b=20;

c=30;

System.***out***.println("Parent Class Default Constructor");

}

Parent1(**int** a,**int** b)

{

**this**.a=a;

**this**.b=b;

//this.c=c;

System.***out***.println("Parent class parametrized Constrcutor");

}

/\*void display()

{

System.out.println("Parent class Method");

}\*/

}

**class** Child **extends** Parent1

{

**int** x,y;

Child()

{

x=100;

y=200;

}

Child(**int** x,**int** y)

{

**super**(x,y);

**this**.x=x;

**this**.y=y;

}

**void** display()

{

System.***out***.println(a);

System.***out***.println(b);

System.***out***.println(x);

System.***out***.println(y);

}

}

**public** **class** LaunchConstructor {

**public** **static** **void** main(String[] args) {

Child c=**new** Child();

c.display();

Child c1=**new** Child(10,20);

c1.display();

}

}

Output:

Parent Class Default Constructor

10

20

100

200

Parent class parametrized Constrcutor

10

20

10

20

Ex 2:

**class** Parent1

{

**int** a,b,c;

Parent1()

{

a=10;

b=20;

c=30;

System.***out***.println("Parent Class Default Constructor");

}

Parent1(**int** a,**int** b)

{

**this**.a=a;

**this**.b=b;

//this.c=c;

System.***out***.println("Parent class parametrized Constrcutor");

}

/\*void display()

{

System.out.println("Parent class Method");

}\*/

}

**class** Child **extends** Parent1

{

**int** x,y;

Child()

{

**this**(111,112);

x=100;

y=200;

}

Child(**int** x,**int** y)

{

**super**(x,y);

**this**.x=x;

**this**.y=y;

}

**void** display()

{

System.***out***.println(a);

System.***out***.println(b);

System.***out***.println(x);

System.***out***.println(y);

}

}

**public** **class** LaunchConstructor {

**public** **static** **void** main(String[] args) {

Child c=**new** Child();

c.display();

Child c1=**new** Child(10,20);

c1.display();

}

}

Output :

Parent class parametrized Constrcutor

111

112

100

200

Parent class parametrized Constrcutor

10

20

10

20



