Inheritance

- It represents parent-child relationship
- superclass: The class whose features are inherited is known as super class / base class / parent class.
- subclass: The class that inherits the other class is known as subclass (derived class, extended class, child class) The subclass can add its own fields & methods in addition to the superclass fields & methods.
- Reusability: This is done by creating new classes, reusing the properties of existing ones.
- keyword used for ~~creating~~ inheritance

extend

syntax:

```
class derived class extends base class
{
//methods + fields
}
```

The keyword extends signifies that the properties of the super class name are extended to the sub-class name.

Inheritance example :

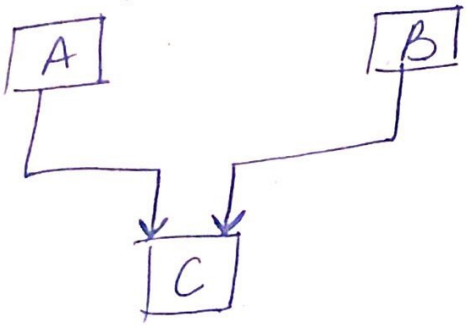
```
class Employee {
    float salary = 20000;
}
class programmer extends Employee
{
    int bonus = 5000;
public static void main (String args[])
{
programmer p = new programmer();
S.O.println ("prog sallary is : " + p.salary);
S.O.println ("Bonus of prog is : " + p.bonus);
}}
```

→ Types of Inheritance.

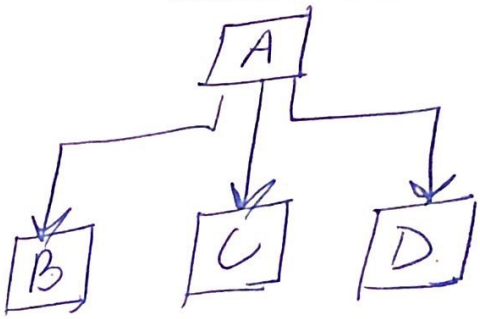
① single inheritance.

② multilevel inheritance

③ multiple Inheritance



④ Hierarchical inheritance.



- It doesnot directly implement multiple inheritance, this concept is implemented by using interfaces.

① single inheritance.

```
class A
{ pubeic void methodA()
  {
  S.O.pln("Base class method");
  }}
```

```
class B extends A
{ public void methodB()
  {
    S.O.println("child class method");
  }
public static void main(String args[])
  {
    B obj = new B();
    obj.methodA();
    obj.methodB(); }}
```

(2) Multiple inheritance

The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes. This is very complex. and leads to problems in the hierarchy. It is rarely used in software projects

(3) multilevel inheritance

```
class Y enlends X
  {
  public void method Y()
{
S.o.pln("class Y method");
        }}
class Z enlends Y
{
public void methodZ()
  {
  S.o.pln("class Z method");
      }
public static void main(String args[])
  {
    Z obj = new Z();
     obj.method X();
     Obj.method Y();
     obj.method Z();  }}
```

- It refers to a mechanism where one can inherit from a defined class, thereby making this defined class the base class for the new class. In eg, you can see. X is subclass of Y and. Y is subclass of Z.

(8) Hierarchical inheritence

When a class has more than one child classes (subclasses) or in other words more than one child classes have the same parent class.

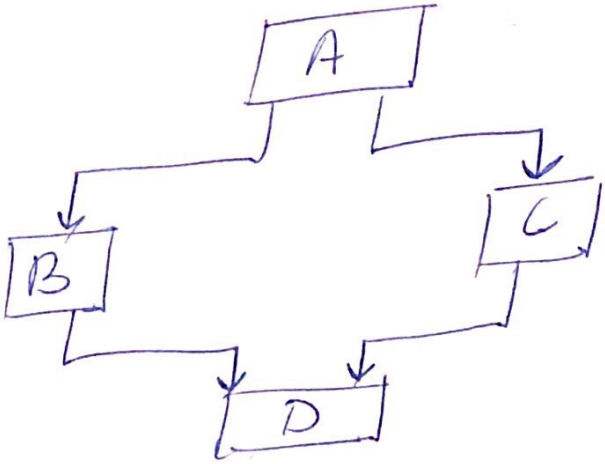
```
class A
{ public void methodA()
  {
  S.O.println("method of class A");
  }}

  class B extends A
{ public void methodB()
{
S.O.println("method of class B");
  }}
class C extends A
  { public void methodC()
{
S.O.println("method of class C");
  }}

  class D extends A
{ public void methodD()
{ S.O.println("method of class D");
  }}
```

```
class example
{
public static void main(String args[])
{
B obj1 = new B();
C obj2 = new C();
D obj3 = new D();
obj1.method A();
obj2. method A();
obj3. method A();
}}
```

output:
method of class A
method of class A
method of class A

(5) **Hybrid inheritance**

It is combination of single and multiple inheritence. It can be achieved in java using interfaces.



→ Construction chaining

- calling one construction from another construction is generally referred to as construction chaining.

this() super()

- this()

eg.

```
class Person
{ person()
{ this(10,20);
  S.O.pln("A");
}

Person(int x)
{
S.O.pln("B");}

Person(int x, int y)
{ this(x);
  S.O.pln("C");
}}

class chain
{public static void main(String args[])
{ person p2 = new Person(); }}
```

output : B C A

- **super()**
- If we want any argumented constructor of the parent class to be called with a child class constructor then we can use super() to invoke it.
- using super() without arguments will call the default constructor of the parent class.
- when a child class object is created, even though the control comes to the child class constructor first, it will not execute the code in child class constructor. From child class constructor it will go to the parent class constructor and executes it, then only it comes back to execute the child class constructor.

eg of super().

```
class Person
{ Person()
  { s.o.p("A"); }
Person(int x)
{ s.o.p("B"); }
```

(1) (3) (5)

```
Person(int x, int y.)
{
  S.o.p("C"); }}.

class Emp extends Person
{ Emp() → this will refer to parent's default constructor as there is no constructor specified. so, it will print
  {
    S.o.p("X"); (2)
  }
Emp(int x)
{
  super(); → refers to parent default constructor
  S.o.p("Y"); (4)
}
```

AX

AY

```
Emp(int x, int y)
{ super(x); → refers to parent 1 argument constructor
  S.o.p("Z"); (6) }}
```

BZ.

```
class Chain
{ public static void main(String args[])
{
  Emp e1 = new Emp();
  Emp e2 = new Emp(10);
  Emp e3 = new Emp(20, 30);
}}
```

output.

AX AYBZ.

1 2 3 4 5 6