UNIT-2

Inheritance: Deriving new classes from existing classes such that the new classes acquire all the feature of existing classes is called inheritance.

Advantage:

→ In inheritance, a programmer reuses the super class code without rewriting it, in creation of sub classes.

→ So, developing the classes becames very easy. Hence, the programmer's productivity is increased.

Syntax:

→ The key word "extends" is used to define inheritance in Java.

```
class subclassname extends superclassname
{
    =
    =
    =
}
```

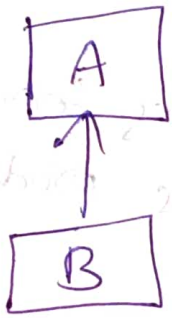
Types of inheritance.

(1) Single inheritance (2) Hierarchical inheritance.

(3) Multilevel inheritance (4) Multiple inheritance (not available in java)

(1) Single inheritance.

→ Derivation of a class from only one base class is called single inheritance.



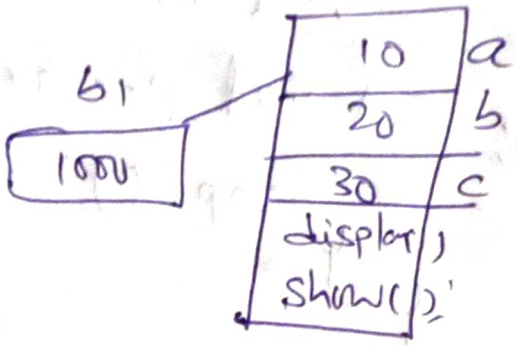

```
class A
{ // members of A class
}
class B Extends A
{ // members of "B" class
  & members of "A" class
}
```

Eg:-

```
class A
{ int a=10, b=20;
  void display()
  { sop(a);
    sop(b);
  }
}
```

```
class B extends A
{ int c=30;
  void show()
  { sop(a);
    sop(b);
    sop(c);
  }
}
```

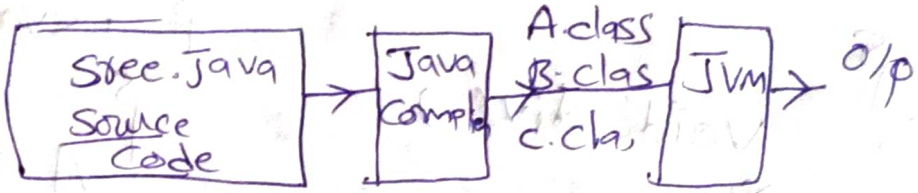
```
class C
{   psvm(S z[])
    {
        B b1 = new B();
```



b1.display(); o/p: 10, 20

b1.show(); o/p: 10, 20, 30

```
    }
}
```



> Javac Sree.java

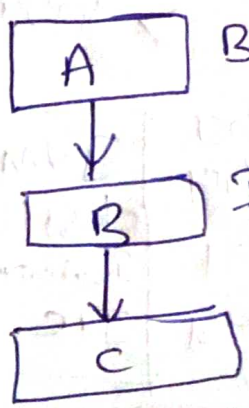
A.class
B.class
C.class

(ii)

> java B

(ii) Multilevel Inheritance:-

→ In multilevel inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.



```
class A
{ int a=10, b=20;
  void sum()
  { int s=a+b;
    sop(s);
  }
}
```

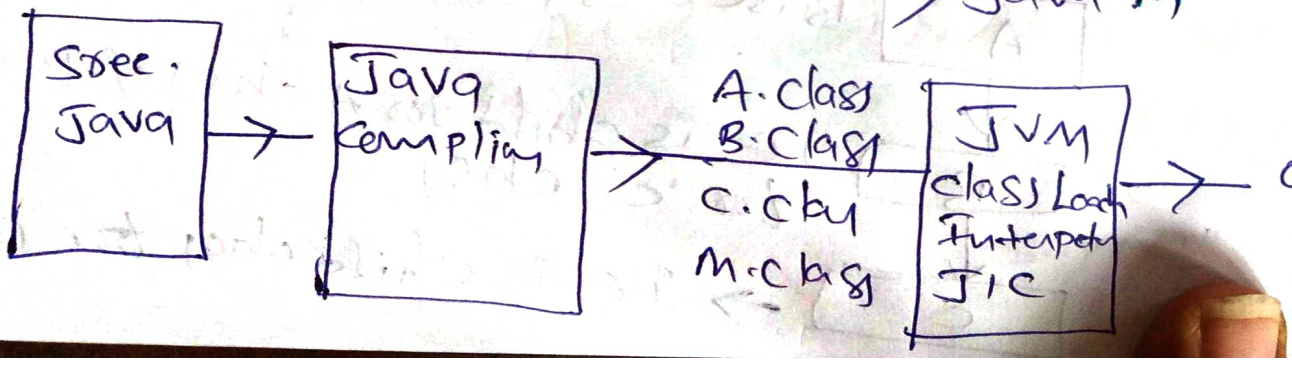
```
class B extends A
{ int c;
  void mul()
  { int p=a*b*c;
    sop(p);
  }
}
```

```
class C extends B
{
  void div()
  { int d=(a+b)/c;
    sop(d);
  }
}
```

```
class M
{ psvm(String z[])
  { A a1=new A();
    a1.sum();
    B b1=new B();
    b1.sum();
    b1.mul();
    C c1=new C();
    c1.sum(); c1.mul();
    c1.div();
  }
}
```

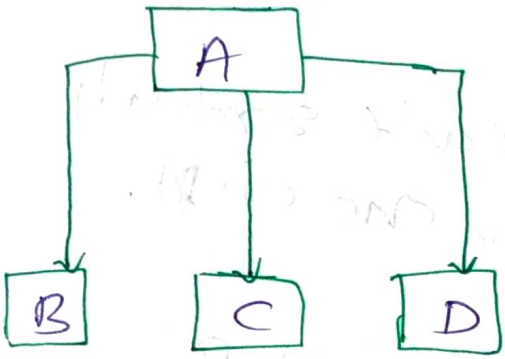
Source Code > Javac sree.java

> java M



(iii) Hierarchical Inheritance:

→ One class as a superclass (base class) for more than one sub class is called hierarchical Inheritance.



→ Here "A" is base class for the derived class B, C, D.

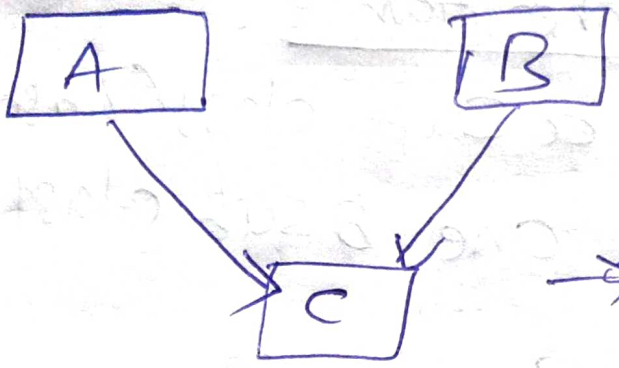
Eg:-

```
class A
{
}

class B extends A
{
}

class C extends A
{
}

class D extends A
{
}
```

(IV) Multiple Inheritance: One class can have more than one superclass is called multiple Inheritance.

→ Note: Java does not support multiple inheritance with classes.

→ In Java we can achieve multiple inheritance only through Interfaces.



→ Here "C" class has two parents.

class C extends A, B // Not valid.
{
}

Note: We can't extends more than one class.

(or)

class C extends A extends B
{
}

It is also wrong.

Note: We can extends only one class at a time.

→ Relation of subclass object & superclass object (or) Reference;

→ Subclass object we can assign to superclass reference variable (or) object. but reverse is not possible.

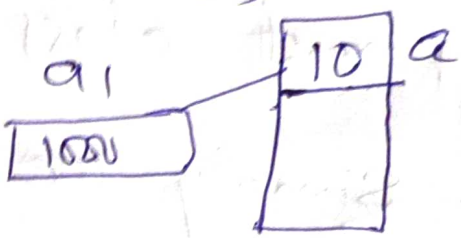
Syntax:

Eg:-

```
A a1 = new A();
A a2;
B b1 = new B();
a1 = b1;
a2 = b1;
```

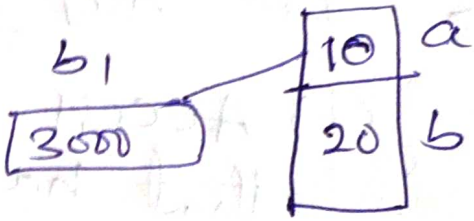
Eg:
```
class A
{ int a=10;
}
class B extends A
{ int b=20;
}
```

```
class M
{ psvm(String z[])
  { A a1=new A();
    A a2;
```



a2
2000 → memory is not allocated

```
    B b1=new B();
```




```
    a1=b1;
    a2=b1;
    b1=a1 // Not valid
    SOP(a1.a);
    SOP(a1.b) // error.
  }
}
```

→ Using a_2 we can access only "A" class members.

→ for obied memory will be allocated.

→ javac Sree.java
// A.class, B.class, M.class

→ java M.

O/P: 10.

super keyword

→ The super keyword in java is a reference variable that is used to refer parent class object.

→ simply we can say that super keyword refers to superclass objects.

→ It has 2 advantages

(i) Access the member of superclass that has been hidden by member of subclass

(ii) To calls the superclass constructor.

First advantage example: It is like a this keyword

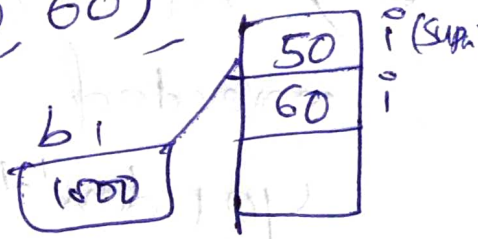
Syntax: super.member.

→ Here member can be either a method (or) an instance variable.

→

Eg1:

```
class A
{ int i=10;
  void display()
  { sop(i);
  }
}

class B extends A
{ int i=20;
  B(int a, int b)
  { super.i=a;
    i=b;
  }
  void show()
  { sop("superclass i"+super.i);
    sop("subclass i"+i);
  }
}

class C
{ public static void main(String ZL)
  { B b1 = new B(50, 60);
    b1.show();
  }
}
```



2nd advantage of super keyword:

→ Using super keyword call the superclass constructors from subclass constructors

Syntax: super(parameter-list);

→ parameter-list specifies the what type of constructor you want to call

super(10) // It can call the one parameter constructor

super() // It is nothing but default constructor.

Note:

→ super() must be the first statement executed inside a subclass constructor. If you don't call any parameter constructor.

Eg1:

```
class A
{
    A()
    {
        SOP("It is default constructor in superclass");
    }
    A(int a)
    {
        SOP("It is 1 parameter constructor in super
                class");
    }
}
```

```
class B Extends A
{
    B()
    { sop("It is default constructor in subclass");

    }
}
class C
{
    psvm(String zL))
    {
        B b1 = new B();

    }
}
```

O/P:-

Eg 2:

```
class A
{
  A()
  {
    sop("superclass");
  }
  A(int a)
  {
    sop("paramed con"+a);
  }
}
```

```
class B extends A
{
  B()
  {
    super(30);
    sop("Subclass");
  }
}
```

```
class C
{
  psvm(S [])
  {
    B b1 = new B();
  }
}
```

When constructors are called:

→ In class hierarchy constructors are called in order of derivation, from super class to subclass.

→ super() must be the first statement is executed in a subclass constructor.

→ Eg1:-

```
class A
{
    A()
    {
        SOP("Inside A's Constructor");
    }
}
class B extends A
{
    B()
    {
        SOP("B Constructor");
    }
}
class C extends B
{
    C()
    {
        SOP("C Constructor");
    }
}
```

```
class M
{
    PSVM(S Z[])
    {
        C c1 = new C();
    }
}
```

O/P:-

polymorphism-

→ Assigning multiple meaning to the same method name is called polymorphism.

→ Implemented using late binding (or) dynamic binding (run-time binding). It means, method to be executed is determined at execution time, not at compile time.

polymorphism can be implented in two ways.

1. Method overloading
2. method overriding.

Method overriding:-

→ Declaring a method in subclass which is already present ~~class~~ in super class is known as method overriding.

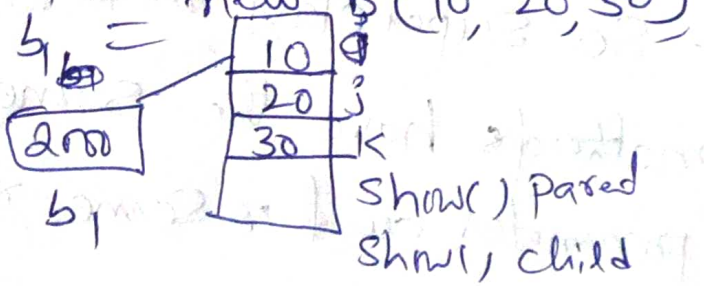
→ So superclass method and subclass methods have the same name, same parameter and same return type.

Eg:-

```
class A
{ int i,j;
  A(int a, int b)
  {
     i=a;
     j=b;
  }
  void show()
  { sop(i);
    sop(j);
  }
}
```

```
class B extends A
{ int k;
  B(int a, int b, int c)
  {
     super(a,b);
     k=c;
  }
  void show()
  { sop(i)
    sop(k);
  } sop(j);
}
```

```
class M
{ psvm(String z[])
  {
    B b1 = new B();
    B b1 = new B(10, 20, 30);
  }
}
```




```
    b1.show();
  }
}
```

*** Dynamic Method Dispatch:

→ Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run-time, rather than compile time.

→ when an overridden method is called through a superclass reference, the method to execute will be based on the type of object being referred to at the time the call occurs. Not the type of the reference variable.

Eg1:

```
class A
{
    void callme()
    {
        sop("It is 'A' method");
    }
}
class B extends A
{
    void callme()
    {
        sop("It is B method");
    }
}
class C extends A
{
    void callme()
    {
        sop("It is C method");
    }
}
class M
{
    psvm(S z[])
    {
        A r1;
        B b1 = new B();
        r1 = b1;
        r1.callme();
        C c1 = new C();
        r1 = c1;
        r1.callme();
    }
}
```

Using final with Inheritance:

The keyword final has three uses

(i) To create a constant variable

(ii) To prevent method overriding

(iii) To prevent Inheritance.

To create a Constant Variable:

→ A variable can be declared as final that variable - we can't modify (or) change.

eg:-
```
class A
{
  psvm(String Z[])
  {
    final int i=20;
    sop(i);
    i=i+1 // not possible.
  }
}
```

To prevent overriding:-

→ A method, declared with the final keyword, cannot be overridden (or) hidden by subclasses.

→ method overriding is not possible with final keyword

Eg:-

```
class A
{   int a=10;
final void show()
    {  sop(a);
    }
}
```

```
class B extends A
{   int b=20;
    void show()
    {  sop(a);
       sop(b);
    }
}
```

Note: Not possible.

(iii) To prevent inheritance:-

→ A class declared as a final class can't be subclassed (or) can't be inherited.

Eg:-

```
final class A
{   int a=10;
    void show()
    {  sop(a);
    }
}
class B extends A // It gives error.
{
  =
  =
}
```

Abstract classes

→ Any method that has been declared without body in a class is called abstract method

→ Any class that contains at least one abstract method that class became abstract class.

→ You must declare the abstract method with the keyword abstract.

Syntax:- ~~abstract returntype name(the~~

```
abstract returntype name(the method(---);
```

Eg:- abstract void show();

Eg1:-
```
class A
{
    int a=10, b=20;
    void sum()
    {
        int s=a+b;
        sop(s);
    }
abstract void mul();
}
```

complete difference b/w abstract class and class.

class	Abstract class
① class is model for creating the objects	① Abstract class is a class that contains one or more abstract methods.
② class has only concrete methods only.	② It has concrete methods & abstract method.
③ It hay complete class with method body	③ An abstract class is incomplete, It has "missing method bodies"
④ we can create object & reference variable	④ we can't create object but we can create reference variable.

→ A abstract class is completed by inheritance.

→ you can extend an abstract class.

→ If the subclass define all inherited abstract method, then subclass became complete.

→ If the sub class does not define all inherited abstract method, it is also become an abstract class.

Eg1:-

```
class A
{ int a=10,b=20;
  void sum()
  { int s=a+b;
    sop(s);
  }
abstract void mul();
}
```

```
class B extends A
{
  void mul()
  { int m=a*b;
    sop(m);
  }
  int div()
  { int d=a/b;
    return d;
  }
}
```

```
class C
{
  psvm(S z[])
  { A a1=new A();
    A a2;
    B b1=new B();
    a2=b1;
    a2.sum();
    a2.mul();
    b1.div();
  }
}
```

Object class

→ Object is a special class, defined by Java.
→ It is available in lang package.
→ Object is a superclass of all other classes.

Object defines the following methods

(i) **Object clone()**:- creates a new object that is the same as the object being cloned.

(ii) **equals()**:- This method compares the references of two objects, and if they are equal it returns true, otherwise false.

```
A a1 = new A();
A a2 = new A();
if(a1.equals(a2))
```

(iii) ~~get~~

(iii) **getClass()**:- This method gives an object that contains the name of a class to which an object belongs

(iv) **hashCode()**:- This method returns hash code number of an object.

(v) notify() :- This method sends a notification to a thread which is waiting for an object.

(vi) notifyAll() :- This method sends a notification for all waiting threads for the object.

(vii) wait() :- This method causes a thread to wait till notification is received from a notify() or notifyAll() methods.

(viii) finalize() :- This method is called by the garbage collector when object is removed from memory.

Interface

→ An interface contains only abstract methods which are all incomplete methods

→ so we can't create an object to interface but we can create reference variable.

→ In this case, we can create separate classes where we can implement all the methods of the interface. These class are called implementation classes.

→ So implementation classes will have all method with body, it is possible to create objects to the implementation classes.

Syntax:- Declaration of interface.

```
interface interfacename
{
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
}
```

Example:

```
interface A
{
    void sum();
    void mul();
}
```

→ Using the keyword implements, you can implement any number of interfaces.

→ The methods in interface are abstract by default.

→ The variables in interface are final by default.

→ Using class we can implement the interface.

Syntax for implementation:

```
class classname implements interfacename
{
    // class body.
}
```

Eg:

```
class B implements A
{
    void sum()
    {
        ≡
    }
    void mul()
    {
        ≡
    }
}
```

Difference between an abstract class and interface

Abstract class	Interface.
(1) An abstract class contains some abstract methods and also some concrete methods.	(1) An interface contains only abstract methods.
(2) An abstract class can contain instance variable also	(2) An interface can't contain instance variables. If contains only constants.
(3) Abstract class can extends by class	(3) It can be implement by class.
(4) we can extends only one abstract class at a time	(4) we can implement more than one interface at a time.
(5)	

Example of interface:

```
interface A
{ void sum(int a, int b);
  void power(int x, int y);
}

class B implements A
{ void sum(int a, int b)
  { int sum = a+b;
    Sop(sum);
  }
  void power(int x, int y)
  { for(int i=1; i<=y; i++)
    { int p = p*x;
    }
    Sop(P);
  }
}

class C
{ psvm(S zL)
  { A a1;
    B b1 = new B();
    a1 = b1;
    a1.sum(10, 30);
  } a1.power(2, 3);
```

→ one interface can extends another interface.

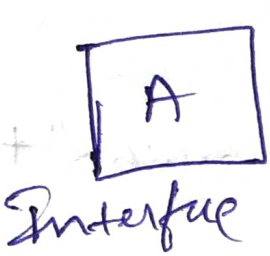
Eg:

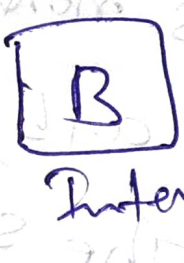
```
interface A
{
    void sum();
    void sub();
}

interface B extends A
{
    void mul();
}

class C implements B
{
    void sum()
    { int a, b;
      Scanner s=new Scanner(System.in);
      a=s.next
      sop("Enter" a"value");
      a=s.nextInt();
      b=s.nextInt();
      int s=a+b;
      sop("The sum of two number"+s);
    }
    void sub(){ ≡ }
    void mul(){ ≡ }
}

class D
{
    psvm(S ☰ L)
    {
        A a1;
        B b1
        C c1=new C();
```

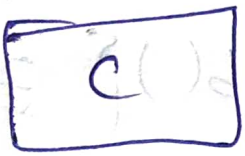
```
a1 = c1;
a1.sum();
a1.sub();
a1.mul(); // Not possible.

b1 = c1;
b1.sum();
b1.sub();
b1.mul();

c1.sum()
c1.sub()
c1.mul()
```

Multiple Inheritance using interface

→ we can implement more than one interface at a time is called multiple inheritance







class C implements A, B

{

}

Eg:-

```
interface A
{
    void show();
}
interface B
{
    void show();  void display();
}
class C implements A, B
{
    int a=10, b=20;
    void show()
    {
        sop(a);
        sop(b);
    }
    void display()
    {
        sop(a*a);
        sop(b*b);
    }
}
class D
{
    psvm(String[] args)
    {
        A a1;
        C c1=new C();
        a1=c1;
        a1.show();
        a1.display();//
        B b1;
        b1=c1;
        b1.display();
    }
}
```

Eg2:

```
interface A
{
    void show();
}
interface B
{
    void show();
}
class C implements A, B
{
    int a, b;
    void show()
    {
        sop(a);
        sop(b);
    }
}
class D
{
    psvm(S ZL)
    {
        A a1;
        C c1 = new C();
        a1 = c1;
        a1.show();
        B b1;
        b1 = c1;
        b1.show();
    }
```