20/12/18

Unit-III

⇒ Inheritence :-

Creating a class from another class is known as inheritence. Inheritence provide the idea of reuseability and also by using inheritence we can reduce the size of the program.

* Types of inheritence

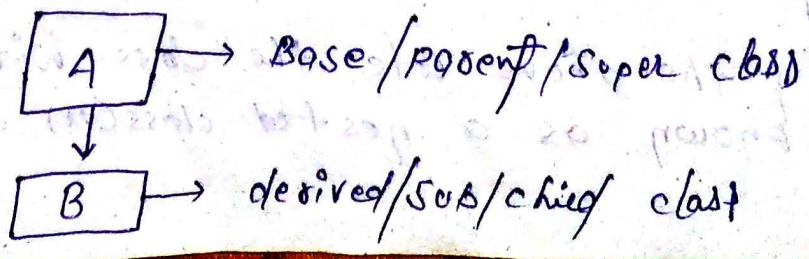
The inheritence can be classified into 5 types:-

① Single inheritence :-

if a single class is derived from an existing class is known as single inheritence. The existing class is known as base class (or) Super class (or) parent class and newly created class is known as derived class (or) Sub class (or) child class. This can be represented as :-

Syntax :-

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



in java the inheritence can be done by using the keyboards extends. This can be represented by above syntax.

In the above declaration first we create a class A and then by using that class we created a class B with the help of extend keyboard. Now we inheritate the class B from class A The newly created class B contains the data that is defined in the class A and the new data that is return in class B.

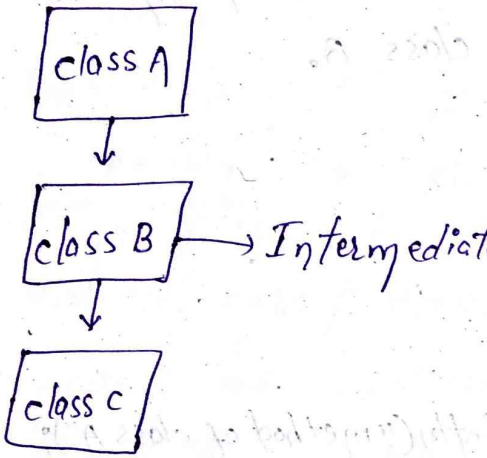
eg: program

```
class A
{
  void display()
  {
   System.out.println("method of class A");
  }
}
class B extends A
  {
   void show()
   {
    System.out.println("method of class B");
   }
}
class single
{
  Public static void main(string args[])
  {
   B b = new B();
       b.display();
       b.show();
  }
}
```

o/p: method of class A
method of class B.

② multilevel inheritance :-

In multilevel inheritance we inheritate class B from class A again we inheritate a class C from class B. Here the class B is act as a derived class for class A and the same class B act as Base class for class C, because of this reason the class B is known as intermediatory class.
A multilevel inheritance can be represented as:-



✓ eg: program

```
class A
{
  void display()
  {
    System.out.println("method of class A");
  }
}
class B extends A
{
  void show()
  {
    system.out.println("method of class B");
  }
}
class c extends B
{
```

```
    void ssc()
    {
      System.out.println("method of class c");
    }
}
class multi
{
  Public static void main(String args[])
  {
    c  s=new c();
       s.display();
       s.showc();
       s.ssc();
  }
}
```

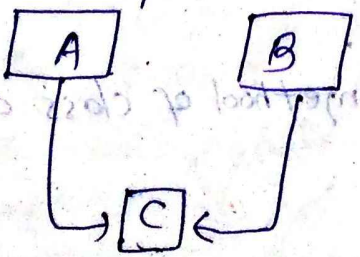
O/P:- method of class A
method of class B
method of class C

③ multiple inheritance:-

If we create one child class from more than one base class is known as multiple inheritance.

Java does not support the multiple inheritance concept directly with the help of interface concept we can implement the M.I in Java.

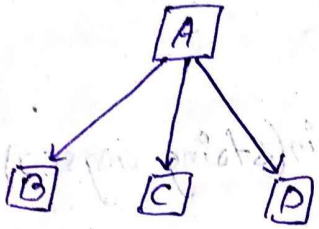
A M.I can be represented as:-



22/12/15

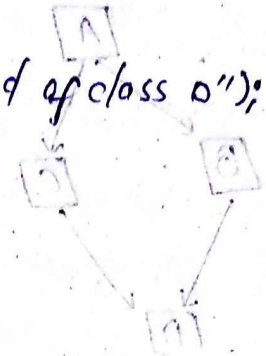
(4) Hierarchical inheritance :-

if we create more than one child class from a single base class that type of inheritance is known as hierarchical inheritance. This can be represented as —



eg:-
```
class A
{
  void methodA()
  {
   System.out.println("method of class A");
  }
}
class B extends A
{
  void method B()
  {
   System.out.println("method of class B");
  }
}
class C extends A
{
  void method C()
  {
   System.out.println("method of class C");
  }
}
```

```
class D extends A
{
  void method D()
  {
   System.out.println("method of class D");
  }
}
class Hierarchical
{
  Public static Void main(string args[])
  {
   B ss = new B();
   c sh = new c();
   D sn = new D();
   ss.method of A();
   sh.method of B();
   sn.method of D();
  }
}
```



o/p:
method of class a
method of class c
method of class d

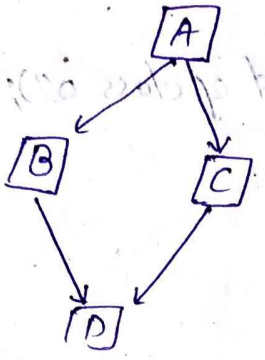
⑤ Hybrid inheritence:-

The combination of multiple & multilevel inheritence is known as hybrid inheritence.

As the hybrid inheritence contains multiple inheritence it is not possible to directly implement the code of the hybrid inheritance

we develop the ~~the~~ hybrid inheritance indirectly by using the concept of interface.

A hybrid inheritance can be represented as—



⇒ Super keyword

if we create an object for a super class we can access only super class members, but if we create an object to a sub class we can access both super class and sub class members.

if the super class and the sub class having the variable with the same name then if we create an object for a sub class, we can only access the sub class members not the super class members. To overcome this problem we can use super keyword.

By using the super keyword in a sub class we can access the both super class and the sub-class members.

eg:
```
class one
 {
   int i=10;
   void show()
   {
    System.out.println("Super class method :"+i);
   }
 }
```

```
class two extends one
{
  int i = 20;
  void show()
  {
   System.out.println("Subclass method i:"+i);
   super.show();
   system.out.println("super. i="+super.i);
  }
}
class ss
{
  public static void main(String args[])
  {
   Two t=new Two();
   t.show();
  }
}
```

O/P:

Subclass method
i=20
Superclass method
i=10
Super.i=10

⇒ **final keyword**

→ A final keyword can be used to declare constant values.

final double PI=3.14;

→ A final keyword can be used to prevent the inheritance concept means, if we declare the final keywords before the class that class can

not be inherited. This can be represented as-

Syntax:-

```
final class A
{
  =
}
class B extends A
{
  =
}
```

26/12/2018

⇒ method overriding

if the sub class have the same method that was declared in a super class means if a subclass and a super class having the same method name and the same parameter is known as method over riding.

The rules for creating a method overriding is-

(i) A method name must be the same in both super and sub classes.

(ii) A method must have the same parameters in both super and sub class.

(iii) The class must be in inheritance relationship.

eg: program:-

```
class A
{
void show()
{
System.out.println("class A");
}
}
```

```
class B extends A
{
  void show()
  {
   System.out.println("class B");
  }
}
class over
{
  Public  static void main(String args[])
  {
   B  ob= new ob();
      ob.show();
  }
}
```

o/p:-

class B

⇒ Abstract class

An abstract class contains zero (or) more abstract method by using the keyboard abstract before a class we can declare an abstract class similarly if we declare an abstract keyboard before the method name that method is known as Abstract method an abstract method does not contains any body so, we can't implement the abstract method in abstract class.

if we want to implement an abstract method we have to derive a seperate class from abstract class then only we can implement the body of the abstract method in the derive

class.

This can be represented as:-

```
abstract myclass
{
 abstract void calculate(double x)
}
class sub extends myclass
{
 void double calculate(double x)
 {
 }
}
```

eg: program:-

```
abstract class myclass
{
 abstract void calculate(double x);
}
class sub1 extends myclass
{
public void double calculate(double x)
 {
  System.out.println("square:"+(x*x));
 }
}
class sub2 extends myclass
{
public void calculate(double x)
 {
  System.out.println("square root:"+math.sqrt(x));
 }
}
```

```
class sub3 extends myclass
{
public void calculate(double x)
    {
    System.out.println("cube:"+(x*x*x));
    }
}
class demo
{
  Public static void main(string args[])
  {
    Sub1 obj1 = new sub1();
    Sub2 obj2 = new sub2();
    Sub3 obj3 = new sub3();
    obj1.calculate(5);
    obj2.calculate(36);
    obj3.calculate(2);
  }
}
```

o/p:- 25
6
8

⇒ Interface

All the methods declared in a interface is either Public (or) abstract by default. An interface Contains abstract methods which are incomplete methods, so it is not possible to create an object for an interface. we can create a sepeate class in which we can implement all the methods of interface these classes can be known as implementation classes.

Since the implementation class we have the method body. we can create an object to the implementation class.

Program:

```
eg: Interface myinter
    {
      void connect();
      void disconnect();
    }
    class ss implements myInter
    {
public void connect()
      {
        system.out.println("connected");
      }
public void disconnect()
      {
        system.out.println("disconnect");
      }
    }
    class demo
    {
      Public static void main(Sting args[])
      {
        ss B=new ssc();
           B.connect();
           B.disconnect();
      }
    }
```

o/p: Connect
disconnect

⇒ Implementation of multiple inheritance using Interface.

program:-

```
//
  Interface father
  {
double float ht=6.2;
    void height();
  }
  Interface mother
  {
double float ht=5.8;
    void height();
  }
  class child Implements father, mother
  {
public void height()
    {
double float ht=(father.ht + mother.ht)/2;
      System.out.println("child height:"+ht);
    }
  }
  class multiple
  {
    Public static void main (string args[])
    {
      child ch=new child();
      ch.height();
    }
  }
```

o/p:- child height:6

$\frac{6.2+5.8}{2}=6$

In multiple inheritance a subclass is derived from multiple superclasses. if two superclasses having the same name for their members (variable & method) they this members inherited into the Sub class is the main confusion in multiple inheritance. this is the reason why Java doesn't support the concept of multiple inheritance directly.

To achieve the multiple inheritance concept in Java we have to use the interface. by using the interface concept we can implement the multiple inheritance in Java.

⇒ Implementation of hybrid inheritance program:-

```
Interface A
{
  void methodA();
}
interface B extends A
{
  void methodB();
}
Interface C extends A
{
  void methodC();
}
class D Implement B,C
{
  void methodA();
  {
    System.out.println("methods A");
  }
```

```
void method B()
{
System.out.println("method B");
}
void method C()
{
System.out.println("method C");
}
Public static void main(String args[])
{
D obj1 = new D();
obj1.method A();
obj1.method B();
obj1.method C();
}
}
```

o/p:- method a
method b
method c

⇒ Difference b/w abstract class and Interface

Abstract class	Interface
(i) An abstract class can be declared by using the keyboard abstract.	(i) An interface can be declared by using the keyboard interface.
(ii) An abstract class is written when they are some common features shared by all the objects	(ii) An interface is written when all the features are implemented differently in different objects.
(iii) When an abstract class is written, it is the responsibility of the programmer to provide sub class.	(iii) When an interface is written the programmer can leave the implementation to third party vendors.
(iv) An abstract class contains abstract methods	(iv) The methods in interface are abstract method by default
(v) An Abstract class contains an instance variable	(v) An interface contains constant values.

(vi) All the abstract method of abstract class can be implemented in it's sub-classes

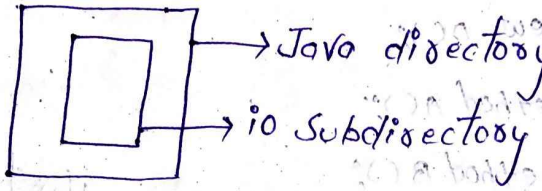
(vi) All the methods of interface should be implemented in it implementation classes

⇒ Package:-

→ A package represents a directory that contents of a group of classes and interfaces.

eg:- consider the following packages-

```
import java.io.*
```



→ In the above package Java is the name of the directory and io is the subdirectory. The * indicates all the classes and interfaces of the sub directory.

* Types of packages

→ A packages can be classified into two types -

① built-in-packages

② user-defined packages

① Built in packages:-

→ The package that are already available in java is known as built in packages.

→ These packages provide all the necessary classes an interfaces, that are essential for developing a java program.

Some of the built-in-packages are-

① Java.lang:- lang stands for language, this package contents classes and interfaces that are essential to develop a program.

② Java.util:- util stands for utility this package also contents the classes and the interfaces.

③ Java.io:-

→ io stands for input and output, this package contents a stream.

→ A stream represents a flow of data from one place to another.

④ Java.awt:-

awt stands for abstract window tool, this package is useful to develop graphical user interface (GUI). Such as buttons, colorfull, screamns and image.

⑤ Java x.swing:-

This is also useful to develop graphical user interface application and it is an extension to java.awt package.

⑥ Java.applet:-

This package is used for designing applet program.

⑦ Java.net:-

Net stands for network client server programming can be done by using this package.

⑧ Java.sql:-

sql stands for structure query language, this package can be used to connect with the data base.

② User-defined package:-

→ if the user created a package on it's own, that type of package can be known as user defined package.

Creating a package:-

A package can be created by using the following

Syntax:-

```
package packagename;
```

eg:
```
package mypack;
```
→ In the above syntax package is the key word followed by the name of the package.

by using the above syntax, we can create a package, with a name mypack.

eg:
```
Package mypack;
class simple
{
Public static void main (String args[])
{
System.out.println ("welcome to package");
}
}
```
o/p:
welcome to package

→ for compiling the above program, we can use the following statement.

```
Javac -d.simple.Java
```

→ In the above statement -d indicates a seperates directory is created and .Operator after -d indicates the directory is created in the current directory.

→ for running the above program, we have to write the following statement.

```
Java mypack.simple
```

* using a package:-

→ for using a package first we have to create a package in the below example, we create a package with a name pack in the package.

→ we implement a code for finding the addition of two number this code can be represented as-

eg:
```
package pack;
class addition
{
 int d1, d2;
 addition (int a, int b)
 {
  d1=a;
  d2=b;
 }
 void show()
 {
 System.out.println("Sum:"+(d1+d2));
 }
}
class use
{
 Public static void main (string args[])
 {
  Pack.addition obj=new pack addition (10,20);
  obj.show();
 }
}
```
o/p:-
= Sum:30

→ After creating a package we can import, that package for later use, this can be represented as-

```
import pack.addition;
class use
{
 Public static void main (string args[])
 {
  addition obj=new addition (30,60);
      obj.show();
 }
}
```
o/p:- Sum: 70.

⇒ Exception Handling :-

→ An exception is a run time error.
→ An exception handling mechanism contents the following blocks.

① Try :-

→ A try block contents set of statements in which the possibilities of exceptions.
→ The syntax of try block is-

```
try
{
  statements;
}
```

② catch :-

→ A catch block can be used to display the exceptions details

OR

→ A catch block can be used to catch the exception that are arises in the try blocks.
→ The syntax of catch block is-

```
catch (Exceptions e)
{
  statements;
}
```

③ finally :-

→ The statements that are written in the finally block are executed whether there is an exception or not.
→ The syntax of a finally block is-

```
finally
{
  statements;
}
```

eg:-
```
class ex
{
 Public static void main (String args[])
 {
  try
  {
  System.out.println ("open files");
  int n = args.length;
  System.out.println ("n = "+n);
  int a = 45/n;
  System.out.println ("a = "+a);
  }
  catch (Arithmetic exception ae)
  {
  System.out.println (ae);
  System.out.println ("please pass the data while
                       running this program");
  }
  finally
  {
  System.out.println("close the file");
  }
}
```
o/p:-

* use of throws :-

→ if the programmer does not want to handle the run time exception in a program by using the throws class, we can permanently throws the run time exception from the program.

eg:-
```
import java.io.*;
class simple
{
private string name;
void accept() throws Io Exception
{
 Buffered Reader br=new Buffered Reader(new input
                    Stream Reader(System.in));
 System.out.println("enter name");
 name = br.read line();
}
void display()
{
System.out.println("name:"+name);
}
}
class ex1
{
Public static void main(string args[]) throws Io
                                        Exception.
{
 Sample S=new Sample();
 S.accept();
 S.display();
}
}
```
o/p:-

* Throw closure:-

→ The through closure is useful to through the exception from the program and later catch the exception by using catch block.

* user defined Exception:-

→ If the user created an exception in a program on it's own that type of exception can be known as user defined exception.

eg: class myException extends exception.

```
{
private static int acc no[] = {1001, 1002, 1003, 1004,
                               1005};
private static string name[] = {"Raju", "Ram", "mohan",
                                "Lalu"};
private static double bal[] = {10,00000, 12,000.00, 5600.00,
                               999.00, 1100, 55};
my exception()
{
}
my exception(string str);
{
super(str);
}
Public static void main(string args[])
{
  try
  {
  System.out.println("Acc no." + "\t" + "customer" + "\t" +
                     "BALANCE");
  for(int i=0; i<5; i++)
  {
  System.out.println("acc[i]" + "\t" + name[i] + "\t" +
                     bal[i]);
  if (bal[i] <1000)
  {
  myException me = new myException("Bal amount
                                    is less");
  throw me;
  }
  }
}
catch (myException me)
{
  me.printStackTrace();
  }
}
}
```

Output:-

Acc no.	CUSTOMER	BALANCE
1001	Raju	10000.00
1002	Rama	12000.00
1003	mohan	56.00,00
1004	LaLU	999.00
1005	Ravi	1100.55

⇒ Assertion:-

→ An assertion is a statement in java, it can be done to test the errors in the program.

→ while executing an assertion first we have to enable the assertion.

→ This can be done by using the following statement

```
Java -ea ass;
```

→ In the above statement ea stands for enable, which is used for enabling the assertion.

→ The syntax for asserting is as—

```
assert expression
```

→ The main advantage of an assertion is to detect and correct the errors in a program.

→ It is mainly used for testing purpose.

eg:
```
import java.util.scanner;
class ass
{
Public static void main(Strings args[])
{
Scanned sc=new scanner(system.in);
system.out.println("enter range");
int value =sc.next(nte);
assert value >18;
System.out.println("value is"+value);
}
}
```

o/p:-