16/07/2013

PAGE NO.
DATE :

# INHERITANCE

⇒ It is a process of extending one class from another class and adding up some additional functionality with in child class is known as inheritance.

⇒ It is a process of creating a new object from existing object and adding up some additional features and behaviours with in new object is known as inheritance.

Object are made of two types in real life :-

(i) is a .... → non-blood is a...
                → Blood is a...
↓
It is created by alone i.e without mixing
for eg:- bread creation with flour only.

(ii) has a .... → strong (composed of)
                → weak (aggregation)
↓
It is created by mixing diff. object
for eg:- CPU = ram + motherboard.

⇒ In java →

is a .... inheritance concept is used
↓↓
Blood is a... class inheritance
Non blood is a.... interface inheritance.
↓↓
this relationship only provides name but do not get features & behaviour of parent class to the child class.

has a.... nested class concept is used.
↓↓
Strong (composed of) Non static nested class is used.
Weak (Aggregation) Static nested class is used.

PAGE NO.
DATE :

Syntax:- It uses the Keyword "extends"

→ Using extends Keyword we can derive one class from another class.

⇒ # Implementation of blood is a relation using class relation.

For eg:-
```
class X
{
}
class Y extends X
{
}
```

# Types of Inheritance.

(i) Single level
(ii) Multi level
(iii) Hierarchical inheritance level.

Note:- Multiple inheritance is not allowed in class inheritance but allowed in interface inheritance.

(i) Single level

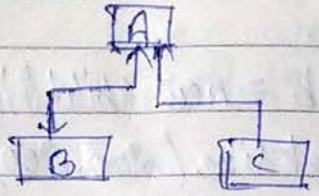
```
class X
{
}
class Y extends X
{
}
```


(ii) Multi level

```
class Z extends Y
{
}
```

PAGE NO.
DATE :

(iii) Hierarchical




For eg:-

```
class Parent
 {
   public void talk()
    {
     System.out.println("parent can spk in hindi");
    }
 }
class Child extends Parent
 {
 }
class Test
 {
   public static void main(String args[])
    {
      Child ob = new Child();
       ob.talk();
      }
 }
```

To create Window Now,

```
import java.awt.*;
class Child extends java.awt.Frame
 { Button b;
   Child()
    { b = new Button("OK");
      setLayout(new java.awt.FlowLayout());
      add(b);
    }
 P.T.O.
```

PAGE NO.
DATE :

```
class Test
{ public static void main (String args[]).
  {
    Child ob = new Child();
      ob.setSize(200,200);
      ob.setVisible(true);
    }
}
```

→ It should be used in constructor above.

If we create two object as,

```
Child ob1 = new Child();
Child ob2 = new Child();
ob2.setLocation(50,50);
ob1.setLocation(150,150);
```

→ It creates two windows.

## # Method Overriding:

⇒ If child class pre-define parent behaviour/method then, it results in method overriding.

⇒ If child class redefine parent class method with same prototype, then it is known as method overriding.

for eg:-

```
class Parent
{   public int sum(int x, int y) {return(x+y);}
    public void talk()
    {
    System.out.println("parent can spk in hindi");
    }
}
class Child extends Parent
{ public int sum(int x, int y, int z)
    {return(x+y+z);}.
```

while over-riding take following precautions:-

(1) Do not change return type of method within child class.
(2) Donot change arguments datatype.
(3) Do not assign weaker access specifier (not - public.

PAGE NO.
DATE :

```
public void talk().
{
  System.out.println("child can spk in English");
}
}
class Test1
{
public static void main(String args[])
  {
    Child ob2 = new Child();
    ob2.talk(); → (of class, the void talk())
  System.out.println(ob2.sum(2,3)); of class child.
  }
}
```

(i) public method of parent can't be overriden as private method in child.
(ii) Protected method of parent can't be - private in child.
(iii) No modifier as private in child is not allowed.
(iv) No modifier as protected in child is not allowed.
(v) static method can't be overriden as non-static method.
(vi) Non static method - static method

Note:- The above same concept of sum calling in C++ creates method overriding in C++ but not in Java because in C++, it depends on the name but in Java it depends on whole signature i.e. prototype.

Here, if we create method below in place of int sum(int x, int y, int z)

```
public float sum(float x, float y)
{
  return(x+y);
}
```

This also do not show over-riding because no same signature is used.

* Same signature does not include access modifier and Return type of methods.

PAGE NO.
DATE :

* One New thing ⇒ We can restrict the method overriding concept in Java for the same ⇒ use final keyword with parent class method.

```
class Parent
{
  public final void talk()
    {
      System.out.println("——");
    }
}
class Child extends Parent
{
  public void talk()
    {
      ——
    }
}
class Test1
{
  public static void main(String args[])
    {
      Child ob2 = new Child();
      ob2.talk();
    }
}
```

Here the inheritance is multilevel because Child is derived from Parent & Parent is derived from Object class

The method of same return type was introduced in Java 1.5.
It is introduced in Java 5.0.

PAGE NO.
DATE :

⇒ We can also restrict the concept of inheritance in Java
for the same → we final Keyword with the class.

for eg:-

```
final class Parent
{

}
```

Now Parent can't be helpful for inheritance.

## # Use of Keyword "Super"

⇒ It is useful only in child class.
⇒ Within child class, we can use it only within methods.
⇒ Within child class, super keyword identify/represents his immediate parent.
⇒ With Super we can use Parent class instance variable, Parent class methodes, Parent class constructer.

```
Super.x;      → instance variable
Super.show(); → instance methods
super();      → constructer
```

⇒ If the above variable, methods & constructor are derived, then only they can be used.
⇒ It is not used with child reference variable

```
class Parent
{
    same as above
}
```

this.super.talk(); ← not allowed.

PAGE NO.
DATE :

```
class Child extends Parent
  {
    public void talk()
      {
        System.out.println("child can speak in Eng");
        super.talk(); → It calls the talk() method of class Parent
    // talk() → It creates recursion here because it calls the same method of child class
      }
  }
class Test2
  {  same as above
      }
```

As instance variable.

```
class Parent
{ public int money = 8000;
    public void show()
      {
        System.out.println("parent money" + money);
      }
  }
class Child extends Parent
  {
    public int money = 9000; // int. var. overriding
  public void show()
    {
      System.out.println("child money " + money);
      System.out.println("child money " + super.money + money);
```

PAGE NO.
DATE

```
}}
class Test2
{
    public static void main(String args[])
    {
        Child obj = new Child();
        obj.show();
    }
}
```

```
of. in show()
{ int money = 5600;
    System.out.println(money); → 5600
    System.out.println(this.money); → 9000
    System.out.println(super.money); → 8000
```

* We cannot use super with in Child class static method.
* We cannot use super with in Child class static methods and with in static Block.
* We cannot use final with static method and we can not override static final method within child class.
* If parent having static var. in parent then we can have non static var. as well as with the same name in child class.
* We can't override static behaviour.

PAGE NO.
DATE :

```
Class A
{
  public static int money = 900;
  public static void talk()
    {
    System.out.println("hindi...");
     }
}
Class B extends A
  {
   public static int money = 88;
   public static void talk()
     {  A.talk();
      System.out.println("eng..");
       System.out.println(money);
       System.out.println(A.money);
      }
  }
Class Test
  {
   public static void main(String args[])
    {
       B ob = new B();
        ob.talk();
         }
      }
```

PAGE NO.
DATE :

# Child class object creation Process

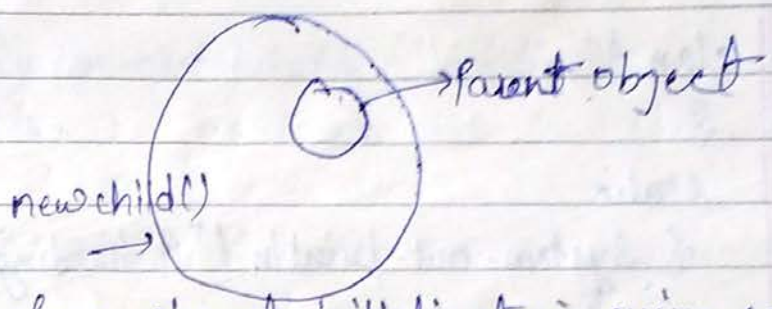
⇒ When we create object of child class then Java also create object of all possible ~~para~~ object of parent class.

```
class A
{
static
{ System.out.println("A loading...");
}
}
class B extends A
{
static
{
System.out.println("B loading...");
}
}
class Test9
{
public static void main(String arg[])
{
B ob = new B();
}
}.
```

1) When we create object of child class then Java JVM looks for meta info. of his parent class in class area. If parent meta info. is not present in class area. then JVM will load parent class in memory then process is applied to child class.

PAGE NO.
DATE :

⇒ After getting meta info. of both classes then jvm will create the object of parent and object of child in heap and with in the object of child jvm will put ref. of parent object.




⇒ Now perform object initialization using constructor.

⇒ At first jvm will invoke cont. of child but before performing anything for child jvm will call default constructor of parent class from child constructor in nested manner.

for eg.

```
class Parent
 {
  static
   { System.out.println("Parent loading...");
   }
  Parent()                    // Parent(int c)
   { System.out.println("parent cons...");
   }
 }
class child extends Parent
 {
  static
   { System.out.println("child loaded..");
   }
```

Javap Parent ↵

PAGE NO.
DATE :

```
Child()  super();          //super(3);
{ System.out.println("child const...");
  }
}
class Test9
{
 public static void main(String args[])
   {
      new Child();
   }
}
```

Output

```
Parent loading...
child load...
parent const...
child const...
```

```
class Test9
{
 public static void main(String args[])
  {
     Parent ob = new Parent(44);
     try { Thread.sleep(3000); }
     catch(InterruptedException e) { }
     Child obl = new Child();
  }
}
```

create delay of 3 sec.

Make the above main function in class Parent & then check the loading concept.

PAGE NO.
DATE :

Note :-

In constructor calling, by default, default constructor of parent class is encountered.

If default constructor is not found in the Parent class, then use of super keyword is must.

⇒ super must be the first statement in child class constructor.

⇒ Constructor chaining i.e. this(), & super both can't be used in combination so we have to ignore any one of them. Generally constructor chaining is ignored.

⇒ If we make private constructor of Parent class then it can't be derived in child class from such a parent.

```
class Parent
{ private Parent()
  {
  }
  public Parent(int x)
  { this();
  } }
class child extends Parent
{
  public child()
  { super(50);
  }
}
class Test3
{ public static void main(String args[])
  { child obj = new child();
  }
}
```

PAGE NO.
DATE :

⇒ In constructer there is no any concept of constructer over-riding..

This is not parameterized const. but a simple method with arguments.

```
class child
{
    public void child.(int x)
    {
    }
}
```

Special case:-

```
class Parent
{
    public static void main(String args[])
    {
        System.out.println("parent class main...");
    }
}
class child extends Parent
{
    public static void main(String args[])
    {
        String name[]={"aaa","bbb","ccc"};
        Parent.main(name);
        System.out.println("child class main...");
    }
}
```

Java child;

Java Parent;

DATE :

```
class Parent
{
  public int sum(int x, int y)
  {
  }
}
class Child extends Parent
{
  public float sum(int x, int y)
  {
  }
}.
```

same method with different return type.
In JDK 1.5 or less, the above two methods don't allow overriding.

⇒ In JDK 5.0, the return type should be compatible but not necessary to be same to allow method overriding.

Q How to Acheive polymorphism by inheritance

Means :- feature + behaviour + name.

⇒ or, Object of child can be referenced by any of his parent class.

⇒ when we refer to child class object via the reference variable of parent class then

⇒ Parent can pick only those details of child which are provided by parent to his child class.

for eg:-

```
class Parent
{
  public void bike()
  { System.out.println("bike with non alloy wheels");
```

PAGE NO.
DATE :

```
class Child extends Parent
{  public void sportsBike()
   {  }  }
class Test
{
  public static void main(String args[])
  {
     Parent obj = new Child();
     obj.bike();
     obj.sportsBike();
  }}
```

upcasting

→ This line gives error because. sportsBike is the method of Child class. & we have reference of Parent class.

In place of Parent obj = new Child(); we write Child obj = new Child(); then the statement obj.sportsBike() allows.

⇒ Process of assigning reference of child class object to parent class reference variable is also known as upcasting.

```
class Parent if static →
{  public void bike()
   {
   }
}
```

PAGE NO.
DATE :

```
class Child extends Parent
{   If static → then
    public void bike()
    {
    }
}
class Test
{
    public static void main(String args[])
    {
        Parent obj = new Child();
        obj.bike();  → It calls the bike() method of child class due to overloading concept.
    }
}
```

OR, in words.

⇒ If we points to child class object via the reference variable of parent class and if child class override same behaviour of parent class then with parent reference variable we get overriden behaviour of child. (But this rule is not applicable with static method because static methods are not inherited

⇒ (i.e. with parent ref. we do not get overriden behaviour of child. instead java will give parent version of same method.).

So, Now,

public Parent afun (Parent p)

Similarly, it can return ref. of parent as well as its child class.

Now this p of type Parent can receives reference of Parent as well as ref. of its child class.

PAGE NO.
DATE :

Note:- for comparing two string variables, if we write,

```
String name = "Tajmahal";
if (name == "Tajmahal")
{
}
```

It do not compare but it compare their reference.

So, to compare we use,

use for Scanner: `import java.util.*;`

```
String username;
Scanner cin = new Scanner(System.in);
System.out.println("Enter user name");
username = cin.nextLine();
if (username.equals("Arun"))
{ System.out.println("valid user name"); }
else { System.out.println("invalid user"); }
}
}
```

→ `public boolean equals(String a);`

Methods of System's class equals

PAGE NO.
DATE :

⇒ A java class method which can produce object is known as Factory Method.

```
class Shape
{ public int area()
    { System.out.println("Not able to compute area..."); return(0); }
}
class Rect extends Shape
{
  public int area()
  {
    System.out.println("_____");
    return(0); }
  }
class Cube extend Shape
  {
    public int area()
      {
      System.out.println("_____");
        return(0); }
      }
class Test
{
  public static Shape getShape(String shapetype)
  {
    if (shapetype.equals("cube"))
      {return(new Cube()); }
    else if (shapetype.equals("rect"))
      { return(new Rect(); }
    else
      { return(null);
      } }
```

PAGE NO.
DATE :

```
public static void main(String args[])
  {
     String shptype;
     Scanner cin = new Scanner(System.in);
      System.out.println("Enter shp type...");
     shptype = cin.nextLine();
      Shape res = getShape(shptype);
      if (res != null)
         { res.area(); }
     }
}
```

# Concept of Downcasting

```
class Test
{ public static void main(String args[])
  {
     Rect obj = new Rect();
  System.out.println("obj = " + obj);
     Object oref;
     oref = obj;     // upcasting
// Java compiler allows upcasting in default manner
        Rect robj;
        robj = (Rect) oref; // Downcasting
// Java compiler do not allow downcasting in
   implicit manner, to perform downcasting we
   require to use type casting operator of
   destination child type.
   System.out.println("robj = " + robj);
        robj.area(); } }
```

PAGE NO.
DATE :

9f Object oref = new Cube();
Rect obj;
obj = (Rect) oref;

This gives error in runtime.

⇒ It allows at compiletime but gives error in runtime.

⇒ For downcasting, upcasting is necessary.

## ABOUT EQUALS METHOD :-

① It is a built in method of Object class.

② Syntax.

```
public boolean equals(Object o)
    {
    }
```

③ By default, it compares two objects in terms of their reference.

For eg :-

```
class Rect
  {
    private int length;
    private int breadth;
public Rect();
    Rect(int x, int y)
      {
        length = x;
        breadth = y;
      }
```

```
public void show()
  {
    System.out.println(length + "  " + breadth);
  }
}
class Test
  {
    public static void main(String args[])
    {
      Rect obj1 = new Rect(3,5);
      Rect obj2 = new Rect(15,16);
false → System.out.println(obj1 == obj2);
false → System.out.println(obj1.equals(obj2));
    }
  }
```

Both are same thing.

If we include overriding of equals method as:-

Most important Method

```
public boolean equals(Object o) // (Object o = obj2)
  {
    Rect &obj = (Rect) o;
    return (this.length == &obj.length &&
            this.breadth == &obj.breadth);
  }
```

Now,

```
System.out.println(obj1.toString());
System.out.println(obj1);
```

Both are same

PAGE NO.
DATE :

## About tostring() method of Object class.

(1) Built in method of object class.
(2) Try to produce string represent of class object.
(3) Object class implementation of tostring() produce type of object + hash code of object if req. then child class can override tostring() method of Object class.

(4) Syntax

public String tostring().

Overriding of tostring() method.

```
public String tostring()
{
return ("length = " + length + " breadth = " + breadth);
}
System.out.println(obj1);
System.out.println(obj2);
```

# also, return (super.tostring() + "\n length = " + length + "breadth = " + breadth); }

PAGE NO.
DATE :

# ABSTRACT CLASS & ABSTRACT METHOD

⇒ Abstract class is a special kind of class which is used to remove duplicacy of code from n number of similar kind of class.

⇒ We can not create instance of abstract class.

⇒ To create abstract class just put abstract keyword in front of class declared.

for eg:- → Special keyword of Java

```
abstract class Shape
{
    String outlinecolor;
    void fillcolor()
    {
    }
}
class Test
{ public static void main(String args[])
  {
      new Shape(); // gives error.
  }
}
```

⇓

can not be instantiated.

⇓

i.e. Object can't be created.

⇒ We can use abstract class concept to attach a common name to n no of similar kind of classes.

PAGE NO.
DATE :

* Abstract method ⇒ Method without body. (same as C++ virtual int area();)
for eg:- abstract int area();
⇒ We can place abstract method only within abstract class.
⇒ Abstract method is also known as sub class duty. i.e. abstract method must be defined by child class otherwise child class will also treated as abstract class.
⇒ Abstract method is expected to be give definition by child class
⇒ Abstract method is meaningful while refering to child class object via the reference variable of parent class.
* ⇒ Abstract class can have both type (i.e. Abstract as well as non-abstract) methods.
⇒ We cannot apply private access level with Abstract method.
⇒ We also cannot apply final & static with Abstract method
⇒ We cannot have abstract constructor.

Q. Does Abstract class Needs Constructor?

⇒ Yes, because the constructor of Abstract class is used to initialize the initial value to data m. when we create instance of any of the child class of abstract class.

for eg:-

```
abstract class Shape
{ String outlinecolor;
    void fillcolor()
    { System.out.println("filling color..."); }
```

PAGE NO.
DATE :

```
abstract int area();   <- Abstract class
public Shape()              constructor
{ System.out.println("Shape con...");
    outlinecolor = "Red";
        }
}
class Rect extends shape
{
    int area()
    {
      return(0);
        }
    }
class Test
{ public static void main(String args[])
  {
    new Rect();
      }
    }.
```

Special case :-

(1)

```
class shape
  { same as above }
class Alpha extends shape
  {
    public int area()
        { return(0);
            }
    }
class Rect extends Alpha
    { }
```

PAGE NO.
DATE :

class Test
{ some as above }

② abstract class Shape
{
String outlinecolor;
abstract int area();
}
abstract class SolidShape extends Shape
{
abstract void fillshape();
}
class Rect extends SolidShape
{
void fillshape() { }
int area() { return (0); }
}
class Test20
{
public static void main (String args[])
{
}
}

Full example :-

```
abstract class Shape
{   String outlinecolor;
    abstract int area();
    public void show()
    { System.out.println("outlinecolor=" +outlinecolor);
    }
```

PAGE NO.
DATE :

```
public void fillShape()
 {
  System.out.println("filling shape...");
  }
 }
class Rect extends Shape
  {
    int length; int breadth;
    public int area()
     { return (length * breadth);
     public void show()
        {
          super.show();
        System.out.println(length + " " + breadth);
      }
    Rect(int l, int b)
     { length = l;
       breadth = b;
        }
    }
// We can have ref. of abstract class.
   class Test20
   {
     public static void main(String args[])
     {
       Shape ob = new Rect(3,3);
     System.out.println(ob.area());
       ob.show();
        }
     }
```

PAGE NO.
DATE :

# Method Binding

⇒ Binding means Resolving method call.

⇒ When we call any method using ref. variable with '.' operator, jvm will have to search its proper definition

⇒ Method Binding is of two types:-
(i) Static Binding
(ii) Dynamic Binding.

It is the activity of compiler → Static Binding :- means mapping of method call to its def. is resolved during compile time.

It is the activity of JVM → Dynamic Binding :- means mapping of method call to its def. is resolved during runtime.

Some Rules :- The things which can't over-ride are static binding otherwise dynamic binding

⇒ Static Binding is used in below types:-
↳ Class constructor
↳ Static Methods.
↳ calling using super keyword.
↳ Private Method of java class
↳ final method. (because they are not over-ride)

⇒ Rest of above 5 types shows Dynamic Binding.

⇒ Static Binding is better than Dynamic Binding in terms of speed.

PAGE NO.
DATE :

⇒ Abstract method also participate in Dynamic Binding.

⇒ Interface method also participate in Dynamic Binding.

# Static Binding is not flexible than Dynamic Binding because static Binding can't change its behaviour whereas Dynamic can change its behaviour

Assembly:→

⇒ We have four Keywords:—

(i) invoke static
↳ It writes invoke static in byte code of static methods.

(ii) invoke special
↳ It writes invoke special in byte code of const, super and private things

(iii) invoke virtual
↳ It writes invoke virtual in byte code of dynamic Binding methods.

(iv) invoke interface → for interface methods.

for eg:-

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

PAGE NO.
DATE :

```
class B extends A
 {
  public void show()
   {
    System.out.println("B class show");
   }
 }
class Test
 {
  public static void invoke(A ob)
    {
      ob.show();
    }
  public static void main(String args[])
   {
    A obj1 = new A();
    B obj2 = new B();
invoke(obj1);  |  invoke(obj2);
    }
 }
```

This show dynamic binding ⇒

⇒ [javap -c Test.java]

If invoke() method is kept in A class, and private access modifier is used with void show of A class,

```
private void show() { }
public static void invoke(A ob)
      { ob.show();
      }
```

DATE :

Now, in main,
A.invoke(obj1); is used to call the invoke function.

⇒ In static binding, the meta info. of ~~reference~~ object type is used whereas in dynamic binding, the meta info. of reference type is used.

## JAVA INTERFACE (झाँकना)

⇒ Interface in java is way of telling other what minimum they must do but not how to do.

⇒ Syntax wise it looks like a java class.
for eg:-

```
interface Shape
{
    Body of interface
}
```

⇒ Just like class, java compiler create class file for every interface.

⇒ With in interface body, we can have only public members because to achieve polymorphism in 100% manner.

⇒ Interface allows only final static variables.

⇒ Interface allow only abstract methods.

⇒ Within interface we can not include the following :-

PAGE NO.
DATE :

1. Constructor
2. Method with body
3. Non static variable
4. Static Block
5. Anonymous Block.

⇒ We can have nested interface.
⇒ We cannot create object of interface.
⇒ Within java application we can have n number of interfaces.
⇒ Just like class we can place them with in java package.

⇒ Interface represent implementation of non blood is a relationship.

for eg:-

```
interface Shape
{
    int x=90;   int area();
}
```

Use byte code :-

C:\> javap -c Shape ↵

In byte code,

⇒ It don't create constructor
⇒ It automatically allows public, static & final to int x;
⇒ It automatically makes public, abstract to area();
⇒ Interface variable must be initialized at the time of their declaration.

PAGE NO.
DATE :

⇒ Interface variables are just used to create global constants in java.

```
interface Shape
  {
    public int x=80;
       int area();
class interface SolidShape ← Nesting of interface.
        {
          }
       }
```

⇒ class is also allowed inside interface.

⇒ Interface also supports inheritance using 'extends' Keyword.

```
interface Shape
    { int area();
      }
interface SolidShape extends Shape
   {
     void fillShape();
     }
```

⇒ Interface inheritance support multiple inheritance whereas class inheritance do not support multiple inheritance.

```
interface Shape
{   int area(); }
interface Alpha
  {   }
```

```
interface SolidShape extends Shape, Alpha
  {
     void fillShape();
     }.
```

⇒ A java class can provide implementation of interface.

⇒ A single java class can provide implementation of multiple interfaces.

Syntax:-

```
interface Shape
  {
     int area();
     }
interface SolidShape
   {
      void fillShape();
   }
class Rect implements Shape
   {
      }
```

Keywords to implements interface

Shape → name of interface

Note this class must define the methods of interfaces because they are abstract methods & if the class which implements the interfaces methods do not define them it must be abstract class.

PAGE NO.
DATE :

⇒ Always use public Access specifier in front of methods defined in the class which implements interface.

```
interface Shape
  {
     int area();
  }
interface SolidShape
  {
     void fillShape();
  }
class Rect implements Shape, SolidShape
  {
    public int area()
      {
        System.out.println("computing area of rect");
          return(0);
        }
    public void fillShape()
    {
     System.out.println("fill the shape...");
      }
    }
```

Also:

```
class RoundRect
  {
  }
class Rect extends RoundRect implements Shape, SolidShape
```

(extends comes first; No commas)

PAGE NO.
DATE :

⇒ A single java class can also extends other class as well as interface.

⇒ We can create reference variable of interface and it can point to object of such a class which implements same interface.

⇒ In the above class,

```
class Test
{
    public static void main(String args[])
    {
        //new Shape(); → gives error because we can't create object of interface
        Shape s = new Rect();
        s.area();
        s.fillShape(); → gives error
    }
}
```

⇒ Just like class, interface also have user defined data type.

Name of interface → Shape afun()

```
{
}
```

Now, the above method will return ref. of such class which implements such interface.

⇒ Interface can act as method return type.

⇒ interface can act as formal arguments of method

PAGE NO.
DATE :

Also, If in above clas. if interface SolidShape extends Shape. then, in main.

we can have,

```
{ SolidShape s = new Rect();
    s.area();
    s.fillShape();
  }
```

] Both the method of Shape & SolidShape

⇒ MARKER INTERFACE

↳ The interface which do not have any members of its own is known as marker interface.

for eg:-

```
interface A
  { No member.
  }
```

⇒ They are just need to assign multiple name to a java class.

If in above class, if in clas RoundRect.

```
class RoundRect implements SolidShape.
  {
    public void fillShape()
     { System.out.println("filling shape...");
     }
  }.
```

So, in clas Rect, we are not need to define the fillShape() method.

PAGE NO.
DATE :

⇒ Interface can't be private & protected. They can be no access modifier and public.

public interface A
{
}

can be public & No access modifier but can't be private & protected.

Full example:-

```
interface Shape
  { int area(); }
interface SolidShape
   { int fillShape(); }
class RoundRect implements SolidShape
  { public void fillShape()
      { System.out.println("fill shape by round rect..."); }
  }
class Rect extends RoundRect implements SolidShape, Shape
  {
     public void fillShape()
       { System.out.println("fill shape by Rect..."); }
     public int area()
     { System.out.println("area..."); }
  }
}
```

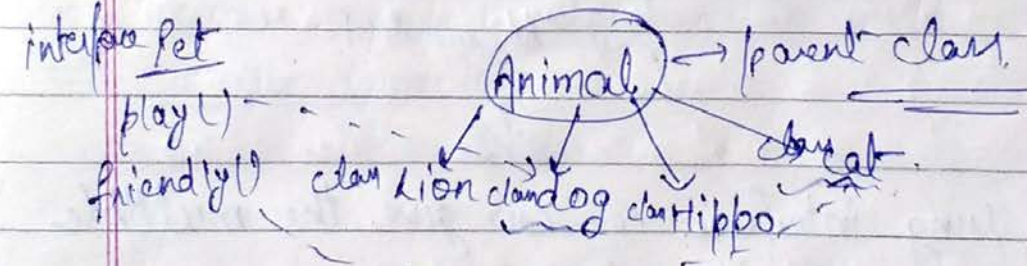
DATE :

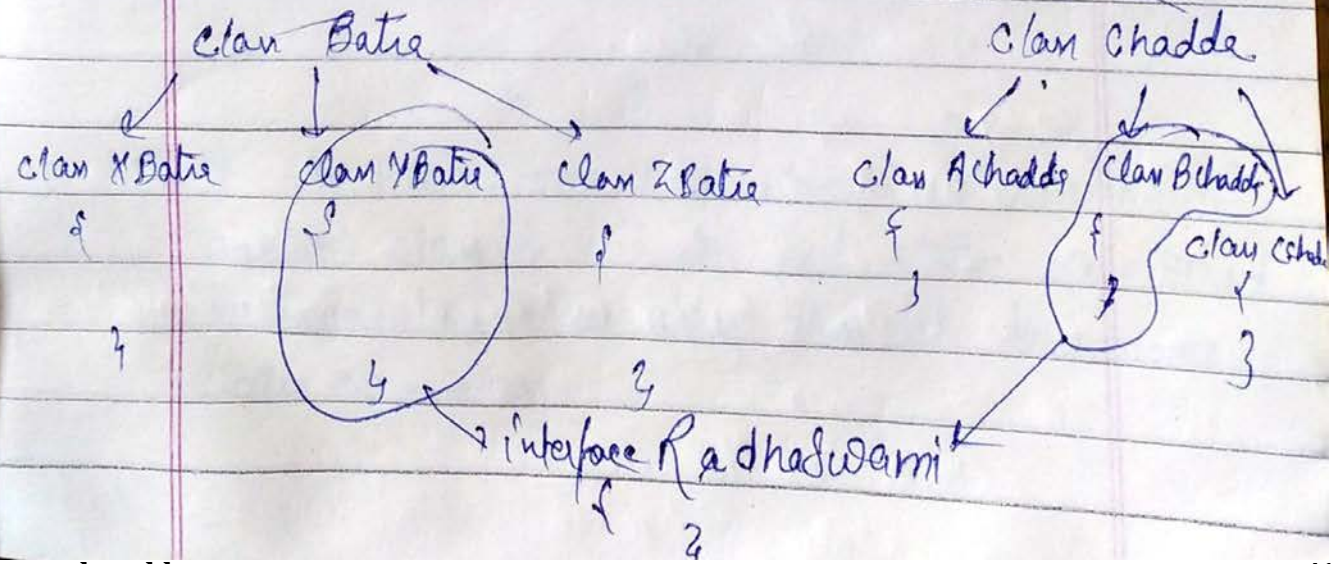
```
class Test
{
    public static void main (String args[])
    {
        Rect obj = new Rect();
        obj.fillShape();
        RoundRect obj1 = new Rect();
        obj1.fillShape();
        SolidShape obj2 = new Rect();
        obj2.fillShape();
    }
}
```

All this will show the fillShape block of Rect class

⇒ We can also use :-

⇒ `System.out.println(SolidShape x);`



PAGE NO.
DATE :

⇒ To assign common names to classes which belongs to different familarity, we can use interface.

```
User A                    interface A              Class utility implements A,B,C
{ A obj = new utility();  { p.v. a compute();      {
                            p.v. b compute();        public void a compute()
  }                       }                        {
                                                     }
User B                    interface B              public void b compute()
{                                                    {
B obj = new utility();    { p.v. c compute();      }
                            p.v. b compute();
  }                       }                        public void c compute()
                                                     {
User C                                               }
{ C obj = new utility();  interface C              public void d compute()
                          {                        {
  }                         p.v. d compute();      }.
                            p.v. b compute();      }.
                          }
```

⇒ Using interface we can give the multiple name to the class.

DATE :

# NESTED CLASSES.

Def :→ If we place one class with in the definition of another class then it results in nesting of class.

Types :- (1) Static inner class.
(Using static inner class we can understand weak has a relation).

(2) Non-static inner class.
→ (Using non-static inner class we can understand strong has a)

(3) Anonymous inner class.
(4) Local inner class.

STATIC INNER CLASS

for eg :-

⇒ Static inner class

```
class X
{
    static class Y
    {
    }
}
```

⇒ If we place one class with in the scope of another class & if we put static keyword in front of inner class then it becomes static inner class.

DATE :

Note:- Java compiler creates class files for each outer & inner class. Naming convention of inner class is Outerclassname $ Innerclassname class as in current e.g.

⇒ Java compiler will create 2 class files.
(1) X class (2) X$Y class.

⇒ We can create object of static inner class without having object of outer class.

⇒ Within static in a class, we can use static data members of outer class.

⇒ static inner class can access, private, public, protected & no-modifier static member of outer class. But reverse is not true.

⇒ If static inner class do not use any of the no. of outer class then at a time of object creation of inner class, Java JVM do not load outer class.

⇒ Syntax of creation of inner static class

→ X.Y obj = new X.Y();

where X is the outer class & Y is the inner static class.

for eg:-

①
```
class X
{
    static { System.out.println("outer class loading"); }

    static class Y
    { static { System.out.println("Inner class loading"); } }
```

DATE :

```
class Test
{
  public static void main(String args[])
    {
      X.Y obj = new X.Y();
    }
}
```

(2)

```
class Outer
{
  private static int data = 90;
  static class Inner
    {
      public void show()
        {
          System.out.println("data = "+data);
          System.out.println("value = "+value);
        }
    }
class Test
  {
    public static void main(String args[])
      {
        Outer.Inner obj = new Outer.Inner();
        obj.show();
      }
  }
```

Non-~~bis~~static values.

private int value = 190

This line gives error because it uses non static member which is not possible.

{ javap -c Outer$Inner / more }

PAGE NO.
DATE :

Note :→ If we have static data member in a class & for using that static data member we should make static method in a class.

Q. ~~Why~~ How it is possible to use private data members in Inner static class?

⇒
```
class Outer
{
    private static int data = 90;

static int access$000()
{
    return (data);
}

static class inner
{ public void show()
{
System.out.println("data = "+(data));
}
}
}
```

To see use javap outer

replace it with

{ Outer.access$000() }

⇒ It is done by Java compiler automatically.

⇒ It is done only for private data member in outer class.

PAGE NO.
DATE :

⇒ We cannot create our own method as,
static void access$000() { }.
while using private data member in inner class because java compiler do not allow this.

⇒ Now, also, we can use the ~~private~~ non-static data member of outer class in inner class by making obj. of outer class in inner class.
For eg:-

```
class Outer
{  public int data=90;
   static class Inner
      {
      public void show()
       {
        Outer ob = new Outer();
   System.out.println("inner show..."+ob.data);
       }
   }
}
```

⇒ We can declare object of inner class in outer class & also use it inside outer class.

⇒ We can also extends (use inheritance) in static inner class.

PAGE NO.
DATE :

```
class Alpha
{ public void aShow()
    { System.out.println("Alpha..show");}
    }
class Outer
  {
    static class Inner extends Alpha
      {
        public void show()
      {
      System.out.println("in show..");
        aShow();
        }
      }
  }
class Child extends Outer.Inner
    {
    }
class Test
  {
  public static void main(String args[])
    {
      Child obj = new Child();
        obj.show();
      }
```

PAGE NO.
DATE :

## Non-Static Inner Class

⇒ If we place a class within the another class & if we do not put static keyword in front of inner class, then it become non-static Inner class.

⇒ Non-static inner class represents strong has a relationship.

⇒ Within non-static Inner class we can use static as well as non-static data member of outer class irrespective of level of abstraction (private, public + protected)

⇒ We can not create object of non-static Inner class without having object of his outer class.

### Syntax of creating object of non-static Inner class

↳ first of all, create object of outer class.

```
Outer outobj = new Outer();
```

↳ Now to create object of inner class, we use

```
Outer.Inner iobj = outobj.new Inner();
```

(outobj: reference variable of outer class; argument)

for eg:-

```
class Outer
{ private int xdata=90;        → non-static data members
  private static int ydata=190;
  class Inner
  { public void show()
    { System.out.println("xdata="+xdata);
      System.out.println("ydata="+ydata);
```

static data member

PAGE NO.
DATE :

```
class Test
{ public static void main(String args[])
  {
    Outer outobj = new Outer();
    Outer.Inner inobj = outobj.new Inner();
    inobj.show();
  }
}
```

Both has a strong has a relationship

Java compiler replaces it as

`new Inner(outobj);`

Non-static Inner class.

↳ It is the class in which the default constructor of Inner class is called by Outer class and also it is parameterized.

⇒ Inner class object has reference of outer class object.

for eg:-

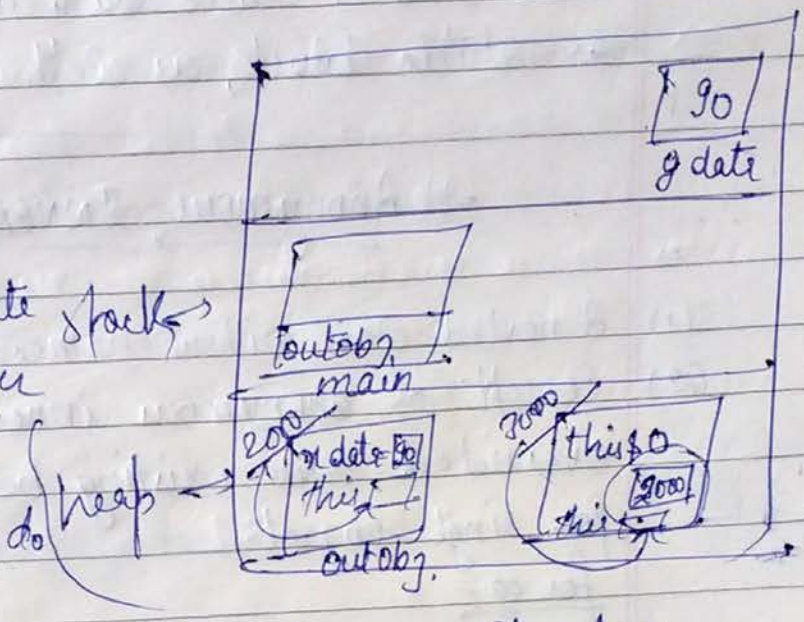
```
class Outer
{ private int data = 90;
  private static int ydata = 190;
  class Inner
  {
    final outer this$0;
    Inner(outer oref)
    { this$0 = oref;
    }
```

created by Java compiler automatically

PAGE NO.
DATE :

```
public void show()
{ System.out.println("xdata = "+ xdata);
  System.out.println("ydata = "+ ydata);
  }
}
}
```

To access the private nonstatic member java compiler automatically do ⇓.




(1) creates a method in ~~inner class~~ Outer class

static int access$00(Outer ref) { return (ref.xdata); } → creates in Outer class

Made static because to create static binding which is fast.

↳ replaces the statement as,

→ System.out.println("xdata = "+ xdata)

↓

Outer.access$00(this$0)

This is done in inner outer

⇒ javap Outer$Inner ↵

javap -c Outer$Inner | more ↵

⇓

To see page wise

PAGE NO.
DATE :

Some Restrictions of Non-static Inner class:-

→ Inner class do not get loaded without outer class.

⇒ In a non-static Inner class we cannot have static data members as well as static block as well as static method.

## # Anonymous INNER CLASS

⇒

(1) A nested class without name. ~~concept of~~.

(2) Concept of anonymous inner class is used to overrided some existing behaviour of class for single object.

for eg:-

```
class Jatcommunity
 { int age = 21;
   public void marigeSystem()
    {
    System.out.println("our custom allow only
                        Arranged mrg..");
    }
 }
class TestJat
  {
   public static void main (String args[])
   {
    Jatcommunity obj1 = new Jatcommunity();
      obj1.marigeSystem();
    Jatcommunity obj2 = new Jatcommunity()
      { public void marigesystem()
```

```
{
System.out.println("M not agree with the condition");
  }
  };
ob2.maritalSystem();
Jatcommunity obj3 = new Jatcommunity()
  { int age = 19;
  public void maritalSystem()
  {
    System.out.println("diff. system..." + age);
  }
  };
obj3.maritalSystem();
  }
```

This is the inner anonymous class.

It prints 19.

=

⇒ We can also have more methods inside inner anonymous class but it can be called inside that block only. It cannot allow at object level.

⇒ On dir↵, it creates class for above example as:-

TestJat.java → for class TestJat
TestJat$01.java → for anonymous inner class
TestJat$02.java → " " " " ,

for eg:-

```
Jatcommunity obj3 = new Jatcommunity()
  { void show()
  { System.out.println("aaa"); }
  public void maritalSystem()
  { show();
    System.out.println("diff.system"); } }
```

diff. methods allow inside this block only.

⇒ We cannot override constructor in this class because it is itself a constructor calling.

## # LOCAL INNER CLASS

⇒ If we define a java class within the scope of java class methods. then that class is known as local inner class.

⇒ It is used rarely in our programming.

for eg:-

```
class Alpha
  {
    public void show()
      {
        class Beta
        { System.out.println("alpha show..");
          public void disp()
            {
          System.out.println("Local class disp...");
            }
        Beta obj = new Beta();
          obj.disp();
        }
      }
class Test
  { public static void main(String args[])
    {
      Alpha ob = new Alpha();
        ob.show();
    } }
```

We cannot create object of Beta class here. ⟵

PAGE NO.
DATE :

⇒ We cannot create instance of local inner class outside java class method.

⇒ Within local inner class, static variables are not allowed.

⇒ Within local inner class, we can have constructor

⇒ Naming convention of Beta class is as:-

Alpha$01Beta.class.

⇒ Within local inner class, we can have static & non-static & private data members of Outer class, irrespective of their level of abstraction.

for eg:-

```
class Person
 {
   public void doublePersonality()
   {
     System.out.println("Vidya talking normally.");
      class Mangulika
       {
          public void talking()
          {
         System.out.println("Me Mangulika...");
          }
       Mangulika ob = new Mangulika();
          ob.talking();
       }
 }
```

PAGE NO.
DATE :

```
class Test
{
  public static void main(String args[])
    {
      Person ob = new Person();
        ob.doublePersonality();
      }
    }
```

Modified Example :->

```
interface Ghost
 {
   public Ghost talking();
     }
class Person
   {
     public Ghost doublePersonality()
     {
     System.out.println("Vidya talking normally...");
     class Manjulika implements Ghost
       {
         public Ghost talking()
        {
         System.out.println("Me Manjulika...");
         return (this); // will return ref. of aGhost
              }                          obj.
        }                    from talking method
     Manjulika aGhost = new Manjulika();
     return (aGhost.talking());
          // will return ref. of aGhost object from
             double Personality method
}}
```

PAGE NO.
DATE :

```
class Baba
{
  public static void main (String args[])
  {
    Person vidya = new Person();
    Ghost gobj = vidya.doublePersonality();
    System.out.println("Akshaya talking to..");
    gobj.talking();
  }
}
```

⇒ **One Concept Releated to Non-static Inner class:→**

This class can't be private → class HumanBeing

```
{
  (private) class Brain
  {
  }
  Brain obj = new Brain();
}
```

→ This class is not accessible outside

→ and object is also created inside the HumanBeing class.

⇒ Such type of class is used when we need a object of such class only within the outer class.

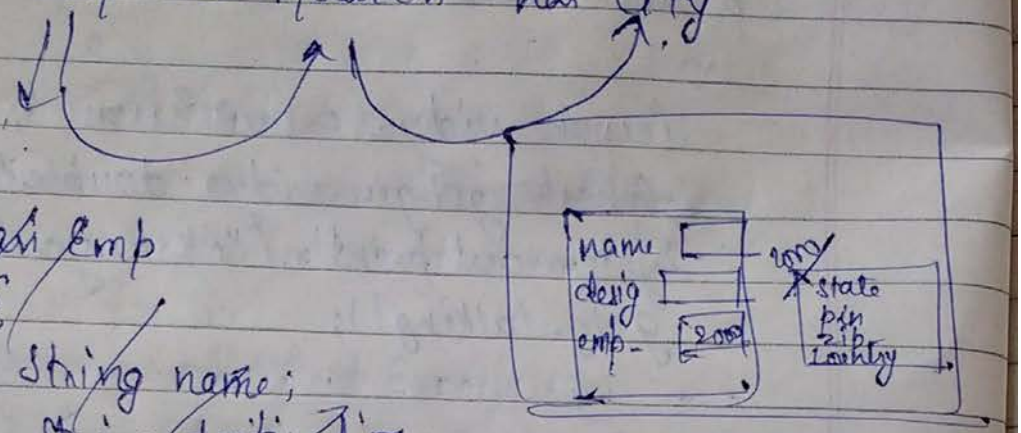
for this case → char ch = (char) System.in.read(); throws java.io.IOException

PAGE NO.
DATE :

# Reference Chaining (Has a Relationship)

↳ chain of dii-similar objects.

for eg:- Emp has Address has City




```
class Emp
{
    String name;
    String designation;

import java.util.*;
class Address
{
 private String city;
 private String state;
 public void getAddress()
    {
      Scanner cin = new Scanner(System.in);
      System.out.println("Enter city");
          city = cin.nextLine();
     System.out.println("Enter state");
          state = cin.nextLine();
    }
public void showAddress()
  {
  System.out.println("city :" + this.city);
  System.out.println("state=" + this.state);
      }
  }
```

PAGE NO.
DATE :

```
clas Author
 {
  private String auth_name;
  private Addres auth_addres = new Addres();
  public void getData()
   { Scanner cin = new Scanner(System.in);
    System.out.println("Enter auth name");
     auth_name = cin.nextLine();
     auth_addres.getAddres();
   }
 public void showData()
   {
    System.out.println("Auth name = "+auth_name);
     auth_addres.showAddres();
      }
   }
clas Emp
 {
  String emp_name;
   float salary;
  Addres emp_add = new Addres();
  public void getData()
  {
   Scanner cin = new Scanner(System.in);
   System.out.print("Enter emp name");
    emp_name = cin.nextLine();
     emp_add.getAddres();
      }
  public void showData()
```

```
{ System.out.println("emp name = "+emp_name);
    emp_add.showAddress();
    }
}
class TestAuth
{
  public static void main(String args[])
  {
    Author ob = new Author();
      ob.getData();
      ob.showData();
      Emp eob = new Emp();
       eob.getData();
       eob.showData();
    }
}
```

# Anonymous Inner Class:-

Java supports two types of anonymous Inner class:-

1) By extending existing class.
2) By implementing existing interface

First case is discussed above with example of Tutcommunity.

In second case,

Steps 1 :- At first create a interface.

```
interface DrawShape
{
  void draw2DShape();
}
```

PAGE NO.
DATE :

Step2 :→ Then create anonymous inner class for above interface. in following manner:-

new <interface Name> ()
{ Definition of anonymous inner class }

eg.

```
class Demo
{ public static void main(String args[])
  {
DrawShape obj = new DrawShape()
        {
        public void draw2DShape()
         {
        System.out.println("2D drawing...");
        }
      };
   obj.draw2DShape();
    }
}
```