Object Oriented Programming

principles of oop

1. Abstraction → Hiding Internal detail & showing only required features
2. Encapsulation
3. Inheritance → Inheriting the previous code & add some extra feature to make new code or specilization
4. polymorphism → It is a Genralize term

TV
↙ ↓ ↘
LG Sony Samsung ← This is polymorphism

Object

→ To define object it must have i) property
ii) Behaviour.

- Class Vs object

ex Car Design → class
Car model → object

- one class can have many object

ex :-

```
class Television                    class
{
    private int channel;
    private int volume;             } property

    public void changechannel()
    {
        s.o.p("change channel");    } method
    }
    public void changeVolume()
    {
        s.o.p("change volume");
    }
}

class Test
{
    p.s.v.m (String arg[])
    {
        Television t = new Television();   ← object
        t.changechannel();
        t.change volume();
    }
}
```

reference

We converty property as variable or Data member
& Behaviour as method or function

Data Hiding

getLength
setLength

→ Data can be hide by setting them as private
→ It is accesable only in class.

ex
```
class Rectangle
{
  private int length;   ← attribute
  private int breadth;
  int get length()
  {
    return length;
  }
  void setLength (int l)
  {
    Length = l;
  }
```

← if we dont want to change value we can write it as final int

reading

initalizing

for point

r1. SetLength(l);

constructor

→ It is method of class which is automatically called whenever an object is created.
→ every class in Java have some default constructor
→ It does not have any returntype.
→ Name of constructor is same as class name
→ It may be public or private

* There are two types of constructor
  1). Non parametrized constructors ()
  2) parameterized constructors (int l, int b)

Property
constructor()
get ()
set ()
Method()

ex of parametirized constructor

```
class product
{ // property //
  private string itemno;
  private String name;      } property
  // constructor //
  Public product (String itemno)
  {
    itemno= itemno;
  }
  Public product (string ~~no~~ itemno, string name)
  {
    itemno = itemno;
    SetName(name);
  }
  // get method //
  public string get itemno()
  {
    return itemno;
  }
  public string getname()
  {
    return name;
  }
  // Set method //
  public void ~~set name~~ SetName (string name)
  {
    this. name= name;
  }
```

constructor

- purpose of user define constructor?

→ To initialise the properties of class.

Inheritance

- **Generalization** :-
  Bottom up.

  Interfaces

  Smart phone → Super / Base

  iphone, Samsung, vivo → sub / Derived

- **Specialization** :-
  Top down.

  iphone x → iphone xs

  Innova → Fortuner.

  Inheritance

- Inheritance → Specialization
- Abstract classes
- Interface → Generalization.

Inheritance

→ It is process of aquiring the features of existing class into a new class.

e.g. Cylinder extends circle. ← key word for inheritance

```
ex Clas circle
{
  privat double radius; ← property
  public circle() ← constructor
  {
    radius = 0.0;
  }
  public double area() {}
  public double perimeter() {}
}
class cylinder extends circle
{
  private double height;
  public cylinder()
  {
    height = 0.0;
  }
  public double volume() {}
}
```

} method

circle → parent / Base / super

↑

cylinder → child / der / sub.

* Note

① whenever we have cylinder extends circle → everything from circle class available in cylinder class

② whetere everything is accesable or not.

③ Inheritance cannot access the properties of inherited class

* Constructor

```
class Account {
  private ab - a
          bc - b
          c
          !
          x

  public account ( a, b - - ):
  {
    ab = a;
    bc = b; ..
    }

  public string get ab name()
  {
    return ab }
    !
    !
  public void set ab ( string a)
  {
    ab = a;
```

* concept of super in Inheritance

```
class ~~parent~~ rect
    int length
    int breadth.
    {
    |
    |
    |

    public rect ( int l, int b)
    {
      le- -
      - -
    }.

class cuboid extends rect

    public cuboid (int l, int b, int h)
    {
      super (l, b);  →  calls the constructor
                        having 2 parament from
      height = h;       a base class
    }
```

Method overriding

→ Redefing a method of super (base) class in sub (child) class

→ When both base class & derived class have same name of method at that time method overriding is used

Ex

```
class super
{
  public void display()
  {
    s.o.p ("Hello");
  }
}

class sub extends super
{
  public void display()
  {
    s.o.p ("Hello welcome");
  }
}

class Test
{ p.s.v.m()
  {
    super su = new super();
      su.display() → Hello

    sub s = new sub();
      s.display() →
```

expected	actual
hello hello display	→ hello display

↓

* Dynamic method dispatch

A super (base) class reference holding an object of sub class (child) & overridded method is called.

→ then method of an object is called not the method of reference.

ex

```
super su = new sub();
su.display ();  - - - Hello welcome.
```

→ It is useful for acheving run-time polymorphism using method overriding

→ If we are doing so then we can access only those methods which is in super, the methods which is sub() we cannot access that extra method coz reference is super

ex

```
class super
{
 void meth1() { sop - - }
 void meth2() { - - - }
}

class sub extends super
{ void meth2() { - - - } - meth2 is overridden
 void meth3() { - - - } }

class test
ps.v main()
{ super s = new sub();
 s.meth1();
 s.meth2(); - - is from sub class
 s.meth3(); ✗ not accessible
```

* polymorphism

→ one name different actions

→ polymorphism is achieve using method overloading & method overriding

Abstract classes → useful for polymorphism
→ achieve generalization

- There are two types of classes
  - Concrete class → normal class
  - Abstract class → you can create reference but cannot create a object.

→ Abstract method

→ A method which does not have body

ie abstract void meth 2();

→ It is undefine method.

→ Abstract class

→ When any class have at least one or more than one Abstract method then that class will be an abstract class.

→ If any class inherting from abstract class then that sub class will also became an abstract

↑

To become concrete it must override all abstract methods

eg

```
abstract class KFC
{ KFC() {}
  void makeItem()
  {=
  }
  abstract void billing();
  abstract void offer();
}.
```

It's obj cannot be created.

```
class myKFC extends KFC
{ myKFC()
  {=
  }
  void billing()
  {=
  }
  void offer().
  {=
  }
  void festive offer()
  {=
  }
}
```

→ all abstract method is override

```
KFC K= new myKFC();
```

→ It is allowed.

---

* Do & Don'ts

- if any class is abstract then we cannot make that class as final
- We cannot make abstract class as static
- Abstract method cannot be final & static

Interfaces

* Difference in abstract class & Interfaces

* **Abstract class**

```
abstract class Test1
{
abstract public void meth1();
abstract public void meth2();
}
Class Test2 extends Test1
{
public void meth1()
{=
}
public void meth2()
{=
}
}.

Test1 t = new Test 2();
```

Interface

```
{
  Interface Test 1
{ void meth1();
  void meth 2();
}
Class Test 2 implements Test 1
{
  public void meth1()
  {=
  }
  public void meth2()
  {=
  }.

Test1 · t = new test 2();
```

ex

```
class phone
{ public void call () { --- }
public void sms () { --- }
}
interface Icamera.
{ void click ();
  void record ();
}
interface Imusic Player
{ void play ();
  void pause();
  void stop ();
}
```

```
class smartphone extends
Phone implements camera,
musicplayer.
{
public void video call () { --- }
void click () { --- };
void pause () { --- ; }
void record () { --- }
void play () { --- }
void stop () { --- }

Smartphone s = new smartphone
Phone p = s;
camera c = s;
music player m = s;
```

Call Back method

→ Call Back method are used in event handelling.

do's & donts

- methods in interface are public & abstract
- If we declare any variable like int x=10
  x should be in upper case
- variable is final & static ie constant
- We can directly use methods of interfare without creating their object
- Interfare can be extended ie inherited
- Interface can have default methods
- method of Interfare can be private

Inner Class

→ A class inside a class known as inner class also known as nested Inner class

• need of Inner class

→ If you have a class & it is becoming big & complex, so if you want to make it simple then you can define inner class or interface

Types of Inner class

1. Nested Inner class
2. local Inner class
3. Anonymous Inner class
4. Static inner class

1. Nested Inner class → A class inside a class.

Note: inner class can access the member of outer class directly & outer class cannot access the member of inner class directly unless it has created an object

ex code

```
class outer
{ int x = 10;
  class Inner
  { int y = 20;
    void innerDisplay()
    {  S.o.p (x);
       S.o.p (y);
    }
  }
void outerDisplay()
  { Inner i = new Inner();
    i.innerDisplay();
    S.o.p (i.y);
} }
```

... inner class can access the member of outer class

```
class Test
{
  p.s.v.m()
  {
    outer o= new outer();
    o.outerDisplay();
    outer.Inner i= new outer().new Inner();
  }
}
```

2. **local Inner class** :- A class inside a method

Note:-

if you want to any class to inherit from some existing class or implement interface & that class is useful only inside the method not outside then local inner class is useful.

Ex code:-

```
class outer {
  void Display()
  {
    class Inner
    {
      void innerDisplay()
      {
        S.o.p("Hello");
      }
    }
    Inner i= new Inner();
    i.innerDisplay();
  }
}
```

3. Anonymous Inner Class

It can be define at a time of creation of object itself, you can define a class & create a object

- It does not have any name.
- It is created just below where object is created
- It is used to override the methods of abstract class

ex - code

It can extend exactly one class or implement exactly one interface

```
abstract class My
{
  abstract void display();
}
class outer
{
  public void meth()
  {
    my m = newmy();
    {
      public void display()
      {
        S.o.p ("Hello");
      }
    };
    m.display();
  }
}
```

// Anonymous class

Note

In Java we can create a object of a class in start i.e without creating a class

4. Static Inner class

→ It is static members of outer class & a obj of static inner classes can be created outside the outer class, you can create/access them from anywhere, & you don't have to create a obj of outer class

• In inner class we only access the static method from outer class not all

Ex code

```
class outer
{
  Static int x=10;
        int y = 20;
  static class Inner
  {
    void display()
    {
      s.op (x);   ✓ allowed
      s.op (y);   ✗ not allowed
    }
  }
}
class Test.
{
  p.s.v.m ()
  {
    outer.Inner i=new outer.Inner();
    i.display();
  }
}
```

not i= new outer() new Inner()

Static keyword

- It is used in
  - Static variables
  - Static Methods
  - Static nested class
  - Static Blocks

→ It is used to represent meta data
- meta data ⇒ Data about data

* It is useful for representing info of the class not about obj

Static members / variables

* Static members belongs to class & they can be shared by all the obj of class
* Static members can also be use as shared data between all the obj of class
  → if one obj modified the value of member then for other obj that value be changed
* Static member can be accessed using class name insted of creating object
* It is created in method area

Static methods

* It belongs to a class
* It is common for all the objects
* They can be called by using class name
* It can access only static member

ex - code

```
class Honda city
{
  static long price =10;
  int a, b;

  static double onRoadPrice (string city)
  {
  switch (city)
  {
   case "delhi":
     return price + price * 0.1;
   case " mumbai":
     return price + price * 0.09;

  }
}
```

- For variable

```
public class Test
  {
    p.s.v.main()
    {
      Hondacity h1= new Honda city();
      Hondacity h2= new Honda city();
      h1.price = 20;
      S.op(h2.price);-- -- it will 20 not 10
```

- For method

```
class Test
{
  p.s.v.main()
  {
  S.O.P(HondaCity.onRoadPrice("delhi"));
  }
}
```

- **Static Block**

We can write set of statement as a block & declare it as static & those set of statement executed ones the class is loaded before the creation of any object of that class

ex - code

```
public class practice
{
  static
  {
    S.o.p("Block 2");
  }

  ps.v.m()
  {
    s.o.p("main");
  }

  static
  {
    s.o.p("Block 1");
  }
```

output

Block 2
Block 1
main

It is not commonly use

---

* **Final keyword**

→ It is used to declare variable as const or value of variable does not change

1. Final variable
2. Final method
3. Final class

- **Final Variable**
- It is considered as const
- The variable name should be written in upper case

ex

```
Final int MAX;
```

- we can declare final variable in 3 ways.

ex

```
class My
{
① final int MIN = 1;
  final int NORMAL;
  final int MAX;

  static
② {
    NORMAL = 5;
  }

  My ()
③ {
    MAX = 10;
  }
}
```

- Final method

- If any method is final then we cannot override that method

ex

```
class super
{
  final void meth()
  {
    s.o.p ("Hello");
  }
}

class sub extends super
{ void meth () ✗ (error)
  { s.o.p ("Hi");
  }

  void meth2()
  {
    s.o.p ("Bye");
  }
```

- Final class

- A final class cannot be extended ie inherited, we can create obj of class & use it but cannot inherit it.

ex

```
final class super
{
  =
  =
}

class sub extends super ✗
{ =
}; (error)
```

* Singleton class

A class which has only one obj, more than one obj is not allowed or created

→ In this class only one obj is created

ex:

```
Class coffeeMachine
{
  private float coffeeQty;
  private float waterQty;
  =
  static private coffeeMachine our = null;

  private coffeeMachine ()
  {
    coffeeQty = 1;
    waterQty = 1;
  }

  Static public coffeeMachine getInstance()
  {
    if (our == null)
      our = new coffeeMachine();
    return our;
  }
  in p.s.vm();
  coffeeMachine c = CoffeeMachine.getInstance();
```

- This is how obj is created
- We cannot create a obj of such class directly