Object Oriented Programming

- Java is the object-oriented programming language.
- In object-oriented programming language everything is represented through the object.
- We follow Object-oriented programming concepts for the development and maintenance of the program is very simple.

Components of the oops are:-

- class
- object
- inheritance
- polymorphism
- abstraction
- encapsulation

Inheritance:-

- If we want to get the properties (variable) and behavior (method) of another class to our class then we should go for Inheritance.
- By using inheritance, we get an IS-A relationship between parent class and child class.
- By using extends keyword we get an IS-A relationship.
- The main objective of the inheritance concept is code reusability.
- In inheritance a class that provides properties to another class is called parent class or base class and the class which acquires the properties from the other class is called child or derived class.

Example:-

```
class Fruit
{
}
class Mango extends Fruit
{
}
```

In this example, IS-A relationship between Mango and Fruit and we can say Mango is a Fruit. we get the properties of the Fruit class in our Mango class by using extends keyword. here Fruit is the parent class and Mango is a child.

Case1:-The most common properties which are applicable for all classes and with those properties we declared a class as a parent. And all child classes can use parent class properties by using extends keyword.

Example:-In this program we have not used Inheritance so the length of the code will increase and the readability of the code will down.

```
1. class Parent
2. {
3. public void m1()
4. {
5. System.out.println("m1() method");
6. }
7. public void m2()
8. {
9. System.out.println("m2() method");
10. }
11. }
12. class Child
13. {
14. public void m1()
15. {
16. System.out.println("m1() method");
17. }
18. public void m2()
19. {
20. System.out.println("m2() method");
21. }
22. public void m3()
23. {
24. System.out.println("m3() method");
25. }
26. }
27. class GrandChild
28. {
29. public void m1()
30. {
31. System.out.println("m1() method");
32. }
33. public void m2()
```

```
34. {
35. System.out.println("m2() method");
36. }
37. public void m3()
38. {
39. System.out.println("m3() method");
40. }
41. public void m4()
42. {
43. System.out.println("m4() method");
44. }
45. }
```

In the above program length of the code is increased so the most common code we defined inside the parent class and all child classes inherited parent that class.

```
1.  class Parent
2.  {
3.  public void m1()
4.  {
5.  System.out.println("m1() method");
6.  }
7.  public void m2()
8.  {
9.  System.out.println("m2() method");
10. }
11. }
12. class Child extends Parent
13. {
14. public void m3()
15. {
16. System.out.println("m3() method");
17. }
18. }
19. class GrandChild extends Parent
20. {
21. public void m3()
22. {
23. System.out.println("m3() method");
24. }
25. public void m4()
26. {
27. System.out.println("m4() method");
28. }
29. }
```

Note:-In the first program we have not used inheritance the length of the code is a total of 45 lines so in the second program we used inheritance then the length of the code is only 29 lines, the main purpose inheritance concept is code reusability.

Case2:-

Example:-

```
1. class ParentDemo
2. {
3. public void m1()
4. {
5. System.out.println("Parent Demo");
6. }
7. }
8. class ChildDemo extends ParentDemo
9. {
10. public void m2()
11. {
12. System.out.println("child Demo");
13. }
14. public static void main(String[]args)
15. {
16. ChildDemo c1=new ChildDemo();
17. c1.m1();
18. c1.m2();
19. ParentDemo p1=new ParentDemo();
20. p1.m1();
21. p1.m2();//line 21
22. ParentDemo p2=new ChildDemo();
23. p2.m1();
24. p2.m2();//line 24
25. ChildDemo c2=new ParentDemo();//line 25
26. }
27. }
```

Result:-

```
line 21 :-we get compile time error
Test.java:21: error: cannot find symbol
p1.m2();//line 21
symbol:method m2()
location: variable p1 of type ParentDemo 1 error
Line 24:-
Test.java:24: error: cannot find symbol
```

```
p2.m2();//line 24
symbol:   method m2()
location: variable p2 of type ParentDemo
Line 25:-
Test.java:25: error: incompatible types: ParentDemo cannot be
converted to ChildDemo
ChildDemo c2=new ParentDemo();//line 25
```

Note:-1 by using child reference can call both the Parent class method and Child class method because all methods present inside the parent class by default available to the child class

Note:-2 by using parent reference we can call only the parent class method because the child class method is not by default available to the parent class so by using parent reference if we are trying to call the child class method then we will get compile time error.

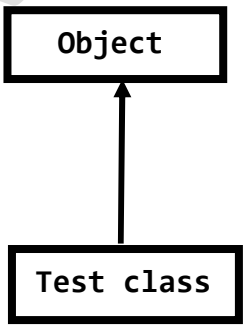
Note:-3 parent references can be used to hold the child object but by using that reference we can call only the parent method if we are trying to call the child method then we will get compile error.

Note:-4 Child reference can't hold parent object if we try to hold parent object then we will get compile time error.

Case:-3 If our class extends any other class then it is an indirect child of the Object class if our class not extends any class then it direct child of Object class. It's performed Multilevel inheritance not multiple inheritance.

Example:-1

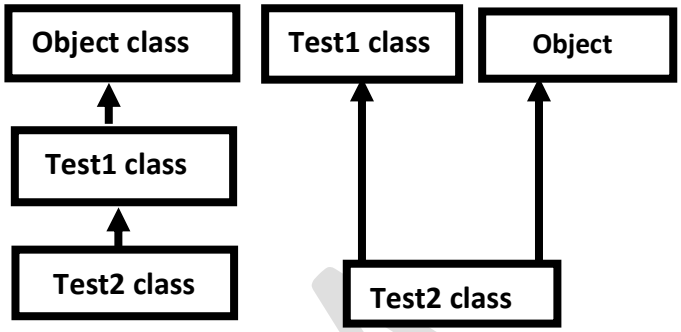
```
class Test
{

}
```



Note:-In this example Test class does not extend any other class it means it is a direct child class of Object.

Example:-2

```
class Test1
{
}
class Test2 extends Test1
{
}
```



valid invalid

Note:-In this Example class Test2 extends Test1 then Test2 is the direct child class of Test1 and the indirect child class of Object. its performed Multilevel inheritance not Multiple inheritance .at any cost java does not support multiple inheritance.

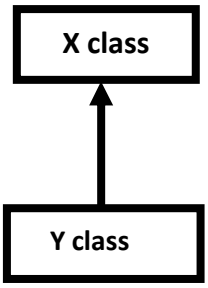
Types of Inheritance:-There are total five types of Inheritance

1.) Single inheritance
2.) Multilevel inheritance
3.) Hierarchical inheritance
4.) Multiple inheritance
5.) Hybrid inheritance

Note:-java not support Multiple and Hybrid inheritance

Single inheritance:-A class has only one direct parent class is considered as single inheritance.

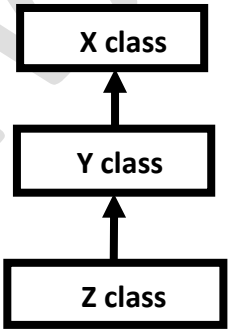
Example:-




```
class X
{
public void m1()
{
System.out.println("m1() method X class");
}
}
class Y extends X
{
public void m2()
{
```

```
System.out.println("m2() method Y class");
}
public static void main(String[]args)
{
Y y=new Y();
y.m1();
y.m2();
}
}
Result:- m1() method x class
         m2() method y class
```

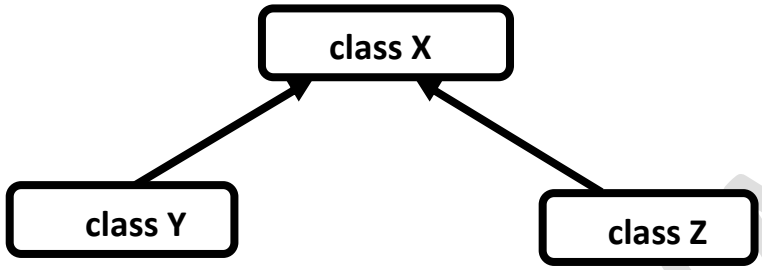
Multilevel Inheritance:- A class sub class extending parent class then that sub class will becomes parent class of next extended class this flow is called multilevel inheritance.




```
class X
{
public void m1()
{
System.out.println("m1() method x class");
}
}

class Y extends X
{
public void m2()
{
System.out.println("m2() method y class");
}
}

class Z extends Y
{
public void m3()
{
System.out.println("m3() method Z class");
}
public static void main(String[]args)
{
Z z=new Z();
z.m1();
z.m2();
z.m3();
}
}
```

Result:- m1() method x class
m2() method y class
m3() method Z class

Hierarchical inheritance:-A parent class extended by multiple child classes is called Hierarchical inheritance.



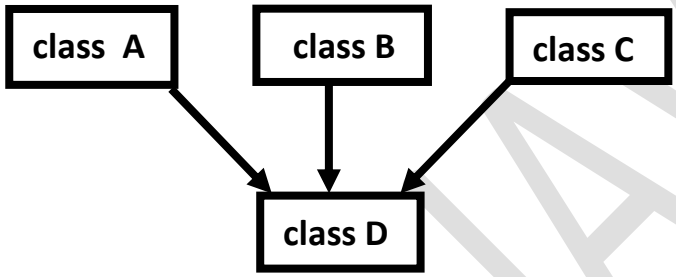
Example:-

```
class X
{
public void m1()
{
System.out.println("m1() method X class");
}
}
class Y extends X
{
public void m2()
{
System.out.println("m2() method Y class");
}
}
class Z extends X
{
public void m3()
{
System.out.println("m3() method Z class");
}
}
class Test
{
public static void main(String[]args)
{
Y y=new Y();
y.m1();
y.m2();
Z z=new Z();
z.m1();
z.m3();
}
}
```

Result:-
m1() method X class
m2() method Y class
m1() method X class
m3() method Z class

Multiple Inheritance:-A child class is extending more than one parent class is called multiple inheritance but java not support multiple inheritance because there may be chance of ambiguity error. If we are extending more than one class then we will get compile time error.

Example:-




```
class X
{
public void m1()
{
System.out.println("hello m1() x class");
}
}
class Y
{
public void m1()
{
System.out.println("hello m1() y class");
}
}
class Z extends X,Y
{
public static void main(String[]args)
{
Z z=new Z();
z.m1();
}
}
```

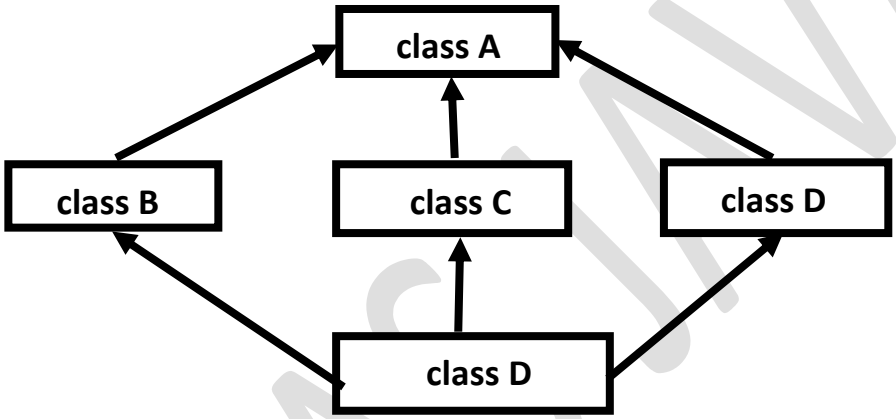
Result:- Test.java:15: error: '{' expected
class Z extends X,Y

Note:-In this program by using object reference of Z child class we are calling parent class m1() method and m1() method present in both parent X and Y so we get an ambiguity problem. If we are trying to perform multiple inheritance then we will get compile time error.

Hybrid inheritance:-

- It is the combination of hierarchy and multiple inheritance , java is not support multiple inheritance therefore java is also not supported the hierarchy interface.
- If we are trying to perform Hybrid inheritance then we will get compile time error

Example:-



Cyclic inheritance:- class A extends itself or class A extends class B and class B extends class A is called cyclic inheritance. Java does not support cyclic inheritance. If we are trying to perform cyclic inheritance then we will compile time error.

Example:-1

```
Class A extends A
{
}
```

Result:-

```
A.java:1: error: cyclic inheritance involving A
class A extends A
1 error
```

Example:-2

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

Result:- Test.java:1: error: cyclic inheritance involving A
class A extends B

Abstraction in java:- Abstraction is a process through which we hide the internal implementation and show or highlight functionality.

The main purpose of abstraction is:-

1) Achieve security because its shows only functionality and hide the internal implementation.
2) Make user friendly.
3) Improve maintainability.
4) Enhancement (upgradation) is very easy because without affecting the outside we can perform any change.
5) By using abstract class and interface we achieve abstraction.

abstract modifier:- abstract modifier we can apply on class and method only.

There are two types of methods:-

A) Normal Method
B) Abstract Method

Normal Method:-The method which contains declaration and implementations such types of the method we can say Normal Method.

Example:-

```
public void m1()//method declaration
{
System.out.println("Hello");//method implementation
}
```

abstract method:-If we don't know about the implementation of the method still we know only about the declaration of a method such types of methods are declared as abstract, hence abstract method end with **;** a semicolon.

In another word, A method that has a declaration but does not have implementation is called an abstract method.

Example:-

```
public abstract void m1();//valid
public abstract void m1()//invalid because ;(semicolon) is not
available.
{
}
```

Example:-

```
abstract class Car
{
public abstract String typeOfCar();
}
class MarutiSwifts extends Car
{
public String typeOfCar()
{
return "Hatchback";
}
}
class LandRover extends Car
{
public String typeOfCar()
{
return "SUV";
}
}
class Test
{
public static void main(String[]args)
{
MarutiSwifts ms=new MarutiSwifts();
String  types1=ms.typeOfCar();
System.out.println(types1);
LandRover lr=new LandRover();
String types2=lr.typeOfCar();
System.out.println(types2);
}
}
```

Result:-

```
Hatchback
SUV
```

Some important point of abstract method:-

1) Order of modifier is not important public abstract or abstract public both are same.
2) By declaring the abstract method we defined the standard of method in the parent class. with these standards, child class compulsory provides the implementation.

abstract class:-A java class that is declared with an abstract modifier such types of the class is called abstract class. We can't create object of abstract class because of partial implementation.

partial implementation means the abstract class contains both methods Normal method and the abstract method.

Note:-Suppose it is possible to create an object of abstract class, by using that object reference if we are calling abstract method then abstract method will be executed but abstract method does not contain body so it is a nonsense approach to create an object of abstract class.

```
abstract class Test
{
public static void main(String[]args)
{
Test t=new Test();
}
}
Result:-
Test.java:5: error: Test is abstract; cannot be instantiated
Test t=new Test();
```

Relation between abstract class and abstract method:-

Case1:-If a class contain at least one abstract method then the corresponding class must be abstract otherwise we will get compile time error.

Example:-This class contains one abstract method and if the class is not declared as abstract then we get compile time error.

```
class Test
{
public void m1()
{
System.out.println("Hello Test0");
}
public abstract void m2();
}
```

Result:- Test.java:1: error: Test is not abstract and does not override abstract method m2() in Test class Test.

Case2:-class Test does not contain any abstract method but class Test may be abstract or may not be abstract. If we don't want to instantiate a class such types of classes are declared as abstract.

Example:-Adopter class and Httpservlet class is abstract class but do not contain any abstract method. Generally, all adopter class recommended declared as abstract.

Case3:- Abstract class contains abstract methods for that method we provide implementation in child classes if the child class is unable to provide implementation of all abstract method then that class declared as abstract .in situation next level child class is responsible to provide an implementation of the remaining abstract method.

Example:-

```
abstract class Test1
{
public abstract void m1();
public abstract void m2();
public abstract void m3();
public void m4()
{
System.out.println("Hello abstract Test1");
}
}

abstract class Test2 extends Test1
{
public void m1()
{
System.out.println("Hello Test2");
}
}

abstract class Test3 extends Test2
{
public void m2()
{
System.out.println("Hello Test3");
}
}

class Test4 extends Test3
{
public void m3()
{
```

```
System.out.println("Hello Test4");
}
public static void main(String[]args)
{
Test4 t=new Test4();
t.m1();
t.m2();
t.m3();
t.m4();
}
}
```

Result:-

```
Hello Test2
Hello Test3
Hello Test4
Hello abstract Test1
```

Case4:-Abstract method can contain any return type.

Example:-

```
abstract class Test1
{
public abstract int m1(int a);
public abstract boolean m2(boolean b);
}

class Test2 extends Test1
{
public int m1(int a)
{
System.out.println(a);
return 10;
}
public boolean m2(boolean b)
{
System.out.println(b);
return false;
}
public static void main(String[]args)
{
Test2 t2=new Test2();
t2.m1(10);
t2.m2(true);
}
}
```

Result:- 10
True

Data Hiding:-If we want our internal data or important data can't go outside the class after validation/authentication outside of class can be accessed is called Data Hiding.

We achieve data hiding by declaring data members as private. The main objective of Data Hiding is to gain security so it is highly recommended to declare data members as private.

Example:-

```
class Email
{
private String userid;
}
```

Encapsulation:-The process binding the data member and corresponding method into a single unit is called Encapsulation.

Note:-The main advantage of Encapsulation is security and the main disadvantage is an increase in the length of the code reduces the processing.

➔We are achieving Encapsulation by declaring data members as private because private data members we can access only within class outside of the class we can't access.

➔ We can access private data members outside of the class by using public setter and getter method.

Syntax of setter method and getter method:-

setter method:-It should be public return type must be void it take some parameter

```
public void setXXX(parameter)
```

getter method:-It should public return types can't be void and it not contain any parameter.

```
public returntype getXXX()
```

Example:-

```
class EncapsulationDemo
{
private int id;
private String name;
public void setId(int id)
{
```

```
this.id=id;
}
public void setName(String name)
{
this.name=name;
}
public int getId()
{
return id;
}
public String getName()
{
return name;
}
}
class Test
{
public static void main(String[]args)
{
EncapsulationDemo ed=new EncapsulationDemo();
ed.setId(1118);
ed.setName("vikas");
int ID=ed.getId();
System.out.println(ID);
String NAME=ed.getName();
System.out.println(NAME);
}
}
```

Result:-

1118

vikas

Tightly Encapsulation:-Inside a class If all data members are declared as private then we can say that This class is Tightly Encapsulated whether the class contains public getter and setter methods or not this thing is not required to check.

Example1:-This class is tightly Encapsulated because all data member present inside the class is private.

```
class Test
{
privat int x;
public int getX()
```

```
{
}
}
```

Example2:-In this program class Test1 Test3 is tightly encapsulated but Test2 is not tightly encapsulated

```
class Test1//Tightly Encapsulated
{
private int x=12;
}
class Test2 extends Test1//Not Tightly Encapsulated
{
 int y=13;
}
class Test3 extends Test1//Tightly Encapsulated
{
private int z=14;
}
```

Example:-In this program all classes are not tightly encapsulated because parent class data member declared as public. If parent class is not tightly encapsulated then there no chance to child class is tightly encapsulated.

```
class Test1//not tightly encapsulated
{
int x=12;
}
class Test2 extends Test1//not tightly encapsulated
{
private int y=13;
}
class Test3 extends Test1//not tightly encapsulated
{
private int z=14;
}
```

Method Overloading:- More than one method with the same name but a different number or type of parameter those methods are called method overloading.

C language does not support method overloading. If we perform any change in argument (number or type) compulsory we should go for a new method name which increases the complexity of programming. we solve this problem in java by using the method overloading.

Problem in C language:-

avar(int a);➔avar(10);
bvar(char ch);➔bvar(101);
cvar(flat f);➔cvar(10.30);

Solve this problem in java:-
var(int a)➔var(10);
var(long l)➔var(101);
var(float)➔var(10.36);

Example:- A class containing multiple methods with the same name but a different number of arguments is method overloading.

```
class Test
{
public void m1()
{
System.out.println("no-arg method");
}
public void m1(int x)
{
System.out.println("int-arg method");
}
public void m1(double d)
{
System.out.println("double-arg method");
}
public static void main(String[]args)
{
Test t=new Test();
t.m1();
t.m1(10);
t.m1(12.30);
}
}
```

Result:- no-arg method
int-arg method
double-arg method

Note-In overloading method resolution always takes care by the compiler based on the reference type. In overloading run time object won't play any role hence overloading is always considered as compile time polymorphism or static polymorphism or easy binding.

Method Overriding:-All methods present in the parent class are by default available to the child class. parent class method implementation we do not want to use in child class then child class redefined method implementation based on own requirement this process is called method overriding.

➔In other words If the child class not required parent class method implementation then override the parent class method in the child class and the child class provide method implementation based on requirement.

➔Child class overrides parent class method to provide method implementations.
Parent class method is called:-overridden method.
Child class method is called: - overriding method.

Example:-

```
class Animal
{
public void eat()
{
System.out.println("grass");// overridden method.
}
}
class Monkey extends Animal
{
public void eat()//overriding method.
{
System.out.println("fruit");
}
public static void main(String[]args)
{
Monkey m=new Monkey();
m.eat();
}
}
```

Result:-fruit

Note:-Animal class eat method is overridden method and Monkey class eat method is overriding method.

Overriding with respect static method:-

Case:-1 we can't override static method as non-static if we are trying to override static method as non-static then we will get compile time error.

Example:-

```
class Parent
{
public static void m1()
{
}
}
class Child extends Parent
{
public void m1()
{
}
}
```

Result:- Test.java:9: error: m1() in Child cannot override m1() in Parent
public void m1()
overridden method is static

Case:-2 we can't override non-static method as static if we are trying to override non-static method as static then we will get compile time error.

Example:-

```
class Parent
{
public void m1()
{
}
}
class Child extends Parent
{
public static void m1()
{
}
}
```

Result:- Test.java:9: error: m1() in Child cannot override m1() in Parent
public static void m1()
overriding method is static

Case:-3 If we are trying to override static method as static its look like method overriding but it is not method overriding its method hiding.

Example:-

```
class Parent
{
public static void m1()
{
}
}
class Child extends Parent
{
public static void m1()
{
}
}
```

Method hiding:-It is similar to the method overloading but some basic difference

1) Overridden method and overriding method both should be static.
2) Method resolution takes care by java compiler.
3) It is also known as static polymorphism, compile time polymorphism, and early binding.

Difference between Method overloading and Method overriding

Method Overloading	Method Overriding
1. In this multiple method with name same but different types of arguments at least order or we can say method signature must be different.	1. In this method name and argument type must be same of parent class & child class or we can say method signature must be same.
2.private,static,final method can be overloaded	2. private,static,final method can't be overloaded.
3.Access modifier there is no restriction	3.Overriding method not reduce the scope of overridden method otherwise we will get compile time error.
4.There is no restriction with exception .	4. If child class overriding method thorough any checked exception then parent class overridden must throw same or parent types of exception otherwise we will get compile time error.
5. Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.	5.Performace wise it not mention compare to overloading.
6.It is also known as compile time polymorphism, static polymorphism ,early binding	6.It is also known as runtime polymorphism/dynamic polymorphism/ late binding.

Polymorphism:-

- One name but multiple form is called polymorphism
- One functionality with different action is called polymorphism.

There are two type of polymorphism:-

A) Compile time polymorphism/static polymorphism/early binding:-we can achieve compile time polymorphism by using method overloading. Method resolution of compile time polymorphism takes care by compiler.

B) Runtime polymorphism/dynamic polymorphism/late binding:-we can achieve runtime polymorphism by using method overriding. Method resolution of runtime polymorphism takes care by jvm based on runtime object.

Compile time Polymorphism:-

Example:-

```
class Test
{
public void m1(int x)
{
System.out.println("int-ars");
}
public void m1(double d)
{
System.out.println("double-arg");
}
public static void main(String[]args)
{
Test t=new Test();
t.m1(10);
t.m1(12.36);
}
}
```

Result:-

```
int-ars
double-arg
```

Note:-This program contain two method with same method name but both method performed different operation so we can say One name but multiple form.

Runtime Polymorphism:-

Example:-

```
class Animal
{
public void eat()
{
System.out.println("grass");
}
}
class Monkey extends Animal
{
public void eat()
{
System.out.println("Fruit");
}
```

```
public static void main(String[]args)
{
Monkey m=new Monkey();
m.eat();
}
}
```

Result:-Fruit

Note:-In this program Parent class and Child class both contain same method name and performed different operation so we can say one name but multiple form.