# ABSTRACTION

## (1) Abstract class:

Abstraction is an important concept of OOPS that allows us to hide unnecessary details and only show the needed information. This allow us to manage complexity or hiding details with a simpler, higher-level idea. There are two ways to achieve abstraction (a) abstract class (b) interface

- We use abstract keyword to declare an abstract class.
- An abstract class can have both the regular methods and abstract methods.
- A method that doesn't have its body is known as an abstract method.
- We can not create object of abstract class but only it reference we can. And we can create subclasses from it then we can access member of abstract class using the object of the subclasse.
- If the abstract class includes any abstract method, then all the child classes inherited from the abstract super class must provide the implementation the abstract method.

Ex-

```
package org.example.opps_concept.abstraction;

public class Abstract {
    public static void main(String[] args) {
        // not allowed
//        Vehicle v1 = new Vehicle();

        Car c1 = new Car();
        c1.accelerate();
        c1.honks();
        Vehicle.monks();
    }
}


abstract class Vehicle {
    abstract void accelerate();
    abstract void breaks();

    void honks(){
        System.out.println("honks");
    }

    static void monks(){
        System.out.println("monks");
    }

    // both not allowed
//    void t1();
//    static void t2();
}

class Car extends Vehicle {

    @Override
    void accelerate() {

    }

    @Override
    void breaks() {

    }
}
```

INTERFACE :

- An interface have fully abstract method before java 8.
- We use the interface keyword to create an interface in java.
- Like abstract class , we can not create objects of interfaces. But reference can create.
- To use an interface , other classes must implement it , we use the implements keyword to implement an interface.
- In interface by default abstract methods are public and abstract don’t need to write.
- In interface by default variables are public, static and final don’t need to write.
- Interface help us to achieve multiple inheritance in java.

Ex-

```
package org.example.opps_concept.abstraction;

public class LearnInterface {
    public static void main(String[] args) {
      // not allowed
//        Animal a1 = new Animal();

        Monkey m = new Monkey();
        m.age=Animal.a;
    }
}

interface Animal {

 public static final int a = 20;
    public abstract void  eats();

}


class Monkey implements Animal {

    int age ;

    @Override
    public void eats() {

    }
}
```

(2) INNER CLASS :

There are two types of inner class

(a) Non-static inner class (inner class):

A non-static nested class is a class withing another class. It has access to members of the enclosing class (outer class ) . it is commonly known as inner class, since the inner class exists within the outer class , you must create the outer class object first, in order to create object of inner class.

(b) Static inner class (nested class):

A static nested class can not access the member variables of the outer class . it is because the static nested class doesn't require you to create an instance of the outer class.

- * Using nested class makes your code more readable and provide better encapsulation.

Ex-

```
package org.example.opps_concept.abstraction;

public class InnerClass {

    // inner class
    class Toy {
        int price;

    }

    // nested class
    static class Playstation {
        int price;


    }

    public static void main(String[] args) {
        // non static class (inner class)
      InnerClass obj = new InnerClass();
      Toy toy = obj.new Toy();
      toy.price=45;


      // static class (nested class)
        Playstation ob = new InnerClass.Playstation();
        ob.price=50;
    }
}
```

(3) Anonymous class :

In java, a class can contain another class known as nested class , it's possible to create a nested class without giving any name. a nested class that doesn't have any name is known as an anonymous class.

Anonymous class usually extends subclasses or implement interfaces.

(a) A superclass that an anonymous class extends

(b) An interface that an anonymous class implements

- * now anonymous class can be replaced with lambda expression.

  Ex-

```java
package org.example.opps_concept.abstraction;

public class LearnAnonymous {

    // this way also can create sub class of outer class
//    class InnerCls extends OuterClass {
//
//    }


    OuterClass obj = new OuterClass(){

        void sing(){

        }


        @Override
        public void outerMethod() {
            System.out.println("override from outer class");
        }
    };


    SuperInterface sup = new SuperInterface() {
        @Override
        public void interfaceMethod() {

        }
    };



    WalkAble walkAble = (steps -> {
        System.out.println("walk "+steps+" steps");
        return steps;
    });


    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        System.out.println(outerClass);

        LearnAnonymous learnAnonymous = new LearnAnonymous();
        learnAnonymous.obj.outerMethod();
        System.out.println(learnAnonymous.obj);

        // Use the lambda expression
        learnAnonymous.walkAble.walk(5);


    }
}



class OuterClass {
    public void outerMethod(){

    }
}


interface SuperInterface {

     void interfaceMethod();
}

@FunctionalInterface
interface WalkAble {
    int walk(int steps);
}
```