→ Inner Class - A class declared inside another class.

→ Without existing one type of object, If there is no chance of existing another type of object then we should go for inner class.

```
→ class Test
  {
     class A {

     }
  }
```

```
→ class University {

     class Departments {

     }
  }
```

```
→ class Car {

     class Engine {

     }
  }
```

```
→ Interface Map {

     Interface Entry {

     }
  }
```

4 Category
1. Normal or regular Inner class
2. Method local Inner class
3. Anonymous Inner class
4. Static Nested class.

(1) Normal or Regular Inner class

→ class outer
{
class Inner {
}
}

→ Declaring named class without static modifier directly inside class.

→ class Outer {
class Inner {
psvm ( ) {
}
}
}

→ Javac Outer.java

Outer.class
Outer$Inner.class

→ Inner class can not have static member.

```
class Outer
{
    class Inner {
        void m1() {
            Sout("Inner class method");
        }
    }
    psvm (String[] args)
    {
        Outer o = new Outer();
        Outer.Inner i = o.new Inner();
    }
}
```

Outer.Inner i = new Outer().new Inner(); m1();

Accessing Inner class code from static area of outer class. (3)

```
--> class Outer
    {
        class Inner
        {
            Public void m1()
            {
                Sout (" Inner class ");
            }
        }

        Public void m2()
        {
            Inner i = new Inner();
            i.m1();
        }

        psvm (String[] args)
        {
            Outer O = new Outer();
            O.m2();
        }
    }
```

Instance area.

It is easy to call inner class from instance area rather than static area.

Static area ⟸

Accessing Inner class code:

① From static Area of outer class
(or)
② From outside of outer class

```
Outer o = new Outer();
Outer.Inner i = o.new Inner();
i.m();
```

From Instance Area of outer class

```
Inner i = new Inner();
i.m();
```

Case-1

```
class Outer
{
  int x = 10;
  static y = 20;
  class Inner
  {
    Public void m1() {
      Sout(x);
      Sout(y);
    }
  }
  PSVM ( )
  {
    new Outer.new Inner.m1();
  }
}
```

← Yes we can't declare static variable in inner class
But we can access static variable from outer class.

Case-2

```
class Outer
{
  int x = 10;
  class Inner
  {
    int x = 100;
    Public void m1(){
      int x = 1000;
      Sop(x); //1000
      Sop(this.x); //100   Sop(Inner.this.x) //100
      Sop(Outer.this.x) //10
    }
  }
}
```

Method Local Inner classes

--> Some time we declare a class inside method, such type of Inner

--> Nested method concept is not allowed in Java.

--> To define method specific repeatedly functionality.

Outside method we can't access Inner class --> Less Scope.

```
class Test {
    Public void m1()
    {
        class Inner
        {
            Public void Sum (int x, int y)
            {
                Sout (" x + y);
            }
        }
        Inner i = new Inner();
        i.Sum (x, y);
            :
        i.Sum(x100, y100);
            :
            :
            i.Sum(x1000, y1000);
    }
    psvm (String[] args)
    {
        Test t = new Test();
        t.m1();
    }
}
```

We can declare method local inner class in both instance & static method.

→
```
class Test
{
    int x=10;
    static int y=20;
    public void m1()
    {
        class Inner
        {
            public void m2()
            {
                sop(x);
                sop(y);
            }
        }
        Inner i = new Inner();
        i.m2();
    }
}
```

& static method

Non-static var x cannot be referenced from a static context

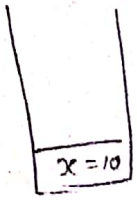
→ If we declare Inner class inside instance method then from that method local Inner class we can access both static & non-static member of outer class directly.

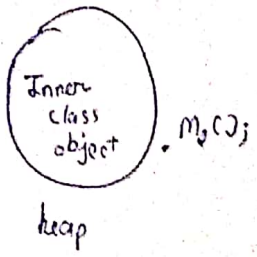
→ If we declare Inner class inside static method then we can access only static member of outer class directly from static method local inner class.

```
class Test
{
    Public void m1()
    {
        int x = 10;
        class Inner
        {
            Public void m2()
            {
                Sout(x);
            }
        }
        Inner i = new Inner
        i.m2();
    }

    PSVM()
    {
        Test t = new Test();
        t.m1();
    }
}
```

→ local variable x is accessed within inner need to be declare as final





→ From method local Inner class we can't access local variable of the method in which we declare inner class.

→ If the local variable (x) declare as final then we can access.

→ If we declare final then at x is replace with 10. And in heap memory value exist.

```
class Test
{
    int i = 10;
    static int j = 20;
    Public void m1()
    {
        int k = 30;
        Static int m = 40;
        final
        class Inner
        {
            Public void m2()
            {
            }
        }
    }
}
```

From instance method (m_1)

i ✓
j ✓
k ✗
m ✓

From static method (m_1)

i ✗
j ✓
k ✗
m ✓

From static method (m_2)

i ✗
j ✓
k
m

→ Inside inner class we can't declare static member / method

~~But we can access static member~~

→ The only applicable modifier for variable in local method is final.

→ Applicable modifier for method local Inner class is [final, strictfp, abstract]

Anonymous Inner classes - without name
- Used for instant use / One time use

① Anonymous Inner class that extends a class
② Anonymous Inner class that implements an interface
③ Anonymous Inner class that defined inside arguments.

```
Popcorn p = new Popcorn();

Popcorn p = new Popcorn()
{

};
```

→ Extend popcorn class
new is an object create for child class
p is reference of parent class

→ We are declaring class that extends the popcorn class without name (Anonymous class)
→ For that child class we are creating an object with parent reference.

```
Thread t = new Thread();

Thread t = new Thread()
{

};
```

```
Runnable r = new
    Runnable ()
{

};
```

Defining a thread by extending Thread class.

```
class MyThread extends Thread
{
    Public void run(){
        for(int i=0; i<n; i++)
        {
            sout(i);
        }
    }
}
class ThreadDemo
{ psvm( )
  {
    MyThread t = new MyThread();
    t.start();
  }
}
```

Defining a Thread
Job of Thread

```
class Thread{
  psvm(){
    Thread t = new Thread()
    {
      Public void run()
      {
        for(i to n)
        sout(i)
      }
    };
    t.start();
  }
}
```

Runnable Interface

```
class MyRunnable implements Runnable
{
   Public void Run()
   {
      for(i to n)
      Sout(i);
   }
}
Class ThreadDemo
{
    PSVMain ( )
    {
      MyRunnable r = new MyRunnable();
      Thread t = new Thread(r);
       t.start();
    }
}
```

→ Now Anonymous class

```
class Thread Demo
{
      psv main ( )
      {
        Runnable r = new Runnable( )
        {
           Public void run( )
           { for (i to n)
             Sout(i);
           }
        }; Thread t = new Thread(r);
             t.start( );
      }
}
```

Short technique

```
class ThreadDemo
{
  psv main ()
  {
    new Thread( new Runnable ()
                {
                  Public void run()
                  {
                    for(i to n)
                    Sout(i);
                  }
    }).start();
```

Normal Java class V/s Anonymous Inner class.

(1) A normal Java class can extend only one class at a time
of course Anonymous Inner class also can extend only one class at a time.

(2) A Normal (J.C) can implement multiple interface simultaneously
But Anonymous (J.C.) can implement one interface at a time.

(3) A normal (J.C) can extend a class & can implement any number of interface simultaneously
But Anonymous (J.C) can extend a class or can implement an interface but not both simultaneously.

(4) Normal Java class have any number of constructor
Anonymous (J.C) no constructor concept available.

4) Static Nested Class - Innerclass with static modifier

```
class Test
{
    int x = 10;        → Instance variable
                         // strongly connected with test object
    .
    .
    static int y = 20; → Static variable
                         // weakly connected with test object
}
```

→ With test class static variable can available.

```
class Outer
{
    static class Nested
    {                    // Weakly attached with outer class
    .
    .
    }
}
```

Example

```
class Outer
{
  static class Nested
  {
    Public void m1()
    {
      Sout ("Static Nested class");
    }
  }
  p s v main (String[] args)
  {
    Nested n = new Nested();
    n.m1();
  }
}
```

** // outside of outer class

```
Outer.Nested n =
    new OuterNested();
```

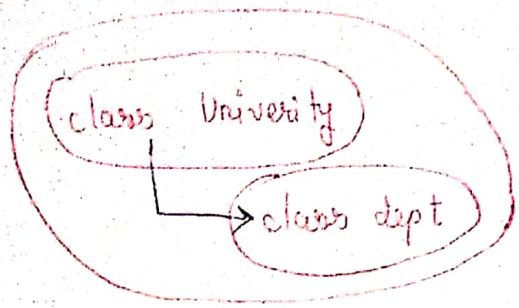
→ Static nested class can declare & access static member component. (14)

→ Non-static member can not be access from static nested class.

→ Anything inside anything is possible in context of (class & Interface)

→ Various nested classes & Interfaces.

Case 1] Class inside class



case 2] class inside interface

```
class vehicleType
{
    interface vehicle
    {
        Public int getNo.of Wheels();
    }
    class Bus implements vehicle
    {
        Public int getNo.of Wheels{
            return 6;
        }
    }
}
```

Case 3] Interface inside Interface

```
Interface Map
{
    Interface Entry
    {
        .
        .
    }
}
```

Map

101	Durga
102	Ravi
103	Shiva
104	Pavan

```
Interface Outer
{
    Public void m1();
    Interface Inner
    {
        Public void m2();
    }
}
class Test1 implements Outer
{
    Public void m1()
    {
        Sout ("Outer Interface");
    }
}
class Test2 implements Outer.Inner
{
    Public void m2()
    {
        Sout ("Inner Interface");
    }
}
```

By default Inner interface is public static

If functionality of class is closely associated with interface then → declar class inside interface

Case - 4] Class inside interface.

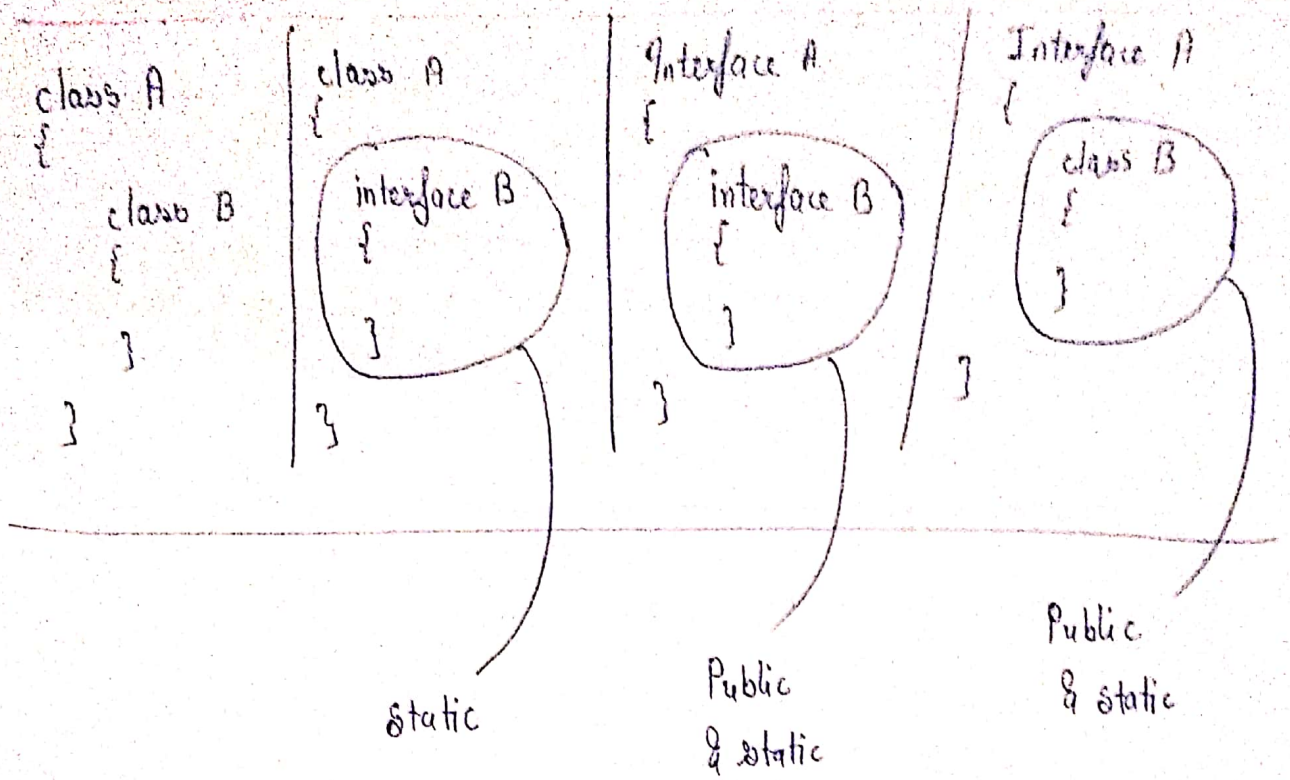
```
Interface Vehicle
{
  Public int getNo.ofWheels();
  class DefaultVehicle implements vehicle
  {
    Public int getNoofDeWheels()
    {
      return 2;
    }
  }
  class Bus implements Vehicle
  {
    Public int getNoofWheels()
    {
      return 6;
    }
  }
}
```

```
Class Test
{
  PSVM (String[] args)
  {
    Vehicle.DefaultVehicle d = new vehicle.DefaultVehicle();
    Sout (d.getNoofWheels());   // 2

    Bus b = new Bus();
    Sout (b.getNoofWheels());   // 6
  }
}
```

→ The inner interface declare inside interface is always public, static

→ The class which is declare inside interface is always public, static whether we are declaring or not.



Blocking Queue - FIFO - Consumer-producer.

Concurrent Map

Concurrent Navigable Map

Blocking Deque

Transfer Queue.

Blocking Queue

① Array blocking Queue

② Delay Queue

③ Linked Blocking Queue

④ priority Blocking Queue

⑤ Synchronous Queue

① Join()

② New-Lock interface over a Synchronized block

Blocking method

↳ block the executing thread untill their operations finished.

Input Stream read()