enum in Java :
--------------
enum is a keyword in java which is introduced from JDK 1.5V onwards.

enum is similar to a class because for enum also .class file will be created internally.

If we want to declare Universal constants then we should use enum concept, all the Universal constants must be separated by comma and at the end ; is optional.

Example :
---------
Color.java
------------

```
public enum Color
{
  RED, BLUE, PINK  //public + static + final (enum constants)
}
```

All the enum constants are by default public static and final

Compiler will automatically generate .class file as shown below :

Color.class
------------

```
public final class Color extends java.lang.Enum
{
  public static final Color RED = new Color();
  public static final Color BLUE = new Color();
  public static final Color PINK = new Color();

  public static Color[] values()
  {
     //return enum array;
  }
}
```

An enum we can define inside a class, Outside of the class and inside a method as well. If we define inside a class then we can declare an enum with private, protected, public, static modifiers. We can't declare final and abstract.

Whenever compiler generates .class file then compiler will automatically add a static method called values() to fetch the enum object, the return type of this method is enum array.

```
public enum[] values()
```

Every enum constant constains order position which starts from number 0, we can retrieve the order position by using ordinal() method of Enum class.

```
public final int ordinal()
```

Every enum impliciltly extends from java.lang.Enum class so enum can't extend any other class but it can implement an interface.

Every enum is impliciltly final so any class can't extends an enum.

Every enum first line is reserved for enum constants so we can't write anything in the first line and that is the reason static block is always executed after constructor.

As we know, an enum is implicitly a class so we can write static, non static variable, static and non static block, static and non static method.

A constructor defined inside an enum must be either private OR package-private.

Every enum constants are Object of type enum.

At the time of loading enum class, static variable will be executed so appropriate constructor will be executed.

An enum represents Universal constant so, We can pass enum constants in switch case statement.

In Enum class name() and ordinal() both are final methods so we can't override hence we can't change the name and order position.

We can compare two enum constants by uisng == operator OR final equals(Object obj) method given by Enum class.

enum Constructor and Methods :
------------------------------
In java.lang package there is a predefined class called Enum which is an abstract class, It contains a parameterized constructor which is used initialized enum field name and order position.

public abstract class Enum

```
{
   private final String name;
   private final int ordinal;

   protected Enum(String name, int ordinal)
   {
        this.name = name;
        this.ordinal = ordinal;
    }
}
```

Methdos :
---------

1) public final String name() : Used to retrieve the name of enum
   constant in String format.

2) public String toString() : It is also used to print/provide the
   enum constant but this method is not final so, we can override

3) public final int ordinal() : It is used to get the order position
   of an enum constant which starts from 0

4) public static T[] values() : It returns array of the enum constant
   in the same order as they were declared in the enum class.

5) public final boolean equals(Object obj) : It is used to compare two    enum constants, Internally it uses == operator only so it will
   compare  the address reference of two enum constants.

6) public static T valueOf(String name) : It is used to convert the
   given String into enum constant, if the given String name is not matching with the corrosponding enum constant then it will throw an exception i.e IllegalArgumentException.

-------------------------------------------------------------------

```
package com.ravi.enum_demo;

enum Bird
{
	PARROT, SPARROW, PEACOCK;  //public static final Bird PARROT = new Bird();

}

public  class Demo
```

```
{
	public static void main(String[] args)
	{
		System.out.println(Bird.PEACOCK);
   }

}
```

-------------------------------------------------------------------

```
package com.ravi.enum_demo;

public class Test1
{
	public static void main(String[] args)
	{
		enum Month
		{
			JANUARY, FEBRUARY,MARCH     //public + static + final
		}

		System.out.println(Month.MARCH);
	}
}
```

We can write an enum inside a method.

-----------------------------------------------------------------

```
package com.ravi.enum_demo;

enum Month
{
	JANUARY,FEBRUARY,MARCH
}
public class Test2
{
	enum Color { RED,BLUE,BLACK }

   public static void main(String[] args)
	{
		enum Week {SUNDAY, MONDAY, TUESDAY }

		System.out.println(Month.FEBRUARY);
		System.out.println(Color.RED);
		System.out.println(Week.SUNDAY);
	}
}
```

We can write an enum inside a class, outside of the class and inside of the method.
---------------------------------------------------------------------

```
//Comparing the constant of an enum

package com.ravi.enum_demo;

public class Test3
{
         enum Color { RED,BLUE }

     public static void main(String args[])
     {
         Color c1 = Color.RED;
         Color c2 = Color.RED;

         if(c1 == c2)
         {
              System.out.println("== Operator");
         }
         if(c1.equals(c2))
         {
               System.out.println("equals method");
         }
     }
}
```

We can compare enum constant by using == operator or equals(Object obj) method of enum class.
--------------------------------------------------------------------

```
package com.ravi.enum_demo;

public class Test4
{
         private enum Season   //private, public, protected, static
         {
         SPRING, SUMMER, WINTER, RAINY;
         }

         public static void main(String[] args)
         {
                  System.out.println(Season.RAINY);
```

```
	}
}
```

An enum defined inside a class can be private, static, default, public and protected, It can't be final and abstract.

---------------------------------------------------------------------

```
//Interview Question
package com.ravi.enum_demo;
class Hello
{
	int x = 100;
}

enum Direction extends Hello
{
	EAST, WEST, NORTH, SOUTH;


}

class Test5
{
	public static void main(String[] args)
	{
		System.out.println(Direction.SOUTH);
	}
}
```

By default every enum extends from java.lang.Enum class so enum can't extend a class explicitly.

--------------------------------------------------------------------

```
//All enums are by default final so can't inherit

package com.ravi.enum_demo;

enum Pizza
{
	SMALL, MEDIUM, BIG;
}
class Test6 //extends Pizza
{
	public static void main(String[] args)
	{
		System.out.println(Pizza.BIG);
	}
}
```

An enum is implicitly final so any class can't extend enum

-------------------------------------------------------------------------

```
//values() to get all the values of enum

package com.ravi.enum_demo;

class Test7
{
	enum Season
	{
	SPRING, SUMMER, WINTER, FALL, RAINY
	}

	public static void main(String[] args)
	{
		Season[] seasons = Season.values();

		for(Season season : seasons)
		{
			System.out.println(season);
		}



	}
}
```

values() method added by compiler, is used to fetch all the enum constants.

-----------------------------------------------------------------------

```
//ordinal() to find out the order position

package com.ravi.enum_demo;

class Test8
{
	static enum Season
	{
	SPRING, SUMMER, WINTER, FALL, RAINY
```

```
        }


        public static void main(String[] args)
        {
                Season seasons[] = Season.values();

                for(Season season : seasons)
                {
                        System.out.println(season.name()+" order position is :"+season.ordinal());
                }
        }
}
```

ordinal() is used to retirn the order position of enum constants.

---------------------------------------------------------------------

```
//We can take main () inside an enum

package com.ravi.enum_demo;
enum Test9
{
        TEST1, TEST2, TEST3;    //Semicolon is compulsory

        public static void main(String[] args)
        {
                System.out.println("Enum  main method");
        }

}
```

After enum constant, if we decalre any member then ; is compulsory

---------------------------------------------------------------------

```
//constant must be in first line of an enum

package com.ravi.enum_demo;

enum Test10
{

        public static void main(String[] args)
        {
                System.out.println("Enum  main method");
        }
```

```
   HR, SALESMAN, MANAGER;

}
```

The code will not compile because enum constant must be in the first line only.

---------------------------------------------------------------------

```
//Writing constructor in enum

package com.ravi.enum_demo;

enum Season
{
	WINTER, SUMMER, SPRING, RAINY, FALL;   //All are object of type enum

	private Season()
	{
		System.out.println("Constructor is executed....");
	}
}
class Test11
{
	public static void main(String[] args)
	{
		System.out.println(Season.WINTER);
		System.out.println(Season.SUMMER);

	}
}
```

Every time enum object will be created, appropriate constructor will be executed.

---------------------------------------------------------------------

```
//Writing constructor with message
package com.ravi.enum_demo;

  enum MySeason
	{
	  SPRING("Pleasant"), SUMMER("UnPleasent"), RAINY("Rain"), WINTER;

      String msg;

      private MySeason(String msg)
```

```
       {
          this.msg = msg;
       }


                private MySeason()
                {
                        this.msg = "Cold";
                }

                public String getMessage()
                {
                        return msg;
                }
        }
class Test12
{
        public static void main(String[] args)
        {
                MySeason seasons[] = MySeason.values();

                for(MySeason season : seasons)
                {
                        System.out.println(season+"  is :"+season.getMessage());
                        System.out.println(season+"  order is :"+season.ordinal());
                        System.out.println("Season name is :"+season.name());
                }
        }
}
```
------------------------------------------------------------------
```
package com.ravi.enum_demo;

import java.util.Scanner;

enum Color
{

        RED, BLUE, PINK
}

public class Test13
{
        public static void main(String[] args)
```

```
            {
                        Scanner sc = new Scanner(System.in);
                        System.out.println("Enter Color Name :");
                        String colorName = sc.next();

                        Color color = Color.valueOf(colorName);
        System.out.println(color);
         sc.close();
            }

}
```

valueOf() used to convert the given String into corrosponding enum object, if enum object is not available throw IllegalArgumentException.

------------------------------------------------------------------------

```
package com.ravi.enum_demo;

public class Test14
{
            enum Day
                        {
                          SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
SATURDAY
                        }

public static void main(String args[])
{
            Day day  = Day.MONDAY;

            switch(day)
            {
             case SUNDAY:
             System.out.println("Sunday");
             break;
             case MONDAY:
             System.out.println("My Monday");
             break;
             default:
             System.out.println("other day");
    }

  }
}
```

```
----------------------------------------------------------------------
package com.ravi.enum_demo;

enum MyColor
{
    RED , BLUE, PINK;

    static
           {
        System.out.println("static block ");
    }

    {
        System.out.println("instance block");
    }

}
public class Test15
{
  public static void main(String[] args)
           {
             System.out.println(MyColor.RED);
           }
}
---------------------------------------------------------------------
```

26-01-2025
------------

Nested classes in java :
------------------------

In java it is possible to define a class (inner class) inside another class (outer class) called nested class.

In the same way, It is also possible to define a class/interface/enum/ annotation/record inside another class/interface/enum/annotation and record as shown below.

Programs that describes all different possible combinations :
-----------------------------------------------------------

```
package com.ravi.neasted_class_demo;

public class Demo1
{
   private class A
   {
   }

   interface B
   {}

   enum C
   {}

   record D()
   {

   }

   //Annotation
   @interface E
   {
   }

}
```

---------------------------------------------------------------

```
package com.ravi.neasted_class_demo;

public interface Demo2
{
          class A
          {
          }

          interface B
          {}

          enum C
          {}

          record D()
          {

          }

          //Annotation
          @interface E
          {
          }
```

```
}
---------------------------------------------------------------
package com.ravi.neasted_class_demo;

public enum Demo3
{
        HR, SALES;

        private class A
          {
          }

          interface B
          {}

          enum C
          {}

          record D()
          {

          }

          //Annotation
          @interface E
          {
          }

}
------------------------------------------------------------
package com.ravi.neasted_class_demo;

public record Demo4()
{
        private class A
          {
          }

          interface B
          {}

          enum C
          {}

          record D()
          {

          }

          //Annotation
          @interface E
          {
          }

}
```
----------------------------------------------------------

```
package com.ravi.neasted_class_demo;

public @interface Demo5
{
        class A
        {
        }

        interface B
        {}

        enum C
        {}

        record D()
        {

        }

        //Annotation
        @interface E
        {
        }
}
```
----------------------------------------------------------------------

Inner classes in Java create a strong encapsulation and  HAS-A  relationship between the inner class and its outer class.

An inner class, .class file will be represented by $ symbol.

Advantages of inner class :
--------------------------------
1) It is a way of logically grouping classes that are only used in one place.

2) It is used to achieve strong encapsulation.

3) It enhance the readability and maintainability of the code because Outer class code is isolated from inner class.

Types of Inner classes in java :
--------------------------------
There are 4 types of inner classes in java, 2 inner classes we can write at class level (static + non static) and two inner classes we can write a method level (local + anonymous)

1) At Class Level
------------------
  a) Non Static Nested class OR Regular class
  b) Static Nested class.

2) At Method Level
-------------------
  c) Method / Method local class
  d) Anonymous inner class

--------------------------------------------------------
Types of nested classes :
--------------------------

```
package com.ravi.neasted_class_demo;

class Ravi{}

public class NestedDemo
{
   public class A  //Nested non static class
   {

   }

   public static class B //Nested static class
   {
   }

   public void method()
   {
             class Local  //Local inner class
             {

             }

             Ravi anonymous = new Ravi()
             {

             };
   }

}
```
-----------------------------------------------------------------------
02-02-2025
-----------
Non Static Nested class OR Regular class
-----------------------------------------
If a non static nested or inner class is defined inside the Outer class but outside of the method then it is called non static nested class.

Example :
---------
```
public class Outer
{
  //Non static nested class
  private class Inner
  {
  }
}
```

In non static nested inner class we can apply private, default (private-package), protected, public, abstract and final.

During the class loading phase an inner class will be loaded with the help of Outer class. If inner class contains non static member

For non static nested inner class, Outer class object is required, We can't create the inner class object directly, Outer class object is required.

It is also known as Regular OR Member class.

---------------------------------------------------------------------

```
//How to create object for inner class and instance block execution flow

package com.ravi.basic;

class FooOuter
{
	 int x = 15;

	{
		System.out.println("Outer class non static block");
	}

	 class Inner
	{

		{
			System.out.println("Inner class non static block");
		}

		public  void m1()
		{
			System.out.println("m1 static method :"+x);
		}
	}
}
public class Example
{
  public static void main(String[] args)
  {
	 FooOuter fo = new FooOuter(); //Outer class object

	 FooOuter.Inner in = fo.new Inner(); //inner class object
	  in.m1();
   }
}
```

Note : Here first of all Outer class non static block will be executed and then only inner class non static block will be executed because first we are creating the object for Outer class and it is required because inner class is a non static member of Outer class.

---------------------------------------------------------------------

```
class Outer
```

```
{
        private int a = 15;

        class Inner
        {
                public void displayValue()
                {
                        System.out.println("Value of a is " + a);
                }
        }
}
public class Test1
{
        public static void main(String... args)
        {
                Outer out = new Outer();

                Outer.Inner in = out.new Inner();

                in.displayValue();

        }
}
```

Note : An inner class can directly access the private data of Outer class.

---------------------------------------------------------------------

```
class MyOuter
{
    private int x = 7;
    public void makeInner()
    {
        MyInner in = new MyInner();
                        System.out.println("Inner y is "+in.y);
        in.seeOuter();
    }

    class MyInner
    {
                    private int y = 15;
        public void seeOuter()
        {
            System.out.println("Outer x is "+x);
        }
    }
}
public class Test2
{
    public static void main(String args[])
    {
        MyOuter m = new MyOuter();
        m.makeInner();
    }
}
```

-----------------------------------------------------------------------

Variable Shadow in Nested class :

---------------------------------

If outer class and inner class variable names are same then inner class variable will be access with inner class object (Variable Shadow bacause in the same scope), If we want to access Outer class variable from inner class then we need OuterClassName.this.variableName.

```
package com.ravi.variable_shadow;

class Outer
{
        private int x = 100;

        public class Inner
        {
                private int x = 200;

                public void access()
                {
                        System.out.println("Inner class x variable is :"+this.x);
                        System.out.println("Outer class x variable is :"+Outer.this.x);
                }
        }
}

public class VariableShadow {

        public static void main(String[] args)
        {
                Outer.Inner in = new Outer().new Inner();
                in.access();

        }
}
--------------------------------------------------------------------
class MyOuter
{
      private int x = 15;
      class MyInner
      {
            public void seeOuter()
            {
                  System.out.println("Outer x is "+x);
            }
      }
}
public class Test3
{
      public static void main(String args[])
      {
                          //Creating inner class object in a single line
           MyOuter.MyInner m = new MyOuter().new MyInner();
                                    m.seeOuter();
      }
}
------------------------------------------------------------------
class MyOuter
{
      static int x = 7;
              class MyInner
          {
            public static void seeOuter()  //MyInner.seeOuter();
            {
                  System.out.println("Outer x is "+x);
            }
         }
}

public class Test4
{
      public static void main(String args[])
      {
          MyOuter.MyInner.seeOuter();
      }
}
-------------------------------------------------------------------
```

```
class Car
{
    private String name;
    private String model;
    private Engine engine;

   public Car(String name, String model, int horsePower)
   {
        this.name = name;
        this.model = model;
        this.engine = new Engine(horsePower);
    }

   //Inner class
   private class Engine
   {
      private int horsePower;

       public Engine(int horsePower)
               {
           this.horsePower = horsePower;
       }

        public void start()
                        {
            System.out.println("Engine started! Horsepower: " + horsePower);
        }

        public void stop()
                {
            System.out.println("Engine stopped.");
        }
    }

    public void startCar()
            {
        System.out.println("Starting " + name + " " + model);
        this.engine.start();
    }

    public void stopCar()
            {
        System.out.println("Stopping " + name + " " + model);
        this.engine.stop();
    }
}
public class Test5
{

    public static void main(String[] args) {

        Car myCar = new Car("Swift", "Desire", 1200);

        myCar.startCar();

        myCar.stopCar();
    }
}
```
-------------------------------------------------------------------
```
class Laptop
{
    private String brand;
    private String model;
    private Motherboard motherboard;

    public Laptop(String brand, String model, String motherboardModel, String chipset)
            {
        this.brand = brand;
        this.model = model;
        this.motherboard = new Motherboard(motherboardModel, chipset);
    }


    public void switchOn()
            {
        System.out.println("Turning on " + brand + " " + model);
        this.motherboard.boot();
    }

   //Motherboard inner class
   public class Motherboard
   {
        private String model;
        private String chipset;
        public Motherboard(String model, String chipset)
                {
            this.model = model;
            this.chipset = chipset;
        }


        public void boot()
                {
            System.out.println("Booting " + brand + " " + model + " with " + chipset + " chipset");
        }
    }
}
public class Test6
{
    public static void main(String[] args)
          {

        Laptop laptop = new Laptop("HP", "ENVY", "IRIS", "Intel");

        laptop.switchOn();
    }
}
---------------------------------------------------------------------
class Person
{
    private String name;
    private int age;
    private Heart heart;

    public Person(String name, int age)
          {
        this.name = name;
        this.age = age;
        this.heart = new Heart();
    }

    public void describe()
          {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Heart beats per minute: " + this.heart.getBeatsPerMinute());
    }
    // Inner class representing the Heart
    private class Heart
            {
        private int beatsPerMinute;

        public Heart()
                    {
            this.beatsPerMinute = 72;
        }

        public int getBeatsPerMinute()
                    {
            return this.beatsPerMinute;
        }

        public void setBeatsPerMinute(int beatsPerMinute)
                    {
            this.beatsPerMinute = beatsPerMinute;
        }
    }
}
public class Test7
{
    public static void main(String[] args)
            {
        Person person = new Person("Virat", 30);
        person.describe();
    }
}
--------------------------------------------------------------------
class University
{
    private String name;

    public University(String name)
            {
        this.name = name;
    }

    public void displayUniversityName()
            {
        System.out.println("University Name: " + name);
    }

    public class Department
        {
        private String name;
        private String headOfDepartment;

        public Department(String name, String headOfDepartment)
                {
            this.name = name;
            this.headOfDepartment = headOfDepartment;
        }

        // Method to display department details
        public void displayDepartmentDetails()
                {
                        displayUniversityName();
            System.out.println("Department Name: " + name);
            System.out.println("Head of Department: " + headOfDepartment);

        }
    }
}
public class Test8
{
    public static void main(String[] args)
        {

        University university = new University("JNTU");

       University.Department cs = university.new Department("Computer Science", "Dr. John");

        University.Department ee = university.new Department("Electrical Engineering", "Dr.
Scott");


        cs.displayDepartmentDetails();
        ee.displayDepartmentDetails();
    }
}
```
---------------------------------------------------------------------
```
class OuterClass
{
          }


          private int x = 200;

          class InnerClass
          {
                    public void display()  //Inner class display method
                    {
                    System.out.println("Inner class display method");
                    }

                    public void getValue()
                    {
                              display();
                              System.out.println("Can access outer private var :"+x);
                    }
          }

                    public void display()  //Outer class display method
                    {
                              System.out.println("Outer class display");
                    }
}
public class Test9
{
          public static void main(String [] args)
          {
                    OuterClass.InnerClass inobj = new OuterClass().new InnerClass();
                    inobj.getValue();

                    System.out.println("...............");

                    new OuterClass().display();

          }
}
---------------------------------------------------------------------
class OuterClass
{
          int x;
          private class InnerClass  //private, def, prot, public, abstract
          {                         //final
                    int x;
          }
}
```

```
public class Test10
{
}
```
---------------------------------------------------------------------
09-02-2025
-----------
static nested inner class :
--------------------------
It is class which we can declare at class level.

If we decalre a class inside an outer class with static modifier then it is called static nested inner class.

Example :
----------
```
public class Outer
{
  static class Inner  //static nested inner class
  {
  }

}
```

In order to access the static nested inner class, Outer class object is not required.

If a static nested inner class contains non static member then we need to create the object static nested class but on the other hand if static nested inner class contains only static member then inner class object is not required.

A static nested inner class represents static area so it can't access non static member of Outer class.
--------------------------------------------------------------------
```
//static nested inner class
class BigOuter
{
    static class Nest   //static nested inner class
    {
        void go()  //Non static method of nested inner class
        {
            System.out.println("Hello welcome to static nested class");
        }
    }
}
```

```
class Test11
{
    public static void main(String args[])
    {
        BigOuter.Nest n = new BigOuter.Nest();
        n.go();
    }
}
-------------------------------------------------------------------
class Outer
{
        static int x = 15;

        static class Inner
        {
                void msg()
                    {
                        System.out.println("x value is  "+x);
                    }
        }
}
class Test12
{
        public static void main(String args[])
        {
                Outer.Inner obj=new Outer.Inner();
                obj.msg();
        }
}
```

We can static static member of Outer class.

```
-------------------------------------------------------------------
//Static block of Outer and Inner class

class Outer
{
        static int x = 100;
        static
        {
                System.out.println("Outer class static block");
        }

        static class Inner
```

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

      public static void m1()
		{
			System.out.println("Static Method of inner class "+x);
		}

	}
}
public class Demo
{
	public static void main(String[] args)
	{
	  Outer.Inner.m1();
	}
}
-----------------------------------------------------------------------
class Outer
{
	static int x = 100;
	static
	{
		System.out.println("Outer class static block");
	}

	static class Inner
	{
		static
	   {
		System.out.println("Inner class static block");
	   }

      public static void m1()
		{
			System.out.println("Static Method of inner class "+x);
		}

	}
}
```

```
public class Demo
{
	public static void main(String[] args)
	{
	  Outer.Inner.m1();
	}
}
```
---------------------------------------------------------------------
```
class Outer
{
	 int x=15;  //Non static variable

	 static class Inner
	 {
			void msg()
				{
					System.out.println("x value is  "+x);
				}
	 }
}
class Test14
{
	public static void main(String args[])
	{
		Outer.Inner obj=new Outer.Inner();
		obj.msg();
	}
}
```

Note : We will get error because from static area (static nested inner class) we can't access the non static member directly.

---------------------------------------------------------------------

Declaring the classes at Method Level :
==========================================

At Method level we can declare two classes which are as follows :

1) Local OR Method Local inner class
2) Anonymous Inner class

Local OR Method Local Inner class :
----------------------------------

If we define a class with class name inside the method body then it is called Local OR Method Local Inner class

Example :
----------
```
public void method()
{
   class Local    //Local OR Method Local Inner class
   {
   }
}
```

The scope of Method level inner class is within the same method body that means it will be executed in the same method Stack Frame.

We can't access or use the method local inner class outside of the method body so basically it is used perform some operation within the method.

We can't apply any kind of access modifier on local inner class except abstract and final.

```
------------------------------------------------------------------
//program on method local inner class
class Outer
{
      private String x = "Outer class private data";

      public void doSttuff()
      {
           String z = "local variable";

           class MyInner  //Only final and abstract applicable
           {
                 public void seeOuter()
                 {
                      System.out.println("Outer x is "+x);
                      System.out.println("Local variable z is : "+z);
                 }
           }
                          MyInner mi = new MyInner();
            mi.seeOuter();

      }

}
public class Test15
```

```
{
      public static void main(String args[])
      {
            Outer m = new Outer();
            m.doSttuff();
      }
}
```
--------------------------------------------------------------------

```
class Outer
{
   private int x = 100;

  public void m1()
  {
      class Inner
      {
                      int x = 200;

          public void m1()
                      {
                                  System.out.println("Inner class value is :"+this.x);
                                  System.out.println("Outer class value is :"+Outer.this.x);
                      }
            }
            Inner i = new Inner();
            i.m1();
  }

}
public class Demo
{
            public static void main(String[] args)
            {
              Outer out = new Outer();
              out.m1();
            }
}
```
--------------------------------------------------------------------

```
class Outer
{
   private int x = 100;

  public void m1()
```

```
 {
    class Inner
    {
                    int x = 200;

        public void m1()
                    {
                            System.out.println("Inner class value is :"+this.x);
                            System.out.println("Outer class value is :"+Outer.this.x);
                    }
         }
 }
  Inner i = new Inner();
         i.m1();

}
public class Demo
{
        public static void main(String[] args)
        {
          Outer out = new Outer();
          out.m1();
        }
}
```

Note : Method Local inner class will be accessible within the same method body only.

------------------------------------------------------------------

Anonymous inner clas :

---------------------

If we decalre a class inside a method without any name then it is called Anonymous inner class.

The main purpose of anonymous inner class to extend a class OR to implement an interface that means to create a sub type.

The anonymous inner class (class without any name) declaration and object creation both are done in the same line at the time of declaration of anonymous inner class.

We can create an object only one time for anonymous inner class so we can represent as a singleton class.

A normal class can extend a class as well as implement many number of interfaces but for Anonymous inner class we can extend either a single or can implement only a single interface.

In an anonymous inner class we can write static, non static block, static and non static method, we can't write constructor as well as we can't decalre as an abstract class.

---------------------------------------------------------------------

```
@FunctionalInterface
interface Vehicle
{
	void move();  //SAM(Single Abstract Method)
}

class Test18
{
	public static void main(String[] args)
	{
		//Anonymous inner class
		Vehicle car = new Vehicle()
		{
			@Override
			public void move()
			{
				System.out.println("Moving with Car...");
			}

		};
		car.move();

      Vehicle bike = new Vehicle()
		{
			@Override
			public void move()
			{
				System.out.println("Moving with Bike...");
			}
		};
		bike.move();
	}
}
```

---------------------------------------------------------------------

```
class Test19
{
	public static void main(String[] args)
	{
       //Anonymous class with Runnable
		Runnable r1 = new Runnable()
```

```
                {
                        @Override
                        public void run()
                        {
            System.out.println("Run method implementation inside Runnable");
                        }
                };
          Thread obj = new Thread(r1);
                         obj.start();



        //Anonymous Thread class with reference
                Thread t1 = new Thread()
                {
                        @Override
                        public void run()
                        {
            System.out.println("Anonymous Thread class with reference...");
                        }
                };
                t1.start();

        //Anonymous Thread class without reference
        new Thread()
                {
                        @Override
                        public void run()
                        {
            System.out.println("Anonymous Thread class without reference...");
                        }
                }.start();
          }
}
```
---------------------------------------------------------------------