Inner class

※ Type of Inner Class:

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

* ya par jo super class bola ha wo "outer-class" hai and sub-class bola hai wo "Inner-class" hai. Ya sabhi self-understanding ka liya hai.

Inner class

Nested Inner class

```
class Outer
{
  ① int x=10;

  ② class Inner
    {
      int y=20;
      void innerDisplay()
      {
        s.o.p(x);
        s.o.p(y);
      }
    }

  ③ void outerDisplay()
    {
      Inner i=new Inner();
      i.innerDisplay();
      S.o.p(i.y);
    }
}
```

--> Nested Inner class mai ham class ka andar hi one or more than one class ko define karta hai.

--> Jo bhi chij sub-class ka andar hota hai uska use ham class ka bhar object bna kar sakta hai. Lekin jo bhi item super class mai declare hai usa ham kabhi bhi kahi bhi direct use kar sakta hai.

```
class Test
{
  p.s.v.main()
  {
    Outer o=new Outer();
    o.outerDisplay();

    Outer.Inner i=new Outer().new Inner();
  }
}
```

--> Agar hma Inner class ka object ko direct call karna ho main method mai to ya uska mth. hai.

Inner class

Local Inner Class

--> Jab ham method ke andhar kisi class ko define karta hai ushe "local inner class bolta hai ".

--> ise inner class ko ham sirf ushe class kai method mai access kar sakta hai jis mai method define hai (ise case mai class outer{}). Agar outside method access karna ka try krenga to error aata hai.

--> method mai local inner class ko call karna ka liya class ka object normaly create karta hai.

```
class Outer
{
    void Display()
    {
        class Inner
        {
            void innerDisplay()
            {
                S.O.P("Hello");
            }
        }

        Inner i=new Inner();
        i.innerDisplay();
    }
}
```

Inner class

Anonymous

--> Ise class ka mostly use abstract class ya phir interface mai hota hai.

--> Jab abstract class ka object create karta hai then use waqt ham abstract class ka method ko override bhi kar deta hai to ek unknown class create hota hai by default jisa anonymous class bolta hai.

--> Same interface ka case mai kaam karta hai.

```
abstract class My
{
  abstract void display();
}
```

```
class Outer
{
  public void meth()
  {
    My m=new My()
    {
      public void display()
      {
        s.o.p("Hello");
      }
    };
    m.display();
  }
}
```

Inner class

Anonymous

```
interface My
{
  void display();
}
```

--> Same theory jo upper wala slide mai hai.

```
class Outer
{

  public void meth()
  {

    My m=new My()
    {
        public void display()
        {
          s.o.p("Hello");
        }
    };

    m.display();
  }
}
```

Inner class

```
class Outer
{
    static int x=10;
        int y=20;

    static class Inner
    {
        void display()
        {
            s.o.p(x);
            s.o.p(y);
        }
    }
}
```

```
class Test
{
    p.s.v.m()
    {
        Outer.Inner i=new Outer.Inner();
        i.display();
    }
}
```

static class ka object create karna ka tareka.

--> Ek satic class jo kis class ki ander define hota hai usa static class bolta hai.

--> Outer class ka non-static variable ko ham static class ka andhar access nahi kar sakta hai.

--> Outer class ka instance ko create kia bina ham main class mai direct object create kar sakta hai static class ka.

Java

Explain type of inner classes in java.

In Java, **inner classes** are classes that are defined inside another class. They provide a way to logically group classes that are only used in one place, helping to maintain better encapsulation and organization. Inner classes can access the members (both fields and methods) of their enclosing class, including private ones. Java supports several types of inner classes, each with distinct behavior and use cases.

Types of Inner Classes in Java:

1. **Nested (Static) Inner Class**
2. **Non-Static Inner Class (also known as Member Inner Class)**
3. **Local Inner Class**
4. **Anonymous Inner Class**

---

1. Nested (Static) Inner Class

A **static inner class**, also called a **nested class**, is declared with the `static` modifier. It behaves like a static member of the enclosing class and can be accessed without creating an instance of the outer class. However, it cannot directly access the instance variables or methods of the enclosing class, unless through an explicit reference.

- **Key Features**:
  - Can be instantiated without creating an instance of the outer class.
  - Can access the `static` members of the outer class.
  - Cannot access the non-static members of the outer class directly.
- **Syntax**:

```java
public class OuterClass {
    static class NestedStaticClass {
        void display() {
            System.out.println("This is a static nested class.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass.NestedStaticClass nested = new OuterClass.NestedStaticClass();  // No need for an instance of OuterClass
        nested.display();
```

```
    }
}
```

---

2. Non-Static Inner Class (Member Inner Class)

A **non-static inner class**, also known as a **member inner class**, is associated with an instance of the outer class. It can access all members of the outer class, including private members. To create an instance of a member inner class, you first need to create an instance of the outer class.

- **Key Features**:
  - Can access all members (including private) of the outer class.
  - Must be instantiated within an instance of the outer class.
  - Does not have the `static` modifier.
- **Syntax**:

```java
public class OuterClass {
    class InnerClass {
        void display() {
            System.out.println("This is a member inner class.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();  // Requires an
instance of OuterClass
        inner.display();
    }
}
```

---

3. Local Inner Class

A **local inner class** is a class defined within a method, constructor, or block of code. Its scope is restricted to the method or block where it is declared. Local inner classes can access the members of the outer class, and they can also access local variables of the method, but only if those variables are declared `final` or are effectively final (i.e., they are not changed after their initialization).

- **Key Features**:
  - Defined inside a method or block.
  - Can access local variables of the method if they are final or effectively final.
  - Has no access modifier since it's local to a method.
- **Syntax**:

```java
public class OuterClass {
    void outerMethod() {
        class LocalInnerClass {
            void display() {
```

```
                System.out.println("This is a local inner class.");
            }
        }
        LocalInnerClass local = new LocalInnerClass();
        local.display();
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.outerMethod();  // Calls method that defines and uses LocalInnerClass
    }
}
```

---

4. Anonymous Inner Class

An **anonymous inner class** is a special type of inner class that is declared and instantiated at the same time. It is used when you need to override or implement methods of a class or an interface without explicitly defining a subclass. Anonymous inner classes are typically used in event handling or passing custom behavior to methods.

- **Key Features**:
  - Has no name and is instantiated directly.
  - Can be used to implement an interface or extend a class.
  - Commonly used when only one instance of the class is required.
- **Syntax**:

```java
public class OuterClass {
    public void display() {
        System.out.println("Outer class method");
    }

    public void createAnonymousClass() {
        OuterClass anonymous = new OuterClass() {  // Anonymous inner class
            @Override
            public void display() {
                System.out.println("Anonymous inner class method");
            }
        };
        anonymous.display();
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.createAnonymousClass();  // Calls anonymous inner class method
    }
}
```

Summary of Inner Classes

Type	Modifier	Access to Outer Class Members	Instance Creation	Common Use
Static Nested Class	`static`	Can access `static` members only	`OuterClass.StaticNestedClass obj = new OuterClass.StaticNestedClass();`	Utility or helper classes
Member Inner Class	No modifier	Can access all members	`OuterClass.InnerClass obj = outer.new InnerClass();`	Closely related classes sharing state
Local Inner Class	No modifier	Can access all members, local variables must be `final` or effectively final	Defined and instantiated inside a method or block	Localized behavior within a method
Anonymous Inner Class	No name, implicit	Can access all members	Defined and instantiated in a single expression	Single-use implementations, event handling

Inner classes allow for better encapsulation, and they enable more readable and maintainable code by logically grouping classes that only make sense within a specific context.