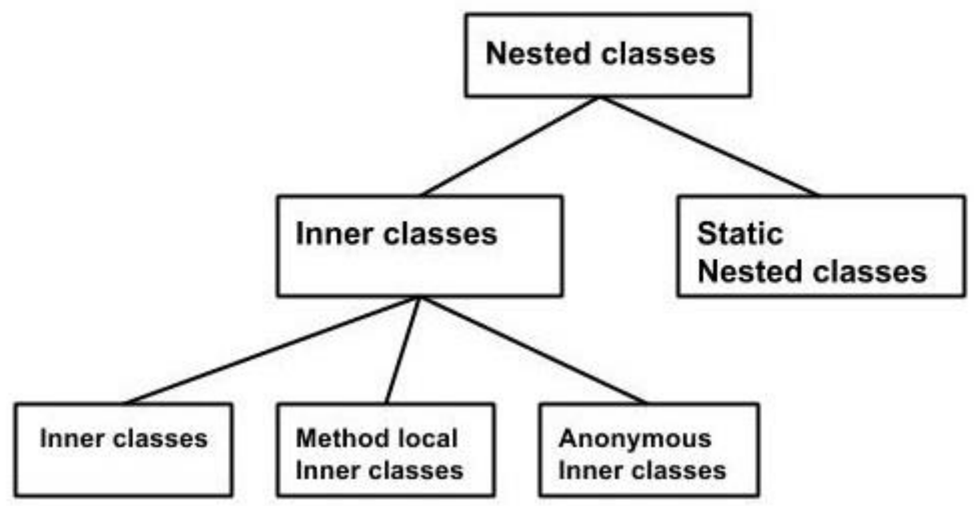
Nested Class in Java

In Java, inner class refers to the class that is declared inside **class** or **interface** In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called **the nested class**, and the class that holds the inner class is called **the outer class**.



There are certain advantages associated with inner classes are as follows:

- **Making code clean and readable and adding more secure for data.**
- **Private methods of the outer class can be accessed, so bringing a new dimension and making it closer to the real world.**

There are basically four types of inner classes in java.

- **Nested Inner Class**
- **Method Local Inner Classes**
- **Static Nested Classes**
- **Anonymous Inner Classes**

Let us discuss each of the above following types sequentially in-depth alongside a clean java program.

Inner Classes (Non-static Nested Classes)

Inner classes are a **security mechanism** in Java. We know a class cannot be associated with the **access modifier private**, but if we have the class as a member of other class, **then the inner class can be made private**. And this is also used to access the private members of a class.

Inner classes are of three types depending on how and where you define them. They are: –

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

Inner Class

Creating an inner class is **quite simple**. You just need to write a class within a class. Unlike a class, an inner class can be **private** and once you declare an inner class private, **it cannot be accessed from an object outside the class.**

Following is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.

Example1: for nested inner class

Java Program to Demonstrate Nested class

```
class Outer {
   // Class 2
   // Simple nested inner class
   class Inner {
      // show() method of inner class
      public void show()
      { // Print statement
         System.out.println("In a nested class method");
      }
   }
}
// Class 3  Main class
class Main {
    public static void main(String[] args)
    {
        // Note how inner class object is created inside main()
         Outer.Inner in = new Outer().new Inner();
        in.show();       // Calling show() method over above object created
    }
}
```

Output
In a nested class method

Example 2:

```
class Outer_Demo {
   int num;
   private class Inner_Demo {    // inner class
      public void print() {
         System.out.println("This is an inner class");
      }
   }
   void display_Inner() {
      Inner_Demo inner = new Inner_Demo();
      inner.print();
   }
}
public class My_class {
   public static void main(String args[]) {
      Outer_Demo outer = new Outer_Demo();      // Instantiating the outer class
         outer.display_Inner();// Accessing the display_Inner() method.
   }
}
```

Output
This is an inner class.

Example 3:

```
class Outer_Demo {
   // private variable of the outer class
   private int num = 175;
      // inner class
   public class Inner_Demo {
    public int getNum() {
      System.out.println("This is the getnum method of the inner class");
      return num;
    }
  }
}
public class My_class2 {
  public static void main(String args[]) {
    // Instantiating the outer class
    Outer_Demo outer = new Outer_Demo();
        // Instantiating the inner class
    Outer_Demo.Inner_Demo inner = outer.new Inner_Demo();
    System.out.println(inner.getNum());
  }
}
```

Output

```
This is the getnum method of the inner class: 175
```

Note: We cannot have a static method in a nested inner class **because an inner class is implicitly associated with an object of its outer class** so it cannot define any static method for itself. For example, the following program doesn't compile.

Example2: for nested inner class

```
// Class 1: Outer class
class Outer {         // Method defined inside outer class
   void outerMethod()
   {  // Print statement
      System.out.println("inside outerMethod");
   }
    // Class 2:  Inner class
   class Inner {              // Main driver method
      public static void main(String[] args)
      {
         System.out.println("inside inner class Method");
      }
   }
}
```

Output: **Error**

Method Local Inner Classes

Inner class can be declared within a method of an outer class which we will be illustrating in the below example where Inner is an inner class in outerMethod().

Example 1: Java Program to Illustrate Inner class can be declared within a method of outer class

```
 // Class 1 Outer class
class Outer {
   void outerMethod()                  // Method inside outer class
    {
        System.out.println("inside outerMethod");          // Print statement
        // Class 2  Inner class It is local to outerMethod()
         class Inner {
            void innerMethod()  // Method defined inside inner class
            {
                // Print statement whenever inner class is called
                System.out.println("inside innerMethod");
            }
        }
     Inner y = new Inner();          // Creating object of inner class
      y.innerMethod();     // Calling over method defined inside it
    }
}
// Class 3  Main class
class Test {
    // Main driver method
    public static void main(String[] args)
    {
        // Creating object of outer class inside main() method
        Outer x = new Outer();
        // Calling over the same method as we did for inner class above
        x.outerMethod();
    }
}
```
Output

inside outerMethod

inside innerMethod

Method Local inner classes can't use a local variable of the outer method until that local variable is not declared as final. For example, the following code generates a compiler error.

Note: "x" is not final in **outerMethod()** and **innerMethod()** tries to access it.

Example 2:

```
class Outer {
  void outerMethod() {
    int x = 98;
    System.out.println("inside outerMethod");
     class Inner {
      void innerMethod() {
        System.out.println("x= "+x);
      }
    }
    Inner y = new Inner();
    y.innerMethod();
  }
}
class test {
  public static void main(String[] args) {
    Outer x=new Outer();
    x.outerMethod();
  }
}
```
Output

inside outerMethod

x= 98

Note: Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in method local inner class.

But the following code compiles and runs fine (Note that x is final this time)

Example 3

```
class Outer {
  void outerMethod() {
    final int x=98;
    System.out.println("inside outerMethod");
    class Inner {
      void innerMethod() {
        System.out.println("x = "+x);
      }
    }
    Inner y = new Inner();
    y.innerMethod();
  }
}
class test {
   public static void main(String[] args){
    Outer x = new Outer();
    x.outerMethod();
   }
}
```

Output

```
inside outerMethod
x = 98
```

Note:

The main reason we need to declare a local variable as a final is that the local variable lives on the stack till the method is on the stack but there might be a case the object of the inner class still lives on the heap.

Method local inner class can't be marked as private, protected, static, and transient but can be marked as abstract and final, but not both at the same time.

Static Nested Classes

A static inner class is a nested class which is a **static member of the outer class**. It can be accessed without **instantiating** the outer class, using other static members. Just like static members, **a static nested class does not have access to the instance variables and methods of the outer class**. The syntax of static nested class is as follows: −

Syntax

```
class MyOuter {
   static class Nested_Demo {
   }
}
```

Example

```
// Java Program to Illustrate Static Nested Classes  Importing required classes

import java.util.*;
// Class 1  Outer class
class Outer {
    // Method
    private static void outerMethod()
    {
        // Print statement
        System.out.println("inside outerMethod");
    }
    // Class 2 Static inner class
    static class Inner {
        public static void display()
        {
            // Print statement
            System.out.println("inside inner class Method");
            // Calling method insid main() method
            outerMethod();
        }
    }
}
 // Class 3 Main class
class main {
     // Main driver method
    public static void main(String args[])
    {
        Outer.Inner obj = new Outer.Inner();
         // Calling method via above instance created
        obj.display();
    }
}
```

Outpinside inner class Method
inside outerMethod

Anonymous Inner Classes

An inner class declared without a class name is known as an anonymous inner class. **In case of anonymous inner classes, we declare and instantiate them at the same time.** Generally, **they are used whenever you need to override** the method of a class or an interface. The syntax of an anonymous inner class is as follows –

Syntax

```
AnonymousInner an_inner = new AnonymousInner() {
   public void my_method() {
      ........
      ........
   }
};
```

They are created in two ways:-

As a subclass of the specified type

As an implementer of the specified interface

Way 1: As a subclass of the specified type

Example: Java Program to Illustrate Anonymous Inner classes Declaration Without any Name

```
// Class 1 Helper class
class Demo {
    void show()
    {
        System.out.println("i am in show method of super class");
    }
}
 // Class 2 Main class
class Flavor1Demo {
    //  An anonymous class with Demo as base class
    Static Demo d = new Demo() {
        // Method 1
        // show() method
        void show()
        {
            // Calling method show() via super keyword
            // which refers to parent class
            super.show();
            // Print statement
            System.out.println("i am in Flavor1Demo class");
        }
    };
     // Method 2 Main driver method
    public static void main(String[] args)
    {
        // Calling show() method inside main() method
        d.show();
    }
}
```

Output

```
i am in show method of super class
i am in Flavor1Demo class
```

In the above code, we have two classes Demo and Flavor1Demo. Here demo act as a **super-class** and the **anonymous class acts as a subclass**, both classes have a method show(). In anonymous class show() method is **overridden**.

Way 2: As an implementer of the specified interface

Example:

```
// As an implementer of Specified interface
 // Interface
interface Hello {
     // Method defined inside interface
    void show();
}
 // Main class
class GFG {
     // Class implementing interface
  static  Hello h = new Hello() {
        public void show()
        {
            // Print statement
            System.out.println("i am in anonymous class");
        }
    };
    public static void main(String[] args)
    {
                h.show();
    }
}
```

Output

```
i am in anonymous class
```

Output explanation:

Note:

In the above code, we create an object of anonymous inner class but this anonymous inner class is an implementer of the interface Hello. Any anonymous inner class can implement only one interface at one time. It can either extend a class or implement an interface at a time.