**Nested Class**

A class defined within another class is known as Nested class. The scope of the nested class is bounded by the scope of its enclosing class.

*Syntax:*

class Outer{

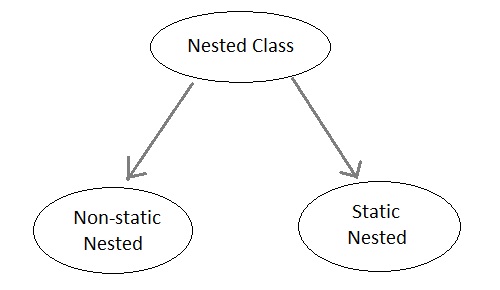
//class Outer members

class Inner{

//class Inner members

}

} //closing of class Outer



**Advantages of Nested Class**

1. It is a way of logically grouping classes that are only used in one place.
2. It increases encapsulation.
3. It can lead to more readable and maintainable code.

If you want to create a class which is to be used only by the enclosing class, then it is not necessary to create a separate file for that. Instead, you can add it as **"Inner Class"**

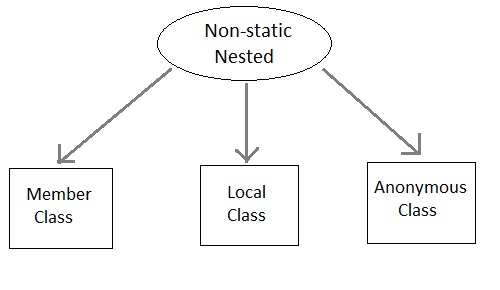
**Static Nested Class**

If the nested class i.e the class defined within another class, has static modifier applied in it, then it is called as **static nested class**. Since it is, static nested classes can access only static members of its outer class i.e it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested class is rarely used.

**Non-static Nested class**

Non-static Nested class is the most important type of nested class. It is also known as **Inner class**. It has access to all variables and methods of Outer class including its private data members and methods and may refer to them directly. But the reverse is not true, that is, Outer class cannot directly access members of Inner class. Inner class can be declared private, public, protected, or with default access whereas an Outer class can have only public or default access.

One more important thing to notice about an Inner class is that it can be created only within the scope of Outer class. Java compiler generates an error if any code outside Outer class attempts to instantiate Inner class directly.



A non-static Nested class that is created outside a method is called **Member inner class**.

A non-static Nested class that is created inside a method is called **local inner class**. If you want to invoke the methods of local inner class, you must instantiate this class inside the method. We cannot use private, public or protected access modifiers with local inner class. Only abstract and final modifiers are allowed.

**Example of Inner class(Member class)**

class Outer

{

public void display()

{

Inner in=new Inner();

in.show();

}

class Inner

{

public void show()

{

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

}

}

}

class Test

{

public static void main(String[] args)

{

Outer ot=new Outer();

ot.display();

}

}

**Output :**

Inside inner

**Example of Inner class inside a method(local inner class)**

class Outer

{

int count;

public void display()

{

for(int i=0;i<5;i++)

{

class Inner //Inner class defined inside for loop

{

public void show()

{

System.out.println("Inside inner "+(count++));

}

}

Inner in=new Inner();

in.show();

}

}

}

class Test

{

public static void main(String[] args)

{

Outer ot=new Outer();

ot.display();

}

}

**Output :**

Inside inner 0

Inside inner 1

Inside inner 2

Inside inner 3

Inside inner 4

**Example of Inner class instantiated outside Outer class**

class Outer

{

int count;

public void display()

{

Inner in=new Inner();

in.show();

}

class Inner

{

public void show()

{

System.out.println("Inside inner "+(++count));

}

}

}

class Test

{

public static void main(String[] args)

{

Outer ot=new Outer();

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

in.show();

}

}

**Output :**

Inside inner 1

**Annonymous class**

A class without any name is called **Annonymous class**.

interface Animal

{

void type();

}

public class ATest {

public static void main(String args[])

{

Animal an = new Animal(){ //Annonymous class created

public void type()

{

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

}

};

an.type();

}

}

**Output :**

Annonymous animal

Here a class is created which implements **Animal** interace and its name will be decided by the compiler. This annonymous class will provide implementation of **type()** method.