A nested class is a class, which is defined inside another class. An outer class
Outer classes allow you to logically group together classes that are only used in one place, thus increasing the
degree of encapsulation and creates more readable and maintainable code.
Nested classes in Java are divided into two categories: static and
non-static.
Non-static classes:
Non-static classes are associated with an instance of the enclosing class and have direct access to the methods of the enclosing class.
Non-static classes have direct access to the methods and fields of the enclosing class. Furthermore, as an
inner class is associated with an instance of the outer class, it cannot define static members itself.
no static members themselves. Let's look at the non-static class types.
Local inner classes
Local inner classes are classes defined in a block, i.e. a group of zero or more statements in balanced braces.
They have zero or more statements in balanced braces. In general, these classes are
defined in a method, but you can define a local class inside any block.
block.
A Java inner class can be implemented like this:

OuterClass outerObject = new OuterClass(); OuterClass.InnerClass innerObject = outerObject.new InnerClass();

1. local inner class

If a class is defined in the body of a method, it is called a local inner class.
OuterClass outerObject = new OuterClass() ;
OuterClass.InnerClass innerObject = outerObject.new InnerClass() ;
Since the local inner class is not associated with Object, we cannot use the access modifiers private,
public, or protected with it. The only modifiers allowed are
abstract or final.
A local inner class can access all members of the enclosing class and final local variables in the scope it occupies.
A local inner class can access all members of the enclosing class and final local variables in the scope it occupies. In addition, it can also access a
In addition, it can also access a non-final local variable of the method in which it is defined, but it cannot

modify them. So if you try to print the value of a non-final local variable, you will be allowed to do so.
allowed, but if you try to change its value from a local method of the inner class,
you will get a compile-time error.

An example:

```
public class MainClass {
    private String s_main_class;
    public void print() {
    String s_print_method = "";
    // local inner class inside the method
    class Logger {
    // able to access enclosing class variables
    String name = s_main_class;
    // able to access non-final method variables
    String name1 = s_print_method;
    public void foo() {
    String name1 = s_print_method;
    // Below code will throw compile time error:
    // Local variable s_print_method defined in an
     // enclosing scope must be final or effectively
     // final
    // s_print_method= ":";
    We can define a local inner class inside any block too, such as static
block,
    if-else block, etc. However, in this case, the scope of the class will
be very
    limited.
    Implementation:
    }
    }
    // instantiate local inner class in the method to use
    Logger logger = new Logger();
    }
    }
```

We can also define a local inner class inside any block, like the static block,
if-else block, etc. However, in this case, the scope of the class will be very limited.
limited.
Implementation :

```
public class MainClass {
    static {
    class Foo {
    }
    Foo f = new Foo();
    }
    public void bar() {
    if(1 < 2) {
    class Test {
    }
    Test t1 = new Test();
    }
    // Below will throw error because of the scope of the
     Class as Isaid it is very limited.
    Test t = new Test();
    Foo f = new Foo();
    }
    }
```

2. Anonymous inner class

An unnamed local inner class is known as an anonymous inner class. An anonymous class
An anonymous class is defined and instantiated in a single declaration.
An anonymous inner class always extends a class or implements an interface.
Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class.
constructor for an anonymous class.
Anonymous inner classes are only accessible where they are defined.
It's a bit difficult to define how to create an anonymous inner class, we'll see
we will see its use in real-time in the test program below.

Some examples:

```
//Using Class:
abstract class Person{
abstract void eat();
}
class TestAnonymousInner{
public static void main(String args[]){
Person p=new Person(){
```

```
void eat(){System.out.println("nice fruits");}
 };
p.eat();
}
}
interface Eatable{
void eat();
}
class TestAnnonymousInner1{
public static void main(String args[]){
 Eatable e=new Eatable(){
 public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```