



NESTED CLASS

Tanjina Helaly

NESTED CLASS

- **Java inner class** or nested class is a class i.e. declared **inside** a class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.
- Structure

```
class Java_Outer_class{  
    //code  
    class Java_Inner_class{  
        //code  
    }  
}
```



ADVANTAGE OF JAVA NESTED CLASSES

- There are few advantages of inner classes in java. They are as follows:

- 1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
- 2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization**: It requires less code to write.
- 4) **Encapsulation**: it increases encapsulation. Inner class can be private. Also inner class can access the private member of outer class.



TYPES OF NESTED CLASSES

- There are two types of nested classes.
 - Non-static nested class(inner class)
 - a)Member inner class
 - b)Anonymous inner class
 - c)Local inner class
 - Static nested class

| Type | Description |
|------------------------------|--|
| <u>Member Inner Class</u> | A class created within class and outside method. |
| <u>Anonymous Inner Class</u> | A class created for implementing interface or extending class. Its name is decided by the java compiler. |
| <u>Local Inner Class</u> | A class created within method. |
| <u>Static Nested Class</u> | A static class created within class. |
| <u>Nested Interface</u> | An interface created within class or interface. |

MEMBER INNER CLASS



INNER CLASS

- A non-static class that is created inside a class but outside a method is called member inner class.
- Example:

```
class TestMemberOuter1{  
    private int data=30;  
    class Inner{  
        void msg(){  
            System.out.println("data is "+data);  
        }  
    }  
    public static void main(String args[]){  
        TestMemberOuter1 obj=new TestMemberOuter1();  
        TestMemberOuter1.Inner in=obj.new Inner();  
        in.msg();  
    }  
}
```

- msg() method in member inner class - accessing the private data member of outer class.



INNER CLASS

- To access inner class's member from outer class, you need to access via object of inner class.

```
public class TestMemberOuter {  
    public void show(){  
        System.out.println("Show method");  
        Inner inner = new Inner();  
        inner.msg();           // Can access private member of inner class  
        System.out.println("--End of Show method--");  
    }  
  
    class Inner{  
        private void msg(){  
            System.out.println("Inner Method");  
        }  
    }  
  
    public static void main(String[] args) {  
        TestMemberOuter tmo = new TestMemberOuter();  
        tmo.show();  
  
        TestMemberOuter.Inner in = tmo.new Inner();  
        in.msg();  
    }  
}
```

Output:

```
Show method  
Inner Method  
--End of Show method--  
Inner Method
```



INNER CLASS - SHADOWING

○ Accessing shadowed variable

```
public class TestMemberOuter {  
    int x = 10;  
  
    class Inner{  
        int x = 20;  
        public void show(int x){  
            System.out.println("Parameter: " + x);  
            System.out.println("Inner Variable: " + this.x);  
            System.out.println("Outer Variable: " + TestMemberOuter.this.x);  
        }  
    }  
  
    public static void main(String[] args) {  
        TestMemberOuter tmo = new TestMemberOuter();  
        TestMemberOuter.Inner in = tmo.new Inner();  
        in.show(30);  
    }  
}
```

Output:

Parameter: 30
Inner Variable: 20
Outer Variable: 10



ANONYMOUS INNER CLASS



ANONYMOUS CLASS

- An **anonymous class** is defined and instantiated in a single succinct expression using the new operator.
- While a local **class** definition is a statement in a block of **Java** code,
- an **anonymous class** definition is an expression, which means that it can be included as part of a larger expression, such as a method call.



ANONYMOUS CLASS

- Java Anonymous inner class can be created by two ways:
 - Class (may be abstract or concrete).
 - Interface
- An anonymous class must implement all the abstract methods in the super class or the interface.
- An anonymous class always uses the default constructor from the super class to create an instance.



ANONYMOUS CLASS

- The anonymous class expression consists of the following:
 - The new operator
 - The name of an interface to implement or a class to extend.
 - Parentheses that contain the arguments to a constructor, just like a normal class instance creation expression.
 - **Note:** When you implement an interface, there is no constructor, so you use an empty pair of parentheses, as in this example.
 - A body, which is a class declaration body. More specifically, in the body, method declarations are allowed but statements are not.
- Because an anonymous class definition is an expression, it must be part of a statement.
 - Always ends with semicolon



ANONYMOUS CLASS – EXAMPLE BY EXTENDING A 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();  
    }  
}
```

Output: nice fruits



INTERNAL WORKING OF GIVEN CODE

- A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
- An object of Anonymous class is created that is referred by p reference variable of Person type.
- Internal class generated by the compiler

```
static class TestAnonymousInner$1 extends Person
{
    TestAnonymousInner$1(){}
    void eat()
    {
        System.out.println("nice fruits");
    }
}
```



ANONYMOUS CLASS – EXAMPLE BY IMPLEMENTING AN INTERFACE

```
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();  
    }  
}
```

Output: nice fruits



INTERNAL CLASS GENERATED BY THE COMPILER

```
static class TestAnonymousInner1$1 implements Eatable
{
    TestAnonymousInner1$1(){}
    void eat(){
        System.out.println("nice fruits");
    }
}
```



WHAT YOU CAN & CAN'T DO

- Like local classes, anonymous classes can capture variables; they have the same access to local variables of the enclosing scope:
 - An anonymous class has access to the members of its enclosing class.
 - An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final.
 - Like a nested class, a declaration of a type (such as a variable) in an anonymous class shadows any other declarations in the enclosing scope that have the same name. See Shadowing for more information.
- Anonymous classes also have the same restrictions as local classes with respect to their members:
 - You cannot declare static initializers or member interfaces in an anonymous class.
 - An anonymous class can have static members provided that they are constant variables.



WHAT YOU CAN & CAN'T DO

- Note that you can declare the following in anonymous classes:
 - Fields
 - Extra methods (even if they do not implement any methods of the supertype)
 - You can access the method only inside the class.
 - but not outside e.g. via object.
 - Because the reference is of Parent type and parent can't access child's method.
 - Instance initializers
 - Local classes
 - However, you cannot declare constructors in an anonymous class.



EXAMPLE

```
public class Person {  
    String name;  
    public Person(String a) {  
        name = a;  
    }  
    public void display() {  
        System.out.println("Hello from Person " + name);  
    }  
  
    public void display(String msg) {  
        System.out.println("Hello from Person "+name+" :"+ msg);  
    }  
}
```



EXAMPLE

```
public class AnonymousWithClass {  
    public static void main(String[] args) {  
  
        Person p = new Person("Tanjina") {  
            @Override  
            public void display() {  
                display(2); // this is fine  
                System.out.println("Hello from Anonymous");  
            }  
            // Extra method  
            public void display(int a) {  
                System.out.println("Number: " + a);  
            }  
        };  
  
        p.display();  
        p.display("3"); // Ok  
        p.display(3); // error: The method display(String) in the type Person  
                       // is not applicable for the arguments (int)  
    }  
}
```



LOCAL INNER CLASS



LOCAL INNER CLASS

- A class i.e. created **inside a method** is called local inner class in java.
- If you want to invoke the methods of local inner class, you must instantiate this class inside the method.
- **Local inner class cannot be invoked from outside the method.**
- **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 local inner class.**



LOCAL INNER CLASS - EXAMPLE

```
class localInner2{  
    private int data=30;//instance variable  
    void display(){  
        int value=50;//local variable must be final till jdk 1.7 only  
        class Local{  
            void msg(){System.out.println(value);}  
        }  
        Local l=new Local();  
        l.msg();  
    }  
    public static void main(String args[]){  
        localInner2 obj=new localInner2();  
        obj.display();  
    }  
}
```



STATIC NESTED CLASS



STATIC NESTED CLASS

- A static class i.e. created inside a class is called static nested class in java.
 - It cannot access non-static (instance) data members and methods. It can be accessed by outer class name.
 - It can access static data members of outer class including private.
- In order to access the instance method of Inner class
 - Need to create the instance of static nested class because it has instance method msg().
 - But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.
- If you have the static member inside static nested class,
 - you don't need to create instance of static nested class.



STATIC NESTED CLASS - EXAMPLE

```
class TestOuter1{  
    static int data=30;  
  
    static class Inner{  
        void msg(){  
            System.out.println("data is "+data);  
        }  
        static void msg(String msg){  
            System.out.println(msg);  
        }  
    }  
    public static void main(String args[]){  
        TestOuter1.Inner obj=new TestOuter1.Inner();  
        obj.msg();  
        TestOuter1.Inner.msg("Hello");//no need to create the instance of static nes  
        ted class  
    }  
}
```



STATIC NESTED CLASS



STATIC NESTED CLASS

- A static class i.e. created inside a class is called static nested class in java.
 - It cannot access non-static (instance) data members and methods. It can be accessed by outer class name.
 - It can access static data members of outer class including private.
- In order to access the instance method of Inner class
 - Need to create the instance of static nested class because it has instance method msg().
 - But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.
- If you have the static member inside static nested class,
 - you don't need to create instance of static nested class.

