Nested Classes in Java

A **nested class** is a class defined within another class. It is used to logically group classes that are only used in a specific context, improving code organization and encapsulation.

Types of Nested Classes

Java has two main types of nested classes:

1. **Static Nested Classes**
2. **Non-static Nested Classes (Inner Classes)**
    - Member Inner Class
    - Local Inner Class
    - Anonymous Inner Class

1. Static Nested Class

A **static nested class** is a static class inside another class. Since it is static, it does not have access to the outer class's instance variables or methods.

Example: Static Nested Class

```
class Outer {
    static class StaticNested {
        void display() {
            System.out.println("Inside Static Nested Class");
        }
    }
}


public class Test {
    public static void main(String[] args) {
        Outer.StaticNested obj = new Outer.StaticNested(); // No need to create an instance of Outer
        obj.display();
    }
}
```

Output:

Inside Static Nested Class

2. Non-static Nested Classes (Inner Classes)

Inner classes are associated with an instance of the outer class and have access to its members.

2.1 Member Inner Class

A **member inner class** is a non-static class inside another class. It can access all members of the outer class, including private members.

Example: Member Inner Class

```
class Outer {
    private String message = "Hello from Outer Class";

    class Inner {
        void display() {
            System.out.println(message); // Accessing outer class private variable
        }
    }
}

public class Test {
    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner(); // Creating instance of Inner class
        inner.display();
    }
}
```

Output:

Hello from Outer Class

2.2 Local Inner Class

A **local inner class** is defined inside a method or a block. It can only be used within that method.

Example: Local Inner Class

```
class Outer {
    void outerMethod() {
        class LocalInner {
            void display() {
                System.out.println("Inside Local Inner Class");
            }
        }
        LocalInner local = new LocalInner(); // Creating an instance inside the method
        local.display();
    }
}

public class Test {
    public static void main(String[] args) {
        Outer obj = new Outer();
        obj.outerMethod();
    }
}
```

Output:

```
Inside Local Inner Class
```

2.3 Anonymous Inner Class

An **anonymous inner class** is a class without a name, created for immediate use, usually for implementing an interface or extending a class.

Example: Anonymous Inner Class

```
abstract class Anonymous {
```

```
    abstract void show();
}

public class Test {
    public static void main(String[] args) {
        Anonymous obj = new Anonymous() { // Anonymous inner class
            void show() {
                System.out.println("Inside Anonymous Inner Class");
            }
        };
        obj.show();
    }
}
```

Output:

Inside Anonymous Inner Class

Comparison: Static Nested Class vs. Inner Class

Feature	Static Nested Class	Inner Class
Instance Dependency	Does NOT require an instance of the outer class.	Requires an instance of the outer class.
Access to Outer Class Members	Can access only static members.	Can access all members (static and non-static).
Object Creation	OuterClass.StaticNested obj = new OuterClass.StaticNested();	OuterClass.InnerClass obj = outerInstance.new InnerClass();
Usage	Used for helper classes that don't need access to outer instance members.	Used when an inner class needs to interact with the outer class.

Case Study: Banking System Using Member Inner Class

A BankAccount class has an inner class Transaction that handles deposits and withdrawals.

```
class BankAccount {
    private double balance = 1000;

    class Transaction {
        void withdraw(double amount) {
            if (amount <= balance) {
                balance -= amount;
                System.out.println("Withdrawal successful! New Balance: " + balance);
            } else {
                System.out.println("Insufficient funds!");
            }
        }
    }
}

public class Test {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        BankAccount.Transaction transaction = account.new Transaction();
        transaction.withdraw(500);
    }
}
```

Output:

Withdrawal successful! New Balance: 500.0