Nested class

An inner class in Java is a class that is defined inside another class (known as the outer class).

Types of Inner Classes

* Non-static (Nested) Inner Class:

Defined within the scope of an instance of the outer class.

Can access instance variables and methods of the outer class directly.

Often used for grouping classes logically.

* Static Nested Class:

Defined as a static member of the outer class.

Can access static members of the outer class directly.

Doesn't have access to instance variables/methods of the outer class without an instance.

* Local Inner Class:

Defined within a method or scope block (like a loop or a conditional statement).

Cannot have access modifiers (public, private, protected) and cannot be declared static or interface.

Useful when a class is only needed within a specific method.

Reasons for Using Inner Classes

**Encapsulation**: Grouping related classes together can improve code organization and readability.

**Access to Outer Class Members:** Inner classes can directly access private members of the outer class, facilitating encapsulation.

Event Handling: Anonymous inner classes are commonly used for event handling in graphical user interfaces (GUIs).

// Outer class

public class Outer {

private int outerField = 10;

private static int outerStaticField = 20;

// Non-static inner class

class Inner {

void display() {

System.out.println("Inner class method");

System.out.println("Accessing outerField: " + outerField);

}

}

// Static nested class

static class StaticNested {

void display() {

System.out.println("Static nested class method");

System.out.println("Accessing outerStaticField: " + outerStaticField);

}

}

// Method with local inner class

void methodWithLocalClass() {

int localVar = 30;

class LocalInner {

void display() {

System.out.println("Local inner class method");

System.out.println("Accessing localVar: " + localVar);

}

}

LocalInner inner = new LocalInner();

inner.display();

}

// Method with anonymous inner class

void methodWithAnonymousClass() {

Runnable r = new Runnable() {

@Override

public void run() {

System.out.println("Anonymous inner class method");

}

};

r.run();

}

}

// Main class to demonstrate inner classes

class Main {

public static void main(String[] args) {

// Creating objects of inner classes

// Non-static inner class

Outer outer = new Outer();

Outer.Inner inner = outer.new Inner(); // Need to use 'outer.new' to instantiate non-static inner class

inner.display();

// Static nested class

Outer.StaticNested staticNested = new Outer.StaticNested(); // Directly instantiate static nested class

staticNested.display();

// Method with local inner class

outer.methodWithLocalClass();

// Method with anonymous inner class

outer.methodWithAnonymousClass();

}

}

**Accessing Members of Outer Class**

* Non-static Inner Class: Can access both static and instance members of the outer class directly.
* Static Nested Class: Can access only static members of the outer class directly. To access instance members, it needs an instance of the outer class.
* Local Inner Class and Anonymous Inner Class: Can access final or effectively final local variables of the enclosing scope (including parameters and local variables declared inside the method).

1. Non-static Inner Class

// Outer class with non-static inner class

public class Outer {

private int outerField = 10;

// Non-static inner class

class Inner {

void display() {

System.out.println("Inner class method");

System.out.println("Accessing outerField: " + outerField);

}

}

public static void main(String[] args) {

Outer outer = new Outer();

// Creating object of non-static inner class

Outer.Inner inner = outer.new Inner();

inner.display();

}

}

2. Static Nested Class

// Outer class with static nested class

public class Outer {

private static int outerStaticField = 20;

// Static nested class

static class StaticNested {

void display() {

System.out.println("Static nested class method");

System.out.println("Accessing outerStaticField: " + outerStaticField);

}

}

public static void main(String[] args) {

// Creating object of static nested class

Outer.StaticNested staticNested = new Outer.StaticNested();

staticNested.display();

}

}

3. Local Inner Class

// Outer class with method containing local inner class

public class Outer {

void methodWithLocalClass() {

final int localVar = 30; // Local variable must be final or effectively final

// Local inner class

class LocalInner {

void display() {

System.out.println("Local inner class method");

System.out.println("Accessing localVar: " + localVar);

}

}

// Creating object of local inner class

LocalInner inner = new LocalInner();

inner.display();

}

public static void main(String[] args) {

Outer outer = new Outer();

outer.methodWithLocalClass();

}

}

Static Nested Class: StaticNested class is defined inside Outer class with the static keyword. It's accessed directly using Outer.StaticNested.

Local Inner Class: LocalInner class is defined inside methodWithLocalClass() without static keyword, and it can access final or effectively final local variables of the method.

# Inner class

* **Scope Limited to Method or Block:**

Local inner classes are only visible and accessible within the method or block where they are defined. They are not accessible outside that scope.

* **Encapsulation and Scope Rules:**

Java enforces encapsulation rules to maintain the integrity and clarity of code. Local inner classes are designed to be used specifically within a limited scope, enhancing code readability and minimizing unintended access.

* **Instantiation Limitation:**

Since local inner classes are local to a method or block, you cannot create instances of these classes outside their defining scope. Attempts to instantiate them outside their scope will result in compilation errors.