✅ Nested Classes in Java (In-Depth Guide)

In Java, a **nested class** is a class declared **within another class or interface**. It helps in logically grouping classes that are only used in one place, and it increases encapsulation and readability.

🔷 TYPES of Nested Classes

Java categorizes nested classes into **2 main types**:

➤ 1. Static Nested Class

- Defined using the static keyword.
- Cannot access **non-static** members of the outer class directly.

➤ 2. Non-static Nested Class (aka Inner Class)

This category includes 3 types:

Type	Defined Where?
Member Inner Class	Inside a class, outside methods
Local Inner Class	Inside a method/block/constructor
Anonymous Inner Class	Declared and instantiated together

🔶 1. Static Nested Class (Class-level only)

✅ Characteristics:

- Declared with static keyword.
- Can **access only static members** of the outer class.
- Does **not require** an instance of the outer class.

🔸 Example:

```
class Outer {
  static int data = 50;

  static class StaticNested {
    void show() {
      System.out.println("Data: " + data);
    }
  }
}

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

🔶 2. Member Inner Class (Non-static Nested Class)

✅ **Characteristics:**

- Non-static.
- Can **access all members** (including private) of the outer class.
- Requires an **instance** of the outer class.

🔸 **Example:**

```
class Outer {
  private int value = 10;

  class Inner {
    void show() {
      System.out.println("Value: " + value);
    }
  }
}

public class Test {
  public static void main(String[] args) {
    Outer outer = new Outer();
    Outer.Inner inner = outer.new Inner();
    inner.show();
  }
}
```

🔶 3. Local Inner Class (Inside method/block)

✅ **Characteristics:**

- Defined inside a **method, block, or constructor**.
- Can access **final or effectively final** variables of the method.
- Scope is **local** to the block.

🔸 **Example:**

```
class Outer {
    void display() {
        int localVar = 100; // must be effectively final

        class LocalInner {
            void show() {
                System.out.println("Local var: " + localVar);
            }
        }

        LocalInner inner = new LocalInner();
        inner.show();
    }
}
```

🔶 4. Anonymous Inner Class (No class name)

✅ **Characteristics:**

- Defined and instantiated **in one statement**.
- Used for:
    - Implementing an interface
    - Extending a class
- Often used in **event handling, threads, GUIs**

🔸 **Example:**

```
abstract class Greet {
    abstract void sayHello();
}

public class Test {
    public static void main(String[] args) {
        Greet greet = new Greet() {
            void sayHello() {
                System.out.println("Hello from Anonymous Class!");
            }
        };
        greet.sayHello();
    }
}
```

✅ 🔒 Rules & Restrictions

Feature	Static Nested	Member Inner	Local Inner	Anonymous
Needs Outer Instance	❌ No	✅ Yes	✅ Yes	✅ Yes
Can access private members	✅ Yes	✅ Yes	✅ Yes	✅ Yes
Can use static members	✅ Yes	❌ No	❌ No	❌ No
Can have static methods	✅ Yes	❌ No	❌ No	❌ No
Name required	✅ Yes	✅ Yes	✅ Yes	❌ No
Use Case	Utility/helper	Logic grouping	Scoped task	Single use

---

🔁 Accessing Nested Classes

```
Outer.StaticNested sn = new Outer.StaticNested(); // static
Outer.Inner inner = new Outer().new Inner(); // non-static
```

✅ 💡 When to Use Which?

Situation	Use This
Helper/utility class	Static Nested Class
Tightly coupled logic to outer class	Member Inner Class
Temporary use in method/block	Local Inner Class
One-time behavior / event	Anonymous Inner Class

✅ Real-life Example (GUI Event)

```
Button b = new Button("Click Me");
b.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      System.out.println("Button clicked!");
   }
});
```

✅ Summary Chart

Nested Class Type	static?	Requires Outer Object?	Scope	Use Case
Static Nested	✅ Yes	❌ No	Class-level	Utility class
Member Inner	❌ No	✅ Yes	Class-level	Encapsulate logic
Local Inner	❌ No	✅ Yes	Method/block	Scoped temporary class
Anonymous Inner	❌ No	✅ Yes	Inline expression	One-time use (event, thread)

Great! Let's now focus specifically on **where you can write nested classes** in Java — that is, where it is **legal and valid** to define a **nested class** (including static and non-static inner classes). This explanation is **fully based on the official Oracle Java documentation**.

✅ Where Can We Write Nested Classes in Java?

A **nested class** (including both static and non-static types) can be declared **inside**:

Enclosing Type	Static Nested Class Allowed	Non-static (Inner) Class Allowed	Example
Class	✅ Yes	✅ Yes	✅

Enclosing Type	Static Nested Class Allowed	Non-static (Inner) Class Allowed	Example
Interface	✅ Yes	❌ No	✅
Enum	✅ Yes	✅ Yes	✅
Record (Java 16+)	✅ Yes	✅ Yes	✅
Method / Block	❌ No (only local/anonymous)	✅ Yes (local, anonymous)	✅

🔷 1. Inside a Class

You can declare both **static nested classes** and **non-static inner classes**.

```
class Outer {
    static class StaticNested {}     // ✅ Allowed
    class Inner {}                   // ✅ Allowed
}
```

🔷 2. Inside an Interface

You can declare **static nested classes only**.

📌 Non-static inner classes are **NOT allowed** inside interfaces.

```
interface MyInterface {
    static class Helper {}       // ✅ Allowed
    // class InnerClass {}       ❌ Compilation Error (non-static not allowed)
}
```

🔸 All members of an interface are implicitly public static, so only static nested classes make sense here.

🔷 3. Inside an Enum

Both **static** and **non-static** nested classes are allowed.

```
enum Status {
    SUCCESS, FAILURE;

    static class StaticHelper {}   // ✅ Allowed
    class InnerHelper {}           // ✅ Allowed
}
```

🔷 4. Inside a Record (since Java 16)

Records can contain **both static and non-static nested classes**.

```
record Person(String name, int age) {
    static class Validator {}      // ✅ Allowed
    class Helper {}                // ✅ Allowed
}
```

💡 Note: The non-static inner class inside a record can access record fields.

🔷 5. Inside a Method / Constructor / Block

You cannot write **static nested classes** inside a method — but you **can write**:

- Local inner classes
- Anonymous inner classes

```
class Outer {
    void someMethod() {
        class LocalInner {
            void show() {
                System.out.println("Local Inner Class");
            }
        }

        Runnable r = new Runnable() {
            public void run() {
                System.out.println("Anonymous Inner Class");
            }
        };
    }
}
```

❌ Where You Cannot Write Nested Classes

Location	Valid?	Reason
Inside a method → static class	❌	Methods can't contain static types
Outside all classes/interfaces	❌	Top-level class only allowed here
In parameter lists	❌	Classes can't be defined in params

✅ Summary Table: Where You Can Declare Nested Classes

Enclosing Type	static Class	non-static Class	Notes
Class	✅ Allowed	✅ Allowed	Common use
Interface	✅ Allowed	❌ Not allowed	Static only
Enum	✅ Allowed	✅ Allowed	Less common
Record	✅ Allowed	✅ Allowed	Java 16+
Method	❌ Not allowed	✅ Local/Anonymous	Scoped to method
Constructor	❌ Not allowed	✅ Local/Anonymous	Temporary
Static Block	❌ Not allowed	✅ Local/Anonymous	Limited use

Awesome! Here's a **visual-style summary + real-world code examples** for each place where you can write **nested classes** in Java — covering both static and non-static types.

✅ Visual Summary – Where You Can Write Nested Classes in Java

```
+------------------+----------------------+-------------------------+
| Enclosing Type   | Static Nested Class  | Non-static Inner Class  |
+------------------+----------------------+-------------------------+
| Class            | ✅ Allowed           | ✅ Allowed              |
| Interface        | ✅ Allowed           | ❌ Not Allowed          |
| Enum             | ✅ Allowed           | ✅ Allowed              |
| Record (Java 16+)| ✅ Allowed           | ✅ Allowed              |
| Method/Block     | ❌ Not Allowed       | ✅ Local/Anonymous Only |
+------------------+----------------------+-------------------------+
```

🎯 Code Examples for Each Case

✅ 1. Inside a Class

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

    class Inner {
        void print() {
            System.out.println("Inside Inner Class");
        }
    }
}

public class Test {
    public static void main(String[] args) {
        Outer.StaticNested sn = new Outer.StaticNested();
        sn.print();

        Outer.Inner in = new Outer().new Inner();
        in.print();
    }
}
```

✅ 2. Inside an Interface

```
interface MyInterface {
    static class StaticNested {
        void print() {
            System.out.println("Static Nested Class inside Interface");
        }
    }

    // ❌ Non-static inner class NOT allowed inside interfaces
}

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

✅ 3. Inside an Enum

```
enum Status {
    SUCCESS, FAILURE;

    static class StaticHelper {
        void print() {
            System.out.println("Static class inside Enum");
        }
    }

    class InnerHelper {
        void print() {
            System.out.println("Inner class inside Enum");
        }
    }
}
```

```
public class TestEnum {
    public static void main(String[] args) {
        Status.StaticHelper sh = new Status.StaticHelper();
        sh.print();

        Status.InnerHelper ih = new Status().new InnerHelper(); // Needs enum instance
        ih.print();
    }
}
```

✅ **4. Inside a Record (Java 16+)**

```
record Person(String name, int age) {
    static class Validator {
        static boolean isValidAge(int age) {
            return age > 0;
        }
    }

    class Printer {
        void print() {
            System.out.println("Name: " + name + ", Age: " + age);
        }
    }
}

public class TestRecord {
    public static void main(String[] args) {
        Person p = new Person("John", 25);
        p.new Printer().print();
        boolean valid = Person.Validator.isValidAge(25);
        System.out.println("Is age valid? " + valid);
    }
}
```

✅ 5. Inside a Method (Local/Anonymous Inner Class)

```
class Outer {
    void show() {
        int x = 10;

        // Local inner class
        class LocalInner {
            void print() {
                System.out.println("Local inner class, x = " + x);
            }
        }

        LocalInner local = new LocalInner();
        local.print();

        // Anonymous inner class
        Runnable r = new Runnable() {
            public void run() {
                System.out.println("Anonymous inner class inside method");
            }
        };
        r.run();
    }
}

public class TestMethodInner {
    public static void main(String[] args) {
        new Outer().show();
    }
}
```

🔍 Real-World Use Cases

Use Case	**Class Type**
Utility/helper class	Static nested class
GUI/event handling (e.g., Swing)	Anonymous inner class
Encapsulating logic	Member inner class
Short-lived use inside method	Local inner class
Scoped validation/helper	Inner class inside Record