**Sealed Classes in Java (Introduced in Java 17)**

Sealed classes and interfaces allow you to **restrict which other classes or interfaces may extend or implement them**. This gives you **explicit control over the inheritance hierarchy**, improving **security**, **readability**, and **maintainability**.

**Why Sealed Classes?**

* Prevents unintended extension.
* Makes domain models and hierarchies more predictable.
* Complements final and non-sealed modifiers.

**Syntax**

public sealed class Shape permits Circle, Square {

// common members

}

Here:

* Shape is a **sealed class**.
* Only Circle and Square are allowed to extend Shape.

**🧱 Modifiers Used**

| **Modifier** | **Description** |
| --- | --- |
| Sealed | Declares the base class/interface with restricted subtypes. |
| Permits | Lists all permitted subclasses. |
| Final | Prevents further subclassing. |
| non-sealed | Allows a permitted subclass to be extended further. |

**Full Example**

// Base sealed class

public sealed class Shape permits Circle, Square, Triangle {}

// Permitted subclasses

public final class Circle extends Shape {}

public final class Square extends Shape {}

public non-sealed class Triangle extends Shape {} // Can be extended by other classes

**❌ Rules**

1. All permitted subclasses **must be in the same module or package**.
2. All permitted subclasses must explicitly declare themselves as final, sealed, or non-sealed.

**🧠 Use Case Example**

Suppose you're building a **drawing application**, and you want to restrict the shape hierarchy:

sealed interface Shape permits Circle, Rectangle {}

final class Circle implements Shape {}

non-sealed class Rectangle implements Shape {} // Allows further extension