\*\*\*sealed class A permits B,C {}

Now, class B,C can be 🡪 sealed/ non-sealed / final

\*\*non-sealed means any class can extends it.

A screen shot of a computer program

AI-generated content may be incorrect.

Here are the **key things to remember** about **sealed classes** in Java:

**1. Sealed Classes Restrict Inheritance**

* A **sealed class** allows you to define a class with a restricted set of subclasses. You specify which classes are permitted to extend or implement the sealed class using the permits keyword.
* **Inheritance is restricted**, meaning you can control which classes or interfaces can extend or implement the sealed class.

public sealed class Animal permits Dog, Cat { }

**2. permits Keyword**

* The permits keyword is used to explicitly list the subclasses that are allowed to inherit from a sealed class. The subclasses must be either **final**, **sealed**, or **non-sealed** classes themselves.
* The subclasses listed in the permits clause are the **only valid subclasses** of the

public sealed class Animal permits Dog, Cat { }

**3. Subclasses Can Be final, sealed, or non-sealed**

* Subclasses of a sealed class can be:
  + **final**: The subclass cannot be further extended.
  + **sealed**: The subclass can also restrict its own subclasses.
  + **non-sealed**: The subclass can be extended freely without restrictions.

public final class Dog extends Animal { }

public sealed class Cat extends Animal permits Siamese, Persian { }

public non-sealed class Bird extends Animal { }

**4. Inheritance Hierarchy Control**

* Sealed classes provide better control over inheritance hierarchies. You can define a clear and limited set of allowed subclasses, which can prevent unintended subclassing and improve maintainability.

**5. Cannot Be Inherited by Any Other Class**

* Unlike regular classes, **sealed classes cannot be inherited by any class not explicitly listed** in the permits clause. This makes them ideal for representing closed hierarchies where you want to limit the possible subclasses.

**6. Cannot Be Abstract**

* Sealed classes themselves can be abstract, but **they are not implicitly abstract**. This means you can still create concrete (non-abstract) classes that can be instantiated, but if the class is abstract, it won’t be directly instantiated.

**7. sealed Modifier on Interfaces**

* The sealed modifier can also be used on **interfaces** in addition to classes. This allows you to restrict which classes can implement a particular interface.

public sealed interface Shape permits Circle, Square { }

**8. Use Case: Limited Inheritance**

* Sealed classes are useful when you have a known set of subclasses and want to **restrict** the ability to add subclasses in the future, enforcing more predictable behaviour.

**9. Sealed Classes Can Be Used with Pattern Matching**

* From Java 16, sealed classes can be used in conjunction with **pattern matching** in switch statements. This allows you to handle the subclasses of a sealed class in a more expressive and readable way.

public static void handleAnimal(Animal animal) {

switch (animal) {

case Dog dog -> System.out.println("It's a dog!");

case Cat cat -> System.out.println("It's a cat!");

default -> System.out.println("Unknown animal!");

}

}

**10. Final, Sealed, or Non-Sealed Subclasses**

* The subclasses of a sealed class can themselves be marked as:
  + **final**: No further subclassing allowed.
  + **sealed**: Subclassing can be further restricted.
  + **non-sealed**: No subclassing restrictions.

// Sealed class with allowed subclasses

public sealed class Vehicle permits Car, Bike {}

// Final subclass of Vehicle

public final class Car extends Vehicle {}

// Sealed subclass of Vehicle

public sealed class Bike extends Vehicle permits MountainBike, RoadBike {}

// Non-sealed subclass of Bike

public non-sealed class MountainBike extends Bike {}

// Final subclass of Bike

public final class RoadBike extends Bike {}