Java 15, 17+

Sealed Classes & Pattern Matching (Java)

PART A: Sealed Classes
Definition:
Sealed Classes (introduced in **Java 17**) restrict **which classes can extend or implement** a class or interface.

- You control inheritance explicitly.

Why Sealed Classes are Needed / Purpose
Problems before sealed classes:

- Anyone could extend your class
- Hard to control class hierarchy
- Risk of misuse

Sealed classes help to:

- Improve **security**
- Maintain **controlled inheritance**
- Work better with **pattern matching**

How Sealed Classes Work (Rules)

1. Use sealed keyword
2. Use permits to specify allowed subclasses
3. Subclasses must be:
    - final or
    - sealed or
    - non-sealed

Syntax Example

```
sealed class Shape
    permits Circle, Rectangle, Square {
}
final class Circle extends Shape {}
non-sealed class Rectangle extends Shape {}
sealed class Square extends Shape permits SmallSquare {}
```

Real-Life Example

- **Government ID system** → Only specific document types allowed
- **Payment methods** → Only Card, UPI, Cash

Advantages

- Controlled inheritance
- Better design clarity
- Safer APIs
- Compiler checks

Interview Questions (Sealed Classes)
Q1. Which Java version introduced sealed classes?
A. Java 17.
Q2. Can sealed classes have constructors?
A. Yes.
Q3. Can interfaces be sealed?
A. Yes.

PART B: Pattern Matching

Definition

Pattern Matching simplifies **type checking and casting,** making code more readable and safer.

- Reduces boilerplate code.

Why Pattern Matching is Needed

Before:

```
if (obj instanceof String) {
    String s = (String) obj;
}
```

With pattern matching:

```
if (obj instanceof String s) {
}
```

Pattern Matching with instanceof (Java 16+)

```
if (obj instanceof Integer i) {
    System.out.println(i + 10);
}
```

Pattern Matching with switch (Java 17+)

```
static String getShape(Shape s) {
    return switch (s) {
        case Circle c -> "Circle";
        case Rectangle r -> "Rectangle";
        default -> "Unknown";
    };
}
```

Sealed Classes + Pattern Matching (Very Important)

When used together:

- Compiler ensures **all cases are handled**
- No need for default

```
return switch (shape) {
    case Circle c -> c.area();
    case Rectangle r -> r.area();
};
```

Real-Time Usage

- Domain modeling
- Payment systems
- State machines
- API design

Advantages of Pattern Matching

- Cleaner code
- Fewer errors
- Better readability
- Compiler safety

Limitations

- Requires newer Java versions
- Legacy systems may not support

Common Interview Questions (Cognizant Level)

Q1. What problem do sealed classes solve?
A. Uncontrolled inheritance.

Q2. Difference between sealed and final?
A. final → no inheritance
sealed → limited inheritance

Q3. Which feature works best with sealed classes?
A. Pattern Matching.

Q4. What does non-sealed mean?
A. Allows further unrestricted inheritance.

Q5. Is pattern matching runtime or compile-time safe?
A. Compile-time safe.

One-Line Summary (Quick Revision)
Sealed classes control inheritance, and pattern matching simplifies type checking with compiler safety.

Text Blocks, Records & Virtual Threads (Modern Java Features)

PART A: Text Blocks
Definition:
Text Blocks (introduced in **Java 15**) allow writing **multi-line strings** easily using """.

- Makes large text readable and clean.

Why Text Blocks are Needed
Before:

- Too many \n
- Hard to read strings

Used for:

- SQL queries
- JSON / XML
- HTML templates

Syntax Example

```
String query = """
    SELECT * FROM users
    WHERE active = true
    ORDER BY name
    """;
```

Real-Life Example
Writing a paragraph in a notebook instead of one long sentence.

Advantages

- Readable
- Less escaping
- Cleaner code

Interview Questions (Text Blocks)

Q1. Which Java version introduced Text Blocks?
A. Java 15.

Q2. Are text blocks immutable?
A. Yes (String).

PART B: Records

Definition:
Records (introduced in **Java 16**) are **special classes** used to store **immutable data.**

- Mainly used for **data carriers (DTOs).**

Why Records are Needed
Before:

- Too much boilerplate (getters, constructor, equals)

Records automatically provide:

- Constructor
- Getters
- equals(), hashCode(), toString()

Syntax Example

```
record User(String name, int age) {}
```

Usage:

```
User u = new User("Ali", 22);
System.out.println(u.name());
```

Key Rules of Records

- Fields are final
- Cannot extend other classes
- Can implement interfaces

Real-Life Example
Aadhar card data → fixed and unchangeable.

Advantages

- Less code
- Immutable
- Perfect for APIs & microservices

Interview Questions (Records)
Q1. Are records mutable?
A. No.
Q2. Can records have methods?
A. Yes.

PART C: Virtual Threads

Definition
Virtual Threads (introduced in **Java 21**) are **lightweight threads** managed by JVM instead of OS.
A. Used for **high concurrency.**

Why Virtual Threads are Needed

Problems with traditional threads:

- Heavy
- Limited by OS
- Memory expensive

Virtual threads:

- Cheap
- Millions can be created
- Better scalability

Syntax Example

```
Thread.startVirtualThread(() -> {
    System.out.println("Running in virtual thread");
});
```

Executor:

```
try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
    executor.submit(() -> task());
}
```

Real-Life Example

- OS threads → Buses
- Virtual threads → Bikes (light & fast)

Advantages

- Massive concurrency
- Simple coding model
- Ideal for I/O tasks

Limitations

- Not ideal for CPU-heavy tasks
- Needs Java 21+

Common Interview Questions (Cognizant Level)

Q1. Which Java version introduced Virtual Threads?

A. Java 21.

Q2. Are virtual threads faster than normal threads?

A. Better scalability, not raw speed.

Q3. Are virtual threads OS threads?

A. No, JVM-managed.

Quick Comparison Table

Feature	Java Version	Purpose
Text Blocks	Java 15	Multi-line strings
Records	Java 16	Immutable data
Virtual Threads	Java 21	High concurrency

One-Line Summary (Quick Revision)

Text Blocks simplify multi-line strings, Records reduce boilerplate for immutable data, and Virtual Threads enable massive concurrency.
