Java Evolution

Key Versions: 8, 11, 17, 19 & 21

Must-Know Features for Backend Developers & Interview Prep

Java 8 (2014)

Lambdas & Functional Interfaces

What: Introduced functional programming in Java by allowing functions as arguments.

Why: Reduces boilerplate code and makes APIs like Streams more powerful.

```
Runnable r = () -> System.out.println("Hello Java 8");
```

Note: Behind the scenes, lambdas map to functional interfaces like Runnable, Callable, Supplier, etc.

Java 8 (2014) – Continued

Streams API

What: A new abstraction to process collections in a declarative way.

Why: Eliminates verbose loops, supports parallelism, and improves readability.

```
List.of(1,2,3,4)
    .stream()
    .filter(n -> n%2==0)
    .map(n -> n*n)
    .forEach(System.out::println);
```

Note: Streams are lazy; computations happen only when a terminal operation is invoked.

Java 8 (2014) – Continued

Optional

What: A container object to represent the presence or absence of a value.

Why: Helps prevent NullPointerExceptions by encouraging explicit handling of missing values.

```
Optional<String> name = Optional.of("Java");
name.ifPresent(System.out::println);
```

Note: Avoid overusing Optional in fields; it’s mainly for return values.

Java 8 (2014) – Continued

Date/Time API & Default Methods

What: New `java.time` API replaced old Date/Calendar. Interfaces gained default/static methods.

Why: Date/Calendar were error-prone; default methods allow evolving APIs without breaking old code.

```
LocalDate today = LocalDate.now();
interface MyInterface { default void log() { System.out.println("log"); } }
```

Java 11 (2018, LTS)

HttpClient API

What: Modern replacement for HttpURLConnection.

Why: Provides synchronous & asynchronous requests, supporting HTTP/2.

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest req = HttpRequest.newBuilder(URI.create("https://example.com")).build();
```

Java 11 (2018, LTS) – Continued

New String Methods

What: Added utility methods like isBlank(), lines(), repeat().

Why: Simplifies common string operations.

```
"hello".repeat(3); // hellohellohello
```

Single-file execution: Run a Java file directly without compiling separately.

```
java HelloWorld.java
```

Java 17 (2021, LTS)

Sealed Classes

What: Restricts inheritance so only specific classes can extend a type.

Why: Adds more control and safer APIs.

```
sealed interface Shape permits Circle, Square {}
```

Java 17 (2021, LTS) – Continued

Pattern Matching

What: Simplifies type checks and casts.

Why: Avoids verbose casting with instanceof.

```
if (obj instanceof String s) {
   System.out.println(s.length());
}
```

Also introduced preview: Pattern Matching for switch.

Java 19 (2022)

Virtual Threads (preview, Project Loom)

What: Lightweight threads that scale massively.

Why: Solves the issue of blocking I/O and improves concurrency.

```
Thread.startVirtualThread(() -> doTask());
```

Note: Each virtual thread is cheap compared to OS threads, enabling thousands of them.

Java 21 (2023, LTS)

Virtual Threads (final)

What: Virtual Threads graduated from preview to final feature.

Why: Game-changer for high-concurrency applications.

```
Thread.startVirtualThread(() -> System.out.println("Hello"));
```

Java 21 (2023, LTS) – Continued

String Templates & Record Patterns

What: Safer string interpolation and pattern matching for records.

Why: Cleaner syntax, easier destructuring of data.

```
String msg = STR."Hello {name}";
if (p instanceof Point(int x, int y)) { ... }
```

Java 21 (2023, LTS) – Continued

Sequenced Collections

What: New interfaces to handle ordered collections consistently.

Why: Gives predictable behavior across List, Set, Map.

```
SequencedSet<String> s = new LinkedHashSet<>();
```

Java has evolved massively since Java 8!

- For interviews, focus on Java 8, 11, 17, and 21. Java 19 is key for Virtual Threads.