AMAN SAHNI

Save for later

JAVA 25 LONG-TERM SUPPORT (LTS) RELEASE FEATURES

@codingWithAman

Swipe ⟶

AMAN SAHNI

Save for later

SIMPLIFIED HELLO WORLD (INSTANCE MAIN METHODS)

NO CLASSES, NO PUBLIC STATIC, NO STRING[] ARGS!

❌ BEFORE (Java 24)

```
public class HelloWorld {
  public static void main(
    String[] args) {
    System.out.println(
      "Hello, World!");
  }
}
```

✅ AFTER (Java 25)

```
void main() {
  println("Hello, World!");
}
```

@codingWithAman

Swipe ⟶

AMAN SAHNI

Save for later

MODULE IMPORT DECLARATIONS (SIMPLIFIED IMPORTS)

IMPORT ENTIRE MODULES WITH ONE STATEMENT!

❌ BEFORE (Java 24)

```java
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Stream;
import java.util.stream.Collectors;
import java.util.function.Function;
import java.nio.file.Path;
import java.nio.file.Files;

public class DataProcessor {
  // Your code here
}
```

✅ AFTER (Java 25)

```java
import module java.base;

public class DataProcessor {
  // All java.base classes
  // available automatically!
  // List, Map, Stream, Path...
}
```

@codingWithAman

Swipe ⟶

AMAN SAHNI

Save for later

PRIMITIVE PATTERN MATCHING (INSTANCEOF WITH PRIMITIVES)

PATTERN MATCHING NOW SUPPORTS PRIMITIVE TYPES DIRECTLY IN INSTANCEOF!

❌ BEFORE (Java 24)

```
void test(Object obj) {
  if (obj instanceof Integer) {
    int i = (Integer) obj;
    if (i >= 0 && i <= 127) {
      // Can safely use as byte
      byte b = (byte) i;
      System.out.println(b);
    }
  }
}
```

✅ AFTER (Java 25)

```
void test(Object obj) {
  if (obj instanceof byte b) {
    // Direct check & assignment!
    System.out.println(b);
  }
}
```

@codingWithAman

Swipe ⟶

PRIMITIVE PATTERN MATCHING (SWITCH WITH PRIMITIVES)

❌ BEFORE (Java 24)

```
String grade(Number n) {
  if (n instanceof Integer i) {
    if (i >= 90) return "A";
    if (i >= 75) return "B";
    if (i >= 60) return "C";
  } else if (n instanceof Double d) {
    if (d >= 59.5) return "C";
  }
  return "D/F";
}
```

✅ AFTER (Java 25)

```
String grade(Number n) {
  return switch (n) {
    case int i when i >= 90 -> "A";
    case int i when i >= 75 -> "B";
    case int i when i >= 60 -> "C";
    case double d when d >= 59.5
      -> "C (rounded)";
    default -> "D/F";
  };
}
```

AMAN SAHNI

Save for later

FLEXIBLE CONSTRUCTOR BODIES CODE BEFORE SUPER()

FAIL-FAST VALIDATION BEFORE SUPERCLASS INITIALIZATION!

❌ BEFORE (Java 24)

```
class Employee extends Person {
  Employee(String name, int age) {
    super(name, age);
    // Validation AFTER super
    if (age < 18 || age > 67) {
      throw new IllegalArgumentException(
        "Invalid age");
    }
  }
}
```

✅ AFTER (Java 25)

```
class Employee extends Person {
  Employee(String name, int age) {
    // Validate BEFORE super!
    if (age < 18 || age > 67) {
      throw new IllegalArgumentException(
        "Invalid age");
    }
    super(name, age);
  }
}
```

@codingWithAman

Swipe ⟶

STRUCTURED CONCURRENCY MANAGE PARALLEL TASKS (PREVIEW)

TREATS PARALLEL TASKS AS A UNIT – BETTER ERROR HANDLING & OBSERVABILITY

❌ BEFORE - ExecutorService

```
ExecutorService executor =
  Executors.newCachedThreadPool();

Future<String> user =
  executor.submit(() -> fetchUser());
Future<Order> order =
  executor.submit(() -> fetchOrder());

String u = user.get();
Order o = order.get();
executor.shutdown();
// Manual error handling needed!
```

✅ AFTER - StructuredTaskScope

```
try (var scope =
  StructuredTaskScope.open()) {

  var user = scope.fork(
    () -> fetchUser());
  var order = scope.fork(
    () -> fetchOrder());

  scope.join();
  return new Result(user.get(),
                    order.get());
} // Auto cleanup & cancellation!
```

COMPACT OBJECT HEADERS (REDUCED MEMORY FOOTPRINT)

❌ BEFORE (Java 24)

`Object Header Size:`

96-128 bits per object

- Mark word (64 bits)
- Class pointer (32/64 bits)
- Array length (32 bits if array)

✅ AFTER (Java 25)

`Object Header Size:`

64 bits per object

⚡ Smaller heap footprint
⚡ Better cache utilization
⚡ Improved performance

AMAN SAHNI

Save for later

SCOPED VALUES (FINAL) BETTER THAN THREADLOCAL

IMMUTABLE, SAFER, BETTER PERFORMANCE THAN THREADLOCAL!

❌ BEFORE - ThreadLocal

```
class UserContext {
  static final ThreadLocal<User> USER =
    new ThreadLocal<>();

  void processRequest(User user) {
    USER.set(user);
    try {
      doWork();
    } finally {
      USER.remove(); // Must cleanup!
    }
  }
}
```

✅ AFTER - Scoped Values

```
class UserContext {
  static final ScopedValue<User> USER =
    ScopedValue.newInstance();

  void processRequest(User user) {
    ScopedValue.where(USER, user)
               .run(() -> doWork());
    // Auto cleanup, immutable!
  }
}
```

AMAN SAHNI

Save for later

STABLE VALUES API DEFERRED IMMUTABILITY (PREVIEW)

❌ PROBLEM - final fields

```java
class Config {
  // Must initialize immediately!
  private final ExpensiveObject obj =
    new ExpensiveObject();

  // Slow startup even if
  // obj is never used!
}
```

⚠️ final = immediate initialization = slow startup

✅ SOLUTION - StableValue

```java
class Config {
  // Lazy init + immutable!
  private final StableValue<Expensive>
    obj = StableValue.of(() ->
      new ExpensiveObject());

  void use() {
    obj.get(); // Created on first use
  }
}
```

⚡ Lazy initialization + JVM optimizations!

@codingWithAman