Java after 11

Overview

- Text Blocks
- Better NullPointerExceptions
- Pattern matching for instanceof
- Switch Expressions
- Sealed Classes
- Records
- Garbage Collection Improvements
- And more!

Text Blocks

Multi-line string **literal** that generally doesn't need escape sequences

```
String grossJson = "{\n\"id\": 1,\n\"qty\": 5,\n\"price\": 100.00}";
String prettyJson = """
            {
                "id": 1,
                "qty": 5,
                "price": 100
            }
            """;
```

Text Block Rules

- Indentation determined by farthest left character & closing quotes
- Single line blocks are supported with `\`:

```
String text = """
              Lorem ipsum dolor sit amet, consectetur adipiscing \
              elit, sed do eiusmod tempor incididunt ut labore \
              et dolore magna aliqua.
              """;
```

Better NullPointerExceptions

- Consider a NullPointerException for this line `a.b.c.i = 99;`

```
Exception in thread "main" java.lang.NullPointerException at Prog.main(Prog.java:5)
```

- Which variable was null? a, b, c, or i? The message doesn't tell you
- NullPointerExceptions now:

```
Exception in thread "main" java.lang.NullPointerException: Cannot read field "c" because "a.b" is null at Prog.main(Prog.java:5)
```

Pattern Matching for instanceof

Before...

```
Object o = someRandomObject();
// Check type
if (o instanceof String) {
    // Cast to String
    String s = (String)o;
    // do something with String s
// Check type
} else if (o instanceof Number) {
    // Cast to Number
    Number n = (Number)o;
    // do something with Number n
}
```

After...

```
Object o = someRandomObject();
// Check type and cast to String
if (o instanceof String s) {
    // do something with String s
// Check type and cast to Number
} else if (o instanceof Number n) {
    // do something with Number n
}
```

Another instanceof example

Before...

```
public final boolean equals(Object o) {
    if (!(o instanceof Point)) return false;
    Point other = (Point) o;
    return x == other.x && y == other.y;
}
```

After...

```
public final boolean equals(Object o) {
    return (o instanceof Point other)
        && x == other.x && y == other.y;
}
```

Sealed Classes

```
class Shape { } // No limits to extension
```

```
final class Shape { } // Nothing can extend
```

- A sealed class can only be extended by classes **permitted** to do so

```
sealed class Shape {
    permits Circle, Rectangle, Triangle {
}
class Circle extends Shape {}
class Rectangle extends Shape {}
class Triangle extends Shape {}
```

Switch Expressions

Before...

```
int numLetters; // eww
switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numLetters = 6;
        break;
    case TUESDAY:
        numLetters = 7;
        break;
    // Thursday, Saturday, Wednesday...
}
```

After...

```
// Can actually returna a value now
int numLetters = switch (day) {
    // Arrows means no breaks needed, they don't "fall through"
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
}
```

- Switch expressions must be exhaustive, but don't require a 'default'

Records

Consider a simple data class:

```
final class Range {
    private final int start;
    private final int end;

    Range(int start, int end) {
        this.start = start;
        this.end = end;
    }

    public int start() { return start; }
    public int end() { return end; }
    public boolean equals(Object o) { /*...*/ }
    public int hashCode() { /*...*/ }
    public String toString() { /*...*/ }
```

Record Properties

- Immutable
- Transparent
- Can't extend any class (implicitly extends record)
- Can't be extended
- But can implement interfaces

Record Constructors

- Automatically given `canonical constructors`
- You can make your own, but **all constructors must ultimately call the canonical constructor**

```
record Range(int start, int end) {
    // Canonical constructor that uses the compact syntax
    Range {
        if (end < begin) { throw new IllegalArgumentException("Begin must be less than end"); }
    }

    // Has to use the canonical constructor
    Range(int end) { this(0, end); }
}
```

Garbage Collectors

https://blogs.oracle.com/javamagazine/post/java-garbage-collectors-evolution
https://www.optaplanner.org/blog/2021/09/15/HowMuchFasterIsJava17.html

G1 (Garbage First)

- replaces CMS (Concurrent Mark Sweep)

ZGC (Z Garbage Collector)

- Low latency

Stream::toList

Before...

```
var nums = IntStream.range(0, 10)
                    .boxed()
                    .collect(Collectors.toList());
```

After...

```
var nums = IntStream.range(0, 10)
                    .boxed()
                    .toList();
```

Stream::mapMulti

Which of the following compile?

```
int x = 1;

int class = 1;

int goto = 1;

int static = 1;

int var = 1;

int void = 1;

int const = 1;
```

Which of the following compile? (Solution)

```
int x = 1;           // Yes...

int class = 1;       // No, java keyword

int goto = 1;        // No, java keyword that is not actually used (reserved)

int static = 1;      // No, java keyword

int var = 1;         // Yes!

var var = "var";     // Yes!

int void = 1;        // No, java keyword

int const = 1;       // No, another reserved java keyword
```

Fun Stuff

- Sorting/compare error
- enums have limits to how many
- bytes are represented as ints
- Regular Expressions Error

Don't include...

- Generics were introduced in bytecode in 1.3