How exception handling has evolved in Java

Java 1.0 (1995):

```
try {
    // Code that may throw an exception
} catch (Exception e) {
    // Exception handling code
} finally {
    // Code to be executed regardless of whether an exception occurred or not
}
```

Java 5.0 (2004) - Introduction of Checked Exceptions:

```
try {
    // Code that may throw a checked exception
} catch (Exception e) {
    // Exception handling code
} finally {
    // Code to be executed regardless of whether an exception occurred or not
}
```

Java 7 (2011) - Multi-catch and Automatic Resource Management (ARM):

```
try {
    // Code that may throw an exception
} catch (IOException | SQLException e) {
    // Multi-catch: Handling multiple exceptions in the same catch block
} finally {
    // Code to be executed regardless of whether an exception occurred or not
}

// Automatic Resource Management (try-with-resources)
try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {
    // Code that uses the resource
} catch (IOException e) {
    // Exception handling code
}
```

Java 8 (2014) - Lambda Expressions and Functional Interfaces:

```
// Lambda expressions and functional interfaces can be used in exception handling
Function<Integer, Integer> divide = x -> {
    try {
        return 10 / x;
    } catch (ArithmeticException e) {
        // Exception handling code
        return 0;
    }
};
```

Java 9 (2017) - Private Methods in Try-With-Resources:

```
// Private methods in try-with-resources
private BufferedReader createReader() throws IOException {
    return new BufferedReader(new FileReader("file.txt"));
}

try (BufferedReader br = createReader()) {
    // Code that uses the resource
} catch (IOException e) {
    // Exception handling code
}
```

Java 12 (2019) - Switch Expressions:

```
int errorCode = 1;
String errorMessage = switch (errorCode) {
    case 1 -> "Error occurred";
    case 2 -> "Another error";
    default -> "Unknown error";
};
```

Java 16 (2021) - Pattern Matching for Switch:

```
Object obj = "Hello, World!";
if (obj instanceof String str) {
    // `str` is a local variable of type String within this block
    System.out.println(str.toUpperCase());
} else {
    System.out.println("Not a string");
}
```