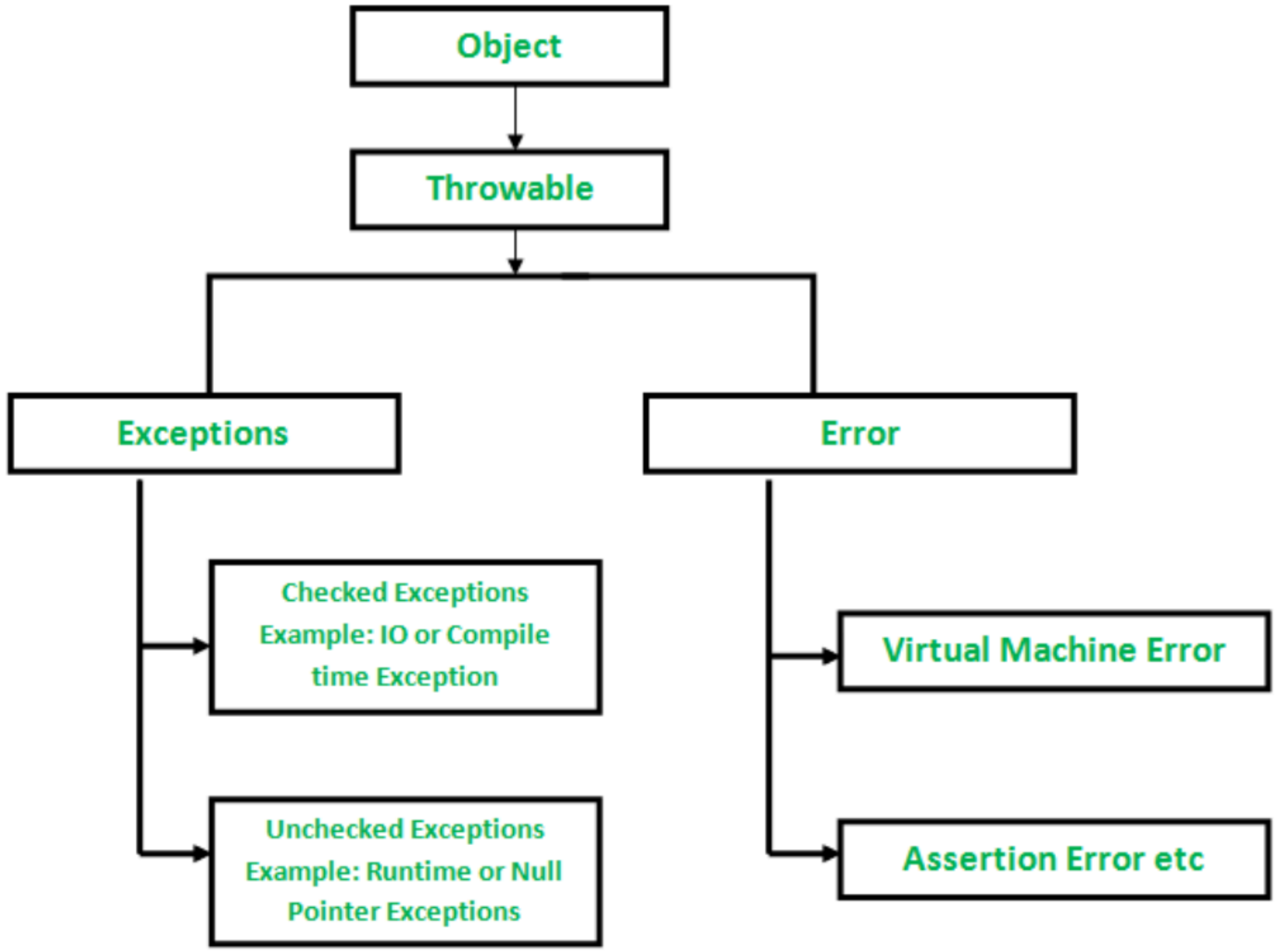
Java Course

Exception Handling

Exception as concept

- An exception is an unexpected event that occurs during program execution
- It affects the flow of the program instructions which can cause the program to terminate abnormally
- Exception can occur for many reasons:
  - Invalid user input
  - Device failure
  - Loss of network connection
  - Code errors
  - etc

Exception Hierarchy


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Exception Hierarchy - Errors

- `Throwable` is the root class of every type of exception
- Errors represent irrecoverable conditions such as JVM running out of memory, memory leaks, stack overflow errors, infinite recursion, etc
- Errors are usually beyond the control of the programmer and we should not try to handle Errors

Exception Hierarchy - Exceptions

- Exceptions can be caught and handled by the program
- When an exception occurs within a method, it creates an object
- This object is called the exception object
- Exception object contains information about the exception such as the name and description of the exception and state of the program when the exception occurred

Exception Types

- Two types of Exceptions:
  - Checked (compile-time) exceptions
  - Unchecked (runtime) exceptions

Checked Exceptions

Compile-time

- Checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions
- Examples:
  - `IOException`,
  - `ClassNotFoundException`,
  - `FileNotFoundException`

Unchecked Exceptions

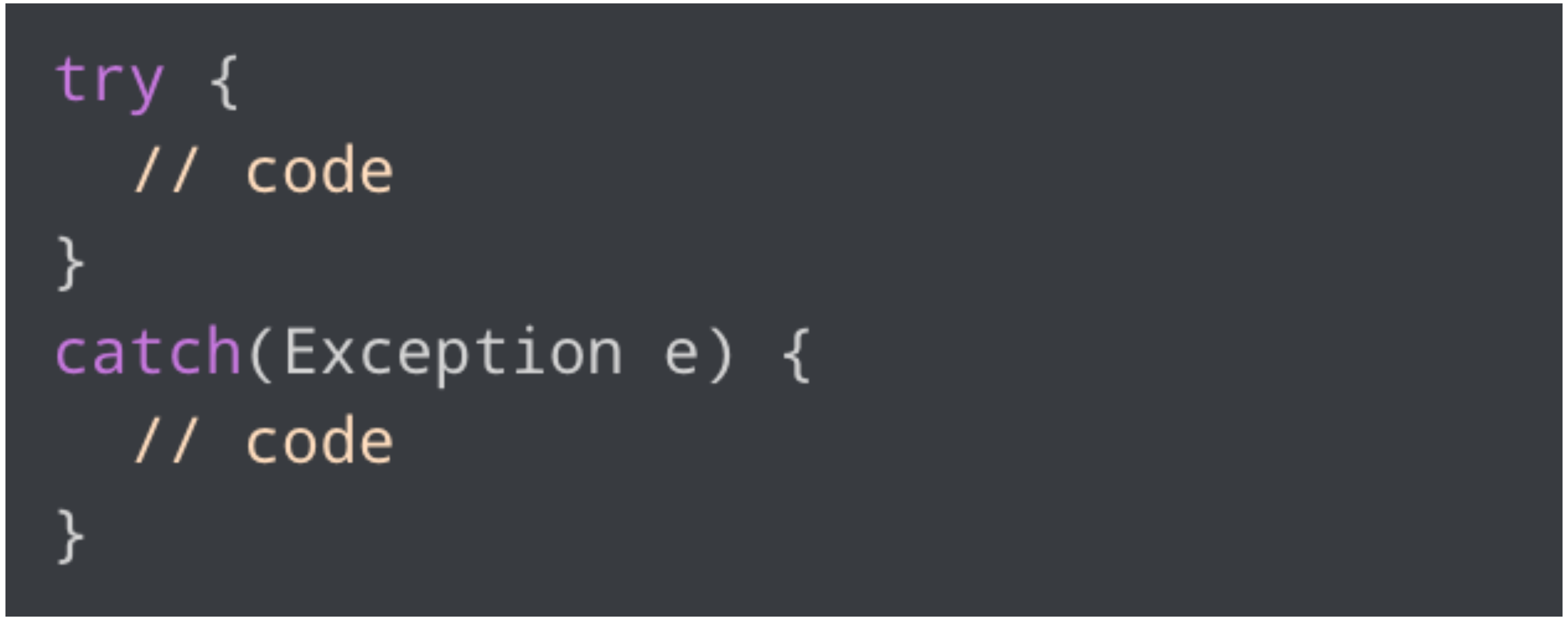
Runtime

- A runtime Exception happens due to a programming error
- These exceptions are checked at run-time
- Examples:
  - `ArithmeticException`,
  - `NullPointerException`,
  - `ArrayIndexOutOfBoundsException`, …

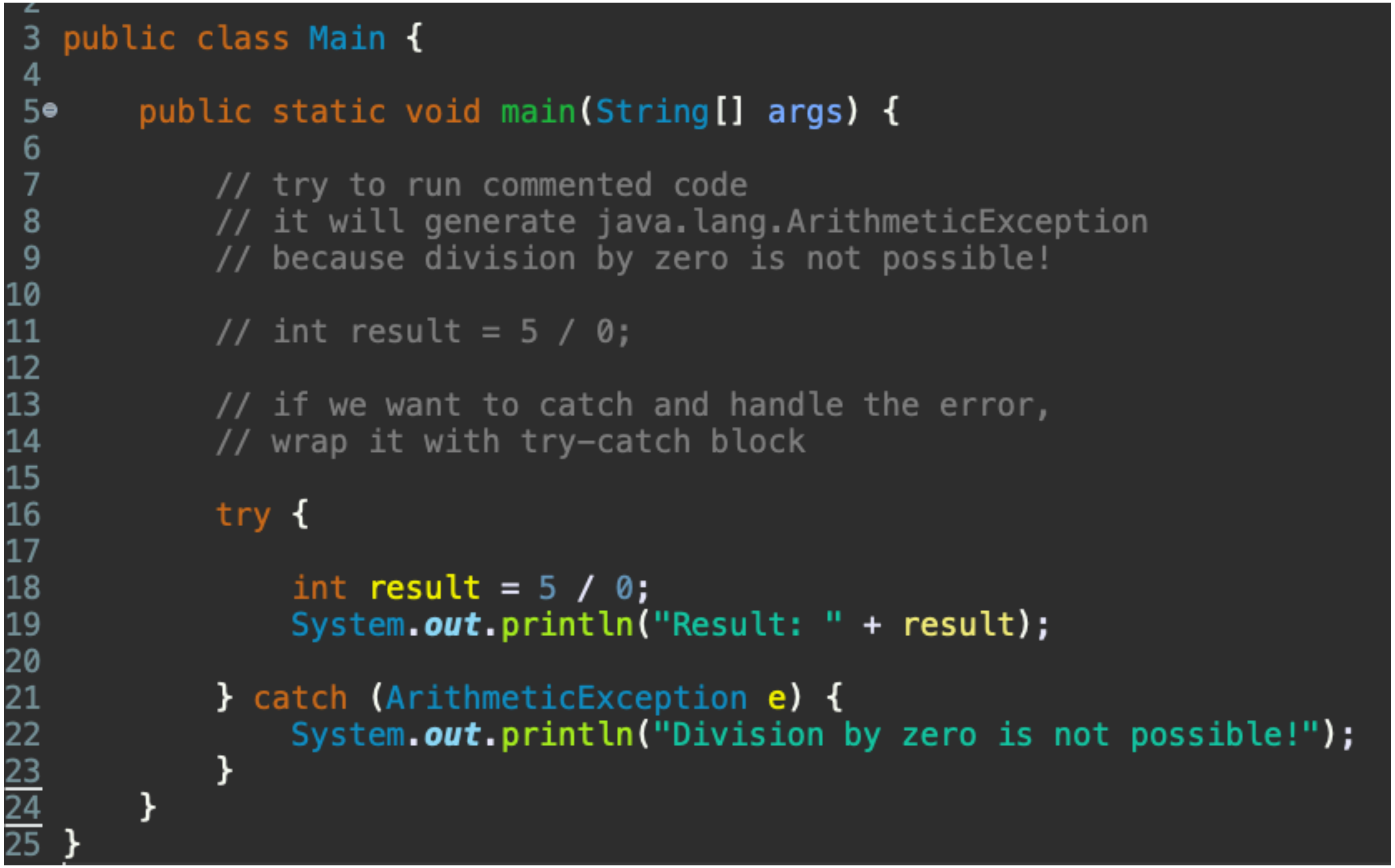
Exception Handling

try…catch…finally

- The `try-catch-finally` block is used to handle exceptions in Java



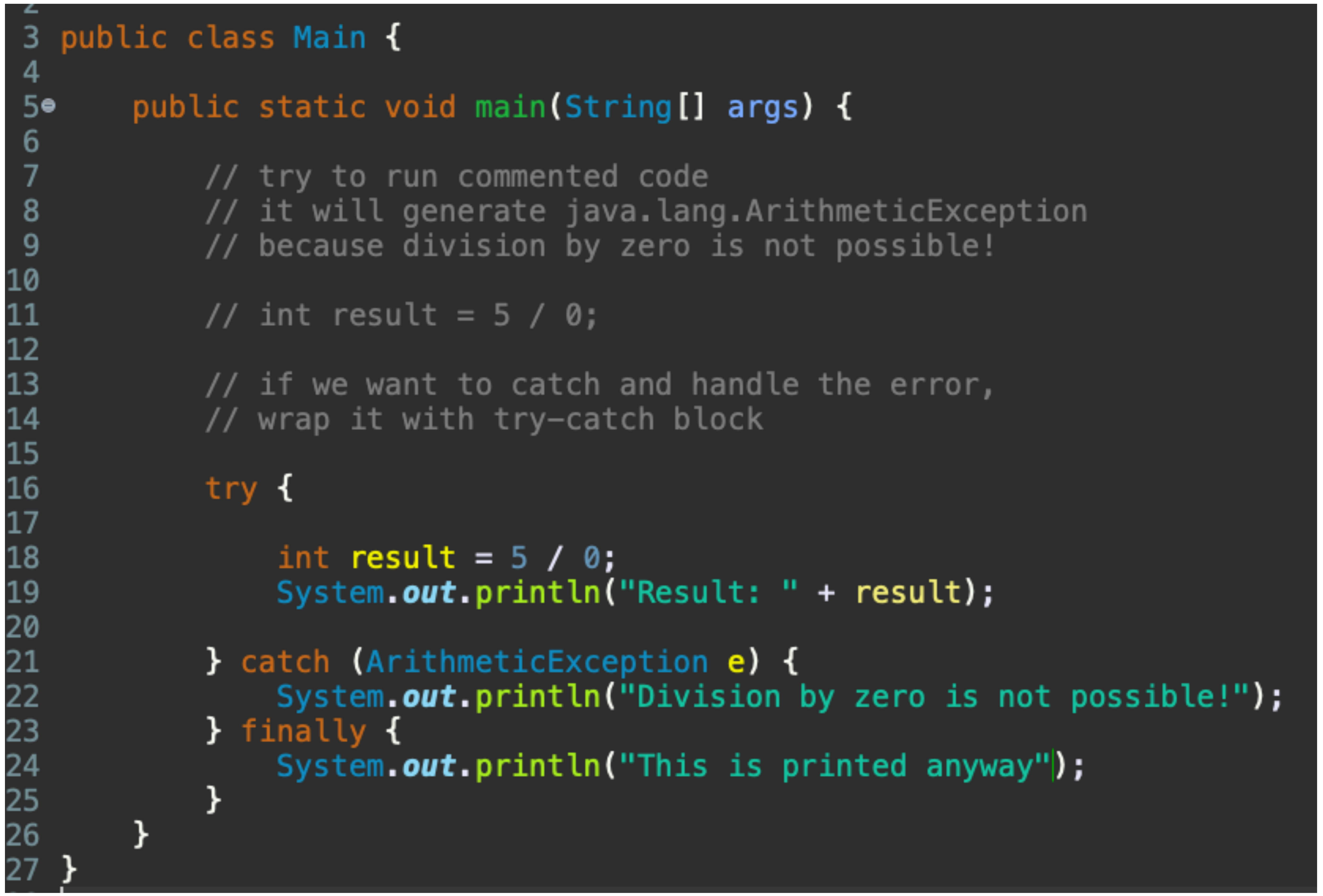
```
try {
  // code
}
catch(Exception e) {
  // code
}
```



```java
public class Main {

    public static void main(String[] args) {

        // try to run commented code
        // it will generate java.lang.ArithmeticException
        // because division by zero is not possible!

        // int result = 5 / 0;

        // if we want to catch and handle the error,
        // wrap it with try-catch block

        try {

            int result = 5 / 0;
            System.out.println("Result: " + result);

        } catch (ArithmeticException e) {
            System.out.println("Division by zero is not possible!");
        }
    }
}
```

`finally` block

- In Java, `finally` block is always executed no matter whether there is an exception or not
- This block is optional - but, there can be only one finally block for each try

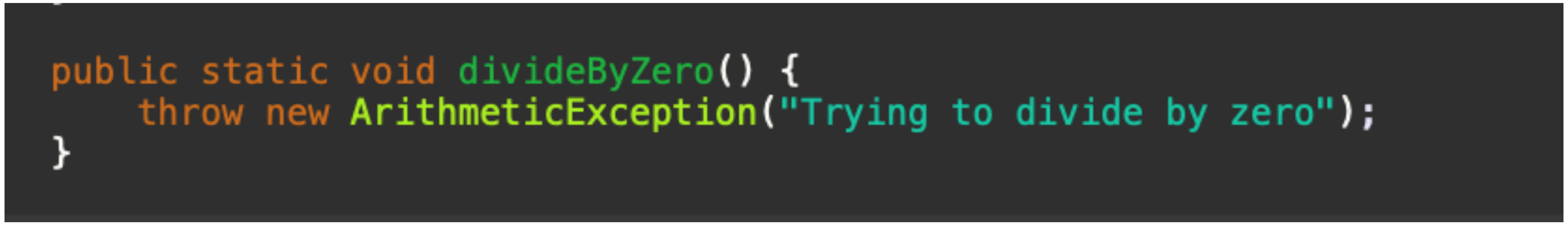


```java
public class Main {

    public static void main(String[] args) {

        // try to run commented code
        // it will generate java.lang.ArithmeticException
        // because division by zero is not possible!

        // int result = 5 / 0;

        // if we want to catch and handle the error,
        // wrap it with try-catch block

        try {

            int result = 5 / 0;
            System.out.println("Result: " + result);

        } catch (ArithmeticException e) {
            System.out.println("Division by zero is not possible!");
        } finally {
            System.out.println("This is printed anyway");
        }
    }
}
```

`throw` and `throws`

throw

- `throw` keyword is used to explicitly throw an exception from a method or any block of code
- We can throw checked or unchecked exception (custom exceptions as well)
- Mainly used for throwing custom exception, but it's not a rule

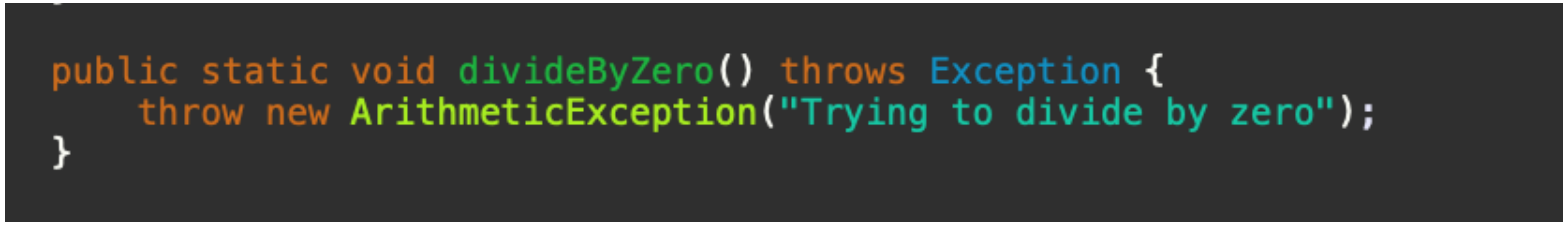


```java
public static void divideByZero() {
    throw new ArithmeticException("Trying to divide by zero");
}
```

`throw` and `throws`

throws

- Throws keyword is used in the signature of the method to indicate that the method might throw one of the listed exceptions
- It can be used to delegate the responsibility of exception handling to the caller of the method



```
public static void divideByZero() throws Exception {
    throw new ArithmeticException("Trying to divide by zero");
}
```