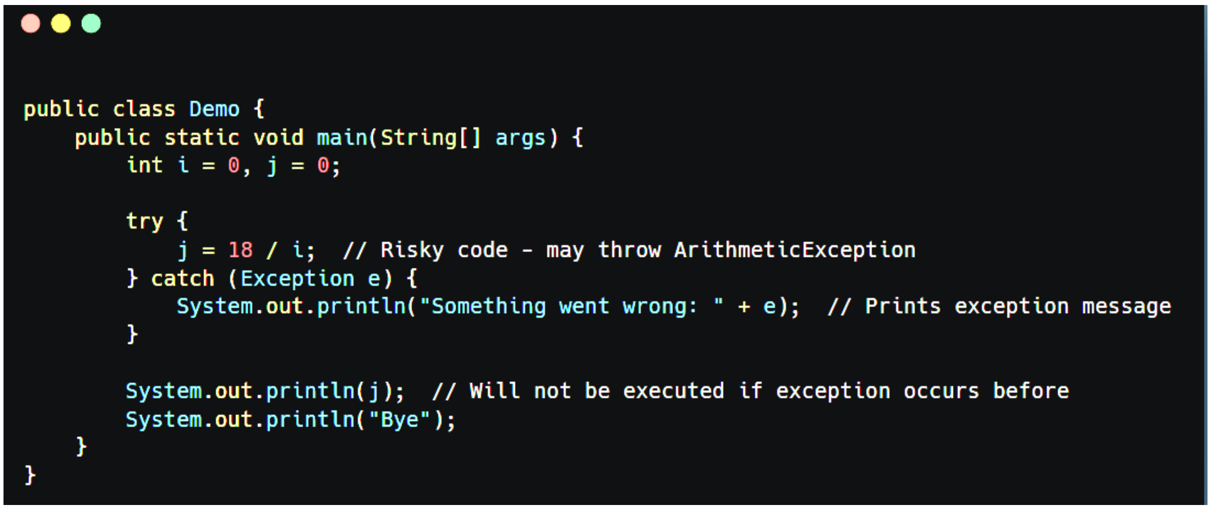
11.3-try with multiple catch

In Java, exceptions are used to handle runtime errors. Often, we may need to handle multiple types of exceptions that could occur within a program. Java provides a mechanism called "**multiple catch blocks**" to handle different exceptions separately and appropriately.

Exploring Exceptions

Example 1: Handling a Single Exception

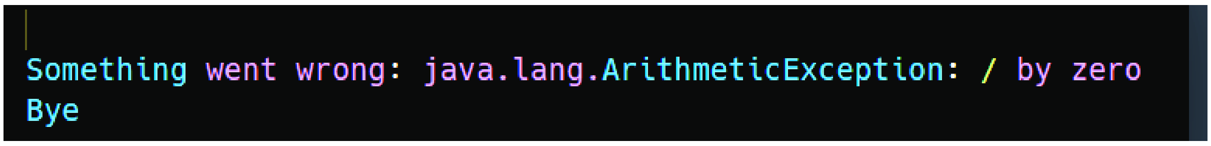


```java
public class Demo {
    public static void main(String[] args) {
        int i = 0, j = 0;

        try {
            j = 18 / i;  // Risky code - may throw ArithmeticException
        } catch (Exception e) {
            System.out.println("Something went wrong: " + e);  // Prints exception message
        }

        System.out.println(j);  // Will not be executed if exception occurs before
        System.out.println("Bye");
    }
}
```

Output:



```
Something went wrong: java.lang.ArithmeticException: / by zero
Bye
```

Explanation:

- In this example, we attempt to divide by zero, which triggers an ArithmeticException.
- The exception is caught in the catch block, and the message is displayed: "Something went wrong: / by zero".
- The program continues and prints "Bye", preventing abrupt termination due to the error.

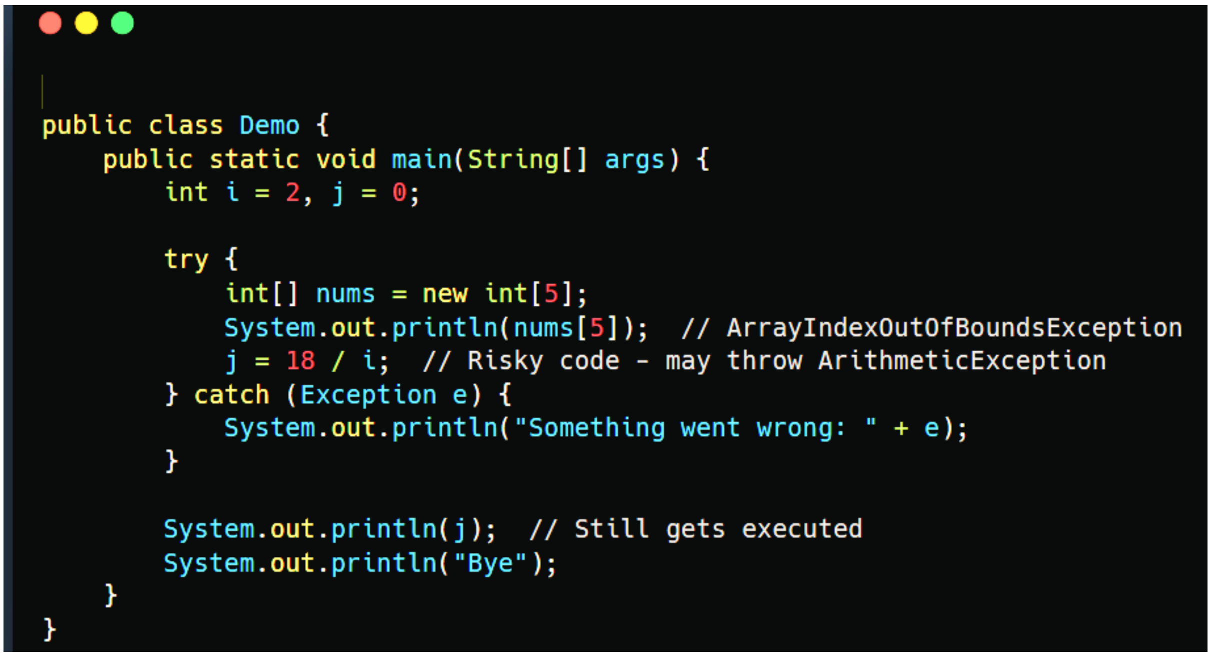
Key Concept:

- The exception object (e) in the catch block provides useful information about the specific error, making debugging easier. Instead of terminating the program, we handle the error gracefully and allow subsequent lines to execute.

Modifying the Code to Add More Exceptions

Now, let’s add a few more risky operations to demonstrate how different exceptions can occur.

Example 2: Handling Another Type of Exception

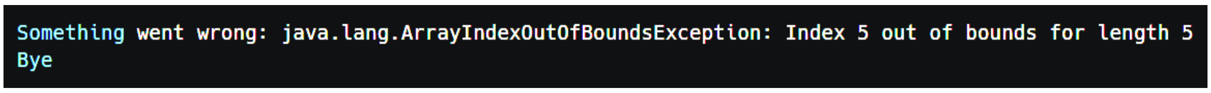


```java
public class Demo {
    public static void main(String[] args) {
        int i = 2, j = 0;

        try {
            int[] nums = new int[5];
            System.out.println(nums[5]);  // ArrayIndexOutOfBoundsException
            j = 18 / i;  // Risky code - may throw ArithmeticException
        } catch (Exception e) {
            System.out.println("Something went wrong: " + e);
        }

        System.out.println(j);  // Still gets executed
        System.out.println("Bye");
    }
}
```

Output:



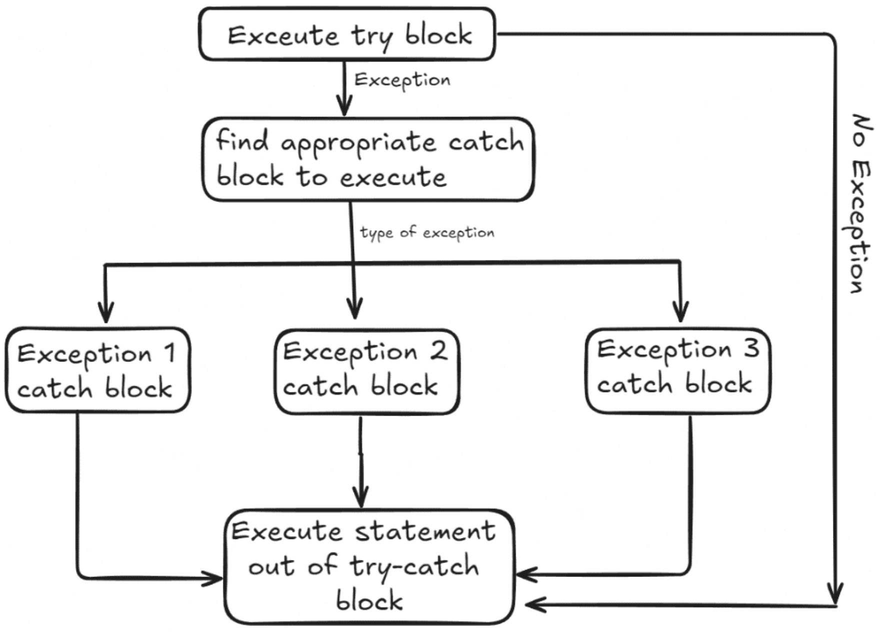
```
Something went wrong: java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
Bye
```

Explanation:

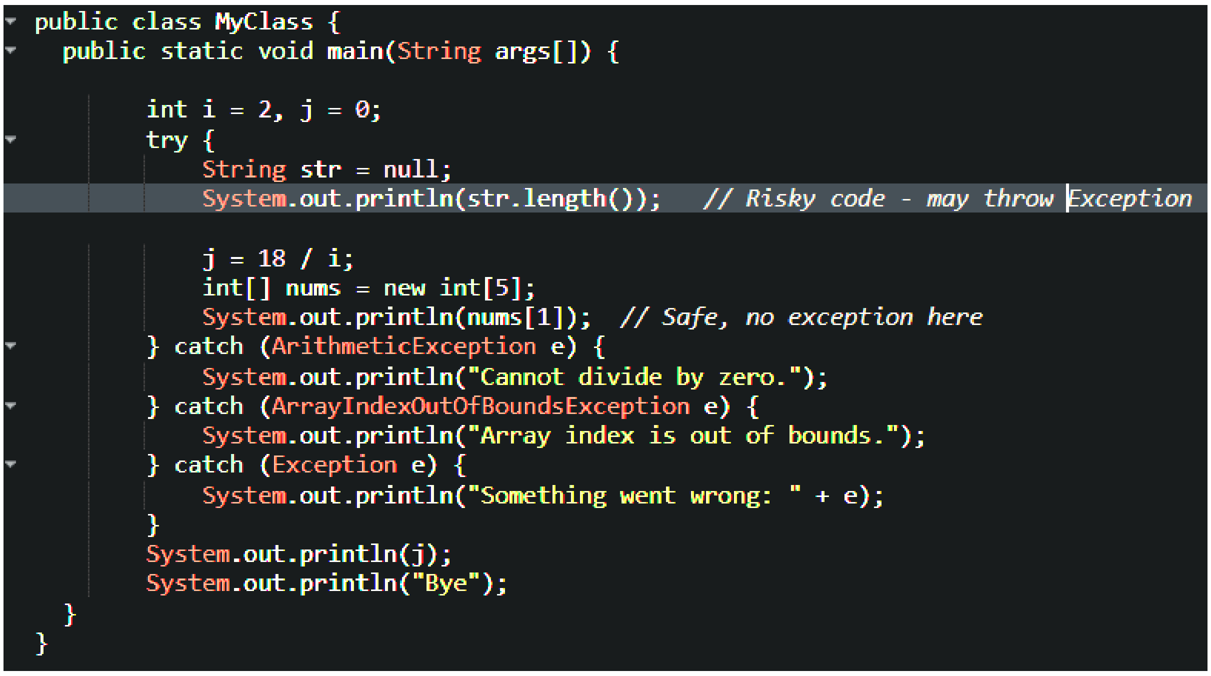
- We attempted to access an array element beyond its valid range, causing an ArrayIndexOutOfBoundsException.
- This error is caught by the catch block, which prints the error message, and the program continues to execute normally.

Try with Multiple Catch Blocks

When multiple types of exceptions may occur in a try block, we can handle them using **multiple catch blocks**. This way, we can provide specific handling for each exception type.



Example 3: Handling Multiple Exceptions Separately

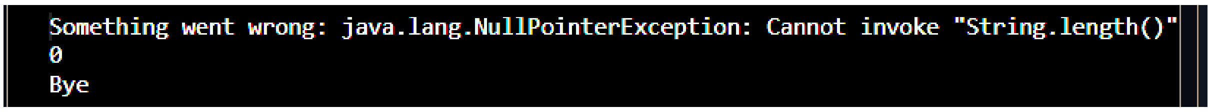


```java
public class MyClass {
  public static void main(String args[]) {

      int i = 2, j = 0;
      try {
          String str = null;
          System.out.println(str.length());   // Risky code - may throw Exception

          j = 18 / i;
          int[] nums = new int[5];
          System.out.println(nums[1]);  // Safe, no exception here
      } catch (ArithmeticException e) {
          System.out.println("Cannot divide by zero.");
      } catch (ArrayIndexOutOfBoundsException e) {
          System.out.println("Array index is out of bounds.");
      } catch (Exception e) {
          System.out.println("Something went wrong: " + e);
      }
      System.out.println(j);
      System.out.println("Bye");
  }
}
```

Output:



```
Something went wrong: java.lang.NullPointerException: Cannot invoke "String.length()"
0
Bye
```

Explanation:

- In this example, we have multiple catch blocks, and a new kind of Exception i.e **NullPointerException** is emerged which the Generic Exception catah block has caught it .Here, each catchdesigned to handle a specific type of exception:
    - **NullPointerException**: Occurs when trying to access the properties of a null object (str.length()).
    - **ArithmeticException**: Handles division by zero.
    - **ArrayIndexOutOfBoundsException**: Handles invalid array indexing.
    - **Exception**: This is the parent class for all exceptions. It catches any remaining exceptions that aren't handled by the previous catch blocks.

Key Takeaway:

- Specific catch blocks handle different exceptions, providing appropriate error messages.
- A final generic Exception catch block handles any other exceptions that aren't anticipated, ensuring the program remains robust.

Why Use Multiple Catch Blocks?

Handling exceptions with multiple catch blocks offers several advantages:

- **Specific Handling:** Each exception can be dealt with individually, allowing for more specific and meaningful error messages.
- **Improved Debugging:** By knowing exactly which type of error occurred, debugging becomes easier.
- **Code Continuation:** Despite encountering an exception, the program can continue to execute following the catch block, preventing abrupt termination.

Common Exceptions in Java

Here are a few of the most common exceptions that developers handle using try-catch blocks:

- **ArithmeticException:** This exception is thrown when an exceptional arithmetic condition occurs, such as division by zero.
- **ArrayIndexOutOfBoundsException:** This is thrown when an array is accessed with an illegal index. The index might be negative or greater than or equal to the array's size.
- **NullPointerException:** This is raised when an attempt is made to access an object or method on a null reference.

Exception Hierarchy in Java

Java exceptions follow a hierarchy. Understanding this structure helps in catching specific exceptions and writing cleaner code.

- **Throwable**
  - **Exception**
    - **RuntimeException**
      - **ArithmeticException**
      - **ArrayIndexOutOfBoundsException**
      - **NullPointerException**

Key Points:

- **Exception** is the parent class for all exceptions.
- **RuntimeException** and its subclasses are unchecked exceptions, meaning they don't need to be explicitly caught or declared.
- **Throwable** is the top-level parent for both exceptions and errors.