
# Types of Exceptions in Java

Exceptions are a vital part of Java programming. Let's explore the different types of exceptions and learn how to handle them effectively.

# Checked Exceptions

## 1 What are they?

Checked exceptions are standard exceptions that are checked at compile time, requiring you to declare them in your source code.

## 2 Examples

FileNotFoundException, SQLException, IOException

## 3 How to Handle Them

You need to handle checked exceptions using a try-catch block or by declaring them with the 'throws' keyword.

# Example:

```java
/*
 * @author Edison Verdesoto, Code Warriors, DCCO-ESPE
 */
public class CheckedException {
    public static void main(String[] args) {
     Scanner scanner = new Scanner(source:System.in);
       String option;
       do {
           try {
               System.out.print(s: "Enter age: ");
               int age = scanner.nextInt();
               scanner.nextLine();
               validateAge(age);
               System.out.println(x: "Valid age!");
           } catch (InvalidAgeException e) {
               System.out.println("Invalid age: " + e.getMessage());
           }

           System.out.print(s: "Do you want to enter another age? (y/n): ");
           option = scanner.nextLine();
       } while (option.equalsIgnoreCase(anotherString:"y"));

       scanner.close();
   }

   public static void validateAge(int age) throws InvalidAgeException {
       if (age < 18) {
           throw new InvalidAgeException(message: "Age must be 18 or above.");
       }
   }
}

class InvalidAgeException extends Exception {
   public InvalidAgeException(String message) {
       super(message);
   }
}
```

# Unchecked Exceptions

## What are they?

Unchecked exceptions are runtime exceptions that are not checked at compile time. They can occur anywhere in your program.

## How to Handle Them

Since unchecked exceptions are not checked at compile time, you can handle them using a try-catch block or let them propagate up the call stack.

## Examples

NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException

# Example:

```java
/*
 * @author Edison Verdesoto, Code Warriors, DCCO-ESPE
 */
public class CustomExceptions {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(source:System.in);
        String option;
        do {
            try {
                System.out.print(s: "Enter dividend: ");
                int dividend = scanner.nextInt();
                System.out.print(s: "Enter divisor: ");
                int divisor = scanner.nextInt();
                scanner.nextLine();
                int result = performDivision(dividend, divisor);
                System.out.println("Result: " + result);
            } catch (ArithmeticException e) {
                System.out.println(x: "Error: Division by zero is not allowed.");
            }

            System.out.print(s: "Do you want to perform another division? (y/n): ");
            option = scanner.nextLine();
        } while (option.equalsIgnoreCase(anotherString: "y"));

        scanner.close();
    }

    public static int performDivision(int dividend, int divisor) {
        return dividend / divisor;
    }
}
```

# Throw vs. Throws

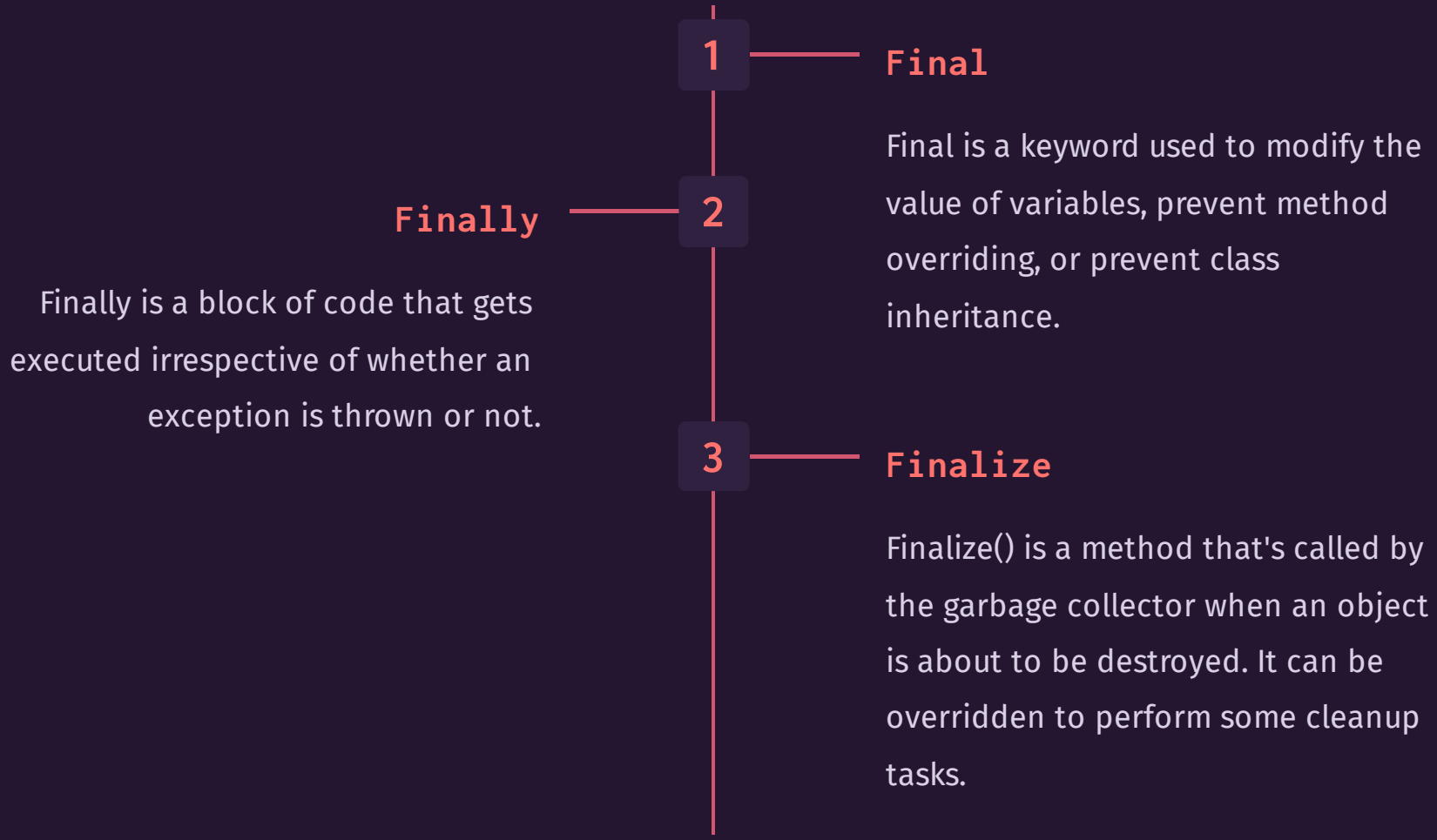
### Throw

Throw is used to throw a new exception explicitly in your code.

### Throws

Throws can be used to declare the exceptions a method can throw. This is useful when a method does not handle a particular exception, but wants to leave it to the calling method.

# Final vs. Finally vs Finalize



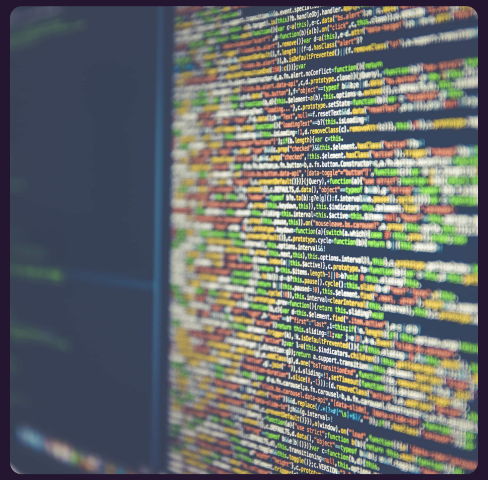
# Exception Handling Best Practices

## Small, Specific Try-Catch Blocks

Try to keep your try-catch blocks small and specific to the code that can potentially throw an exception.

## Log Your Exceptions

Logging your exceptions can help you pinpoint the root cause of the exception and find quick solutions.



## Test Your Code for Exceptions

Testing your code for exceptions can help you identify and fix problems before they get to production.

# Common Exception Handling Mistakes

## Ignoring Exceptions

Ignoring exceptions can cause your code to fail silently, making it difficult to find and debug issues.

## Catching General Exceptions

Catching general exceptions like Exception or Throwable can catch all exceptions, including unchecked exceptions, and make it harder to pinpoint the root cause of the problem.

## Not Wrapping Exceptions

Not wrapping your exceptions in a custom exception class can make it difficult to understand the cause of the exception and lead to unclear solutions.