17.1 Exception handling using try-catch-finally

🎯 What is `try-catch`?

The `try-catch` block is used to handle **exceptions (runtime errors)** that occur during the execution of a program. It **prevents abnormal termination** of the program.

📌 Basic Syntax

```
try {
    // Code that may throw exception
} catch (ExceptionType e) {
    // Handling code
}
```

✅ Example: Handling Division by Zero

```
public class Example {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        }
    }
}
```

Output:

```
Cannot divide by zero!
```

🔄 `finally` Block

- The `finally` block **always executes**, whether or not an exception occurred.
- It is used for **resource cleanup**, like closing files or database connections.

```
try {
    // risky code
} catch (Exception e) {
    // handling
} finally {
    // always executed
}
```

🔥 All Scenarios & Variations

✅ 1. try-catch-finally (Most Common)

```java
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Handled exception");
} finally {
    System.out.println("Cleanup done");
}
```

📝 Use this when you want to **handle** the exception **and** perform **mandatory cleanup**.

✅ 2. try-finally (No catch)

```java
try {
    int a = 10 / 0;
} finally {
    System.out.println("This always runs");
}
```

❗ This compiles and runs, but if an exception occurs, and there is **no** `catch`, the program will crash **after** executing the `finally` block.

❌ 3. catch without try (Invalid)

```java
catch (Exception e) {
    // ❌ This won't compile
}
```

❌ You **cannot use** `catch` **without** `try`. It will result in a compile-time error.

❌ 4. finally without try (Invalid)

```java
finally {
    // ❌ This won't compile
}
```

❌ The `finally` block **must be used with** `try`. Alone, it is a compile-time error.

✅ 5. Multiple Catch Blocks

You can catch different types of exceptions separately.

```java
try {
    String str = null;
    System.out.println(str.length());
} catch (ArithmeticException e) {
    System.out.println("Math error");
} catch (NullPointerException e) {
    System.out.println("Null reference error");
}
```

📝 Catch **more specific exceptions** before general ones.

✅ 6. Multi-catch (Java 7+)

You can catch **multiple exceptions** in a single catch block using `|`.

```java
try {
    int[] arr = new int[3];
    arr[5] = 10;
} catch (ArithmeticException | ArrayIndexOutOfBoundsException e) {
    System.out.println("Exception caught: " + e);
}
```

🚫 In multi-catch, the variable `e` is **effectively final** — you can't reassign it.

✅ 7. Nested try-catch

You can nest try-catch blocks inside each other.

```java
try {
    try {
        int a = 10 / 0;
    } catch (ArithmeticException e) {
        System.out.println("Inner catch");
    }
} catch (Exception e) {
    System.out.println("Outer catch");
}
```

Use case: When a block of code inside `try` also needs individual handling.

✅ 8. Multiple try blocks

Multiple `try-catch` blocks can exist **separately** in a method.

```java
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Math error");
}

try {
    String str = null;
    System.out.println(str.length());
} catch (NullPointerException e) {
    System.out.println("Null error");
}
```

📝 Each try block is **independent**.

❓ Can `try` block exist alone?

No. It must be followed by either:

- `catch`
- `finally`
- Or both

```java
try {
    // risky code
}
// ✅ must follow with catch or finally
```

🧪 `finally` Block Behavior

Case 1: Exception thrown, caught

```java
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Caught");
} finally {
    System.out.println("Finally always runs");
}
```

Output:

```
Caught
Finally always runs
```

Case 2: Exception not thrown

```java
try {
    int a = 10 / 2;
} catch (ArithmeticException e) {
    System.out.println("Caught");
} finally {
    System.out.println("Finally always runs");
}
```

Output:

```
Finally always runs
```

Case 3: Exception thrown but not caught

```java
try {
    int a = 10 / 0;
} finally {
    System.out.println("Cleanup even on crash");
}
```

Output:

```
Cleanup even on crash
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

✅ Finally block runs, then the program crashes.

🚫 Common Mistakes

Mistake	Why It's Wrong
`catch` without `try`	Not allowed
`finally` without `try`	Not allowed
Writing `catch` after `finally`	Order must be `try → catch → finally`
Catching `Throwable` or `Exception` too early	Prevents handling of specific exceptions

---

📌 Best Practices

- Catch **specific exceptions first**, general later.
- Always use `finally` for cleanup.
- Never leave catch block empty.
- Use **multi-catch** for cleaner code when needed.
- Avoid using `Exception` as a generic catch unless logging or re-throwing.

🔚 Summary Table

Structure	Allowed?	Notes

`try-catch`	✅ Yes	Basic handling
`try-catch-finally`	✅ Yes	Full error handling + cleanup
`try-finally`	✅ Yes	No catch, but must handle outside
`catch` alone	❌ No	Syntax error
`finally` alone	❌ No	Syntax error
Multiple catch	✅ Yes	Each for different exception
Nested try-catch	✅ Yes	Localized handling
Multiple try blocks	✅ Yes	For separate parts of code

🧠 Final Thoughts

Exception handling in Java gives you control over **what happens when something goes wrong**. Mastering `try`, `catch`, and `finally` is crucial for writing **robust and reliable applications**.

> "Don't just handle exceptions—**understand** them. That's where true debugging power lies."

Document By **Suyash** 😇