Exception Handling in Java

Java provides a powerful mechanism to handle runtime errors so that the **normal flow of the program is maintained**.
Main keywords: **try catch throw finally** (and throws).

1. try

- Used to wrap code that **may cause an exception**.
- Must be followed by either `catch` or `finally`.

Example:

```java
try {
    int a = 10 / 0;    // risky code
}
```

2. catch

- Handles the exception thrown by the `try` block.
- Can have multiple `catch` blocks for different exceptions.

Example:

```java
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero!");
}
```

3. throw

- Used to **explicitly throw an exception**.
- Usually for custom or specific exception cases.

Example:

```
public class ThrowExample {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Not eligible to vote!");
        } else {
            System.out.println("Eligible to vote.");
        }
    }
    public static void main(String[] args) {
        checkAge(15);  // will throw exception
    }
}
```

4. finally

- Always executed whether exception occurs or not.
- Used for cleanup (closing files, DB connections, etc.).

Example:

```
try {
    int a = 10 / 2;
    System.out.println("Result: " + a);
} catch (ArithmeticException e) {
    System.out.println("Error: " + e);
} finally {
    System.out.println("Finally block executed.");
}
```

Flow Example (All Together)

```java
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            int num = 5 / 0;    // risky code
        } catch (ArithmeticException e) {
            System.out.println("Caught: " + e);
        } finally {
            System.out.println("This will always run.");
        }
    }
}
```

Summary Table

Keyword	Purpose
try	Defines risky code block that may cause exception.
catch	Handles the exception.
throw	Used to explicitly throw an exception.
finally	Always executes (cleanup code).