What is Exception Handling?

- **Exceptions** are events that **disrupt the normal flow of a program**.
- Java provides a **robust exception handling mechanism** using `try`, `catch`, `finally`, `throw`, and `throws`.

---

2 Types of Exceptions

Type	Description	Examples
Checked Exception	Must be declared in method signature or handled	`IOException`, `SQLException`
Unchecked Exception	Runtime exceptions; compiler doesn't force handling	`NullPointerException`, `IllegalArgumentException`
Errors	Serious system errors, not meant to be caught	`OutOfMemoryError`, `StackOverflowError`

---

3 Basic Exception Handling Syntax

```
try {

    int result = 10 / 0; // may throw ArithmeticException

} catch (ArithmeticException e) {

    System.out.println("Cannot divide by zero: " + e.getMessage());

} finally {
    System.out.println("Finally block always executes");
}
```

- `try` → code that may throw exception
- `catch` → handle the exception
- `finally` → optional, executes **always**, used for cleanup

4️⃣ Strategies for Exception Handling

a) Catch only what you can handle

- Don’t catch generic `Exception` unless necessary.
- Example of bad practice:

```
try {
    // some code
} catch (Exception e) { // ❌ too generic
    e.printStackTrace();
}
```

- Better:

```
try {
    // some code
} catch (IOException e) {
    System.out.println("File not found: " + e.getMessage());
}
```

---

b) Use `throws` to delegate

- If a method cannot handle an exception, declare it in the signature:

```
public void readFile(String path) throws IOException {
    Files.readAllLines(Paths.get(path));
}
```

- Caller must handle or further propagate it.

---

c) Don't suppress exceptions silently

- Avoid empty catch blocks:

```
catch (IOException e) { } // ❌
```

- Always log or rethrow, or handle meaningfully.

---

d) Use try-with-resources for AutoCloseable

- For I/O, DB connections, etc., this ensures resources are closed automatically:

```
try (BufferedReader reader = new BufferedReader(new
FileReader("file.txt"))) {

    System.out.println(reader.readLine());

} catch (IOException e) {

    e.printStackTrace();

}
```

- Equivalent to `finally` closing, but cleaner.

---

e) Wrap exceptions for clarity

- For libraries or APIs, wrap low-level exceptions in **custom exceptions**:

```
public void process() {

    try {

        riskyOperation();

    } catch (SQLException e) {

        throw new DataProcessingException("Failed to process data",
e);

    }

}
```

- Keeps your API clean and meaningful.

---

f) Use unchecked exceptions for programming errors

- Use `IllegalArgumentException`, `IllegalStateException` for **invalid inputs or illegal state**.
- Checked exceptions should represent **recoverable conditions**, unchecked for **programming mistakes**.

5️⃣ Best Practices

1. **Be specific** with exception types.
2. **Handle exceptions as close as possible** to where they occur.
3. **Don't use exceptions for control flow**.
4. **Always clean up resources** (`try-with-resources` or `finally`).
5. **Log exceptions** with meaningful messages.
6. **Propagate if you cannot handle**.
7. **Prefer immutable custom exception classes** if creating your own.

6️⃣ Modern Example Combining Best Practices

```
public class FileProcessor {



    public void processFile(String path) {

        try (BufferedReader reader = new BufferedReader(new
FileReader(path))) {

            String line;

            while ((line = reader.readLine()) != null) {
                System.out.println(line);

            }

        } catch (FileNotFoundException e) {

            System.err.println("File not found: " + path);

        } catch (IOException e) {

            throw new RuntimeException("Error reading file: " + path,
e);

        }

    }

}
```

- Uses **try-with-resources**
- **Specific exceptions** caught first
- **Wraps IOExceptions** in unchecked exception to propagate meaningful info

---

in some cases finally block does not execute:
i. Use of System.exit() // will exit the process
2. Some exception occur in the finally block
3 If death of thread

we uses the throws keyWord in the function declaration , which tells this function might throw the exception , then this function should be placed in the try and catch block as it can throw an exception
throw keyWord is used to throw the exception manually