17. Exception Handeling

🧭 What is an Exception in Java?

In Java, an **exception** is an **unexpected or abnormal condition** that occurs **during program execution** (runtime), which **disrupts the normal flow** of the program.

🚨 Example:

```java
int a = 10 / 0;
```

This line will throw:

```
java.lang.ArithmeticException: / by zero
```

Because dividing by zero is **not allowed**, the program throws an **exception** instead of crashing the system.

🎯 Why is Exception Handling Needed?

Java provides a powerful mechanism called **exception handling** to deal with runtime issues **gracefully**. Without it, your program may:

- Crash suddenly
- Lose unsaved data
- Behave unpredictably

✅ Benefits of Exception Handling:

Benefit	Description
🚫 Prevents program crash	Errors can be caught and handled properly
📜 Shows meaningful messages	Helps users and developers understand issues
🔄 Allows graceful recovery	You can retry operations or log for analysis
🎯 Separates error logic	Clean code — error logic is kept in one place

---

❗ What is an Error in Java?

An **Error** in Java is a **serious issue** that typically occurs due to problems **outside your code's control**, such as:

- System crashes
- Out of memory
- Stack overflow

🔥 Common Examples:

Error Name	Description
`OutOfMemoryError`	JVM runs out of memory
`StackOverflowError`	Too much recursion (infinite method calls)
`NoClassDefFoundError`	Required class not found at runtime

These are subclasses of the `java.lang.Error` class and **should not be handled using try-catch**. They are usually **fatal**.

🧬 Java Exception Hierarchy

Detailed explanation of hierarchy will be provided in session

```
Object
 └── Throwable
       ├── Exception (Recoverable)
       │     ├── Checked (IOException)
       │     └── Unchecked (NullPointerException)
       └── Error (Unrecoverable)
```

⚖ Difference Between Error and Exception

Feature	Exception	Error
📌 Base Class	`java.lang.Exception`	`java.lang.Error`
📦 Package	`java.lang`	`java.lang`
🔄 Recoverable?	✅ Usually recoverable	❌ Usually not recoverable

🛠️ Example	`IOException`, `ArithmeticException`	`OutOfMemoryError`, `StackOverflowError`
🎯 Handling Mechanism	Use `try-catch`	Usually not handled (shouldn't be)
👨‍💻 Caused by	Application logic	JVM or system-level issues
✅ Programmer Fixable	Yes	No (most of the time)

📌 When to Use Try-Catch?

Use **try-catch** to handle **exceptions**, not **errors**.

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

❌ Do **not** try to catch `OutOfMemoryError` or `StackOverflowError`. These need architectural fixes, not exception handling.

🧠 Real-World Analogy

Imagine you're cooking:

- **Exception**: You forgot to add salt — the food is bad but you can add it later and fix it.
- **Error**: The stove exploded — you cannot continue cooking. Time to call emergency services!

🧠 Summary

Term	What is it?	Should You Catch It?	Example
Exception	Recoverable issue in code logic	✅ Yes	`IOException`, `NPE`
Error	Critical issue with JVM/system	❌ No	`OutOfMemoryError`, `SOError`

💬 Final Thoughts

Understanding the difference between **exceptions and errors** is **crucial** for writing reliable Java programs. Remember:

> 🔧 **Exceptions** = Problems you can fix
>
> 🔥 **Errors** = Problems you should avoid

Let the JVM handle **Errors**, and you handle **Exceptions**.

Document by ***Suyash*** 😇