# 11.4- Exception Hierarchy

## Exception Hierarchy in Java

### Introduction:

In Java, all classes inherit from the topmost superclass, the Object class. This class is implicitly extended by all other classes, and the same applies to exception handling. Throwable class is the root class of Java Exception hierarchy and immediate parent of all exceptions.

**Note**: Throwable is the only class in Java whose name ends with "able" and is a class, whereas other terms ending with "able" like Runnable, Callable, and Serializable are interfaces.

### Throwable Class Hierarchy:

The Throwable class has two main subclasses:

1. **Error**
2. **Exception**

---

### 1. Error

Errors are severe problems that arise during program execution, which generally cannot be resolved. When an error occurs, it usually leads to program termination. Errors are not meant to be handled in a program.

### Examples of Errors:

- **Thread Death**: Occurs when a thread (the smallest unit of a program) dies unexpectedly.
- **I/O Error**: Issues related to input/output operations.
- **Virtual Memory Error**: Occurs due to insufficient memory. This can also lead to an **OutOfMemoryError**, which is an internal problem that can't be handled within the program.

---

## 2. Exception

Exceptions are events that can be caught and handled by the program. They occur due to unexpected or abrupt execution of the program. Once an exception occurs, any lines of code following it are not executed unless the exception is properly handled.

### Types of Exceptions:

Exceptions are categorized into two main types:

1. **Checked Exceptions**
2. **Unchecked Exceptions (Runtime Exceptions)**

### Checked Exceptions:

These exceptions are known to the compiler and must be handled; otherwise, they will result in a compile-time error. You can either handle them using try-catch blocks or "duck" them by using the throws keyword.

- **Example Checked Exceptions**:
  - IOException
  - SQLException
  - ClassNotFoundException

**Key Point**: Checked exceptions are called "**compile-time exceptions**" because the compiler forces you to handle them.

### Unchecked Exceptions (Runtime Exceptions):

These exceptions occur at runtime, after the program has successfully compiled. If an unchecked exception occurs and it is not properly handled, the program may crash.

- **Example Unchecked Exceptions**:
  - ArithmeticException
  - NullPointerException
  - ArrayIndexOutOfBoundsException

Unchecked exceptions do not need to be declared or handled at compile-time. The program will not show any errors at compile-time even if they are not handled.

---

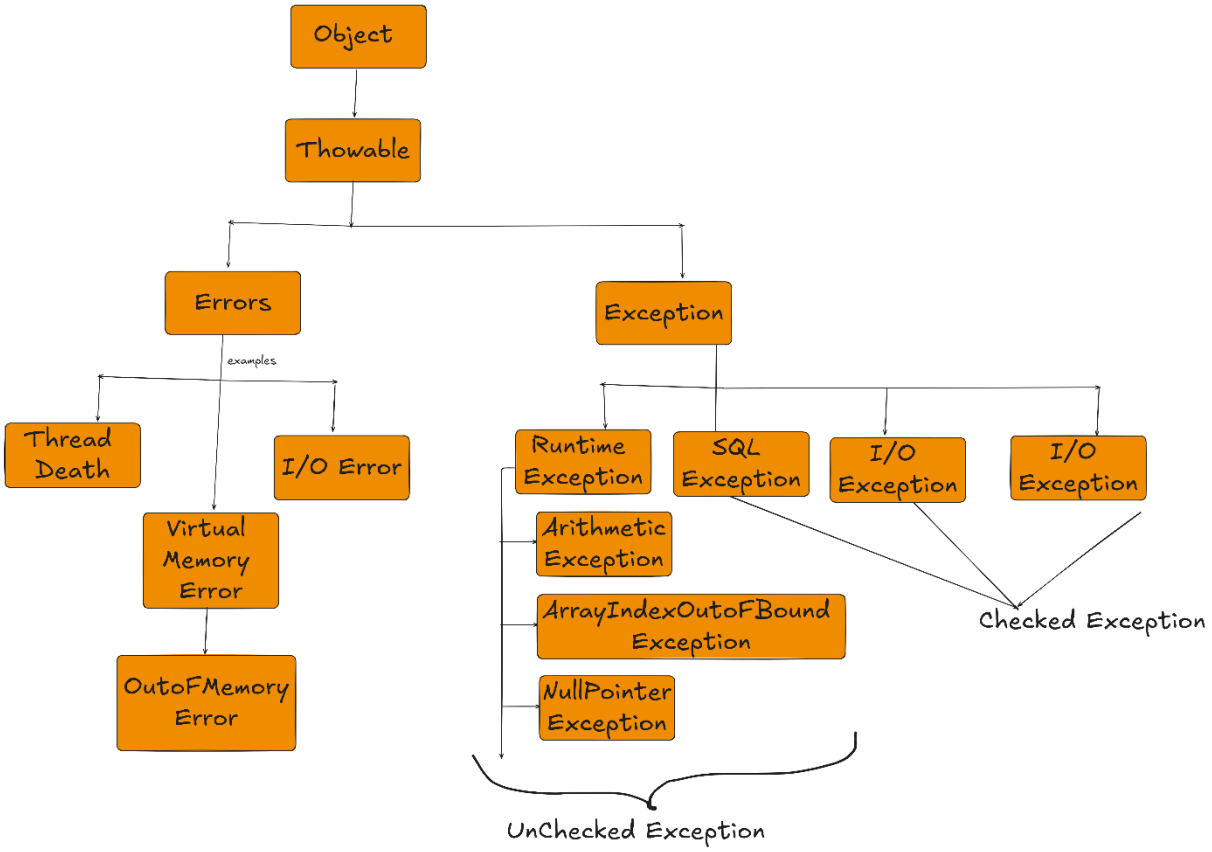
**Runtime Exceptions:**

Runtime exceptions occur after the compilation of the program. If not handled by the program, they are caught by the default handler (JVM exception handler). However, it's a bad practice to rely on the default handler, and you should handle these exceptions in your code.

**Common Runtime Exceptions**:

- **ArithmeticException:** Occurs when there is a mathematical error, like dividing by zero.
- **ArrayIndexOutOfBoundsException:** Occurs when you try to access an index outside the bounds of an array.
- **NullPointerException:** Occurs when you try to use a null object reference.

---



**Differences Between Checked and Unchecked Exceptions:**

| Aspect | Checked Exceptions | Unchecked Exceptions |
|---|---|---|
| **Known at Compile-Time** | Yes, must be handled by the programmer. | No, known only at runtime. |
| **Handling** | Must be handled or declared using throws. | Can be handled optionally; no compile error. |
| **Examples** | IOException, SQLException, ClassNotFoundException | NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException |
| **Category** | Known as compile-time exceptions. | Known as runtime exceptions. |

TELUSKO