EXCEPTION HANDLING

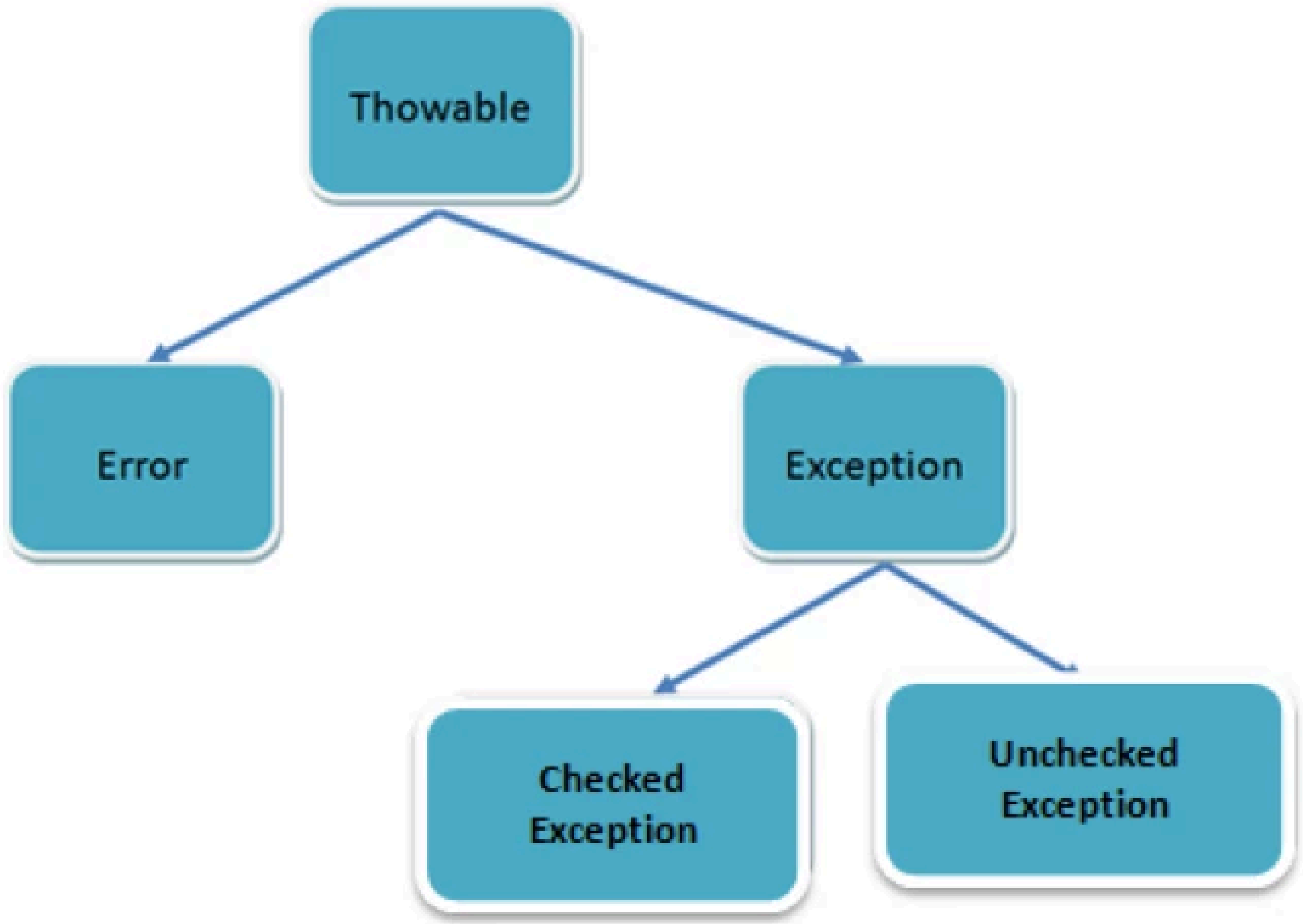
Basic Java Course

Advisor: Prof. Thap Thareoun

Present By : Ngouch HongCheng
: Saing LymChhun

INTRODUCTION

- **Exception Handling** in Java is one of the effective means to handle runtime errors so that the regular flow of the applicatio can be preserved.
- **Java Exception** Handling is a mechanisn to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.



1. THROWABLE

- In Java, **Throwable** is the superclass for all errors and exceptions that can be thrown and caught.
- It's at the root of the exception hierarchy, which means both Exception and Error are subclasses of Throwable.
- **Throwable** is used to handle both exceptions (which are recoverable issues) and errors (which are usually critical problems).

2. ERRORS

- In Java, **an Error** is a subclass of Throwable and represents serious issues that a reasonable application typically cannot handle.
- **Errors** usually indicate problems with the Java Virtual Machine (JVM) or other critical resources, which are not intended to be caught or managed within the application code.
- **Errors** are typically unrecoverable and indicate conditions where the program is unable to continue operating safely.

2. ERRORS

- <u>**Key Points about Errors**</u>
- **Unrecoverable:** Errors usually represent conditions that the application cannot recover from, such as resource exhaustion or system-level failures.
- **JVM-related Issues:** Many errors are related to JVM issues, such as memory problems, class loading issues, or problems in native code.
- **Should Not Be Caught:** Errors should generally not be caught or handled in application code. They often indicate critical conditions where continuing execution could lead to inconsistent program states or further errors.

3. EXCEPTION

- In Java, **an Exception** is an event that disrupts the normal flow of a program.
- **Exceptions** are situations that a program might want to handle to prevent crashing or incorrect behavior.
- They represent conditions that applications can anticipate and attempt to recover from, unlike Error types, which are usually unrecoverable.
- **Exceptions** often represent issues that can be anticipated and handled within the program, like a file not being found or a network connection failing.

3. EXCEPTION

Common Exception Types

- **IOException**: Thrown when an I/O operation fails, such as reading from or writing to a file that doesn't exist.
- **SQLException**: Thrown when there is an issue with accessing a database, like a bad query or connection failure.
- **ArithmeticException**: Thrown when there is an illegal arithmetic operation, such as dividing by zero.
- **NullPointerException**: Thrown when trying to access an object that has not been initialized.
- **ArrayIndexOutOfBoundsException**: Thrown when accessing an array with an invalid index.

4. CHECKED EXCEPTION

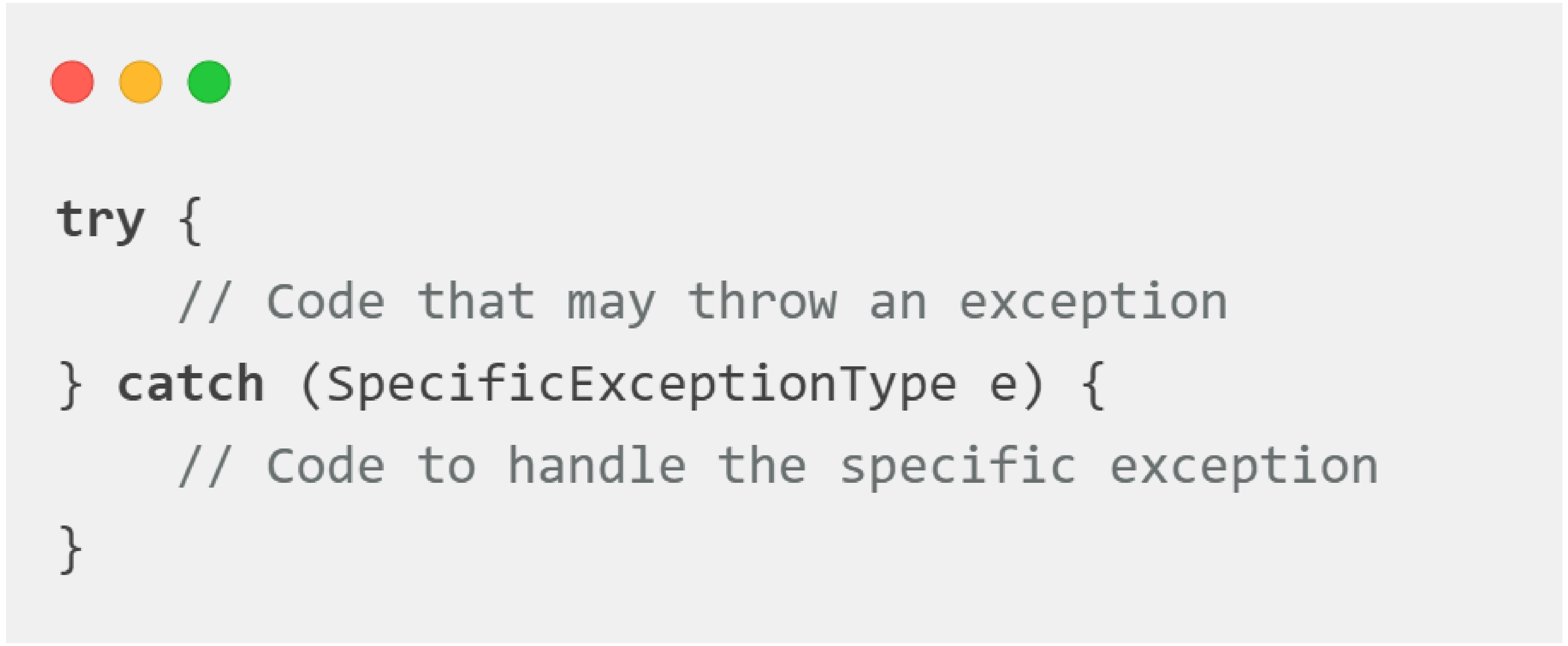
- A **checked exception** in Java is a type of exception that is checked at compile time.
- These exceptions must be either caught using a try-catch block or declared in the method signature using the throws keyword.
- The compiler checks for these exceptions during the compilation process, ensuring that the programmer handles them properly.
- This mechanism encourages robust error handling and makes the code more reliable.

4. CHECKED EXCEPTION

A. USING TRY-CATCH BLOCKS

- A try-catch block is the most common way to handle checked exceptions.
- The code that might throw an exception is placed within the try block, and if an exception occurs, it is caught by one or more catch blocks.
- This allows you to handle the exception directly where it occurs.

Syntax



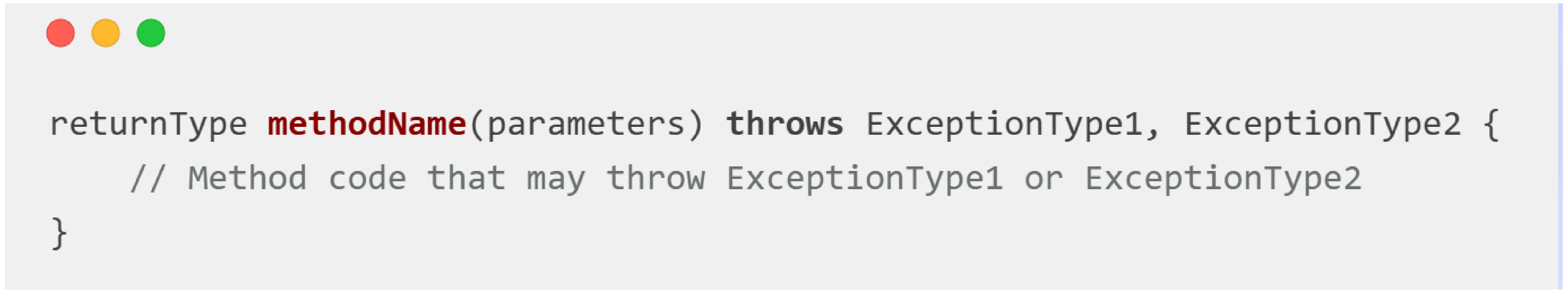
```
try {
    // Code that may throw an exception
} catch (SpecificExceptionType e) {
    // Code to handle the specific exception
}
```

4. CHECKED EXCEPTION

B. THROWS KEYWORD

- The throws keyword is used in a method declaration to indicate that the method might throw specific checked exceptions.
- This approach shifts the responsibility of handling the exception to the calling method, rather than catching it directly within the method.

Syntax



```
returnType methodName(parameters) throws ExceptionType1, ExceptionType2 {
    // Method code that may throw ExceptionType1 or ExceptionType2
}
```

4. CHECKED EXCEPTION

SUMMARY

- Use **try-catch** when you want to handle exceptions directly within a method, allowing for specific responses to each exception type.
- Use **throws** to delegate exception handling to the calling method, which is useful in layered applications where exceptions may be handled at a higher level.

4. UNCHECKED EXCEPTION

- Unchecked exceptions in Java are exceptions that do not require explicit handling in the code.
- They are subclasses of the RuntimeException class and are typically indicative of programming errors, such as logic mistakes or incorrect assumptions in the code.
- Since they are not checked at compile time, programmers are not forced to handle them using try-catch blocks or to declare them using the throws clause in method signatures.

END