**Exception Handling**

**─**

Contents

[**1. What are Exceptions** 2](#_Toc188248905)

[# Hierarchy of Java Exceptions 2](#_Toc188248906)

[**2. Throwing and Catching Exceptions** 4](#_Toc188248907)

[Module 3: Custom Exceptions 4](#_Toc188248908)

[Creating Custom Exceptions 4](#_Toc188248909)

[**4. Interview Questions** 6](#_Toc188248910)

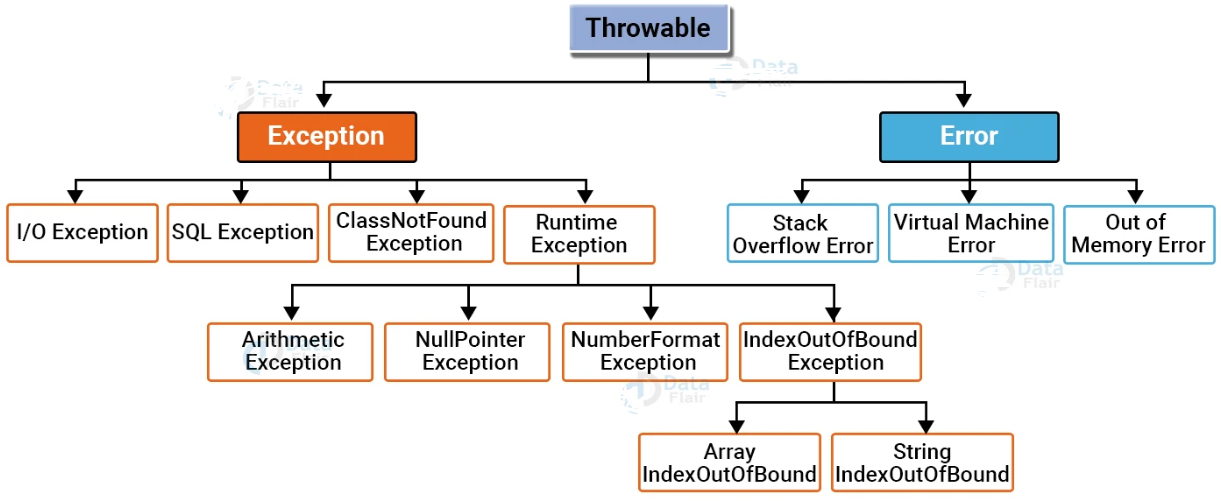
## **1. What are Exceptions**

An exception is an event that occurs during the execution of a program that disrupts its normal flow.

Key points:

* Exceptions provide a way to handle errors systematically.
* Without exceptions, the program would terminate abruptly when an error occurs.
* Using exceptions, you can gracefully recover from errors.

### # Hierarchy of Java Exceptions



**Compile time exception (checked exception):**

* Exceptions that are checked at compile time.
* The program will not compile unless they are handled or declared.
* Examples: IO Exception, SQLException

import java.io.File;  
import java.io.FileReader;  
  
public class CheckedExceptionExample {  
 public static void main(String[] args) {  
 try {  
 File file = new File("nonexistent.txt");  
 FileReader reader = new FileReader(file);  
 } catch (Exception e) {  
 System.*out*.println("File not found: " + e.getMessage());  
 }  
 }  
}

**Run time exception (unchecked exception):**

* Exceptions that occur during the execution of a program.
* They are not checked by the compiler.
* Examples: NullPointerException , ArithmeticException

public class UncheckedExceptionExample {  
 public static void main(String[] args) {  
 String str = null;  
 System.*out*.println(str.length()); // Causes NullPointerException  
 }  
}

**Error:**

* Represent serious problems that the application cannot recover from.
* Examples: OutOfMemoryError, StackOverflowError

public class StackOverflowErrorExample {  
 public static void recursiveMethod() {  
 *recursiveMethod*(); // Infinite recursion  
 }  
  
 public static void main(String[] args) {  
 *recursiveMethod*();  
 }  
}

## **2. Throwing and Catching Exceptions**

Throwing an Exception

The throw keyword is used to explicitly throw an exception.

//syntax  
throw new ExceptionType("Error Message");

­­­

### Module 3: Custom Exceptions

### Creating Custom Exceptions

Extend the Exception class.

class CustomException extends Exception {

public CustomException(String message) {

super(message);

}

}

public class CustomExceptionDemo {

public static void main(String[] args) {

try {

throw new CustomException("This is a custom exception");

} catch (CustomException e) {

System.out.println("Caught: " + e.getMessage());

}

}

}

## **4. Interview Questions**

**1. What are differences between Exceptions and Errors?**

| **Aspect** | **Exception** | **Error** |
| --- | --- | --- |
| **Definition** | An Exception is a condition that an application might want to catch and handle. | An Error represents a serious problem that an application cannot typically recover from. |
| **Hierarchy** | Subclass of Throwable. | Subclass of Throwable. |
| **Recoverability** | Exceptions are generally recoverable with proper handling. | Errors are usually not recoverable and indicate critical system issues. |
| **Occurrence** | Caused by application-related issues (e.g., invalid input, network failures). | Caused by system-related issues (e.g., memory leaks, hardware failures). |
| **Examples** | IOException, SQLException, NullPointerException, etc. | OutOfMemoryError, StackOverflowError, etc. |
| **Usage in Code** | Applications should try to handle exceptions using try-catch blocks. | Errors are not meant to be caught by applications, though it is technically possible. |
| **Checked vs. Unchecked** | Exceptions can be checked (IOException) or unchecked (ArithmeticException). | Errors are unchecked and not subject to compile-time checks. |
| **Typical Cause** | Result of flaws in the program logic or unexpected conditions. | Caused by issues beyond the application’s control, like resource depletion or JVM failures. |
| **Handling** | Handled using try-catch or declared using throws. | Handling errors is not recommended, as they indicate critical issues. |

**2. Is it possible to handle errors?**

While technically possible, catching Error is not advisable since it indicates critical failures that usually require fixing the underlying issue (e.g., optimizing memory usage in case of OutOfMemoryError).

Exception handling is used for expected issues like invalid user input or file not found. Error handling is avoided because errors like OutOfMemoryError require fixing the root cause rather than catching them in code.