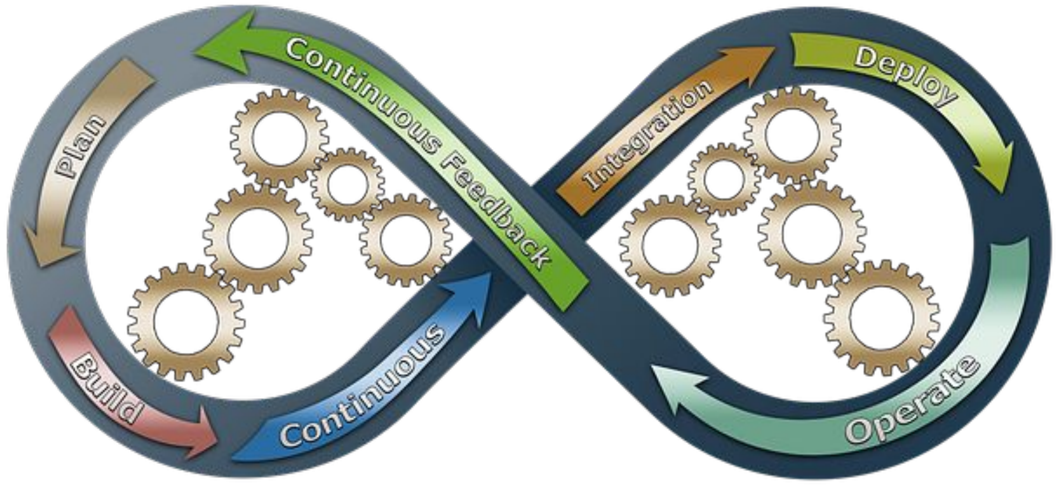


# Session 3.7

# Exceptions

AN INITIATIVE BY

UNICAL ACADEMY

# Introduction

UNICAL ACADEMY

Let's go!!!

## Exceptions

**Introduction:**

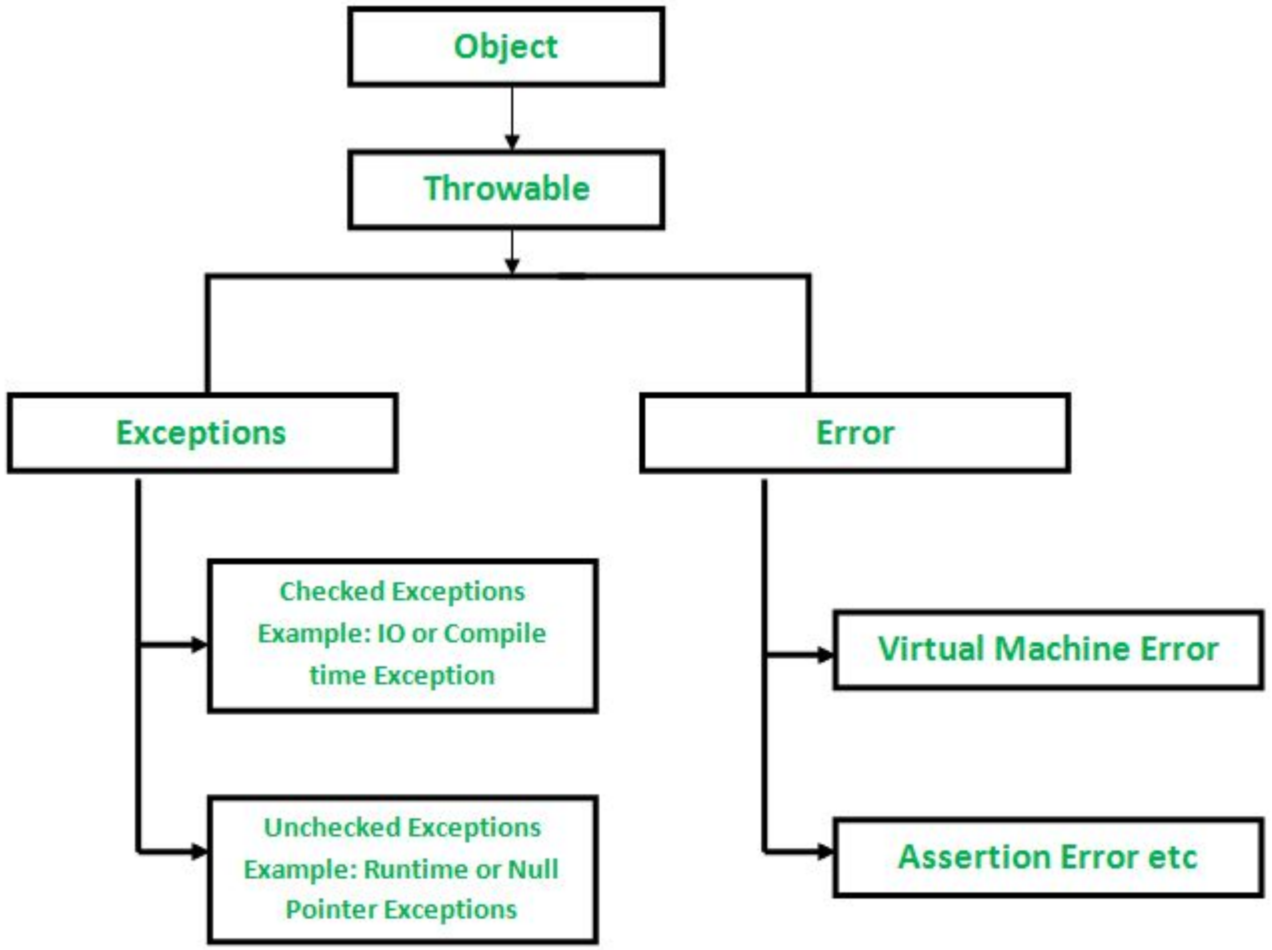
- An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally.

**Scenarios Where an Exception Occurs:**

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

# Exception Hierarchy

UNICAL ACADEMY


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

## Difference between Exception & Error

**Exception:**

- It can be classified into unchecked and checked exceptions.
- It belongs to the class 'java.lang.Exception'.
- It can be recovered from.
- It can occur at runtime as well as compile time.
- Examples of exceptions include –
  - NullPointerException
  - SqlException

**Error:**

- It is classified as an unchecked type.
- It belongs to the class 'java.lang.error'.
- It can't be recovered from.
- It can't occur at compile time.
- Examples of errors include –
  - 'OutOfMemoryError'
  - 'IOError'

# Exception handling using : try-catch-finally

**Try:**

- The try statement allows you to define a block of code to be tested for errors while it is being executed.

**Catch:**

- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

**Flow control in try-catch-finally**

- try-catch clause (or) try-catch-finally clause
  - **Case 1:** Exception occurs in try block and handled in catch block
  - **Case 2:** Exception occurs in try-block is not handled in catch block
  - **Case 3:** Exception doesn't occur in try-block
- try-finally clause
  - **Case 1:** Exception occurs in try block
  - **Case 2:** Exception doesn't occur in try-block

**The try and catch Syntax:**

```
try {
 // Block of code to try
 }
catch(Exception e) {
// Block of code to handle errors
}
```

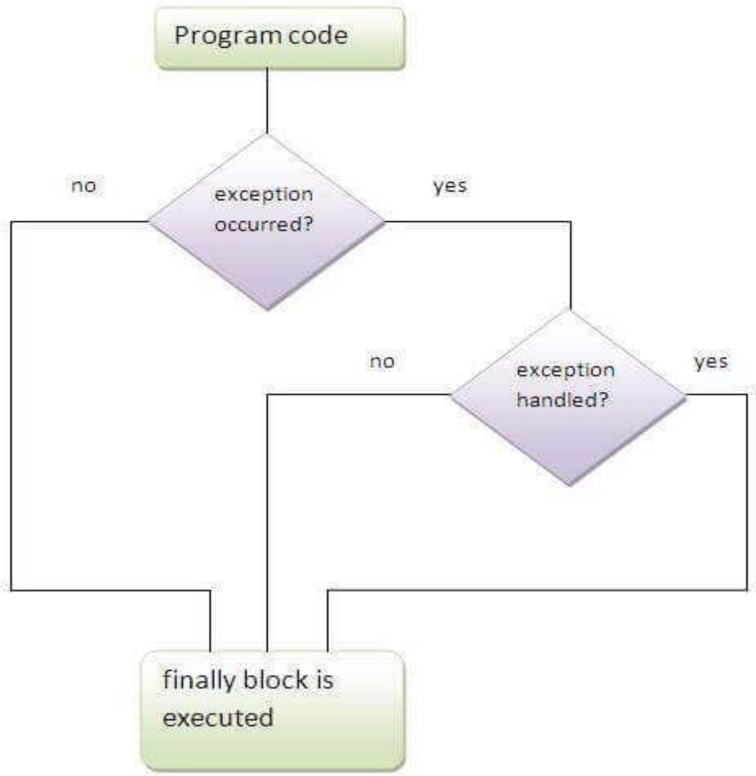
## Finally

:

- The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.
- finally block is used to execute important code such as closing connection, stream etc.
- finally block is always executed whether exception is handled or not.
- finally block follows try or catch block.

## Syntax:

```
try {
 // Protected code
}
catch (ExceptionType1 e1) {
 // Catch block
 }
catch (ExceptionType2 e2) {
 // Catch block
 }
catch (ExceptionType3 e3) {
 // Catch block
 }
finally {
// The finally block always executes.
 }
```



UNICAL ACADEMY

## Methods to display error information

| S. No. | Method & Description |
|---|---|
| 1 | **Public String getMessage()** Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor. |
| 2 | **Public Throwable getCause()** Returns the cause of the exception as represented by a Throwable object. |
| 3 | **Public String toString()** Returns the name of the class concatenated with the result of getMessage(). |
| 4 | **Public void printStackTrace()** Prints the result of toString() along with the stack trace to System.err, the error output stream. |
| 5 | **Public StackTraceElement [] getStackTrace()** Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack. |
| 6 | **Public Throwable fillInStackTrace()** Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace. |

UNICAL ACADEMY

# Checked and Unchecked exceptions

**Checked Exception:**

- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

**Unchecked Exception:**

- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**Error:**

- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Unchecked Exception
Checked Exception
Error
Exception Handling

# User defined exceptions (Customized Exceptions)

**Common Exceptions:**

- JVM Exceptions − These are exceptions/errors that are exclusively or logically thrown by the JVM. Examples: NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException.
- Programmatic Exceptions − These exceptions are thrown explicitly by the application or the API programmers. Examples: IllegalArgumentException, IllegalStateException.

**User-defined Exceptions:**

- We can create our own exceptions in Java.
- Below points to be followed when writing own exception classes −
- All exceptions must be a child of Throwable.
- If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
- If you want to write a runtime exception, you need to extend the RuntimeException class.

**We can define our own Exception class as below:**

```
class MyException extends Exception {
 }
```

# Throw and Throws

UNICAL ACADEMY

| S. No. | Key | throw | throws |
|---|---|---|---|
| 1 | Definition | Throw is a keyword which is used to throw an exception explicitly in the program inside a function or inside a block of code. | Throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code. |
| 2 | Internal implementation | Internally throw is implemented as it is allowed to throw only single exception at a time i.e we cannot throw multiple exception with throw keyword. | On other hand we can declare multiple exceptions with throws keyword that could get thrown by the function where throws keyword is used. |
| 3 | Type of exception | With throw keyword we can propagate only unchecked exception i.e checked exception cannot be propagated using throw. | with throws keyword both checked and unchecked exceptions can be declared and for the propagation checked exception must use throws keyword followed by specific exception class name. |
| 4 | Syntax | Syntax wise throw keyword is followed by the instance variable. | syntax wise throws keyword is followed by exception class names. |
| 5 | Declaration | In order to use throw keyword we should know that throw keyword is used within the method. | throws keyword is used with the method signature. |

UNICAL ACADEMY

## Session Recap