**National University of Computer & Emerging Sciences, Karachi**



**Computer Science Department Fall 2023, Lab Manual – 10**

|  |  |
| --- | --- |
| **Course Code: CL-1004** | **Course : Object Oriented Programming Lab** |
| **Instructor(s) :** | **Shafique Rehman** |

LAB - 10

Exception Handling

**Exception Handling**

The Exception Handling in Java is o ne of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

# statement 1;

**statement 2;**

# statement 3;

**statement 4;**

# statement 5;//exception occurs statement 6;

**statement 7;**

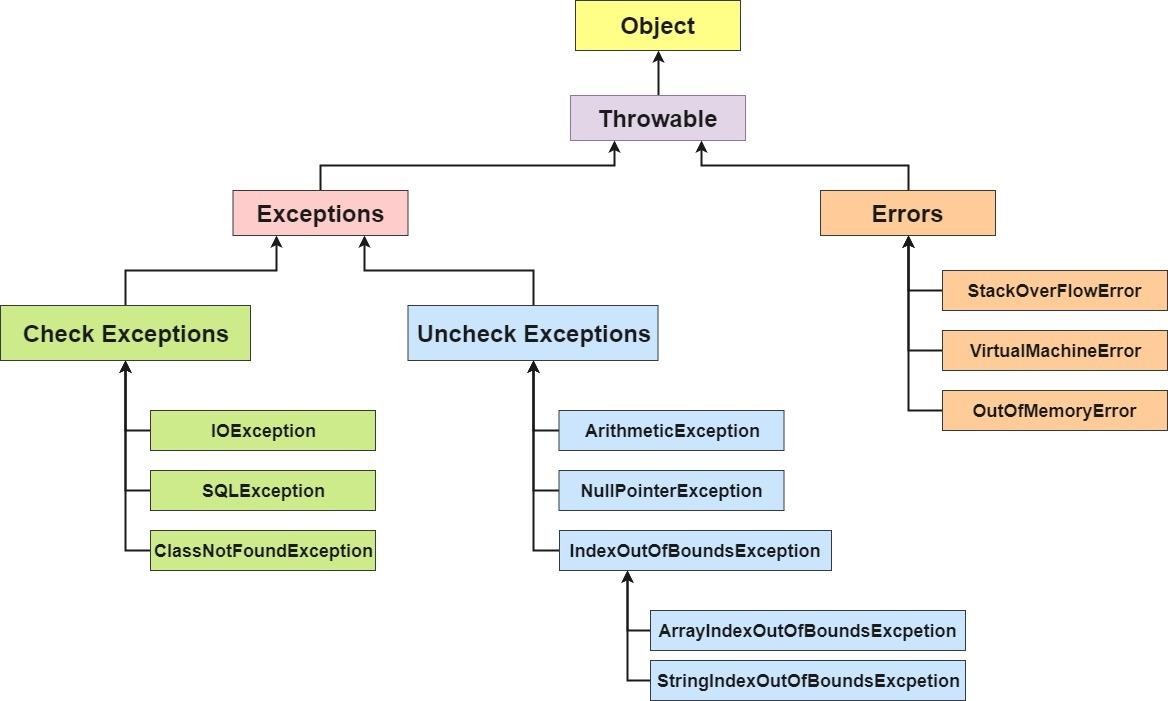
# statement 8;

**statement 9;**

# statement 10;

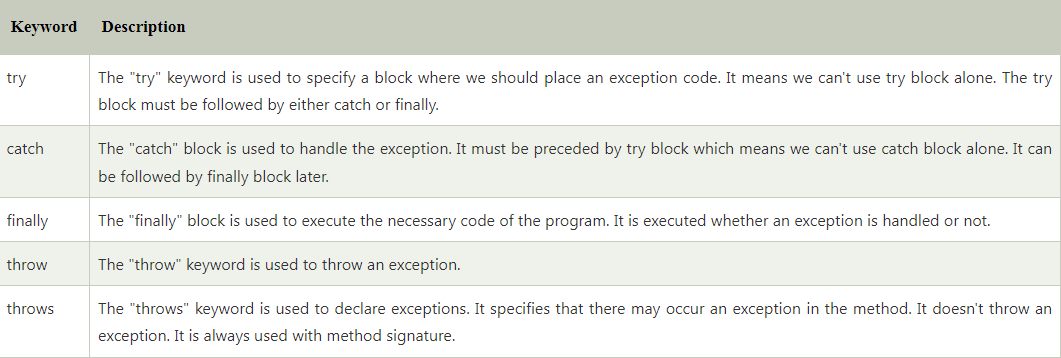
Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.

# Hierarchy of Java Exception classes

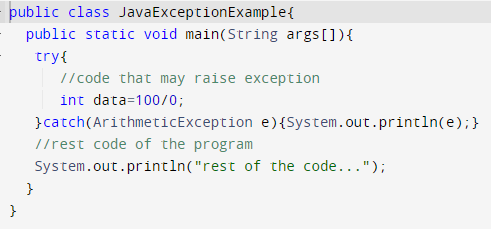


**Java Exception Keywords**

Java provides five keywords that are used to handle the exception. The following table describes each.



# Simple Try Catch Example:



**Output:**



# Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1. A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

**int a=50/0;//ArithmeticException**

1. A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

**String s=null; System.out.println(s.length());//NullPointerException**

1. A scenario where NumberFormatException occurs

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

**String s="abc";**

**int i=Integer.parseInt(s);//NumberFormatException**

1. A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

**int a[]=new int[5];**

**a[10]=50; //ArrayIndexOutOfBoundsException**

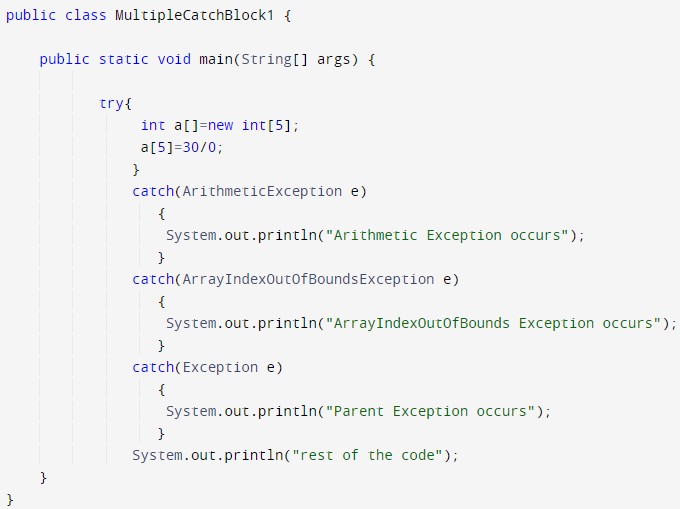
# Multi - Catch:

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

# Points to remember:

* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception or a compile – time error occurs.

# Multi - Catch Example:



**Output:**



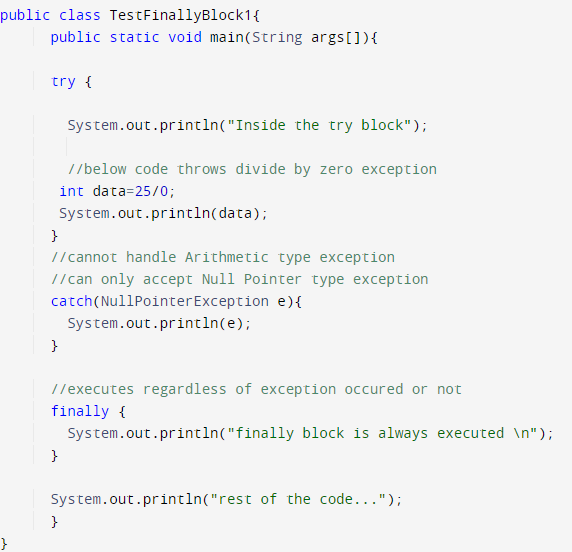
# Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

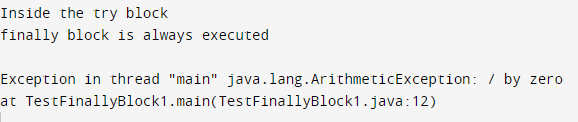
Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

The finally block follows the try-catch block.

**Finally Example:** the code throws an exception however the catch block cannot handle it. Despite this, the finally block is executed after the try block and then the program terminates abnormally.



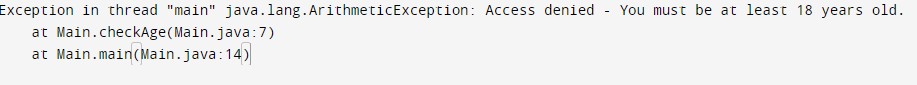
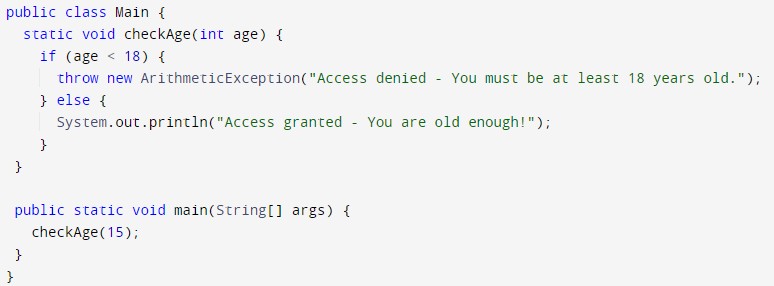
# Output:



**Throw Keyword:**

The throw keyword is used to create a custom error.

The throw statement is used together with an exception type. There are many exception types available in Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.



**Java throws keyword**

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception.

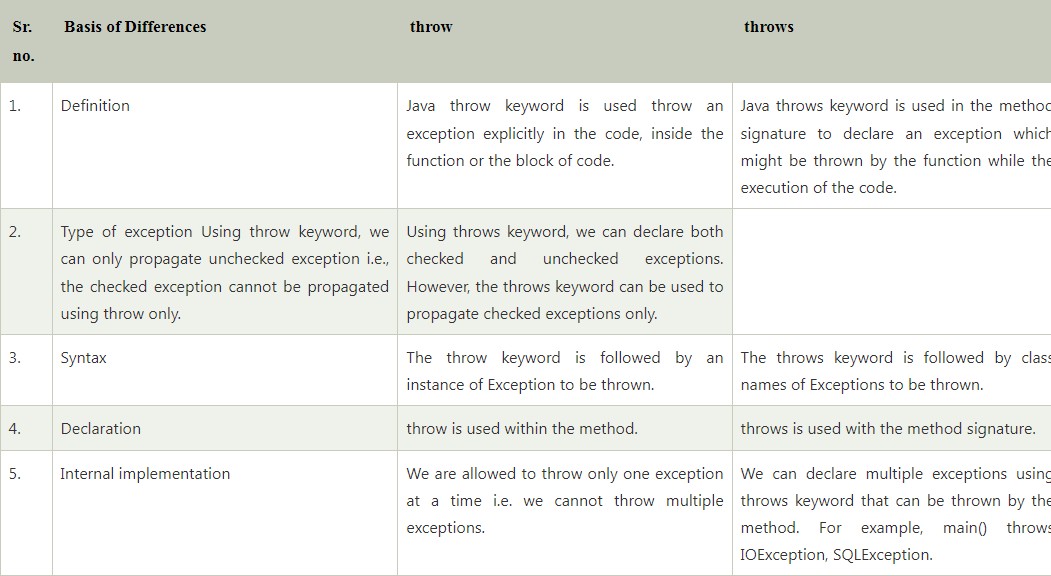
**Syntax of Java throws**

**return\_type method\_name() throws exception\_class\_name{**

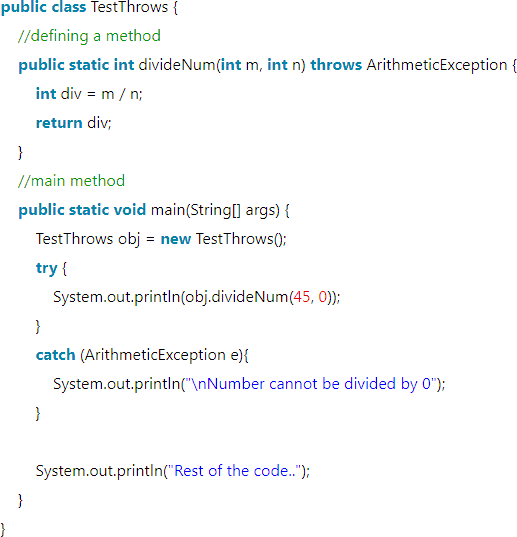
**//method code**

**}**

**The throws keyword is used in a method signature and declares which exceptions can be thrown from a method. The throws keyword can be useful for propagating exceptions in the call stack and allows exceptions to not necessarily be handled within the method that declares these exceptions. Only checked exceptions are required to be thrown using the throws keyword. Unchecked exceptions don’t need to be thrown or handled explicitly in code.**



**Java throws Example**



* 1. You are building a program that calculates the average of a list of integers. Write a Java program that handles the following exceptions:
     + If the list is null or empty, the program should display an appropriate error message.
     + If any of the integers in the list are not valid integers (i.e., they contain non- numeric characters), the program should skip that integer and display only numeric data’s average.
     + If the list contains only invalid integers (non – numeric), the program should display an appropriate message.

Your program should use try-catch blocks to handle the exceptions. You can assume that the list is provided as an argument.

* 1. Calculate the factorial of a given integer n. The program should take input from the user in the form of a string and convert it to an integer (Integer.parseInt()). Write a Java program that handles the following exceptions:
     + If the input string is null or empty, the program should display an appropriate error message.
     + If the input string does not represent a valid integer (i.e., it contains non-numeric characters), the program should display an appropriate error message.
     + If the input integer is negative, the program should display an appropriate error message.
     + If the input integer is too large to calculate the factorial (i.e., it causes an integer overflow), the program should display an appropriate error message.

Your program should use try-catch blocks to handle the exceptions. If the input integer is valid and can be used to calculate the factorial, the program should calculate the factorial and display it to the user.

* 1. Ask the user to enter two numbers and perform a mathematical operation on them. The program should handle the following exceptions:
     + If the input for either number is null or empty, the program should display an appropriate error message.
     + If the input for either number does not represent a valid integer or double (i.e., it contains non-numeric characters), the program should display an appropriate error message.
     + If the user attempts to divide by zero, the program should display an appropriate error message.

Your program should use try-catch blocks to handle the exceptions. If the input is valid, the program should perform the mathematical operation and display the result to the user.

* 1. Prompt the user to enter a sentence and then perform a series of operations on the sentence. The program should handle the following exceptions:
     + If the input string is null or empty, the program should display an appropriate error message.
     + If the input string contains less than two words, the program should display an appropriate error message.

Your program should use try-catch blocks to handle the exceptions. If the input and operation are valid, display the modified sentence to the user.