

Exceptions in Java

What is an Exception?

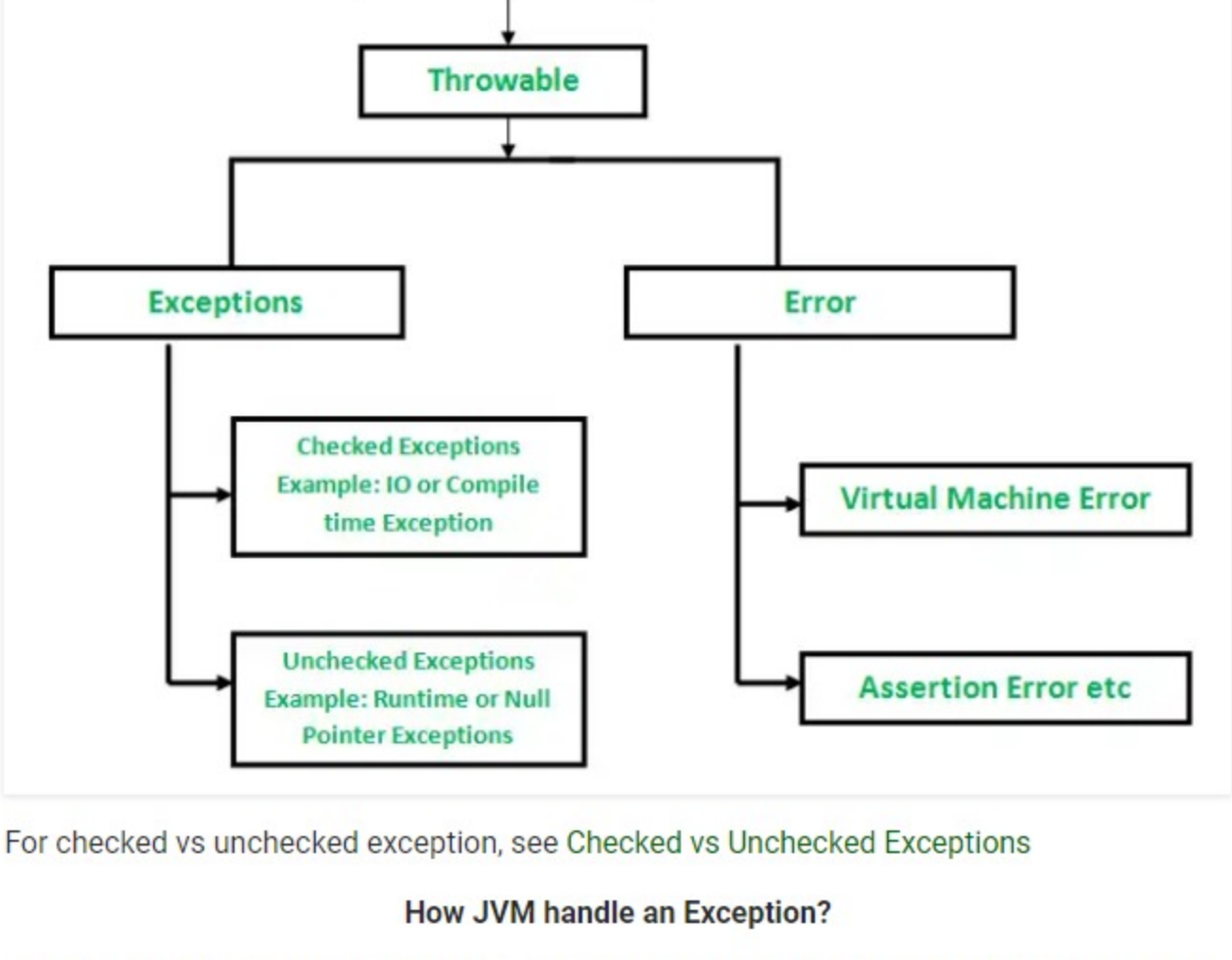
An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

Error vs Exception

Error: An Error indicates serious problem that a reasonable application should not try to catch.
Exception: Exception indicates conditions that a reasonable application might try to catch.

Exception Hierarchy

All exception and errors types are sub classes of class **Throwable**, which is base class of hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. **NullPointerException** is an example of such an errors having to do with the run-time environment itself (JRE). **StackOverflowError** is an example of such an error.



For checked vs unchecked exception, see [Checked vs Unchecked Exceptions](#)

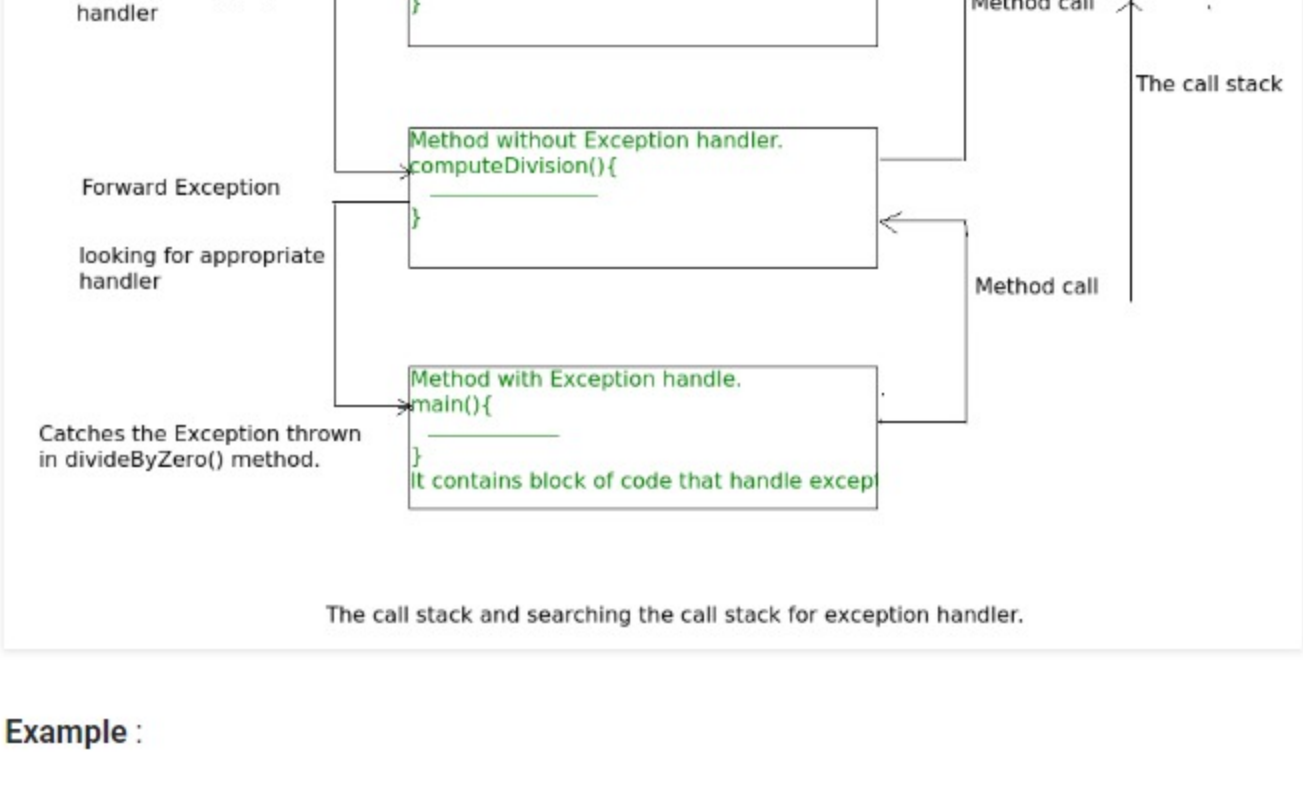
How JVM handle an Exception?

Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an **Object** known as **Exception Object** and hands it off to the run-time system (JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the **Exception Object** and handling it to the run-time system is called **throwing an Exception**. There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called **Call Stack**. Now the following procedure will happen.

- The run-time system searches the call stack to find the method that contains **block of code** that can handle the occurred exception. The block of the code is called **Exception handler**.
- The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
- If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
- If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the **Exception Object** to **default exception handler**, which is part of run-time system. This handler prints the exception information in the following format and terminates program **abnormally**.

```
Exception in thread "xxx" Name of Exception : Description
... .. // Call Stack
```

See the below diagram to understand the flow of the call stack.



Example :

```
// Java program to demonstrate how exception is thrown.
class ThrowsExcep{

    public static void main(String args[]){

        String str = null;
        System.out.println(str.length());

    }

}
```

Output :

```
Exception in thread "main" java.lang.NullPointerException
at ThrowsExcep.main(File.java:8)
```

Let us see an example that illustrate how run-time system searches appropriate exception handling code on the call stack :

```
// Java program to demonstrate exception is thrown
// how the runTime system searches th call stack
// to find appropriate exception handler.
class ExceptionThrown
{
    // It throws the Exception(ArithmeticException).
    // Appropriate Exception handler is not found within this method.
    static int divideByZero(int a, int b){

        // this statement will cause ArithmeticException(/ by zero)
        int i = a/b;

        return i;
    }

    // The runTime System searches the appropriate Exception handler
    // in this method also but couldn't have found. So looking forward
    // on the call stack.
    static int computeDivision(int a, int b) {

        int res =0;

        try
        {
            res = divideByZero(a,b);
        }
        // doesn't matches with ArithmeticException
        catch(NumberFormatException ex)
        {
            System.out.println("NumberFormatException is occurred");
        }
        return res;
    }

    // In this method found appropriate Exception handler.
    // i.e. matching catch block.
    public static void main(String args[]){

        int a = 1;
        int b = 0;

        try
        {
            int i = computeDivision(a,b);
        }

        // matching ArithmeticException
        catch(ArithmeticException ex)
        {
            // getMessage will print description of exception(here / by ze
            System.out.println(ex.getMessage());
        }
    }
}
```

Output :

```
/ by zero.
```

How Programmer handles an exception?

Customized Exception Handling : Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**. Briefly, here is how they work. Program statements that you think can raise exceptions are contained within a **try** block. If an exception occurs within the **try** block, it is thrown. Your code can catch this exception (using **catch** block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword **throw**. Any exception that is thrown out of a method must be specified as such by a **throws** clause. Any code that absolutely must be executed after a **try** block completes is put in a **finally** block.

Detailed Article: [Control flow in try catch finally block](#)

Need of try-catch clause(Customized Exception Handling)

Consider the following java program.

```
// java program to demonstrate
// need of try-catch clause

class GFG {
    public static void main (String[] args) {

        // array of size 4.
        int[] arr = new int[4];

        // this statement causes an exception
        int i = arr[4];

        // the following statement will never execute
        System.out.println("Hi, I want to execute");

    }
}
```

Output :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
at GFG.main(GFG.java:9)
```

Explanation : In the above example an array is defined with size i.e. you can access elements only from index 0 to 3. But you trying to access the elements at index 4 (by mistake) that's why it is throwing an exception. In this case, JVM terminates the program **abnormally**. The statement `System.out.println("Hi, I want to execute");` will never execute. To execute it, we must handle the exception using **try-catch**. Hence to continue normal flow of the program, we need **try-catch** clause.

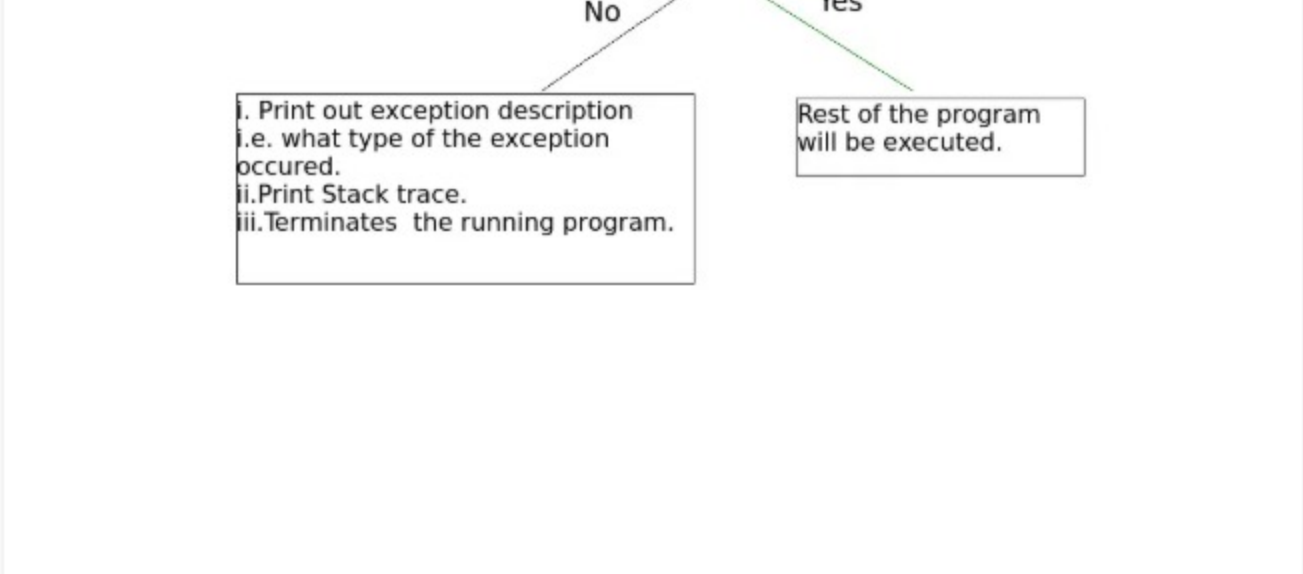
How to use try-catch clause

```
try {
    // block of code to monitor for errors
    // the code you think can raise an exception
}
catch (ExceptionType1 exObj) {
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exObj) {
    // exception handler for ExceptionType2
}
// optional
finally {
    // block of code to be executed after try block ends
}
```

Points to remember :

- In a method, there can be more than one statements that might throw exception. So put all these statements within its own **try** block and provide separate exception handler within own **catch** block for each of them.
- If an exception occurs within the **try** block, that exception is handled by the exception handler associated with it. To associate exception handler, we must put **catch** block after it. There can be more than one exception handlers. Each **catch** block is a exception handler that handles the exception of the type indicated by its argument. The argument, **ExceptionType** declares the type of the exception that it can handle and must be the name of the class that inherits from **Throwable** class.
- For each **try** block there can be zero or more catch blocks, but **only one** finally block.
- The finally block is optional. It always gets executed whether an exception occurred in **try** block or not . If exception occurs, then it will be executed after **try** and **catch** blocks. And if exception does not occur then it will be executed after the **try** block. The finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection.

Summary



Related Articles:

- [Types of Exceptions in Java](#)
- [Checked vs Unchecked Exceptions](#)
- [Throw- Throws in Java](#)

Reference :

<https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html>

This article is contributed by **Nitsdheerendra** and **Gaurav Miglani**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute@geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

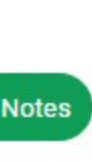
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

- [Chained Exceptions in Java](#)
- [Errors V/s Exceptions In Java](#)
- [Checked vs Unchecked Exceptions in Java](#)
- [Built-in Exceptions in Java with examples](#)
- [Using throw, catch and Instanceof to handle Exceptions in Java](#)
- [User-defined Exceptions in Python with Examples](#)
- [Java.util.LinkedList.poll\(\), pollFirst\(\), pollLast\(\) with examples in Java](#)
- [Java.util.LinkedList.peek\(\), peekFirst\(\), peekLast\(\) in Java](#)
- [Java lang.Long.reverse\(\) method in Java with Examples](#)
- [Java lang.Long.lowestOneBit\(\) method in Java with Examples](#)
- [Java.util.Collections.rotate\(\) Method in Java with Examples](#)
- [Java.util.LinkedList.offer\(\), offerFirst\(\), offerLast\(\) in Java](#)
- [Java lang.Long.numberOfLeadingZeros\(\) method in Java with Examples](#)
- [Java lang.Long.numberOfTrailingZeros\(\) method in Java with Examples](#)
- [Java lang.Long.highestOneBit\(\) method in Java with Examples](#)

Article Tags : [Java](#) [School Programming](#) [Java-Exceptions](#)

Practice Tags : [Java](#)



3

1.8

☐ To-do ☐ Done

Based on 20 vote(s)

[Feedback](#) [Add Notes](#) [Improve Article](#)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

[Creating a file using FileOutputStream](#)

Next

[java.lang.Boolean class methods](#)

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Share this post!](#)

Most popular in Java
Max Heap in Java
Traverse through a HashMap in Java
AbstractList in Java with Examples
Find the Index of an array element in Java
ArrayList forEach() method in Java

Most visited in School Programming
Print matrix in snake pattern from the last
Print all prime numbers less than or equal to N
Check if the given chessboard is valid or not
Interchange elements of first and last columns in matrix
Sum of array Elements without using loops and recursion

Most visited in Java
HashMap keySet() Method in Java
Implementing a Linked List in Java using Class
Print all nodes between two given levels in Binary Tree
ArrayList subList() method in Java with Examples
Errors V/s Exceptions In Java
PriorityQueue comparator() Method in Java
Check if a string contains only alphabets in Java using Regex
Difference between throw and throws in Java
How to Remove Duplicates from ArrayList in Java
Check if a value is present in an Array in Java
Writing a CSV file in Java using OpenCSV
ArrayList of ArrayList in Java
ArrayList remove() method in Java
Check if a string contains only alphabets in Java using ASCII values
ArrayDeque in Java