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, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some 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.

**Types of Java Exceptions**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

**Difference between Checked and Unchecked Exceptions**

**1) 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.

**2) 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.

**3) Error**

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

**Java Exception Keywords**

There are 5 keywords which are used in handling exceptions in Java.

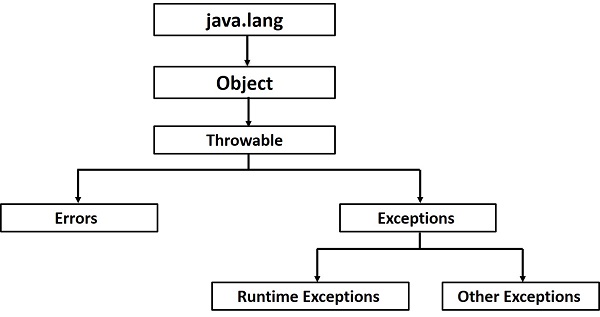
|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

**Exception Hierarchy**

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and RuntimeException Class.



Java defines several exception classes inside the standard package **java.lang**.

The most general of these exceptions are subclasses of the standard type RuntimeException. Since java.lang is implicitly imported into all Java programs, most exceptions derived from RuntimeException are automatically available.

Java defines several other types of exceptions that relate to its various class libraries. Following is the list of Java Unchecked RuntimeException.

|  |  |
| --- | --- |
| **Sr.No.** | **Exception & Description** |
| 1 | **ArithmeticException**  Arithmetic error, such as divide-by-zero. |
| 2 | **ArrayIndexOutOfBoundsException**  Array index is out-of-bounds. |
| 3 | **ArrayStoreException**  Assignment to an array element of an incompatible type. |
| 4 | **ClassCastException**  Invalid cast. |
| 5 | **IllegalArgumentException**  Illegal argument used to invoke a method. |
| 6 | **IllegalMonitorStateException**  Illegal monitor operation, such as waiting on an unlocked thread. |
| 7 | **IllegalStateException**  Environment or application is in incorrect state. |
| 8 | **IllegalThreadStateException**  Requested operation not compatible with the current thread state. |
| 9 | **IndexOutOfBoundsException**  Some type of index is out-of-bounds. |
| 10 | **NegativeArraySizeException**  Array created with a negative size. |
| 11 | **NullPointerException**  Invalid use of a null reference. |
| 12 | **NumberFormatException**  Invalid conversion of a string to a numeric format. |
| 13 | **SecurityException**  Attempt to violate security. |
| 14 | **StringIndexOutOfBounds**  Attempt to index outside the bounds of a string. |
| 15 | **UnsupportedOperationException**  An unsupported operation was encountered. |

**Exceptions Methods**

Following is the list of important methods available in the Throwable class.

|  |  |
| --- | --- |
| **Sr.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. |

**Catching Exceptions**

A method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following −

**Syntax**

try {

// Protected code

} catch (ExceptionName e1) {

// Catch block

}

// File Name : ExcepTest.java

import java.io.\*;

public class ExcepTest {

public static void main(String args[]) {

try {

int a[] = new int[2];

System.out.println("Access element three :" + a[3]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Exception thrown :" + e);

}

System.out.println("Out of the block");

}

}

**Output**

Exception thrown :java.lang.ArrayIndexOutOfBoundsException: 3

Out of the block

**Multiple Catch Blocks**

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following −

**Syntax**

try {

// Protected code

} catch (ExceptionType1 e1) {

// Catch block

} catch (ExceptionType2 e2) {

// Catch block

} catch (ExceptionType3 e3) {

// Catch block

}

**The Throws/Throw Keywords**

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

**The Finally Block**

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax −

**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.

}