**Exception Handling** in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**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. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

**Major reasons why an exception Occurs**

* Invalid user input
* Device failure
* Loss of network connection
* Physical limitations (out of disk memory)
* Code errors
* Opening an unavailable file

**Errors** represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

Let us discuss the most important part which is the **differences between Error and Exception** that is as follows:

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

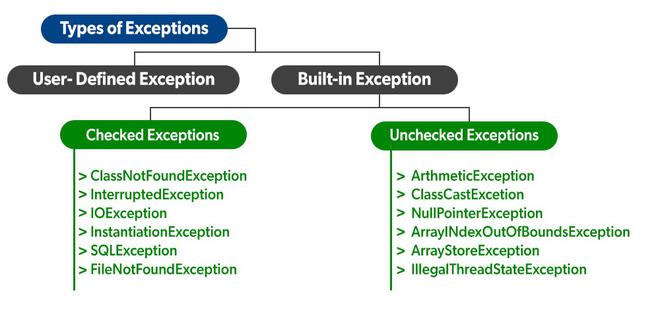
**Exception Hierarchy**

All exception and error types are subclasses of class **Throwable**, which is the base class of the 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 exception. Another branch, **Error** is used by the Java run-time system([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.



**Types of Exceptions**

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



**Exceptions can be categorized in two ways:**

1. **Built-in Exceptions**
   * Checked Exception
   * Unchecked Exception
2. **User-Defined Exceptions**

Let us discuss the above-defined listed exception that is as follows:

**A. Built-in Exceptions:**

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

* **Checked Exceptions:**Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
* **Unchecked Exceptions:**The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn’t handle or declare it, the program would not give a compilation error.

The ***advantages of Exception Handling in Java***are as follows:

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

## **Java Exception Keywords**

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

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

# **Java try-catch block**

## **Java try block**

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

## **Problem without exception handling**

Let's try to understand the problem if we don't use a try-catch block.

### Example 1

**TryCatchExample1.java**

1. **public** **class** TryCatchExample1 {
3. **public** **static** **void** main(String[] args) {
5. **int** data=50/0; //may throw exception
7. System.out.println("rest of the code");
9. }
11. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TryCatchExample1)

**Output:**

Exception in thread "main" java.lang.ArithmeticException: / by zero

As displayed in the above example, the **rest of the code** is not executed (in such case, the **rest of the code** statement is not printed).

There might be 100 lines of code after the exception. If the exception is not handled, all the code below the exception won't be executed.

# **Java Catch Multiple Exceptions**

## **Java Multi-catch block**

**MultipleCatchBlock1.java**

1. **public** **class** MultipleCatchBlock1 {
3. **public** **static** **void** main(String[] args) {
5. **try**{
6. **int** a[]=**new** **int**[5];
7. a[5]=30/0;
8. }
9. **catch**(ArithmeticException e)
10. {
11. System.out.println("Arithmetic Exception occurs");
12. }
13. **catch**(ArrayIndexOutOfBoundsException e)
14. {
15. System.out.println("ArrayIndexOutOfBounds Exception occurs");
16. }
17. **catch**(Exception e)
18. {
19. System.out.println("Parent Exception occurs");
20. }
21. System.out.println("rest of the code");
22. }
23. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=MultipleCatchBlock1)

**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.

### Flowchart of finally block



Case 1: When an exception does not occur

Let's see the below example where the Java program does not throw any exception, and the finally block is executed after the try block.

**TestFinallyBlock.java**

1. **class** TestFinallyBlock {
2. **public** **static** **void** main(String args[]){
3. **try**{
4. //below code do not throw any exception
5. **int** data=25/5;
6. System.out.println(data);
7. }
8. //catch won't be executed
9. **catch**(NullPointerException e){
10. System.out.println(e);
11. }
12. //executed regardless of exception occurred or not
13. **finally** {
14. System.out.println("finally block is always executed");
15. }
17. System.out.println("rest of phe code...");
18. }
19. }

**Output:**



# **Java throw Exception**

In Java, exceptions allows us to write good quality codes where the errors are checked at the compile time instead of runtime and we can create custom exceptions making the code recovery and debugging easier.

## **Java throw keyword**

The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. We will discuss custom exceptions later in this section.

The syntax of the Java throw keyword is given below.

throw Instance i.e.,

1. **throw** **new** exception\_class("error message");

Let's see the example of throw IOException.

1. **throw** **new** IOException("sorry device error");

Where the Instance must be of type Throwable or subclass of Throwable. For example, Exception is the sub class of Throwable and the user-defined exceptions usually extend the Exception class.

## **Java throw keyword Example**

1. **public** **class** TestThrow1 {
2. //function to check if person is eligible to vote or not
3. **public** **static** **void** validate(**int** age) {
4. **if**(age<18) {
5. //throw Arithmetic exception if not eligible to vote
6. **throw** **new** ArithmeticException("Person is not eligible to vote");
7. }
8. **else** {
9. System.out.println("Person is eligible to vote!!");
10. }
11. }
12. //main method
13. **public** **static** **void** main(String args[]){
14. //calling the function
15. validate(13);
16. System.out.println("rest of the code...");
17. }
18. }

# **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. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.

### Syntax of Java throws

1. return\_type method\_name() **throws** exception\_class\_name{
2. //method code
3. }

### Example

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

public class Main {

static void checkAge(int age) **throws** ArithmeticException {

if (age < 18) {

throw new ArithmeticException("Access denied - You must be at least 18 years old.");

}

else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(15); // Set age to 15 (which is below 18...)

}

}

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_throws)

## **Definition and Usage**

The throws keyword indicates what exception type may be thrown by a method.

There are many exception types available in Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.

Differences between [throw](https://www.w3schools.com/java/ref_keyword_throw.asp) and throws:

|  |  |
| --- | --- |
| **throw** | **throws** |
| Used to throw an exception for a method | Used to indicate what exception type may be thrown by a method |
| Cannot throw multiple exceptions | Can declare multiple exceptions |
| **Syntax:**   * throw is followed by an object (new type) * used inside the method | **Syntax:**   * throws is followed by a class * and used with the method signature |

# **Difference between throw and throws in Java**

The throw and throws is the concept of exception handling where the throw keyword throw the exception explicitly from a method or a block of code whereas the throws keyword is used in signature of the method.

There are many differences between [throw](https://www.javatpoint.com/throw-keyword)

and [throws](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)

keywords. A list of differences between throw and throws are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. no.** | **Basis of Differences** | **throw** | **throws** |
| 1. | Definition | Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code. | Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code. |
| 2. | Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only. | Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only. |  |
| 3. | Syntax | The throw keyword is followed by an instance of Exception to be thrown. | The throws keyword is followed by class names of Exceptions to be thrown. |
| 4. | Declaration | throw is used within the method. | throws is used with the method signature. |
| 5. | Internal implementation | We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions. | We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException. |

# **Thread**

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously. A+b,a-b,a/b

In order to perform complicated tasks in the background, we used the **Thread concept in Java**. All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.

Another benefit of using **thread** is that if a thread gets an exception or an error at the time of its execution, it doesn't affect the execution of the other threads. All the threads share a common memory and have their own stack, local variables and program counter. When multiple threads are executed in parallel at the same time, this process is known as [**Multithreading**](https://www.javatpoint.com/multithreading-in-java).

Thread.sleep(5000); ms 5 sec

### ****How to Create Threads using Java Programming Language?****

We can create Threads in java using two ways, namely :

1. By extending Thread Class
2. By Implementing a Runnable interface

## **Running Threads**

If the class extends the Thread class, the thread can be run by creating an instance of the class and call its start() method:

### Extend Example

public class Main extends Thread {

public static void main(String[] args) {

Main thread = new Main();

thread.start();

System.out.println("This code is outside of the thread");

}

public void run() {

System.out.println("This code is running in a thread");

}

}

If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method:

### Implement Example

public class Main implements Runnable {

public static void main(String[] args) {

Main obj = new Main();

Thread thread = new Thread(obj);

thread.start();

System.out.println("This code is outside of the thread");

}

public void run() {

System.out.println("This code is running in a thread");

}

}

New ---

Active—Start method runnable,running

Waiting--

Timed wait--

Terminated----