**Exception Handling**

Exception handling in Java is a mechanism to handle runtime errors, ensuring the normal flow of the application. Java uses try-catch-finally blocks for exception handling, along with throw and throws keywords.

## 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.** |
| **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.** |

**Exception**: An event that disrupts the normal flow of the program.

* **Checked Exception**: Checked at compile time (e.g., IOException, SQLException).
* **Unchecked Exception**: Checked at runtime (e.g., NullPointerException, ArrayIndexOutOfBoundsException).

**Throwable Class**: The superclass for all errors and exceptions. It has two main subclasses:

* Error: Represents serious issues (e.g., OutOfMemoryError), which a program shouldn't handle.
* Exception: Represents recoverable errors in the program

try {

// Code that may throw an exception

} catch (ExceptionType1 e1) {

// Handle exception of type ExceptionType1

} catch (ExceptionType2 e2) {

// Handle exception of type ExceptionType2

} finally {

// Code that will always execute, regardless of an exception

}

public class ExceptionHandlingExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw ArithmeticException

} catch (ArithmeticException e) {

System.out.println("ArithmeticException caught: " + e);

} finally {

System.out.println("This block is always executed.");

}

}

}

**Important Keywords:**

1. **try**: Defines a block of code where an exception may occur.
2. **catch**: Catches and handles exceptions.
3. **finally**: A block that always executes after the try block, whether an exception occurs or not. It is typically used for cleanup code (e.g., closing files).
4. **throw**: Used to explicitly throw an exception.
   * Example: throw new ArithmeticException("Division by zero");
5. **throws**: Declares an exception that a method might throw.
   * Example: public void method() throws IOException { ... }

Multi-Catch

In Java, you can use multiple catch blocks to handle different types of exceptions that may arise within the same try block. Each catch block handles a specific type of exception. When an exception is thrown, Java looks for the first matching catch block and executes it.

Syntax for Multiple Catches:

try {

// Code that may throw different types of exceptions

} catch (ExceptionType1 e1) {

// Handle ExceptionType1

} catch (ExceptionType2 e2) {

// Handle ExceptionType2

} catch (ExceptionType3 e3) {

// Handle ExceptionType3

} finally {

// Optional block for cleanup

}

public class MultipleCatchesExample {

public static void main(String[] args) {

try {

int[] array = new int[5];

array[5] = 10; // This will throw ArrayIndexOutOfBoundsException

int result = 10 / 0; // This will throw ArithmeticException

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index is out of bounds: " + e);

} catch (ArithmeticException e) {

System.out.println("Division by zero: " + e);

} finally {

System.out.println("This block is always executed.");

}

}

}

Array index is out of bounds: java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5

This block is always executed.

In the example above:

* If ArrayIndexOutOfBoundsException is thrown, the first catch block is executed.
* If ArithmeticException is thrown, the second catch block is executed.
* The finally block is always executed, regardless of whether an exception occurred.

**Handling multiple exceptions in one catch block**: Since Java 7, you can handle multiple exceptions in a single catch block using the pipe | symbol.

try {

// Code that may throw exceptions

} catch (IOException | SQLException e) {

System.out.println("Caught IOException or SQLException: " + e.getMessage());

}

The throw keyword in Java is used to explicitly throw an exception. Unlike Java's automatic exception handling, where the system throws an exception when it encounters errors like division by zero or invalid array access, the throw statement allows the developer to intentionally trigger an exception when certain conditions are met in the code.

### Key Points about throw:

1. **Explicit Exception Throwing**: It allows you to manually raise an exception in the program. This is useful when a method detects an error condition that needs to be handled by the calling code.
2. **Exception Object**: The throw keyword requires an instance of the Throwable class or one of its subclasses (like Exception or Error) to be thrown. This is typically created using the new keyword.

throw new ExceptionType("Error message");

public class ThrowExample {

public static void validate(int age) {

if (age < 18) {

throw new IllegalArgumentException("Age must be 18 or above.");

}

System.out.println("Valid age.");

}

public static void main(String[] args) {

validate(15); // This will throw an IllegalArgumentException

}

}

The throws keyword in Java is used in a method signature to declare that the method might throw one or more exceptions. It informs the caller of the method that they must handle or propagate the exception(s). This is particularly important for **checked exceptions**, which the Java compiler forces you to handle explicitly.

### Key Points about throws:

1. **Method Declaration**: The throws keyword is used to specify exceptions that a method might throw during its execution. These are exceptions that the method does not handle directly (i.e., does not catch inside the method) but instead passes on to its caller.
2. **Checked Exceptions**: The throws clause is typically used for **checked exceptions**, which must be either caught with a try-catch block or declared in the method signature using throws. If a checked exception is not handled or declared, it results in a compile-time error.
3. **Unchecked Exceptions**: Although it's not required, you can also declare **unchecked exceptions** (like RuntimeException or its subclasses) in the throws clause. These don't need to be handled or declared by the caller.
4. **Multiple Exceptions**: A method can declare multiple exceptions in the throws clause by separating them with commas.

## Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.