**Exception Handling in Java**

## **What is Exception Handling in Java?**

Exception handling in java helps in minimizing exceptions and helps in recovering from exceptions. It is one of the powerful mechanisms to handle runtime exceptions and makes it bug-free. Exception handling helps in maintaining the flow of the program. An exception handling is defined as an abnormal condition that may happen at runtime and disturb the normal flow of the program

## **What is an Exception?**

An expectation is an unexpected event that occurs while executing the program, that disturbs the normal flow of the code.

For handling exceptions, there are**2 possible approaches**

### **1. JVM**

If an exception is not handled explicitly, then JVM takes the responsibility of handling the exception.

Once the exception is handled, JVM will halt the program and no more execution of code will take place.



### **2**. **Developer**

Developers can explicitly write the implementation for handling the exception. Once an exception is handled, the normal execution of code will continue.

**Preferable**: handle exceptions to ensure your code gets executed normally

## **Java Exception Hierarchy**

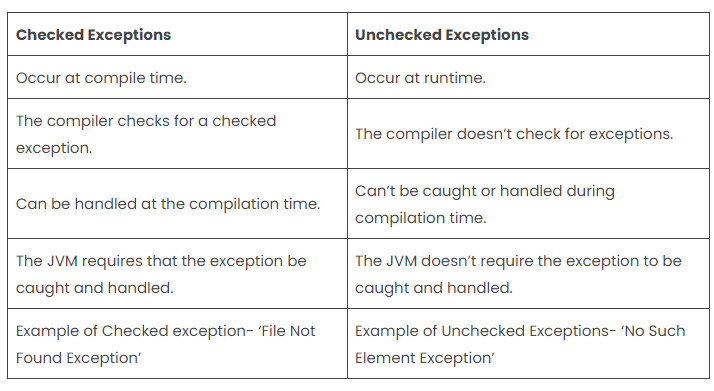
**Exception Hierarchy –**Following is the Exception Handling in Java handling hierarchy.

* **Throwable**–
  + It is the root class for the exception hierarchy in java.
  + It is in the java.lang package.
* **Error** –
  + Subclass of Throwable.
  + Consist of abnormal condition that is out of one’s control and depends on the environment
  + They can’t be handled and will always result in the halting of the program.
  + Eg: StackOverFlowError that can happen in infinite loop or recursion
* **Exception**–
  + Subclass of Throwable.
  + Consist of abnormal conditions that can be handled explicitly.
  + If one handles the exception then our code will continue to execute smoothly.

## **Types of exception in Java**

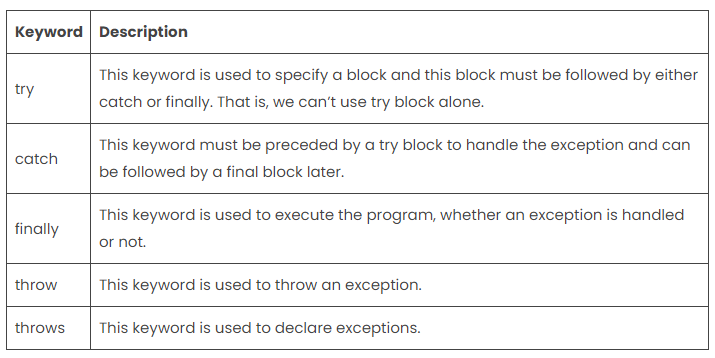
* **Checked Exceptions**
  + Those exceptions that are checked at compile-time comprises checked exceptions.
  + They are child classes of Exception except for RuntimeException.
  + The program will not compile if they are not handled.
  + Example: IOException, ClassNotFoundException, etc.
* **Unchecked Exceptions**
  + Those exceptions that are checked at runtime comprises unchecked exceptions.
  + They are child classes of RuntimeException.
  + They give runtime errors if not handled explicitly.
  + Example: ArithmeticException, NullPointerException etc.

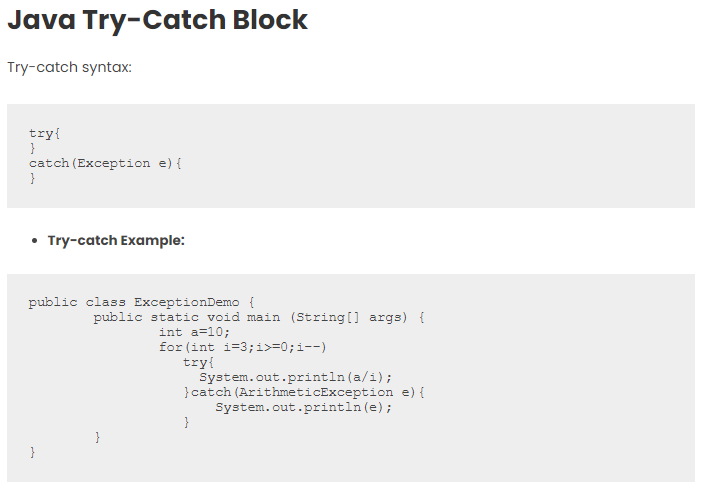
## **Difference between Checked and Unchecked Exception**

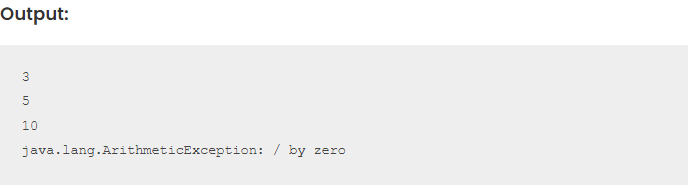


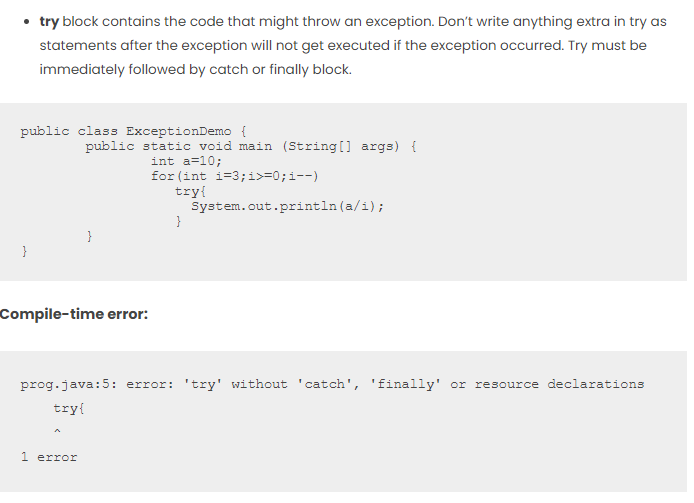
## **Java Exception Keywords**

Exception Handling in java is managed via five keywords: try, catch, throw, throws, and finally. Here are 5 keywords that are used in handling exceptions in Java



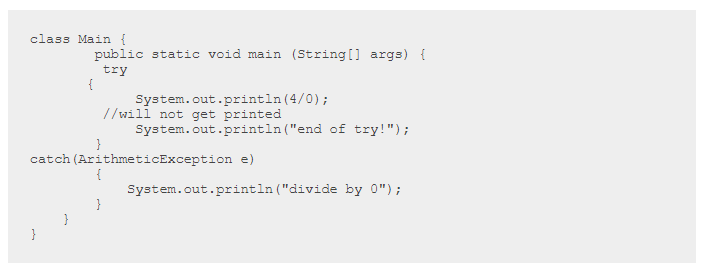


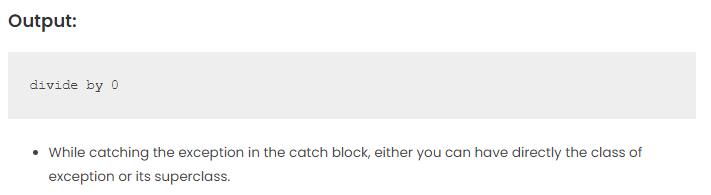




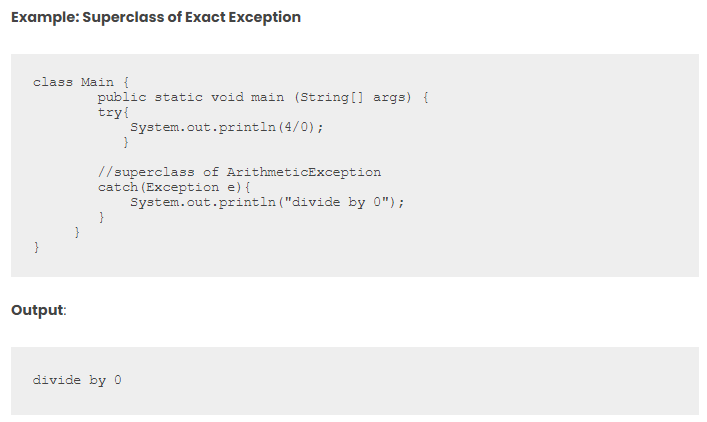
#### **Things to Remember:**

Do not keep any code after the statement which is prone to exception. Because if an exception occurred, it will straight away jump to the catch or finally block, ignoring all other statements in the try block.





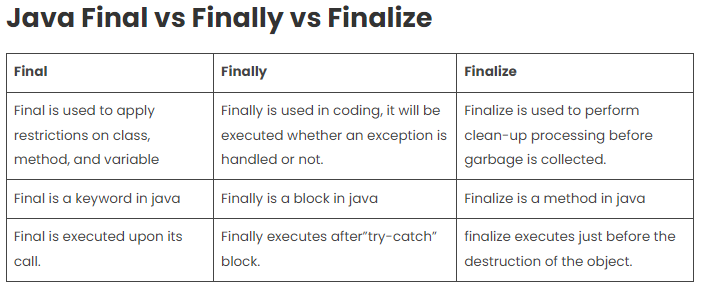




## **Java Finally Block**

Contains code that must be executed no matter if an exception is thrown or not. It contains code of file release, closing connections, etc.

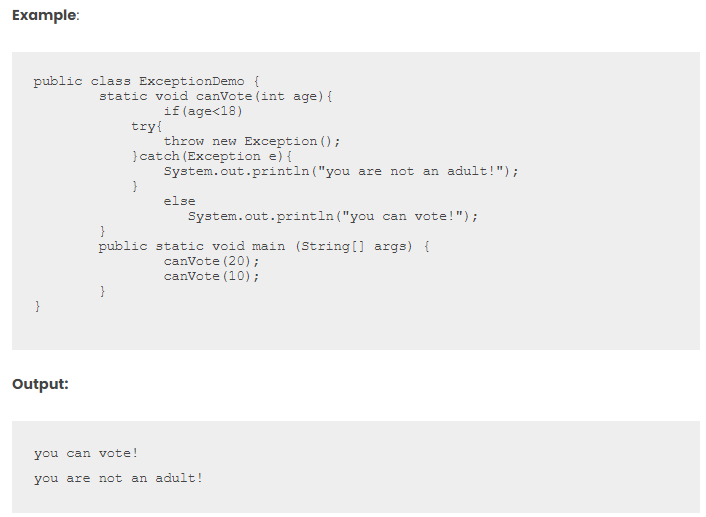
Finally, will execute even when we do not handle exceptions. Before halting the program, JVM checks if there is a “finally” block



## **Java Throw Keyword**

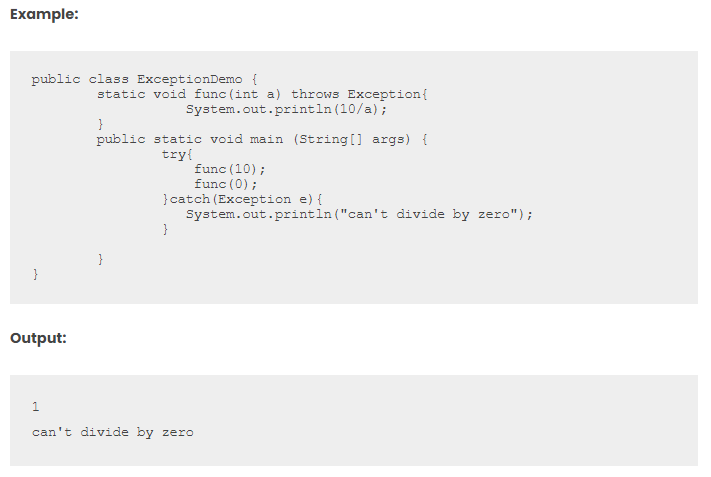
It is a keyword that is used to explicitly throw an exception.

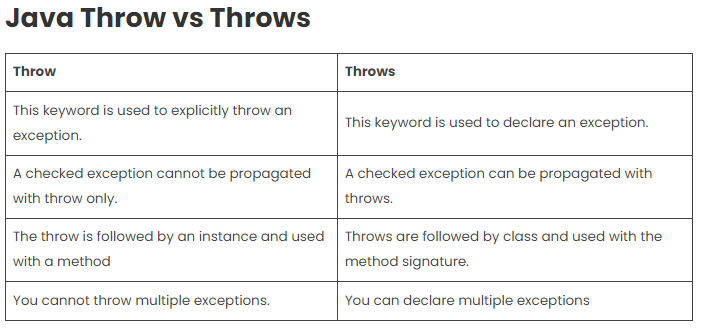
We can use throw where according to our logic an exception should occur.



## **Java Throws Keyword**

* Throws keyword is used when callee doesn’t want to handle the exception rather it wants to extend this responsibility of handling the exception to the caller of the function.
* Basically says what sort of exception the code can throw and relies on the caller to handle it.
* It is used to handle checked Exceptions as the compiler will not allow code to compile until they are handled.





## **Java Custom Exception**

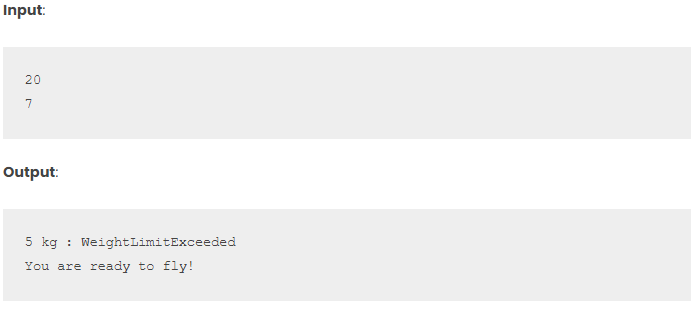
You can create your own exception and give implementation as to how it should behave. Your exception will behave like a child’s class of Exception.

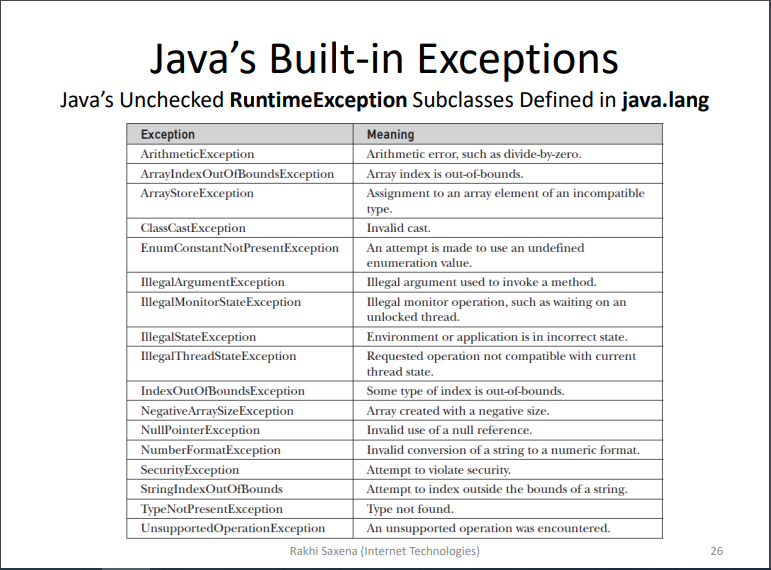
#### **Syntax**:

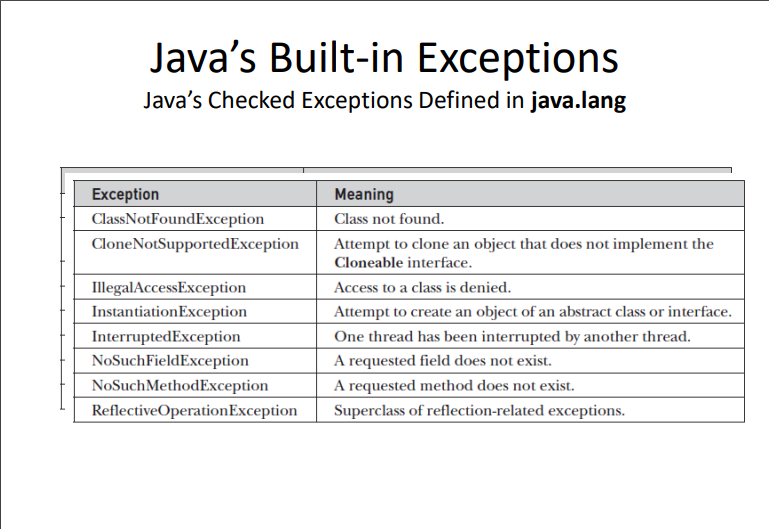
|  |  |
| --- | --- |
| 1 | class YourException extends Exception{} |

* **Example:**
  + let’s say, you are working with an airline company
  + You are in the luggage check-in department and as per rules, you can allow 15kg per customer.
  + So now more than 15kg of weight is an abnormal condition for us or in other words its an exception
  + This is our logic-based exception, so we’ll create our custom exception WeightLimitExceeded
  + As per syntax, it will extend Exception.
  + We define the constructor which will get invoked as soon as an exception will be thrown
  + We have to explicitly throw the exception and hence we will use throw keyword for that.
  + Using throws keyword is as per our need. If we are handling an exception where it is getting thrown then we can avoid throws, else we will use throws and handle it in the caller.









[**https://www.mygreatlearning.com/blog/exception-handling-in-java/**](https://www.mygreatlearning.com/blog/exception-handling-in-java/)

[**https://www.interviewkickstart.com/interview-questions/java-exception-handling-interview-questions**](https://www.interviewkickstart.com/interview-questions/java-exception-handling-interview-questions)