# [Exception Handling In Java](https://javaconceptoftheday.com/exception-handling-java/)

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons. Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Exception Handling in java is implemented using five keywords.

1) try       2)catch       3)finally       4)throw         5)throws

****try block :****In try block, keep those statements which may throw exceptions during run time.

****catch block :**** This block handles the exceptions thrown by try block. It takes one argument of type java.lang.Exception.

****finally block :**** Whether exception is thrown or not and thrown exception is caught or not, this block will be always executed.

**public** **class** ExceptionHandling

{

**public** **static** **void** main(String[] args)

    {

        String[] s = {"abc", "123", "xyz", "456"};   //String Array containing valid and invalid numeric values

**for** (**int** i = 0; i < s.length; i++)

        {

**try**

            {

**int** intValue = Integer.parseInt(s[i]); //This statement may throw NumberFormatException

            }

**catch**(NumberFormatException ex)

            {

                System.out.println("The thrown NumberFormatException will be caught here");

            }

**finally**

            {

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

            }

        }

    }

}

* When a statement throws an exception in the try block, the remaining part of the try block will not be executed. Program control comes out of the try block and enters directly into catch block.
* try, catch and finally blocks form one unit. i.e You can’t keep other statements in between try, catch and finally blocks.
* You can display the description of an exception thrown using Exception object in the catch block.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **public** **class** ExceptionHandling  {  **public** **static** **void** main(String[] args)      {  **try**          {  **int** i = 10/0;           //This statement throws ArithmeticException                System.out.println("This statement will not be executed");          }  //You can't keep statements here  **catch**(Exception ex)          {              System.out.println("This block is executed immediately after an exception is thrown");  ex.printStackTrace();     //This prints stack trace of exception          }  //You can't keep statements here  **finally**          {              System.out.println("This block is always executed");          }      }  } |
|  |  |

**Multiple Catch Blocks In Java:**

**public** **class** ExceptionHandling

{

**public** **static** **void** main(String[] args)

    {

        String[] s = {"abc", "123", **null**, "xyz"};   //String array containing one null object

**for** (**int** i = 0; i < 6; i++)

        {

**try**

            {

**int** a = s[i].length() + Integer.parseInt(s[i]);

                //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException

            }

**catch**(NumberFormatException ex)

            {

                System.out.println("NumberFormatException will be caught here");

            }

**catch** (ArrayIndexOutOfBoundsException ex)

            {

                System.out.println("ArrayIndexOutOfBoundsException will be caught here");

            }

**catch** (NullPointerException ex)

            {

                System.out.println("NullPointerException will be caught here");

            }

            System.out.println("After executing respective catch block, this statement will be executed");

        }

    }

}

* From Java 7 onward, there is one more way for handling multiple exceptions. Multiple exceptions thrown by the try block can be handled by a single catch block using ****pipe (|) operator****. By using pipe operator, the above example can be written as,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **public** **class** ExceptionHandling  {  **public** **static** **void** main(String[] args)      {          String[] s = {"abc", "123", **null**, "xyz"};   //String array containing one null object    **for** (**int** i = 0; i < 6; i++)          {  **try**              {  **int** a = s[i].length() + Integer.parseInt(s[i]);                    //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException              }    **catch**(NumberFormatException | NullPointerException | ArrayIndexOutOfBoundsException ex)              {                  System.out.println("Now, this block handles NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException");              }          }      }  } |

* java.lang.Exception is super class of all types of exception. (Types of exceptions will be discussed later). It handles all types of exceptions. In the above example, all catch blocks can be replaced by one catch block which handles all types of exceptions. This type of exception handling comes very handy when you are not sure about the types of exceptions your code may throw.

**try**

            {

**int** a = s[i].length() + Integer.parseInt(s[i]);

                //This statement may throw NumberFormatException, NullPointerException and ArrayIndexOutOfBoundsException

            }

**catch**(Exception ex)

            {

                System.out.println("This block handles all types of exceptions");

            }

* The order of catch blocks should be from most specific to most general ones. i.e Sub classes of Exception must come first and super classes later. If you keep the super classes first and sub classes later, you will get compile time error : ****Unreachable Catch Block****.

# [Nested try catch Blocks In Java](https://javaconceptoftheday.com/nested-try-catch-blocks-java/)

try-catch blocks can be nested. i.e one try block can contain another try-catch block. The syntax for nesting try blocks is,

**try** {    //Outer try block

    //Some Statement

**try** {   //Inner try block

        //Some Statements

    }**catch** (Exception ex)    //Inner catch block

    {

    }

}

**catch**(Exception ex)     //Outer catch block

{

}

* If the exception thrown by the inner try block can not be caught by it’s catch block, then this exception is propagated to outer try blocks. Any one of the outer catch block should handle this exception otherwise program will terminate abruptly.

public class ExceptionHandling

{

public static void main(String[] args)

{

String[] s = {"abc", "123", null, "xyz"}; //String array containing one null object

for (int i = 0; i < s.length; i++)

{

//First Level try-catch block

try

{

int a = s[i].length(); //This statement may throw NullPointerException

//second level try-catch block

try

{

System.out.println(s[i+1]); //This statement may throw ArrayIndexOutOfBoundsException

//third level try-catch block

try

{

a = Integer.parseInt(s[i]); //This statement may throw NumberFormatException

}

catch (NullPointerException e)

{

System.out.println("NumberFormatException will not be caught here");

}

}

catch (NumberFormatException ex)

{

System.out.println("NumberFormatException will be caught here");

}

}

catch(Exception ex)

{

System.out.println("This block catches all types of exceptions");

}

}

}

}

**public** **class** ExceptionPropogation {

/\*

\* In the following example also, try-catch blocks are nested. main() method

\* calls nestedTry() method. nestedTry() method has one try-catch block. First

\* statement in try block throws NumberFormatException which is not handled by

\* it’s catch block. So, It propagates to try-catch block of main method which

\* handles this exception.

\*/

**public** **static** **void** main(String[] args) {

**try** {

*nestedTry*();

} **catch** (Exception ex) {

System.***out***.println("NumberFormatException will be caught here in main method");

}

}

**static** **void** nestedTry() {

**try** {

**int** i = Integer.*parseInt*("abc"); // This statement throws NumberFormatException

} **catch** (NullPointerException ex) {

System.***out***.println("NumberFormatException will not be caught here");

}

}

}

* finally block overrides any return values from try and catch blocks.

**public** **static** **void** main(String[] args)

    {

        System.out.println(methodReturningValue());    //Output : 50

    }

**static** **int** methodReturningValue()

    {

**try**

        {

**return** 10;

        }

**catch** (Exception e)

        {

**return** 20;

        }

**finally**

        {

**return** 50;    //This method returns 50 not 10 or 20

        }

    }

# [Hierarchy Of Exceptions In Java](https://javaconceptoftheday.com/hierarchy-exceptions-java/)

To understand how exception handling works in Java, you need to understand the categories of exceptions:

## Checked Exceptions

Checked exceptions are known to compiler i.e they are the exceptions that are checked at compile time. Checked exceptions are also called compile time exceptions, because they can be known during compile time.

## Unchecked Exceptions

Unchecked exceptions are not known to compiler.  They are the exceptions that are not checked at compile time, because they occur only at run time.That’s why these exceptions are also called run time exceptions.

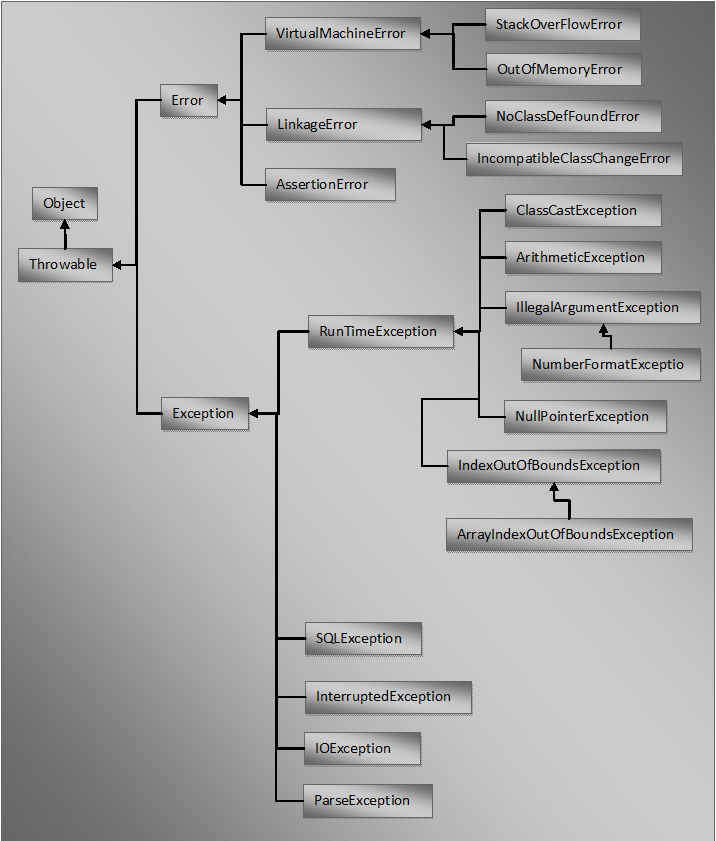
****Errors****:  
These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

## java.lang.Throwable :

java.lang.Throwable is the super class of all errors and exceptions in java. Throwable class extends java.lang.Object class. The only argument of catch block must be it’s type or it’s sub class type. You can check the documentation of Throwable class [here](http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html" \t "https://javaconceptoftheday.com/hierarchy-exceptions-java/_blank). It has two sub classes.

1)java.lang.Error

2)java.lang.Exception



## Throwing An Exception :

We all know that Throwable class is super class for all types of errors and exceptions. An object to this Throwable class or it’s sub classes can be created in two ways.

* First one is using an argument of catch block. In this way, Throwable object or object to it’s sub classes is implicitly created and thrown by java run time system.
* Second one is using new operator. In this way, Throwable object or object to it’s sub classes is explicitly created and thrown by the code.

****throw InstanceOfThrowableType;****

where, InstanceOfThrowableType must be an object of type Throwable or subclass of Throwable.

Such explicitly thrown exception must be handled some where in the program, otherwise program will be terminated.

It is not compulsory that explicitly thrown exception must be handled by immediately following try-catch block. It can be handled by any one of it’s enclosing try-catch blocks

**public** **class** ExceptionHandling

{

**public** **static** **void** main(String[] args)

    {

**try**

        {

            methodWithThrow();

        }

**catch**(NumberFormatException ex)

        {

            System.out.println("NumberFormatException object thrown in methodWithThrow() method will be handled here");

        }

    }

**static** **void** methodWithThrow()

    {

**try**

        {

            NumberFormatException ex = **new** NumberFormatException("This is an object of NumberFormatException");

**throw** ex;        //throwing NumberFormatException object explicitly using throw keyword

        }

**catch**(ArithmeticException ex)

        {

            System.out.println("Explicitly thrown NumberFormatException object will not be caught here");

        }

    }

}

## Re-throwing An Exception :

We all know that exceptions occurred in the try block are caught in catch block. Thus caught exceptions can be re-thrown using ****throw**** keyword. Re-thrown exception must be handled some where in the program, otherwise program will terminate abruptly.

**static** **void** methodWithThrow()

    {

**try**

        {

            String s = **null**;

            System.out.println(s.length());   //This statement throws NullPointerException

        }

**catch**(NullPointerException ex)

        {

            System.out.println("NullPointerException is caught here");

**throw** ex;     //Re-throwing NullPointerException

        }

    }

# [throws Keyword In Java](https://javaconceptoftheday.com/throws-keyword-java/)

If a method is capable of throwing an exception that it could not handle, then it should specify that exception using throws keyword. It helps the callers of that method in handling that exception. The syntax for using throws keyword is,

|  |  |
| --- | --- |
| 1  2  3  4 | return\_type method\_name(parameter\_list) **throws** exception\_list  {       //some statements  } |

**static** **void** methodWithThrows() **throws** NumberFormatException, NullPointerException

{

**int** i = Integer.parseInt(“abc”);

        String s = **null**;

        System.out.println(s.length());   //This statement throws NullPointerException

    }

* The main use of throws keyword in java is that an exception can be propagated through method calls.
* Even constructor can use throws keyword.For this, object creation statement must be enclosed in try-catch blocks.

When a method is throwing unchecked type of exceptions, then you need not to mention it using throws keyword. But for a method throwing checked type of exceptions, you must declare it with throws keyword or enclose the statement which is throwing an exception in try-catch block.