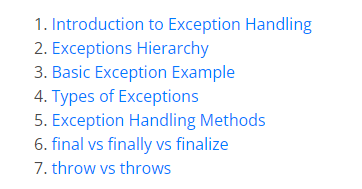
**What is meant by exception handling?**

Errors arise unexpectedly and can result in disrupting the normal flow of execution. This is something that every programmer faces at one point or the other while coding. [**Java,**](https://www.edureka.co/blog/java-tutorial/) being the most prominent [**object-oriented language,**](https://www.edureka.co/blog/object-oriented-programming/) provides a powerful mechanism to handle these errors/exceptions.

**What happens if exceptions are not handled?**

When an exception occurs, and if you don’t handle it, the program will terminate abruptly (the piece of code after the line causing the exception will not get executed).

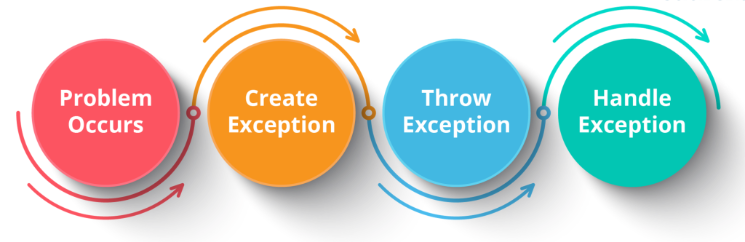


## ****Introduction to Exception Handling****

An exception is a problem that arises during the execution of a program. It can occur for various reasons say-

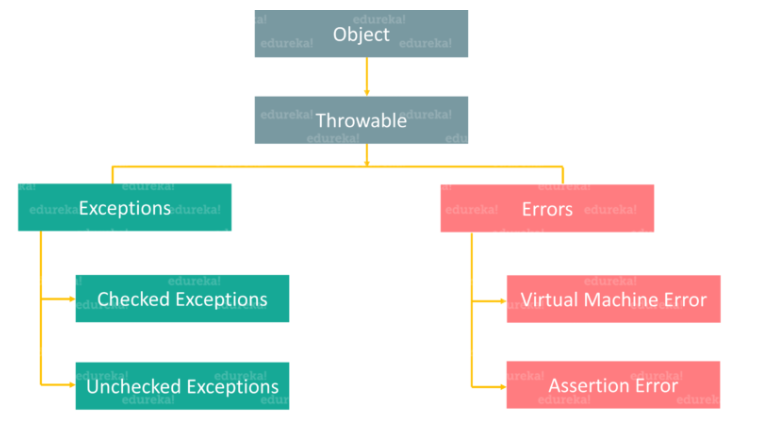
* A user has entered an invalid data
* File not found
* A network connection has been lost in the middle of communications
* The JVM has run out of a memory

Exception Handling mechanism follows a flow which is depicted in the below figure. But if an exception is not handled, it may lead to a system failure. That is why handling an exception is very important.



## ****Exceptions Hierarchy****

All exception and error types are subclasses of class ***Throwable***, which is the base class of hierarchy. One branch is headed by ***Error*** which occurs at run-time and other by **Exception**that can happen either at compile time or run-time.



Basically,  an **Error**is used by the Java run-time system (JVM) to indicate errors that are associated with the run-time environment (JRE). StackOverflowError is an example of such an error. Whereas **Exception**is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.

Now that you know what errors and exceptions are, let’s find out the basic difference between them. Take a look at the below table which draws a clear line between both of them.

|  |  |
| --- | --- |
| **Errors** | **Exceptions** |
| 1. Impossible to recover from an error | 1. Possible to recover from exceptions |
| 2. Errors are of type ‘unchecked’ | 2. Exceptions can be either ‘checked’ or ‘unchecked’ |
| 3. Occur at runtime | 3. Can occur at compile time or run time |
| 4. Caused by the application running environment | 4. Caused by the application itself |

Now, we will dive deeper into exceptions and see how they can be handled. First, let’s see the different types of exceptions.

* **Checked Exception**  
  It is an exception that occurs at compile time, also called compile time exceptions. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.
* **Unchecked Exception**  
  It is an exception that occurs at the time of execution. These are also called Runtime Exceptions. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to specify or catch the exceptions.

## ****Basic Example of Exception****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **class** Exception{  **public** **static** **void** main(String args[]){  **try**{  //code that may raise exception  }  **catch**(Exception e){  // rest of the program    }   }  } |

Above code represent an exception wherein inside try block we are going to write a code that may raise an exception and then, that exception will be handled in the catch block.

## ****Types of Exceptions****

### **Built-in Exceptions**

|  |  |
| --- | --- |
| **Built-in Exceptions** | **Description** |
| ***ArithmeticException*** | It is thrown when an exceptional condition has occurred in an arithmetic operation. |
| ***ArrayIndexOutOfBoundsException*** | It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array. |
| ***ClassNotFoundException*** | This exception is raised when we try to access a class whose definition is not found. |
| ***FileNotFoundException*** | An exception that is raised when a file is not accessible or does not open. |
| ***IOException*** | It is thrown when an input-output operation is failed or interrupted. |
| ***InterruptedException*** | It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted. |
| ***NoSuchFieldException*** | It is thrown when a class does not contain the field (or variable) specified. |

### **User-Defined Exceptions**

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, a user can also create exceptions which are called ‘User-Defined Exceptions’.  
**Key points to note:**

* 1. A user-defined exception must extend Exception class.
  2. The exception is thrown using throw keyword.

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **class** MyException **extends** Exception{   String str1;   MyException(String str2) {str1=str2;}  **public** String toString(){  **return** ("MyException Occurred: "+str1);   }  }  **class** Example1{  **public** **static** **void** main(String args[]){  **try**{        System.out.println("Start of try block");  **throw** **new** MyException(“Error Message");      }  **catch**(MyException exp){System.out.println("Catch Block");      System.out.println(exp);   }  } |

Now that you have seen the different types of exceptions, let’s dive deeper into this Java Exception Handling blog to understand various methods for handling these exceptions.

## ****Exception Handling Methods****

### **How to handle exceptions in Java?**

As I have already mentioned, handling an exception is very important, else it leads to system failure. But how do you handle these exceptions?

Java provides various methods to handle the Exceptions like:

* try
* catch
* finally
* throw
* throws

Let’s understand each of these methods in detail.

### **try block**

The try block contains a set of statements where an exception can occur. It is always followed by a catch block, which handles the exception that occurs in the associated try block. A try block must be followed by catch blocks or finally block or both.

|  |  |
| --- | --- |
| 1  2  3 | **try**{  //code that may throw exception  }**catch**(Exception\_class\_Name ref){} |

### **Nested try block**

try block within a try block is known as nested try block in java.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **class** Exception{  **public** **static** **void** main(String args[]){  **try**{  **try**{            System.out.println("going to divide");  **int** b=59/0;           }**catch**(ArithmeticException e){System.out.println(e);}  **try**{  **int** a[]=**new** **int**[5];           a[5]=4;           }  **catch**(ArrayIndexOutOfBoundsException e) {System.out.println(e);}              System.out.println("other statement);          }**catch**(Exception e)           {System.out.println("Exception handeled");}         System.out.println("casual flow");      }  } |

### **catch block**

A catch block is where you handle the exceptions. This block must follow the try block and a single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in a try block, the corresponding catch block that handles that particular exception executes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **public** **class** Testtrycatch1{  **public** **static** **void** main(String args[]){  **int** data=50/0;//may throw exception      System.out.println("rest of the code...");   }  } |

### **Multi-catch block**

If you have to perform various tasks at the occurrence of various exceptions, you can use the multi-catch block.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **public** **class** SampleMultipleCatchBlock{  **public** **static** **void** main(String args[]){  **try**{  **int** a[]=**new** **int**[5];         a[5]=30/0;        }  **catch**(ArithmeticException e)          {System.out.println("task1 is completed");}  **catch**(ArrayIndexOutOfBoundsException e)          {System.out.println("task 2 completed");}  **catch**(Exception e)          {System.out.println("task 3 completed");}        System.out.println("remaining code");    }  } |

### **finally block**

A finally block contains all the crucial statements that must be executed whether an exception occurs or not. The statements present in this block will always execute, regardless an exception occurs in the try block or not such as closing a connection, stream etc.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **class** SampleFinallyBlock{  **public** **static** **void** main(String args[]){  **try**{  **int** data=55/5;       System.out.println(data);      }  **catch**(NullPointerException e)         {System.out.println(e);}  **finally** {System.out.println("finally block is executed");}      System.out.println("remaining code");    }  } |

So, this was all about the various methods of handling exceptions.

You might have heard that final, finally and finalize are keywords in Java. Yes, they are, but they differ from each other in various aspects. So, let’s see how **final, finally and finalize** are different from each other with the help of below table.

## ****final vs finally vs finalize****

|  |  |  |
| --- | --- | --- |
| **final** | **finally** | **finalize** |
| It is a keyword. | It is a block. | It is a method. |
| Used to apply restrictions on class, methods & variables. | Used to place an important code. | Used to perform clean-up processing just before the object is garbage collected. |
| final class can’t be inherited, method can’t be overridden & the variable value can’t be changed. | It will be executed whether the exception is handled or not. | – |

Similarly, throw & throws sound alike, but they are different from each other. Let’s see how, with the help of the below table.

## ****throw vs throws****

|  |  |
| --- | --- |
| **throw** | **throws** |
| 1. Used to explicitly throw an exception | 1. Used to declare an exception |
| 2. Checked exceptions cannot be propagated using throw only | 2. Checked exceptions can be propagated |
| 3. Followed by an instance | 3. Followed by a class |
| 4. Used within a method | 4. Used with a method signature |
| 5. Cannot throw multiple exceptions | 5. Can declare multiple exceptions |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | //Java throw example  **void** a()  {  **throw** **new** ArithmeticException("Incorrect");  }  //Java throws example  **void** a()**throws** ArithmeticException  {    //method code  }  //Java throw and throws example  **void** a()**throws** ArithmeticException  {  **throw** **new** ArithmeticException("Incorrect");  } |

