Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions. Here, we will learn what is an exception, types of it, exception classes and how to handle exceptions in java with examples.

What is an exception?

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exception Handling

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user. For example look at the system generated exception below:
An exception generated by the system is given below

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
ExceptionDemo.main(ExceptionDemo.java:5)
 ExceptionDemo : The class name
 main : The method name
 ExceptionDemo.java : The filename
 java:5 : Line number
```

This message is not user friendly so a user will not be able to understand what went wrong. In order to let them know the reason in simple language, we handle exceptions. We handle such conditions and then prints a user friendly warning message to user, which lets them correct the error as most of the time exception occurs due to bad data provided by user.

Advantage of exception handling

Exception handling ensures that the flow of the program doesn’t break when an exception occurs. For example, if a program has bunch of statements and an

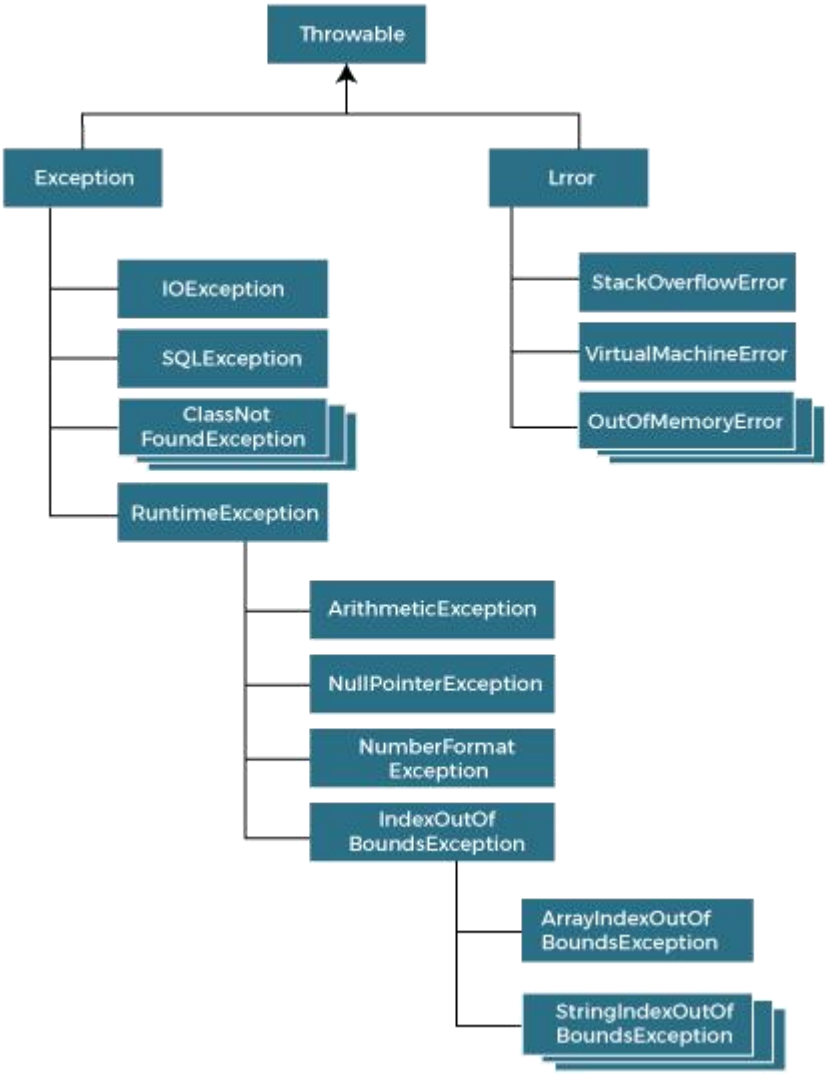
exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.
By handling we make sure that all the statements execute and the flow of program doesn't break.

Difference between error and exception

Errors indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions. Few examples:
NullPointerException – When you try to use a reference that points to null.
ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.

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 exceptions

There are two types of exceptions in Java:
1)Checked exceptions
2)Unchecked exceptions

Checked exceptions

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

Unchecked Exceptions

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Compiler will never force you to catch such exception or force you to declare it in the method using throws keyword.

```
public void myMethod() throws ArithmeticException, NullPointerException
{
  // Statements that might throw an exception
}

public static void main(String args[]) {
  try {
    myMethod();
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
}
```

Example of throws Keyword

In this example the method myMethod() is throwing two **checked exceptions** so we have declared these exceptions in the method signature using **throws** Keyword. If we do not declare these exceptions then the program will throw a compilation error.

```
import java.io.*;
class ThrowExample {
  void myMethod(int num)throws IOException, ClassNotFoundException{
    if(num==1)
      throw new IOException("IOException Occurred");
    else
      throw new ClassNotFoundException("ClassNotFoundException");
  }
}
```

```
public class Example1{
  public static void main(String args[]){
   try{
     ThrowExample obj=new ThrowExample();
     obj.myMethod(1);
   }catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```

Output:

java.io.IOException: IOException Occurred

```
try{
//
}
catch(ArithmeticException ae){

}
catch(NullPointerException npe){

}
catch(ArrayIndexOutOfBoundsException aoe){

}catch(Exception e){
System.out.println("Exception caught");
}
```