Exception Handling in Java

Iroshan Aberathne Follow
Nov 25, 2019 · 4 min read

Most of the beginners are struggling to understand exception and the proper ways of handling them. The main intention of this small article is to give some insight on exception handling with some of coding examples.

What is an Exception ?

1. An Exception is an abnormal condition that arises in a code sequence.
2. In other computer languages that do not support exception handling, errors must be checked and handle manually through the error codes.
3. Java brings run time error management in to the object oriented world.
4. Exceptions can be handled by the programmer (try-catch) or handle by the java environment(throws).

Some Examples

1. **Java.lang.ArrayIndexOutOfBoundsException**

Here the length of the array is 5 then last index will be 4 (length — 1) but for loop iterates until fifth index.

```
public class ExceptionDemoOne{
    public static void main(String[] arg){
        int[] arr = {3,4, 5, 6,9};
        for(int x=0; x<6; x++){
            System.out.println(arr[x]);
        }
    }
}
```

2. Java.lang.NullPointerException

Student type array is initialized with the size of five but without initializing zeroth index student object developer tries to assign values for name and age attributes.

```
class Student{
    int age;
    String name;
}
public class ExceptionDemoTwo{
    public static void main(String[] arg){
        Student [] students = new Student[5];
        students [0].age = 21;
        students [0].name= "Kamal";
    }
}
```

3. Java.lang.ArithmaticException:/by zero

The java interpreter will be confused at 56/a statement since a = 0 and interpreter is unable to get exact answer for 56/a.

```
public class ExceptionDemoThree{
    public static void main(String[] arg){
        int a  =0;
        float avg = 56/a;
        System.out.println(avg);
    }
}
```

Types of Exception

Compile Time Exception — Checked Exception

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.

1. SQLException
2. IOException

3. ClassNotFoundException

Run Time Exception — Unchecked Exception

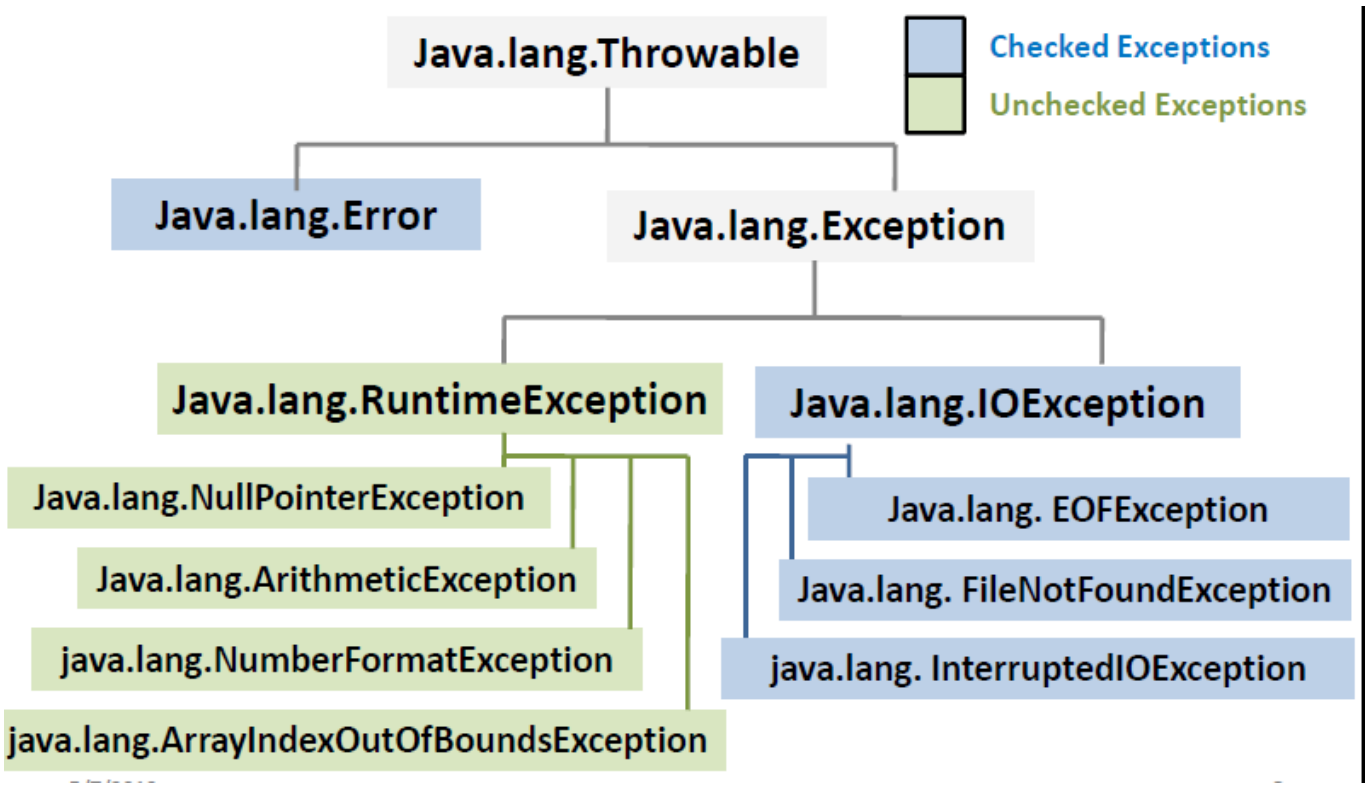
These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit.

1. ArithmeticException
2. NulPointerException
3. ArrayIndexOutOfBoundsException

What is an Error ?

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.

Exception Inheritance Hierarchy



Exception Hierarchy

How do we handle Exception ?

1. Exception handling is accomplished through the “try-catch” mechanism, or by “throws” clause in the method declaration.
2. For any code that throws a checked exception, you can decide to handle the exception yourself, or pass the exception “up the chain” (to a parent class).
3. To handle the exception, you write a “try-catch” block. To pass the exception “up the chain”, you declare a “throws” clause in your method or class declaration.
4. If the method contains code that may cause a checked exception, you MUST handle the exception OR pass the exception to the parent class.

Try-catch Mechanism

Try block

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

Catch block

A catch block is where you handle the exceptions, this block must follow the try block. 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 try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

try-catch example:

```
class Example{
    public static void main(String[] arg) {
        int x, y;
        try{
```

```
            x = 0;
            y = 34/x;
            System.out.println(y);
        } catch(ArithmeticException e) {
            System.out.println("Should not divide  number by zero");
        }
        System.out.println("Out of try-catch block in Java.");
    }
}
```

try with multiple catch blocks example:

```
class Example{
    public static void main(String[] arg) {
        try{
            int[] x = new int[7];
            x[0] = 34/0;
            System.out.println("First Print in Try Block");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException");
        }
        System.out.println("Out of try-catch block in Java.");
    }
}
/************************ OUT PUT ************************/
Arithmetic Exception
Out of try-catch block in Java.

Process finished with exit code 0
```

Finally block

1. You can attach a finally-clause to a try-catch block.
2. The code inside the finally clause will always be executed, even if an exception is thrown from within the try or catch block.
3. If your code has a return statement inside the try or catch block, the code inside the finally-block will get executed before returning from the method.

```
class Example{
    public static void main(String[] arg) {
        try{
            int num =12/0;
            System.out.println(num);
        } catch (ArithmeticException e) {
            System.out.println("Divide by zero error");
        } finally {
            System.out.println("This will Execute Finally");
        }
    }
}
/************************ OUT PUT ************************/
Divide by zero error
This will Execute Finally

Process finished with exit code 0
```

I think this article will help you to get basic understanding on exception handling.