Bahria University
Discovering Knowledge

Object Oriented Programming

Topic: Exception Handling in Java

Bahria University
Discovering Knowledge

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is Exception in Java?

Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Bahria University
Discovering Knowledge

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as
ClassNotFoundException,
IOException,
SQLException,
RemoteException, etc.

Advantage of Exception Handling

The core advantage of exception handling is ***to maintain the normal flow of the application***. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

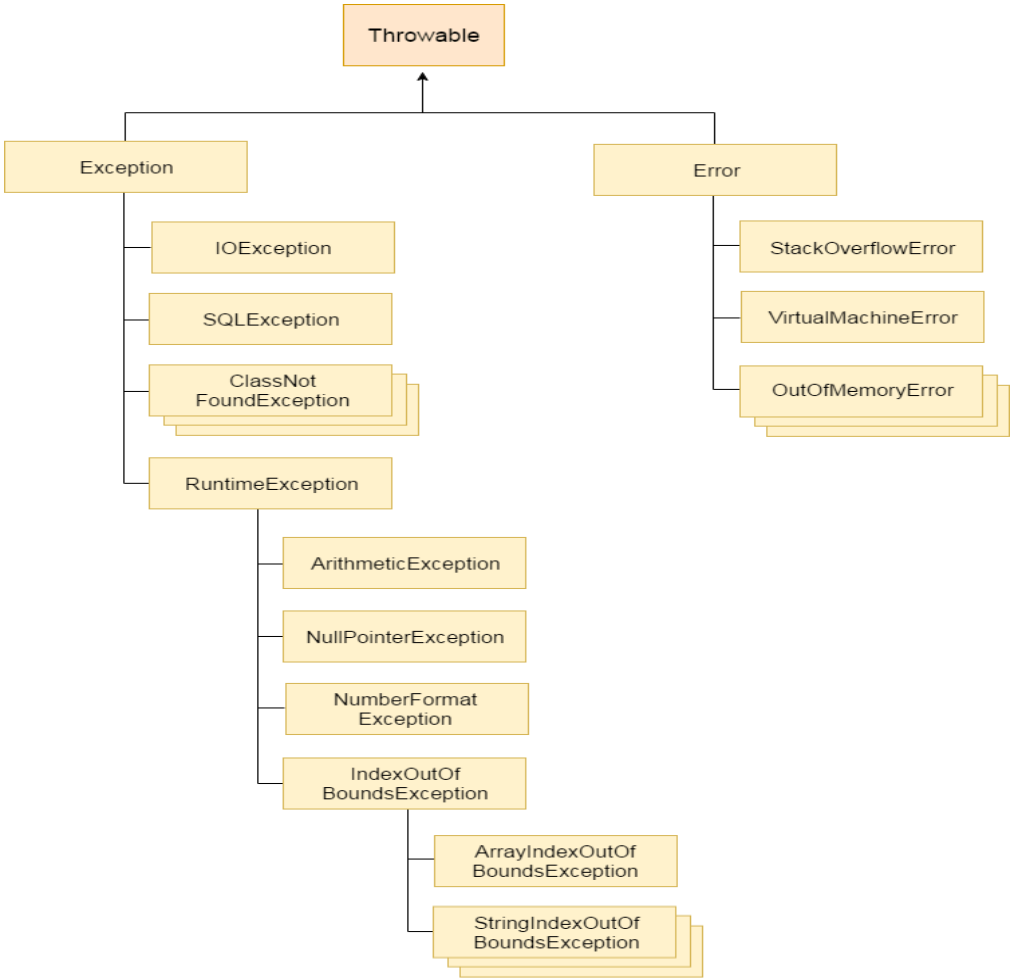
Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Bahria University
Discovering Knowledge

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. **Checked Exception**
2. **Unchecked Exception**
3. **Error**

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Bahria University
Discovering Knowledge

Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

OUTPUT

Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

Bahria University
Discovering Knowledge

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

2) A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

Bahria University
Discovering Knowledge

Common Scenarios of Java Exceptions

3) A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

```
public class Unchecked_Demo
{ public static void main(String args[])
{ int num[] = {1, 2, 3, 4};
System.out.println(num[5]); }
}
```

Arithmetic Exception

```
class ArithmeticException_Demo
{
        public static void main(String args[])
        {
                try {
                        int a = 30, b = 0;
                        int c = a/b; // cannot divide by zero
                        System.out.println ("Result = " + c);
                }
                catch(ArithmeticException e) {
                        System.out.println ("Can't divide a number by 0");
                }
        }
}
```

NullPointer Exception

```
class NullPointer_Demo
{
        public static void main(String args[])
        {
                try {
                        String a = null; //null value
                        System.out.println(a.charAt(0));
                } catch(NullPointerException e) {
                        System.out.println("NullPointerException..");
                }
        }
}
```

Bahria University
Discovering Knowledge

StringIndexOutOfBounds Exception

```
class StringIndexOutOfBound_Demo
{
        public static void main(String args[])
        {
                try {
                        String a = "This is like chipping "; // length is 22
                        char c = a.charAt(24); // accessing 25th element
                        System.out.println(c);
                }
                catch(StringIndexOutOfBoundsException e) {
                        System.out.println("StringIndexOutOfBoundsException");
                }
        }
}
```

Bahria University
Discovering Knowledge

FileNotFound Exception

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class File_notFound_Demo {

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

                        // Following file does not exist
                        File file = new File("E://file.txt");

                        FileReader fr = new FileReader(file);
                } catch (FileNotFoundException e) {
                System.out.println("File does not exist");
                }
        }
}
```

NumberFormat Exception

```
class NumberFormat_Demo
{
        public static void main(String args[])
        {
                try {
                        // "akki" is not a number
                        int num = Integer.parseInt ("akki") ;

                        System.out.println(num);
                } catch(NumberFormatException e) {
                        System.out.println("Number format
exception");
                }
        }
}
```

Bahria University
Discovering Knowledge

ArrayIndexOutOfBound Exception

```
class ArrayIndexOutOfBound_Demo
{
        public static void main(String args[])
        {
                try{
                        int a[] = new int[5];
                        a[6] = 9; // accessing 7th element in an array of
                                                // size 5
                }
                catch(ArrayIndexOutOfBoundsException e){
                        System.out.println ("Array Index is Out Of
Bounds");
                }
        }
}
```

Bahria University
Discovering Knowledge

User-Defined Exceptions

•User can also create exceptions which are called ‘user-defined Exceptions’.
•The built-in exceptions in Java are not able to describe a certain situation.
Following steps are followed for the creation of user-defined Exception.

The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as

```
class MyException extends Exception
```

We can write a default constructor exception class.

```
MyException(){}
```

Cont.

We can also create a parameterized constructor with a string as a parameter. Where we can store exception details. We can call super class(Exception) constructor from this and send the string there.

```
MyException(String str)
    { super(str); }
```

To raise exception of user-defined type, we need to create an object of exception class and throw it using throw clause, as:

```
MyException me = new MyException("Exception details");
throw me;
```

Cont.

The following program illustrates how to create own exception class MyException. Details of account numbers, customer names, and balance amounts are taken in the form of three arrays.
In main() method, the details are displayed using a for-loop. At this time, check is done if in any account the balance amount is less than the minimum balance amount to be ept in the account.
If it is so, then MyException is raised and a message is displayed “Balance amount is less”.

```
class MyException extends Exception

        private static int accno[] = {1001, 1002, 1003, 1004};
        private static String name[] = {"Nish", “Ahmed", “Ali", “Sana", "Akash"};
        private static double bal[] = {10000.00, 12000.00, 5600.0, 999.00, 1100.55};
        MyException() { }
        MyException(String str)
       { super(str); }
        public static void main(String[] args)
        {
                try {
                        System.out.println("ACCNO" + "\t" + "CUSTOMER" + "\t" + "BALANCE");
                        for (int i = 0; i < 5 ; i++)
                        {
                                System.out.println(accno[i] + "\t" + name[i] + "\t" + bal[i]);
                                if (bal[i] < 1000)
                                {
                                MyException me = new MyException("Balance is less than 1000");
                                throw me;
                                }
                        }
                }

                catch (MyException e) {
        e.printStackTrace();
        //method of Java.lang shows  details like class name and line number where the exception occurred
                } } }
```

Java catch multiple exceptions

• A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

• At a time only one exception occurs and at a time only one catch block is executed.

• All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Example

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
    }
}
```

Example

```
public class MultipleCatchBlock2 {

   public static void main(String[] args) {

        try{
            int a[]=new int[5];

            System.out.println(a[10]);
           }
           catch(ArithmeticException e)
             {
              System.out.println("Arithmetic Exception occurs");
             }
           catch(ArrayIndexOutOfBoundsException e)
             {
              System.out.println("ArrayIndexOutOfBounds Exception occurs");
             }
           catch(Exception e)
             {
              System.out.println("Parent Exception occurs");
             }
           System.out.println("rest of the code");
   }
}
```

Finally block in Java

Finally block is always executed whether an exception is thrown or not.
Or The finally block is **executed irrespective of an exception being raised** in the try block. It is **optional** to use with a try block.

The finally block always executes immediately after try-catch block exits.
The finally block is executed incase even if an unexpected exception occurs.
The main usage of finally block is to do clean up job. Keeping cleanup code in a finally block is always a good practice, even when no exceptions are occured.
The runtime system always executes the code within the finally block regardless of what happens in the try block. So it is the ideal place to keep cleanup code.

Finally block code

```
class Allocate {
  public static void main(String[] args) {
    try {
      long data[] = new long[1000000000];
    }
    catch (Exception e) {
      System.out.println(e);
    }
    finally {
      System.out.println("finally block will
execute always.");
    }
  }
}
```

Bahria University
Discovering Knowledge

Throw Clause Examples

Use ***'throw'*** statement to throw an exception or simply use the throw clause with an object reference to throw an exception.
The syntax is ***'throw new Exception();'.*** Even you can pass the error message to the Exception constructor.

```
package com.myjava.exceptions;

public class MyExplicitThrow {
    public static void main(String a[]){
        try{
            MyExplicitThrow met = new MyExplicitThrow();
            System.out.println("length of JUNK is "+met.getStringSize("JUNK"));
            System.out.println("length of WRONG is "+met.getStringSize("WRONG"));
            System.out.println("length of null string is "+met.getStringSize(null));
        } catch (Exception ex){
            System.out.println("Exception message: "+ex.getMessage());
        }
    }

    public int getStringSize(String str) throws Exception{
        if(str == null){
            throw new Exception("String input is null");
        }
        return str.length();
    }
}
```

Bahria University
Discovering Knowledge

Throws Clause

The ***'throws'*** clause in java programming language is belongs to a method to specify that the method raises particular type of exception while being executed.
The ***'throws'*** clause takes arguments as a list of the objects of type ***'Throwables'*** class.
Anybody calling a method with a throws clause is needed to be enclosed within the try catch blocks.

```
package com.myjava.exceptions;

public class MyThrowsClause {
    public static void main(String a[]){
        MyThrowsClause mytc = new MyThrowsClause();
        try{
            for(int i=0; i<5; i++){
                mytc.getJunk();
                System.out.println(i);
            }
        } catch (InterruptedException iex){
            iex.printStackTrace();
        }
    }

    public void getJunk() throws InterruptedException {
        Thread.sleep(1000);
    }
}
```

https://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html