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 Handling

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

### dvantage 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:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

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.

## 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. |

**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...

## Common Scenarios of Java Exceptions

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

### 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

### 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

## Java try block

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception.

## Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception ( i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

## Problem without exception handling

**public** **class** TryCatchExample1 {

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

**int** data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}

**Output:**

Exception in thread "main" java.lang.ArithmeticException: / by zero

As displayed in the above example, the **rest of the code** is not executed (in such case, the **rest of the code** statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

## Solution by exception handling

**public** **class** TryCatchExample2 {

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

**try**

        {

**int** data=50/0; //may throw exception

        }

            //handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

java.lang.ArithmeticException: / by zero

rest of the code