

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

Dictionary Meaning: 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.

What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IO, SQL, Remote 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:

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

Handling the Exception

We use five keywords for Handling the Exception

- try
- catch
- finally
- throws
- throw

Syntax for handling the exception

```
try
{
    // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
    // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
    // statements provides user friendly error message
}
```

```
finally
{
    // statements which will execute compulsory
}
```

1. try block

inside try block we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even though we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try
{
    ....
}
/* Here no other statements are allowed
between try and catch block */
catch()
{
    ....
}
```

- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
- One try block can contains another try block that is nested or inner try block can be possible.

Syntax

```
try
{
    .....
    try
    {
        .....
    }
}
```

2. catch block

Inside catch block we write the block of statements which will generates user friendly error messages.

catch block important points

- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

Example without Exception Handling

```
class ExceptionDemo
```

```

{
public static void main(String[] args)
{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be zero");
}
}

```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".



```

C:\Windows\system32\cmd.exe
D:\java>javac ExceptionDemo.java
D:\java>java ExceptionDemo
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo.main(ExceptionDemo.java:8)
D:\java>

```

Example with Exception Handling

```

class ExceptionDemo
{
public static void main(String[] args){
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}

```

Output

Denominator not be zero

Multiple catch block

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.

Example

```

import java.util.*;
class ExceptionDemo
{
public static void main(String[] args)
{
int a, b, ans=0;
Scanner s=new Scanner(System.in);
System.out.println("Enter any two numbers: ");
try
{
a=s.nextInt();
b=s.nextInt();
}
}
}

```

```

        ans=a/b;
        System.out.println("Result: "+ans);
    }
    catch(ArithmeticException ae)
    {
        System.out.println("Denominator not be zero");
    }
    catch(Exception e)
    {
        System.out.println("Enter valid number");
    }
}
}
}

```

Output

Enter any two number: 5 0
Denominator not be zero

3. finally block

Inside finally block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

finally block important points

- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block.

Example

```

class ExceptionDemo
{
    public static void main(String[] args)
    {
        int a=10, ans=0;
        try
        {
            ans=a/0;
        }
        catch (Exception e)
        {
            System.out.println("Denominator not be zero");
        }
        finally
        {
            System.out.println("I am from finally block");
        }
    }
}

```

Output

Denominator not be zero
I am from finally block

4. throw

throw is a keyword in java language which is used to throw any user defined exception to the same signature of method in which the exception is raised.

Note: throw keyword always should exist within method body.

whenever method body contain throw keyword than the call method should be followed by throws keyword.



Syntax

```
class className{
returntype method(...) throws Exception_class
{
throw(Exception obj)
}
}
```

5. throws

throws is a keyword in java language which is used to throw the exception which is raised in the called method to it's calling method throws keyword always followed by method signature.

Example

```
returnType methodName(parameter)throws Exception_class....
{
.....
}
```

Difference between throw and throws

| | throw | throws |
|---|--|---|
| 1 | throw is a keyword used for hitting and generating the exception which are occurring as a part of method body | throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages |
| 2 | The place of using throw keyword is always as a part of method body. | The place of using throws is a keyword is always as a part of method heading |
| 3 | When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading | When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body. |

Example of throw and throws

```
// save by DivZero.java

package pack;

public class DivZero
{
    public void division(int a, int b)throws ArithmeticException
    {
        if(b==0)
        {
            ArithmeticException ae=new ArithmeticException("Does not enter zero for Denominator");
            throw ae;
        }
        else
        {
            int c=a/b;
            System.out.println("Result: "+c);
        }
    }
}
```

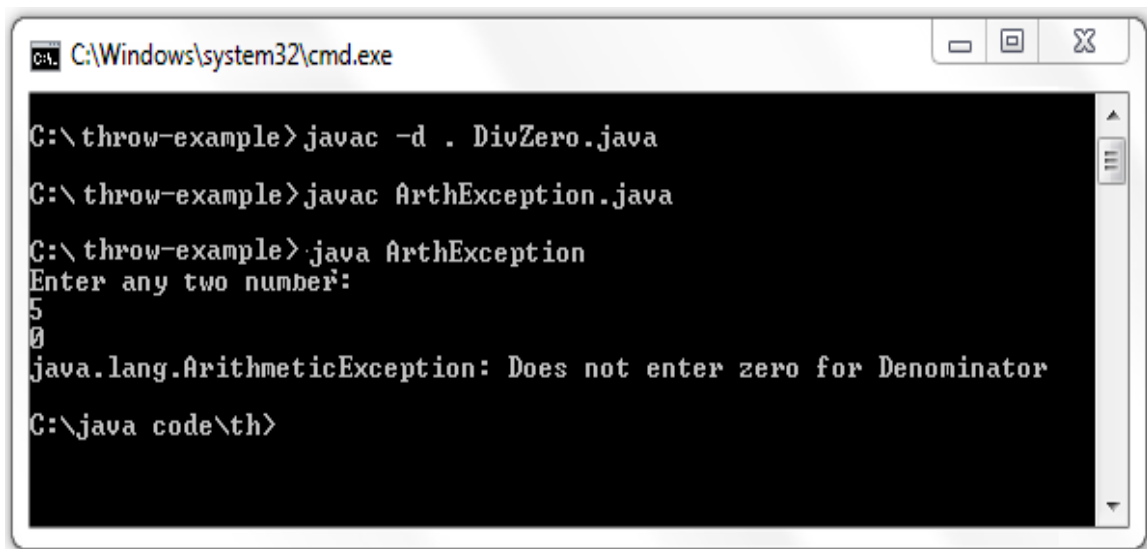
Compile: javac -d . DivZero.java

```
// save by ArthException.java

import pack.DivZero;
import java.util.*;

class ArthException
{
    public static void main(String args[])
    {
        System.out.println("Enter any two number: ");
        Scanner s=new Scanner(System.in);
        try
        {
            int a=s.nextInt();
            int b=s.nextInt();
            DivZero dz=new DivZero();
            dz.division(a, b);
        }
        catch(Exception e)
        {
            System.err.println(e);
        }
    }
}
```

Compile: javac ArthException.java



```
C:\Windows\system32\cmd.exe

C:\throw-example>javac -d . DivZero.java
C:\throw-example>javac ArthException.java
C:\throw-example>java ArthException
Enter any two number:
5
0
java.lang.ArithmeticException: Does not enter zero for Denominator
C:\java code\th>
```