

ASSIGNMENT NO: 6

Problem Statement: Implement a program to handle Arithmetic exception, Array Index Out of Bounds. The user enters two numbers Num1 and Num2. The division of Num1 and Num2 is displayed. If Num1 and Num2 were not integers, the program would throw a Number Format Exception. If Num2 were zero, the program would throw an Arithmetic Exception. Display the exception.

Aim: Exception handling **Theory:**

Exception handling in java with examples

Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions. In this guide, 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 a 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 occur 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.

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.

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.

Syntax of try block try{

//statements that may cause an exception

}

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

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.

Syntax of try catch in java try

{

```
//statements that may cause an exception
}
catch(exception(type) e(object))
{
//error handling code
}
```

Example: try catch block

If an exception occurs in try block then the control of execution is passed to the corresponding catch block. A single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last (see in the example below).

The generic exception handler can handle all the exceptions but you should place it at the end, if you place it at the before all the catch blocks then it will display the generic message. You always want to give the user a meaningful message for each type of exception rather than a generic message.

```
classExamp classExample1{
publicstaticvoid main(Stringargs[]){
int num1, num2;
try{
/* We suspect that this block of statement can throw
* exception so we handled it by placing these statements
* inside try and handled the exception in catch block
*/
num1 =0; num2
=62/ num1;
System.out.println(num2);
System.out.println("Hey I'm at the end of try block");
}
catch(ArithmeticException e){
/* This block will only execute if any Arithmetic exception
* occurs in try block
*/
```

```

System.out.println("You should not divide a number by zero");
}
catch(Exception e){
/* This is a generic Exception handler which means it can handle
* all the exceptions. This will execute if the exception is not * handled by previous
catch blocks.
*/
System.out.println("Exception occurred");
}
System.out.println("I'm out of try-catch block in Java.");
}
}

```

Output:

You should not divide a number by zero I'm
out of try-catch block in Java.

Multiple catch blocks in Java

The example we seen above is having multiple catch blocks, lets see few rules about multiple catch blocks with the help of examples. To read this in detail, see catching multiple exceptions in java.

1. As mentioned above, a single try block can have any number of catch blocks.
2. A generic catch block can handle all the exceptions. Whether it is `ArrayIndexOutOfBoundsException` or `ArithmeticException` or `NullPointerException` or any other type of exception, this handles all of them. To see the examples of `NullPointerException` and `ArrayIndexOutOfBoundsException`, refer this article: [Exception Handling example programs.](#)

```

{
//This catch block catches all the exceptions
}

```

If you are wondering why we need other catch handlers when we have a generic that can handle all. This is because in generic exception handler you can display a message but you are not sure for which type of exception it may trigger so it will display the same message for all the exceptions and user may not be able to understand which exception occurred. Thats the reason you should place is at the end of all the specific exception catch blocks.

3. If no exception occurs in try block then the catch blocks are completely ignored.

4. Corresponding catch blocks execute for that specific type of exception:

`catch(ArithmeticException e)` is a catch block that can handle `ArithmeticException`

`catch(NullPointerException e)` is a catch block that can handle `NullPointerException`

5. You can also throw exception, which is an advanced topic and I have covered it in separate tutorials: user defined exception, throws keyword, throw vs throws.

Example of Multiple catch blocks `classExample2{ publicstaticvoid`

`main(Stringargs[]){ try{ int a[]=newint[7]; a[4]=30/0;`

`System.out.println("First print statement in try block");`

`}`

`catch(ArithmeticException e){`

`System.out.println("Warning: ArithmeticException");`

`}`

`catch(ArrayIndexOutOfBoundsException e){`

`System.out.println("Warning: ArrayIndexOutOfBoundsException");`

`}`

`catch(Exception e){`

`System.out.println("Warning: Some Other exception");`

`}`

`System.out.println("Out of try-catch block...");`

`}`

`}`

Output:

Warning: ArithmeticException Out

of try-catch block...

In the above example there are multiple catch blocks and these catch blocks executes sequentially when an exception occurs in try block. Which means if you put the last catch block (`catch(Exception e)`) at the first place, just after try block then in case of any exception this block will execute as it can handle all exceptions. This catch block should be placed at the last to avoid such situations.

Finally block

You should place those statements in finally blocks, that must execute whether exception occurs or not.

Input: Values of Text field T1, T2 Output:

Displays the result in Text field T3 Step1:

Start.

Step2: Import java.awt package

Step3: Import java.lang.string,awt. event,applet.Applet packages.

Step4: Create Class

Step5: Create Buttons and Text Fields.

Step6: Create the Data.

Step7: Perform the division.

Step8: Print the Data.

Step9: Stop.

Program:

```
import java .util.Scanner; public
class Try_Catch_6
{
    public static void main(String[] args) {
Scanner sc= new Scanner(System.in);      try
{
    System.out.print("Enter 1st number- ");
    String num1 = sc.next();
    System.out.print("Enter 2nd number- ");
String num2 = sc.next();      int a =
Integer.parseInt(num1);      int b =
Integer.parseInt(num2);
System.out.println("Result:- "+ a/b);
    }
```

```

        catch(ArithmeticException e){
            System.out.println();
            System.out.println(e+"\n---Cannot divide number by zero---");
        }
        catch(NumberFormatException e){
            System.out.println();
            System.out.println(e+"\n---Enter an integer value---");
        }
    }
}

```

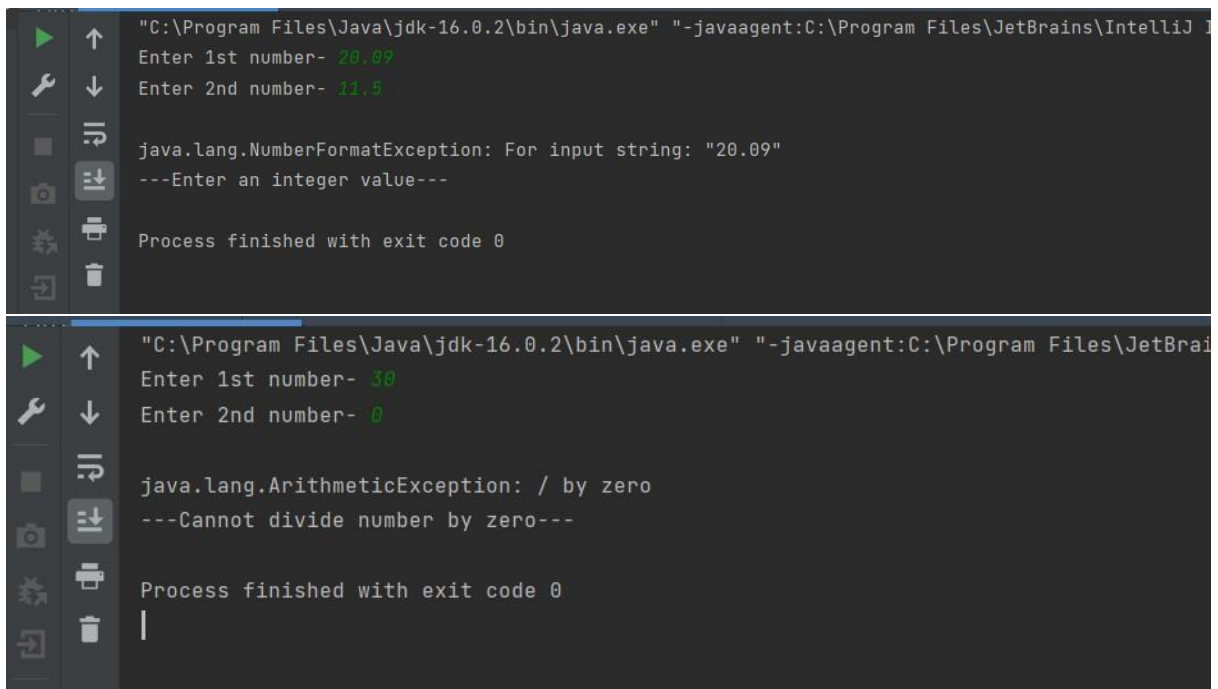


```

2
3 import java.util.Scanner;
4 public class Try_Catch_6
5 {
6     public static void main(String[] args) {
7         Scanner sc= new Scanner(System.in);
8         try {
9             System.out.print("Enter 1st number- ");
10            String num1 = sc.next();
11            System.out.print("Enter 2nd number- ");
12            String num2 = sc.next();
13            int a = Integer.parseInt(num1);
14            int b = Integer.parseInt(num2);
15            System.out.println("Result:- "+ a/b);
16        }
17        catch(ArithmeticException e){
18            System.out.println();
19            System.out.println(e+"\n---Cannot divide number by zero---");
20        }
21        catch(NumberFormatException e){
22            System.out.println();
23            System.out.println(e+"\n---Enter an integer value---");
24        }
25    }
26 }

```

Output:



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ I
Enter 1st number- 20.09
Enter 2nd number- 11.5

java.lang.NumberFormatException: For input string: "20.09"
---Enter an integer value---

Process finished with exit code 0

"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrai
Enter 1st number- 30
Enter 2nd number- 0

java.lang.ArithmeticException: / by zero
---Cannot divide number by zero---

Process finished with exit code 0
|
```

Conclusion: Thus, we have studied exception handling concept using java