Exception handling in java

Presentation · December 2019

DOI: 10.13140/RG.2.2.36568.03843

CITATIONS
0

READS
191

1 author:

Hana Esmaeel
Al-Nahrain University
25 PUBLICATIONS **16** CITATIONS

SEE PROFILE

Exception handling

BY

Senior lecturer

Hana Rashied Esmaeel

Def: Exception handling enables programmers to remove error-handling code from the "main line" of the program's execution, improving program clarity and enhancing modifiability. Programmers can decide to handle any exceptions they choose .First we demonstrate what happens when errors arise in a program that does not use exception handling, In this example, we will see that exceptions are thrown (i.e., the exception occurs) when a method detects a problem and is unable to handle it.

A **catch block** (also called a **catch clause** or **exception handler**) catches (i.e., receives) and handles an exception. A catch block begins with the keyword catch and is followed by a parameter in parentheses (called the exception parameter) and a block of code enclosed in curly braces.

A try/catch block tells the compiler that you know an exceptional thing could happen in the method you're calling, and that you're prepared to handle it. That compiler doesn't care how you handle it; it cares only that you say you're taking care of it.

Exception Handling Syntax:

```
try
{

//code to be tried for errors

}

catch(ExceptionType1 obj1)
{

//Exception handler for ExceptionType1
}
catch(ExceptionType2 obj2)
{
//Exception handler for ExceptionType2
}
```

finally
{
//code to be executed before try block ends.
This executes whether or not an //
exception occurs in the try block.
}
Figure(1) below show some Exception in java languages:

List of exceptions in Java(part of java.lang package):

Essential exception classes include -

Exception	Description
ArithmeticException	Caused by exceptional conditions like divide by zero
ArrayIndexOfBounds Exception	Thrown when an array is accessed beyond its bounds
ArrayStoreException	Thrown when an incompatible type is stored in an array
ClassCastException	Thrown when there is an invalid cast
IllegalArgument Exception	Thrown when an inappropriate argument is passed to a method
IllegalMonitorState Exception	Illegal monitor operations such as waiting on an unlocked thread
IllegalThreadState Exception	Thrown when a requested operation is incompatible with the current thread state.
IndexOutOfBounds Exception	Thrown to indicate that an index is out of range.
NegativeArraySize Exception	Thrown when an array is created with negative size.

Figure(1) list of some exception in java

Example (1): Ask the user to enter two integers then passes them to method **(quotient)**, which calculates the
quotient and returns an **int** result.

Sol:

```
import java.util.Scanner;

public class DivideByZeroNo
{

 public static int quotient( int num1, int num2 )
 {
  return num1/num2 ; // possible division by zero
 } // end method quotient

 public static void main( String args[] )
  {
   Scanner scanner = new Scanner( System.in ); // scanner for input
   System.out.print( "Please enter first integer number: " );
   int num1 = scanner.nextInt();
   System.out.print( "Please enter second integer number: " );
   int num2 = scanner.nextInt();
   int result = quotient( num1, num2 );
   System.out.printf("\nResult: %d / %d = %d\n", num1, num2, result );
  }
}
```

The output of the First run is:

```
Please enter first integer number:  100
Please enter second integer number:  7
Result: 100 / 7 = 14
```

The output of the second run is

```
Please enter first integer number: 100
Please enter second integer number: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at DivideByZeroNo.quotient(DivideByZeroNo.java:8)
at DivideByZeroNo.main(DivideByZeroNo.java:19)   */
```

/* the output of the Third run is

```
Please enter first integer number: 100
Please enter second integer number: hello
```

Exception in thread "main" java.util.InputMismatchException */

Remark: In example (1) Both types of exceptions were detected in method **main.**

Example (2): In this example, we will see how to handle these exceptions to enable the program to run to normal completion.

```
import java.util.InputMismatchException;
import java.util.Scanner;

  public class DivideByZero
  {
     // demonstrates throwing an exception when a divide-by-zero occurs
    public static int quotient( int num1, int num2 ) throws ArithmeticException
    {
      return num1  / num2 ;                                  // possible division by zero
    }                                                         // end method quotient

    public static void main( String args[] )
    {
      Scanner scanner = new Scanner( System.in );     // scanner for input
      boolean continueLoop = true;                    // determines if more input is needed
      do
      {
       try            // read two numbers and calculate quotient
          {
           System.out.print( "Please enter first integer: " );
           int  num1= scanner.nextInt();
           System.out.print( "Please enter second integer: " );
           int num2 = scanner.nextInt();
           int result = quotient( num1, num2 );
           System.out.printf( "\nResult: %d / %d = %d\n", num1,num2, result );
           continueLoop = false;                          // input successful; end looping
           }                                               // end try
       catch ( InputMismatchException inputMismatchException )
          {
           System.err.printf( "\nException: %s\n", inputMismatchException );
           scanner.nextLine( );           // discard input so user can try again
           System.out.println( "You must enter integers. Please try again.\n" );
          }       // end catch
         catch ( ArithmeticException   arithmeticException )
         {
           System.err.printf( "\nException: %s\n", arithmeticException );
           System.out.println("Zero is an invalid denominator. Please try again.\n" );
```

```
            }       // end catch
        } while ( continueLoop );        // end do...while
    } // end main
} //      end  class  DivideByZero
```

The output of first run is:
Please enter first integer: 100
Please enter second integer: 7
Result: 100 / 7 = 14

The output of second run is:
Please enter first integer: 100
Please enter second integer: 0
Zero is an invalid denominator. Please try again.
Exception: java.lang.ArithmeticException: / by zero

Please enter first integer: 100
Please enter second integer: 7
Result: 100 / 7 = 14
The output of third run is :
Please enter first integer: 100
Please enter second integer: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter first integer: hello
Exception: java.util.InputMismatchException
You must enter integers. Please try again.
Please enter first integer: 100
Please enter second integer: 8
Result: 100 / 8 = 12

Example (3): Write java application program which do the following:
1-initialize array of (10) integer numbers.
2-assign the value (1) to array [100]
3-find and print the maximum number
Solution

```
public class arrayassignment
{
  public static void main (String args[])
  {
     int Array[] = new int[10];
     System.out.print("the element of array are = ");
```

```
      for(int i=0;i<Array.length;i++)
        {
         Array[i]=i;
         System.out.print(Array[i]+" ");
        }
        System.out.println();
        System.out.println("after invalid array assignment");
        Array[100] = 1;
        System.out.println(Array[100]);
        int max=Array[0];
       for(int i=0;i<Array.length;i++)
        {
          if(Array[i]>max)
              max=Array[i];
       }
        System.out.println("maxmum value="+max);
        System.out.println("this is the end");
  }
}
```

The output of the program is:

```
the element of array are = 0 1 2 3 4 5 6 7 8 9
after invalid array assignment
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 100
at arrayassignment.main(arrayassignment.java:14)
```

Example (4):Write java application program using try and catch and do the following:

1-initialize array of (10)integer numbers.
2-assign the value (1) to array[100]
3-find and print the maximum number

Solution

```
public class Trycash
{
 public static void main (String args[])
 {
     int Array[] = new int[10];
     System.out.print("the element of array are = ");
     try {

       for(int i=0;i<Array.length;i++)
       {
        Array[i]=i;
       System.out.print(Array[i]+" ");
```

```
        }
        System.out.println("\n\nafter invalid array assignment");
        Array[100] = 1;
        System.out.println(Array[100]);
          }//end of try
       catch(ArrayIndexOutOfBoundsException e)
        {
         int max=Array[0];
         for(int i=0;i<Array.length;i++)
         {
          if(Array[i]>max)
              max=Array[i];
         }
         System.out.println("maxmum value="+max);
       }//end of cash
      finally
      {
       System.out.println("this is the end");
      }//end of finally
   }//end of main
}//end of class
```

The output of the program is

```
The element of array are = 0 1 2 3 4 5 6 7 8 9
after invalid array assignment
maxmum value=9
this is the end
```

Example (5):Using Exception handling technique write a program which read file name(file1) then print

1-file content

2-number of lines in that file

3-the "exit now" message

hint(try more than one if you not found that file

sol:

```
import java .io.*;
import java.util.Scanner;
public class read
{
   static String s;

   public static void main(String[] args) throws IOException
   {
      Scanner input=new Scanner(System.in);
   try{
      BufferedReader inone=new  BufferedReader(new FileReader("file1"));
     int count=0;
```

```
      String t=inone.readLine();
      while(t!=null)
      {
         System.out.println(t);
         count++;
         t=inone.readLine();
      }
     System.out.println(count+"lines");
    }//end try
    catch(FileNotFoundException x)
    {
      System.out.println("input file name");
      String s=input.nextLine();
       BufferedReader inone =new  BufferedReader(new FileReader(s));
       System.out.println("the file "+s+"is found" );
       System.out.println("it's content are :" );
       int count=0;
      String t=inone.readLine();
      while(t!=null)
      {
         System.out.println(t);
         count++;
         t=inone.readLine();
      } }
     catch(ArithmeticException  x1) //this catch is not  used if catch one is not match
     {
      System.out.println("input two integer numbers");
      int num1=input.nextInt();
      int num2=input.nextInt();
      int sum=num1+num2;
      int av=sum/2;
      System.out.println("the average is ="+av);
     }
    Finally
    {
      System.out.println("exit now");
    }    }}
```

the output is:

input file name

any name

the file any name is found

it's content are :

first line

second line

exit now

2-User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called 'user-defined Exceptions'.

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 in his own exception class.

MyException( ){ }

We can also create a parameterized constructor with a string as a parameter.

We can use this to 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 to his exception class and throw it using throw clause, as:

MyException me = new MyException("Exception details");
Throw me;

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 accept in the account.

If it is so, then MyException is raised and a message is displayed "Balance amount is less".

```
// Java program to demonstrate user defined exception
 // This program throws an exception whenever balance amount is below 1000
class MyException extends Exception
{
   //store account information
   private static int accno[] = {1001, 1002, 1003, 1004,1005};
     private static String name[] = {"Nish", "Shubh", "Sush", "Abhi", "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  {
    // display the heading for the table
   System.out.println("ACCNO" + "\t" + "CUSTOMER" +  "\t" +"BALANCE");
    // display the actual account information
```

```
 for (int i = 0; i < 5 ; i++)
  {
  System.out.println(accno[i] + "\t" + name[i] +  "\t" + bal[i]);
                // display own exception if balance < 1000
               if (bal[i] < 1000)
               {
  MyException  me =new  MyException("Balance is less than 1000");
                  throw  me;
   }
          }
       } //end of try
         catch (MyException  e) {
           e.printStackTrace();
       }
   }
}
```

RunTime Error

```
 MyException: Balance is less than 1000
    at MyException.main(fileProperty.java:36)
```

Output:

ACCNO	CUSTOMER	BALANCE
1001	Nish	10000.0
1002	Shubh	12000.0
1003	Sush	5600.0
1004	Abhi	999.0

How to throw exception in java with example

In Java we have already defined exception classes such as ArithmeticException, NullPointerException, ArrayIndexOutOfBounds exception etc. These exceptions are set to trigger on different-2 conditions. For example when we divide a number by zero, this triggers ArithmeticException, when we try to access the array element out of its bounds then we get ArrayIndexOutOfBoundsException.

We can define our own set of conditions or rules and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException when we divide number by 5, or any other numbers, what we need to do is just set the condition and throw any exception using throw keyword. Throw keyword can also be used for throwing custom exceptions,

Syntax of throw keyword:

```
throw   new  exception_class("error message");
```

For example:

```
throw  new ArithmeticException("dividing a number by 5 is not allowed in this
program");
```

Example of throw keyword

Assume we have a requirement where we need to only register the students when their age is less than 12 and weight is less than 40, if any of the condition is not met then the user should get an ArithmeticException with the warning message “Student is not eligible for registration”. We have implemented the logic by placing the code in the

method that checks student eligibility if the entered student age and weight doesn't met the criteria then we throw the exception using throw keyword.

```
/* In this program we are checking the Student age
 * if the student age<12 and weight <40 then our program
 * should return that the student is not eligible for registration.
 */
public class ThrowExample {
   static void checkEligibilty(int stuage, int stuweight){
      if(stuage<12 && stuweight<40) {
         throw new ArithmeticException("Student is not eligible for registration");
      }
      else {
         System.out.println("Student Entry is Valid!!");
      }
   }

   public static void main(String args[]){
     System.out.println("Welcome to the Registration process!!");
     checkEligibilty(10, 39);
     System.out.println("Have a nice day..");
 }
}
```

Output:

```
Welcome to the Registration process!!Exception in thread "main"
java.lang.ArithmeticException: Student is not eligible for registration
at beginnersbook.com.ThrowExample.checkEligibilty(ThrowExample.java:9)
at beginnersbook.com.ThrowExample.main(ThrowExample.java:18)
```

Throws clause in java – Exception handling

Throws keyword is used for handling checked exceptions . By using throws we can declare multiple exceptions in one go.

What is the need of having throws keyword when you can handle exception using try-catch?

Well, thats a valid question. We already know we can handle exceptions using try-catch block.

The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch. For example:

Lets say we have a method myMethod( ) that has statements that can throw either ArithmeticException or NullPointerException, in this case you can use try-catch as shown below:

```
public void myMethod()
{
  try {
    // Statements that might throw an exception
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
```

```
}
```
But suppose you have several such methods that can cause exceptions, in that case it would be boring to write these try-catch for each method. The code will become unnecessary long and will be less-readable.
One way to overcome this problem is by using throws like this:
declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.
Another advantage of using this approach is that you will be forced to handle the exception when you call this method, all the exceptions that are declared using throws, must be handled where you are calling this method else you will get compilation error.

```
public void myMethod( ) throws ArithmeticException, NullPointerException
{
  // Statements that might throw an exception
}
public static void main(String  args[ ]) {
  try {
    myMethod();
  }
  catch (ArithmeticException  e) {
    // Exception handling statements
  }
  catch (NullPointerException  e) {
    // Exception handling statements
  }
}
```

Example of throws Keyword

In this example the method myMethod() is throwing two **checked exceptions** so we have declared these exceptions in the method signature using **throws** Keyword. If we do not declare these exceptions then the program will throw a compilation error.

```
import java.io.*;
class ThrowExample {
  void  myMethod(int num) throws IOException, ClassNotFoundException{
     if(num==1)
        throw  new  IOException("IOException Occurred");
     else
        throw  new ClassNotFoundException("ClassNotFoundException");
  }
}
public class Example1{
  public static void main(String  args[ ]){
   try{
     ThrowExample  obj=new  ThrowExample();
     obj.myMethod(1);
   }catch(Exception ex){
     System.out.println(ex);
    }
  }
}
```

Output:

```
java.io.IOException: IOException Occurred
```

Example 1: Exception propagation using throws keyword

As you can see that we have an exception occurred in method1 which has been handled in the chain-calling method method3(). This example shows how exception propagation works.

```
class Example1{
  void method1() throws ArithmeticException{
          throw new ArithmeticException("Calculation error");
  }
  void method2() throws ArithmeticException{
          method1();
  }
  void method3(){
          try{
            method2();
          }
          catch(ArithmeticException e){
            System.out.println("ArithmeticException handled");}
  }
  public static void main(String args[]){
          Example1 obj=new Example1();
          obj.method3();
          System.out.println("End Of Program");  }  }
```

Output:

```
ArithmeticException handled
End Of Program
```

References:

1-"java How to program" , H.M.Deitel „ Sixth edition ,2006

2- **https://www.tutorialspoint.com/javaexamples/java_exceptions.htm**