**Exception Handling**

Consider the following Java program which divides two integers.

import java.util.Scanner;

class Division

{

public static void main(String[] args)

{

int a, b, result;

Scanner input = new Scanner(System.in);

System.out.println("Input two integers");

a = input.nextInt();

b = input.nextInt();

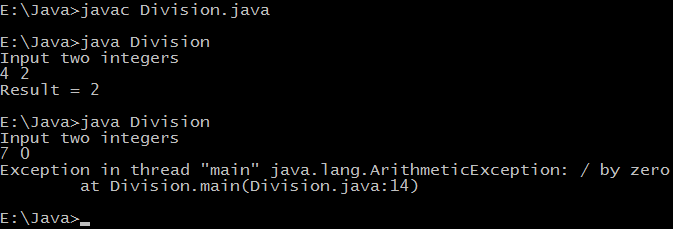
result = a / b;

System.out.println("Result = " + result);

}

}

Output:



* In the second case we are dividing 'a' by zero which isn't allowed in mathematics, so a run time error will occur i.e. an exception will occur.
* Java provides a powerful way to handle such exceptions which is known as exception handling. In it we write vulnerable code i.e. code which can throw exception in a separate block called as **try** block and exception handling code in another block called **catch** block.

**Java exception handling example**

**1)** Java program which divides two integers.

import java.util.Scanner;

class Division

{

public static void main(String[] args)

{

int a, b, result;

Scanner input = new Scanner(System.in);

System.out.println("Input two integers");

a = input.nextInt();

b = input.nextInt();

// try block

try

{

result = a / b;

System.out.println("Result = " + result);

}

// catch block

catch (ArithmeticException e)

{

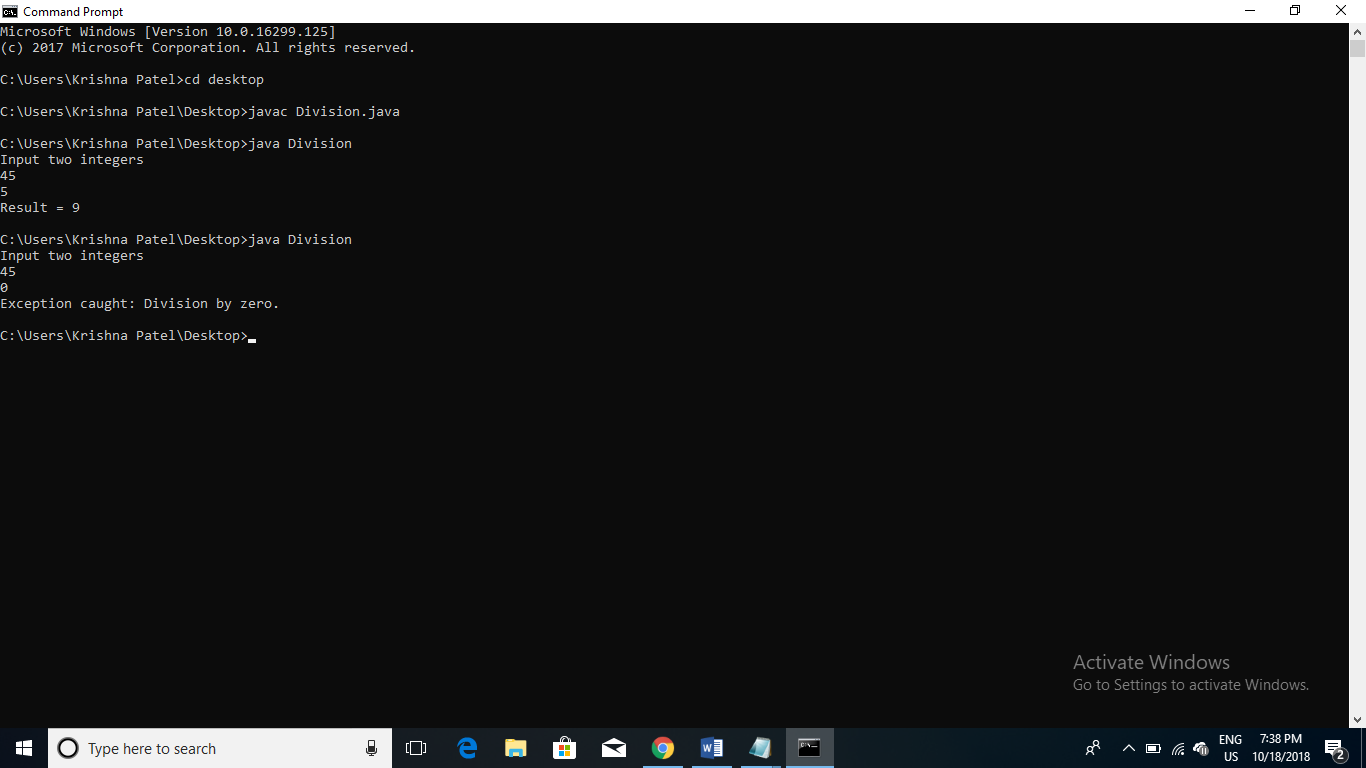
System.out.println("Exception caught: Division by zero.");

}

}

}

**OUTPUT:**



**Catching multiple exceptions**

**Example 1:**

//Program of Multiple catch

class MultipleCatch1{

public static void main(String args[]){

try

{

int arr[]=new int[7];

arr[4]=30/0;

System.out.println("Last Statement of try block");

}

catch(ArithmeticException e){

System.out.println("You should not divide a number by zero");

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("Accessing array elements outside of the limit");

}

catch(Exception e){

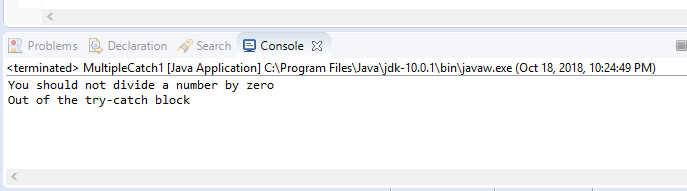
System.out.println("Some Other Exception");

}

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

}

}



**Example 2:**

//Program of Multiple catch

class MultipleCatch2{

public static void main(String args[]){

try{

int arr[]=new int[7];

arr[10]=10/5;

System.out.println("Last Statement of try block");

}

catch(ArithmeticException e){

System.out.println("You should not divide a number by zero");

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("Accessing array elements outside of the limit");

}

catch(Exception e){

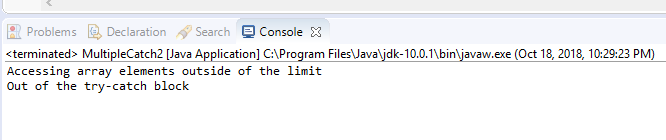
System.out.println("Some Other Exception");

}

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

}

}



**Example 3:**

//Program of Multiple catch

class MultipleCatch3{

public static void main(String args[]){

try{

int arr[]=new int[7];

arr[10]=10/5;

System.out.println("Last Statement of try block");

}

catch(Exception e){ // Exception is handled here

System.out.println("Some Other Exception");

}

catch(ArithmeticException e){

System.out.println("You should not divide a number by zero");

}

catch(ArrayIndexOutOfBoundsException e){

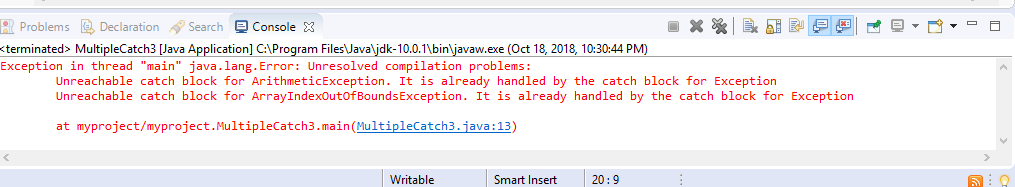
System.out.println("Accessing array elements outside of the limit");

}

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

}

}



**This program generates an error because the exception was already handled by the Catch Block.**

**Nesting of Try and Catch Block**

**Example 1:**

//Program of multiple try and catch

class NestingDemo1{

public static void main(String args[]){

//main try-block

try{

//try-block2

try{

//try-block3

try{

int arr[]= {1,2,3,4};

/\* I'm trying to display the value of

\* an element which doesn't exist. The

\* code should throw an exception

\*/

System.out.println(arr[10]);

}catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block3");

}

}

catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block2");

}

}

catch(ArithmeticException e3){

System.out.print("Arithmetic Exception");

System.out.println(" handled in main try-block");

}

catch(ArrayIndexOutOfBoundsException e4){

System.out.print("ArrayIndexOutOfBoundsException");

System.out.println(" handled in main try-block");

}

catch(Exception e5){

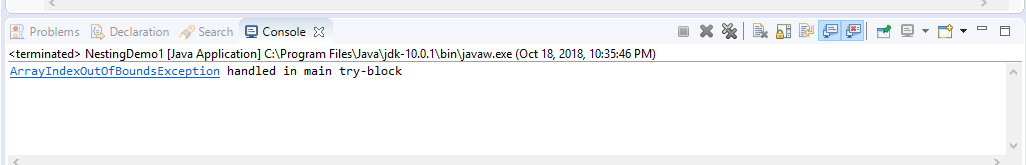
System.out.print("Exception");

System.out.println(" handled in main try-block");

}

}

}



**Example 2:**

//Program of multiple try and catch

class NestingDemo2{

public static void main(String args[]){

//Parent try block

try{

//Child try block1

try{

System.out.println("Inside block1");

int b =45/0;

System.out.println(b);

}

catch(ArithmeticException e1){

System.out.println("Exception: e1");

}

//Child try block2

try{

System.out.println("Inside block2");

int b =45/0;

System.out.println(b);

}

catch(ArrayIndexOutOfBoundsException e2){

System.out.println("Exception: e2");

}

System.out.println("Just other statement");

}

catch(ArithmeticException e3){

System.out.println("Arithmetic Exception");

System.out.println("Inside parent try catch block");

}

catch(ArrayIndexOutOfBoundsException e4){

System.out.println("ArrayIndexOutOfBoundsException");

System.out.println("Inside parent try catch block");

}

catch(Exception e5){

System.out.println("Exception");

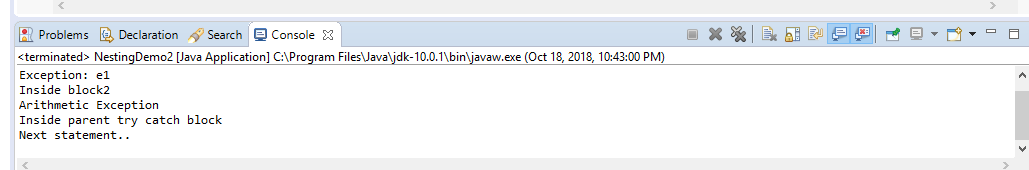
System.out.println("Inside parent try catch block");

}

System.out.println("Next statement..");

}

}



**Use of Finally Block**

**Sample**

//A Simple Example of finally block

public class Example {

public static void main(String[] args) {

try{

int num=121/0;

System.out.println(num);

}

catch(ArithmeticException e){

System.out.println("Number should not be divided by zero");

}

/\* Finally block will always execute

\* even if there is no exception in try block

\*/

finally{

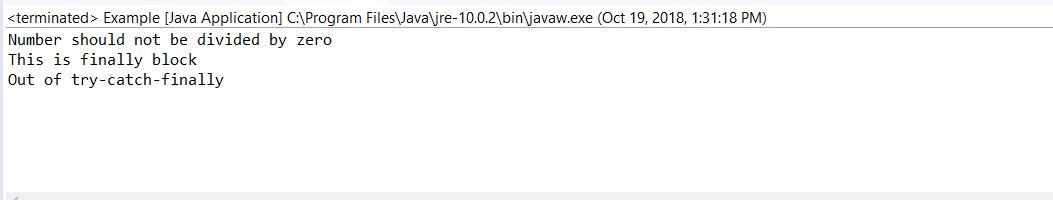
System.out.println("This is finally block");

}

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

}

}



**Example 1:**

//The working of finally block when no exception occurs in try block

class Example1{

public static void main(String args[]){

try{

System.out.println("First statement of try block");

int num=45/3;

System.out.println(num);

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("ArrayIndexOutOfBoundsException");

}

finally{

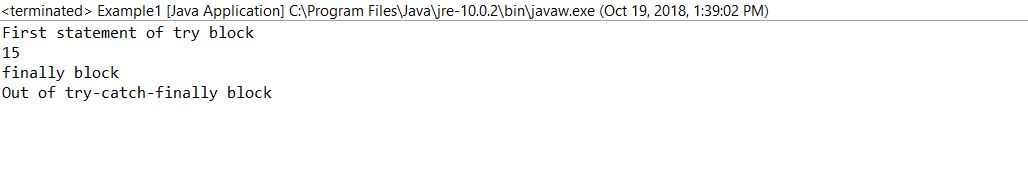
System.out.println("finally block");

}

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

}

}



**Example 2:**

//The working of finally block when an exception occurs in try block but is not handled in the catch block

public class Example2 {

public static void main(String[] args) {

try{

System.out.println("First statement of try block");

int num=45/0;

System.out.println(num);

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("ArrayIndexOutOfBoundsException");

}

finally{

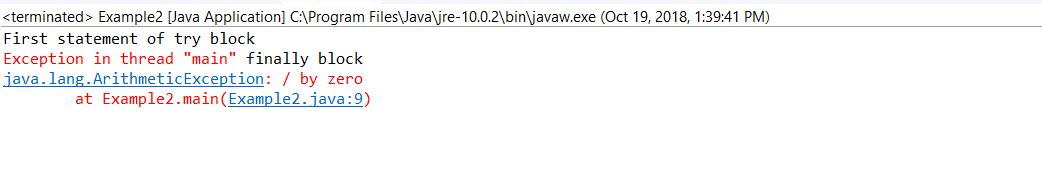
System.out.println("finally block");

}

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

}

}



**Example 3:**

//When exception occurs in try block and handled properly in catch block

public class Example3 {

public static void main(String[] args) {

try{

System.out.println("First statement of try block");

int num=45/0;

System.out.println(num);

}

catch(ArithmeticException e){

System.out.println("ArithmeticException");

}

finally{

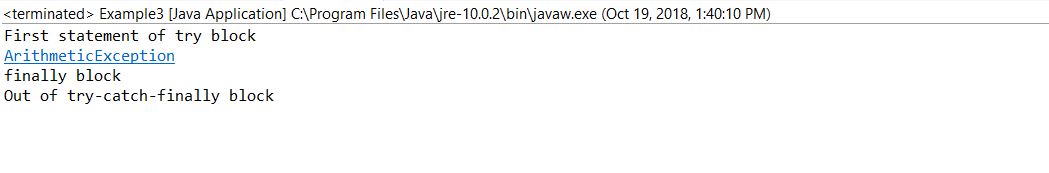
System.out.println("finally block");

}

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

}

}



**Example 4:**

//finally block and return statement

class JavaFinally{

public static void main(String args[])

{

System.out.println(JavaFinally.myMethod());

}

public static int myMethod()

{

try {

return 112;

}

finally {

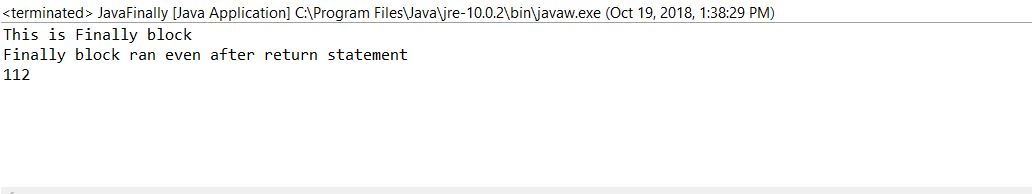
System.out.println("This is Finally block");

System.out.println("Finally block ran even after return statement");

}

}

}



**Flow Control in Try, Catch and Finally**

**Example 1:**

//Flow of control in try/catch blocks when exception doesn’t occur

public class Example4 {

public static void main(String[] args) {

int x = 10;

int y = 10;

try{

int num= x/y;

System.out.println("next-statement: Inside try block");

}

catch(Exception ex)

{

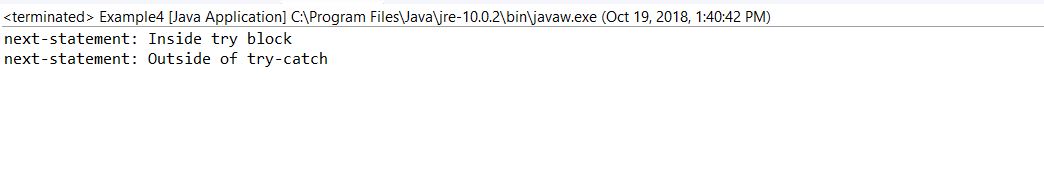
System.out.println("Exception");

}

System.out.println("next-statement: Outside of try-catch");

}

}



**Example 2:**

//Flow of control in try/catch blocks when exception occurs

public class Example5 {

public static void main(String[] args) {

int x = 0;

int y = 10;

try{

int num= y/x;

System.out.println("next-statement: Inside try block");

}

catch(Exception ex)

{

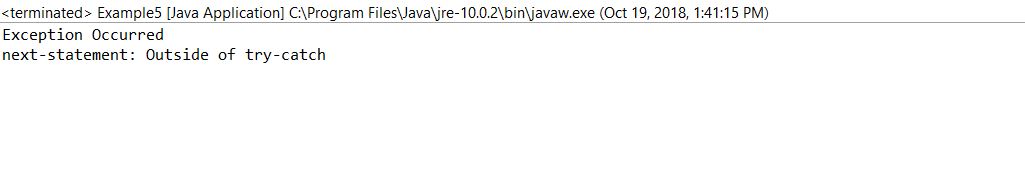
System.out.println("Exception Occurred");

}

System.out.println("next-statement: Outside of try-catch");

}

}



**The Throw Keyword**

The Syntax of Throw Keyword is:

“ throw new exception\_class("error message"); ”

**Example for 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[]){

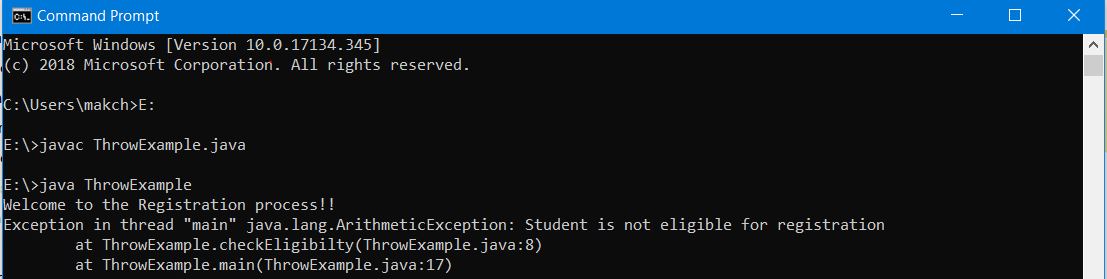
System.out.println("Welcome to the Registration process!!");

checkEligibilty(10, 39);

System.out.println("Have a nice day..");

}

}



**Example 2:**

/\* In this example the method myMethod() is throwing two checked except**ions** 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.\*/

**// This is program that demonstrates the use of Throw**

**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);**

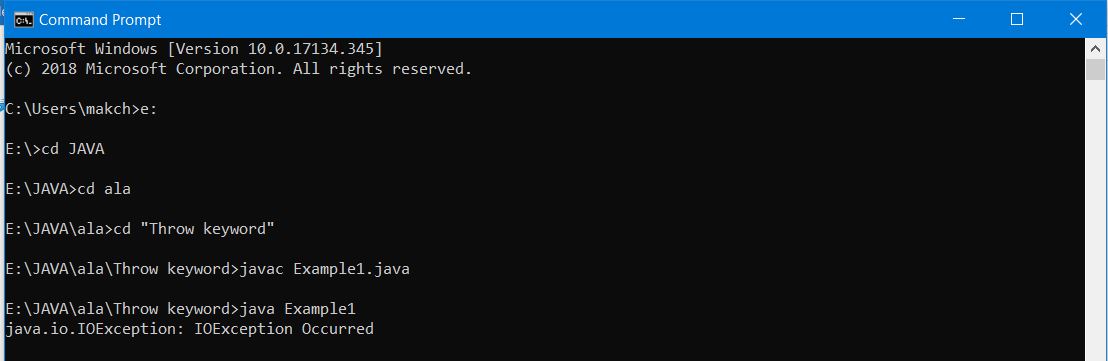
**}catch(Exception ex){**

**System.out.println(ex);**

**}**

**}**

**}**



**The Throws Clause**

**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[]){

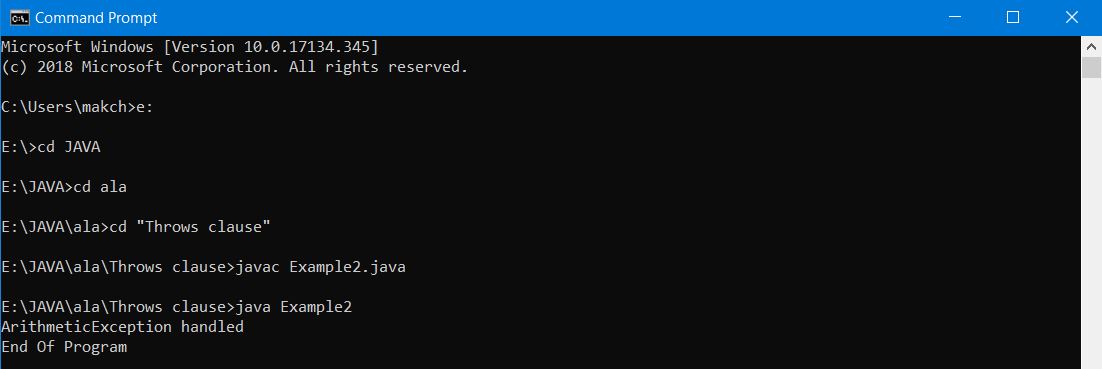
Example1 obj=new Example1();

obj.method3();

System.out.println("End Of Program");

}

}



**Example 2:** When you don’t handle exception and instead declare it at all the places

/\*The ideal way to use throws is by declaring the exceptions in method signature and handle the exceptions using try-catch in calling method. Lets see what happens when we declare the exception at both the places, in method signature as well as in calling method.\*/

class ExceptionExample{

void method()throws ArithmeticException{

throw new ArithmeticException("ArithmeticException Occurred");

}

}

class Example1{

public static void main(String args[])throws ArithmeticException{

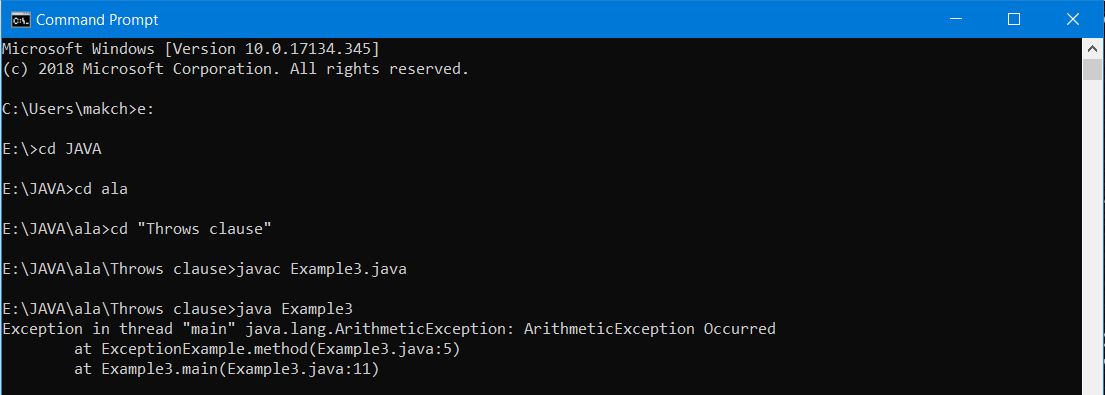
ExceptionExample obj=new ExceptionExample();

obj.method();

System.out.println("End Of Program");

}

}



**Example 3:**

/\*Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.\*/

import java.io.IOException;

class Testthrows1{

  void m()throws IOException{

    throw new IOException("device error");//checked exception

  }

  void n()throws IOException{

    m();

  }

  void p(){

   try{

    n();

   }catch(Exception e){System.out.println("exception handled");}

  }

  public static void main(String args[]){

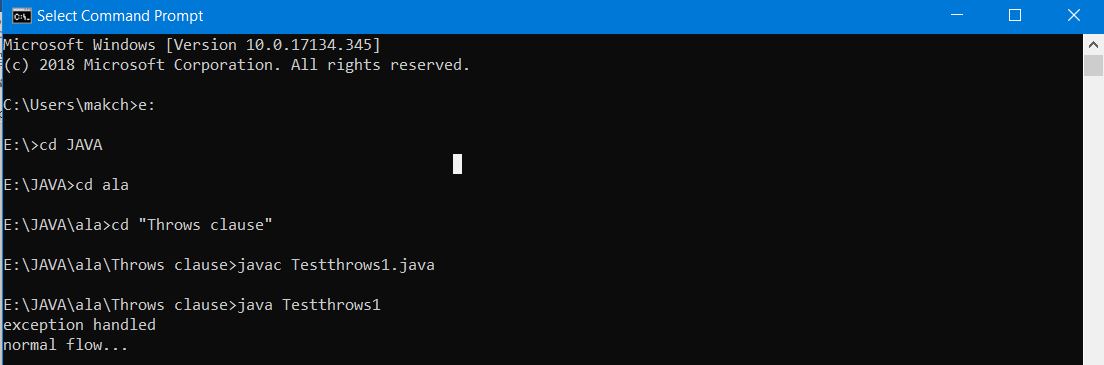
   Testthrows1 obj=new Testthrows1();

   obj.p();

   System.out.println("normal flow...");

  }

}



**Custom Exception or User Defined Exception**

**Example 1:**

/\* This is my Exception class, I have named it MyException. you can give any name, just remember that it should extend Exception class \*/

class MyException extends Exception{

String str1;

/\* Constructor of custom exception class

\* here I am copying the message that we are passing while

\* throwing the exception to a string and then displaying

\* that string along with the message.

\*/

MyException(String str2) {

str1=str2;

}

public String toString(){

return ("MyException Occurred: "+str1) ;

}

}

class Example1{

public static void main(String args[]){

try{

System.out.println("Starting of try block");

// I'm throwing the custom exception using throw

throw new MyException("This is My error Message");

}

catch(MyException exp){

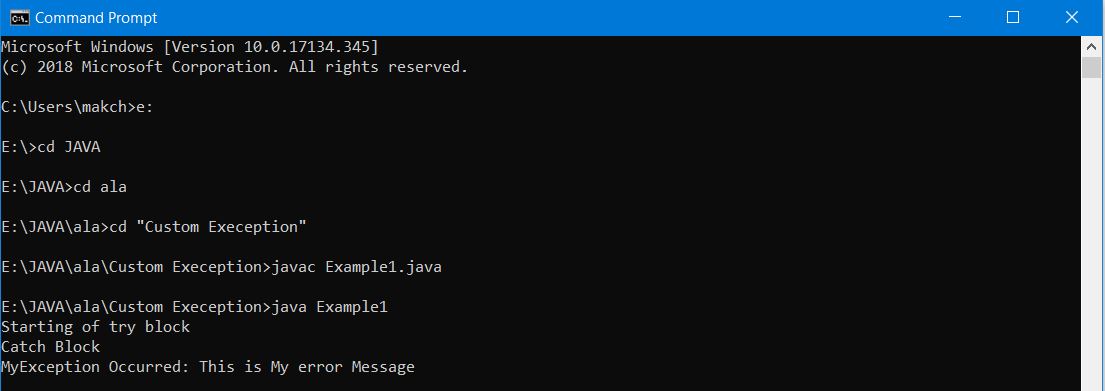
System.out.println("Catch Block") ;

System.out.println(exp) ;

}

}

}



**Example 2:**

/\* A simple example of using a customised exception by extending Exception \*/

import java.io.\*;

class CustomException extends Exception //A Class that represents custom exception

{

    public CustomException()

    {

        super("Number is negative"); // Call constructor of parent Exception

    }

}

public class CustomExceptionExample // A Class that uses above MyException

{

    public static void main(String args[])

    {

        int n = Integer.parseInt(args[0]);

        try

        {

            if(n<0)

            {

                throw new CustomException();

            }

        }

        catch(CustomException ce)

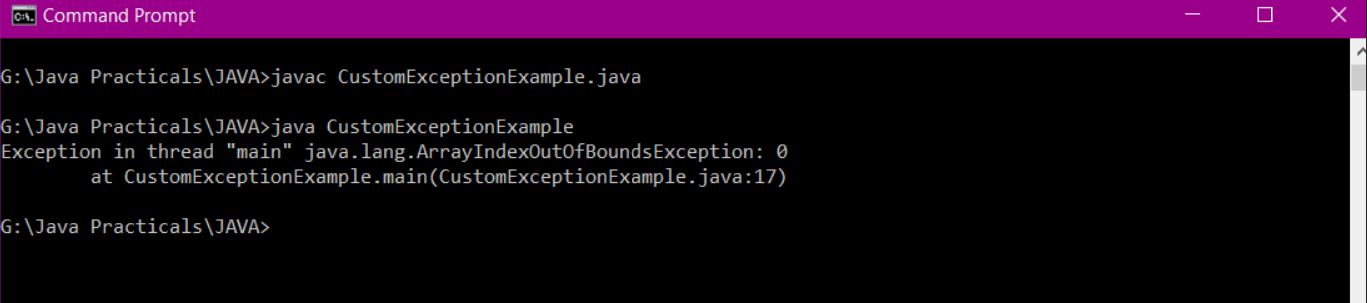
        {

            System.out.println("Custom Exception Occurs");

        }

    }

}



**Checked and Unchecked Exception**

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1) **FileNotFound Exception**   |  | | --- | | //Java program to demonstrate FileNotFoundException  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](file:///E:\file.txt)");    FileReader fr = new FileReader(file);           }  catch (FileNotFoundException e)  {              System.out.println("File does not exist");           }       }  } | |  | | |  | | --- | |  | |  | | |
|  |

**2) SQL Exception**

import java.sql.\*;

public class SQLExceptionExample // JDBC driver name and database URL

{

static final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

static final String DB\_URL = "jdbc:mysql://localhost/EMP";

// Database credentials

static final String USER = "username";

static final String PASS = "password";

public static void main(String[] args) {

Connection conn = null;

try{

Class.forName("com.mysql.jdbc.Driver"); //Register JDBC driver

System.out.println("Connecting to database..."); //Open a connection

conn = DriverManager.getConnection(DB\_URL,USER,PASS);

System.out.println("Creating statement..."); //Execute a query

Statement stmt = conn.createStatement();

String sql;

sql = "SELECT id, first, last, age FROM Employees";

ResultSet rs = stmt.executeQuery(sql);

while(rs.next()) //Extract data from result set

{

int id = rs.getInt("id"); //Retrieve by column name

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

System.out.print("ID: " + id); //Display values

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

}

rs.close(); //Clean-up environment

stmt.close();

conn.close();

}

catch(SQLException se)

{

se.printStackTrace(); //Handle errors for JDBC

}

catch(Exception e)

{

e.printStackTrace(); //Handle errors for Class.forName

}

finally //finally block used to close resources

{

try

{

if(conn!=null)

conn.close();

}

catch(SQLException se)

{

se.printStackTrace();

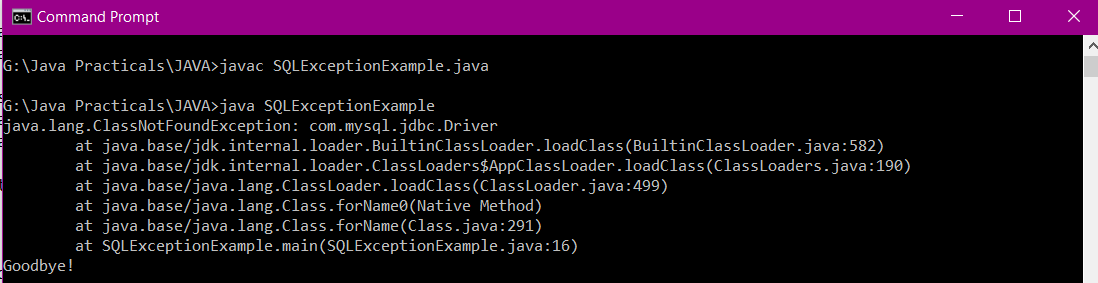
}//end finally try

}//end try

System.out.println("Goodbye!");

}//end main

}//end JDBCExample



**3) IO Exception**

**Example 1:**

//Example of IOException using throws method

import java.io.\*;

class IOExceptionThrows

{

public static void main(String args[]) throws IOException

{

FileInputStream fis = null; //This constructor throws FileNotFoundException which is a checked exception

fis = new FileInputStream("G:/myfile.txt"); //change location according to your file

int k;

while(( k = fis.read() ) != -1)

{

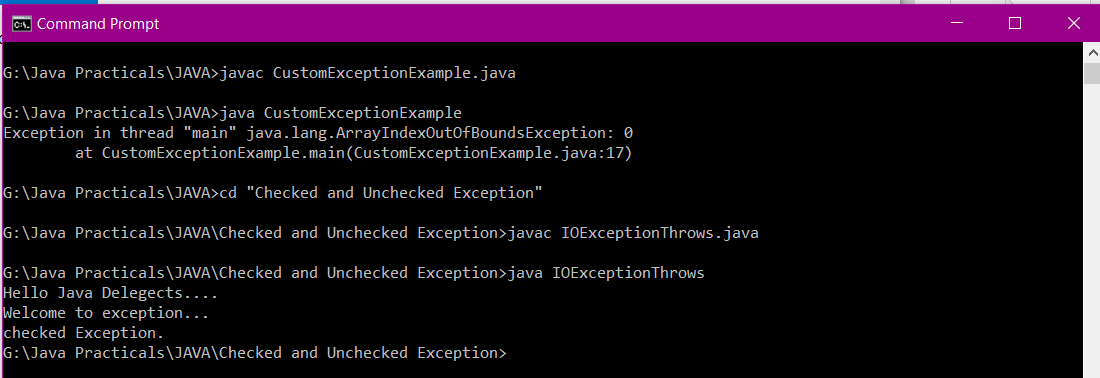
System.out.print((char)k);

}

fis.close();

}

}



**Example 2:**

//Example of IOException using try-catch

import java.io.\*;

class IOExceptionTryCatch

{

public static void main(String args[])

{

FileInputStream fis = null;

try

{

fis = new FileInputStream("G:/myfile.txt"); //change location according to your file

}

catch(FileNotFoundException fnfe)

{

System.out.println("The specified file is not " + "present at the given path");

}

int k;

try

{

while(( k = fis.read() ) != -1)

{

System.out.print((char)k);

}

fis.close();

}

catch(IOException ioe)

{

System.out.println("I/O error occurred: "+ioe);

}

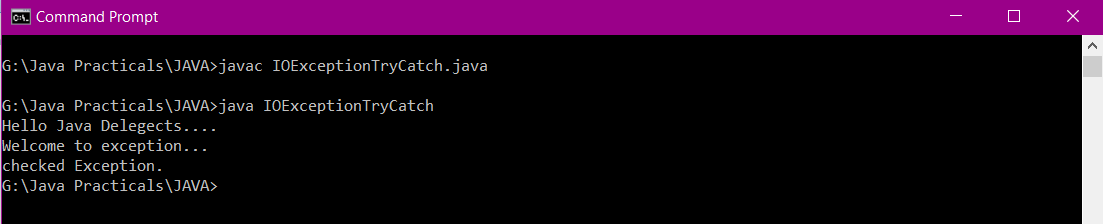
}

}

This file’s path need to be specified in the path.



Output:



**Unchecked Exceptions**

**1) Arithmetic exception**

// Java program to demonstrate ArithmeticException

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)

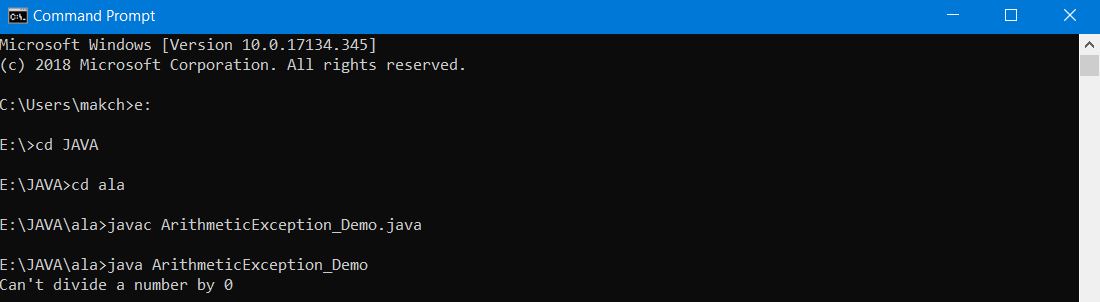
{

            System.out.println ("Can't divide a number by 0");

        }

    }

}



**2) NullPointer Exception**

//Java program to demonstrate NullPointerException

class NullPointer\_Demo

{

    public static void main(String args[])

    {

        try

{

             String a = null; //null value

             System.out.println(a.charAt(0));

        }

catch(NullPointerException e)

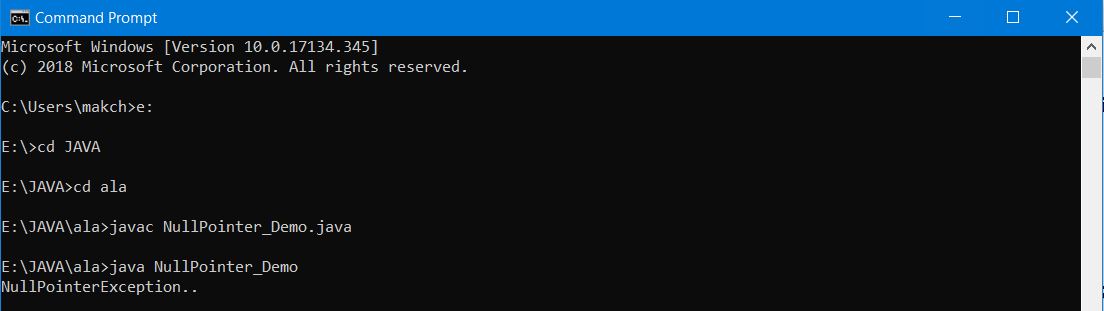
{

            System.out.println("NullPointerException..");

         }

    }

}



**3) ArrayIndexOutOfBounds Exception**

class ArrayIndexOutOfBoundsExceptionExample

{

    public static void main(String args[])

    {

        try

        {

             int arr[] ={1,2,3,4,5};

             System.out.println(arr[7]);

        }

     catch(ArrayIndexOutOfBoundsException e)

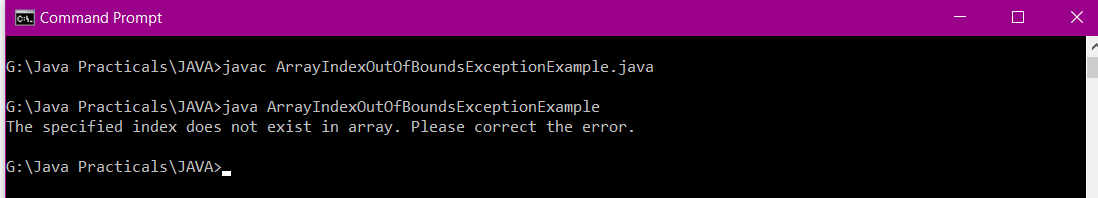
        {

             System.out.println("The specified index does not exist " + "in array. Please correct the error.");

        }

 }

}



**4)** **IllegalArgumentException**

import java.io.File;

public class IllegalArgumentExceptionExample

{

    public static String createRelativePath(String parent, String filename)

    {

        if(parent == null)

            throw new IllegalArgumentException("The parent path cannot be null!");

        if(filename == null)

            throw new IllegalArgumentException("The filename cannot be null!");

        return parent + File.separator + filename;

    }

    public static void main(String[] args)

    {

        // The following command will be successfully executed.

        System.out.println(IllegalArgumentExceptionExample.createRelativePath("dir1", "file1"));

        System.out.println();

        // The following command throws an IllegalArgumentException.

        System.out.println(IllegalArgumentExceptionExample.createRelativePath(null, "file1"));

    }

}

