

class Example1 {

public static void main(String args[]) {

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.");

}

}

UserException:

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) ;

}

}

}

1. User-defined exception must extend Exception class.  
   2. The exception is thrown using throw keyword.

class InvalidProductException extends Exception

{

public InvalidProductException(String s)

{

// Call constructor of parent Exception

super(s);

}

}

public class Example1

{

void productCheck(int weight) throws InvalidProductException{

if(weight<100){

throw new InvalidProductException("Product Invalid");

}

}

public static void main(String args[])

{

Example1 obj = new Example1();

try

{

obj.productCheck(60);

}

catch (InvalidProductException ex)

{

System.out.println("Caught the exception");

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

}

}

}

1. throws clause is used to declare an exception and throw keyword is used to throw an exception explicitly.
2. If we see syntax wise then throw is followed by an instance variable and throws is followed by exception class names.
3. The keyword throw is used inside method body to invoke an exception and throws clause is used in method declaration (signature).

For example

**throw**

throw new Exception("You have some exception")

throw new IOException("Connection failed!!")

**throws**

public int myMethod() throws IOException, ArithmeticException, NullPointerException {}

1. You cannot declare multiple exceptions with throw. You can declare multiple exception e.g. public void method()throws IOException,SQLException.
2. checked exceptions can not be propagated with throw only because it is explicitly used to throw an particular exception. checked exception can be propagated with throws.

**Exception propagation:** An exception propagates from method to method, up the call stack, until it's caught. So if a() calls b(), which calls c(), which calls d(), and if d() throws an exception, the exception will propagate from d to c to b to a, unless one of these methods catches the exception.[what is exception propagation?](https://stackoverflow.com/questions/10633664/what-is-exception-propagation)

**Example 1: Arithmetic exception**

Class: Java.lang.ArithmeticException  
This is a built-in-class present in java.lang package. This exception occurs when an integer is divided by zero.

class Example1

{

public static void main(String args[])

{

try{

int num1=30, num2=0;

int output=num1/num2;

System.out.println ("Result: "+output);

}

catch(ArithmeticException e){

System.out.println ("You Shouldn't divide a number by zero");

}

}

}

**Output of above program:**

You Shouldn't divide a number by zero

**Explanation:** In the above example I’ve divided an integer by a zero and because of this ArithmeticException is thrown.

**Example 2: ArrayIndexOutOfBounds Exception**

Class: Java.lang.ArrayIndexOutOfBoundsException  
This exception occurs when you try to access the array index which does not exist. For example, If array is having only 5 elements and we are trying to display 7th element then it would throw this exception.

class ExceptionDemo2

{

public static void main(String args[])

{

try{

int a[]=new int[10];

//Array has only 10 elements

a[11] = 9;

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println ("ArrayIndexOutOfBounds");

}

}

}

**Output:**

ArrayIndexOutOfBounds

In the above example the array is initialized to store only 10 elements indexes 0 to 9. Since we are try to access element of index 11, the program is throwing this exception.

**Example 3: NumberFormat Exception**

Class: Java.lang.NumberFormatException

This exception occurs when a string is parsed to any numeric variable.

For example, the statement int num=Integer.parseInt ("XYZ"); would throw NumberFormatExceptionbecause String “XYZ” cannot be parsed to int.

class ExceptionDemo3

{

public static void main(String args[])

{

try{

int num=Integer.parseInt ("XYZ") ;

System.out.println(num);

}catch(NumberFormatException e){

System.out.println("Number format exception occurred");

}

}

}

Output:

Number format exception occurred

**Example 4: StringIndexOutOfBound Exception**

Class: Java.lang.StringIndexOutOfBoundsException

* An object of this class gets created whenever an index is invoked of a string, which is not in the range.
* Each character of a string object is stored in a particular index starting from 0.
* To get a character present in a particular index of a string we can use a [method charAt(int)](https://beginnersbook.com/2013/12/java-string-charat-method-example/) of [java.lang.String](https://beginnersbook.com/2013/12/java-strings/" \t "_blank) where int argument is the index.

E.g.

class ExceptionDemo4

{

public static void main(String args[])

{

try{

String str="beginnersbook";

System.out.println(str.length());;

char c = str.charAt(0);

c = str.charAt(40);

System.out.println(c);

}catch(StringIndexOutOfBoundsException e){

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

}

}

}

Output:

13

StringIndexOutOfBoundsException!!

Exception occurred because the referenced index was not present in the String.

**Example 5: NullPointer Exception**

Class: Java.lang.NullPointer Exception  
An object of this class gets created whenever a member is invoked with a “null” object.

class Exception2

{

public static void main(String args[])

{

try{

String str=null;

System.out.println (str.length());

}

catch(NullPointerException e){

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

}

}

}

Output:

NullPointerException..

Here, length() is the function, which should be used on an object. However in the above example String object str is null so it is not an object due to which NullPointerException occurred.

https://howtodoinjava.com/best-practices/java-exception-handling-best-practices/

Example of Unreachable Catch block:

class Excep

{

public static void main(String[] args)

{

try

{

int arr[]={1,2};

arr[2]=3/0;

}

catch(Exception e) //This block handles all Exception

{

System.out.println("Generic exception");

}

catch(ArrayIndexOutOfBoundsException e) //This block is unreachable

{

System.out.println("array index out of bound exception");

}

}

}

Output:

Generic exception

2)

class Excep

{

public static void main(String[] args)

{

try

{

int arr[]={5,0,1,2};

try

{

int x=arr[3]/arr[1];

}

catch(ArithmeticException ae)

{

System.out.println("divide by zero");

}

arr[4]=3;

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("array index out of bound exception");

}

}

}

divide by zero

array index out of bound exception

1. If you do not explicitly use the try catch blocks in your program, java will provide a default exception handler, which will print the exception details on the terminal, whenever exception occurs.
2. Super class **Throwable** overrides **toString()** function, to display error message in form of string.
3. While using multiple catch block, always make sure that sub-classes of Exception class comes before any of their super classes. Else you will get compile time error.
4. In nested try catch, the inner try block uses its own catch block as well as catch block of the outer try, if required.
5. Only the object of Throwable class or its subclasses can be thrown.