**Command line arguments**

|  |
| --- |
| class commandline{public static void main(String args[]){int i,a=args.length;System.out.println("no.of arguments entered :"+a);for(i=0;i<a;i++){System.out.println("args["+i+"] :"+args[i]);}}} |

**Output**

|  |
| --- |
| D:\javaprgs>java commandline 5 6 7 8 9no.of arguments entered :5args[0] :5args[1] :6args[2] :7args[3] :8args[4] :9 |

**https://beginnersbook.com/2013/04/exception-handling-examples/**

**Exception handling**

**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.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

* A user has entered an invalid data.
* A file that needs to be opened cannot be found.
* A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

**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 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.

## 1)it allows the user to fix the error

## 2) It prevents the program from automatically terminating

## 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 occurs 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.

Java exception handling is managed via five keywords: **try, catch, throw, throws, and finally**.

Program statements that are to be monitored are entered within **try** block. If an exception occurs within the **try** block, it is thrown. The user code can catch this exception(using **catch**) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword **throw**. Any exception that is thrown out of a method must be specified as such by **throws** clause. Any code that absolutely must be executed before a method returns is put in a **finally** block.

General format of an exception-handling block

try

{

//block of code to monitor for errors

}

catch(Exception Type1 exOb)

{

// exception handler for ExceptionType1

}

catch(Exception Type2 exOb)

{

// exception handler for ExceptionType2

}

finally

{

//block of code to be executed before try block ends.

}

Rules

* We can’t have catch or finally clause without a try statement.
* A try statement should have either catch block or finally block, it can have both blocks.
* We can’t write any code between try-catch-finally block.
* We can have multiple catch blocks with a single try statement.
* try-catch blocks can be nested similar to if-else statements.
* We can have only one finally block with a try-catch statement.
* Java catch block is used to handle the Exception. It must be used after the try block only.
* You can use multiple catch block with a single try.

Exception types

All exception types are subclasses of the built-in class **Throwable. Throwable** is at the top of the exception class hierarchy

**Hierarchy of Java Exception classes**



Types of Exception

1. Checked Exception
2. Unchecked Exception
3. Error

**Difference between checked and unchecked exceptions**

**1) Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

**2) Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

**3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Common scenarios where exceptions may occur**

There are given some scenarios where unchecked exceptions can occur. They are as follows:

**1) Scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

1. int a=50/0;//ArithmeticException

**2) Scenario where NullPointerException occurs**

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

1. String s=null;
2. System.out.println(s.length());//NullPointerException

**3) Scenario where NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose a string variable that have characters, converting this variable into digit will occur NumberFormatException.

1. String s="abc";
2. int i=Integer.parseInt(s);//NumberFormatException

**4) Scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

## Java catch block

## Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

1. public class Testtrycatch1{
2. public static void main(String args[]){
3. int data=50/0;//may throw exception
4. System.out.println("rest of the code...");
5. }
6. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testtrycatch1)

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

## Solution by exception handling

Let's see the solution of above problem by java try-catch block.

1. public class Testtrycatch2{
2. public static void main(String args[]){
3. try{
4. int data=50/0;
5. }catch(ArithmeticException e){System.out.println(e);}
6. System.out.println("rest of the code...");
7. }
8. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testtrycatch2)

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

**Internal working of java try-catch block**



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

* Prints out exception description.
* Prints the stack trace (Hierarchy of methods where the exception occurred).
* Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Example program : where exception is not handled.

|  |
| --- |
| public class unchecked\_demo{public static void main(String args[]){int num[]={1,2,3,4};System.out.println(num[5]);}} |

Output

|  |
| --- |
| D:\javaprgs>javac unchecked\_demo.javaD:\javaprgs>java unchecked\_demoException in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5 at unchecked\_demo.main(unchecked\_demo.java:6) |

The above program when the exception is handled.

|  |
| --- |
| public class unchecked\_demo1{public static void main(String args[]){try{int num[]={1,2,3,4};System.out.println(num[5]);}catch(ArrayIndexOutOfBoundsException e){System.out.println("Exception thrown : "+e);}System.out.println("out of block");}} |

Output

|  |
| --- |
| D:\javaprgs>javac unchecked\_demo1.javaD:\javaprgs>java unchecked\_demo1Exception thrown : java.lang.ArrayIndexOutOfBoundsException: 5out of block |

Example 2:

|  |
| --- |
| class exceptiondemo1{public static void main(String args[]){int d,a;try{d=0;a=42/d;System.out.println("This will not be printed");}catch(ArithmeticException e){System.out.println("this is user message");System.out.println("Division by zero ");System.out.println("This is jvm error message "+e);}System.out.println("After catch statment");}} |

Output

|  |
| --- |
| D:\javaprgs>java exceptiondemo1this is user messageDivision by zeroThis is jvm error message java.lang.ArithmeticException: / by zeroAfter catch statement |

In the above program the println statement inside try block is not executed. Because the error occurs in a=42/d; immediately the error is thrown and it is caught in catch block. It checks whether the respective error is handled or not and do accordingly.

Multiple catches

In some cases , more than one exception could be raised by a single piece of code

|  |
| --- |
| class multiexception1{public static void main(String args[]){try{int a[]=new int[5];a[5]=30/0;}catch(ArithmeticException e){System.out.println("task1 is completed");}catch(ArrayIndexOutOfBoundsException e){System.out.println("task2 is completed");}catch(Exception e){System.out.println("common task is completed");}System.out.println("rest of the code");}} |

Output

|  |
| --- |
| D:\javaprgs>javac multiexception1.javaD:\javaprgs>java multiexception1task1 is completedrest of the code |

#### Rule: At a time only one Exception is occured and at a time only one catch block is executed.

#### Rule: All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .

Example where the exc eption order is changed

|  |
| --- |
| class multiexception2{public static void main(String args[]){try{int a[]=new int[5];a[5]=30/0;}catch(Exception e){System.out.println("common task is completed");}catch(ArithmeticException e){System.out.println("task1 is completed");}catch(ArrayIndexOutOfBoundsException e){System.out.println("task2 is completed");}System.out.println("rest of the code");}} |

Output

|  |
| --- |
| D:\javaprgs>javac multiexception2.javamultiexception2.java:15: error: exception ArithmeticException has already been caughtcatch(ArithmeticException e)^multiexception2.java:20: error: exception ArrayIndexOutOfBoundsException has already been caughtcatch(ArrayIndexOutOfBoundsException e)^2 errors |

Example for multiple catch

|  |
| --- |
| class multiexception3{public static void main(String args[]){try{int a=args.length;System.out.println("a= "+a);int b=42/a;int c[]={1};c[42]=99;}catch(ArithmeticException e){System.out.println("Divide by zero : "+e);}catch(ArrayIndexOutOfBoundsException e){System.out.println("Array index is violated - out of boundry "+e);}System.out.println("this is after try/catch block");}} |

Output

|  |
| --- |
| D:\javaprgs>java multiexception3a= 0Divide by zero : java.lang.ArithmeticException: / by zerothis is after try/catch block |

Output

|  |
| --- |
| D:\javaprgs>java multiexception3 5 6 7a= 3Array index is violated - out of boundry java.lang.ArrayIndexOutOfBoundsException: 42this is after try/catch block |

Nested try block -exceptions

When a [try catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) is present in another try block then it is called the nested try catch block. Each time a try block does not have a catch handler for a particular [exception](https://beginnersbook.com/2013/04/java-exception-handling/), then the catch blocks of parent try block are inspected for that exception, if match is found then that catch block executes.

If neither current catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don’t handle exception.

**Syntax of Nested try Catch**

....

//Main try block

try {

 statement 1;

 statement 2;

 //try-catch block inside another try block

 try {

 statement 3;

 statement 4;

 //try-catch block inside nested try block

 try {

 statement 5;

 statement 6;

 }

 catch(Exception e2) {

 //Exception Message

 }

 }

 catch(Exception e1) {

 //Exception Message

 }

}

//Catch of Main(parent) try block

catch(Exception e3) {

 //Exception Message

}

....

Example

|  |
| --- |
| class nestingdemo{public static void main(String args[]){//main try block1 try { try //try block2 { try //try block3 { int arr[]={1,2,3,4}; System.out.println(arr[10]); }  catch(ArithmeticException e3) { System.out.println("Artihmetic Exception"); System.out.println("handled in try-block3"+e3); } } catch(ArithmeticException e2) { System.out.println("Artihmetic Exception"); System.out.println("handled in try-block2 :" +e2); } } catch(ArithmeticException e1) { System.out.println("Artihmetic Exception"); System.out.println("handled in try-block1 "+e1); } catch(ArrayIndexOutOfBoundsException e4) { System.out.println("ArrayIndexOutOfBoundsException"); System.out.println("handled in try-block1 "+ e4); } catch(Exception e5) { System.out.println("Exception"); System.out.println("handled in try-block1 "+e5); }} //end of public static main} //end of nestingdemo.java |

Output

|  |
| --- |
| E:\javaprgs>javac nestingdemo.javaE:\javaprgs>java nestingdemoArrayIndexOutOfBoundsExceptionhandled in try-block1 java.lang.ArrayIndexOutOfBoundsException: 10 |

Example2

|  |
| --- |
| class nestingdemo2{public static void main(String args[]){//main try block1 try { try //try block2 { System.out.println("Inside try block2"); int b=45/0; System.out.println(b); } catch(ArithmeticException e1) { System.out.println("Exception :"+e1); }  try //try block3 { int arr[]={1,2,3,4}; System.out.println(arr[10]); }  catch(ArrayIndexOutOfBoundsException e2) { System.out.println("ArrayIndexOutOfBoundsException"); System.out.println("handled in try-block3"+e2); }   System.out.println("just another statement in main try block"); } catch(ArithmeticException e4) { System.out.println("Artihmetic Exception"); System.out.println("handled in try-block1 "+e4); } catch(ArrayIndexOutOfBoundsException e5) { System.out.println("ArrayIndexOutOfBoundsException"); System.out.println("handled in try-block1 "+ e5); } catch(Exception e6) { System.out.println("Exception"); System.out.println("handled in try-block1 "+e6); } System.out.println("just next statement in main block");}} |

Output

|  |
| --- |
| E:\javaprgs>javac nestingdemo2.javaE:\javaprgs>java nestingdemo2Inside try block2Exception :java.lang.ArithmeticException: / by zeroArrayIndexOutOfBoundsExceptionhandled in try-block3java.lang.ArrayIndexOutOfBoundsException: 10just another statement in main try blockjust next statement in main block |

Example3

|  |
| --- |
| class nestingdemo2{public static void main(String args[]){//main try block1 try { try //try block2 { System.out.println("Inside try block2"); int b=45/0; System.out.println(b); } catch(ArithmeticException e1) { System.out.println("Exception :"+e1); }  try //try block3 { int arr[]={1,2,3,4}; System.out.println(arr[10]); }  catch(ArithmeticException e2) { System.out.println("ArithmeticEception"); System.out.println("handled in try-block3"+e2); }   System.out.println("just another statment in main try block"); } catch(ArithmeticException e4) { System.out.println("Artihmetic Exception"); System.out.println("handled in try-block1 "+e4); } catch(ArrayIndexOutOfBoundsException e5) { System.out.println("ArrayIndexOutOfBoundsException"); System.out.println("handled in try-block1 "+ e5); } catch(Exception e6) { System.out.println("Exception"); System.out.println("handled in try-block1 "+e6); } System.out.println("just next statment in main block");}} |

Output

|  |
| --- |
| E:\javaprgs>javac nestingdemo2.javaE:\javaprgs>java nestingdemo2Inside try block2Exception :java.lang.ArithmeticException: / by zeroArrayIndexOutOfBoundsExceptionhandled in try-block1 java.lang.ArrayIndexOutOfBoundsException: 10just next statment in main block |

The important point to note here is that whenever the child catch blocks are not handling any exception, the jumps to the parent catch blocks, if the exception is not handled there as well then the program will terminate abruptly showing system generated message.

# Java Finally block – Exception handling

A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

Syntax of finally block

try {

 //Statements that may cause an exception

}

catch {

 //Handling exception

}

finally {

 //Statements to be executed

}

Exampl1

|  |
| --- |
| **class finallyexception1****{****public static void main(String args[])****{****try****{****int num=12/0;****System.out.println(num);****}****catch(ArithmeticException e)****{****System.out.println("Number should not be divided by zero"+e);****}****finally****{****System.out.println("this is finally block");****}****System.out.println("this is in main program");****}****}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac finallyexception1.java****E:\javaprgs>java finallyexception1****Number should not be divided by zerojava.lang.ArithmeticException: / by zero****this is finally block****this is in main program** |

**Few Important points regarding finally block**

1. A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.

2. Finally block is optional, that a try-catch block is sufficient for [exception handling](https://beginnersbook.com/2013/04/java-exception-handling/), however if you place a finally block then it will always run after the execution of try block.

3. In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.

4. An exception in the finally block, behaves exactly like any other exception.

5. The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue.
Lets see an example to see how finally works when return statement is present in try block:

**Example**

|  |
| --- |
| **class finallyeg****{****public static int mymethod()****{****try****{****return 20;****}****finally****{****System.out.println("this is finally block");****}****}****}****class finallyexception2****{****public static void main(String args[])****{** **finallyeg obj=new finallyeg();** **System.out.println(obj.mymethod());****}****}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac finallyexception2.java****E:\javaprgs>java finallyexception2****this is finally block****20** |

**Cases when the finally block doesn’t execute**

The circumstances that prevent execution of the code in a finally block are:
– The death of a Thread
– Using of the System. exit() method.
– Due to an exception arising in the finally block.

**Finally and Close()**

**close()** statement is used to close all the open streams in a program. Its a good practice to use close() inside finally block. Since finally block executes even if exception occurs so you can be sure that all input and output streams are closed properly regardless of whether the exception occurs or not.

**Example 1:** The following example demonstrate the working of finally block when no exception occurs in try block

|  |
| --- |
| **class finallyexception4****{****public static void main(String args[])****{** **try** **{** **System.out.println("First statement of try block");** **int num=45/5;** **System.out.println(num);** **}** **catch(ArrayIndexOutOfBoundsException e)** **{** **System.out.println("exception is "+e);** **}**  **finally** **{** **System.out.println("finally block");** **}** **System.out.println("out of try-catch-finally");****}****}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac finallyexception4.java****E:\javaprgs>java finallyexception4****First statement of try block****9****finally block****out of try-catch-finally** |

**Example 2:** This example shows the working of finally block when an exception occurs in try block but is not handled in the catch block:

|  |
| --- |
| **class finallyexception5****{****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("exception is "+e);** **}**  **finally** **{** **System.out.println("finally block");** **}** **System.out.println("out of try-catch-finally");****}****}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac finallyexception5.java****E:\javaprgs>java finallyexception5****First statement of try block****finally block****Exception in thread "main" java.lang.ArithmeticException: / by zero** **at finallyexception5.main(finallyexception5.java:8)** |

As you can see that the system generated exception message is shown but before that the finally block successfully executed.

**Example 3**: When exception occurs in try block and handled properly in catch block

|  |
| --- |
| **class finallyexception6****{****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("exception is "+e);** **}**  **finally** **{** **System.out.println("finally block");** **}** **System.out.println("out of try-catch-finally");****}****}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac finallyexception6.java****E:\javaprgs>java finallyexception6****First statement of try block****exception is java.lang.ArithmeticException: / by zero****finally block****out of try-catch-finally** |

# How to throw exception in java with example

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.

|  |
| --- |
| **class throwexample****{****static void checkeligibility(int stage,int stweight)****{** **if(stage<12 && stweight<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 registration process");****checkeligibility(10,39);****System.out.println("welcome");****}****}** |

**Output**

|  |
| --- |
| **D:\javaprgs>javac throwexample.java****D:\javaprgs>java throwexample****welcome to registration process****Exception in thread "main" java.lang.ArithmeticException: Student is not eligibl****e for registration** **at throwexample.checkeligibility(throwexample.java:7)** **at throwexample.main(throwexample.java:21)** |

# Throws clause in java – Exception handling

As we know that there are two types of exception [checked and unchecked](https://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/). Checked exception (compile time) force you to handle them, if you don’t handle them then the program will not compile.
On the other hand unchecked exception (Runtime) doesn’t get checked during compilation. **Throws keyword** is used for handling checked exceptions . By using throws we can declare multiple exceptions in one go.

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 tedious 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.

**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.

**Which exception should be declared**

**Ans)** checked exception only, because:

* **unchecked Exception:** under your control so correct your code.
* **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

|  |
| --- |
| **import java.io.\*;****class throweg****{****void mymethod(int num) throws IOException, ClassNotFoundException****{****if(num==1)****{****throw new IOException("this is throw IOException occured");****}****else****throw new ClassNotFoundException("class not found error");****}****}****public class throwexample2****{****public static void main(String args[])****{****try****{****throweg obj=new throweg();****obj.mymethod(1);****}****catch(Exception e)****{****System.out.println("this is JVM error message "+e);****System.out.println();****}****}****}** |

**Output**

|  |
| --- |
| **D:\javaprgs>javac throwexample2.java****D:\javaprgs>java throwexample2****this is JVM error message java.io.IOException: this is throw IOException occur****ed** |

**User defined or custom exception**

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

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

The following methods defined in Exception class are used

1. getLocalizedMessage()
2. getMessage()
3. toString()

|  |
| --- |
| **class InvalidAgeException extends Exception****{****InvalidAgeException(String s)****{****super(s);****}****}****class testCustomException** **{****static void validate(int age) throws InvalidAgeException****{****if (age<18)****throw new InvalidAgeException("not valid");****else****System.out.println("welcome to vote");****}****public static void main(String args[])****{****try****{****validate(30);****}****catch(Exception m)****{****System.out.println("Exception occured : "+m);****} //try-catch****System.out.println("rest of the code");****}** **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac testCustomException.java****E:\javaprgs>java testCustomException****welcome to vote****rest of the code** |

**If you pass the value less than 18 you will get as follows**

|  |
| --- |
| **E:\javaprgs>javac testCustomException.java****E:\javaprgs>java testCustomException****Exception occured : InvalidAgeException: not valid****rest of the code** |

**Example**

|  |
| --- |
| **class InvalidProductException extends Exception****{****InvalidProductException(String s)****{****super(s);****}****}****class testCustomException1** **{****void productcheck(int weight) throws InvalidProductException****{****if (weight<100)****throw new InvalidProductException("produt not valid");****}****public static void main(String args[])****{****testCustomException1 obj=new testCustomException1();****try****{****obj.productcheck(30);****}****catch(InvalidProductException ex)****{****System.out.println("Exception occured : "+ex.getMessage());****} //try-catch****System.out.println("rest of the code");****}** **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac testCustomException1.java****E:\javaprgs>java testCustomException1****Exception occured : produt not valid****rest of the code** |

**When the productcheck() is given more than 100 as value the following output occurs**

|  |
| --- |
| **E:\javaprgs>javac testCustomException1.java****E:\javaprgs>java testCustomException1****rest of the code** |