Problem without exception handling

**class** TryCatchExample1 {

**public** **static** **void** main(String[] args) {

**int** data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}

## Solution by exception handling

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

### Example 2

**TryCatchExample2.java**

**class** TryCatchExample2 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

            //handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

### Example 3

In this example, we also kept the code in a try block that will not throw an exception.

**TryCatchExample3.java**

**public** **class** TryCatchExample3 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

                         // if exception occurs, the remaining statement will not exceute

        System.out.println("rest of the code");

        }

             // handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

    }

### Example 4

Here, we handle the exception using the parent class exception.

**TryCatchExample4.java**

**public** **class** TryCatchExample4 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

            // handling the exception by using Exception class

**catch**(Exception e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

### Example 5

Let's see an example to print a custom message on exception.

**TryCatchExample5.java**

**public** **class** TryCatchExample5 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

             // handling the exception

**catch**(Exception e)

        {

                  // displaying the custom message

            System.out.println("Can't divided by zero");

        }

    }

}

### Example 6

Let's see an example to resolve the exception in a catch block.

**TryCatchExample6.java**

**public** **class** TryCatchExample6 {

**public** **static** **void** main(String[] args) {

**int** i=50;

**int** j=0;

**int** data;

**try**

        {

        data=i/j; //may throw exception

        }

            // handling the exception

**catch**(Exception e)

        {

             // resolving the exception in catch block

            System.out.println(i/(j+2));

        }

    }

}

### Example 7

In this example, along with try block, we also enclose exception code in a catch block.

**TryCatchExample7.java**

**public** **class** TryCatchExample7 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data1=50/0; //may throw exception

        }

             // handling the exception

**catch**(Exception e)

        {

            // generating the exception in catch block

**int** data2=50/0; //may throw exception

        }

    System.out.println("rest of the code");

    }

}

### Example 8

In this example, we handle the generated exception (Arithmetic Exception) with a different type of exception class (ArrayIndexOutOfBoundsException).

**TryCatchExample8.java**

**public** **class** TryCatchExample8 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

            // try to handle the ArithmeticException using ArrayIndexOutOfBoundsException

**catch**(ArrayIndexOutOfBoundsException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

### Example 9

Let's see an example to handle another unchecked exception.

**TryCatchExample9.java**

**public** **class** TryCatchExample9 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** arr[]= {1,3,5,7};

        System.out.println(arr[10]); //may throw exception

        }

            // handling the array exception

**catch**(ArrayIndexOutOfBoundsException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

### Example 10

Let's see an example to handle checked exception.

**TryCatchExample10.java**

**import** java.io.FileNotFoundException;

**import** java.io.PrintWriter;

**public** **class** TryCatchExample10 {

**public** **static** **void** main(String[] args) {

        PrintWriter pw;

**try** {

            pw = **new** PrintWriter("jtp.txt"); //may throw exception

            pw.println("saved");

        }

// providing the checked exception handler

**catch** (FileNotFoundException e) {

            System.out.println(e);

        }

    System.out.println("File saved successfully");

    }

}

# Java Catch Multiple Exceptions

## Java Multi-catch block

### Example 1

Let's see a simple example of java multi-catch block.

**MultipleCatchBlock1.java**

**public** **class** MultipleCatchBlock1 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

                a[5]=30/0;

               }

**catch**(ArithmeticException e)

                  {

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

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

### Example 2

**MultipleCatchBlock2.java**

**public** **class** MultipleCatchBlock2 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

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

               }

**catch**(ArithmeticException e)

                  {

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

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

**MultipleCatchBlock3.java**

**public** **class** MultipleCatchBlock3 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

                a[5]=30/0;

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

               }

**catch**(ArithmeticException e)

                  {

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

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

### Example 4

In this example, we generate NullPointerException, but didn't provide the corresponding exception type. In such case, the catch block containing the parent exception class **Exception** will invoked.

**MultipleCatchBlock4.java**

**public** **class** MultipleCatchBlock4 {

**public** **static** **void** main(String[] args) {

**try**{

                String s=**null**;

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

               }

**catch**(ArithmeticException e)

                  {

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

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

### Example 5

Let's see an example, to handle the exception without maintaining the order of exceptions (i.e. from most specific to most general).

**MultipleCatchBlock5.java**

**class** MultipleCatchBlock5{

**public** **static** **void** main(String args[]){

**try**{

**int** a[]=**new** **int**[5];

    a[5]=30/0;

   }

**catch**(Exception e){System.out.println("common task completed");}

**catch**(ArithmeticException e){System.out.println("task1 is completed");}

**catch**(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

   System.out.println("rest of the code...");

 }

}

# Java Nested try block

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the **inner try block** can be used to handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle the **ArithemeticException** (division by zero).

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

### Syntax:

....

//main try block

**try**

{

    statement 1;

    statement 2;

//try catch block within another try block

**try**

    {

        statement 3;

        statement 4;

//try catch block within nested try block

**try**

        {

            statement 5;

            statement 6;

     }

**catch**(Exception e2)

        {

//exception message

        }

    }

**catch**(Exception e1)

    {

//exception message

    }

}

//catch block of parent (outer) try block

**catch**(Exception e3)

{

//exception message

}

....

## Java Nested try Example

### Example 1

Let's see an example where we place a try block within another try block for two different exceptions.

**NestedTryBlock.java**

**public** **class** NestedTryBlock{

**public** **static** **void** main(String args[]){

 //outer try block

**try**{

  //inner try block 1

**try**{

     System.out.println("going to divide by 0");

**int** b =39/0;

   }

    //catch block of inner try block 1

**catch**(ArithmeticException e)

    {

      System.out.println(e);

    }

    //inner try block 2

**try**{

**int** a[]=**new** **int**[5];

    //assigning the value out of array bounds

     a[5]=4;

     }

    //catch block of inner try block 2

**catch**(ArrayIndexOutOfBoundsException e)

    {

       System.out.println(e);

    }

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

  }

  //catch block of outer try block

**catch**(Exception e)

  {

    System.out.println("handled the exception (outer catch)");

  }

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

 }

}

**NestedTryBlock.java**

**public** **class** NestedTryBlock2 {

**public** **static** **void** main(String args[])

    {

        // outer (main) try block

**try** {

            //inner try block 1

**try** {

                // inner try block 2

**try** {

**int** arr[] = { 1, 2, 3, 4 };

                    //printing the array element out of its bounds

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

                }

                // to handles ArithmeticException

**catch** (ArithmeticException e) {

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

                    System.out.println(" inner try block 2");

                }

            }

            // to handle ArithmeticException

**catch** (ArithmeticException e) {

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

                System.out.println("inner try block 1");

            }

        }

        // to handle ArrayIndexOutOfBoundsException

**catch** (ArrayIndexOutOfBoundsException e4) {

            System.out.print(e4);

            System.out.println(" outer (main) try block");

        }

**catch** (Exception e5) {

            System.out.print("Exception");

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

        }

    }

}

Why use Java finally block?

* finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
* The important statements to be printed can be placed in the finally block.

Usage of Java finally

Let's see the different cases where Java finally block can be used.

### Case 1: When an exception does not occur

Let's see the below example where the Java program does not throw any exception, and the finally block is executed after the try block.

**TestFinallyBlock.java**

**class** TestFinallyBlock {

**public** **static** **void** main(String args[]){

**try**{

//below code do not throw any exception

**int** data=25/5;

   System.out.println(data);

  }

//catch won't be executed

**catch**(NullPointerException e){

System.out.println(e);

}

//executed regardless of exception occurred or not

**finally** {

System.out.println("finally block is always executed");

}

System.out.println("rest of phe code...");

  }

}

**Output:**



### Case 2: When an exception occurr but not handled by the catch block

Let's see the the fillowing example. Here, the code throws an exception however the catch block cannot handle it. Despite this, the finally block is executed after the try block and then the program terminates abnormally.

**TestFinallyBlock1.java**

**public** **class** TestFinallyBlock1{

**public** **static** **void** main(String args[]){

**try** {

        System.out.println("Inside the try block");

        //below code throws divide by zero exception

**int** data=25/0;

       System.out.println(data);

      }

      //cannot handle Arithmetic type exception

      //can only accept Null Pointer type exception

**catch**(NullPointerException e){

        System.out.println(e);

      }

      //executes regardless of exception occured or not

**finally** {

        System.out.println("finally block is always executed");

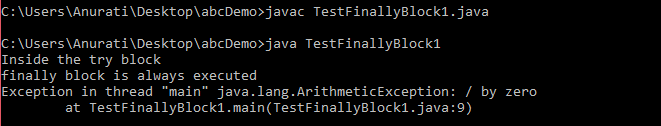
      }

      System.out.println("rest of the code...");

      }

    }

**Output:**



### Case 3: When an exception occurs and is handled by the catch block

**Example:**

Let's see the following example where the Java code throws an exception and the catch block handles the exception. Later the finally block is executed after the try-catch block. Further, the rest of the code is also executed normally.

**TestFinallyBlock2.java**

**public** **class** TestFinallyBlock2{

**public** **static** **void** main(String args[]){

**try** {

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

        //below code throws divide by zero exception

**int** data=25/0;

       System.out.println(data);

      }

      //handles the Arithmetic Exception / Divide by zero exception

**catch**(ArithmeticException e){

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

        System.out.println(e);

      }

      //executes regardless of exception occured or not

**finally** {

        System.out.println("finally block is always executed");

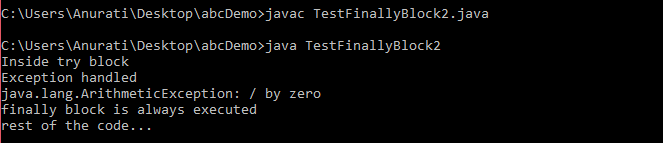
      }

      System.out.println("rest of the code...");

      }

    }

**Output:**



#### Rule: For each try block there can be zero or more catch blocks, but only one finally block.

#### Note: The finally block will not be executed if the program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).

Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

We specify the **exception** object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.

he syntax of the Java throw keyword is given below.

throw Instance i.e.,

1. **throw** **new** exception\_class("error message");

Let's see the example of throw IOException.

1. **throw** **new** IOException("sorry device error");

## Java throw keyword Example

### Example 1: Throwing Unchecked Exception

In this example, we have created a method named validate() that accepts an integer as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

**TestThrow1.java**

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

**class** TestThrow1 {

    //function to check if person is eligible to vote or not

**public** **static** **void** validate(**int** age) {

**if**(age<18) {

            //throw Arithmetic exception if not eligible to vote

**throw** **new** ArithmeticException("Person is not eligible to vote");

        }

**else** {

            System.out.println("Person is eligible to vote!!");

        }

    }

    //main method

**public** **static** **void** main(String args[]){

        //calling the function

        validate(13);

        System.out.println("rest of the code...");

  }

}

### Example 2: Throwing Checked Exception

#### Note: Every subclass of Error and RuntimeException is an unchecked exception in Java. A checked exception is everything else under the Throwable class.

**TestThrow2.java**

**import** java.io.\*;

**public** **class** TestThrow2 {

    //function to check if person is eligible to vote or not

**public** **static** **void** method() **throws** FileNotFoundException {

        FileReader file = **new** FileReader("C:\\Users\\Anurati\\Desktop\\abc.txt");

        BufferedReader fileInput = **new** BufferedReader(file);

**throw** **new** FileNotFoundException();

    }

    //main method

**public** **static** **void** main(String args[]){

**try**

        {

            method();

        }

**catch** (FileNotFoundException e)

        {

            e.printStackTrace();

        }

        System.out.println("rest of the code...");

  }    }

### Example 3: Throwing User-defined Exception

exception is everything else under the Throwable class.

**TestThrow3.java**

// class represents user-defined exception

**class** UserDefinedException **extends** Exception

{

**public** UserDefinedException(String str)

    {

        // Calling constructor of parent Exception

**super**(str);

    }

}

// Class that uses above MyException

**public** **class** TestThrow3

{

**public** **static** **void** main(String args[])

    {

**try**

        {

            // throw an object of user defined exception

**throw** **new** UserDefinedException("This is user-defined exception");

        }

**catch** (UserDefinedException ude)

        {

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

            // Print the message from MyException object

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

        }

    }

}

# Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.

### Syntax of Java throws

return\_type method\_name() **throws** exception\_class\_name{

//method code

}

### Which exception should be declared?

**Ans:** Checked exception only, because:

* **unchecked exception:** under our control so we can correct our code.
* **error:** beyond our control. For example, we are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

### Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

## Java throws Example

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

**Testthrows1.java**

**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...");

  }

}

**There are two cases:**

1. **Case 1:** We have caught the exception i.e. we have handled the exception using try/catch block.
2. **Case 2:** We have declared the exception i.e. specified throws keyword with the method.

### Case 1: Handle Exception Using try-catch block

In case we handle the exception, the code will be executed fine whether exception occurs during the program or not.

**Testthrows2.java**

**import** java.io.\*;

**class** M{

**void** method()**throws** IOException{

**throw** **new** IOException("device error");

 }

}

**public** **class** Testthrows2{

**public** **static** **void** main(String args[]){

**try**{

     M m=**new** M();

     m.method();

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

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

  }

}

### Case 2: Declare Exception

* In case we declare the exception, if exception does not occur, the code will be executed fine.
* In case we declare the exception and the exception occurs, it will be thrown at runtime because **throws** does not handle the exception.

Let's see examples for both the scenario.

**A) If exception does not occur**

**Testthrows3.java**

**import** java.io.\*;

**class** M{

**void** method()**throws** IOException{

  System.out.println("device operation performed");

 }

}

**class** Testthrows3{

**public** **static** **void** main(String args[])**throws** IOException{//declare exception

     M m=**new** M();

     m.method();

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

  }

}

**B) If exception occurs**

**Testthrows4.java**

**import** java.io.\*;

**class** M{

**void** method()**throws** IOException{

**throw** **new** IOException("device error");

 }

}

**class** Testthrows4{

**public** **static** **void** main(String args[])**throws** IOException{//declare exception

     M m=**new** M();

     m.method();

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

  }

}