# 'throw' keyword:

'throw' is a java keyword, it can be used to rise exceptions intentionally as per the developers application requirement.

**Syntax:**

**throw new Exception\_Name("----Exception Description-----");**

**EX:**

class Test

{

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

String accNo=args[0];

String accName=args[1];

int pin\_Num=Integer.parseInt(args[2]);

String accType=args[3];

System.out.println("Account Details");

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

System.out.println("Account Number :"+accNo);

System.out.println("Account Name :"+accName);

System.out.println("Account Type :"+accType); System.out.println("Account PIN Number:"+pin\_Num); if(pin\_Num>=1000 && pin\_Num<=9999) {

System.out.println("valid PIN Number");

}

else

{

**throw new RuntimeException("Invalid PIN Number, enter Valid 4 digit PIN Number”);**

}

}}

D:\javaapps>javac Test.java

D:\javaapps>java Test abc123 AAA 1234 Savings

--- Account details without Exception-----

D:\javaapps>java Test abc123 AAA 123 Savings

--- Account details are displayed with Exception-----

**---Account details----**

**Exception in thread "main" java.lang.RuntimeException: Invalid PIN Number, enter valid 4 digit PIN number**

**at Test.main(Test.java:17)**

**There are two ways to handle exceptions .**

**1.By using 'throws' keyword.**

**2.By using try-catch-finally block.**

1. **'throws' keyword:**

* It is a Java keyword, it can be used to bypass the generated exception from the present method or constructor to the caller method (or) constructor.
* In Java applications, 'throws' keyword will be used in method declarations, not in method body.
* In Java applications,”throws” keyword allows an exception class name,it should be either same as the generated exception or super class to the generated exception. It should not be subclass to the generated Exception.
* 'throws' keyword allows more than one exception in method prototypes.
* In Java applications,”throws” keyword will be utilized mainly for checked exceptions.

**EX:** void m1() throws RuntimeException{

throw new ArithmeticException();

}

**Status**: Valid

**EX:**void m1() throws FileNotFoundException{

throw new IOException();

}

**Status**:InValid

**EX:**

Void m1() throws NullPointerException,ClassNotFoundException{ }

**Status**:Valid

**EX:**

Void m1() throws IOException,FileNotFoundException

{

}

**Status**:Valid , Not Suggestable

If we specify any super exception class along with throws keyword,then it is not necessary to specify any of its child exception classes along with “throws” keyword.

**NOTE: In any Java method, if we call some other method which is bypassing an exception by using “throws” keyword,then we must handle that exception either by using “throws” keyword in the present method[Caller Method] prototype or by using “try-catch-finally” in the body of the present method[Caller method].**

**EX:**

Void m1() throws Exception{

-----

------

}

Void m2(){

try{

m1();

}

catch(Exception e){

e.printStackTrace();

}

}

**EX:**

void m1() throws Exception{

----

}

void m2() throws Exception{

m1();

}

**EX:**

import java.io.\*;

class A{

void add() throws Exception{

concat();

}

Void concat() throws IOException{

throw new IOException();

}

}

class Test{

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

A a=new A();

a.add();

}

}

**Internal Flow of Throws keyword:**

**If we execute the above program,then JVM will recognize throw keyword in concat() method and JVM will rise an exception in concat() method, due to throws keyword in concat() method prototype, Exception will be bypassed to concat() method call that is in add(), due to throws keyword in add() method Exception will be bypassed to add() method call , that is , in main() method, Due to 'throws' keyword in main() method Excepotion will be bypassed to main() method call , that is, to JVM, where JVM will activate "Default Exception Handler" , where Default Exception Handler will display exception details.**

Q)What are the differences between “throw” and “throws” keywords?

**Ans:**

1.'throw' keyword can be used to rise the exceptions intentionally as Per the application reqirement.

'throws' keyword is able to by pass the exception from the present method to the caller method.

2.'throw' keyword will be utilized in method body.

'throws' keyword will be used in method declarations or in method prototype (or) in method header part.

3.'throw' keyword allows only one exception class name.

'throws' keyword allows more than one exception class name.

# try-catch-finally:

In Java application “throws” keyword is not really an exception handler,because “throws” keyword will bypass the exception handling responsibility from present method to the caller method.

If we want to handle the exceptions,the location where exceptions are generated then we have to use “try-catch-finally”.

**Syntax:**

try{

----

}

catch(Exception\_Name e){

----

}

finally{

----

}

where the purpose of try block is to include a set of instructions , which may rise an exception.

If JVM identify any exception inside "try" block then JVM will bypass flow of execution to "catch" block by skipping all the remaining instructions in try block and by passing the generated Exception object reference as parameter.

**If no exception is identified in "try" block then JVM will execute completely "try" block,at the end of try block, JVM will bypass flow of execution to "finally" block directly.**

The main purpose of catch block is to catch the exception from try block and to display exception details on command prompt.

To display exception details on command prompt,we have to use the following three approaches.

1.e.printStackTrace()

2.System.out.println(e):

3.System.out.println(e.getMessage());

1. **e.printStackTrace():**

It will display the exception details like Exception Name,Exception Description and Exception Location.

1. **System.out.println(e):**

If we pass Exception object reference variable as parameter to System.out.println(-) method then JVM will access Exception class toString() method internally,it will display the exception details like Exception name,Exception description.

1. **System.out.println(e.getMessage()):**

Where getMessage() method will return a String contains the exception details like only Description of the exception.

**EX:**

class Test{

public static void main(String args[]){

try{

throw new ArithmeticException("My Arithmetic Exception"); }

catch(ArithmeticException e){

e.printStackTrace();

System.out.println();

System.out.println(e);

System.out.println();

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

}

finally{

}

}

}

**OP:**

java.lang.ArithmeticException:My Arithmetic Exception at Test.main(Test.java:7)

java.lang.ArithmeticException:My Arithmetic Exception

**Where the main purpose of finally block is to include some Java code , it will be executed irrespective of getting exception in "try" block and irrespective of executing "catch" block.**

**Q)What is the difference between "final","finally" and "finalize" in JAVA?**

**Ans:**

1."final" is a keyword it can be used to declare constant expressions.

There are three ways to use final keyword in java applications.

1. final variable:It will not allow modifications over its value.
2. final methods:It will not allow method overriding.
3. final class:It will not allow inheritance, that is, sub classes.

**2.finally block**: It is part of try-catch-finally syntax,it will include some instructions,**which must be executed by JVM irrespective of getting exception from try block and irrespective of executing catch block.**

**3.finalize(): It is a method in java.lang.Object class,it will be executed just before destroying objects inorder to give final notification to the user about to destroy objects.**

Q)Find the output from the following programs?

class Test{

public static void main(String args[]){

System.out.println("Before Try");

try{

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

}

catch(Exception e){

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

}

finally{

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

}

System.out.println("After Finally");

}

}

**OP:**

Before try

Inside try

Inside finally

After finally

**EX:** class Test{

public static void main(String args[]){

System.out.println("Before Try");

try{

System.out.println("Before Exception in try");

float f=100/0;

System.out.println("After Exception in try");

}

catch(Exception e){

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

}

finally{

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

}

System.out.println("After Finally");

}

}

**OP**: Before try

Before exception in try

Inside catch

Inside finally

After finally

Q)Find the output from the following program?

class A{

int m1(){

try{

return 10;

}

catch(Exception e){

return 20;

}

finally{

return 30;

}

}

}

class Test{

public static void main(String args[]){

A a=new A();

int val=a.m1();

System.out.println(val);

}

}

**OP:**

30

**NOTE: finally block provided return statement is the finally return statement for the method**

**Q)Is it possible to provide "try" block without "catch" block?**

**Ans:**

Yes,it is possible to provide try block with out catch block but by using "finally" Block.

**Syntax**:

try{

}

finally{

}

**EX:**

class Test{

public static void main(String args[]){

System.out.println("Before try");

try{

System.out.println("Before Exception inside try");

int i=100;

int j=0;

float f=i/j;

System.out.println("After Exception inside try");

}

finally{

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

}

System.out.println("After Finally");

}

}

**Status:** No Compilation Error.

**OP:**

Before try

Before exception inside try

Inside finally

Exception in thread “main” java.lang.ArithmeticException:/by zero at Test.main(Test.java:11)

**REASON: When JVM encounter exception in try block,JVM will search for catch block,if no catch block is identified,then JVM will terminate the program abnormally after executing finally block.**

Q)Is it possible to provide "try" block with out "finally" block?

**Ans:**

Yes,it is possible to provide "try" block with out "finally" block but by using "catch" block.

**Syntax:**

try{

-------

--------

}

catch(Exception e){

-------------

-------------

}

1. Is it possible to provide try-catch-finally

a)inside try block,

b)inside catch block and

c)inside finally block

**Ans:**

Yes,it is possible to provide try-catch-finally inside try block,inside catch block and inside finally block.

**Syntax-1**:

try{

try{

}

catch(Exception e){

}

finally{

}

}

catch(Exception e){

}

finally{

}

**Syntax-2**:

try{

}

catch(Exception e){

try{

}

catch(Exception e)

{

}

finally{

}

}

finally{

}

**Syntax-3**:

try{

}

catch(Exception e){

}

finally{

try{

}

catch(Exception e){

}

finally{

}

}

Q)Is it possible to provide more than one catch block for a single try block?

**Ans:**

Yes,it is possible to provide more than one catch block for a single try block but with the following conditions.

**1.If no inheritance relation existed between exception class names which are specified along with catch blocks then it is possible to provide all the catch blocks in any order.If inheritance relation is existed between exception class names then we have to arrange all the catch blocks as per Exception classes inheritance increasing order.**

1. **In general,specifying an exception class along with a catch block is not giving any guarantee to rise the same exception in the corresponding try block,but if we specify any pure checked exception along with any catch block then the corresponding "try" block must rise the same pure checked exception.**

**Ex1:**

try{

}

catch(ArithmeticException e){

}

catch(ClassCastException e){

}

catch(NullPointerException e){

}

**Status**:Valid Combination

**Ex2:**

try{

}

catch(NullPointerException e){

}

catch(ArithmeticException e){

}

catch(ClassCastException e){

}

**status**:Valid Combination

**Ex3:**

try{

}

catch(ArithmeticException e){

}

catch(RuntimeException e){

}

catch(Exception e){

}

**Status:**Valid

**Ex4**:

try{

}

catch(Exception e){

}

catch(RuntimeException e){

}

catch(ArithmeticException e){

}

**status:** Invalid

**Ex5:**

try{

throws new ArithmeticException("My Exception");

}

catch(ArithmeticException e){

}

catch(IOException e){

}

catch(NullPointerException e){

}

**Status**:Invalid

**Ex6:**

try{

throw new IOException("My Exception");

}

catch(ArithmeticException e){

}

catch(IOException e){

}

catch(NullPointerException e){

}

**status**:Valid