1. **Exception Introduction.**
2. **Exception Handling.**
3. **Exception Message Format.**
4. **Inner try-catch-finally.**
5. **Finally.**
6. **Try with variables.**
7. **Exception Propagation between Calling Method and Called Method.**
8. **Throw.**
9. **Throws.**
10. **Custom Exceptions.**
11. **Important Point about Exception.**

**EXCEPTION INTRODUCTION**

**1.Types Of Error:-** In object-oriented programming languages, we will have two types of errors.

a. Compile-time Errors.

b. Run-time Errors

**1.a.Compile-time Errors:-** The compile time errors are occurred during the compilation of Java program. we get compile time errors because of syntax mistakes.

Ex:- missing (;) semicolon.

Variable not found.

**1.b.Run-time Errors:-** Run-time errors are occurred during execution of class byte code. we will get run-time error because of following 3 reasons.

a. Wrong Inputs.

b. Wrong logic.

c. Because of Memory issues in JVM runtime areas.

When runtime error occurs, JVM throws error and JVM terminates program execution abnormally.

**2. Exception Definitons:-**

**Def1:-** Exception is run-time error.

**Def2:-** Exception is an object that is instance of one of the sub classes of **java.lang.Throwable**.

**Def3:-** An Exception is an event, which occurs during the program execution and it disrupts the normal execution flow of program.

**3. Exception Categories:-** Based on exception handling procedure all exceptions are divided into two categories:

3.1. Checked Exception.

3.2. UnChecked Exception.

**3.1.Checked Exception:-** when exception is thrown using throw keyword, If that exception handling is checked by compiler then that exception is checked exception .

If checked exceptions are not caught by using “try/catch” or not reported by using “throws” keyword, compiler throws

**CE: “Unreported exception must be caught or declared to be thrown”.**

**Example:-Throwable , Exception and Dircect sub classes of throwable are checkedException.**

**3.2.UnChecked Exception:-** when exception is thrown using thrown keyword, if that exception handling is not checked by compiler then that exception is Unchecked exception.

**Example:- Error,RuntimeException are Unchecked Exception.**

**4.Throwable Class:-** Exception is an object it is an instance of one of the subclass of java.lang.Throwble class. It is used for representing the logical mistake occurred in the program. SUN(Oracle) defined different exception classes in java.lang, java.io, java.awt, …etc for representing different logical mistakes. All these classes are subclasses of “**java.lang.Throwable**” class.

When Exception occurred, JVM creates exception class object that is associated with that logical mistake and terminates the current method execution by throwing this exception object by using “throw” keyword.

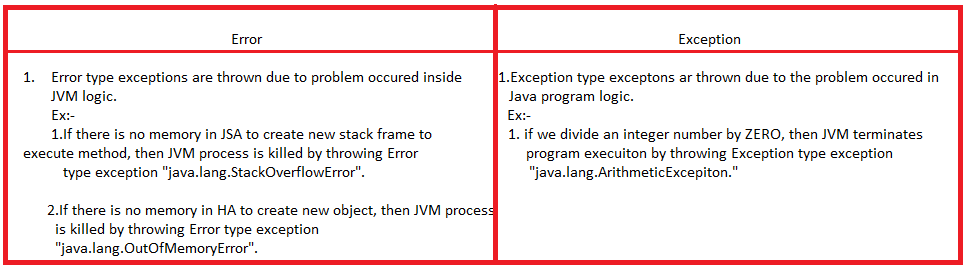
Basically Throwable class all sub classes are divided into two types.They are

1.Error.

2.Exception.

PICTURE

**4.1. Diff between Error and Exception**.



**EXCEPTION HANDLING**

**1.Definiton:-** The process of reporting an exception to method caller or catching an exception is called exception handling.

The program should handle exception

1. To stop the abnormal termination
2. To provide user understandable messages when exception is raised. So that user can take decision without developer help.

The following keywords are used in the process of handling exception.

1. **try b. catch c. finally d. throw e. throws.**

**2.Try:-** The programmer writes **exception causing statements** in try block.

**3.Catch:-** The catch block is used to catch exception which is thrown from its corresponding try block. The catch block syntax is looks like constructor syntax. It does not take accessibility modifier,return types. It takes only single parameter of type “java.lang.Throwable” or Its subclass.

**The single catch block with parameter java.lang.Exception catch all exceptions.**

**4.finally:-** The finally keyword is used for create statement block to execute them compulsory when control come out from try block.

Goto-Finally file for more information.

**5.Rules in using try,catch finally.**

**Rule1:-**try must be followed either “Zero” or ‘n’ number of catch blocks OR ‘1’ finally block else it leads to CE: **“try without catch or finally”.**

**Rule2:-**catch must be placed immediately after try block else it leads to

CE: **Catch without try.**

**Rule3:-**After try, Multiple catch statements can be placed. But programmer should follow below two rules in placing multiple catch blocks.

1. Catch blocks should not be duplicated.
2. Super class parameter catch block should not be placed before child class parameter catch block.

Q) when should we define multiple catch blocks to single try block?

A) to print message specific to an exception

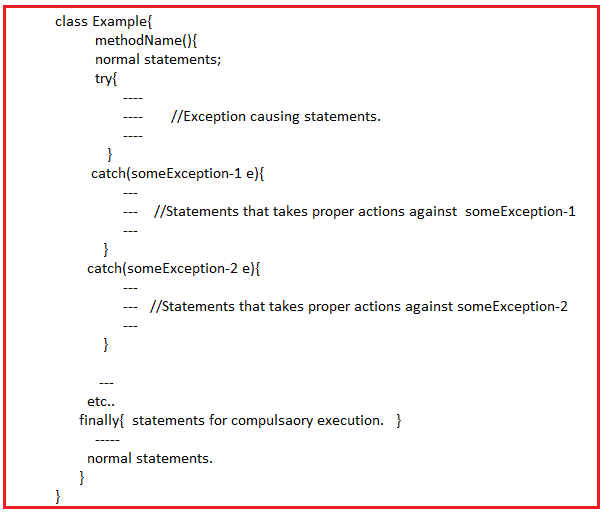
To execute some logic specific to an exception.

**Rule4:-**finally must be placed either immediately after “try” or after “try/catch” else it leads to

CE:”final with out try”.

**Rule4:-** try/catch/finally blocks are not allowed at class level.

**6.Syntax to use try,catch,finally:-**



**7.try/catch/finally blocks execution control flow:-**

When exception is raised in try block JVM creates that exception type class object and check for match in among all catch blocks. If match is found, JVM executes that catch block , finally and normal statements .

If catch block is not matched then **JVM default handler** handles exception and terminates program execution abnormally means catch blocks and normal statements.

1.

class Sample{

public static void main(String arg[])

{

try{}

}

}

Error: It leads to CE: try without 'catch' or 'finally'

2. class Sample{

public static void main(String arg[])

{

try{}

catch(Exception e){}

}

}

NoError.

3. class Sample{

public static void main(String arg[])

{

try{}

catch(Exception e){}

finally{}

}

}

NoError

4. class Sample{

public static void main(String arg[])

{

catch(Exception e){}

}

}

Error: Catch without try.

5. class Sample{

public static void main(String arg[])

{

finally{}

}

}

Error: final with out try.

6. class Sample{

public static void main(String arg[])

{

try{}

catch(ArithmeticException ae){}

catch(Exception e){}

finally{}

}

}

NoError

7. class Sample{

public static void main(String arg[])

{

try{}

catch(Exception e){}

catch(ArithmeticException ae){}

finally{}

}

}

We will get error because parent class exeception is before to child class exception.

8. class Sample{

public static void main(String arg[])

{

try{}

catch(ArithmeticException ae){}

catch(Exception e){}

finally{}

try{}

catch(ArithmeticException e1){}

catch(Exception e1){}

finally{}

}

}

No Error

**Important points:-**

**1.try/catch/finally can be written inside method, another try block, another catch block and another finally block.**

**2.one or More than one try/catch/finally blocks are written in a method, try,catch or finally.**

**Examples:-**

import java.util.Scanner;

class Sample{

public static void main(String arg[])

{

int a;

Scanner s1=new Scanner(System.in);

System.out.print("Enter a value:");

a=s1.nextInt();

try{

System.out.println("Division Result:"+20/a);

}

catch(ArithmeticException e)

{

System.out.println("Catch block:"+"Number is divided by Zero");

}

finally{

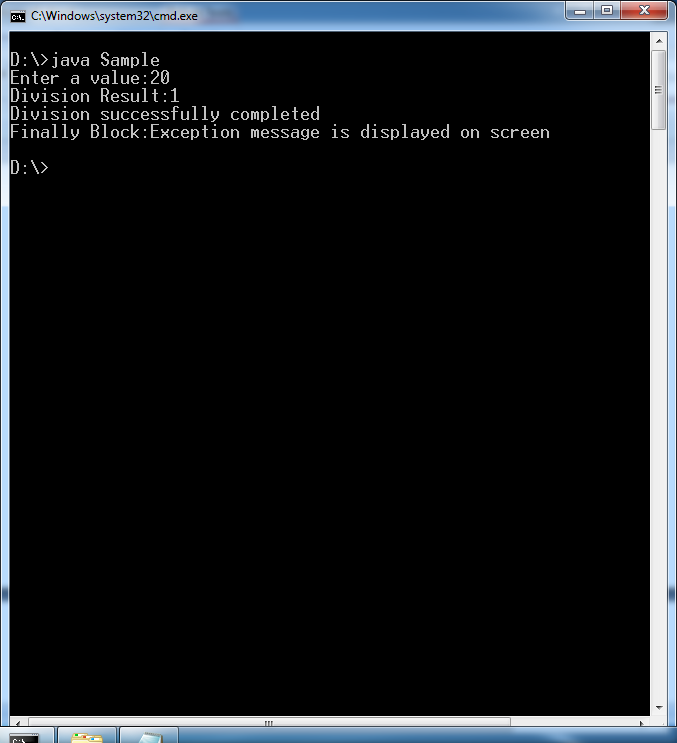
System.out.println("Finally Block:"+"Exception message is displayed on screen");

}

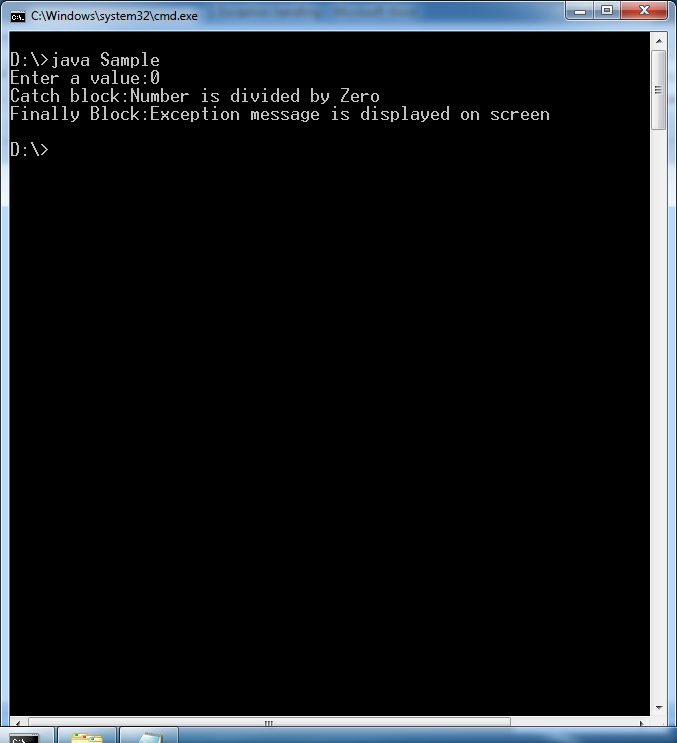
}

}

**Run-1:**

****

**Run-2**

****

**Example:2**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int [] a={1,2,3};

String s1=new String("rock");

try{

System.out.println("Third Element:"+a[4]);

System.out.println("element Accessed:");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Catch block:"+"From Unexisting location, u tried to read value from Array");

}

finally{

System.out.println("Finally Block:"+"Logic of file closing.");

}

try{

System.out.println("Third Element:"+s1.charAt(10));

System.out.println("element Accessed:");

}

catch(StringIndexOutOfBoundsException e)

{

System.out.println("Catch block:"+"From Unexisting location, u tried to read value from String");

}

finally{

System.out.println("Finally Block:"+"Logic of DB closing");

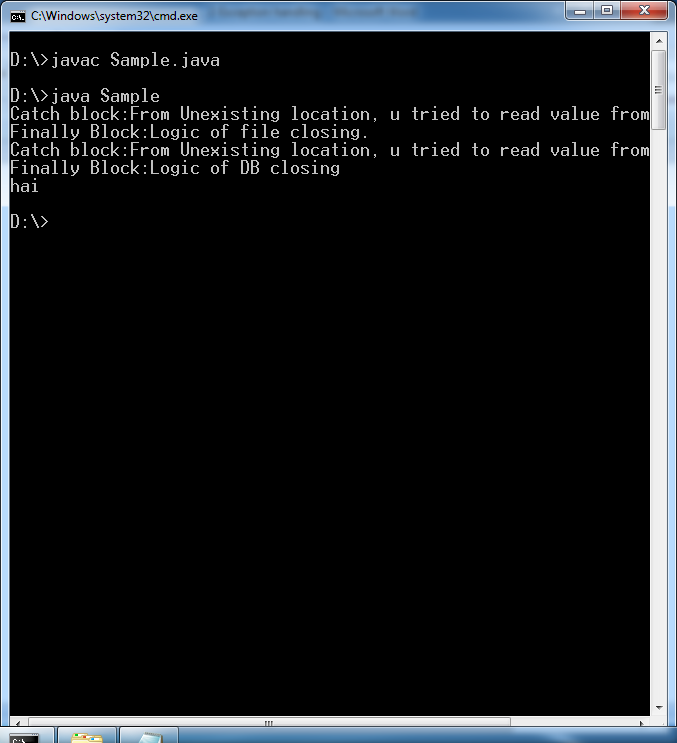
}

System.out.println("hai");

}

}

Output:



Example:3

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int [] a={1,2,3};

String s1=new String("rock");

int x;

Scanner s2=new Scanner(System.in);

try{

System.out.println("Third Element:"+a[4]);

System.out.println("element Accessed:");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Catch block:"+"From Unexisting location, u tried to read value from Array");

}

finally{

System.out.println("Finally Block:"+"Logic of file closing.");

try{

System.out.println("Enter a value:");

x=s2.nextInt();

System.out.println("Division Result:"+20/x);

}

catch(ArithmeticException e)

{

System.out.println("User Entered 0");

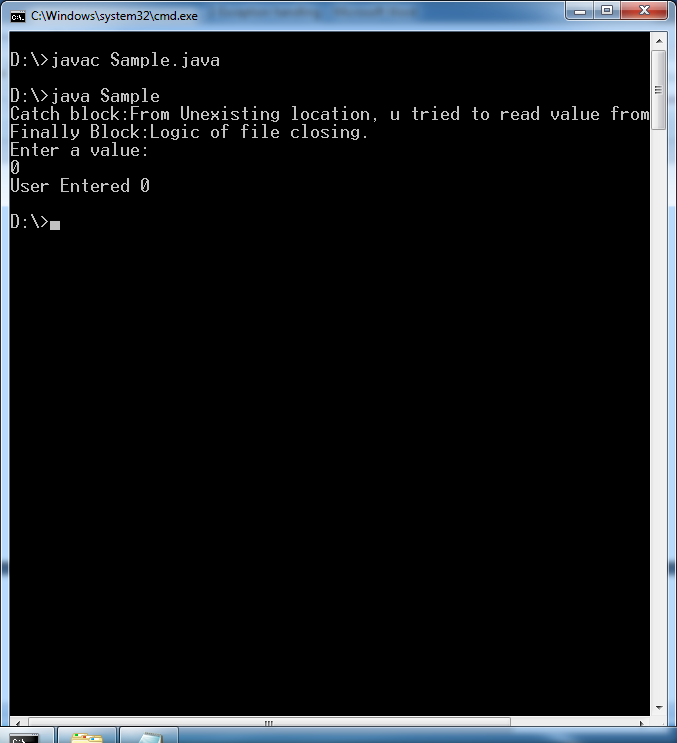
}

}

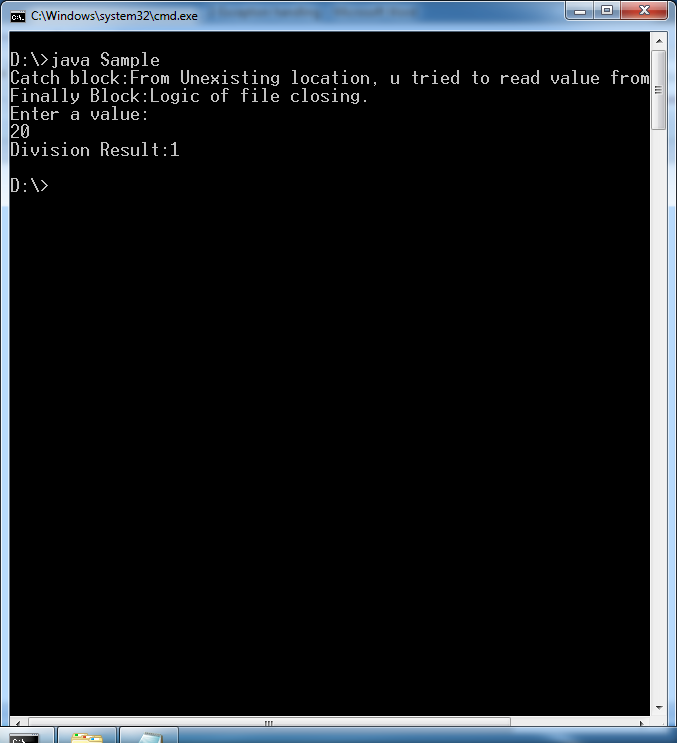
}

}

Run:1



Run-2:



**EXCEPTION MESSAGE FORMAT**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

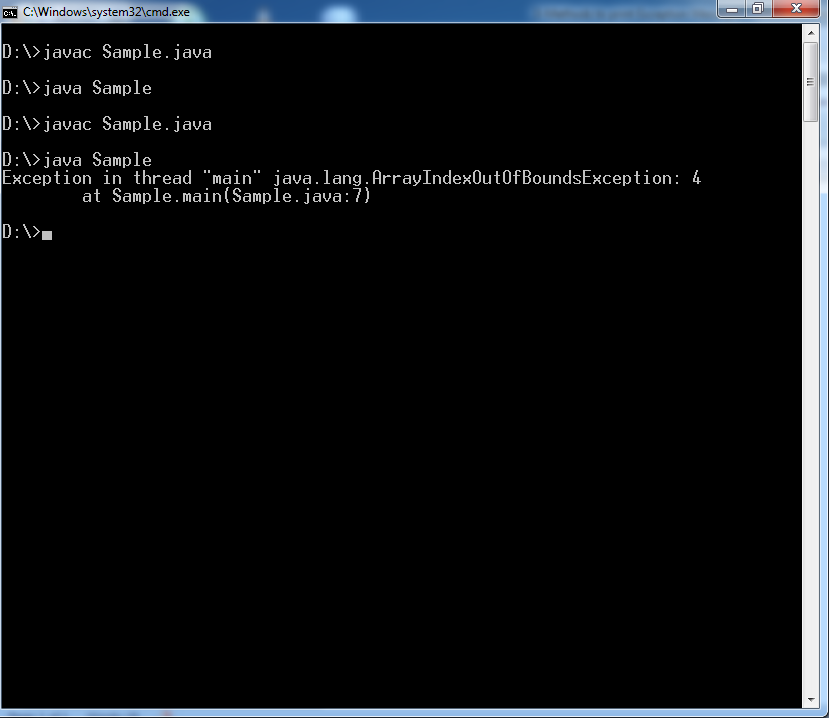
{

int [] a={1,2,3};

System.out.println("Third Element:"+a[4]);

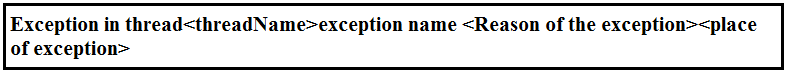
}

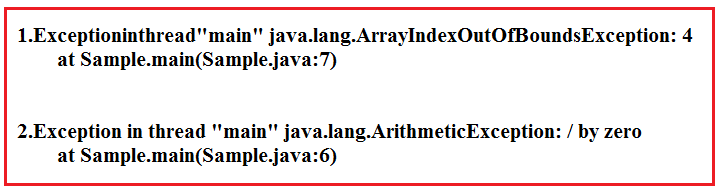
}



Definitely, The user can’t under stand exception message. Because It is JVM printed exception messages.

**1.JVM printed Exception Message Format:-**

****

**Examples**

The throwable class has below methods to print exception messages in another formats.

1.Public void printStackTrace():



2. public String toString():



3. public String getMessage():



**Example:-**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

System.out.println(20/0);

}

catch(ArithmeticException e){

e.printStackTrace();

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

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

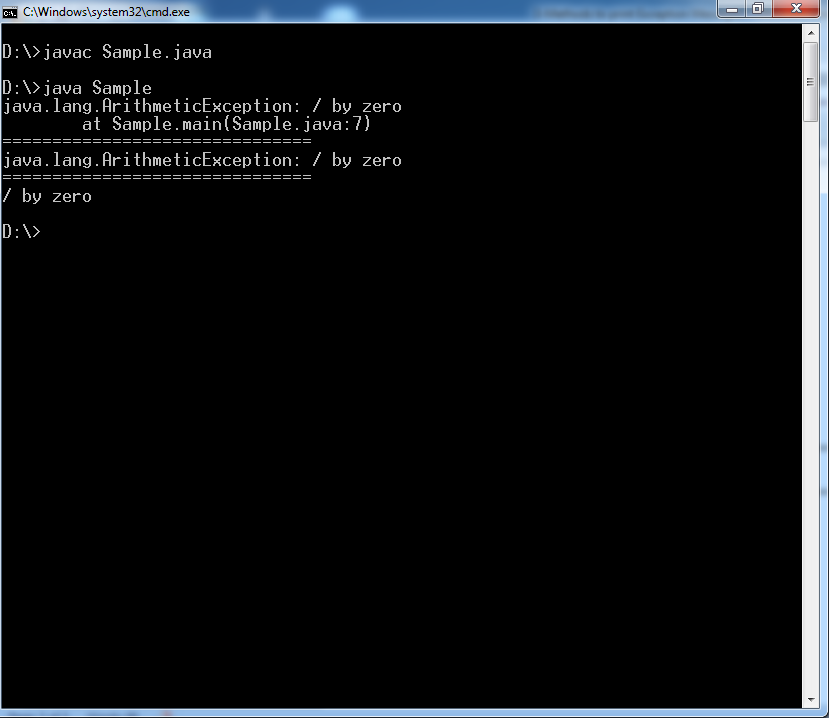
System.out.println("===============================");

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

}

}

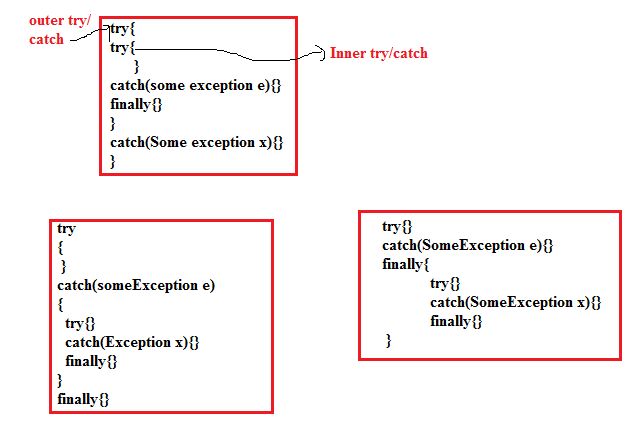
}



**INNER TRY-CATCH-FINALLY**

**1.Definition:-**The try/catch placed inside another try/catch/finally is called inner try/catch.

Examples:



**2.Inner try,catch execution control flow:**

* The exception raised in inner try is caught by inner catch and the statements placed after inner try/catch are executed.
* If the inner catch parameter is not matched with exception raised in inner try, that exception is propagated to outer try and it is caught by the outer catch and the statements placed after outer try/catch are executed.
* If outer catch also not matched, that exception is propagated to JVM and program execution is terminated abnormally.

Case1:-Exception is raised before outer try

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

System.out.println(20/0);

try{

}

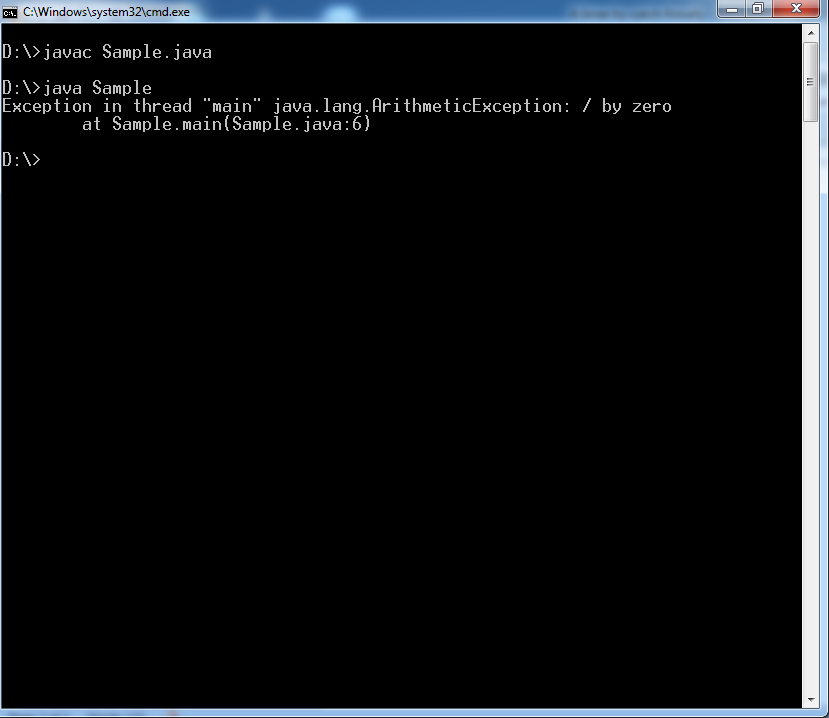
catch(ArithmeticException e){

e.printStackTrace();

}

}

}



Case2:- Exception is raised in outer try, inner try’s catch and also outer try’s catch is matched.

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

System.out.println(20/0);

try{System.out.println("Inner Try");}

catch(ArithmeticException e){ System.out.println("Inner Catch");}

}

catch(ArithmeticException e){

e.printStackTrace();

}

}

}



Case3: Exception is raised in inner try, inner try’s catch and also outer try’s catch is matched.

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

try{System.out.println("Inner Try");

System.out.println(20/0);

}

catch(ArithmeticException e){ System.out.println("Inner Catch");}

}

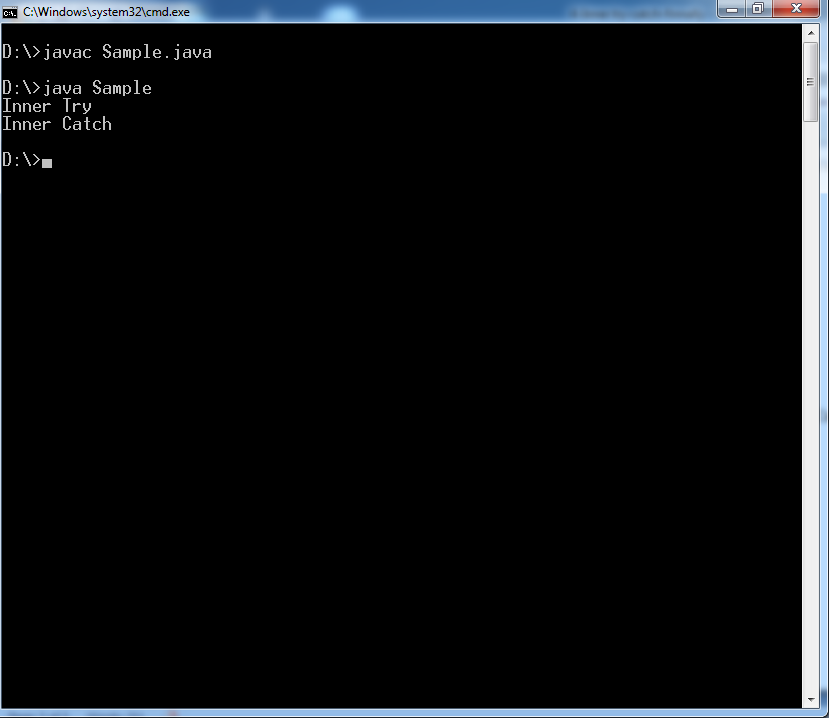
catch(ArithmeticException e){

e.printStackTrace();

}

}

}



**Case4:** Exception is raised in inner try,inner catch is not matched, but outer catch is matched.

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

try{System.out.println("Inner Try");

System.out.println(20/0);

}

catch(ArrayIndexOutOfBoundsException e){ System.out.println("Inner Catch");}

}

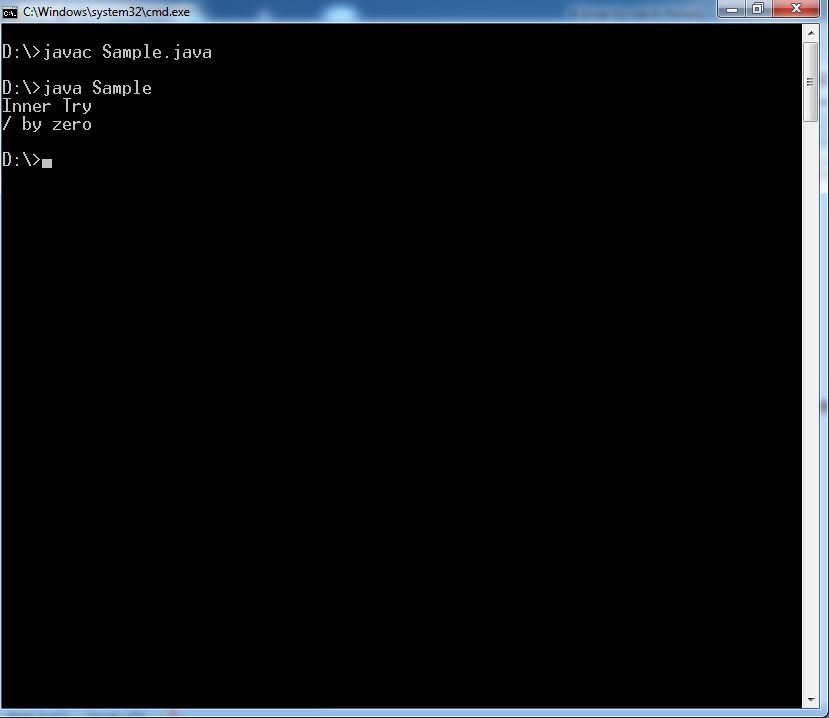
catch(ArithmeticException e){

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

}

}

}



**Case5:** Exception is raised in inner try, inner catch is matched , again exception is raised in inner catch, then who will catch this exception.

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

try{System.out.println("Inner Try");

System.out.println(20/0);

}

catch(ArithmeticException e){ System.out.println("Inner Catch");

System.out.println(20/0);

}

}

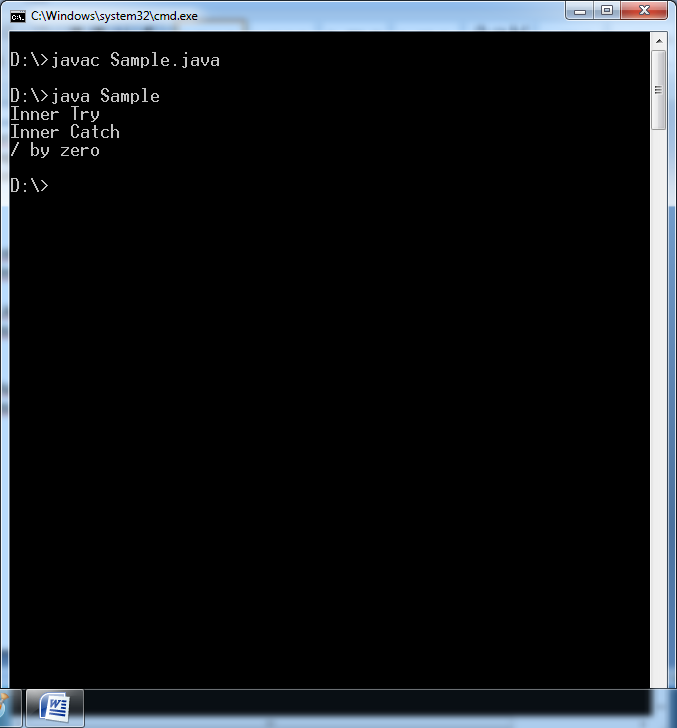
catch(ArithmeticException e){

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

}

}

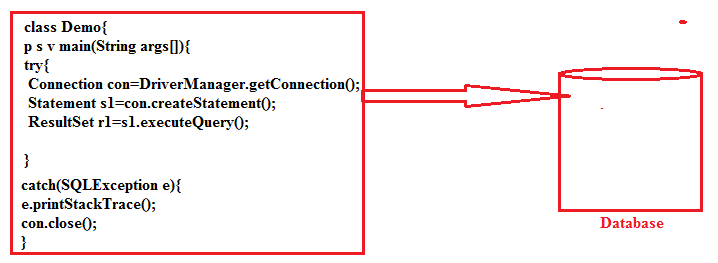
}



**FINALLY**

In real time projects, we create JDBC objects for establishing connection in try block and at the end of try block we must close those objects. Since the statements written in try and catch are not guaranteed to be executed we must place them in finally block.

Example:-



In above example, when exception will raise, opened connection will be closed. Otherwise connection will not be closed.

Whether exception will raise or not, program has to close that opened connection. Therefore connection closing statement must be written inside the final block.

**1.The final block execution happens in 7 cases.**

Case1:try block completed normally.

Case2:try block execution is terminated with exception and it is caught.

Case3:try block execution is terminated with exception and it is not caught.

Case4:.try block execution is terminated with break statement

Case5: try block execution is terminated with continue statement.

Case6: try block execution is terminated with return statement.

Case7: try block execution is terminated with System.exit(0) method call.

**Case1 Example:**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

System.out.println("Try execution Completed");

}

catch(ArithmeticException e){

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

}

finally{

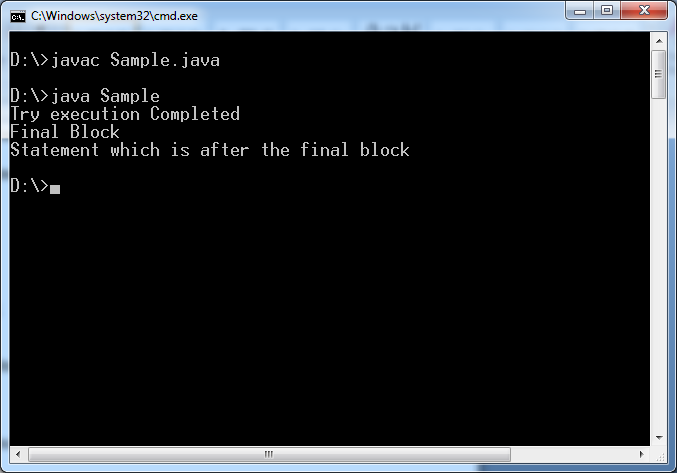
System.out.println("Final Block");

}

System.out.println("Statement which is after the final block");

}

}



**Case2 Example:**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

System.out.println("Exception will be raised and will be caught");

int a=20/0;

}

catch(ArithmeticException e){

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

}

finally{

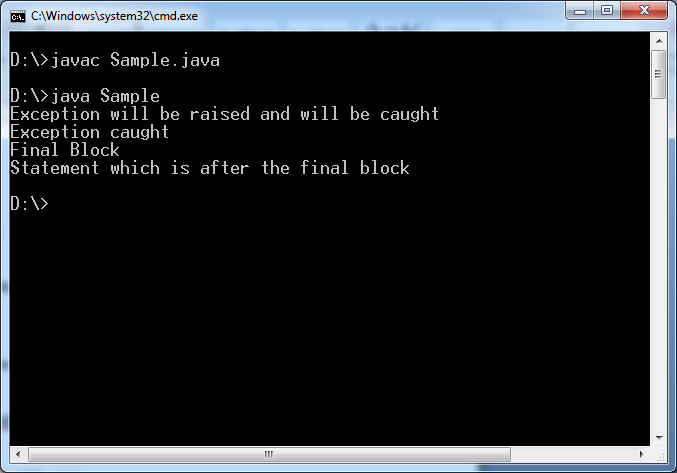
System.out.println("Final Block");

}

System.out.println("Statement which is after the final block");

}

}

****

**Case3 Example:**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

try{

System.out.println("Exception will be raised and will not be caught");

int a=20/0;

}

catch(NullPointerException e){

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

}

finally{

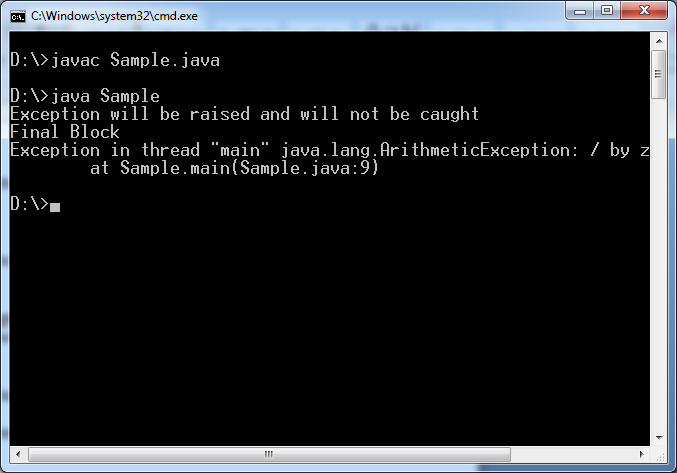
System.out.println("Final Block");

}

System.out.println("Statement which is after the final block");

}

}



**2.Finally with return statement.**

**2.1. The value returned from try block or from catch block is overridden by finally block returning value.**

Example1:

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int x=m1();

System.out.println(x);

}

public static int m1(){

try{

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

return 10;

}

catch(ArithmeticException e){

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

return 20;

}

finally{

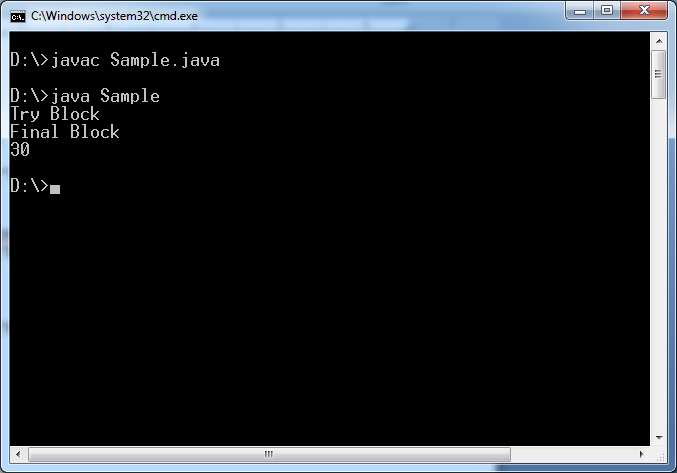
System.out.println("Final Block");

return 30;

}

}

}



Example:2

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int x=m1();

System.out.println(x);

}

public static int m1(){

try{

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

int a=20/0;

return 10;

}

catch(ArithmeticException e){

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

return 20;

}

finally{

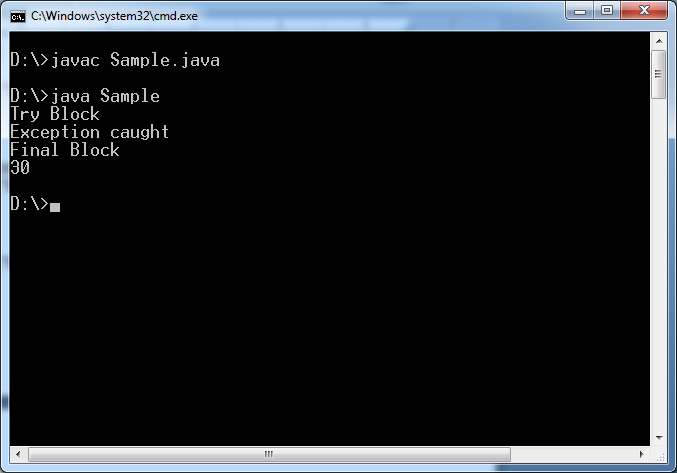
System.out.println("Final Block");

return 30;

}

}

}



2.2 If return statement is placed in finally block, we can not place any statement after try/catch/finally block. . Otherwise compiler will throw “**Unreachable statement error”. These statements never executed by JVM.**

Example:1

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int x=m1();

System.out.println(x);

}

public static int m1(){

try{

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

}

catch(ArithmeticException e){

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

}

finally{

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

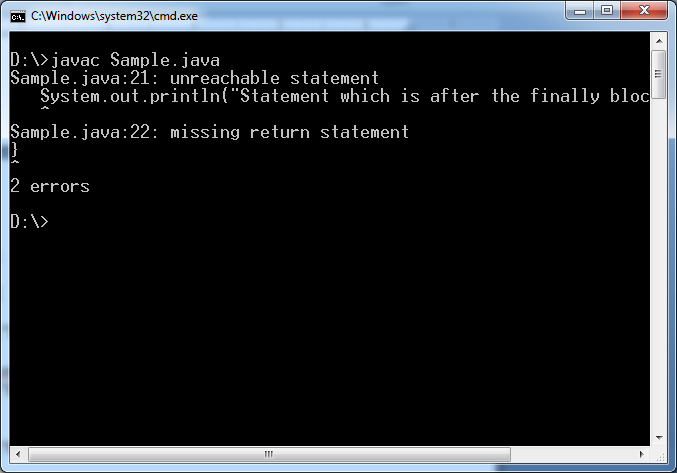
return 30;

}

System.out.println("Statement which is after the finally block");

}

}



**2.3.** **The exception throwing from try block will be replaced with finally block throwing exception, and then finally block exception will be passed to main().**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

m1();

}

public static void m1(){

try{

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

int ab=20/0;

}

catch(NullPointerException e){

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

}

finally{

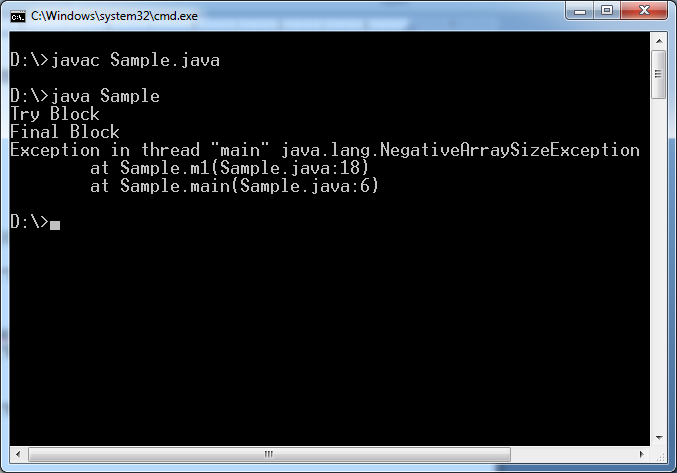
System.out.println("Final Block");

int a[]=new int[-3];

}

}

}



**2.4. when control come out from inner try block both inner finally and outer finally are compulsory executed.**

Example:

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

try{

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

try{int a=20/0;}

catch(ArithmeticException e){

System.out.println("Inner Exception caught");

}

finally{ System.out.println("Inner finally block");}

}

catch(NullPointerException e){

System.out.println("Outer Exception caught");

}

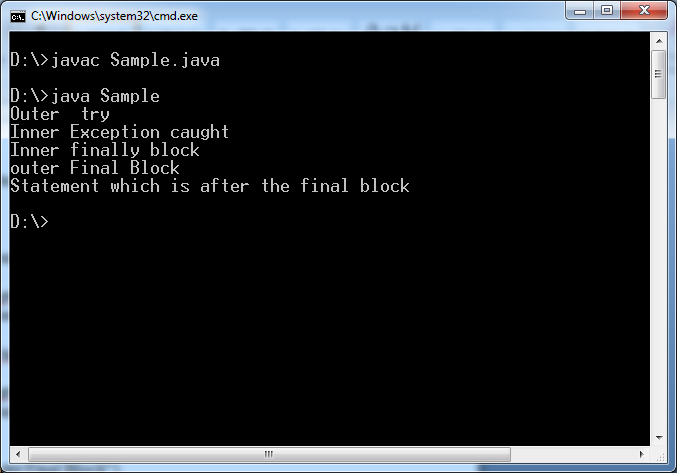
finally{

System.out.println("outer Final Block");

}

System.out.println("Statement which is after the outer final block");

}}



**2.5.** **if final block has return statement,The statements should not be written after the inner try/catch/finally statement. Otherwise we will get CE: Unreachable statement error.**

Example:1

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

System.out.println(m1());

}

public static int m1(){

try{

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

try{int a=20/0;}

catch(ArithmeticException e){

System.out.println("Inner Exception caught");

}

finally{ System.out.println("Inner finally block"); return 30;}

System.out.println("Statement which is after the inner final block");

}

catch(NullPointerException e){

System.out.println("Outer Exception caught");

}

finally{

System.out.println("outer Final Block");

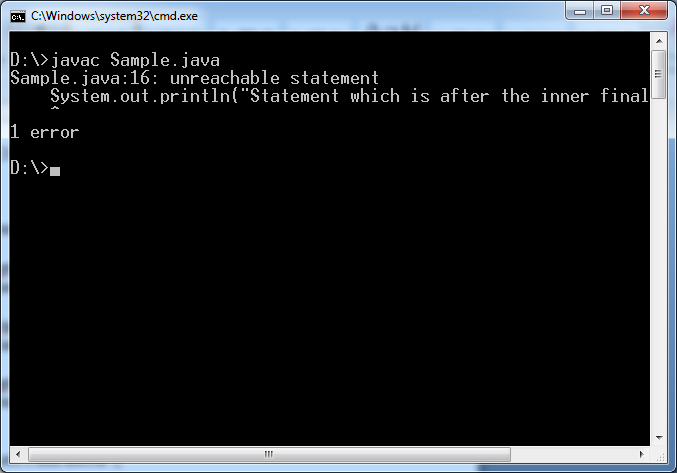
}

System.out.println("Statement which is after the outer final block");

return 50;

}

}



Example:2

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

System.out.println(m1());

}

public static int m1(){

try{

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

try{int a=20/0;}

catch(ArithmeticException e){

System.out.println("Inner Exception caught");

}

finally{ System.out.println("Inner finally block"); return 30;}

}

catch(NullPointerException e){

System.out.println("Outer Exception caught");

}

finally{

System.out.println("outer Final Block");

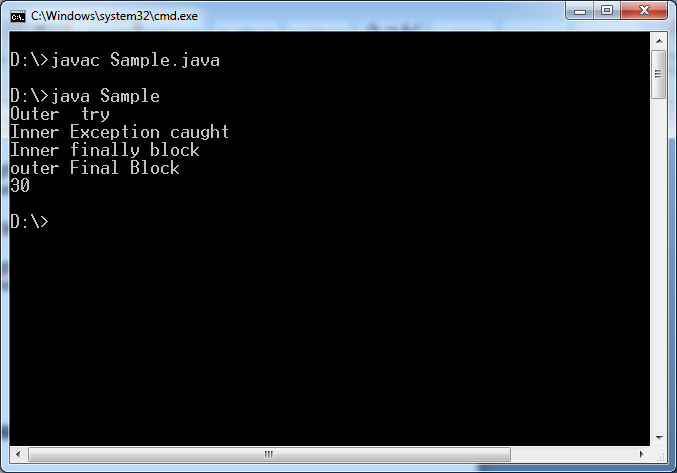
}

System.out.println("Statement which is after the outer final block");

return 50;

}

}



**2.6. If inner try/catch/finally block and outer try/catch/finally block has return statement then outer return statement has more priority than inner return statement. So outer return statement value is returned to calling methods.**

Example:

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

System.out.println("Returned Value:"+m1());

}

public static int m1(){

try{

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

try{int a=20/0;}

catch(ArithmeticException e){

System.out.println("Inner Exception caught");

}

finally{ System.out.println("Inner finally block"); return 30;}

}

catch(NullPointerException e){

System.out.println("Outer Exception caught");

}

finally{

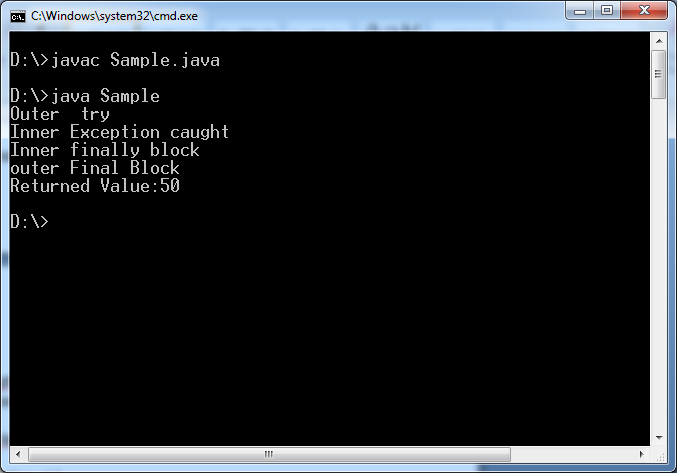
System.out.println("outer Final Block");

return 50;

}

}

}



**TRY WITH VARIABLES**

**1.if we initialize a local variable in try block, and if we access it in one of the catch blocks or after try/catch blocks it leads to**

**CE: “variable might not have been initialized”.**

**Because variable initialization statement is not definitely executed or may throw exception.**

Example:

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[]){

int a;

try{

a=20;

System.out.println("In Try block: "+ a);

}

catch(Exception e){

System.out.println("In Catch block:"+a);

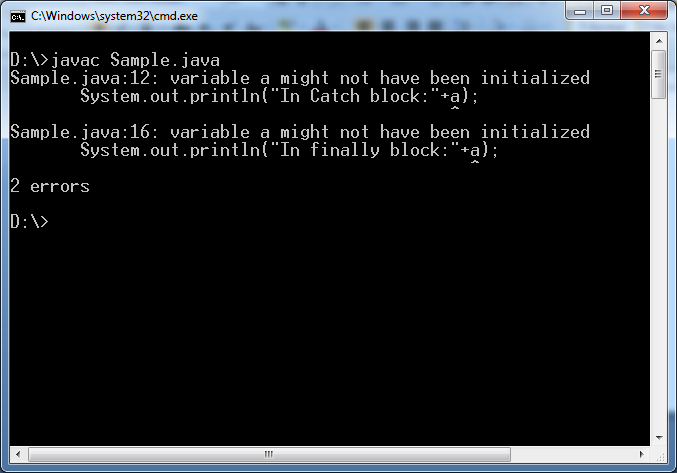
}

finally{

System.out.println("In finally block:"+a);

}

} }

****

**2.The variable is declared inside a method. The try,catch and finally block first must initialize variable in their blocks .Then only inside their blocks, the variable can be accessed.**

**Otherwise we get CE: variable might not have been initialized.**

**But we don’t get above CE in accessing static,non-static and parameter variables.**

**Example:1**

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int a;

try{

a=20;

System.out.println("In Try block: "+ a);

}

catch(Exception e){

a=40;

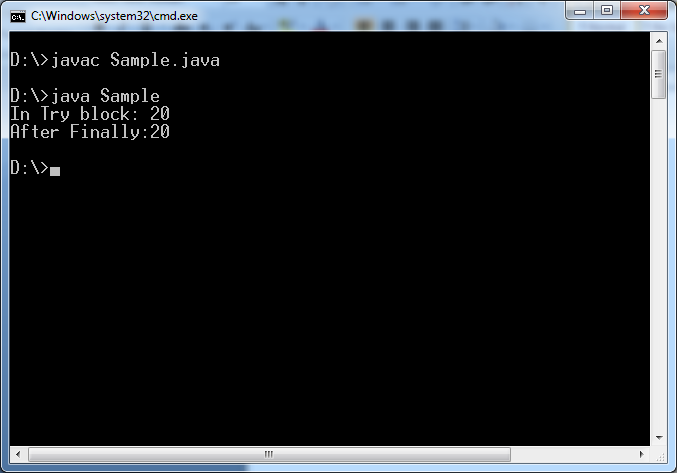
System.out.println("In Catch block:"+a);

}

System.out.println("After Finally:"+a);

}

}



Example:2

import java.util.Scanner;

import java.lang.\*;

class Sample{

public static void main(String arg[])

{

int a;

try{

a=20;

System.out.println("In Try block: "+ a);

}

catch(Exception e){

a=40;

System.out.println("In Catch block:"+a);

}

finally{

a=60;

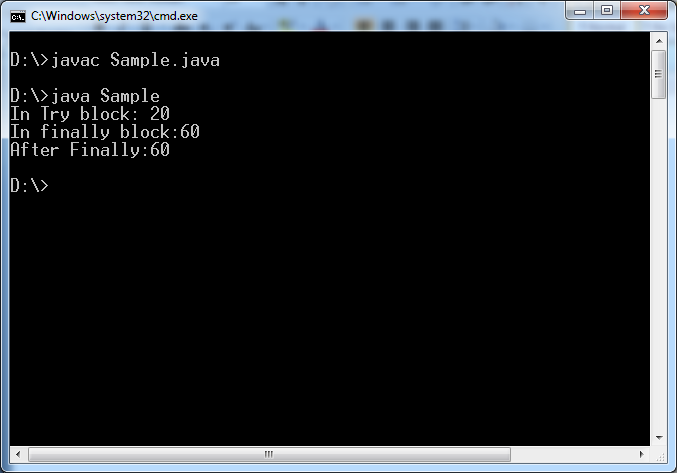
System.out.println("In finally block:"+a);

}

System.out.println("After Finally:"+a);

}

}



**Example:3**

import java.util.Scanner;

import java.lang.\*;

class Sample{

int a=100;

static int b=200;

public static void main(String arg[])

{

Sample s1=new Sample();

s1.m1(300);

}

public void m1(int c){

try{

System.out.println("In Try block: "+ a);

System.out.println("In Try block: "+ b);

System.out.println("In Try block: "+ c);

}

catch(ArithmeticException e){

System.out.println("In Catch block:"+a);

System.out.println("In Catch block:"+b);

System.out.println("In Catch block:"+c);

}

catch(NullPointerException e1){ System.out.println(a);}

finally{

System.out.println("In finally block:"+a);

System.out.println("In finally block:"+b);

System.out.println("In finally block:"+c);

}

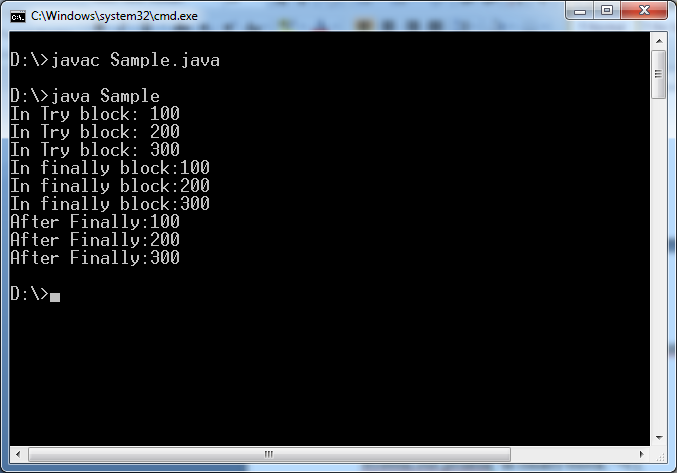
System.out.println("After Finally:"+a);

System.out.println("After Finally:"+b);

System.out.println("After Finally:"+c);

}

}



**EXCEPTION PROPAGATION BETWEEN CALLING METHOD AND CALLED METHOD**

The process of sending an exception from called method to calling method is called exception propagation.

**Case1:** If exception raised in called method and it is caught then exception is not propagated to calling method.

Example:

import java.util.Scanner;

import java.lang.\*;

class Sample{

int a=100;

static int b=200;

public static void main(String arg[])

{

Sample s1=new Sample();

s1.m1();

}

public void m1(){

try{

int a=20/0;

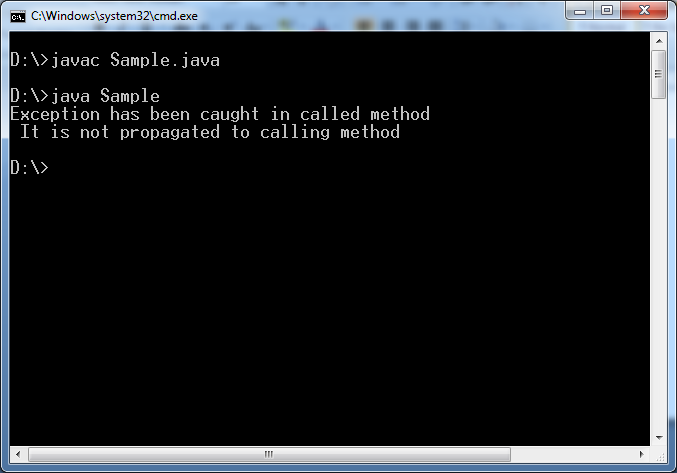
}

catch(ArithmeticException e){

System.out.println("Exception has been caught in called method\n It is not propagated to calling method");

}

} }



**Case2:-** if exception is not caught by called method, it is propagated to calling method. If calling method can ‘t caught it then only it is propagated to JVM.

Example1:

import java.util.Scanner;

import java.lang.\*;

class Sample{

int a=100;

static int b=200;

public static void main(String arg[])

{

Sample s1=new Sample();

try{s1.m1();}

catch(ArithmeticException e){System.out.println("Exception has been caught at calling method");}

}

public void m1(){

try{

int a=20/0;

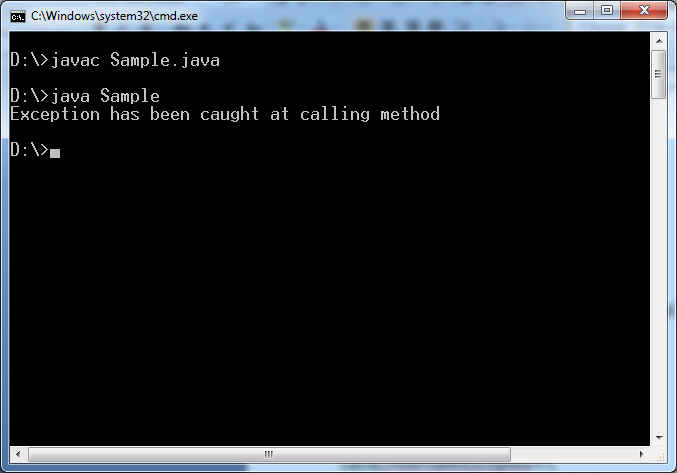
}

catch(NullPointerException e){

System.out.println("Exception has been caught in called method\n It is not propagated to calling method");

}

} }



Example2:

import java.util.Scanner;

import java.lang.\*;

class Sample{

int a=100;

static int b=200;

public static void main(String arg[])

{

Sample s1=new Sample();

S1.m1();

}

public void m1(){

try{

int a=20/0;

}

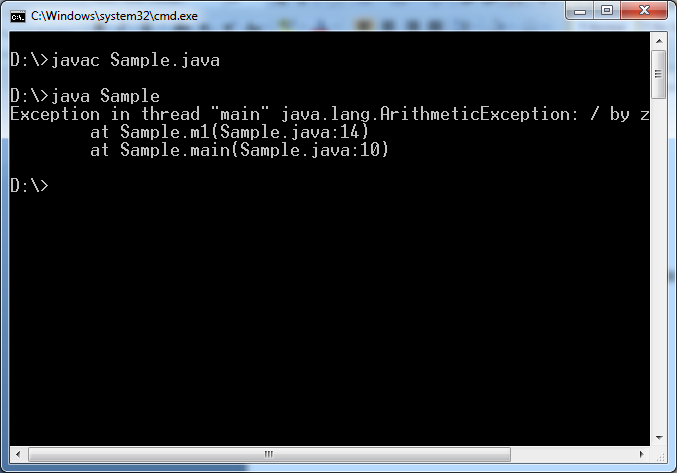
catch(NullPointerException e){

System.out.println("Exception has been caught in called method\n It is not propagated to calling method");

}

}

}



**THROW**

1. **throw:-** It is keyword. It is used to throw an Custom exception or Predefined Exception manually from method /constructor.

**Note1:** when The unchecked exception is thrown by throw keyword in method or constructor, The programmer may/may not write logic to handle exception in called method and the programmer may/may not report exception at called method.

**Note2:**- when checked exception is thrown by throw keyword in method or constructor, either the programmer should write a logic to handle exception in called method or the programmer should report that checked exception.

**Rule1:-**throw is transfer statement, so we can not place statements after throw statement directly compiler throws error:”unreachable statement”.

Void m1(){

Throw new ArithmeticException():

System.out.println(“suku”);

}

It leads to CE: Unreachablestatement.

If we want place statements after throw statement, we must place throw statements inside if condition.

Void m2(){

Throw new ArithmeticException();

If(true){ System.out.println(“suku”);}

}

**THROWS**

**1.throws:-** It is keyword . when checked/unchecked exception is thrown by throw keyword in method or constructor, either the programmer should write a logic to handle exception in called method or the programmer should report that checked exception/unchecked exception using throws keyword to its caller.

Example:

Void m1()

{

trhow new InrerryptedException();

}

**Case1:**The m1() method throw checked Exception, It must be handled in called method.

Class abc{

Public static void main(String args[]){

Try{

M1() ;

}

Catch(InterruptedException e){}

}

**Case2:** m1() method throws checked Exception. The m1() is called by main () method. the called method(main) should report about this checked exception to its caller(JVM) using throws keyword. Otherwise it leads to CE:”Unreported Exception” .

Class abc{

Public static void main(String args[])throws InterruptedException{

M1();

}

**Rules:**

**Rule-1:** with out throwing checked exception from try block we are not allowed to write catch block for it, compiler throws error:” exception never thrown from corresponding try statement”.

Example:

Void m1(){

Try{}

Catch(InterryptedException e){}

}

It leads to CE.

Void m1(){

Try{ throw new InteruptedException ();}

Catch(InterryptedException e){}

}

It does not leads to CE.

**Rule2 :** It is special case. We are allowed to place catch block for Exception and Throwable even though they are not thrown from try block because they are super classes of both checked and unchecked exception.

Example:

Void m1(){

Try{}

Catch( Throwable/Exception e){}

}

**CUSTOM EXCEPTION**

**1.Definition:-** the new class developed by a programmer deriving from either Throwable or any one of its subclass is called custom exception or user defined Exception.

According to business requirements programmer must develop their own exception class with suitable and meaning names relevant to project context.

Note:-

1. The programmer should not derive class from java.lang.Throwable. By creating like that , There are two problems.

Problem:-1 one new exception category under Throwable class will be created.

Problem:-2 This custorm exception is not caught by **catch(Exception e)** statement.

1. The programmer should not derive class from java.lang.Error because programmer is not writing this exception for handling errors in JVM internal logic.
2. The programmer derive class from java.lang.RuntimeException class; if we don’t want validate these exceptions handling by compiler.
3. Usually, it is recommended to derive custom exception class from **“Exception”** class.

**2.Procedure to develop custom exception class:**

**a.** Define public class which extends the java.lang.Exception class.

**b.** Define public no-arg and String parameter constructors with super() call.

No-arg constructor for creating exception object without message.

Parameterized constructor for creating exception object with message.

**Example:-**

Define custom exception invalidAmountException,InsufficientFundsException to handle wrong operations done by customer in deposit and withdraw operations.

TestCases are:

a.Throw invalidAmountException if user enter zero or –ve number in deposit or withdraw.

b.Throw InsufficientFundsException if user enter amt > balance in withdraw operation.

Public class InvalidAmountException extends Exception{

Public InvalidAmountException(){

Super();

}

Public InvalidAmountException(String msg){

Super(msg);

}

Public class InSufficientFundException extends Exception{

Public InSufficientFundException(){

Super();

}

Public InsufficientFundException(){

Super(msg);

}

}

Public class Bank{

Private double balance;

Public void deposit(double amt)

{

If(amt<=0)

{ throw new InvalidAmountException(“amount is less than or equal to zero”);}

Balance=balance+amt;

}

Public void withdraw( double amt)

{

If(amt<=0)

{ throw new InvalidAmountException(“amount is less than or equal to zero”);}

If(balance<amt){

Throw new InSuffientFundException(“Insufficient Fund”);}

Balance=balance-amt;

}

}

**IMPORTANT POINTS ABOUT EXCEPTIONS**

**Case-1:** If super class method is not throwing checked exception, subclass overriding method is not allowed to throw **checked** exception . It leads to CE:

Example:-1

Class A{

Public void m1(){}

}

Class B extends A{

Public void m1() throws Exception{}

}

It leads to CE.

Class C extends A{

Public void m1() throws ArithmeticException{}

}

It does not leads to CE because it is “checkedException”.

**Case-2:** If super class method throws checked exception, subclass overriding method may or may not throw this checked exception. If overriding method throws exception then exception must be same checked exception or its subclass. But exception should not be super class or sibling.

Example:

1.Class A{ void m1() throws InterruptedException{}}

2.Class B extends A{

Void m1() throws InterruptedException{}

}

3.class C extends A{

Void m1(){}

}

4. class D extends A{

Void m1()throws Exception {}

}

It leads to CE. Because Exception is super class.

5. class E extends A{

Void m1() throws ClassNotFoundException{}

}

It leads to CE. Because overriding method has sibling type.