**JAVA : EXCEPTION HANDLING**

Exception : an unwanted or unexpected type of event which disrupts the normal flow of program or an abnormal conditions.

The main purpose of the Exception handling is the gracefull termination of the program.

Exception Handling : We have to define an alternative way at the beginning to handle the exception and continue the flow of the program normally.

Runtime Stack Mechanism

Every java program contains atleast one thread i.e. main thread . And for every thread JVM will create one runtime stack . Every method call performed one entry will be stored in the stack and this entry is called stack frame or activation record. Once the method call will be completed the corresponding entry from the stack will be removed. And once the stack is empty i.e. all the method call is completed so before terminating the thread JVM will first destroy the Runtime Stack of main thread .

Default Exception Handling in Java

class Test { p s v m(String args[] ) { doStuff(); } p s v doStuff()

{ domoreStuff(); } p s v domoreStuff() { Sopln(10/0) ; } }

here the problem occurs as 10/0 the compiler will give the error .

Name : Arithmetic Exception , Description : / by zero ,

Stack Trace/Location : domoreStuff() , doStuff() , main() this is the complete stack trace .

Here the JVM will go to domoreStuff() as the error occurred here and he will ask oi where is the handling code then domoreStuff will tell i am not having any handling code then JVM will immediately terminated it abnormally and remove it from stack frame . Then JVM will approach to doStuff as it is the caller of the domoreStuff () so JVM will ask him do you have the handling code the doStuff method does not have the handling code thus it will terminate it as well and then it will go to the main method the caller of the doStuff and ask do you have the main method but it will tell no i also don't have the handling code . Thus the JVM will terminate the main method as well . and who is the caller of the main method ? JVM is the caller of the main method so it JVM is ultimately responsible to handle the exception . So for this JVM has kept an assistant known as the default exception Handling. It will only print the exception information to the console and terminate the program abnormally . and it will show the exception as : Exception in thread " main " : Arithmetic Exception division by zero and it will also print the stack trace.

Exception Hierarchy and Difference between Exception and Error

Throwable class acts as a root for java exception hierarchy . It contains two child classes 1 . Exception are recoverable by catch block 2 . Error : these are not recoverable it is caused by os.

checked and unchecked exception or any type of exception occurs at run time only . It's not advisable to say checked exception is shown at compile time as at compile it shows the syntactical mistakes.

import java.io.\*; //PrintWriter is present here

class Test

{ psvm(String []args)

{ PrintWriter pw = new PrintWriter("abc.text");

pw.hello("Hello");

}}So here compiler may not compile this code if abc.txt does not exist

so the error will it give : unexpected exception java.io.FileNotFoundException ; must be caught or declared to be thrown.

so you have to use try and catch or throws.

checked exception : is the exception checked by the compiler at runtime to check whether the user has handled the exception or not.

unchecked exception : class Test

{psvm(String[]args) { Sopln(10/0); }} it will compile fine but at runtime exception occurs . We will get Exception : in Thread Main. So such type of exception where compiler does not check the exception is called unchecked exception.

Errors are not caused by program it is caused by lack of system of resources. So being a programmer you cannot do anything.

So all errors are unchecked exception are by default unchecked exception.

Also runtime exception and its exception and its child classes are unchecked exception .

Excepts the above mentioned exceptions all exceptions are checked exception.

Difference b/w fully checked and partially checked exception

When you are visiting the shopping mall the security will check only parent but for the small kid it will not checked it is called partially checking .

Ex : IO exception and its child class are fully checked exception

But in the airport everyone will be checked and this type of checking means fully checked .

Ex : Exception is checked but its Runtime Exception i.e. child class are unchecked . Same with throwable as some of its child classes are unchecked .

The only two exceptions which are partially checked in java is throwable and Exception. Except these all are fully checked exceptions.

IOException ---> Checked (Fully checked )

RuntimeException ----> Unchecked

InterruptedException ----> Checked (Fully checked )

Error -----> Unchecked

Throwable -----> Checked (Partially checked )

ArithimeticException -----> Unchecked

NullPointer Exception ----> Unchecked

Exception -----> Checked (Partially checked )

FileNotFound Exception -----> Checked (Fully Checked)

Customized Exception Handling By using try-catch

without try-catch

class Test

{

psvm(---)

{

Sopln("Sta");

Sopln(10/0); // program will be terminated here only // risky code

Sopln("St-3");

}

}

o/p : Sta

RE : AE : / by Zero

so here is abnormal termination

with try-catch

The code which may rise an exception is called risky code . The risky code we have to take in try block.

try

{ Risky code } in this risky code if any exception raised we have to take the corresponding code inside the catch block

catch (Exception e )

{ Handling Code } and once handled we continue the code from below

class Test

{

psvm ()

{

Sopln("St-1");

try

{

Sopln(10/0); //risky code

}

catch(ArithmeticException e )

{

Sopln(10/2); //corresponding handling code

}

Sopln("St-3" ) ;

}}

o/p : St-1

5

St-3

It is normal termination and a graceful termination.

Control-Flow inside try-catch

try

{ st-1 ; st-2 ; st-3;

}

catch(anyexception e )

{ st-4; }

st-5;

o/p :

case1 : If there is no exception

st-1 , st-2 , st-3 , st-5 Normal Termination . if there is no exception catch will not execute .

case 2 : IF an exception matched at st-2 and corresponding catch block matched . St-1 , st-4 . Here keep in mind that once the control goes to catch block it will not go back to try block for executing st-3 . Ex : When the bus ticket is not available and you booked the train ticket and now that you have the train ticket why you purchase the bus ticket again .

So the o/p : st-1 , st-4 , st-5; Normal Termination.

Note : Inside the try block try to add as less code as possible . Assume you have 10,000 lines of code inside try block and the exception occurred at line 1 only so without executing the rest 9,999 lines of code the control will go to catch block . So it is advisable to include as less line of code inside the try block and don't include normal code only write the risky code in the try block.

So if you have multiple risky code in your source code you can use multiple try catch block

case 3 : IF an exception raised at st-2 and corresponding catch block not matched . Ex : You have heart problem and the hospital has a Eye Specialist . o/ p : st-1 and then Abnormal termination .

case 4 : If an exception occurred at st-4 and st-5.

o/p : Abnormal termination

Note : Don't feel that exceptions will be raised only in try block it can be raised inside the catch and even the final block as well.

st-1

try { statements .................... }

catch ( x e ) { --------------------- } here if an exception occurred at st-1 catch block is not at all responsible for handling the exception . Catch block is only associated with the try block not for any other statements.

Ex : Suppose you have security person at you home so if any robbery occurs at your house the security person is responsible to handle the robbers . But if the robbery happened at you neighbour house your security person is not responsible to handle the robbers.

If any statement rises an exception which is not inside the try block then it is always abnormal termination. So inside catch block you again use try catch .

Methods to print exception information

There are three methods available :

1 . e.printStackTrace()

2. e.toString () Sopln(e) ; Sopln(e.toString());

3. Sopln(e.getMessage());

These methods are available in Throwable class.

class Test

{

psvm

{

try{ Sopln(10/0); }

catch(AE e)

{

e.printStackTrace();

}}} so it will display the name of the exception : Description

and stack trace

class ExceptionHandling

{

 public static void main(String args[] )

 {

  try

  {

   System.out.println(10/0);

  }

  catch(ArithmeticException e )

  {

   e.printStackTrace();

  }

  System.out.println("Normal Termination");

 }

}

o/p : java.lang.ArithmeticException: / by zero

        at ExceptionHandling.main(ExceptionHandling.java:7)

Normal Termination.

So if you want complete information about the exception then highly recommended to go for printStackTrace exception .

if you only want the name of exception and its description then better to go for toString whenever we are trying to print the object reference internally toString method will be called automatically.

class Test

{

psvm

{

try{ Sopln(10/0); }

catch(AE e)

{

Sopln(e); // trying to print object reference

Sopln(e.toString()); //Calling it explicitly it will to do the same thing as the above code would do.

}}}

o/p :

java.lang.ArithmeticException: / by zero

Normal Termination

.getMessage() : It only gives description

class ExceptionHandling

{

 public static void main(String args[] )

 {

  try

  {

   System.out.println(10/0);

  }

  catch(ArithmeticException e )

  {

   //System.out.println(e);

   //internally toString is called automatically

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

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

  }

  System.out.println("Normal Termination");

 }

}

o/p : / by zero

Normal Termination.

The printStackTrace method gives the complete description of the exception so by default the exception handler always use the printStackTrace method to display the exception at the console.

e.printStackTrace method internally contains sopln statements.

try with multiple catch blocks

Suppose some one ask my name my answer was Yash . he ask my qualification my answer was yash . he asked about my skills my answer was yash . So what i am trying to tell is that for every different question there must be a different answers . Similarly, The way of handling exception varies from exception to exception . For every exception type seperate catch block must be required.

Ex :

try{ if any error occurred like AE , FNFE , SQLE}

catch(Exception e ) { it will give the same answer }

here for all exceptions there is same handling code . So this type of exception is not at all recommended .

try { ----- }

catch (AE e )

{ //perform alternative arithmetic exception }

catch(FNFE e )

{// use local file instead of remote file }

catch(SQLE e )

{ //use oracle db instead of mysql db}

other than the above mentioned exception if any other exceptions occur then the below catch block will be responsible .

catch(Exception e )

{ //default handling code }

So highly recommend to use this approach.

Loophole in try with multiple catch block

case1 :

try

{ Sopln(10/0); }

catch(Exception e)

{ Sopln(Exception);}

catch(ArithmeticException e) // here JVM will tell arithmetic exception is already handled why are you handling it again . Also its the child class of

Exception . So it should written before Exception i.e. child exception should written before parent exception .

{ Sopln("AE");}

case2 :

try

{ Sopln(10/0); }

catch(ArithmeticException e)

{ Sopln("AE");}

catch(Exception e)

{ Sopln(Exception);}

try

{ Sopln(10/0); }

catch(ArithmeticException e)

{ Sopln("AE");}

catch(ArithmeticException e)

{ Sopln("AE");}

when there are multiple catch blocks JVM will always consider catch block from top to bottom . Catch block parent exception can handle child class exception . The order is very important in catch block we should always first write the child class exception in catch block

case 1 : o/p ----> exception : java.lang.AE has already been caught. Compile time error. Parent to child exception not possible . So first child and then parent.

case 2 : o/p -----> AE . here if any exception other than arithmetic exception occurs then the second catch will take care of it.

case 3 : o/p ------> compile time error . As we cannot take multiple catch block to handle the same exception as when JVM will analyze the code it will see that the exception is already been resolved by the first catch block then why the second catch block is handling the exception again . So it will display the exception has already been caught.

Purpose and speciality of finally block

open db connection

read data from db

close db connection // deallocation code or cleanup code

Approach -1

try{ open db connection

read data from db

close db connection

}

catch(Exception e)

{//corresponding handling code}

suppose we open db and while reading data exception occurred and then the control goes to catch block and it handles the exception . But now who is going to close the db . So its not the better approach to do so .

So its never recommended to write the cleanup code inside the the try block because there is no gurantee that every statement inside the try block will be executed because if any exception occurred the control will go to the catch block and the control will never return to the try block again.

Approach - 2

try{ open db connection

read data from db

}

catch(Exception e)

{close db connection}

again it is not recommended as if there is no exception catch will not execute and thus db connection will not be executed.

So i want some place to maintain the cleanup code that will be executed always whether exception raised or not raised , whether the exception handled or not handled it should be always executed . In any case it should be executed . So for that we have the finally block.

So the recommended approach is

try

{ open db connection

read data }

catch (Exception e )

{ Handling code }

finally

{Cleanup code i.e. close db connection }

So finally block will be executed always whether exception raised or not raised , whether the exception handled or not handled it should be always executed . It will also be executed if there is an abnormal termination of the program . The biggest speciality of finally block is it will be executed always .

try { risky code }

catch { handling Code }

finally { cleanup code }

case 1 : IF there is no exception

try

{ Sopln("try"); }

catch (Exception e )

{ Sopln("catch"); }

finally { Sopln("finally" ); }

first try will be executed and as there is no error the control goes to finally block . o/p : try

finally

case 2 : IF an exception raised and handled.