**EXCEPTIONS**

**Definition:**

Exception is an Event that gets triggered when JVM is not able to execute a statement. It can be handled using try catch block

Program:

package com.qsp.ExceptionDemo;

public class Demo1 {

public static void main(String[] args) {

int i=10;

int j;

try

{

j=i/0;

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

}

catch(ArithmeticException e)

{

e.printStackTrace();

System.out.println("inside catch block");

}

}

}

Here, j=i/0 is triggering in try block so we are catching it in catch block

O/P  
java.lang.ArithmeticException: / by zero

inside catch block

at com.qsp.ExceptionDemo.Demo1.main(Demo1.java:9)

Note:

Stack trace means it will actually print what is exception, why is it arised and which line it got occurred

We should not put this line e.printStackTrace(); in system.out.println();

With in catch block

We can print the address of the exception type.

We can print Stack Trace

We can print our own message

Ex:

catch(ArithmeticException e)

{

System.out.println(e);// prints address of e

OR

e.printStackTrace();// prints stack trace

OR

System.out.println("inside catch block");//prints our own message

}

Program:

package com.qsp.ExceptionDemo;

public class Demo2 {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i=10;

int j;

try{

j=i/0;//exception occurs here

int k=Integer.parseInt("test");//compiler won’t executes this statement as exception occurs in above statement it will directly go to the appropriate catch block

}

catch(ArithmeticException e)

{

System.out.println("in the arthematic exception catch block");

e.printStackTrace();

}

catch(NumberFormatException e)

{

System.out.println("in the number format exception catch block");

e.printStackTrace();

}

}

}

Note: In this program java searches for the catch block which matches with the exception and executes the corresponding block

O/p:

in the arithmetic exception catch block

java.lang.ArithmeticException: / by zero

at com.qsp.ExceptionDemo.Demo2.main(Demo2.java:13)

NOTE:

Handling multiple statements that can generate Exceptions, use separate try –catch block

Look at the below example

Program

package com.qsp.ExceptionDemo;

public class Demo3 {

public static void main(String[] args) {

int i=10;

int j;

try{

System.out.println("inside arthematic exception try block");

j=i/0;

System.out.println("exiting arthematic exception try block");//this wont get executed

}

catch(ArithmeticException e)

{

System.out.println("in the arthematic exception catch block");

e.printStackTrace();

System.out.println("exiting the catch block");

}

try{

System.out.println("inside number format exception try block");

int k=Integer.parseInt("test");

System.out.println("exiting try block");//this wont get executed as above line gives exception

}

catch(NumberFormatException e)

{

System.out.println("in the number format exception catch block");

e.printStackTrace();

System.out.println("exiting the format exception catch catch block");

}

}

}

O/P:

inside arthematic exception try block

in the arthematic exception catch block

java.lang.ArithmeticException: / by zero

exiting the catch block

inside number format exception try block

at com.qsp.ExceptionDemo.Demo3.main(Demo3.java:9)

in the number format exception catch block

java.lang.NumberFormatException: For input string: "test"

at java.lang.NumberFormatException.forInputString(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at com.qsp.ExceptionDemo.Demo3.main(Demo3.java:20)

exiting the format exception catch catch block

NOTE:

Alternatively, multiple statements can be handled using nested try catch. First statement should be inside the inner try catch along with the corresponding catch block and the second statement should be inside outer try.

Program:

package com.qsp.ExceptionDemo;

public class Demo4 {

public static void main(String[] args) {

int i=10;

int j;

try{

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

try

{

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

int k=Integer.parseInt("test");

}

catch(NumberFormatException e)

{

System.out.println("inside inner catch block");

e.printStackTrace();

System.out.println("exiting the inner catch");

}

System.out.println("back to out try block");

j=i/0;

}

catch(ArithmeticException e)

{

System.out.println("inside outer catch");

e.printStackTrace();

}

}

}

O/P:

inside outer try block

inside inner try block

inside inner catch block

java.lang.NumberFormatException: For input string: "test"

at java.lang.NumberFormatException.forInputString(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at com.qsp.ExceptionDemo.Demo4.main(Demo4.java:12)

exiting the inner catch

back to out try block

inside outer catch

java.lang.ArithmeticException: / by zero

at com.qsp.ExceptionDemo.Demo4.main(Demo4.java:21)

We Cannot have like below

Program:

package com.qsp.ExceptionDemo;

public class Demo4 {

public static void main(String[] args) {

int i=10;

int j;

try{

j=i/0;// we cannot have like this as this throws exception and it will not execute inner try block

try

{

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

int k=Integer.parseInt("test");

}

catch(NumberFormatException e)

{

System.out.println("inside inner catch block");

e.printStackTrace();

System.out.println("exiting the inner catch");

}

}

}

catch(ArithmeticException e)

{

System.out.println("inside outer catch");

e.printStackTrace();

}

}

}

NOTE:

Finally, block is used along with try catch(Try-catch-finally) mostly or can be used along with try alone(try-finally). This block gets executed irrespective of a statement generating an exception or not , we can have

Try-catch-finally OR try-finally

Program:

package com.qsp.ExceptionDemo;

public class Demo5 {

public static void main(String[] args) {

int i=10;

int j;

try{

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

j=i/0;

}

catch(ArithmeticException e)

{

System.out.println("inside catch block");

}

finally

{

System.out.println("inside finally block");

}

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

}

}

O/P:

inside try block

inside catch block

inside finally block

i= 10

Program:

package com.qsp.ExceptionDemo;

public class Demo6 {

public static void main(String[] args) {

// TODO Auto-generated method stub

System.out.println("main method starts");

System.out.println(test());

System.out.println("main ends");

}

static String test()

{

int j;

int i=10;

try{

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

j=i/0;

}

catch(ArithmeticException e)

{

System.out.println("inside catch block");

}

finally

{

System.out.println("inside finally block");

}

return "from outside";

}

}

O/P:

main method starts

inside try block

inside catch block

inside finally block

from outside

main ends

Multicatch Block:

Program: this also gives input mismatch exception

public class Demo7 {

public static void main(String[] args) {

String str="12.56";

Scanner sc=new Scanner(str);//this will also give Input mismatch exception

//Scanner sc=new Scanner("hundred");

//Scanner sc=new Scanner("");// this is give no such element exception

String i;

//sc.close();this will give no illegal state exception as before using scanner we are closing the resource

try

{

i=sc.nextInt();//This will give input mis match exception as nextInt takes only int

System.out.println(i);

}

catch(InputMismatchException imexp)

{

System.out.println(imexp);

}

catch(NoSuchElementException nsexp)

{

System.out.println(nsexp);

}

catch(IllegalStateException inexp)

{

System.out.println(inexp);

}

}

}

Note:

Single try block can have mutliple catch blocks if 2 exception class in catch block are sub class and super class, first more specific than super class.

Elaboration:

Here, statement in try block gives mutiple exception, so there are mutiple catch blocks to catch the different exception thrown by the statement

Now elaboration for Statement” first more specific than super class.”

Ex:

Class fruit{

}

Class orange

{

}

(Orange o1)/ if orange object comes that is caught by o1 if any other object comes that is caught by fruit f1 so first is more specific /than super class

(fruit f1)

IMPORTANT POINTS ON MULTI CATCH BLOCK

1. Multi catch blocks should catch exception of any type and should not have relationship

2. You cannot have super and sub class in a multi catch block

3. When you need to handle exception when one exception is super class and another is sub class then class Exception , providing the

Catch handler for super class alone is enough as it will handle derived class also

Program:

package com.qsp.ExceptionDemo;

import java.util.Scanner;

import java.util.InputMismatchException;

import java.util.NoSuchElementException;

public class Demo11 {

public static void main(String[] args) {

// TODO Auto-generated method stub

String str=" ";

Scanner sc=new Scanner(str);

try{

sc.nextInt();

}

catch(NoSuchElementException | IllegalArgumentException exp)

{

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

}

}

}

Note:

Here, instead of using mutiple catch blocks we are using only one catch block with OR operator

Program:

package com.qsp.ExceptionDemo;

public class Demo12 {

public static void print(){

try{

String[] str={"Hello"};

System.out.println(str[1]);//this will throw arrayoutof bound exception

}

catch(ArrayIndexOutOfBoundsException aiob)//this will catch the exception and put it in the new exception

{

throw new RuntimeException (aiob);//so it will put the caught exception in the new exception and throw it

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

try{

print();

}

catch(RuntimeException re)

{

System.out.println(re.getClass());//this will give the class of the exception type

System.out.println(re.getCause());//this will return the cause which arised exception

}

}

}

O/P:

class java.lang.RuntimeException

java.lang.ArrayIndexOutOfBoundsException: 1

Note:

String[] str={"Hello"};

System.out.println(str[1]);//this will throw arrayoutof bound exception

This will throw exception because this is string array and “Hello” is stored in index 0, if you try to access indexes other than 0 it will throw array out of bound exception

Program

package com.qsp.ExceptionDemo;

import java .util.Scanner;

public class Demo9 {

public static void main(String[] args) {

try(Scanner sc=new Scanner(System.in))

{

int i=sc.nextInt();

}

catch(Exception e)

{

System.out.println("hi karma fellow");

}

}

}

Note:

we have resource inside the paranthesis of try block, compiler will internally convert try with resource to try finally block

if catch is present it will be retained , Once work is done resource are automatically closed, don’t close it explicitly.

Below is the program when exception does not occur:

Program

package com.qsp.ExceptionDemo;

public class Demo10 {

public static void main(String[] args) {

System.out.println("main starts");

System.out.println(test());

System.out.println("main ends");

}

static String test(){

int i=10; this statement does not arise exception and return statement is executed

int j;

try{

j=i/2;

return "pass"; this catch block will not get executed

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

}

}

O/P:

main starts

pass

main ends

Program:

package com.qsp.ExceptionDemo;

public class Demo10 {

public static void main(String[] args) {

System.out.println("main starts");

System.out.println(test());

System.out.println("main ends");

}

static String test(){

int i=10;

int j; this statement will arise exception and return statement is not executed

try{

j=i/0;

return "pass";

this catch gets executed

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

}

}

O/P:

main starts

java.lang.ArithmeticException: / by zero

fail

main ends

When we use finally block and when we use return statements in all blocks then , look at the below program

Program:

package com.qsp.ExceptionDemo;

public class Demo10 {

public static void main(String[] args) {

System.out.println("main starts");

System.out.println(test());

System.out.println("main ends");

}

static String test(){

int i=10;

int j;

try{ this wont get executed

j=i/2;

return "pass";

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

finally{

return "passfail";

}

}

}

O/P:

main starts

passfail

main ends

CORRECT

Different ways of using return ways

1.You can have return statement in try and catch block

try{

j=i/2;

return "pass";

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

2.Or in finally

try{

j=i/2;

}

catch(ArithmeticException e)

{

e.printStackTrace();

}

finally{

return "passfail";

3.or only outside

try{

j=i/2;

}

catch(ArithmeticException e)

{

e.printStackTrace();

}

finally{

}

return "passfail";

}

4.or in try-catch-finally block

try{

j=i/2;

return "pass";

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

finally{

return "passfail"

}

5.or in catch and finally

try{

j=i/2;

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

finally{

return "passfail";

}

Or in catch and outside

try{

j=i/2;

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

finally{

}

return "passfail";

WRONG:

Differnet possible ways of using incorrectly using return statement which gives unreacheable statement error

1. try{

j=i/2;

}

catch(ArithmeticException e)

{

e.printStackTrace();

}

finally{

return "passfail";

}

return "passfail";

}

2.or in try catch and outside catch block

try{

j=i/2;

return "pass";

}

catch(ArithmeticException e)

{

e.printStackTrace();

return "fail";

}

finally{

}

return "passfail";

}

Below is the table which consolidates the right and wrong usages of return statement

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Serial Number | Inside try block | Inside catch block | Inside finally block | Outside finally block | Result |
| 1 | ✔ | 🗙 | 🗙 | 🗙 | Compile time error |
| 2 | 🗙 | ✔ | 🗙 | 🗙 | Compile time error |
| 3 | ✔ | ✔ | 🗙 | 🗙 | Success |
| 4 | 🗙 | 🗙 | ✔ | ✔ | Success |
| 5 | 🗙 | 🗙 | 🗙 | ✔ | Success |
| 6 | ✔ | ✔ | 🗙 | ✔ | Compile time error |
| 7 | 🗙 | 🗙 | ✔ | ✔ | Compile time error |
| 8 | ✔ | 🗙 | ✔ | 🗙 | Success |
| 9 | ✔ | ✔ | ✔ | 🗙 | Success |
| 10 | 🗙 | ✔ | ✔ | ✔ | Compile time error |

IMPORTANT POINTS:

Exception Handling:

• Whenever jvm finds an abnormal statement an exception will be raised.

• An exception is an event triggered during execution which interrupts the normal execution of any statement

Handling the event is known as exception handling

• Whenever exception occurs jvm terminates the execution by throwing the exception message

• In order to complete the execution the exception should be handled by using try..catch block

• Always the try ..catch block should be written in sequence.

• try block alone is not allowed

• catch should take an argument of type exception

• try block should contain the statements which generates exception. Once an exception is occurred execution will go to catch block , remaining portion of the try block will never be executed

• In the catch body we can print the object or we can print the entire exception trace. After executing the catch body the execution continues from the remaining program

• In between try…catch block no other executable statements are allowed

**Stack Unwinding**

Program:

package com.qsp.ExceptionDemo;

public class Demo13 {

public static void main(String[] args) {

// TODO Auto-generated method stub

Demo13 d1=new Demo13();

d1.test1();

}

void test1(){

test2();

}

void test2()

{

test3();

}

void test3()

{

test4();

}

void test4(){

int i=10/0;

}

}

O/p:

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

at com.qsp.ExceptionDemo.Demo13.test4(Demo13.java:23)

at com.qsp.ExceptionDemo.Demo13.test3(Demo13.java:20)

at com.qsp.ExceptionDemo.Demo13.test2(Demo13.java:16)

at com.qsp.ExceptionDemo.Demo13.test1(Demo13.java:11)

at com.qsp.ExceptionDemo.Demo13.main(Demo13.java:8)

Note:

Exception causing statement is in test4 method, this statement created exception and thrown to search of try-catch block, first exception went to test3 and later test2 and test1 finally main method,nothing is catching this exception , so it finally thows exception

Test1()

Test2()

Test3()

Test4()

Test3()

Test4()

Test1()

Test2()

Main d1

main

java

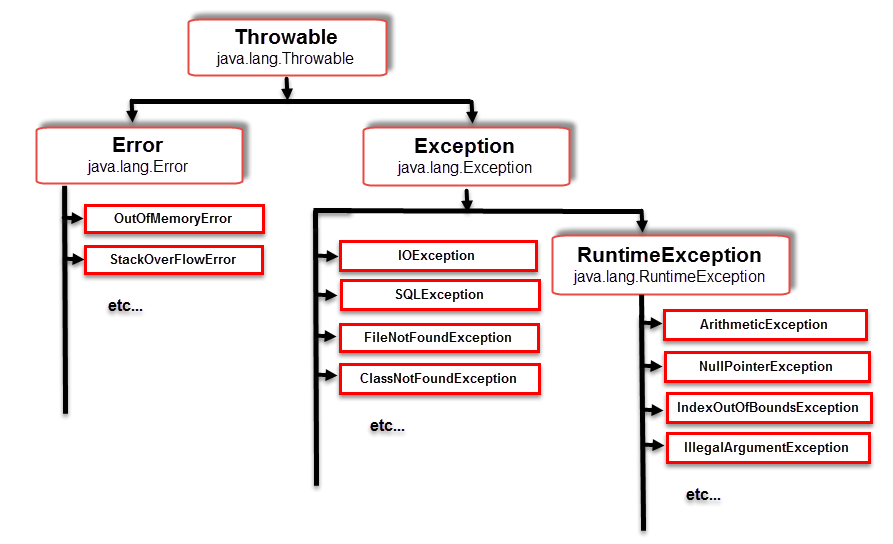
Stack Memory Heap Memory

Q. When does Stack Unwinding Happens?

A. While Executing any program the method of the classes will be loaded into the stack for the execution purpose.Stack will be loaded with methods in the orders the way its called. If the last entered method generates any exception then JVM looks for the handler in the same method, if no handler is found in the method then the exception propogrates to the called method , In the called method if the exception is not handled then the exception will propagates to its called method, finally the exception reaches main method.if the exception is not handled , then JVM forcibily removes all the method from the stack and terminates program execution .this is known as Stack Unwinding

When an exception is thrown it can be caught by reference variable of same class or its super class or its super class….in the catch block

BLOCK DIAGRAM:



Referring the above diagram we can know that Throwable is the super most class of all exception classes and its super class is Object class

Error class and Exception class are the two classes which extends thowable class.

under error class there are 3 exception class

1.Out of memory error

2.no suchelement error

3.stack over flow error

These error class cannot be handled

Exception class extends 4 exception class

1.runtime exception

2.class not found exception

3.IO exception  
4.SQL exception

Note:

If we get IO exception then we can catch argument of same class or its super class(Exception class) or its super class(throwable)

For null pointer exception use same class or runtime exception or exception or throwable

THROWABLE: it is the super most class of all exception classes and its super class is Object class

THROW: throw keyword is used to generate or throws an exception(existing exception class or user defined exception) in the program.

THROWS: its used to handle the exception

The throws declaration should be done in the method signature, the throws keyword should be used only for checked exceptions.

Syntax:

Thows Exceptionname

Exception in Java are classified into 2 types

1.Checked Exception

2.Unchecked Exception

Checked Exception:

An Exception where the compiler can check at the time of compilation is known as checked Exception

• Checked Exception should be handled in order to compile the program successfully.

• Checked Exception can be handled in 2 ways

1)Surrounding try-catch block

2)By using throws declaration statement

The throws declaration should be done in the method signature, the throws keyword should be used only for checked exceptions.

Unchecked Exception: Exceptions which are not been able to be identified by the compiler at the time of compilation is known as unchecked exceptions.

1 and 2 are Unchecked exception

3 are checked exception

throws keyword cannot be used for unchecked exception and it should be handled only through try,catch block.

• throw keyword is used to generate or throws an exception(existing exception class or user defined exception) in the program.

• We can develop our own exception class by inheriting one of Java exception classes.

Program:

package com.qsp.ExceptionDemo;

public class Demo16 {

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

Class c1=Class.forName("com.qsp.pack1.Demo16");

}

}

How to generate our own exceptions(user defined)?

Look the below Program

package com.qsp.ExceptionDemo;

import java.util.Scanner;

class InvalidAgeException extends RuntimeException{

InvalidAgeException(String str){

System.out.println(str);

}

}

public class Demo17 {

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter your age");

int age=sc.nextInt();

try{

if (age>=60 || age<=0)

{

// System.out.println("Invalid age: Please

//enter age b/w 1-60");

throw new InvalidAgeException("Invalid Age:Please enter age b/w 1-59");

}

}

catch(InvalidAgeException iae)

{

}

System.out.println("Age entered is " + age);

System.out.println("main method ends");

}

}

O/P:

Enter your age

62

Invalid Age:Please enter age b/w 1-59

Age entered is 62

main method ends

Note:

We created our own excetion called InvalidAgeException simply by extending with the runtime exception

class InvalidAgeException extends RuntimeException{

InvalidAgeException(String str){

System.out.println(str);

}

}