DAY-57 [EXCEPTION HANDLING-4]

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

'throws' keyword

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

--> In Java program if there is a possibility of rising checked exception then compulsorly we should handle it.

case-1

-------

EXAMPLE:

---------

import java.io.\*;

class Demo

{

public static void main(String[] args)

{

PrintWriter pw = new PrintWriter("abc.txt");

System.out.println("Hello World!");

}

}

OUTPUT:

-------

unreported exception FileNotFoundException; must be caught or declared to be thrown

PrintWriter pw = new PrintWriter("abc.txt");

Case-2

-------

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args)

{

Thread.sleep(10000);

}

}

OUTPUT:

--------

unreported exception InterruptedException; must be caught or declared to be thrown

Thread.sleep(10000);

NOTE: From the above examples we came to know that whenever checked exception arisied we must either caught is using try catch block or we must declared

it as thrown using 'throws' keyword.

1. handling checked exception using try and catch block:

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

case-3

------

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args)

{

try

{

Thread.sleep(10000);

}

catch (InterruptedException e)

{

System.out.println("exception caught..!");

}

}

}

OUTPUT:

-------

\*normal termination of the program.\*

2. handling the checked exception using throws keyword

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

case-4

------

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args) throws InterruptedException

{

Thread.sleep(10000);

}

}

OUTPUT:

-------

\*normal termination of the program.\*

NOTE: throws keyword is used to duck an exception.

--> throws keyword will not handle the exception rather it will duck the exception or propagate the exception to the caller.

--> throws keyword is used only for checked exception.using throws keyword on unchecked exception will not make any impact.

--> it will not lead to normal termination of the program it just helps for compiling the code successfully.

case-5

-------

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args) throws ArithmeticException

{

System.out.println(10/0);

}

}

OUTPUT:

-------

--> throws keyword is having no impact on unchecked exception.

NOTE: throws keyword we can use to delegate responsbility of exception handling to the caller[method,main,jvm].

case-6

-------

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args) throws InterruptedException

{

method1();

}

public static void method1() throws InterruptedException

{

method2();

}

public static void method2() throws InterruptedException

{

method3();

}

public static void method3() throws InterruptedException

{

Thread.sleep(10000);

}

}

OUTPUT:

-------

\*normal termination of the program.\*

NOTE: throws keyword is applicable for methods constructors.

EXAMPLE:

---------

import java.io.\*;

class Demo

{

public Demo() throws Exception

{

}

public void m1() throws Exception

{

}

public static void main(String[] args)

{

}

}

OUTPUT:

-------

\*normal termination of the program.\*

NOTE: throws keyword is not applicable for top-level class.

EXAMPLE:

--------

import java.io.\*;

class Demo throws Exception

{

public Demo() throws Exception

{

}

public void m1() throws Exception

{

}

public static void main(String[] args)

{

}

}

OUTPUT:

-------

Demo.java:2: error: '{' expected

class Demo throws Exception

^

NOTE: throws keyword is applicable only for throwable objects[CHECKED EXCEPTION ONLY]

EXAMPLE:

--------

import java.io.\*;

class Demo

{

public static void main(String[] args) throws Demo

{

}

}

OUTPUT:

-------

incompatible types: Demo cannot be converted to Throwable

public static void main(String[] args) throws Demo

^

EXAMPLE:

--------

import java.io.\*;

class Demo extends RuntimeException

{

public static void main(String[] args) throws Demo

{

}

}

OUTPUT:

-------

\*normal termination of the program.\*

EXAMPLE:

--------

import java.io.\*;

class Demo

{

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

{

throw new Exception();

}

}

OUTPUT:

-------

\*normal termination of the program.\*

EXAMPLE:

---------

import java.io.\*;

class Demo

{

public static void main(String[] args) throws Error

{

throw new Error();

}

}

OUTPUT:

-------

\*throw and throws keywords impacts only on exceptions and not on errors\*

conclusion:

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

try : to maintain the risky code of the program.

catch : to maintain exception handlind code of the program

finally : to maintain the clean up code of the program

throw : to excplicitly create the exception object and hand over to jvm.

throws : to delegate the responsibility of exception handling to the caller method [where caller method may be a method or constructor or jvm]

USER-DIFIEND EXCEPTION or CUSTOMIZED EXCEPTION

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

--> We can create our own exceptions by extending exception class

EXAMPLE:

--------

import java.util.\*;

class DistinctionException extends RuntimeException

{

DistinctionException(String exp\_msg)

{

super(exp\_msg);

}

}

class FirstClassException extends RuntimeException

{

FirstClassException(String exp\_msg)

{

super(exp\_msg);

}

}

class SecondClassException extends RuntimeException

{

SecondClassException(String exp\_msg)

{

super(exp\_msg);

}

}

class CustException

{

public static void main(String[] args)

{

System.out.println("welcome to grade checking app..!");

System.out.println("Enter your total percentage");

Scanner sc = new Scanner(System.in);

int percentage = sc.nextInt();

if (percentage>=75)

{

throw new DistinctionException("Distinction ....congrats");

}

else if (percentage>=60)

{

throw new FirstClassException("FirstClass ....still you have scope to improve");

}

else if (percentage>=35)

{

throw new SecondClassException("SecondClass ....work hard to get good marks..!");

}

else

{

System.out.println("Fail ... better luck next time...!");

}

}

}

OUTPUT:

-------

welcome to grade checking app..!

Enter your total percentage

65

Exception in thread "main" FirstClassException: FirstClass ....still you have scope to improve

at CustException.main(CustException.java:39)

RE-THROWING THE EXCEPTION:

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

--> The process of catching an exception and throwing it to the caller method is called as re-throwing the exception.

EXAMPLE:

--------

import java.io.\*;

class Demo1

{

public static void main(String[] args)

{

//String path = "C:\\Users\\SagarRam\\Desktop\\day-12pics\\8790234-6615085-image-a-90\_1548071159289.jpg";

String path = "abc.txt";

try

{

open(path);

}

catch (FileNotFoundException e)

{

e.printStackTrace();

}

}

public static void open(String path) throws FileNotFoundException

{

try

{

FileReader r = new FileReader(path);

}

catch (FileNotFoundException e)

{

System.out.println(e);

throw e;

}

}

}

OUTPUT:

-------

java.io.FileNotFoundException: abc.txt (The system cannot find the file specified)

java.io.FileNotFoundException: abc.txt (The system cannot find the file specified)

at java.io.FileInputStream.open0(Native Method)

at java.io.FileInputStream.open(FileInputStream.java:195)

at java.io.FileInputStream.<init>(FileInputStream.java:138)

at java.io.FileInputStream.<init>(FileInputStream.java:93)

at java.io.FileReader.<init>(FileReader.java:58)

at Demo1.open(Demo1.java:22)

at Demo1.main(Demo1.java:10)

LISKO'S Substitution principles (rule of Overriding)

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

--> If the parent class method throws the exception then the child class overridden method may throw same exception or may not trow any exception.

case-1

------

EXAMPLE:

--------

class Parent

{

public void play() throws RuntimeException

{

System.out.println("dancing");

}

}

class child extends Parent

{

public void play() throws Exception

{

System.out.println("playing");

}

}

class Demo

{

public static void main(String args[])

{

child c = new child();

c.play();

}

}

OUTPUT:

-------

Demo.java:10: error: play() in child cannot override play() in Parent

public void play() throws Exception

^

overridden method does not throw Exception

1 error

NOTE: parent and child class is present in is-a relationship hence the exception which they throws must be in is-a relationship only.

\* new features in exception handling \*

--> multiple catch block in a single line:

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

EXAMPLE:

--------

import java.util.\*;

class Demo2

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

try

{

System.out.println("enter the value for A");

int a = sc.nextInt();

System.out.println("enter the value for B");

int b = sc.nextInt();

System.out.println(a/b);

}

catch (ArithmeticException | InputMismatchException | NullPointerException e)

{

System.out.println(e);

}

catch(Exception e)

{

System.out.println(e);

}

}

}

OUTPUT:

-------

enter the value for A

10

enter the value for B

ASFSFAS

java.util.InputMismatchException

enter the value for A

10

enter the value for B

0

java.lang.ArithmeticException: / by zero

VALID and INVALID BLOCKS w.r.t EXCEPTION HANDLING:

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

1. invalid

----------

try

{

}

2. invalid

----------

try

{

}

try

{

}

catch ()

{

}

3. valid

---------

try

{

try

{

}

catch ()

{

}

}

catch ()

{

}

4. invalid

----------

catch ()

{

}

5.invalid

---------

catch ()

{

}

finally

{

}

6. valid

---------

try

{

}

catch ()

{

}

finally

{

}

7. invalid

-----------

try

{

}

finally

{

}

finally

{

}

8.valid

---------

catch (ArithmeticException | InputMismatchException | NullPointerException e)

9.invalid

----------

catch (FileNotFoundException | IOException e)

10. valid

-----------

try

{

try

{

}

catch ()

{

}

}

catch ()

{

}

finally

{

}