**LAB 8.1**

*Write a program – where divide a number with zero; run it and check behavior*

**package** cg.javaflp.practice;

**public** **class** ExceptionPractice {

**public** **static** **void** main(String[] args) {

**int** x = 10;

**int** y = 0;

**int** z = x/y;

System.*out*.println(z);

}

}

**OUTPUT**

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

at cg.javaflp.practice.ExceptionPractice.main(ExceptionPractice.java:9)

**Learning**

* What is exception object

**LAB 8.2**

*Now write the divide by zero code in some method of different class, call the method from main method, run it. Check the exception stack on console.*

**package** cg.javaflp.practice;

**public** **class** ExceptionPractice {

**public** **static** **void** main(String[] args) {

MyMath myMath = **new** MyMath();

**int** x = 10;

**int** y = 0;

**int** z = myMath.divide(x, y);

System.*out*.println(z);

}

}

**OUTPUT**

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

at cg.javaflp.practice.MyMath.divide(MyMath.java:48)

at cg.javaflp.practice.ExceptionPractice.main(ExceptionPractice.java:11)

**Learning**

* Exception call stack

**LAB 8.3**

*Write a program, in which an exception occur; handle the checked exception with Try-catch*

**package** cg.javaflp.practice;

**public** **class** ExceptionPractice {

**public** **static** **void** main(String[] args) {

MyMath myMath = **new** MyMath();

**int** x = 10;

**int** y = 0;

**int** z = 0;

**try** {

z = myMath.divide(x, y);

} **catch** (Exception e) {

System.*out*.println("exception occured");

}

System.*out*.println(z);

}

}

**OUTPUT**

exception occured

0

**Learning**

* Handle exception using try-catch block

**LAB 8.3 A**

*Modify 8.3 – use finally block as well and check how it works*

**package** cg.javaflp.practice;

**public** **class** ExceptionPractice {

**public** **static** **void** main(String[] args) {

MyMath myMath = **new** MyMath();

**int** x = 10;

**int** y = 0;

**int** z = 0;

**try** {

z = myMath.divide(x, y);

} **catch** (Exception e) {

System.*out*.println("exception occured");

} **finally** {

System.*out*.println();

}

System.*out*.println(z);

}

}

**OUTPUT**

exception occured

inside finally

0

Now modify this exercise and change y = 5; and run it again

**OUTPUT**

inside finally

2

**Learning**

* Finally will execute in all case whether exception occur or not

**LAB 8.4**

*Handle the exception of LAB 8.1 with throws*

Handling arithmetic exception in divide method of MyMath class, using throw

**public class MyMath implements IMath, ISplMath, Cloneable{**

**public** **int** divide(**int** firstNumber, **int** secondNumber) **throws**

ArithmeticException{

**return** firstNumber / secondNumber;

}

}

**Learning**

* Handle exception using throws

**LAB 8.5**

*Write a program, where class has many possibility of occurrence of an exception. Handle them using nested try catch block.*

**Learning**

* Nested try catch

**LAB 8.6**

*Write a program with a checked exception code; do not handle it and try to run the program.*

**LAB 8.7**

*Write a program, which have two classes – MainClass (having main method) and MyMath class (having a method called divide). These divide methods takes two numbers as argument and divide them and return the result. Now call divide method and pass two numbers (make sure that second number zero), so that divideByZero exception should occur at MyMath’s divide method.*

Now, run following five aspects, and check the behaviors -

* *do not handle exception at any place*
* *handle it at cls2 method using throws*
* *handle it at cls2 method using try catch and do not handle at calling class*
* *handle it at cls2 method using throws and try catch at calling method of cls1*
* *handle it using try catch at both cls2 and calling class*

*This exercise to understand the exception handling stack and how it traverses*

**Learning**

* Difference between checked and unchecked exceptions
* Checked exception are compulsory to handle

**LAB 8.8**

*Write a user-define exception, and throw it from some other class and handle it. Aim of this assignment is to understand that whether user defined exceptions are checked or unchecked and how do we handle them.*

**Steps:**

* Steps to realize / do the assignments

**TODO - Paste code snippet here**

**OUTPUT**

TODO - Paste Output of the program

**Learning:**

* How to write our own exception
* How to handle user defined exception
* User define exception (which extends

**LAB 8.9**

*Write a program to check simple assertion value. Run this program when assertion is disabled and when it is enabled.*

**package** cg.javaflp.practice;

/\*\*

\* This class is to check assertion functionality

\* Run this as java application (u will find assert not working)

\* then run as configure and set VM argument -ea

\*/

**public** **class** AssertionsPractice {

**public** **static** **void** main(String[] args) {

**int** x = 10;

**int** y = -5;

**int** z = **new** MyMath().divide(x, y);

**assert** z > 0;

System.*out*.println(z);

}

}

**OUTPUT**

**When run as application (assertion disabled)**

-2

**When run configure to VM argument as -ea (assertion enabled)**

Exception in thread "main" java.lang.AssertionError

at cg.javaflp.practice.AssertionsPractice.main(AssertionsPractice.java:15)

**Learning:**

* How to write assertion
* How and when to use it

**LAB 8.9 A**

*Modify 8.9 - use second style of writing assertion.*

**assert** z > 0 : "one of the numbers is negative";

**OUTPUT**

Exception in thread "main" java.lang.AssertionError: one of the numbers is negative

at cg.javaflp.practice.AssertionsPractice.main(AssertionsPractice.java:15)