**PROGRAM 9**

Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.

**SOURCE CODE**

//Custom exception class

class DivisionByZeroException extends Exception {

public DivisionByZeroException(String message) {

super(message);

}

}

public class CustomExceptionDemo {

// Method to perform division and throw custom exception if denominator is zero

static double divide(int numerator, int denominator) throws DivisionByZeroException {

if (denominator == 0) {

throw new DivisionByZeroException("Cannot divide by zero!");

}

return (double) numerator / denominator;

}

public static void main(String[] args) {

int numerator = 10;

int denominator = 0;

try {

double result = divide(numerator, denominator);

System.out.println("Result of division: " + result);

} catch (DivisionByZeroException e) {

System.out.println("Exception caught: " + e.getMessage());

} finally {

System.out.println("Finally block executed");

}

}

}

**OUTPUT**

Exception caught: Cannot divide by zero!

Finally block executed

**PROGRAM 10**

Develop a JAVA program to create a package named mypack and import & implement it in a suitable class

**SOURCE CODE**

package mypack;

public class MyPackageClass {

public void displayMessage() {

System.out.println("Hello from MyPackageClass in mypack package!");

}

// New utility method

public static int addNumbers(int a, int b) {

return a + b;

}

}

//Main program outside the mypack folder

import mypack.MyPackageClass;

//import mypack.\*;

public class PackageDemo {

public static void main(String[] args) {

// Creating an instance of MyPackageClass from the mypack package

MyPackageClass myPackageObject = new MyPackageClass();

// Calling the displayMessage method from MyPackageClass

myPackageObject.displayMessage();

// Using the utility method addNumbers from MyPackageClass

int result = MyPackageClass.addNumbers(5, 3);

System.out.println("Result of adding numbers: " + result);

}

}

**OUTPUT**

Hello from MyPackageClass in mypack package!

Result of adding numbers: 8

**PROGRAM 11**

Write a program to illustrate creation of threads using runnable class. (start method start each of the newly created thread. Inside the run method there is sleep() for suspend the thread for 500 milliseconds).

**SOURCE CODE**

class MyRunnable implements Runnable {

private volatile boolean running = true;

public void run() {

while (running) {

try {

// Suppress deprecation warning for Thread.sleep()

Thread.sleep(500);

System.out.println("Thread ID: " + Thread.currentThread().getId() + " is running.");

} catch (InterruptedException e) {

System.out.println("Thread interrupted.");

}

}

}

public void stopThread() {

running = false;

}

}

public class RunnableThreadExample {

public static void main(String[] args) {

// Create five instances of MyRunnable

MyRunnable myRunnable1 = new MyRunnable();

MyRunnable myRunnable2 = new MyRunnable();

MyRunnable myRunnable3 = new MyRunnable();

MyRunnable myRunnable4 = new MyRunnable();

MyRunnable myRunnable5 = new MyRunnable();

// Create five threads and associate them with MyRunnable instances

Thread thread1 = new Thread(myRunnable1);

Thread thread2 = new Thread(myRunnable2);

Thread thread3 = new Thread(myRunnable3);

Thread thread4 = new Thread(myRunnable4);

Thread thread5 = new Thread(myRunnable5);

// Start the threads

thread1.start();

thread2.start();

thread3.start();

thread4.start();

thread5.start();

// Sleep for a while to allow the threads to run

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Stop the threads gracefully

myRunnable1.stopThread();

myRunnable2.stopThread();

myRunnable3.stopThread();

myRunnable4.stopThread();

myRunnable5.stopThread();

}

}

**OUTPUT**

Thread ID: 17 is running.

Thread ID: 15 is running.

Thread ID: 14 is running.

Thread ID: 16 is running.

Thread ID: 18 is running.

**PROGRAM 12**

Develop a program to create a class MyThread in this class a constructor, call the base class constructor, using super and start the thread. The run method of the class starts after this. It can be observed that both main thread and created child thread are executed concurrently.

**SOURCE CODE**

class MyThread extends Thread {

// Constructor calling base class constructor using super

public MyThread(String name) {

super(name);

start(); // Start the thread in the constructor

}

// The run method that will be executed when the thread starts

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " Count: " + i);

try {

Thread.sleep(500); // Sleep for 500 milliseconds

} catch (InterruptedException e) {

System.out.println(Thread.currentThread().getName() + " Thread interrupted.");

}

}

}

}

public class ThreadConcurrentExample {

public static void main(String[] args) {

// Create an instance of MyThread

MyThread myThread = new MyThread("Child Thread");

// Main thread

for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " Thread Count: " + i);

try {

Thread.sleep(500); // Sleep for 500 milliseconds

} catch (InterruptedException e) {

System.out.println(Thread.currentThread().getName() + " Thread interrupted.");

}

}

}

}

**OUTPUT**

main Thread Count: 1

Child Thread Count: 1

main Thread Count: 2

Child Thread Count: 2

main Thread Count: 3

Child Thread Count: 3

main Thread Count: 4

Child Thread Count: 4

main Thread Count: 5

Child Thread Count: 5