| Experiment No. 13 |
| --- |
| To handle runtime errors using Exception Handling to keep a program running smoothly |
| Date of Performance:19/09/24 |
| Date of Submission:26/09/24 |

**Aim:-** To handle runtime errors using Exception Handling to keep a program running smoothly

**Objective:-** Develop a Java program that demonstrates exception handling by performing the division of two numbers input by the user. The program should use try, catch, finally, throw, and throws to handle potential exceptions, such as division by zero and invalid input. The try block should contain the code that might throw exceptions, catch blocks should handle specific exceptions, finally should execute cleanup code, throw should manually throw an exception for invalid input, and throws should be used in method signatures to indicate the exceptions that might be thrown.

**Theory :-** An exception is an issue (run time error) that occurred during the execution of a program. When an exception occurred the program gets terminated abruptly and, the code past the line that generated the exception never gets executed.

Java provides us the facility to create our own exceptions which are basically derived classes of Exception. Creating our own Exception is known as a custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user needs. In simple words, we can say that a User-Defined Exception or custom exception is creating your own exception class and throwing that exception using the ‘throw’ keyword.

For example, MyException in the below code extends the Exception class.

Why use custom exceptions?

Java exceptions cover almost all the general types of exceptions that may occur in the programming. However, we sometimes need to create custom exceptions.

Following are a few of the reasons to use custom exceptions:

To catch and provide specific treatment to a subset of existing Java exceptions.

Business logic exceptions: These are the exceptions related to business logic and workflow. It is useful for the application users or the developers to understand the exact problem.

In order to create a custom exception, we need to extend the Exception class that belongs to java.lang package.

**Code:**

import java.util.Scanner;

public class Exp13 {

static void checkAge(int age) throws ArithmeticException {

if (age < 18) {

throw new ArithmeticException("Access Denied - You must be at least 18 years old.");

} else {

System.out.println("Access granted - You are old enough");

}

}

public static double divide(int numerator, int denominator) throws ArithmeticException {

if (denominator == 0) {

throw new ArithmeticException("Division by zero is not allowed.");

}

return (double) numerator / denominator;

}

public static void main(String[] args) {

System.out.println("Output of Code 1 : ");

try {

System.out.println("try Block::Begin");

int myArray[] = new int[5];

myArray[5] = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Arithmetic Exception :: Divide by zero!!");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBounds :: Accessed index out of bounds");

} catch (Exception e) {

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

}

System.out.println("Rest of the code");

System.out.println("Output of code 2 : ");

try {

int a[] = new int[5];

System.out.println(a[10]);

} catch (ArithmeticException e) {

System.out.println("ArithmeticException Occurs");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("ArrayIndexOutOfBoundsException occurs");

} catch (Exception e) {

System.out.println("ParentException occurs");

}

System.out.println("rest of the code");

System.out.println("Output of code 3 : ");

try {

checkAge(15);

} catch (ArithmeticException e) {

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

}

System.out.println("Output of code 4 : ");

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter the numerator: ");

int numerator = Integer.parseInt(scanner.nextLine());

System.out.print("Enter the denominator: ");

int denominator = Integer.parseInt(scanner.nextLine());

double result = divide(numerator, denominator);

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

} catch (NumberFormatException e) {

System.out.println("Invalid input: Please enter valid integers.");

} catch (ArithmeticException e) {

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

} finally {

System.out.println("Program execution completed.");

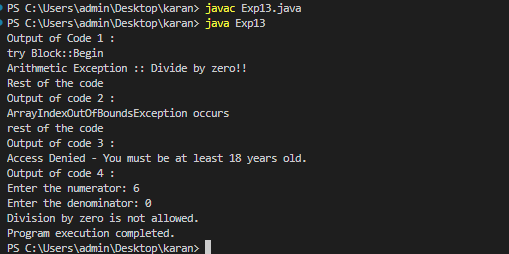
}

scanner.close();

}

}

**Output:**

****

**Conclusion:** In Java, the throw keyword is used to explicitly throw an exception from a method or block of code, while throws is used in a method signature to declare that a method may throw one or more exceptions, allowing calling methods to handle those exceptions.