|  |
| --- |
| Experiment No. 9 |
| Implement a program on Exception handling. |
| Date of Performance: |
| Date of Submission: |

**Aim:** Implement a program on Exception handling.

**Objective**: To able handle exceptions occurred and handle them using appropriate keyword

## Theory:

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

## public class JavaExceptionExample{

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

## try{

## //code that may raise exception

## int data=100/0;

## }catch(ArithmeticException e){System.out.println(e);}

## //rest code of the program

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

## }

## }

**Output:**

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

rest of the code...

**Code:**

import java.util.Scanner;

public class ExceptionHandlingExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

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

int numerator = scanner.nextInt();

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

int denominator = scanner.nextInt();

int result = divide(numerator, denominator);

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

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero is not allowed.");

} catch (java.util.InputMismatchException e) {

System.out.println("Error: Please enter valid integer values.");

} finally {

scanner.close();

}

public static int divide(int numerator, int denominator) {

if (denominator == 0) {

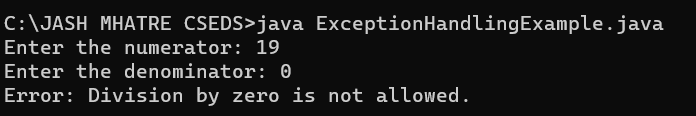
throw new ArithmeticException("Division by zero is not allowed.");

}

return numerator / denominator;

}

}



## Conclusion:

Exception handling is a critical aspect of programming that allows developers to manage and respond to unexpected or exceptional situations that can occur during code execution.implementing exception handling is a fundamental practice in software development that enhances the robustness, maintainability, and reliability of applications. It allows for graceful error recovery, custom error management, and structured error handling