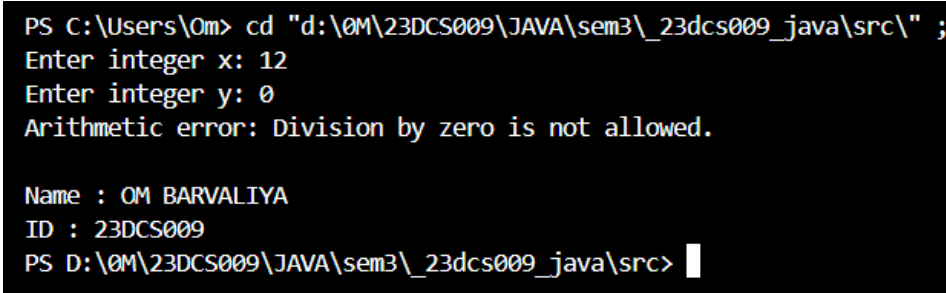
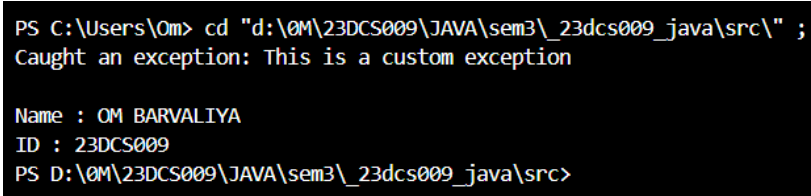


PART-V Exception Handling

No.	Aim of the Practical
24.	<p><u>AIM :</u> Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.</p> <p><u>PROGRAM CODE :</u></p> <pre>import java.util.Scanner; public class Prac_24 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); try { System.out.print("Enter integer x: "); int x = Integer.parseInt(scanner.nextLine()); System.out.print("Enter integer y: "); int y = Integer.parseInt(scanner.nextLine()); int d = x / y; System.out.println("Division " + d); } catch (NumberFormatException e) { System.out.println("Invalid input: Please enter valid integers."); } catch (ArithmeticException e) { System.out.println("Arithmetic error: Division by zero is not allowed."); } finally { scanner.close(); System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 "); } } }</pre> <p><u>OUTPUT:</u></p>  <p style="text-align: center;"><u>OUTPUT: PRACTICAL-24</u></p>

	<p><u>CONCLUSION:</u></p> <p>This code is a simple console application that prompts the user to input two integers and performs division. It includes error handling for invalid input and division by zero. The Scanner object is used for input and is properly closed in the finally block. The program also prints a name and ID at the end. This ensures the program handles common input errors gracefully while demonstrating basic exception handling in Java.</p>
25.	<p><u>AIM :</u> Write a Java program that throws an exception and catch it using a try-catch block.</p> <p><u>PROGRAM CODE :</u></p> <pre>public class Prac_25 { public static void main(String[] args) { try { throw new Exception("This is a custom exception"); } catch (Exception e) { System.out.println("Caught an exception: " + e.getMessage()); } System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 "); } }</pre> <p><u>OUTPUT:</u></p>  <p style="text-align: center;"><u>OUTPUT: PRACTICAL-25</u></p> <p><u>CONCLUSION:</u></p> <p>This java code demonstrates basic exception handling. It intentionally throws a custom exception and catches it, printing the exception message. The program then prints a name and ID. This example illustrates how to use try-catch blocks to manage exceptions in Java. It ensures that even when an exception occurs, the program continues to execute subsequent statements.</p>
26.	<p><u>AIM :</u> Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).</p>

PROGRAM CODE :

```
import java.io.IOException;
import java.sql.SQLException;

class CustomCheckedException extends Exception {
    public CustomCheckedException(String message) {
        super(message);
    }
}

class CustomUncheckedException extends RuntimeException {
    public CustomUncheckedException(String message) {
        super(message);
    }
}

public class Prac_26 {

    public static void methodThrowingCheckedException() throws
    CustomCheckedException {
        throw new CustomCheckedException("This is a custom checked exception");
    }

    public static void methodThrowingUncheckedException() {
        throw new CustomUncheckedException("This is a custom unchecked exception");
    }

    public static void methodThrowingStandardCheckedExceptions() throws
    IOException, SQLException {
        throw new IOException("This is an IOException");
    }

    public static void methodThrowingStandardUncheckedExceptions() {
        throw new NullPointerException("This is a NullPointerException");
    }

    public static void main(String[] args) {
        try {
            methodThrowingCheckedException();
        } catch (CustomCheckedException e) {
            System.out.println("Caught checked exception: " + e.getMessage());
        }

        try {
            methodThrowingStandardCheckedExceptions();
        } catch (IOException | SQLException e) {
            System.out.println("Caught standard checked exception: " + e.getMessage());
        }
    }
}
```

```

    }

    try {
        methodThrowingUncheckedException();
    } catch (CustomUncheckedException e) {
        System.out.println("Caught unchecked exception: " + e.getMessage());
    }

    try {
        methodThrowingStandardUncheckedExceptions();
    } catch (NullPointerException | ArithmeticException e) {
        System.out.println("Caught standard unchecked exception: " + e.getMessage());
    }
    System.out.print("\nName : OM BARVALIYA \nID : 23DCS009 ");
}
}

```

OUTPUT:

```

PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\" ;
Caught checked exception: This is a custom checked exception
Caught standard checked exception: This is an IOException
Caught unchecked exception: This is a custom unchecked exception
Caught standard unchecked exception: This is a NullPointerException

Name : OM BARVALIYA
ID : 23DCS009
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src>

```

OUTPUT: PRACTICAL-26

CONCLUSION:

This Java code demonstrates handling both checked and unchecked exceptions. It uses multiple try-catch blocks to catch custom and standard exceptions, printing appropriate messages. This ensures robust error handling and program continuity. The program concludes by printing the author's name and ID. This example effectively illustrates exception management in Java.