

Course Name:	Object Oriented Programming	Semester:	III
Date of Performance:	22 / 09 / 2025	Batch No:	Batch A1
Faculty Name:	Prof. Amrita Naiksatam	Roll No:	20
Faculty Sign & Date:		Grade/Marks:	___/15

Experiment No: 6
Title: Exception Handling in Java

Aim and Objective of the Experiment:
Learn the how to handle exceptions in Java

COs to be achieved:
CO3: Define exceptions and use I/O streams.

Tools used:
VS Code

Theory:
(About types of exceptions, the methods used for exceptions and keywords used for exception handling)

Code:
<ol style="list-style-type: none"> The program takes two numbers from users in the command prompt, Num1 and Num2. The division of Num1 and Num2 is displayed in the Result. If Num1 or Num2 were not an integer, the program would throw a NumberFormatException. If Num2 were Zero, the program would throw an Arithmetic Exception and Display the exception in the result. <ol style="list-style-type: none"> Using try and multiple catches. Using multiple exceptions in a single catch block. Create a user defined exception subclass TimeException with necessary constructors and overridden toString method. Write a program which accepts two integers with time in minutes and seconds and find the sum. It throws an object of the TimeException class if the value exceeds 60seconds otherwise it displays the total time. On printing, the exception object should display an exception name, appropriate message for exception

Q1)**a.****Program:**

```
import java.util.Scanner;

public class Q1a
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Num1: ");
        String input1 = sc.nextLine();
        System.out.print("Enter Num2: ");
        String input2 = sc.nextLine();

        try
        {
            int num1 = Integer.parseInt(input1);
            int num2 = Integer.parseInt(input2);
            int result = num1 / num2;
            System.out.println("Result: " + result);
        }
        catch (NumberFormatException e)
        {
            System.out.println("Invalid number format");
        }
        catch (ArithmeticException e)
        {
            System.out.println("Division by zero is not allowed");
        }
    }
}
```

Output:

```
D:\Programming\Java\Exception Handling>java Q1a
Enter Num1: 20
Enter Num2: 5
Result: 4

D:\Programming\Java\Exception Handling>java Q1a
Enter Num1: 5
Enter Num2: 20
Result: 0

D:\Programming\Java\Exception Handling>java Q1a
Enter Num1: 20
Enter Num2: Five
Invalid number format

D:\Programming\Java\Exception Handling>java Q1a
Enter Num1: 42
Enter Num2: 0
Division by zero is not allowed

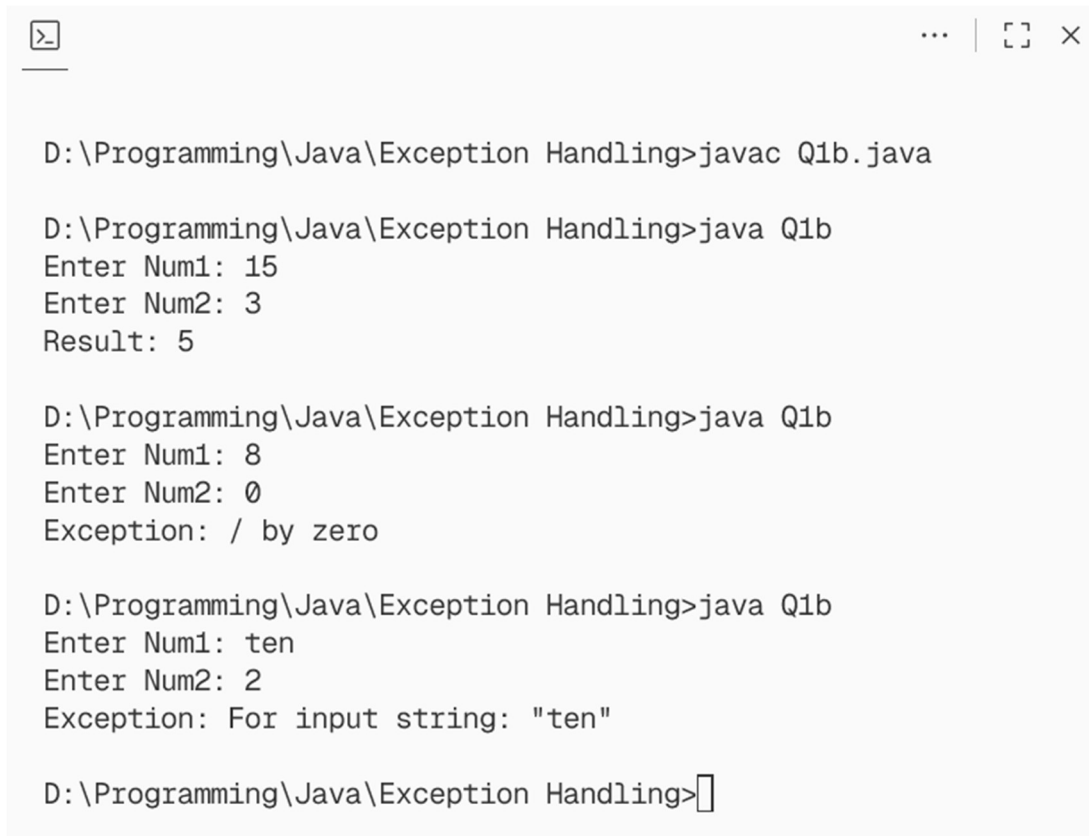
D:\Programming\Java\Exception Handling>
```

b.**Program:**

```
import java.util.Scanner;

public class Q1b
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Num1: ");
        String input1 = sc.nextLine();
        System.out.print("Enter Num2: ");
        String input2 = sc.nextLine();
```

```
try
{
    int num1 = Integer.parseInt(input1);
    int num2 = Integer.parseInt(input2);
    int result = num1 / num2;
    System.out.println("Result: " + result);
}
catch (NumberFormatException | ArithmeticException e)
{
    System.out.println("Exception: " + e.getMessage());
}
}
```

Output:

```
D:\Programming\Java\Exception Handling>javac Q1b.java

D:\Programming\Java\Exception Handling>java Q1b
Enter Num1: 15
Enter Num2: 3
Result: 5

D:\Programming\Java\Exception Handling>java Q1b
Enter Num1: 8
Enter Num2: 0
Exception: / by zero

D:\Programming\Java\Exception Handling>java Q1b
Enter Num1: ten
Enter Num2: 2
Exception: For input string: "ten"

D:\Programming\Java\Exception Handling>
```

Q2**Program:**

```
import java.util.Scanner;

class TimeException extends Exception
{
    TimeException(String msg)
    {
        super(msg);
    }

    public String toString()
    {
        return "TimeException: " + getMessage();
    }
}

public class Q2
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter minutes: ");
        int minutes = sc.nextInt();
        System.out.print("Enter seconds: ");
        int seconds = sc.nextInt();

        try
        {
            if (seconds > 60)
            {
                throw new TimeException("Seconds cannot exceed 60");
            }
            int totalMinutes = minutes + (seconds / 60);
            int totalSeconds = seconds % 60;
            System.out.println("Total Time: " + totalMinutes + " minutes " +
                totalSeconds + " seconds");
        }
        catch (TimeException e)
        {
            System.out.println(e);
        }
    }
}
```

Output:

```
D:\Programming\Java\Exception Handling>javac Q2.java

D:\Programming\Java\Exception Handling>java Q2
Enter minutes: 3
Enter seconds: 45
Total Time: 3 minutes 45 seconds

D:\Programming\Java\Exception Handling>java Q2
Enter minutes: 2
Enter seconds: 75
TimeException: Seconds cannot exceed 60

D:\Programming\Java\Exception Handling>
```

Post Lab Subjective/Objective type Questions:

1. Compare throw and throws.

Ans:

throw: Used inside a method to actually throw an exception.

Example: `throw new ArithmeticException("Divide by zero");`

throws: Used in the method declaration to specify which exceptions a method might throw.

Example: `void myMethod() throws IOException`

2. What are the advantages of using exception handling?

Ans:

- Provides a way to separate error-handling code from normal code.
- Avoids abnormal termination of the program.
- Makes debugging easier by providing clear exception messages.
- Allows grouping and differentiating error types.
- Ensures program robustness and reliability.

3. What is the use of finally block?

Ans:

- The **finally** block is always executed whether an exception occurs or not.
- Typically used to release resources like files, database connections, or closing scanners.
- Ensures cleanup code runs regardless of success/failure of **try**.

4. Suppose the statement2 causes an exception in following try-catch block:

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch(Exception1 e1) {  
}  
catch(Exception2 e2){  
}  
  
statement4;
```

Answer the following questions:

- Will statement3 be executed?

Ans: No. Once an exception occurs in statement2, control jumps to the corresponding catch.

- If the exception is not caught, will statement4 be executed?

Ans: No. If not caught, the program terminates abnormally and control never reaches statement4.

- If the exception is caught in the catch block, will statement4 be executed?

Ans: Yes. After handling the exception, normal flow resumes, so statement4 executes.

- If the exception is passed to the caller, will the statement4 be executed?

Ans: No. If the exception is not handled in this method and is thrown to the caller, control does not reach statement4.

Conclusion:

Exception handling in Java helps to manage runtime errors gracefully without abrupt termination. Using try, catch, throw, throws, and finally, we can separate error-handling logic, create user-defined exceptions, and ensure program robustness and reliability.

Signature of faculty in-charge with Date: