Lab 04: Exception Handling

Objectives:

- Understand the concept of exceptions and their importance in robust programming.
- Learn how to handle exceptions using `try`, `catch`, `finally` blocks.
- Explore different types of exceptions in Java.
- Write programs that gracefully handle potential exceptions.

Theory:

- **Exceptions:** Unexpected events that occur during program execution, disrupting the normal flow.
- **Exception Handling:** A mechanism to prevent program crashes and provide informative messages.
- **`try-catch-finally` Blocks:**
  - `try`: Encloses code that might throw exceptions.
  - `catch`: Handles specific types of exceptions.
  - `finally`: Executes code regardless of whether an exception is thrown.

Example:

```
import java.util.Scanner;

public class DivisionExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int numerator, denominator;

        try {
            System.out.print("Enter numerator: ");
            numerator = scanner.nextInt();
            System.out.print("Enter denominator: ");
            denominator = scanner.nextInt();



            int result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by

 zero is not allowed.");
        } finally {
            scanner.close();
        }
    }
}
```

Exercises:

1. Divide by Zero:

- Create a program that prompts the user for two numbers and divides them.
- Use `try-catch` to handle the `ArithmeticException` that might occur if the denominator is zero.

2. Array Index Out of Bounds:

- Create a program that accesses an array element using an index provided by the user.
- Use `try-catch` to handle the `ArrayIndexOutOfBoundsException` that might occur if the index is invalid.

3. File Not Found:

- Write a program that reads data from a file.
- Use `try-catch` to handle the `FileNotFoundException` that might occur if the file doesn't exist.

4. Input Mismatch:

- Write a program that reads an integer from the user using `Scanner`.
- Use `try-catch` to handle the `InputMismatchException` that might occur if the user enters a non-integer value.

5. Custom Exception:

- Create a custom exception class named `NegativeNumberException` that extends `Exception`.
- Write a program that prompts the user for a number and throws a `NegativeNumberException` if the number is negative.
- Use `try-catch` to handle this custom exception.

Additional Tips:

- Practice using different types of exceptions (e.g., `NullPointerException`, `NumberFormatException`).
- Experiment with nested `try-catch` blocks for handling multiple exceptions.
- Consider using `finally` blocks for tasks that must be executed regardless of exceptions.

Roll No: _____

Name: _____

Subject: _____

Teacher Name:_____