Experiment 3

Student Name: Akshara Chauhan
UID: 23BCS11410
Branch: CSE
Section/Group: KRG_2B
Semester: 5th
Date of Performance: 20/09/25
Subject Name: PBLJ
Subject Code: 23CSH-304

1. Aim:

To design and implement Java programs with exception handling to validate user input, simulate ATM operations, and manage university course enrollment.
• To handle runtime errors using try-catch blocks, throw custom exceptions, and ensure program robustness.

- **Part A – Easy Level:**

• To create a Java program that calculates the square root of a number entered by the user.
• To handle invalid inputs such as negative numbers or non-numeric values using exception handling.

- **Part B – Medium Level:**

• To create a Java program simulating an ATM system with PIN verification and balance withdrawal.
• To implement nested try-catch blocks and custom exceptions for invalid PINs or insufficient balance.

- **Part C – Hard Level:**

• To create a Java program for a university enrollment system with courses and prerequisites.
• To implement user-defined exceptions such as CourseFullException and PrerequisiteNotMetException to enforce business rules.

2. Objective:

- ✓ To understand the concept of exception handling in Java.
- ✓ To implement try-catch blocks to handle runtime errors and invalid inputs.
- ✓ To create and use custom exception classes for specific application scenarios.
- ✓ To ensure program robustness and proper flow even when errors occur.

3. JAVA script and output:

EASY-LEVEL PROBLEM

```
package exp.pkg3;


import java.util.Scanner;


public class Exp3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            String input = sc.nextLine();
            double num = Double.parseDouble(input);
            if (num < 0) {
                throw new IllegalArgumentException("Cannot calculate the square root of a negative number");
            }
            double result = Math.sqrt(num);
            System.out.println("Square root of " + num + " is: " + result);
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid input! Please enter a numeric value.");
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
```

```
    }
  }
}
```

OUTPUT:

```
run:
Enter a number:  64
Square root of 64.0 is: 8.0
BUILD SUCCESSFUL (total time: 3 seconds)
```

Figure 1: Easy Level

MEDIUM LEVEL PROBLEM:

```
package exp.pkg3;

import java.util.Scanner;

public class Exp3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        final int correctPIN = 1234;
        double balance = 3000.0;
        try {
            System.out.print("Enter PIN: ");
            int pin = sc.nextInt();
```

```
            if (pin != correctPIN) {
                throw new SecurityException("Invalid PIN entered!");
            }
            System.out.print("Enter withdrawal amount: ");
            double withdraw = sc.nextDouble();
            try {
                if (withdraw > balance) {
                    throw new Exception("Insufficient balance.");
                }
                balance -= withdraw;
                System.out.println("Withdrawal successful! Amount withdrawn: " + withdraw);
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        } catch (SecurityException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("Current Balance: " + balance);
        }}
}
```

OUTPUT:

```
Enter PIN: 1234
Enter withdrawal amount: 20000
Error: Insufficient balance.
Current Balance: 3000.0
```

Figure 2: Medium Level

HARD LEVEL PROBLEM

```
package exp.pkg3;

import java.util.*;

class Course {
    String courseName;
    int capacity;
    List<String> enrolledStudents = new ArrayList<>();
    String prerequisite;

    public Course(String courseName, int capacity, String prerequisite) {
        this.courseName = courseName;
        this.capacity = capacity;
        this.prerequisite = prerequisite;
    }

    public void enroll(String studentName, boolean prerequisiteCompleted) throws Exception {
        if (enrolledStudents.size() >= capacity) {
            throw new Exception("Course is full: " + courseName);
        }
        if (prerequisite != null && !prerequisiteCompleted) {
            throw new Exception("Complete " + prerequisite + " before enrolling in " +
courseName);
        }
```

```
        enrolledStudents.add(studentName);
        System.out.println("Success: " + studentName + " enrolled in " + courseName);
    }
}

public class Exp3{
    public static void main(String[] args) {
        Course advancedJava = new Course("Advanced Java", 2, "Core Java");
        try {
            advancedJava.enroll("Alice", false);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        try {
            advancedJava.enroll("Bob", true);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        try {
            advancedJava.enroll("Charlie", true);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
        try {
            advancedJava.enroll("David", true);
        } catch (Exception e) {
```

```
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
run:
Error: Complete Core Java before enrolling in Advanced Java
Success: Bob enrolled in Advanced Java
Success: Charlie enrolled in Advanced Java
Error: Course is full: Advanced Java
BUILD SUCCESSFUL (total time: 0 seconds)
```