

Experiment 3

Student Name: Deepanshu

UID: 22BCS15133

Branch: BE-CSE

Section/Group:DL-902/B

Semester: Sixth

Date of Performance: 05/02/2025

Subject Name: Project Based Learning in Java with Lab

Subject Code: 22CSH-359

Problem - 1

- 1. Aim:** Write a Java program to calculate the square root of a number entered by the user. Use try-catch to handle invalid inputs (e.g., negative numbers or non-numeric values).

Input Example:

Enter a number: -16

Output Example:

Error: Cannot calculate the square root of a negative number

2. Objective:

- Accept user input and parse it as a number.
- Check for negative values and throw an exception if found.
- Calculate the square root using `Math.sqrt()` for valid inputs.
- Handle exceptions for negative numbers and invalid inputs.
- Close the scanner to ensure proper resource management.

3. Code:

```
// import java.util.Scanner;
```

```
public class SquareRootCalculator {
```

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

```
        Scanner scanner = new Scanner(System.in);
```

```
System.out.print("Enter a number: ");

try {

    double number = scanner.nextDouble();

    if (number < 0) {

        throw new IllegalArgumentException("Error: Cannot calculate the square root of a negative
number.");

    }

    double result = Math.sqrt(number);

    System.out.println("Square root: " + result);

} catch (IllegalArgumentException e) {

    System.out.println(e.getMessage());

} catch (Exception e) {

    System.out.println("Error: Invalid input. Please enter a numeric value.");

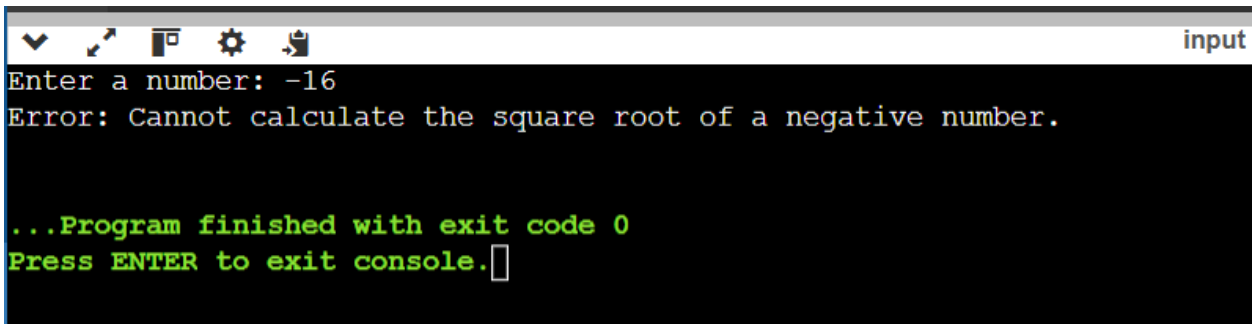
} finally {

    scanner.close();

}

}
```

4. Output



```
input
Enter a number: -16
Error: Cannot calculate the square root of a negative number.

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Learning Outcomes:

- **Exception Handling** – Understand how to use try-catch to handle errors gracefully.
- **Input Validation** – Learn to check user input for invalid values like negative numbers or non-numeric inputs.
- **Mathematical Computation** – Use Math.sqrt() to compute the square root of a valid number.
- **User Interaction** – Gain experience in taking user input using Scanner and providing meaningful error messages.
- **Resource Management** – Learn to use the finally block to close resources like Scanner properly.

Problem - 2

1. Aim: Write a Java program to simulate an ATM withdrawal system. The program should:

Ask the user to enter their PIN.

Allow withdrawal if the PIN is correct and the balance is sufficient.

Throw exceptions for invalid PIN or insufficient balance.

Ensure the system always shows the remaining balance, even if an exception occurs.

Input Example:

Enter PIN: 1234

Withdraw Amount: 5000

Output Example:

Error: Insufficient balance. Current Balance: 3000

2. Objective:

- Authenticate the user by verifying the entered PIN.
- Allow withdrawal if the PIN is correct and the balance is sufficient.
- Throw exceptions for invalid PIN or insufficient funds.
- Handle invalid input (non-numeric values) gracefully.
- Always display the remaining balance, even if an exception occurs

3. Code:

```
import java.util.Scanner;

class InvalidPinException extends Exception {

    public InvalidPinException(String message) {

        super(message);

    }

}

class InsufficientBalanceException extends Exception {

    public InsufficientBalanceException(String message) {

        super(message);

    }

}

public class ATM {

    private static final int CORRECT_PIN = 1234;

    private static double balance = 3000;

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try {

            System.out.print("Enter PIN: ");

            int enteredPin = scanner.nextInt();

            if (enteredPin != CORRECT_PIN) {

                throw new InvalidPinException("Error: Invalid PIN.");

            }

            System.out.print("Withdraw Amount: ");
```

```
double amount = scanner.nextDouble();

if (amount > balance) {

    throw new InsufficientBalanceException("Error: Insufficient balance.");

}

balance -= amount;

System.out.println("Withdrawal successful! Remaining Balance: " + balance);

} catch (InvalidPinException | InsufficientBalanceException e) {

    System.out.println(e.getMessage() + " Current Balance: " + balance);

} catch (Exception e) {

    System.out.println("Error: Invalid input. Please enter a valid number.");

} finally {

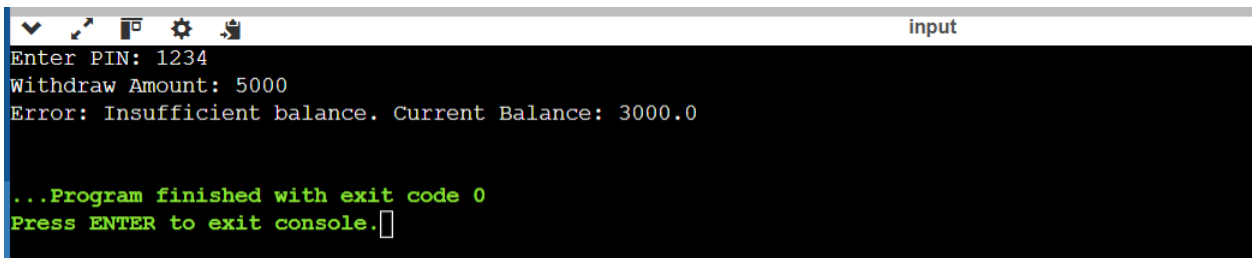
    scanner.close();

}

}

}
```

4. Output:



```
input
Enter PIN: 1234
Withdraw Amount: 5000
Error: Insufficient balance. Current Balance: 3000.0

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Learning Outcomes:

- **Custom Exceptions** – Learn to define and use custom exceptions (InvalidPinException and InsufficientBalanceException).
- **Exception Handling** – Use try-catch-finally to manage errors and ensure the program runs smoothly.
- **User Input Handling** – Understand how to take user input and validate it.
- **Basic Banking Operations** – Implement balance checking and updating logic.
- **Resource Management** – Learn to properly close Scanner in the finally block.

Problem – 3

1. Aim:

Create a Java program for a university enrollment system with exception handling. The program should:

Allow students to enroll in courses.

Throw a CourseFullException if the maximum enrollment limit is reached.

Throw a PrerequisiteNotMetException if the student hasn't completed prerequisite courses.

Input Example:

Enroll in Course: Advanced Java

Prerequisite: Core Java

Status: Prerequisite not completed

Output Example:

Error: PrerequisiteNotMetException - Complete Core Java before enrolling in Advanced Java.

2. Objective:

- **Allow students to enroll** in university courses.
- **Throw a CourseFullException** if the course has reached its maximum enrollment limit.
- **Throw a PrerequisiteNotMetException** if the student has not completed the required prerequisite course.

- **Validate student eligibility** before enrollment and provide meaningful error messages.
- **Demonstrate exception handling** to ensure smooth program execution.

3. Code:

```
import java.util.*;

class CourseFullException extends Exception {

    public CourseFullException(String message) {

        super(message);

    }

}

class PrerequisiteNotMetException extends Exception {

    public PrerequisiteNotMetException(String message) {

        super(message);

    }

}

class Course {

    private String name;

    private String prerequisite;

    private int maxCapacity;

    private List<String> enrolledStudents;

    public Course(String name, String prerequisite, int maxCapacity) {

        this.name = name;

        this.prerequisite = prerequisite;

        this.maxCapacity = maxCapacity;

        this.enrolledStudents = new ArrayList<>();
```

```
}

public String getName() {

    return name;

}

public String getPrerequisite() {

    return prerequisite;

}

public boolean isFull() {

    return enrolledStudents.size() >= maxCapacity;

}

public void enrollStudent(String student) throws CourseFullException {

    if (isFull()) {

        throw new CourseFullException("Course " + name + " is full.");

    }

    enrolledStudents.add(student);

}

}

class Student {

    private String name;

    private Set<String> completedCourses;

    public Student(String name) {

        this.name = name;

        this.completedCourses = new HashSet<>();

    }

}
```



```
public String getName() {

    return name;

}

public void completeCourse(String course) {

    completedCourses.add(course);

}

public boolean hasCompletedCourse(String course) {

    return completedCourses.contains(course);

}

public void enroll(Course course) throws CourseFullException, PrerequisiteNotMetException {

    if (course.getPrerequisite() != null && !course.getPrerequisite().isEmpty() &&
!hasCompletedCourse(course.getPrerequisite())) {

        throw new PrerequisiteNotMetException("Complete " + course.getPrerequisite() + " before
enrolling in " + course.getName() + ".");

    }

    course.enrollStudent(this.name);

    System.out.println("Successfully enrolled in " + course.getName());

}

}

public class UniversityEnrollmentSystem {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        Course coreJava = new Course("Core Java", "", 2);

        Course advancedJava = new Course("Advanced Java", "Core Java", 2);
```

```
Student student = new Student("Alice");

System.out.println("Enroll in Course: Advanced Java");

System.out.println("Prerequisite: " + advancedJava.getPrerequisite());

System.out.println("Status: Prerequisite not completed");

try {

    student.enroll(advancedJava);

} catch (CourseFullException | PrerequisiteNotMetException e) {

    System.out.println("Error: " + e.getClass().getSimpleName() + " - " + e.getMessage());

}

student.completeCourse("Core Java");

System.out.println("\nAttempting to enroll again after completing prerequisite...");

try {

    student.enroll(advancedJava);

} catch (CourseFullException | PrerequisiteNotMetException e) {

    System.out.println("Error: " + e.getClass().getSimpleName() + " - " + e.getMessage());

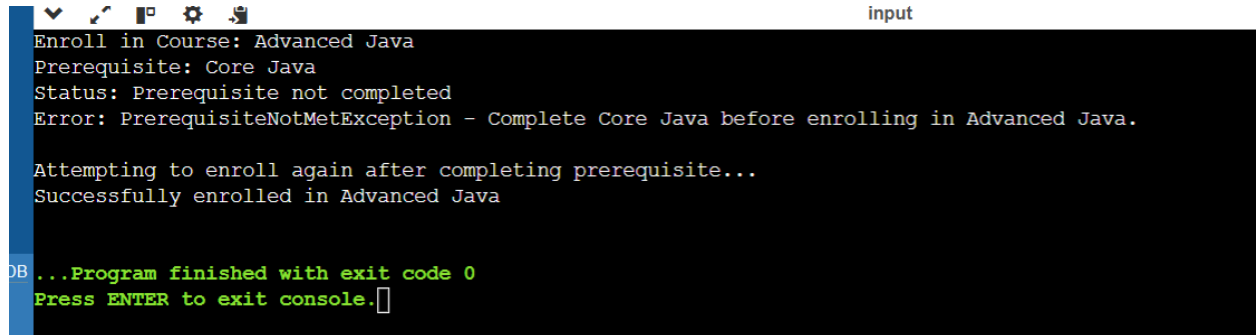
}

scanner.close();

}

}
```

4. Output:



```
Enroll in Course: Advanced Java
Prerequisite: Core Java
Status: Prerequisite not completed
Error: PrerequisiteNotMetException - Complete Core Java before enrolling in Advanced Java.

Attempting to enroll again after completing prerequisite...
Successfully enrolled in Advanced Java

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Learning Outcome:

- **Custom Exception Handling** – Define and use CourseFullException and PrerequisiteNotMetException.
- **Object-Oriented Programming (OOP)** – Implement classes (Course, Student) with encapsulation and methods.
- **Exception Handling (try-catch)** – Catch and handle specific exceptions for controlled program flow.
- **Data Structures Usage** – Utilize List for enrolled students and Set for tracking completed courses.
- **User Interaction & Validation** – Check prerequisites and enrollment limits before allowing registration.