Experiment-3

Student Name: Gagnesh Kakkar
UID: 23BCS11196
Branch: B.E-C.S.E
Section/Group: 23KRG-2B
Semester: 5th
Date of Performance: 18/08/2025
Subject Name: PBLJ
Subject Code: 23CSH-304

Easy Level

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).

2. **Objective:** To understand how to handle invalid input using try-catch blocks in Java.

3. **Input/Apparatus Used:** Java exception classes, try-catch block, Scanner class for input.

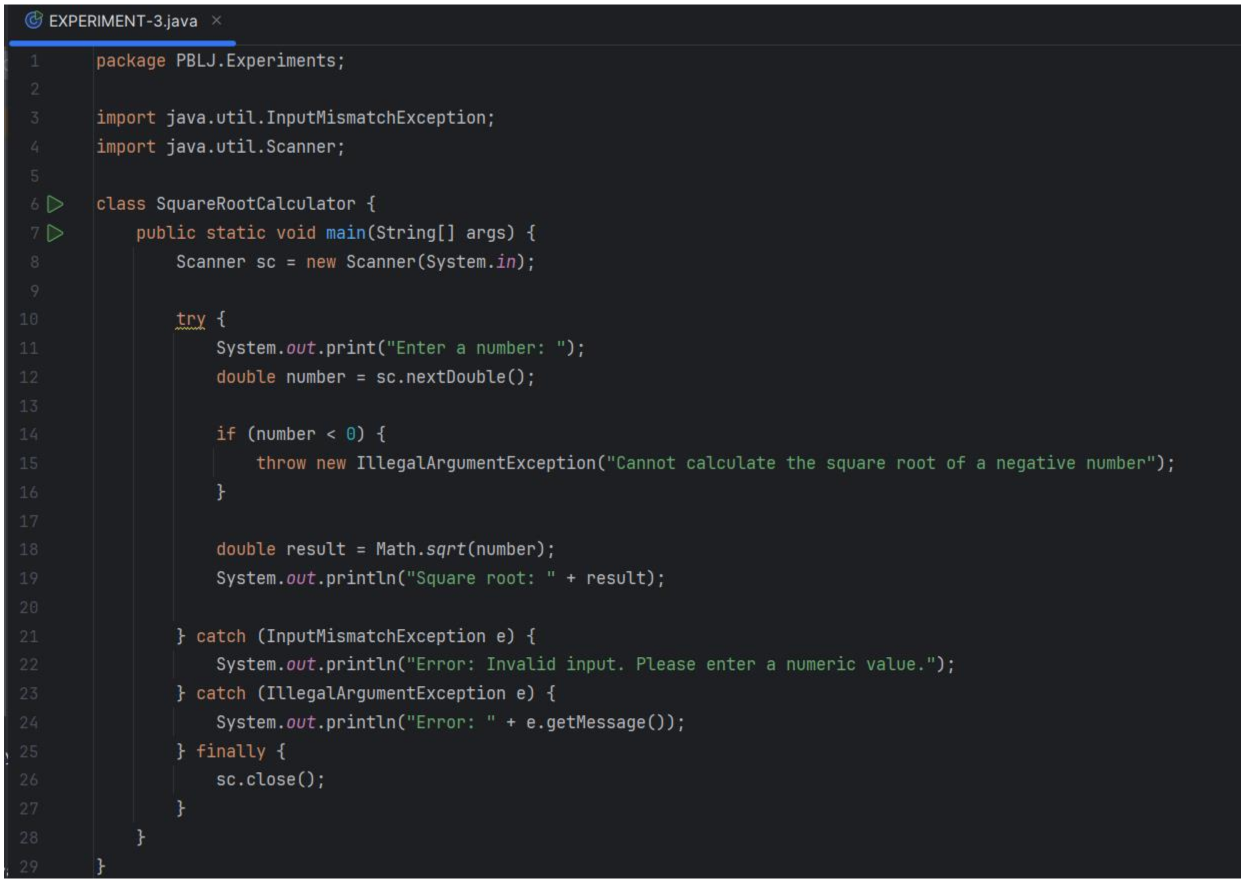
4. **Procedure:**
   1. Prompt the user to input a number.
   2. Convert input to a number type using Scanner.
   3. Use a try-catch block to handle NumberFormatException and check for negative values.
   4. If the number is negative, manually throw an exception.
   5. If the number is valid, calculate and print the square root.

5.

Sample Input:
Enter a number: -16

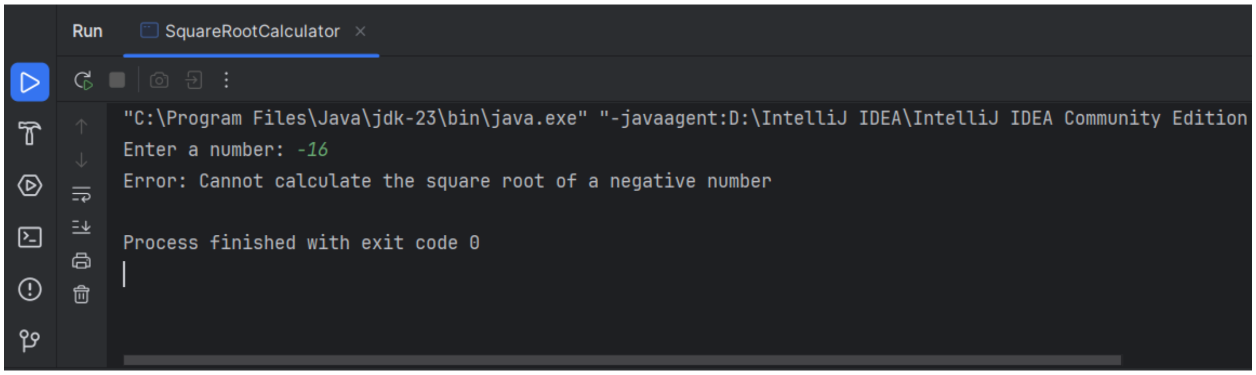
Sample Output:
Error: Cannot calculate the square root of a negative number

6. **Code:**


EXPERIMENT-3.java

```java
package PBLJ.Experiments;

import java.util.InputMismatchException;
import java.util.Scanner;

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

        try {
            System.out.print("Enter a number: ");
            double number = sc.nextDouble();

            if (number < 0) {
                throw new IllegalArgumentException("Cannot calculate the square root of a negative number");
            }

            double result = Math.sqrt(number);
            System.out.println("Square root: " + result);

        } catch (InputMismatchException e) {
            System.out.println("Error: Invalid input. Please enter a numeric value.");
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            sc.close();
        }
    }
}
```

7. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA\IntelliJ IDEA Community Edition
Enter a number: -16
Error: Cannot calculate the square root of a negative number

Process finished with exit code 0
```

Medium Level

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.

2. **Objective:** Implement nested try-catch blocks and create meaningful exception messages.

3. **Input/Apparatus Used:** Java Scanner, custom logic for PIN and balance verification, exception classes.

4. **Procedure:**
   1. Prompt the user to enter their ATM PIN.
   2. Check if the PIN is correct.
   3. If valid, prompt for withdrawal amount.
   4. Check whether the withdrawal amount is less than or equal to the balance.
   5. If not, throw a custom InsufficientBalanceException.
   6. Use finally to print the current balance irrespective of the exception.

5. **Sample Input:**

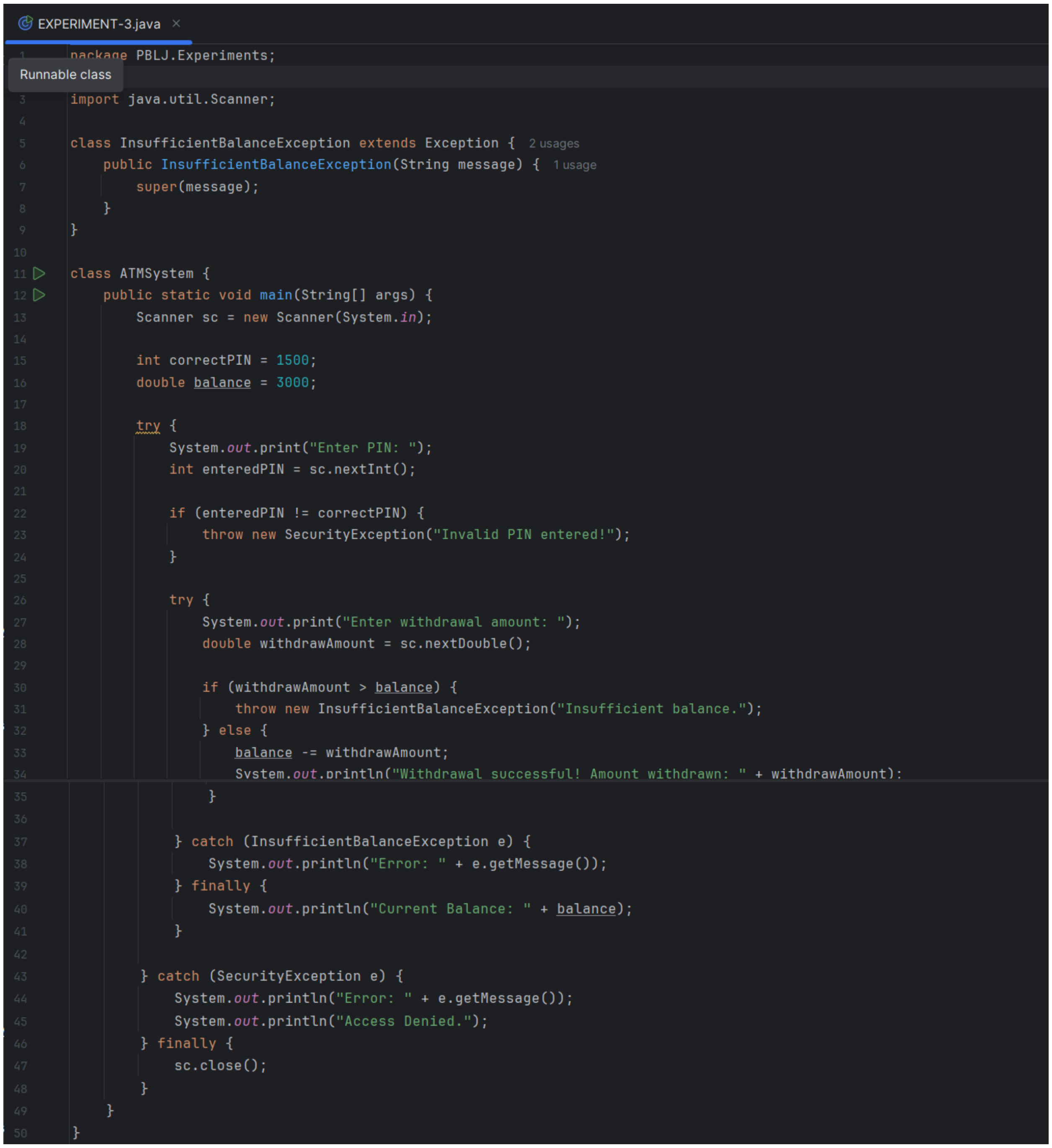
   Enter PIN: 1234

   Withdraw Amount: 5000

   Sample Output :

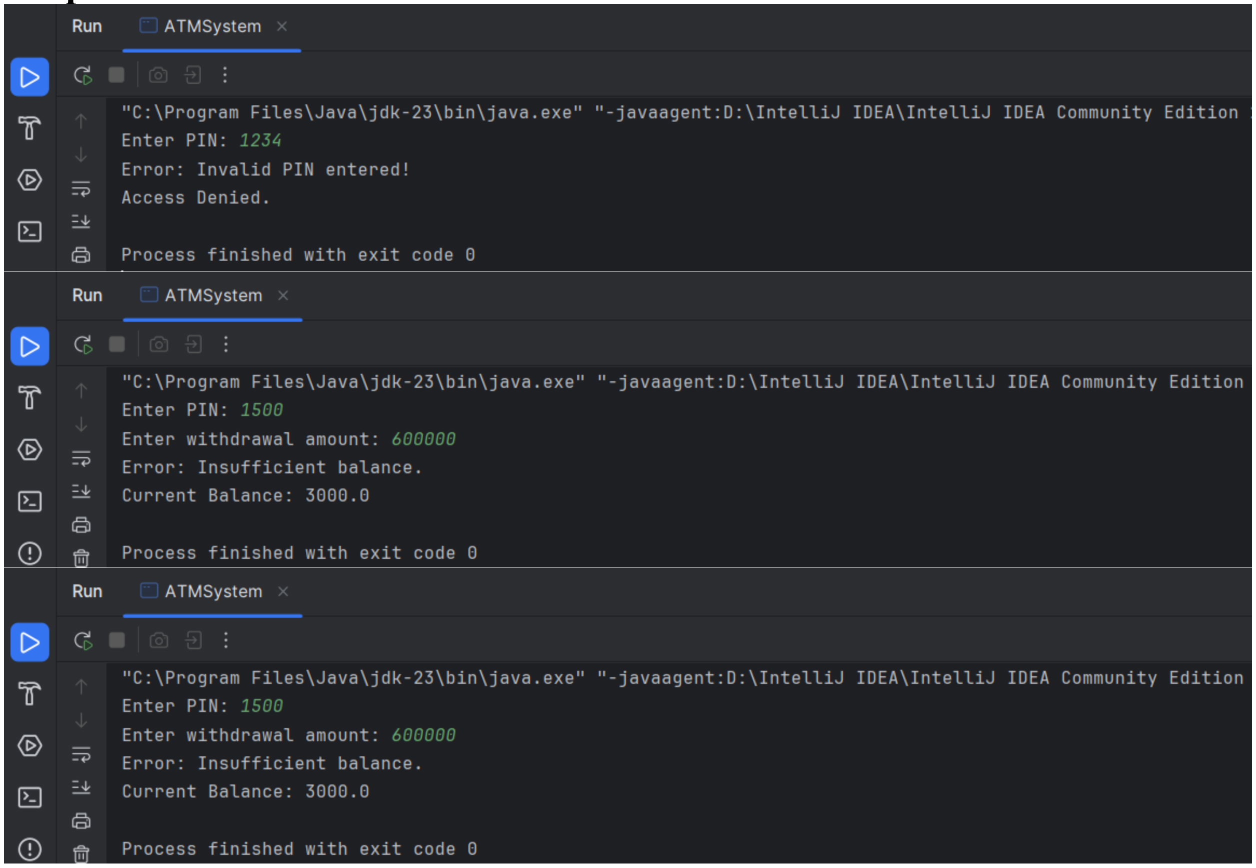
   Error: Insufficient balance. Current Balance: 3000

6. **Code:**



```java
package PBLJ.Experiments;

import java.util.Scanner;

class InsufficientBalanceException extends Exception {  2 usages
    public InsufficientBalanceException(String message) {  1 usage
        super(message);
    }
}

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

        int correctPIN = 1500;
        double balance = 3000;

        try {
            System.out.print("Enter PIN: ");
            int enteredPIN = sc.nextInt();

            if (enteredPIN != correctPIN) {
                throw new SecurityException("Invalid PIN entered!");
            }

            try {
                System.out.print("Enter withdrawal amount: ");
                double withdrawAmount = sc.nextDouble();

                if (withdrawAmount > balance) {
                    throw new InsufficientBalanceException("Insufficient balance.");
                } else {
                    balance -= withdrawAmount;
                    System.out.println("Withdrawal successful! Amount withdrawn: " + withdrawAmount);
                }

            } catch (InsufficientBalanceException e) {
                System.out.println("Error: " + e.getMessage());
            } finally {
                System.out.println("Current Balance: " + balance);
            }

        } catch (SecurityException e) {
            System.out.println("Error: " + e.getMessage());
            System.out.println("Access Denied.");
        } finally {
            sc.close();
        }
    }
}
```

7. Output:



```
Run   ATMSystem ×
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA\IntelliJ IDEA Community Edition
Enter PIN: 1234
Error: Invalid PIN entered!
Access Denied.

Process finished with exit code 0
```

```
Run   ATMSystem ×
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA\IntelliJ IDEA Community Edition
Enter PIN: 1500
Enter withdrawal amount: 600000
Error: Insufficient balance.
Current Balance: 3000.0

Process finished with exit code 0
```

```
Run   ATMSystem ×
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA\IntelliJ IDEA Community Edition
Enter PIN: 1500
Enter withdrawal amount: 600000
Error: Insufficient balance.
Current Balance: 3000.0

Process finished with exit code 0
```

<u>Hard Level</u>

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.

2. **Objective:** Demonstrate advanced exception handling using user-defined exception classes and business rule enforcement.

3. **Input/Apparatus Used:** Java class hierarchy, custom exception classes, and control structures.

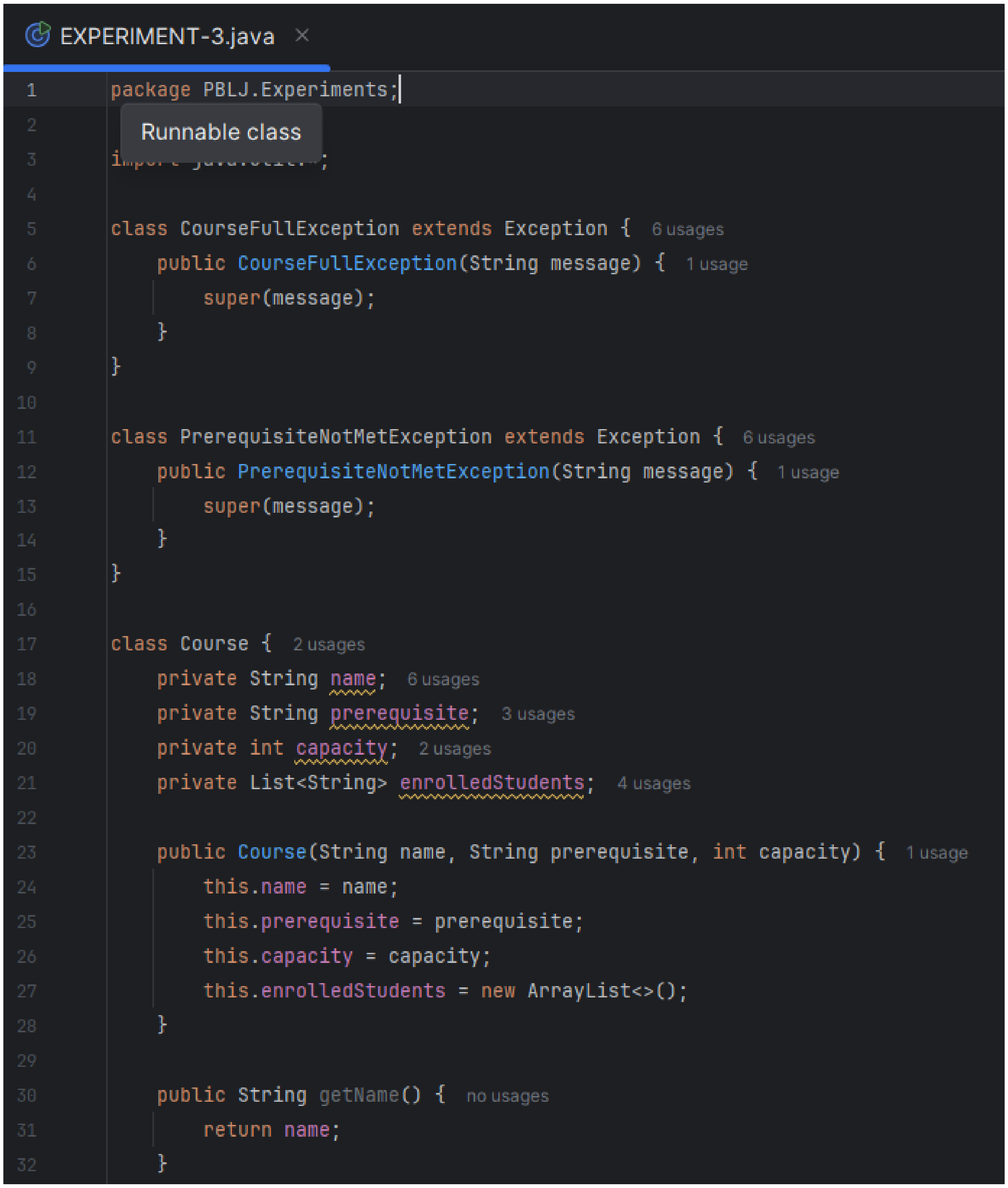
4. Procedure:

1. Define custom exceptions CourseFullException and PrerequisiteNotMetException.
2. Create a class representing course enrollment logic.
3. Validate course capacity; if full, throw CourseFullException.
4. Validate prerequisite completion; if not satisfied, throw PrerequisiteNotMetException.
5. Handle each exception with appropriate error messages.
6. Display successful enrollments and errors separately.

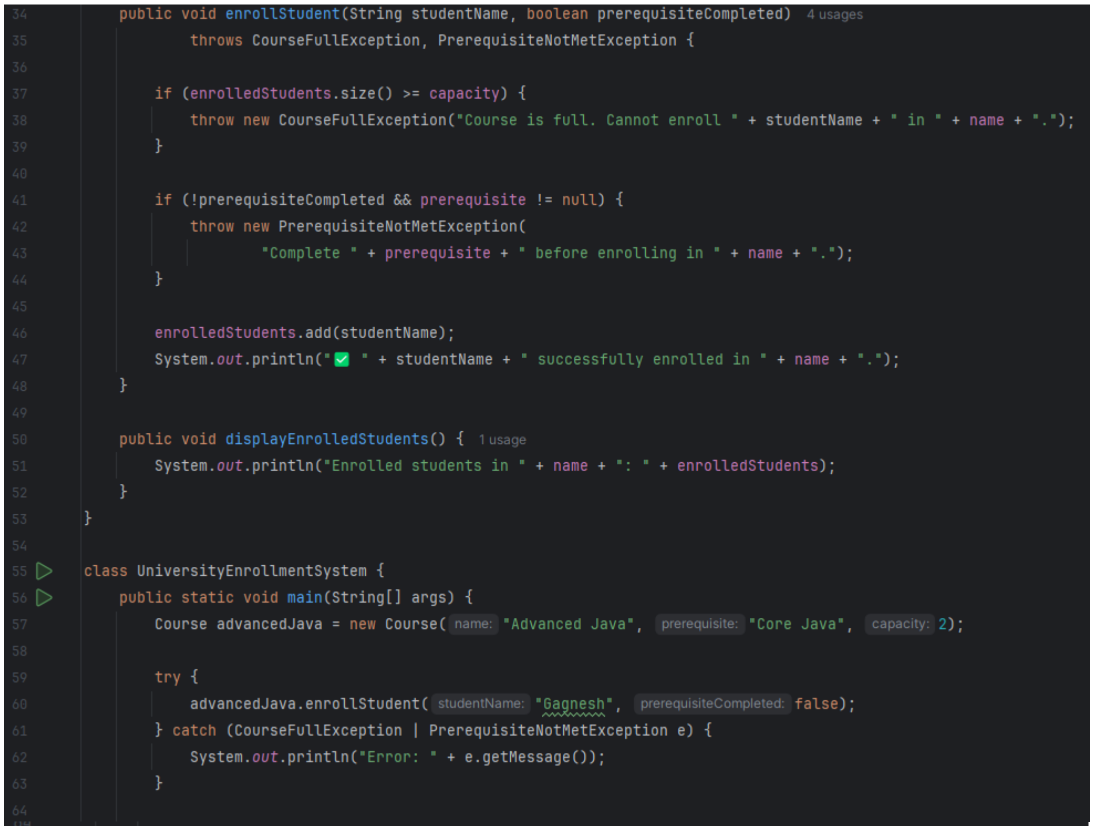
Sample Input:
Enroll in Course: Advanced Java
Prerequisite: Core Java
Status: Prerequisite not completed

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

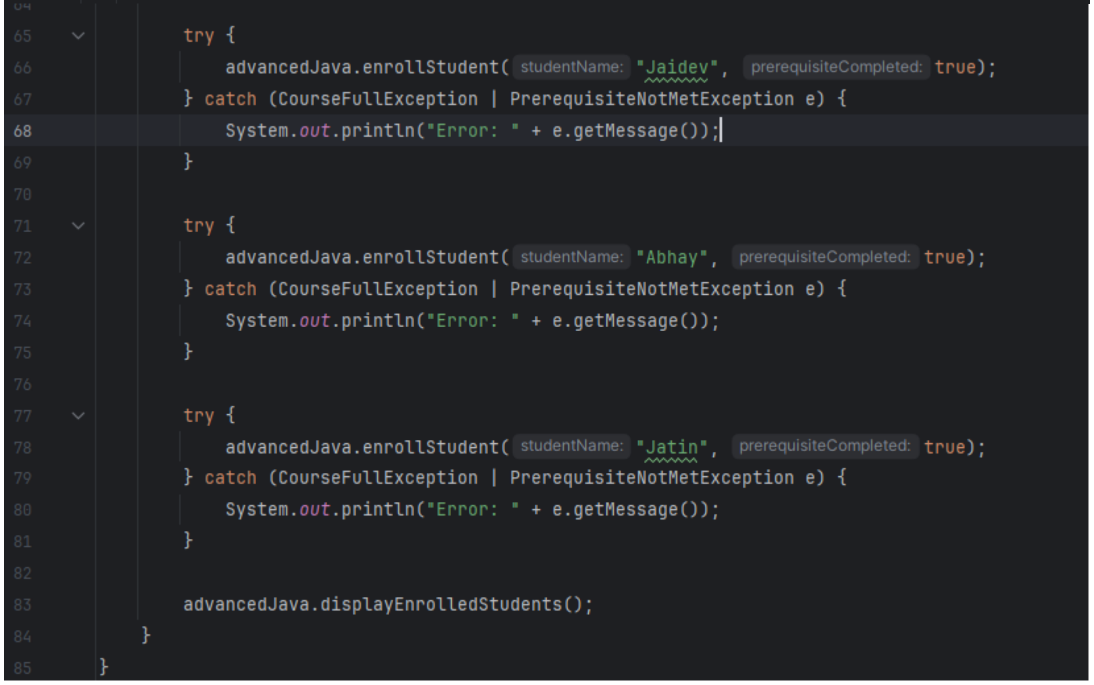
5. Code:



```java
package PBLJ.Experiments;

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 capacity;
    private List<String> enrolledStudents;

    public Course(String name, String prerequisite, int capacity) {
        this.name = name;
        this.prerequisite = prerequisite;
        this.capacity = capacity;
        this.enrolledStudents = new ArrayList<>();
    }

    public String getName() {
        return name;
    }
```

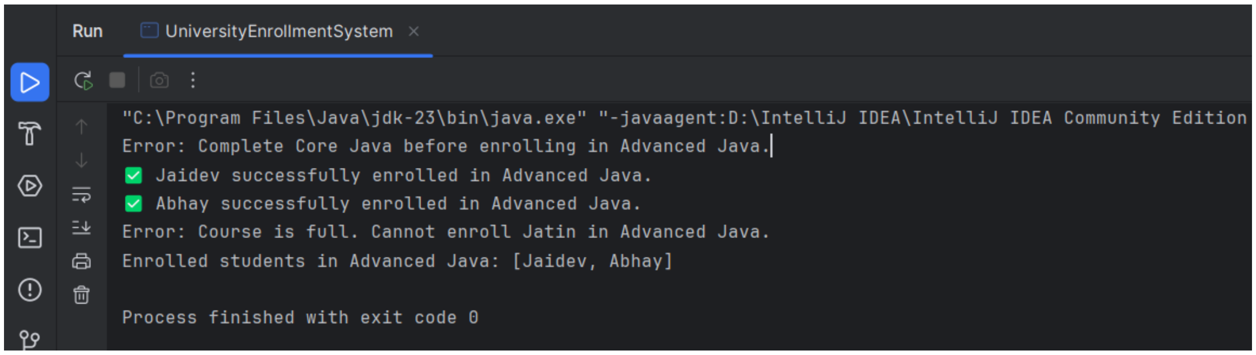


```java
    public void enrollStudent(String studentName, boolean prerequisiteCompleted)  4 usages
            throws CourseFullException, PrerequisiteNotMetException {

        if (enrolledStudents.size() >= capacity) {
            throw new CourseFullException("Course is full. Cannot enroll " + studentName + " in " + name + ".");
        }

        if (!prerequisiteCompleted && prerequisite != null) {
            throw new PrerequisiteNotMetException(
                    "Complete " + prerequisite + " before enrolling in " + name + ".");
        }

        enrolledStudents.add(studentName);
        System.out.println("✅ " + studentName + " successfully enrolled in " + name + ".");
    }

    public void displayEnrolledStudents() {  1 usage
        System.out.println("Enrolled students in " + name + ": " + enrolledStudents);
    }
}

class UniversityEnrollmentSystem {
    public static void main(String[] args) {
        Course advancedJava = new Course( name: "Advanced Java",  prerequisite: "Core Java",  capacity: 2);

        try {
            advancedJava.enrollStudent( studentName: "Gagnesh",  prerequisiteCompleted: false);
        } catch (CourseFullException | PrerequisiteNotMetException e) {
            System.out.println("Error: " + e.getMessage());
        }

```



```java
        try {
            advancedJava.enrollStudent( studentName: "Jaidev",  prerequisiteCompleted: true);
        } catch (CourseFullException | PrerequisiteNotMetException e) {
            System.out.println("Error: " + e.getMessage());
        }

        try {
            advancedJava.enrollStudent( studentName: "Abhay",  prerequisiteCompleted: true);
        } catch (CourseFullException | PrerequisiteNotMetException e) {
            System.out.println("Error: " + e.getMessage());
        }

        try {
            advancedJava.enrollStudent( studentName: "Jatin",  prerequisiteCompleted: true);
        } catch (CourseFullException | PrerequisiteNotMetException e) {
            System.out.println("Error: " + e.getMessage());
        }

        advancedJava.displayEnrolledStudents();
    }
}
```

6. Output:



```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA\IntelliJ IDEA Community Edition
Error: Complete Core Java before enrolling in Advanced Java.
✅ Jaidev successfully enrolled in Advanced Java.
✅ Abhay successfully enrolled in Advanced Java.
Error: Course is full. Cannot enroll Jatin in Advanced Java.
Enrolled students in Advanced Java: [Jaidev, Abhay]

Process finished with exit code 0
```