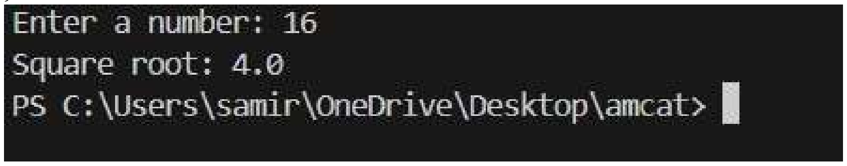
Experiment 3

Student Name: Mohammad Farhan Alam UID: 22BCS13460
Branch: CSE Section/Group:DL_902/A
Semester: 6th DOP: 23/1/2025
Subject: Java Lab Subject Code: 22CSH-359

Problem Statement: 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).

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();
        }
    }
}
```



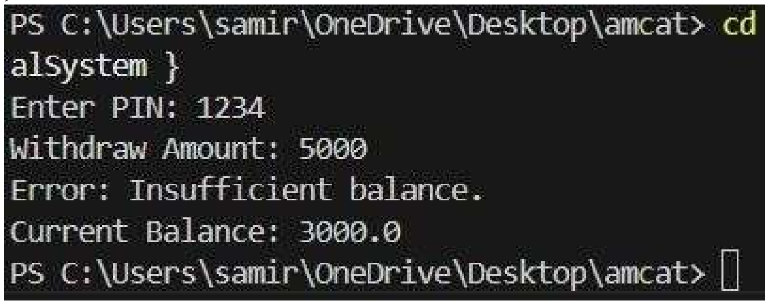
```
Enter a number: 16
Square root: 4.0
PS C:\Users\samir\OneDrive\Desktop\amcat>
```

Problem Statement: 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.

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  ATMWithdrawalSystem  {  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 pin = scanner.nextInt();

         if (pin != 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());
        } catch (Exception e) {
            System.out.println("Error: Invalid input. Please enter numeric values.");
        } finally {
            System.out.println("Current    Balance:    "    +    balance);
            scanner.close();
        }
    }
}
```



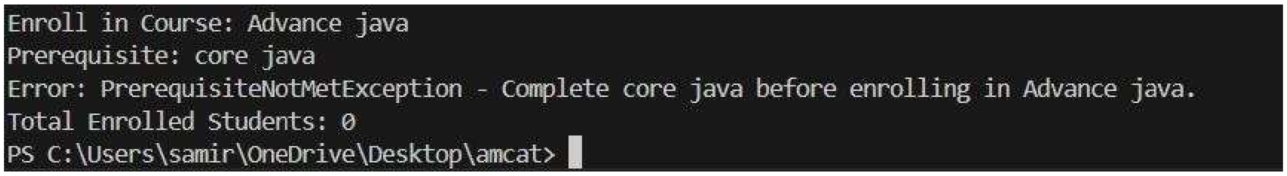
```
PS C:\Users\samir\OneDrive\Desktop\amcat> cd
alSystem }
Enter PIN: 1234
Withdraw Amount: 5000
Error: Insufficient balance.
Current Balance: 3000.0
PS C:\Users\samir\OneDrive\Desktop\amcat>
```

Problem Statement: 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.

CODE :

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.Set;

class CourseFullException extends Exception
    {    public    CourseFullException(String    message)    {
        super(message);
    }
}
```

```
class PrerequisiteNotMetException extends Exception
   {     public     PrerequisiteNotMetException(String     message)     {
      super(message);
   }
}
class UniversityEnrollmentSystem {
   private static final int MAX_ENROLLMENT = 30; private static int
   enrolledStudents    =    0;    private    static    final    Set<String>
   completedCourses = new HashSet<>();

   public static void enroll(String course, String prerequisite) throws CourseFullException,
PrerequisiteNotMetException { if (enrolledStudents >=
      MAX_ENROLLMENT) {
         throw new CourseFullException("Error: Course is full. Cannot enroll.");
      }

      if (!completedCourses.contains(prerequisite)) { throw new PrerequisiteNotMetException("Error:
         PrerequisiteNotMetException - Complete " +
prerequisite + " before enrolling in " + course + ".");
      }

      enrolledStudents++;
      System.out.println("Enrollment successful in " + course + ".");
   }

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

      System.out.print("Enroll in Course: ");
      String course = scanner.nextLine();

      System.out.print("Prerequisite:        ");
      String prerequisite = scanner.nextLine();

      try     {     enroll(course,
         prerequisite);
      } catch (CourseFullException | PrerequisiteNotMetException e)
         { System.out.println(e.getMessage());
      } finally {
         System.out.println("Total    Enrolled    Students:    "    +    enrolledStudents);
         scanner.close();
      }
   }
}
```



```
Enroll in Course: Advance java
Prerequisite: core java
Error: PrerequisiteNotMetException - Complete core java before enrolling in Advance java.
Total Enrolled Students: 0
PS C:\Users\samir\OneDrive\Desktop\amcat>
```

Learning Outcomes:

- Inheritance: Use of base and derived classes for shared attributes and methods.
- Method Overriding: Custom implementation of methods in subclasses.
- Constructor: Initializing object attributes using constructors.
- Encapsulation: Storing and manipulating data within objects.
- Polymorphism: Different behavior of calculateInterest() based on object type.

- Interest Calculation: Implementing FD and RD interest formulas.
- Class Interaction: Creating objects and calling methods to display details.

CHANDIGARH UNIVERSITY Discover. Learn. Empower.

DEPARTMENTOF

COMPUTERSCIENCE&ENGINEERING