Experiment - 3

Student Name : Aman **UID: 22BCS15115**

Branch: BE-CSE **Section/Group: 635-B**

Semester: 6th **Date of Performance: 10/03/25**

Subject Name: Java **Subject Code: 22CSH-352**

1. **Aim: Develop a program for**

   a) Easy Level: Square Root Calculation

   b) Medium Level: ATM Withdrawal System

   c) Hard Level: University Enrollment System

2. **Implementation/Code:**

a)

```
 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 num = scanner.nextDouble();
      if (num < 0) {
         throw new IllegalArgumentException("Error: Cannot calculate the square
        root of a negative number.");
            }
            System.out.println("Square Root: " + Math.sqrt(num));
        } 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();
        }
    }
```

b)

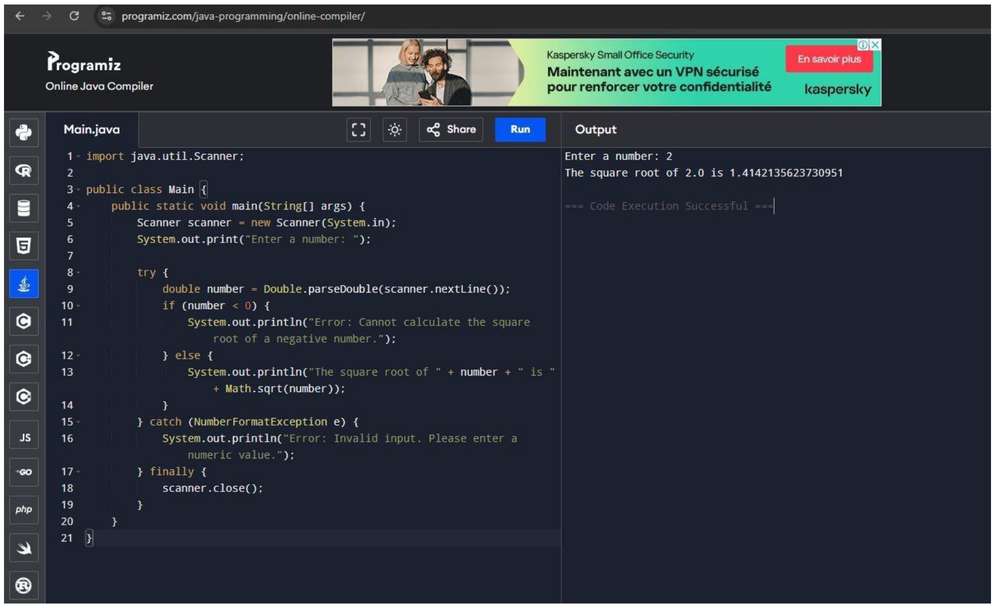
```
import java.util.Scanner;

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 PIN = 1234;
    private static double balance = 3000.0;

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

        try {
            System.out.print("Enter PIN: ");
            int enteredPin = scanner.nextInt();
            if (enteredPin != PIN) {
                throw new InvalidPinException("Error: Invalid PIN.");
            }

            System.out.print("Withdraw Amount: ");
            double withdrawAmount = scanner.nextDouble();

            if (withdrawAmount > balance) {
                throw new InsufficientBalanceException("Error: Insufficient balance. Current
Balance: " + balance);
            }

            balance -= withdrawAmount;
            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.");
        } finally {
            System.out.println("Final Balance: " + balance);
            scanner.close();
        }
    }
```

c)

```
import java.util.HashMap;
import java.util.Scanner;

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

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

public class UniversityEnrollment {
    private static final int MAX_ENROLLMENT = 2;
    private static HashMap<String, Integer> courseEnrollments = new HashMap<>();
    private static HashMap<String, String> prerequisites = new HashMap<>();

    public static void main(String[] args) {
        // Defining course prerequisites
        prerequisites.put("Advanced Java", "Core Java");
        prerequisites.put("Machine Learning", "Mathematics");

        Scanner scanner = new Scanner(System.in);

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

            if (prerequisites.containsKey(course) &&
!prerequisites.get(course).equals(prerequisite)) {
                throw new PrerequisiteNotMetException("Error: PrerequisiteNotMetException
- Complete " + prerequisites.get(course) + " before enrolling in " + course + ".");
            }

            int enrolledCount = courseEnrollments.getOrDefault(course, 0);
            if (enrolledCount >= MAX_ENROLLMENT) {
                throw new CourseFullException("Error: CourseFullException - The course is
full.");
            }

            courseEnrollments.put(course, enrolledCount + 1);
            System.out.println("Enrollment successful for " + course + ".");

        } catch (PrerequisiteNotMetException | CourseFullException e) {
            System.out.println(e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

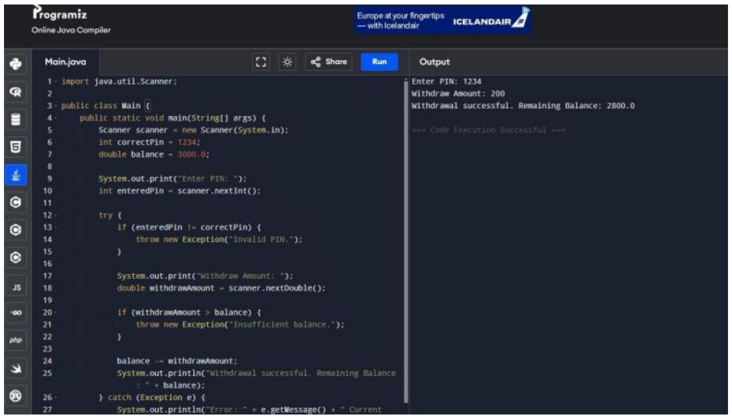
3. Output:

(a)



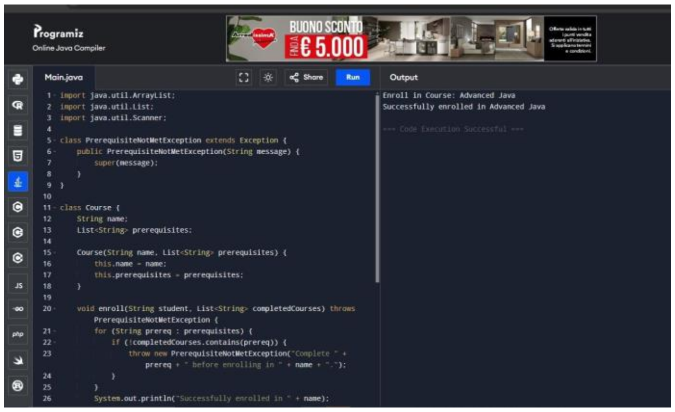
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");

        try {
            double number = Double.parseDouble(scanner.nextLine());
            if (number < 0) {
                System.out.println("Error: Cannot calculate the square root of a negative number.");
            } else {
                System.out.println("The square root of " + number + " is " + Math.sqrt(number));
            }
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid input. Please enter a numeric value.");
        } finally {
            scanner.close();
        }
    }
}
```

```
Enter a number: 2
The square root of 2.0 is 1.4142135623730951

=== Code Execution Successful ===
```



```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int correctPin = 1234;
        double balance = 3000.0;

        System.out.print("Enter PIN: ");
        int enteredPin = scanner.nextInt();

        try {
            if (enteredPin != correctPin) {
                throw new Exception("Invalid PIN.");
            }

            System.out.print("Withdraw Amount: ");
            double withdrawAmount = scanner.nextDouble();

            if (withdrawAmount > balance) {
                throw new Exception("Insufficient balance.");
            }

            balance -= withdrawAmount;
            System.out.println("Withdrawal successful. Remaining Balance: " + balance);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage() + " Current
```

```
Enter PIN: 1234
Withdraw Amount: 200
Withdrawal successful. Remaining Balance: 2800.0

=== Code Execution Successful ===
```



```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

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

class Course {
    String name;
    List<String> prerequisites;

    Course(String name, List<String> prerequisites) {
        this.name = name;
        this.prerequisites = prerequisites;
    }

    void enroll(String student, List<String> completedCourses) throws PrerequisiteNotMetException {
        for (String prereq : prerequisites) {
            if (!completedCourses.contains(prereq)) {
                throw new PrerequisiteNotMetException("Complete " + prereq + " before enrolling in " + name + ".");
            }
        }
        System.out.println("Successfully enrolled in " + name);
```

```
Enroll in Course: Advanced Java
Successfully enrolled in Advanced Java

=== Code Execution Successful ===
```

(b) (c)

6. Learning Outcomes:

- ✓ Exception Handling & Robust Code – Learn to use try-catch, throw, and custom exceptions for handling errors like invalid input, insufficient balance, and unmet prerequisites.
- ✓ User Input & Decision Making – Gain experience in handling user inputs, validating conditions (PIN check, balance check, prerequisites), and controlling program flow.
- ✓ OOP & Data Management – Understand object-oriented principles like custom exception classes and use data structures (e.g., HashMap) for managing enrollments dynamically.