



Experiment 3

Student Name: Divyanshu Jaiswal

Branch: B.E. CSE

Semester: 6th

Subject Name: PBLJ LAB

UID: 22BCS12806

Section/Group: KRG - 2 B

Date of Performance: 25/01/25

Subject Code: 22CSH-359

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 {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
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 + ".");
            }

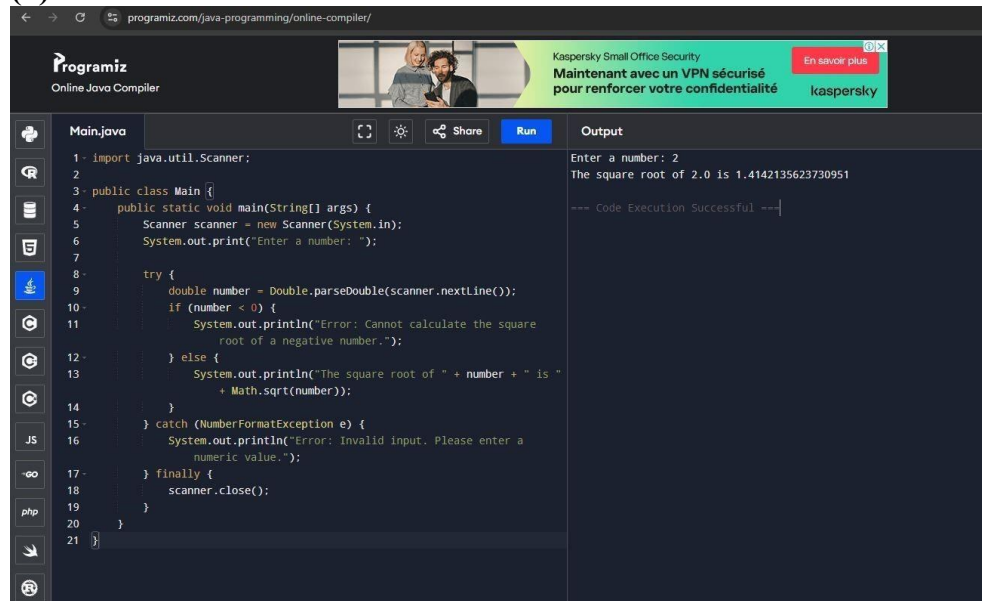
            int enrolledCount = courseEnrollments.getOrDefault(course, 0);
            if (enrolledCount >= MAX_ENROLLMENT) {
                throw new CourseFullException("Error: CourseFullException - The course is
full.");
            }

            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:

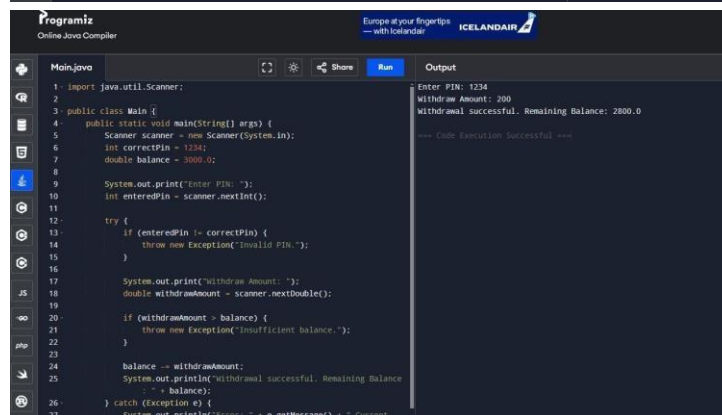
(a)



```

1- import java.util.Scanner;
2
3- public class Main {
4-     public static void main(String[] args) {
5-         Scanner scanner = new Scanner(System.in);
6-         System.out.print("Enter a number: ");
7
8-         try {
9-             double number = Double.parseDouble(scanner.nextLine());
10-            if (number < 0) {
11-                System.out.println("Error: Cannot calculate the square
12-                root of a negative number.");
13-            } else {
14-                System.out.println("The square root of " + number + " is "
15-                + Math.sqrt(number));
16-            }
17-        } catch (NumberFormatException e) {
18-            System.out.println("Error: Invalid input. Please enter a
19-            numeric value.");
20-        } finally {
21-            scanner.close();
22-        }
23-    }
24- }
  
```

Output: Enter a number: 2
The square root of 2.0 is 1.4142135623730951
--- Code Execution Successful ---

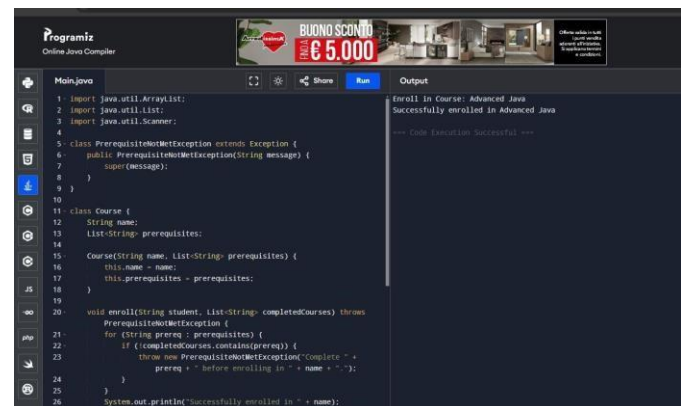


```

1- import java.util.Scanner;
2
3- public class Main {
4-     public static void main(String[] args) {
5-         Scanner scanner = new Scanner(System.in);
6-         int correctPin = 1234;
7-         double balance = 3000.0;
8
9-         System.out.print("Enter PIN: ");
10-        int enteredPin = scanner.nextInt();
11
12-        try {
13-            if (enteredPin != correctPin) {
14-                throw new Exception("Invalid PIN.");
15-            }
16-            System.out.print("Withdraw Amount: ");
17-            double withdrawAmount = scanner.nextDouble();
18-            int enteredPin = scanner.nextInt();
19-            if (withdrawAmount > balance) {
20-                throw new Exception("Insufficient balance.");
21-            }
22-            balance -= withdrawAmount;
23-            System.out.println("Withdrawal successful. Remaining Balance: "
24-            + balance);
25-        } catch (Exception e) {
26-            System.out.println("Error: " + e.getMessage() + " Current
27-            ");
28-        }
29-    }
30- }
  
```

Output: Enter PIN: 1234
Withdraw Amount: 200
Withdrawal successful. Remaining Balance: 2800.0
--- Code Execution Successful ---

(c)



```

1- import java.util.ArrayList;
2- import java.util.List;
3- import java.util.Scanner;
4
5- class PrerequisiteNotMetException extends Exception {
6-     public PrerequisiteNotMetException(String message) {
7-         super(message);
8-     }
9- }
10
11- class Course {
12-     String name;
13-     List<String> prerequisites;
14
15-     Course(String name, List<String> prerequisites) {
16-         this.name = name;
17-         this.prerequisites = prerequisites;
18-     }
19
20-     void enroll(String student, List<String> completedCourses) throws
21-     PrerequisiteNotMetException {
22-         for (String prereq : prerequisites) {
23-             if (!completedCourses.contains(prereq)) {
24-                 throw new PrerequisiteNotMetException("Complete " +
25-                 prereq + " before enrolling in " + name + ".");
26-             }
27-         }
28-         System.out.println("Successfully enrolled in " + name);
29-     }
30- }
  
```

Output: Enroll in Course: Advanced Java
Successfully enrolled in Advanced Java
--- Code Execution Successful ---

(b)

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.