Experiment - 3

Student Name: Garvi Dabas **UID:** 23BCS11346
Branch: BE-CSE **Section/Group:** KRG-2B
Semester: 5th **Date of Performance:**26/8/25
Subject Name: Project Based Learning in Java
Subject Code: 23CSH-304

Aim: To develop Java programs to manage product details, library systems, and student information using classes, inheritance, and abstraction.

Easy-level Problem-

Aim: To write a Java program to calculate the square root of a number entered by the user. The program should use **try-catch** to handle invalid inputs (negative numbers or non-numeric values).

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

Procedure:

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

Sample Input -
Enter a number:
-16

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

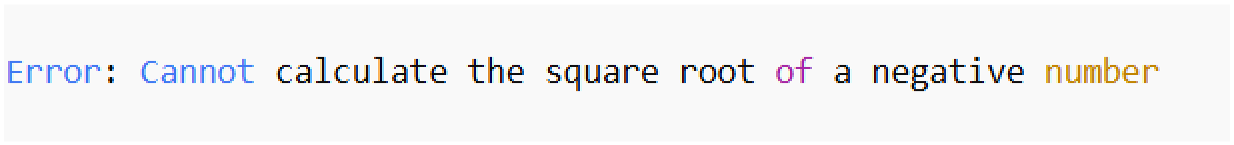
<u>Code -</u>

```
package exp1;

import java.util.Scanner;

public class Easy {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            double num = sc.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 (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

<u>Output</u> -



```
Error: Cannot calculate the square root of a negative number
```

Medium- Level Problem -

Aim : To write a Java program to simulate an ATM withdrawal system using exception handling.

Objective: To implement nested try-catch blocks and create meaningful exception messages.

Procedure:

1. Prompt user to enter ATM PIN.
2. Verify PIN.
3. If correct, allow withdrawal.
4. Check balance before withdrawal; if insufficient, throw InsufficientBalanceException.
5. Use finally block to always display remaining balance.

Code :

```
package exp1;
import java.util.Scanner;
class InsufficientBalanceException extends Exception {
public InsufficientBalanceException(String message) {
super(message);
}
}
public class Medium {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int correctPin = 1234;
double balance = 3000;
try {
System.out.print("Enter PIN: ");
int pin = sc.nextInt();
if (pin != correctPin) {
throw new SecurityException("Error: Invalid PIN");
}
System.out.print("Enter withdrawal amount: ");
double amount = sc.nextDouble();

if (amount > balance) {
throw new InsufficientBalanceException("Error: Insufficient balance.");
}
balance -= amount;
System.out.println("Withdrawal successful! Current Balance: " + balance);

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

Output:

```
Error: Insufficient balance.
Current Balance: 3000
```

Conclusion:

1. Java uses exceptions to handle errors during runtime.
2. try-catch allows catching and handling errors.
3. finally ensures execution of code regardless of exceptions.
4. throw is used to manually throw exceptions.
5. Custom exceptions allow handling business logic errors.