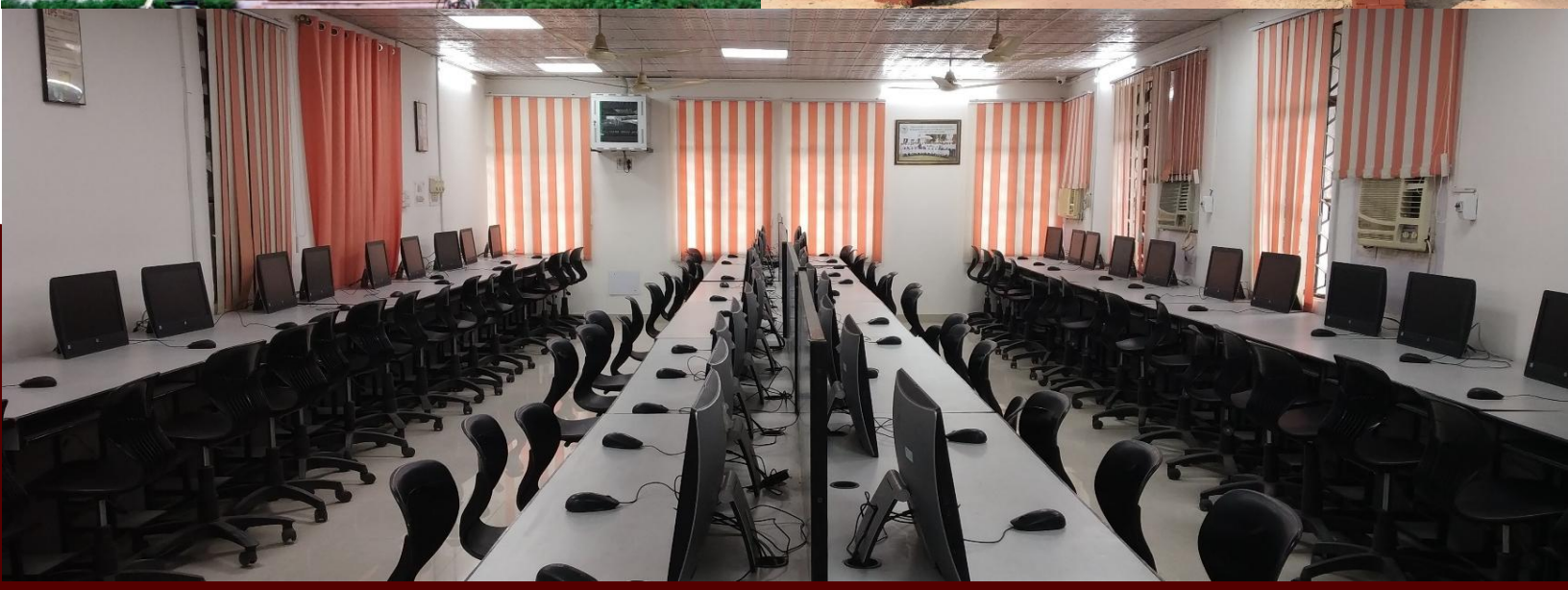
جب کوئی قوم فن اور علم سے عاری ہو جاتی ہے تو وہ غربت کو دعوت دیتی ہے اور جب غربت آتی ہے تو وہ ہزاروں جرائم کو جنم دیتی ہے۔

"When a nation becomes devoid of art and learning, it invites poverty and when poverty comes it brings in its wake thousands of crimes."
-Sir Syed Ahmad Khan

MCA II Semester Lab Manual



Laboratory Course-II

Course Code- CAMS2P01

DEPARTMENT OF COMPUTER SCIENCE

ALIGARH MUSLIM UNIVERSITY, ALIGARH

2025-2026

Credits

The following lab manual up-gradation committee updated the lab manual:

- ❒ *Prof.Arman Rasool Faridi (Chairperson)*
- ❒ *Prof. Mohammad UbaidUllah Bokhari*
- ❒ *Prof. Aasim Zafar*
- ❒ *Prof. Suhel Mustajab*
- ❒ *Dr. Faisal Anwar*
- ❒ *Dr. Mohammad Sajid*
- ❒ *Dr. Mohammad Nadeem*
- ❒ *Dr. Faraz Masood*

The following committee member originally design the lab manual:

- ❒ *Prof. Mohammad Ubaidullah Bokhari*
- ❒ *Dr. Arman Rasool Faridi*
- ❒ *Dr. Faisal Anwer*
- ❒ *Prof. AasimZafar (Convener)*

Design & Compilation:

- ❒ *Dr. Faraz Masood*

Revised Edition: January, 2026

Department of Computer Science, A.M.U.,

Aligarh (U.P.) India

COURSE TITLE: Laboratory Course-II **COURSE CODE:** CAMS2P01
CREDIT: 04 **PERIODS PER WEEK:** 06
CONTINUOUS ASSESSMENT: 40 **EXAMS:** 60

COURSE DESCRIPTION

This course is designed to help students in learning JAVA using object-oriented paradigm. Approach in this course is to take JAVA as a language that is used as a primary tool in many different areas of programming work.

COURSE CONTENT

This course is designed to provide the students the opportunity to learn the differences between C++ and JAVA programming and to develop, debug, and execute JAVA programs.

OBJECTIVES

This course is designed to help students in:

- ❒ Gaining knowledge about basic Java language syntax and semantics to write Java programs and use concepts such as variables, conditional and iterative execution methods etc.
- ❒ Understanding the fundamentals of object-oriented programming in Java, including defining classes, objects, invoking methods, etc., and exception handling mechanisms.
- ❒ Learning the use of arrays, access protection, wrapper classes etc.
- ❒ Understanding the principles of inheritance, packages, and interfaces.
- ❒ Learning to create applet and application and event handling, AWT controls, able to create GUI (frame, menu, button, text boxes, layout manager).

- ❒ Familiarizing with the important topics and principles of software development.
- ❒ Learning the ability to write a computer program to solve specified problems.
- ❒ Understanding to use the Java SDK environment to create, debug and run simple Java programs.

OUTCOMES

After completing this course, the students would be able to:

- ❒ Understand the fundamental features of an object-oriented language like JAVA: object classes and interfaces, exceptions, and libraries of object collections.
- ❒ Understand how to implement, compile, test, and run JAVA programs comprising more than one class to address a particular software problem.
- ❒ Understand how to include arithmetic operators and constants in a JAVA program and able to deploy the JAVA applet and application program.
- ❒ Understand the concept of multithreading, multitasking, and multiprogramming.
- ❒ Understand the use of members of classes found in the JAVA API (such as the Math class, Wrapper classes).
- ❒ Understand the implementation of the keyboard/mouse events.
- ❒ Understand the concept of object-oriented programming, inheritance, constructors, interfaces, and packages.
- ❒ Understand the concept of Exception Handling, Files and Streams, Applets and Graphics, and also learn the concept of Applet classes, Applet life cycle and JAVA Swing (introductory part).
- ❒ How to take the statement of a business problem and, from this, determine suitable logic for solving the problem; then, be able to code that logic as a program written in JAVA.

RULES AND REGULATIONS

Students are required to strictly adhere to the following rules.

- The students must complete the weekly activities/assignments well in time (i.e., within the same week) that need to be checked and signed by the concerned teachers in the lab in the immediate succeeding week. Failing which activities/assignments for that week will be treated as incomplete.
- The students must maintain the Lab File of their completed activities/assignments in the prescribed format (**Appendix-1**).
- At least **TEN (10)** such timely completed and duly signed weekly activities/assignments are compulsory, failing which students will not be allowed to appear in the final Lab Examination.
- The students need to submit the following three deliverables for each exercise duly signed by the Teacher:
  - Coding
  - Input /Output
- Late submissions would not be accepted after the due date.
- Cooperate, collaborate, and explore for the best individual learning outcomes, but copying is strictly prohibited.
- The Continuous Lab assessment will be based on two sessionals, each carrying 20 marks.
- Marks distribution for each sessional:
  - 10 Marks: For a duly signed lab report by the respective lab teacher.
  - 5 Marks: For solving a lab question/program on the day of the sessional.
  - 5 Marks: For the viva conducted during the sessional.
- This distribution ensures a balanced evaluation of students' practical work, problem-solving skills, and conceptual understanding.

APPENDIX-1

Template for the Index of Lab File

WEEK NO.	PROBLEMS WITH DESCRIPTION		PAGE NO.	SIGNATURE OF THE TEACHER WITH DATE
1	1#			
	2#			
	3#			
2	1#			
	2#			
	3#			
3	1#			
	2#			
	3#			

Note: The students should use Header and Footer mentioning their roll no. & name in footer and page no in header.

WEEK #1

E C

- To help the students understand the importance of programming in Object Oriented environment using JAVA
- To help the students in understanding the basic structure of JAVA Program
- To help students get familiar with the JAVA programming environment and IDE.
- To learn the students in writing, debugging and executing JAVA programs.
- How to read input from the keyboard?
- To learn the constant and variables.

T C

After completing this, the students would be able to:

- Write, debug, and execute simple JAVA programs.
- Use the constant and variables in JAVA.

SCENARIO

Scenario: *Student Information System*

You are tasked with developing a simple Java application for a Student Information System. The system should be able to store and display student details, perform basic arithmetic operations, and handle user inputs.

Problems

***Problem** 1: Student Address Display*

Scenario:

You are working on a module to display the address of a student. The system should take the student's address as input and print it.

Test Cases:

1. **Input:** "123 Maple Street, Aligarh, India"
   Output: "123 Maple Street, Aligarh, India"
2. **Input:** "456 Oak Avenue, Delhi, India"
   Output: "456 Oak Avenue, Delhi, India"
3. **Input:** "789 Pine Road, Mumbai, India"
   Output: "789 Pine Road, Mumbai, India"

***Problem [1-9]:** Arithmetic Operations*

Scenario:

You need to develop a module that performs basic arithmetic operations (sum, difference, product, quotient, minimum, and maximum) on two integers provided by the user.

Test Cases:

1. **Input:** num1 = 10, num2 = 5
   Output:
   Sum: 15
   Difference: 5
   Product: 50
   Quotient: 2
   Minimum: 5
   Maximum: 10
2. **Input:** num1 = -3, num2 = 7
   Output:
   Sum: 4
   Difference: -10
   Product: -21

Quotient: -0.42857142857142855
Minimum: -3
Maximum: 7

3. **Input:** num1 = 0, num2 = 0
   Output:
   Sum: 0
   Difference: 0
   Product: 0
   Quotient: 0.0
   Minimum: 0
   Maximum: 0

***Problem** 3: Temperature Conversion*

Scenario:

The system needs a module to convert temperatures from Fahrenheit to Celsius. The formula for conversion is C= $F-32/1.8$

. The result should be displayed in a tabular form.

Test Cases:1

S. No.	Input (Fahrenheit)	Output (Celsius)
1	105	40.6
2	50	10.0
3	-40	-40
4	32	0.0
5	0	-17.8

Test Cases: 2

S. No.	Input (Fahrenheit)	Output (Celsius)
1	257	125
2	318.2	159
3	111.2	44

***Problem** 4: Number Operations*

Scenario:

You are required to write a program that performs various operations on a double variable num. The operations include finding the round, ceil, floor values, and casting to an integer.

Test Cases:

1. **Input:** num = 3.7
   Output:
   numRound: 4.0
   numCeil: 4.0
   numFloor: 3.0
   numInteger: 3
2. **Input:** num = -2.3
   Output:
   numRound: -2.0
   numCeil: -2.0
   numFloor: -3.0
   numInteger: -2
3. **Input:** num = 5.0
   Output:
   numRound: 5.0
   numCeil: 5.0
   numFloor: 5.0
   numInteger: 5

***Problem** 5: Math Class Usage*

Scenario:

Develop a module that demonstrates the usage of various methods from the Math class in Java. The program should showcase at least five different methods (e.g., Math.abs, Math.sqrt, Math.pow, Math.sin, Math.cos).

Test Cases:

1. **Input:** num = 9
   Output:
   Absolute value: 9
   Square root: 3.0
   Power (num^2): 81
   Sine value: 0.4121184852417566
   Cosine value: 0.9111302618846767
2. **Input:** num = -4
   Output:
   Absolute value: 4
   Square root: 2.0
   Power (num^2): 16
   Sine value: -0.7568024953079282
   Cosine value: 0.6536436208636119
3. **Input:** num = 0
   Output:
   Absolute value: 0
   Square root: 0.0
   Power (num^2): 0
   Sine value: 0.0
   Cosine value: 1.0

WEEK #2

OBJECTIVE

- To learn using JAVA programming coding: data types, variable, constants, operators, Control Statement (*if, switch, loops*)
- To learn break, continue statements, ternary operator, bit-wise operators, user-defined data types in JAVA, order of evaluation of different operators in Java.
- To learn the controls statements and loops.

OUTCOME

After completing this,

- The students would be able to use different control statements and loops available in JAVA.

SCENARIO

Scenario: *Basic Arithmetic and Control Flow*
You are tasked with developing a simple Java application that performs basic arithmetic operations and uses control flow statements to handle different scenarios.

Problems

***Problem** 1: Odd and Even Counter*

Scenario:

You are working on a module for a school project that counts the number of odd and even numbers entered by students during a math quiz.

Test Cases:

1. **Input:** 1, 2, 3, 4, 5
   Output:
   Odd count: 3
   Even count: 2
2. **Input:** 2, 4, 6, 8, 10
   Output:
   Odd count: 0
   Even count: 5
3. **Input:** 1, 3, 5, 7, 9
   Output:
   Odd count: 5
   Even count: 0

***Problem** 2: Squares and Cubes Table*

Scenario:

You need to develop a module that prints the squares and cubes of numbers from 1 to 5 for a math tutorial application.

Test Cases:

1. **Input:** None
   Output:
2. TableCopy
3.

Number	Square	Cube

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

***Problem** 3: Sum of Reciprocals*
Scenario:

You are tasked with writing a program to compute the sum of the reciprocals of numbers from 1 to a given number for a math research project.

Test Cases:

1. **Input:** 5
   Output:
   Sum of reciprocals: 2.283333333333333
2. **Input:** 10
   Output:
   Sum of reciprocals: 2.9289682539682538
3. **Input:** 1
   Output:
   Sum of reciprocals: 1.0

***Problem** 4: Sum of Even Numbers*
Scenario:

You need to develop a module that computes the sum of even numbers up to a given number for a financial analysis tool.

Test Cases:

1. **Input:** 10
   Output:
   Sum of even numbers: 30
2. **Input:** 5
   Output:
   Sum of even numbers: 6
3. **Input:** 1
   Output:
   Sum of even numbers: 0

***Problem** 5: Floyd's Triangle*
Scenario:

You are working on a module to print Floyd's triangle for a math visualization tool. Additionally, you need to modify the program to print an alternating pattern of 0s and 1s.

Test Cases:

1. **Input:** 5
   Output:
   Floyd's Triangle:
   1
   2 3
   4 5 6
   7 8 9 10
   11 12 13 14 15
   Alternating Pattern:
   1
   0 1
   1 0 1
   0 1 0 1
   1 0 1 0 1
2. **Input:** 3
   Output:

Floyd's Triangle:
1
2 3
4 5 6
Alternating Pattern:
1
0 1
1 0 1

3. **Input:** 1
   Output:
   Floyd's Triangle:
   1
   Alternating Pattern:
   1

WEEK #3

- To learn using JAVA programming coding: data types, variable, constants, operators, Control Statement (*if, switch, loops*)
- To learn break, continue statements, ternary operator, bit-wise operators, user-defined data types in JAVA, order of evaluation of different operators in Java.
- To learn the controls statements and loops.

After completing this,

- The students would be able to use different control statements and
- loops available in JAVA

.

SCENARIO

Scenario: *Basic Arithmetic and Control Flow*

You are tasked with developing a simple Java application that performs basic arithmetic operations and uses control flow statements to handle different scenarios.

***Problem** 1: Fibonacci Sequence*

Scenario:

You are working on a module for a math tutorial application that calculates and prints the first m Fibonacci numbers using a do...while loop.

Test Cases:

1. **Input:** m=5
   Output:
   Fibonacci sequence: 0, 1, 1, 2, 3
2. **Input:** m=10
   Output:
   Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
3. **Input:** m=1
   Output:
   Fibonacci sequence: 0

***Problem** 2: Combination of Digits*
Scenario:

You need to develop a module that accepts three digits and prints all possible combinations of these digits for a lottery system.

Test Cases:

1. **Input:** Digits = 1, 2, 3
   Output:
   Combinations: 123, 132, 213, 231, 312, 321
2. **Input:** Digits = 4, 5, 6
   Output:
   Combinations: 456, 465, 546, 564, 645, 654
3. **Input:** Digits = 7, 8, 9
   Output:
   Combinations: 789, 798, 879, 897, 978, 987

***Problem** 3: Sum and Product of Numbers*
Scenario:

You are tasked with writing a program that prompts the user to enter 4 numbers, then computes and displays their sum and product for a financial analysis tool.

Test Cases:

1. **Input:** Numbers = 1, 2, 3, 4
   Output:
   Sum: 10
   Product: 24
2. **Input:** Numbers = -1, -2, -3, -4
   Output:
   Sum: -10
   Product: 24
3. **Input:** Numbers = 0, 0, 0, 0
   Output:
   Sum: 0
   Product: 0

***Problem** 4: Digit Separator*
Scenario:

You need to develop a module that reads a 4-digit number and prints the digits on separate lines for a data entry application.

Test Cases:

1. **Input:** Number = 1234
   Output:
   1
   2
   3
   4
2. **Input:** Number = 5678
   Output:
   5
   6
   7
   8
3. **Input:** Number = 9012
   Output:

9
0
1
2

***Problem** 5: Rectangle Intersection*
Scenario:

You are working on a module for a graphics application that computes the intersection of two rectangles. The program should handle cases where rectangles do not overlap.

Test Cases:

1. **Input:** Rectangle 1 (x1=0, y1=0, width1=5, height1=5), Rectangle 2 (x2=3, y2=3, width2=5, height2=5)
   Output:
   Intersection: (3, 3, 2, 2)
2. **Input:** Rectangle 1 (x1=0, y1=0, width1=3, height1=3), Rectangle 2 (x2=4, y2=4, width2=2, height2=2)
   Output:
   No intersection
3. **Input:** Rectangle 1 (x1=1, y1=1, width1=4, height1=4), Rectangle 2 (x2=2, y2=2, width2=3, height2=3)
   Output:
   Intersection: (2, 2, 2, 2)

WEEK #4

E C

- To learn the object oriented features (classification, encapsulation, inheritance, polymorphism, abstraction) and Java features like secure, platform independent, portable, threading.
- To learn the concept of class like how to declare a class, how to define methods inside the class, how the access the properties of base class into derived class, Code reusability.
- To learn the use of arrays, access protection, wrapper classes.
- To learn the use of this keyword, use of toString ( ) method.

P T

After completing this, the students would be able to:

- employ various types of selection constructs in a JAVA program
- demonstrate the hierarchy of JAVA classes to provide a solution to a given set of requirements.

\

SCENARIO

Scenario: *Basic Object-Oriented Concepts*

You are tasked with developing a simple Java application that demonstrates basic object-oriented programming concepts such as classes, methods, encapsulation, and inheritance.

Problems

***Problem** 1: Odd and Even Counter*

Scenario:

You are working on a module for a school project that counts the number of odd and even numbers entered by students during a math quiz. The module should use a class OddAndEven to manage the count and a TestOddAndEven class to test the functionality.

Test Cases:

1. **Input:** Numbers = 1, 2, 3, 4, 5
   Output:
   Number of Odd: 3
   Number of Even: 2
2. **Input:** Numbers = 2, 4, 6, 8, 10
   Output:
   Number of Odd: 0
   Number of Even: 5
3. **Input:** Numbers = 1, 3, 5, 7, 9
   Output:
   Number of Odd: 5
   Number of Even: 0

***Problem** 2: Circle Class*

Scenario:

You need to develop a module for a geometry application that calculates the area and perimeter of a circle. Additionally, the module should check if a given point is inside the circle.

Test Cases:

1. **Input:** Radius = 5, Center = (0, 0), Point = (3, 4)
   Output:
   Area: 78.53981633974483
   Perimeter: 31.41592653589793
   Point (3, 4) is inside the circle.
2. **Input:** Radius = 3, Center = (1, 1), Point = (4, 4)
   Output:
   Area: 28.274333882308138
   Perimeter: 18.84955592153876
   Point (4, 4) is outside the circle.
3. **Input:** Radius = 10, Center = (5, 5), Point = (5, 5)
   Output:
   Area: 314.1592653589793
   Perimeter: 62.83185307179586
   Point (5, 5) is on the circle.

***Problem** 3: Text Analysis*
Scenario:

You are tasked with writing a program for a text analysis tool that counts the number of words, sentences, and occurrences of specific letters in a given text.

Test Cases:

1. **Input:** Text = "Hello world! This is a test."
   Output:
   Number of words: 6
   Number of sentences: 1
   Number of 'e': 2
   Number of 'z': 0
2. **Input:** Text = "Java is fun. Java is powerful. Java is everywhere."
   Output:
   Number of words: 12
   Number of sentences: 3
   Number of 'e': 6
   Number of 'z': 0

3. **Input:** Text = "The quick brown fox jumps over the lazy dog."
   Output:
   Number of words: 9
   Number of sentences: 1
   Number of 'e': 3
   Number of 'z': 0

***Problem** 4: Sales Commission Calculator*

Scenario:

You need to develop a module for a sales management application that calculates the commission based on sales figures. The commission rates are as follows:

- Under 500: 2% of sales
- Between 500 and 5000: 5% of sales
- 5000 and above: 8% of sales

Test Cases:

1. **Input:** Sales = 300
   Output:
   Commission: 6.0
2. **Input:** Sales = 2500
   Output:
   Commission: 125.0
3. **Input:** Sales = 6000
   Output:
   Commission: 480.0
4. **Input:** Sales = -100
   Output:
   Invalid Input

WEEK #5

E C

- To learn the object oriented features (classification, encapsulation, inheritance, polymorphism, abstraction) and Java features like secure, platform independent, portable, threading.
- To learn the concept of class like how to declare a class, how to define methods inside the class, how the access the properties of base class into derived class, Code reusability.
- To learn the use of arrays, access protection, wrapper classes.
- To learn the use of this keyword, use of toString ( ) method.

P T

After completing this, the students would be able to:

- employ various types of selection constructs in a JAVA program
- demonstrate the hierarchy of JAVA classes to provide a solution to a given set of requirements.

SCENARIO

Scenario: *Library Management System*

You are tasked with developing a Java application for a library management system. The system should manage books, users, and the catalog, allowing users to borrow and return books.

Problems

***Problem** 1: Book Class*

Scenario:

You need to design a Book class to represent individual books in the library. The class should include attributes such as book ID, title, author, genre, and availability status. Implement appropriate methods, including getters and setters.

Test Cases:

1. **Input:** Book ID = 1, Title = "Java Basics", Author = "John Doe", Genre = "Programming", Available = true
   Output:
   Book ID: 1
   Title: Java Basics
   Author: John Doe
   Genre: Programming
   Available: true
2. **Input:** Book ID = 2, Title = "Advanced Java", Author = "Jane Smith", Genre = "Programming", Available = false
   Output:
   Book ID: 2
   Title: Advanced Java
   Author: Jane Smith
   Genre: Programming
   Available: false

***Problem** 2: User Class*

Scenario:

You need to design a User class to represent library users. The class should include attributes such as user ID, name, and a list to store borrowed books. Implement methods to borrow and return books.

Test Cases:

1. **Input:** User ID = 101, Name = "Alice", Borrow Book ID = 1
   Output:
   User ID: 101
   Name: Alice
   Borrowed Books: [1]
2. **Input:** User ID = 102, Name = "Bob", Borrow Book ID = 2, Return Book ID = 2
   Output:
   User ID: 102
   Name: Bob
   Borrowed Books: []

***Problem** 3: Catalog Class*
Scenario:

You need to design a Catalog class to represent the library catalog. The class should use collections (e.g., ArrayList) to store a collection of books. Implement methods to add books to the catalog, display available books, and update book status.

Test Cases:

1. **Input:** Add Book ID = 1, Title = "Java Basics", Author = "John Doe", Genre = "Programming", Available = true
   Output:
   Catalog: [Book ID: 1, Title: Java Basics, Author: John Doe, Genre: Programming, Available: true]
2. **Input:** Add Book ID = 2, Title = "Advanced Java", Author = "Jane Smith", Genre = "Programming", Available = false
   Output:
   Catalog: [Book ID: 1, Title: Java Basics, Author: John Doe, Genre: Programming, Available: true, Book ID: 2, Title: Advanced Java, Author: Jane Smith, Genre: Programming, Available: false]

3. **Input:** Display Available Books
   Output:
   Available Books: [Book ID: 1, Title: Java Basics, Author: John Doe, Genre: Programming, Available: true]

***Problem** 4: Encapsulation*
Scenario:

Ensure that proper encapsulation is applied to protect the internal state of the Book and User classes. Use access modifiers to restrict direct access to attributes.

Test Cases:

1. **Input:** Attempt to directly access Book attributes without using getters/setters
   Output:
   Compilation Error: Cannot access private attributes directly.
2. **Input:** Attempt to directly access User attributes without using getters/setters
   Output:
   Compilation Error: Cannot access private attributes directly.

***Problem** 5: Inheritance and Polymorphism*
Scenario:

Utilize inheritance to create a more specific type of book (e.g., FictionBook, NonFictionBook) that extends the base Book class. Demonstrate polymorphism through a method like displayDetails() that can provide different information based on the book type.

Test Cases:

1. **Input:** Create FictionBook with Book ID = 3, Title = "Fiction Novel", Author = "Fiction Author", Genre = "Fiction", Available = true
   Output:
   Book ID: 3
   Title: Fiction Novel
   Author: Fiction Author
   Genre: Fiction

Available: true
Type: Fiction

2. **Input:** Create NonFictionBook with Book ID = 4, Title = "Non-Fiction Guide", Author = "Non-Fiction Author", Genre = "Non-Fiction", Available = true
   Output:
   Book ID: 4
   Title: Non-Fiction Guide
   Author: Non-Fiction Author
   Genre: Non-Fiction
   Available: true
   Type: Non-Fiction

***Problem** 6: Main Program*
Scenario:

In the main program, instantiate the classes, add books to the catalog, create users, and simulate the borrowing and returning of books. Display the catalog's status after each operation.

Test Cases:

1. **Input:** Add Book ID = 1, Title = "Java Basics", Author = "John Doe", Genre = "Programming", Available = true
   User ID = 101, Name = "Alice", Borrow Book ID = 1
   Output:
   Catalog: [Book ID: 1, Title: Java Basics, Author: John Doe, Genre: Programming, Available: false]
   User ID: 101
   Name: Alice
   Borrowed Books: [1]
2. **Input:** Add Book ID = 2, Title = "Advanced Java", Author = "Jane Smith", Genre = "Programming", Available = true
   User ID = 102, Name = "Bob", Borrow Book ID = 2, Return Book ID = 2
   Output:
   Catalog: [Book ID: 1, Title: Java Basics, Author: John Doe, Genre: Programming, Available: false, Book ID: 2, Title: Advanced Java, Author:

Jane Smith, Genre: Programming, Available: true]
User ID: 102
Name: Bob
Borrowed Books: []

WEEK #6

E C

- To learn the concept Interface, how to implement it,
- To learn the use of member functions and how we access them, using interfaces for code reusing,
- To learn converting between class and interface types, using interfaces for callbacks; Polymorphism, Inheritance

T C

After completing this, the students would be able to:

- handle interfaces, converting between class and interface types, callbacks, Polymorphism, Inheritance, Inheriting instance fields and methods.
- access the member functions of a class.

SCENARIO

Scenario: *Text Processing and Shape Management System*

You are tasked with developing a Java application that processes text and manages shapes using interfaces and polymorphism.

Problems

***Problem** 1: Text Reversal*

Scenario:

You are working on a text processing module for a language learning application. The module should read a sentence from the user and print it out with each word reversed, but with the words and punctuation in the original order.

Test Cases:

1. **Input:** "Hello world!"
   Output: "olleH !dlrow"
2. **Input:** "This is a test."
   Output: "sihT si a .tset"
3. **Input:** "Java is fun."
   Output: "avaJ si .nuf"

***Problem** 2: Multiple Inheritance with Interfaces*

Scenario:

You need to develop a module for a software design tool that supports multiple inheritances using interfaces. Implement a standalone Java program that demonstrates this concept.

Test Cases:

1. **Input:** Interface A with method methodA(), Interface B with method methodB(), Class C implementing both A and B
   Output:
   Class C should be able to call both methodA() and methodB().
2. **Input:** Interface X with method methodX(), Interface Y with method methodY(), Class Z implementing both X and Y

Output:
Class Z should be able to call both methodX() and methodY().

***Problem** 3: Lexicographical Sorting*
Scenario:

You are working on a module for a text analysis tool that reads three strings and sorts them lexicographically.

Test Cases:

1. **Input:** Strings = "Charlie", "Able", "Banker"
   Output:
   Sorted Strings: Able, Banker, Charlie
2. **Input:** Strings = "Delta", "Alpha", "Echo"
   Output:
   Sorted Strings: Alpha, Delta, Echo
3. **Input:** Strings = "Zulu", "Golf", "Bravo"
   Output:
   Sorted Strings: Bravo, Golf, Zulu

***Problem** 4: Shape Management*
Scenario:

You need to develop a module for a graphics application that manages different shapes using an interface. Implement classes for shapes (e.g., Circle, Rectangle, Triangle) and use an interface for common methods like draw() and getArea().

Test Cases:

1. **Input:** Create a Circle with radius = 5
   Output:
   Area: 78.53981633974483
   Drawing Circle...
2. **Input:** Create a Rectangle with width = 4, height = 5
   Output:
   Area: 20.0
   Drawing Rectangle...

3. **Input:** Create a Triangle with base = 3, height = 4
   Output:
   Area: 6.0
   Drawing Triangle...

WEEK #7

OBJECTIVE

- Assess file handling, exception handling, and design practices in Java.
- Evaluate the use of try-catch blocks, FileInputStream, BufferedReader, and FileOutputStream.
- Ensure the candidate can implement a complete and effective solution.

OUTCOME

After completing this, the students would be able to:

- Candidates will demonstrate proficiency in file handling, exception handling, and design practices in Java.
- Candidates will effectively utilize try-catch blocks, FileInputStream, BufferedReader, and FileOutputStream.
- Candidates will implement complete and functional solutions following proper coding standards.

SCENARIO

Scenario: *Data Processing System*

You are tasked with developing a Java application that processes numeric data from text files, calculates the average, and handles various exceptions that may occur during file operations.

Problems

***Problem** 1: Data Processor*

Scenario:

You are working on a data processing module for a financial analysis tool. The module should read numeric values from a text file, calculate the average of these values, and output the result to another text file. The program should handle file-related issues and invalid data using exception handling.

Test Cases:

1. **Input:** File input.txt containing "10 20 30 40 50"
   Output:
   File output.txt containing "30.0"
2. **Input:** File input.txt containing "100 200 300"
   Output:
   File output.txt containing "200.0"
3. **Input:** File input.txt containing "abc 20 30"
   Output:
   Exception: InvalidDataException - "abc" is not a valid number.
4. **Input:** File input.txt does not exist
   Output:
   Exception: FileNotFoundException - The file input.txt does not exist.
5. **Input:** File input.txt is empty
   Output:
   Exception: IllegalArgumentException - No numeric values found in the file.

***Problem** 2: Data Validation*

Scenario:

You need to develop a module for a data validation tool that reads numeric values from a text file and validates them. Invalid data should be handled using a custom exception.

Test Cases:

1. **Input:** File data.txt containing "10 20 30 40 50"
   Output:
   Valid values: [10.0, 20.0, 30.0, 40.0, 50.0]
2. **Input:** File data.txt containing "100 200 abc 300"
   Output:
   Valid values: [100.0, 200.0, 300.0]
   Exception: InvalidDataException - "abc" is not a valid number.
3. **Input:** File data.txt containing "10.5 20.7 30.9"
   Output:
   Valid values: [10.5, 20.7, 30.9]

***Problem** 3: Average Calculation*
Scenario:

You are tasked with writing a module for a statistical analysis tool that calculates the average of valid numeric values read from a text file.

Test Cases:

1. **Input:** Valid values: [10.0, 20.0, 30.0, 40.0, 50.0]
   Output:
   Average: 30.0
2. **Input:** Valid values: [100.0, 200.0, 300.0]
   Output:
   Average: 200.0
3. **Input:** Valid values: [10.5, 20.7, 30.9]
   Output:
   Average: 20.7

***Problem** 4: File Output*
Scenario:

You need to develop a module for a data reporting tool that writes the calculated average of numeric values to another text file. The module should handle file writing exceptions.

Test Cases:

1. **Input:** Average = 30.0, Output file result.txt
   Output:
   File result.txt containing "30.0"
2. **Input:** Average = 200.0, Output file result.txt
   Output:
   File result.txt containing "200.0"
3. **Input:** Average = 20.7, Output file result.txt
   Output:
   File result.txt containing "20.7"
4. **Input:** Output file result.txt is not writable
   Output:
   Exception: IOException - Could not write to the file result.txt.

***Problem** 5: Main Program*
Scenario:

In the main program, instantiate the DataProcessor class, call the methods in sequence to read values, validate data, calculate the average, and write the result to another file. Handle any exceptions that occur during the process.

Test Cases:

1. **Input:** File input.txt containing "10 20 30 40 50", Output file output.txt
   Output:
   File output.txt containing "30.0"
2. **Input:** File input.txt containing "100 200 abc 300", Output file output.txt
   Output:
   File output.txt containing "200.0"
   Exception: InvalidDataException - "abc" is not a valid number.
3. **Input:** File input.txt does not exist, Output file output.txt
   Output:
   Exception: FileNotFoundException - The file input.txt does not exist.

4. **Input:** File input.txt is empty, Output file output.txt
   Output:
   Exception: IllegalArgumentException - No numeric values found in the file.

***Problem** 6: Comprehensive Exception Handling*
Scenario:

You are tasked with developing a comprehensive exception handling module for a data processing application. The module should handle various exceptions, including file not found, invalid data, and I/O errors, and provide meaningful feedback to the user.

Test Cases:

1. **Input:** File input.txt containing "10 20 30 40 50", Output file output.txt
   Output:
   File output.txt containing "30.0"
2. **Input:** File input.txt containing "100 200 abc 300", Output file output.txt
   Output:
   File output.txt containing "200.0"
   Exception: InvalidDataException - "abc" is not a valid number.
3. **Input:** File input.txt does not exist, Output file output.txt
   Output:
   Exception: FileNotFoundException - The file input.txt does not exist.
4. **Input:** File input.txt is empty, Output file output.txt
   Output:
   Exception: IllegalArgumentException - No numeric values found in the file.
5. **Input:** Output file output.txt is not writable
   Output:
   Exception: IOException - Could not write to the file output.txt.

WEEK #8

E C

- ❒ Assess understanding of exception handling in Java.
- ❒ Evaluate the ability to develop a program that counts exceptions in a Java program.
- ❒ Test the ability to report the total number of exceptions identified.

T C

After completing this, the students would be able to:

- ❒ Candidates will demonstrate a clear understanding of exception handling concepts in Java.
- ❒ Candidates will develop a functional program capable of counting exceptions in a Java program.
- ❒ Candidates will accurately report the total number of exceptions identified in the program..

SCENARIO

Scenario: *Exception Handling in a Java Application*
You are tasked with developing a Java application that demonstrates advanced exception handling techniques. The application should handle various exceptions, count exceptions in a given Java program, and manage severe runtime exceptions gracefully.

Problems

***Problem** 1: Exception Counter*

Scenario:

You are working on a module for a code analysis tool that counts the total number of exceptions in a given Java program file. The tool should read the specified Java file, analyze its content, and count occurrences of exception-related keywords or patterns.

Test Cases:

1. **Input:** Java file containing try { throw new Exception(); } catch (Exception e) { }
   Output:
   Total exceptions: 2
2. **Input:** Java file containing if (true) throw new RuntimeException();
   Output:
   Total exceptions: 1
3. **Input:** Java file containing throw new IllegalArgumentException();
   Output:
   Total exceptions: 1
4. **Input:** Java file containing try { } catch (Exception e) { throw new Exception(); }
   Output:
   Total exceptions: 2

***Problem** 2: Null Pointer Exception Handling*

Scenario:

You need to develop a module for a debugging tool that identifies, handles, or prevents Null Pointer Exceptions in a Java program. Implement three different versions of the program using the provided blocks of code to demonstrate different approaches to handling Null Pointer Exceptions.

Test Cases:

1. Version A:
   Input: String str = null; System.out.println(str.length());

Output:
Exception: java.lang.NullPointerException

2. Version B:
   Input: String str = null; if (str != null) System.out.println(str.length());
   Output:
   No output (no exception)
3. Version C:
   Input: String str = null; try { System.out.println(str.length()); } catch (NullPointerException e) { System.out.println("Caught NullPointerException"); }
   Output:
   Caught NullPointerException

***Problem** 3: Exception Handling in a Financial Application*
Scenario:

You are tasked with writing a module for a financial application that handles severe runtime exceptions gracefully. The module should log the exception details and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Logged: ArithmeticException occurred.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5]; System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Logged: ArrayIndexOutOfBoundsException occurred.
3. **Input:** Attempt to use a null object: String str = null; System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Logged: NullPointerException occurred.

***Problem** 4: Logging Framework Integration*
Scenario:

You need to develop a module for a logging system that integrates a logging framework (e.g., Log4j) to log exception details. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5]; System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
3. **Input:** Attempt to use a null object: String str = null; System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.

***Problem** 5: Custom Exception Handling*
Scenario:

You are tasked with writing a module for a data processing application that throws a custom exception when encountering invalid data. The module should handle the custom exception gracefully and log the details.

Test Cases:

1. **Input:** Valid data: "10 20 30"
   Output:
   No exception

2. **Input:** Invalid data: "10 abc 30"
   Output:
   Exception: InvalidDataException - "abc" is not a valid number.
   Log: [ERROR] InvalidDataException occurred.
3. **Input:** Empty data: ""
   Output:
   Exception: IllegalArgumentException - No data provided.
   Log: [ERROR] IllegalArgumentException occurred.

***Problem** 6: Advanced Exception Handling in a Database Application*
Scenario:

You need to develop a module for a database application that handles checked exceptions related to database connections. The module should use advanced techniques like "try-with-resources" and custom exceptions to manage database connections effectively.

Test Cases:

1. **Input:** Valid database connection details
   Output:
   Connection successful
   Connection closed
2. **Input:** Invalid database connection details
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Log: [ERROR] DatabaseConnectionException occurred.
3. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: SQLException - An error occurred during the database operation.
   Log: [ERROR] SQLException occurred.
   Connection closed

WEEK #9

E C

- ❒ Assess understanding of logging and reporting in Java, specifically for unchecked exceptions using logging frameworks.
- ❒ Evaluate the ability to handle severe runtime exceptions in Java.
- ❒ Test the ability to implement graceful termination to prevent further damage or data corruption.

E C

After completing this, the students would be able to:

- ❒ Candidates will demonstrate an understanding of logging and reporting unchecked exceptions using logging frameworks in Java.
- ❒ Candidates will effectively handle severe runtime exceptions in Java.
- ❒ Candidates will implement graceful termination techniques to prevent further damage or data corruption.

SCENARIO

Scenario: *Advanced Exception Handling and Logging*
You are tasked with developing a Java application that demonstrates advanced exception handling techniques, logging of unchecked exceptions, and graceful termination of the application.

Problems

***Problem** 1: Unchecked Exception Handling*
Scenario:

You are working on a module for a file processing tool that simulates a scenario where an unchecked exception might occur. The module should handle the exception using a try-catch block and log the details using a logging framework.

Test Cases:

1. **Input:** Attempt to read from a null file object: File file = null; file.getName();
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.
2. **Input:** Attempt to access an index out of bounds: int[] array = new int[5]; System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
3. **Input:** Attempt to use a null object: String str = null; System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.

***Problem** 2: Logging Framework Integration*
Scenario:

You need to develop a module for a logging system that integrates a logging framework (e.g., Log4j) to log the details of unchecked exceptions. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5];
   System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
3. **Input:** Attempt to use a null object: String str = null;
   System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.

***Problem** 3: Graceful Termination*
Scenario:

You are tasked with writing a module for a file processing tool that handles severe runtime exceptions gracefully. The module should log the exception details and terminate the application with an error code.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
   Application terminated with error code 1.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5];
   System.out.println(array[10]);

Output:
Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
Application terminated with error code 1.

3. **Input:** Attempt to use a null object: String str = null;
   System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.
   Application terminated with error code 1.

***Problem** 4: Severe Runtime Exception Handling*
Scenario:

You need to develop a module for a critical system that handles severe runtime exceptions. The module should log the exception details and terminate the application gracefully using System.exit().

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
   Application terminated with error code 1.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5];
   System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
   Application terminated with error code 1.
3. **Input:** Attempt to use a null object: String str = null;
   System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException

Log: [ERROR] NullPointerException occurred.
Application terminated with error code 1.

***Problem** 5: Logging and Reporting*
Scenario:

You are tasked with writing a module for a data processing tool that logs the details of unchecked exceptions using a logging framework. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5]; System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
3. **Input:** Attempt to use a null object: String str = null; System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.

***Problem** 6: Comprehensive Exception Handling*
Scenario:

You need to develop a comprehensive exception handling module for a critical system that handles various unchecked exceptions, logs the details, and terminates the application gracefully.

Test Cases:

1. **Input:** Attempt to divide by zero: int result = 10 / 0;
   Output:
   Exception: java.lang.ArithmeticException: / by zero
   Log: [ERROR] ArithmeticException occurred.
   Application terminated with error code 1.
2. **Input:** Attempt to access an array out of bounds: int[] array = new int[5];
   System.out.println(array[10]);
   Output:
   Exception: java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
   Log: [ERROR] ArrayIndexOutOfBoundsException occurred.
   Application terminated with error code 1.
3. **Input:** Attempt to use a null object: String str = null;
   System.out.println(str.length());
   Output:
   Exception: java.lang.NullPointerException
   Log: [ERROR] NullPointerException occurred.
   Application terminated with error code 1.

***Problem** 7: Custom Exception Handling*
Scenario:

You are tasked with writing a module for a data processing application that throws a custom exception when encountering invalid data. The module should handle the custom exception gracefully and log the details.

Test Cases:

1. **Input:** Valid data: "10 20 30"
   Output:
   No exception
2. **Input:** Invalid data: "10 abc 30"
   Output:
   Exception: InvalidDataException - "abc" is not a valid number.
   Log: [ERROR] InvalidDataException occurred.
3. **Input:** Empty data: ""
   Output:

Exception: IllegalArgumentException - No data provided.
Log: [ERROR] IllegalArgumentException occurred.

WEEK #10

- ❒ Assess mastery of handling checked exceptions in Java.
- ❒ Evaluate the ability to manage database connections effectively in Java.
- ❒ Test proficiency in utilizing advanced techniques for exception handling and resource management.

After completing this, the students would be able to:

- ❒ Candidates will demonstrate mastery in handling checked exceptions in Java.
- ❒ Candidates will effectively manage database connections in Java.
- ❒ Candidates will apply advanced techniques for exception handling and resource management.

SCENARIO

Scenario: *Database Management System*

You are tasked with developing a Java application that manages database connections and handles checked exceptions effectively. The application should use advanced techniques for exception handling and resource management.

Problems

***Problem** 1: Database Connection Management*

Scenario:

You are working on a module for a database management system that handles checked exceptions related to opening a database connection. The module should use the "try-with-resources" statement to manage the Connection object and ensure proper closure of the connection, even if an exception occurs.

Test Cases:

1. **Input:** Valid database connection details
   Output:
   Connection successful
   Connection closed
2. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: SQLException - Could not connect to the database.
   Connection closed
3. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: SQLException - An error occurred during the database operation.
   Connection closed

***Problem** 2: Custom Exception for Database Issues*

Scenario:

You need to develop a module for a database management system that introduces a custom exception class for representing database connection-related issues. The module should throw this custom exception instead of the generic SQLException when encountering database connection **Problem**s.

Test Cases:

1. **Input:** Valid database connection details
   Output:

Connection successful
Connection closed

2. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Connection closed
3. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: DatabaseConnectionException - An error occurred during the database operation.
   Connection closed

***Problem** 3: Logging Database Connection Issues*
Scenario:

You are tasked with writing a module for a database management system that logs the details of database connection issues using a logging framework. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Valid database connection details
   Output:
   Connection successful
   Connection closed
2. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Log: [ERROR] DatabaseConnectionException occurred.
   Connection closed
3. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: DatabaseConnectionException - An error occurred during the

database operation.
Log: [ERROR] DatabaseConnectionException occurred.
Connection closed

***Problem** 4: Advanced Exception Handling with Finally Block*
Scenario:

You need to develop a module for a database management system that uses advanced exception handling techniques, including a finally block to log a message indicating the successful closure of the database connection, regardless of whether an exception occurred.

Test Cases:

1. **Input:** Valid database connection details
   Output:
   Connection successful
   Connection closed
   Log: [INFO] Database connection closed successfully.
2. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Log: [ERROR] DatabaseConnectionException occurred.
   Log: [INFO] Database connection closed successfully.
3. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: DatabaseConnectionException - An error occurred during the database operation.
   Log: [ERROR] DatabaseConnectionException occurred.
   Log: [INFO] Database connection closed successfully.

***Problem** 5: Handling Checked Exceptions in Database Operations*
Scenario:

You are tasked with writing a module for a database management system that handles checked exceptions related to database operations. The module should use advanced techniques for exception handling and resource management.

Test Cases:

1. **Input:** Valid database connection details and successful operation
   Output:
   Operation successful
   Connection closed
2. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: DatabaseOperationException - An error occurred during the database operation.
   Connection closed
3. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Connection closed

***Problem** 6: Custom Exception for Database Operations*
Scenario:

You need to develop a module for a database management system that introduces a custom exception class for representing database operation-related issues. The module should throw this custom exception instead of the generic SQLException when encountering database operation **Problem**s.

Test Cases:

1. **Input:** Valid database connection details and successful operation
   Output:
   Operation successful
   Connection closed
2. **Input:** Valid database connection details, but an exception occurs during the operation

Output:
Exception: DatabaseOperationException - An error occurred during the database operation.
Connection closed

3. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Connection closed

***Problem** 7: Comprehensive Exception Handling in Database Management*
Scenario:

You are tasked with writing a comprehensive exception handling module for a database management system that handles various checked exceptions, logs the details, and ensures proper closure of database connections.

Test Cases:

1. **Input:** Valid database connection details and successful operation
   Output:
   Operation successful
   Connection closed
   Log: [INFO] Database connection closed successfully.
2. **Input:** Valid database connection details, but an exception occurs during the operation
   Output:
   Exception: DatabaseOperationException - An error occurred during the database operation.
   Log: [ERROR] DatabaseOperationException occurred.
   Log: [INFO] Database connection closed successfully.
3. **Input:** Invalid database connection details (e.g., wrong URL)
   Output:
   Exception: DatabaseConnectionException - Could not connect to the database.
   Log: [ERROR] DatabaseConnectionException occurred.
   Log: [INFO] Database connection closed successfully.

WEEK #11

E C

- ❒ Assess advanced understanding of handling unchecked exceptions in Java.
- ❒ Evaluate the ability to handle exceptions during date parsing in Java.
- ❒ Test proficiency in employing advanced techniques for exception handling and providing meaningful feedback.

T C

ter completing this, the students would be able to:

- ❒ Candidates will demonstrate an advanced understanding of handling unchecked exceptions in Java.
- ❒ Candidates will effectively handle exceptions that occur during date parsing in Java.
- ❒ Candidates will apply advanced techniques for exception handling and provide meaningful feedback.

SCENARIO

Scenario: *Date Management System*

You are tasked with developing a Java application that manages date-related operations and handles exceptions effectively. The application should use advanced techniques for exception handling and resource management.

Problems

***Problem** 1: Date Parsing with Custom Exception*

Scenario:

You are working on a module for a date management system that handles checked exceptions related to parsing dates. The module should use the "try-with-resources" statement to manage the DateFormat object and ensure proper closure of the formatter, even if an exception occurs.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
2. **Input:** Invalid date string "2024-13-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-13-01"
   Formatter closed
3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed

***Problem** 2: Custom Exception for Date Parsing Issues*

Scenario:

You need to develop a module for a date management system that introduces a custom exception class for representing date parsing-related issues. The module should throw this custom exception instead of the generic ParseException when encountering date parsing **Problem**s.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:

Parsed Date: Wed Dec 25 00:00:00 UTC 2024
Formatter closed

2. **Input:** Invalid date string "2024-14-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-14-01"
   Formatter closed
3. **Input:** Valid date string "2024-11-01"
   Output:
   Parsed Date: Fri Nov 01 00:00:00 UTC 2024
   Formatter closed

***Problem** 3: Logging Date Parsing Issues*
Scenario:

You are tasked with writing a module for a date management system that logs the details of date parsing issues using a logging framework. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
2. **Input:** Invalid date string "2024-13-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-13-01"
   Log: [ERROR] DateParseException occurred.
   Formatter closed
3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed

***Problem** 4: Advanced Exception Handling with Finally Block*
Scenario:

You need to develop a module for a date management system that uses advanced exception handling techniques, including a finally block to log a message indicating the successful closure of the DateFormat object, regardless of whether an exception occurred.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
   Log: [INFO] DateFormat object closed successfully.
2. **Input:** Invalid date string "2024-13-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-13-01"
   Log: [ERROR] DateParseException occurred.
   Log: [INFO] DateFormat object closed successfully.
3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed
   Log: [INFO] DateFormat object closed successfully.

***Problem** 5: Handling Checked Exceptions in Date Operations*
Scenario:

You are tasked with writing a module for a date management system that handles checked exceptions related to date operations. The module should use advanced techniques for exception handling and resource management.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
2. **Input:** Invalid date string "2024-13-01"
   Output:

Exception: DateParseException - Unparseable date: "2024-13-01"
Formatter closed

3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed

***Problem** 6: Custom Exception for Date Operations*
Scenario:

You need to develop a module for a date management system that introduces a custom exception class for representing date operation-related issues. The module should throw this custom exception instead of the generic ParseException when encountering date operation **Problem**s.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
2. **Input:** Invalid date string "2024-13-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-13-01"
   Formatter closed
3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed

***Problem** 7: Comprehensive Exception Handling in Date Management*
Scenario:

You are tasked with writing a comprehensive exception handling module for a date management system that handles various checked exceptions, logs the details, and ensures proper closure of date formatters.

Test Cases:

1. **Input:** Valid date string "2024-12-25"
   Output:
   Parsed Date: Wed Dec 25 00:00:00 UTC 2024
   Formatter closed
   Log: [INFO] DateFormat object closed successfully.
2. **Input:** Invalid date string "2024-13-01"
   Output:
   Exception: DateParseException - Unparseable date: "2024-13-01"
   Log: [ERROR] DateParseException occurred.
   Log: [INFO] DateFormat object closed successfully.
3. **Input:** Valid date string "2024-01-01"
   Output:
   Parsed Date: Mon Jan 01 00:00:00 UTC 2024
   Formatter closed
   Log: [INFO] DateFormat object closed successfully.

WEEK #12

E C

- ❒ Evaluate advanced understanding of handling checked exceptions in Java.
- ❒ Assess the ability to handle network connections and exceptions effectively in Java.
- ❒ Test proficiency in utilizing advanced exception handling techniques, providing meaningful feedback, and managing resources.

T U

After completing this, the students would be able to:

- ❒ Candidates will demonstrate an advanced understanding of handling checked exceptions in Java.
- ❒ Candidates will effectively manage network connections and handle related exceptions in Java.
- ❒ Candidates will apply advanced exception handling techniques, provide meaningful feedback, and manage resources effectively..

SCENARIO

Scenario: *Network Management System*
You are tasked with developing a Java application that manages network connections and handles exceptions effectively. The application should use advanced techniques for exception handling and resource management.

Problems

***Problem** 1: Network Connection Management*

Scenario:

You are working on a module for a network management system that handles checked exceptions related to establishing network connections. The module should use the "try-with-resources" statement to manage the HttpURLConnection object and ensure proper closure of the connection, even if an exception occurs.

Test Cases:

1. **Input:** Valid URL "http://example.com"
   Output:
   Connection successful
   Connection closed
2. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: IOException - Could not connect to the URL.
   Connection closed
3. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: IOException - An error occurred during the network operation.
   Connection closed

***Problem** 2: Custom Exception for Network Issues*

Scenario:

You need to develop a module for a network management system that introduces a custom exception class for representing network connection-related issues. The module should throw this custom exception instead of the generic IOException when encountering network connection **Problem**s.

Test Cases:

1. **Input:** Valid URL "http://example.com"
   Output:

Connection successful
Connection closed

2. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Connection closed
3. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkConnectionException - An error occurred during the network operation.
   Connection closed

***Problem** 3: Logging Network Connection Issues*
Scenario:

You are tasked with writing a module for a network management system that logs the details of network connection issues using a logging framework. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Valid URL "http://example.com"
   Output:
   Connection successful
   Connection closed
2. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Log: [ERROR] NetworkConnectionException occurred.
   Connection closed
3. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkConnectionException - An error occurred during the network operation.

Log: [ERROR] NetworkConnectionException occurred.
Connection closed

***Problem** 4: Advanced Exception Handling with Finally Block*
Scenario:

You need to develop a module for a network management system that uses advanced exception handling techniques, including a finally block to log a message indicating the successful closure of the HttpURLConnection object, regardless of whether an exception occurred.

Test Cases:

1. **Input:** Valid URL "http://example.com"
   Output:
   Connection successful
   Connection closed
   Log: [INFO] HttpURLConnection object closed successfully.
2. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Log: [ERROR] NetworkConnectionException occurred.
   Log: [INFO] HttpURLConnection object closed successfully.
3. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkConnectionException - An error occurred during the network operation.
   Log: [ERROR] NetworkConnectionException occurred.
   Log: [INFO] HttpURLConnection object closed successfully.

***Problem** 5: Handling Checked Exceptions in Network Operations*
Scenario:

You are tasked with writing a module for a network management system that handles checked exceptions related to network operations. The module should use advanced techniques for exception handling and resource management.

Test Cases:

1. **Input:** Valid URL "http://example.com" and successful operation
   Output:
   Operation successful
   Connection closed
2. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkOperationException - An error occurred during the network operation.
   Connection closed
3. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Connection closed

***Problem** 6: Custom Exception for Network Operations*
Scenario:

You need to develop a module for a network management system that introduces a custom exception class for representing network operation-related issues. The module should throw this custom exception instead of the generic IOException when encountering network operation **Problem**s.

Test Cases:

1. **Input:** Valid URL "http://example.com" and successful operation
   Output:
   Operation successful
   Connection closed
2. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkOperationException - An error occurred during the network operation.
   Connection closed

3. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Connection closed

***Problem** 7: Comprehensive Exception Handling in Network Management*
Scenario:

You are tasked with writing a comprehensive exception handling module for a network management system that handles various checked exceptions, logs the details, and ensures proper closure of network connections.

Test Cases:

1. **Input:** Valid URL "http://example.com" and successful operation
   Output:
   Operation successful
   Connection closed
   Log: [INFO] HttpURLConnection object closed successfully.
2. **Input:** Valid URL "http://example.com", but an exception occurs during the operation
   Output:
   Exception: NetworkOperationException - An error occurred during the network operation.
   Log: [ERROR] NetworkOperationException occurred.
   Log: [INFO] HttpURLConnection object closed successfully.
3. **Input:** Invalid URL "http://invalid-url"
   Output:
   Exception: NetworkConnectionException - Could not connect to the URL.
   Log: [ERROR] NetworkConnectionException occurred.
   Log: [INFO] HttpURLConnection object closed successfully.

WEEK #13

E C

- Assess advanced understanding of handling checked exceptions when dealing with file input in Java.
- Evaluate the ability to apply advanced techniques for exception handling and provide meaningful feedback during file input operations.
- Test proficiency in managing resources effectively during file input in Java.

T C

After completing this, the students would be able to:

- Assess advanced understanding of handling checked exceptions when dealing with file input in Java.
- Evaluate the ability to apply advanced techniques for exception handling and provide meaningful feedback during file input operations.
- Test proficiency in managing resources effectively during file input in Java.

SCENARIO

Scenario: File Management System

You are tasked with developing a Java application that manages file input operations and handles exceptions effectively. The application should use advanced techniques for exception handling and resource management.

Problems

***Problem** 1: File Input Management*

Scenario:

You are working on a module for a file management system that handles checked exceptions related to file input operations. The module should use the "try-with-resources" statement to manage the FileReader object and ensure proper closure of the file reader, even if an exception occurs.

Test Cases:

1. **Input:** Valid file path "input.txt" containing "Hello, World!"
   Output:
   File content: Hello, World!
   File reader closed
2. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileNotFoundException - The file nonexistent.txt does not exist.
   File reader closed
3. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: IOException - An error occurred during file reading.
   File reader closed

***Problem** 2: Custom Exception for File Input Issues*

Scenario:

You need to develop a module for a file management system that introduces a custom exception class for representing file input-related issues. The module should throw this custom exception instead of the generic FileNotFoundException when encountering file input **Problem**s.

Test Cases:

1. **Input:** Valid file path "input.txt" containing "Hello, World!"
   Output:
   File content: Hello, World!
   File reader closed
2. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileInputException - The file nonexistent.txt does not exist.
   File reader closed
3. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: FileInputException - An error occurred during file reading.
   File reader closed

***Problem** 3: Logging File Input Issues*
Scenario:

You are tasked with writing a module for a file management system that logs the details of file input issues using a logging framework. The module should handle exceptions gracefully and provide meaningful feedback to the user.

Test Cases:

1. **Input:** Valid file path "input.txt" containing "Hello, World!"
   Output:
   File content: Hello, World!
   File reader closed
2. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileInputException - The file nonexistent.txt does not exist.
   Log: [ERROR] FileInputException occurred.
   File reader closed
3. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:

Exception: FileInputException - An error occurred during file reading.
Log: [ERROR] FileInputException occurred.
File reader closed

***Problem** 4: Advanced Exception Handling with Finally Block*
Scenario:

You need to develop a module for a file management system that uses advanced exception handling techniques, including a finally block to log a message indicating the successful closure of the FileReader object, regardless of whether an exception occurred.

Test Cases:

1. **Input:** Valid file path "input.txt" containing "Hello, World!"
   Output:
   File content: Hello, World!
   File reader closed
   Log: [INFO] FileReader object closed successfully.
2. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileInputException - The file nonexistent.txt does not exist.
   Log: [ERROR] FileInputException occurred.
   Log: [INFO] FileReader object closed successfully.
3. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: FileInputException - An error occurred during file reading.
   Log: [ERROR] FileInputException occurred.
   Log: [INFO] FileReader object closed successfully.

***Problem** 5: Handling Checked Exceptions in File Operations*
Scenario:

You are tasked with writing a module for a file management system that handles checked exceptions related to file operations. The module should use advanced techniques for exception handling and resource management.

Test Cases:

1. **Input:** Valid file path "input.txt" and successful operation
   Output:
   Operation successful
   File reader closed
2. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: FileInputException - An error occurred during file reading.
   File reader closed
3. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileInputException - The file nonexistent.txt does not exist.
   File reader closed

***Problem** 6: Custom Exception for File Operations*
Scenario:

You need to develop a module for a file management system that introduces a custom exception class for representing file operation-related issues. The module should throw this custom exception instead of the generic IOException when encountering file operation **Problem**s.

Test Cases:

1. **Input:** Valid file path "input.txt" and successful operation
   Output:
   Operation successful
   File reader closed
2. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: FileInputException - An error occurred during file reading.
   File reader closed
3. **Input:** Invalid file path "nonexistent.txt"
   Output:

Exception: FileInputException - The file nonexistent.txt does not exist.
File reader closed

***Problem** 7: Comprehensive Exception Handling in File Management*
Scenario:

You are tasked with writing a comprehensive exception handling module for a file management system that handles various checked exceptions, logs the details, and ensures proper closure of file readers.

Test Cases:

1. **Input:** Valid file path "input.txt" and successful operation
   Output:
   Operation successful
   File reader closed
   Log: [INFO] FileReader object closed successfully.
2. **Input:** Valid file path "input.txt", but an exception occurs during the operation (e.g., file is empty)
   Output:
   Exception: FileInputException - An error occurred during file reading.
   Log: [ERROR] FileInputException occurred.
   Log: [INFO] FileReader object closed successfully.
3. **Input:** Invalid file path "nonexistent.txt"
   Output:
   Exception: FileInputException - The file nonexistent.txt does not exist.
   Log: [ERROR] FileInputException occurred.
   Log: [INFO] FileReader object closed successfully.

WEEK #14

- Assess understanding of the Particle Swarm Optimization (PSO) algorithm.
- Evaluate the ability to implement the PSO algorithm in Java.
- Test the ability to apply PSO for solving a simple example **Problem** in Java.

er completing this, the students would be able to:

- Candidates will demonstrate a clear understanding of the Particle Swarm Optimization (PSO) algorithm.
- Candidates will effectively implement the PSO algorithm in Java.
- Candidates will apply the PSO algorithm to solve a simple example **Problem** in Java.

SCENARIO

Scenario: *Optimization System*

You are tasked with developing a Java application that implements the Particle Swarm Optimization (PSO) algorithm to solve optimization **Problem**s. The application should be able to customize key parameters and provide meaningful feedback on the optimization process.

Problems

***Problem** 1: Implementing PSO Algorithm*

Scenario:

You are working on a module for an optimization system that implements the Particle Swarm Optimization (PSO) algorithm. The module should allow customization of key parameters such as the number of particles, iterations, and any algorithm-specific parameters.

Test Cases:

1. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 10, Iterations = 100
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0
2. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 20, Iterations = 200
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0
3. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 5, Iterations = 50
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0

***Problem** 2: Customizing PSO Parameters*

Scenario:

You need to develop a module for an optimization system that allows users to customize key parameters of the PSO algorithm, such as the number of particles, iterations, and any algorithm-specific parameters.

Test Cases:

1. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 10, Iterations = 100, Cognitive coefficient = 2.0, Social coefficient = 2.0

Output:
Optimized solution: x=0.0
Fitness value: f(x)=0.0

2. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 20, Iterations = 200, Cognitive coefficient = 1.5, Social coefficient = 1.5
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0
3. **Input:** Simple mathematical function $f(x)=x^2$, Number of particles = 5, Iterations = 50, Cognitive coefficient = 2.5, Social coefficient = 2.5
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0

***Problem** 3: Fitness Function Implementation*
Scenario:

You are tasked with writing a module for an optimization system that defines the fitness function specific to the chosen example **Problem** within the PSO algorithm.

Test Cases:

1. **Input:** Simple mathematical function $f(x)=x^2$
   Output:
   Fitness function: $f(x)=x^2$
2. **Input:** `Simple mathematical function` $f(x)=(x-5)^2$
   Output:
   `Fitness function:` $f(x)=(x-5)^2$
3. **Input:** Simple mathematical function f(x)=sin(x)
   Output:
   Fitness function: f(x)=sin(x)

***Problem** 4: PSO Algorithm with Custom Fitness Function*
Scenario:

You need to develop a module for an optimization system that implements the PSO algorithm with a custom fitness function. The module should allow customization of key parameters and provide meaningful feedback on the optimization process.

Test Cases:

1. **Input:** `Custom fitness function f(x)=(x−3)`^{2}, Number of particles = 10, Iterations = 100
   Output:
   Optimized solution: x=3.0
   Fitness value: f(x)=0.0
2. **Input:** `Custom fitness function f(x)=(x−5)`^{2}, Number of particles = 20, Iterations = 200
   Output:
   Optimized solution: x=5.0
   Fitness value: f(x)=0.0
3. **Input:** Custom fitness function f(x)=sin(x) , Number of particles = 5, Iterations = 50
   Output:
   Optimized solution: x=1.5707963267948966 (approx. $\pi/2$)
   Fitness value: f(x)=1.0

***Problem** 5: PSO Algorithm with Multiple Dimensions*
Scenario:

You are tasked with writing a module for an optimization system that implements the PSO algorithm for a multi-dimensional **Problem**. The module should allow customization of key parameters and provide meaningful feedback on the optimization process.

Test Cases:

1. **Input:** Multi-dimensional function f(x,y)=x^2+y^2, Number of particles = 10, Iterations = 100
   Output:
   Optimized solution: (x,y)=(0.0,0.0)
   Fitness value: f(x,y)=0.0

2. **Input:** Multi-dimensional function $f(x,y)=(x-3)^2+(y-4)^2$, Number of particles = 20, Iterations = 200
   Output:
   Optimized solution: (x,y)=(3.0,4.0)
   Fitness value: f(x,y)=0.0
3. **Input:** Multi-dimensional function f(x,y)=sin(x)+cos(y) , Number of particles = 5, Iterations = 50
   Output:
   Optimized solution: (x,y)=(1.5707963267948966,0.0) (approx. π/2)
   Fitness value: f(x,y)=1.0

***Problem** 6: PSO Algorithm with Constraints*

Scenario:

You need to develop a module for an optimization system that implements the PSO algorithm with constraints. The module should allow customization of key parameters and provide meaningful feedback on the optimization process.

Test Cases:

1. **Input:** Constrained function $f(x)=x^2$, with $0 \le x \le 10$, Number of particles = 10, Iterations = 100
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0
2. **Input:** Constrained function $f(x)=(x-5)^2$, with $0 \le x \le 10$, Number of particles = 20, Iterations = 200
   Output:
   Optimized solution: x=5.0
   Fitness value: f(x)=0.0
3. **Input:** Constrained function $f(x)=\sin(x)$ with $0 \le x \le 2\pi$, Number of particles = 5, Iterations = 50
   Output:
   Optimized solution: x=1.5707963267948966 (approx. π/2)
   Fitness value: f(x)=1.0

***Problem 7**: Comprehensive PSO Implementation*

Scenario:

You are tasked with writing a comprehensive PSO implementation for an optimization system that handles various types of optimization **Problem**s, logs the details, and provides meaningful feedback on the optimization process.

Test Cases:

1. **Input:** Simple mathematical function $f(x)=x^2$ Number of particles = 10, Iterations = 100
   Output:
   Optimized solution: x=0.0
   Fitness value: f(x)=0.0
   Log: [INFO] Optimization completed successfully.
2. **Input:** Multi-dimensional function $f(x,y)=x^2+y^2$, Number of particles = 20, Iterations = 200
   Output:
   Optimized solution: (x,y)=(0.0,0.0)
   Fitness value: f(x,y)=0.0
   Log: [INFO] Optimization completed successfully.
3. **Input:** Constrained function $f(x)=(x-5)^2$, with $0 \leq x \leq 10$, Number of particles = 5, Iterations = 50
   Output:
   Optimized solution: x=5.0
   Fitness value: f(x)=0.0
   Log: [INFO] Optimization completed successfully.