Department of
COMPUTER SCIENCE
AND ENGINEERING

23CS3352

Object Oriented Programming through Java

PVP23 Regulations

Prasad V. Potluri
Siddhartha Institute of Technology
(Autonomous)
Approved by AICTE and Affiliated to JNTU Kakinada
Sponsored By: Siddhartha Academy of General and Technical Education, Vijayawada

Lab 1: Java Basics

PRE-REQUISITES:

1. Java Development Kit (JDK)
2. Text Editor
3. Integrated Development Environment (IDE)

PRE LAB PREPARATION:

1) What are Java tokens?
2) Why java is platform independent?
3) What is JIT compiler?
4) What are the advantages of OOP?
5) What are the differences between C++ and Java?
6) Will the program run if we write static public void main?
7) Can Java be said to be the complete object-oriented programming language
8) Can you implement pointers in a Java Program?
9) Why main() method is public, static and void in java ?
10) What is bytecode in java ?

LAB PROGRAMS:

1. Given a number x find the square root of it. If x is not a perfect square, then print the floor value of square root of x.
2. Write a JAVA program that displays the roots of a quadratic equation $ax^2+bx=0$. Calculate the discriminate D and, based on D's value, describe the root's nature.
3. Write a Java program to check if a number is magic. A number is a magic number if the sum of its digits is repeatedly calculated until a single digit is obtained, and that single digit is 1.
4. Write a Java program to count the number of set bits (1s) in the binary representation of a given integer n.

Note: Accept Input as a Command Line Argument.

Lab 2: Arrays

Pre Lab Preparation:

1. What is an array and how is it used in Java?
2. What is type conversion and how is it used in Java?
3. What is casting and how is it used in Java?
4. What will happen if you do not initialize an Array?
5. What is the default value of Array in Java?
6. Can you declare an array without assigning the size of an array
7. We know that Arrays are objects so why cannot we write strArray.length()?
8. Can you remove a particular element from an array?
9. What is the default value of Array in Java?
10. Where is an Array stored in JVM memory?

Lab Programs:

1. Write a Java program to find the contiguous subarray within a given array that has the largest sum. For example, in the array [-2, 1, -3, 4, -1, 2, 1, -5, 4], the contiguous subarray with the largest sum is [4, -1, 2, 1], and the sum is 6. (O(n) Time Complexity or better).

   Other Problems based on the same logic

   a) Maximum Profit in Stock Trading

   You are given an array prices where prices[i] is the price of a given stock on the i-th day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Write a function to find the maximum profit you can achieve from this transaction. If no profit can be achieved, return 0.

   Example

   Input: prices = [7, 1, 5, 3, 6, 4] **Output:** 5

   Explanation: Buy on day 2 (price = 1), sell on day 5 (price = 6), and profit = 6 - 1 = 5.

   Input: prices = [7, 6, 4, 3, 1] **Output:** 0

   Explanation: No transactions are done in this case, and the max profit = 0.
2. Write a Java program to find and print all the leaders in a given array. An element is considered a leader if it is greater than all the elements to its right side.
3. Write a Java program to rearrange an array of integers so that all even numbers come before all odd numbers.

Lab 3: Recursion

PRE-LAB PREPARATION:

1. What Is Recursion?
2. Explain Nested If?
3. What are the Main Features Of OOPS?
4. What Is Pure Virtual Function?
5. What Is Coupling?
6. Explain the Scope of a Variable.
7. What You Mean by Type Casting?

LAB PROGRAMS:

1. Permutations of a String: Generate all permutations of a given string using recursion.
Example: Input: str = "ABC"
Output: ABC, ACB, BAC, BCA, CAB, CBA

2. Write a Java program that uses recursion to check if a given string is a palindrome. A palindrome is a string that reads the same backward as forward.

- The program should accept a string input.
- It should use a recursive method to determine if the string is a palindrome.
- The program should ignore case and non-alphanumeric characters.

3. Write a Java program that uses recursion to calculate the sum of a series of numbers from 1 to a given number n.

- The program should accept an integer n.
- A recursive method should be used to calculate the sum of all integers from 1 to n.
- The program should handle cases where n is zero or negative by returning 0.

Lab 4: Classes & Objects

Pre-Lab Preparation:

1. What is a class?
2. What is an object?
3. What is encapsulation?
4. What is Polymorphism?
5. What is Abstraction?
6. Do we have Copy Constructor in Java?
7. What are private constructors and where are they used?

Lab Programs

1. Design a class that can be used by a health care professional to keep track of a patient's vital statistics. Here's what the class should do:

- Design a class called Patient
    - Store a String name for the patient
    - Store weight and height for patient as doubles
- Constructor: Construct a new patient using these values
- Write a method called BMI which returns the patient's BMI as a double. BMI can be calculated as BMI = (Weight in Pounds / (Height in inches x Height in inches)) x 703
- Create a Patient object and assign some height and weight to that object. Display the BMI of that patient.

2. Bookstore Inventory Management

Design a Java program to manage the inventory of books in a bookstore with the following functionalities:

Class Structure:

- Define a class Book with the following attributes:
    - title: Title of the book (String).
    - author: Author of the book (String).
    - isbn: ISBN (International Standard Book Number) of the book (String).
    - price: Price of the book (double).

Methods:

- **Constructor:**

  You need to implement the following constructors:

  Default Constructor:

  Initializes title to "Unknown Title".
  Initializes author to "Unknown Author".
  Initializes isbn to "000-0-00-000000-0".
  Initializes price to 0.0.

Parameterized Constructor:

Takes title, author, and isbn as parameters and initializes price to a default value of 20.0.

Parameterized Constructor:

Takes title, author, isbn, and price as parameters and initializes all properties accordingly.

Copy Constructor:

Takes another Book object as a parameter and initializes a new Book object with the same property values as the passed object.

- **Update Price:**
  - Implement a method updatePrice(double newPrice) that updates the price of the book.
- **Display Information:**
  - Implement a method displayBookInfo() that displays detailed information about the book including title, author, ISBN, and price.

3. Product Inventory Management System

Design a Java program for managing products in an inventory using method overloading within the same class. Implement the following functionalities:

1. **Class Structure:**
   - **Product Class:**
     - Attributes:
       - productId: Unique identifier for each product (int).
       - productName: Name of the product (String).
       - price: Price of the product (double).
       - quantity: Quantity of the product in stock (int).
     - Constructors:
       - Constructor with four parameters (productId, productName, price, quantity).
       - Constructor with three parameters (productId, productName, price), initializing quantity to 0.
       - Constructor with two parameters (productId, productName), initializing price to 0.0 and quantity to 0.
     - Methods (Overloaded):
       - addStock(int quantity): Increases the stock quantity of the product.
       - sellStock(int quantity): Decreases the stock quantity of the product when sold.
       - sellStock(int quantity, double discount): Decreases the stock quantity of the product when sold with a discount, updating the price accordingly.

- displayProductDetails(): Displays details of the product including productId, productName, price, and quantity.

2. **Example Usage:**
   - Create instances of Product with different constructors.
   - Perform operations such as adding stock, selling stock with and without discounts.
   - Display product details to verify operations.

Lab 5: Strings

PRE-LAB PREPARATION:

1. What are the differences between primitive and non-primitive data types in Java, and where do `String` and `StringBuffer` fit in?
2. How does memory allocation differ between objects created using `new` and string literals in Java?
3. What are the basic operations of a stack, and how does the Last-In-First-Out (LIFO) principle affect string reversal?
4. Can you explain how character indexing works in Java strings, and how it's used to traverse a string with a loop?
5. What role do regular expressions (regex) play in string manipulation, and how is regex used with the `split()` method?
6. Why is mutability important when modifying strings, and how does it affect performance between `String` and `StringBuffer` operations?

LAB PROGRAMS

1: Character Swapping in Strings Using Java's String Class

Objective: To develop a Java program that swaps adjacent character pairs in a string, showcasing the immutability and traversal mechanisms of the `String` class.

Problem Statement: Given a string input, modify the string such that every pair of characters is swapped. For example, "apple" becomes "papel".

2: String Tokenization Using Java's Split Method

Objective: To create a Java program that breaks a sentence into individual words using the `split()` method.

Problem Statement: Split a user-entered sentence such as "The Earth needs our help" into: `["The", "Earth", "needs", "our", "help"]`.

3: Using StringBuffer for Efficient Text Manipulation

Objective: To demonstrate dynamic string manipulation using the mutable `StringBuffer` class.

Problem Statement: Build a program that takes a string and performs various operations—append, insert, replace, and delete—on it.

4: Reversing a String Using Stack and StringBuffer

Objective: To reverse a string using both the stack data structure and `StringBuffer` utilities in Java.

Problem Statement: Reverse a given string like “planet” into “tenalp” using the LIFO property of stacks.

Lab 6: Arrays

PRE-LAB PREPARATION:

1. How do you declare, initialize, and populate a one-dimensional array in Java?
2. What are the common sorting algorithms, and how do you implement one (e.g., bubble sort or selection sort) manually in Java?
3. What is the difference between copying an array by reference and copying its contents?

4. How is a two-dimensional array represented in Java, and how do you access its elements using nested loops?
5. What is scalar multiplication in matrix operations, and how do you apply it to each element of a 2D array in Java?
6. What are jagged arrays in Java, and how do you create and iterate through them with different row lengths?

7. What are the differences between arrays and vectors in Java in terms of memory allocation and resizing?
8. How do you convert an array into a Vector using loops or Java's utility methods?
9. What logic would you use to remove duplicate values from a Vector while preserving the order of elements?

LAB PROGRAMS

1: Sorting a One-Dimensional Array in Java

Objective: To develop a Java program that sorts elements in a one-dimensional array using a comparison-based algorithm.

Problem Statement: Given an unsorted array like `[4, 2, 9, 1, 7]`, sort its elements in ascending order.

2: Assigning One Array to Another in Java

Objective: To understand the difference between shallow copying and deep copying when assigning one array to another.

Problem Statement: Copy contents of array A into array B such that changes in B do not affect A.

3: Scalar Multiplication of a 2D Matrix

Objective: To perform scalar multiplication on a 2D matrix and store the results in a new matrix.

Problem Statement: Given a 2D array `matrix[3][3]` and scalar value `k`, compute `k * matrix[i][j]` for all elements.

4: Handling Jagged Arrays (Arrays of Varying Lengths) in Java

Objective: To design a Java program that reads and displays elements from jagged (non-rectangular) 2D arrays.

Problem Statement: Allow the user to input rows of different lengths, e.g., `[[1,2,3],[4,5],[6,7,8,9]]`.

5: Converting an Array to a Vector and Removing Duplicates

Objective: To create a Java program that converts an array to a `Vector` and removes duplicate elements.

Problem Statement: Convert `[1, 2, 2, 3, 4, 4]` to `[1, 2, 3, 4]` using dynamic structures.

6: Comparing One-Dimensional and Two-Dimensional Array Concepts

Objective: To synthesize the differences, strengths, and use cases between 1D and 2D arrays in Java.

Problem Statement: How do 1D arrays for sorting or assignment differ in structure and usage compared to 2D arrays used for matrix operations?

Lab 7: Inheritance

Pre-Lab Preparation:

1. What is Inheritance in Java?
2. Why do we need to use inheritance?
3. What is Is-A relationship in Java?
4. What is superclass and subclass?
5. Write the syntax for creating the subclass of a class.
6. Which class in Java is the superclass of every other class?
7. How will you prove that the features of the Superclass are inherited in the Subclass?
8. Can a class extend itself?
9. Can a class extend more than one class?

Lab Programs

1. Shape Hierarchy

Define a class Shape with the following attributes and methods:

- Attributes: color (String), filled (boolean).

Methods:

- Shape(): Default constructor initializing color to "green" and filled to true.
- Shape(String color, boolean filled): Parameterized constructor.
- getColor(): Getter method for color.
- setColor(String color): Setter method for color.
- isFilled(): Method to check if the shape is filled.
- setFilled(boolean filled): Method to set whether the shape is filled.

Define a subclass Circle that inherits from Shape with additional attributes and methods:

- Attributes: radius (double).

Methods:

- Circle(): Default constructor initializing radius to 1.0.
- Circle(double radius): Parameterized constructor.
- Circle(double radius, String color, boolean filled): Parameterized constructor invoking superclass constructor.
- getRadius(): Getter method for radius.
- setRadius(double radius): Setter method for radius.
- getArea(): Method to calculate and return the area of the circle (area = π * radius * radius).

- getPerimeter(): Method to calculate and return the perimeter of the circle (perimeter = 2 * π * radius).

2. Employee Hierarchy

Define a superclass Employee with the following attributes and methods:

- Attributes: name (String), age (int), designation (String).

Methods:

- Employee(String name, int age, String designation): Parameterized constructor to initialize attributes.
- getName(), setName(String name): Getter and setter methods for name.
- getAge(), setAge(int age): Getter and setter methods for age.
- getDesignation(), setDesignation(String designation): Getter and setter methods for designation.
- displayDetails(): Method to display details of the employee.

Define subclasses Manager and Developer that inherit from Employee:

Manager:

- Additional attribute: department (String).

Methods:

- Manager(String name, int age, String designation, String department): Parameterized constructor invoking superclass constructor.
- Override displayDetails() to include department information.

Developer:

- Additional attribute: programmingLanguage (String).

Methods:

- Developer(String name, int age, String designation, String programmingLanguage): Parameterized constructor invoking superclass constructor.
- Override displayDetails() to include programming language information.

3. Bank Account Hierarchy

Define a superclass Account with the following attributes and methods:

- Attributes: accountNumber (int), balance (double).

Methods:

- Account(int accountNumber, double balance): Parameterized constructor to initialize attributes.
- getAccountNumber(), setAccountNumber(int accountNumber): Getter and setter methods for accountNumber.
- getBalance(), setBalance(double balance): Getter and setter methods for balance.
- deposit(double amount): Method to add amount to the balance.

- withdraw(double amount): Method to subtract amount from the balance (considering validation for sufficient funds).
- displayAccountDetails(): Method to display account details (accountNumber and balance).

Define subclasses SavingsAccount and CurrentAccount that inherit from Account:

SavingsAccount:

- Additional attribute: interestRate (double).

Methods:

- SavingsAccount(int accountNumber, double balance, double interestRate): Parameterized constructor invoking superclass constructor.
- Override displayAccountDetails() to include interest rate information.

CurrentAccount:

- Additional attribute: overdraftLimit (double).

Methods:

- CurrentAccount(int accountNumber, double balance, double overdraftLimit): Parameterized constructor invoking superclass constructor.
- Override withdraw(double amount) to allow withdrawals beyond balance up to overdraft limit.
- Override displayAccountDetails() to include overdraft limit information.

Lab 8: Polymorphism and Packages

PRE LAB PREPARATION

1. What is an abstract class? How is it different from a regular class?
2. What is dynamic method dispatch? How does it relate to inheritance and polymorphism?
3. When should you use the final keyword with inheritance? Give an example.
4. What are the differences between a class and an interface in Java? Give an example of when you would use each.
5. How do you implement an interface in Java? Give an example.
6. Do I need to import java.lang package any time? Why?
7. Can I import same package/class twice? Will the JVM load the package twice at runtime?
8. What does the default access specifier means ?
9. What is the difference between public class and class ?

LAB PROGRAMS

1. E-commerce Platform

1. **Product Hierarchy**:
   - Create an abstract base class Product with the following attributes:
     - String productId
     - String name
     - double price
     - int quantity
   - Include abstract methods:
     - String getDetails()
     - double calculateDiscount()
2. **Derived Product Classes**:
   - Derive the following classes from Product:
     - Electronics with additional attributes String brand and int warrantyPeriod
     - Clothing with additional attributes String size and String material
     - Grocery with additional attributes String expiryDate and boolean isOrganic
   - Implement the abstract methods in each derived class:
     - getDetails() should return a string with all product details.
     - calculateDiscount() should provide a discount calculation based on specific criteria (e.g., 10% for electronics, 5% for clothing, and no discount for groceries).
3. **Shopping Cart**:
   - Create a ShoppingCart class with a list of Product objects.
   - Include methods to add and remove products from the cart.

- Implement a method double calculateTotal() to compute the total cost of the products in the cart, applying discounts as needed.

4. **Polymorphic Behavior**:
   - In the main application, demonstrate polymorphism by:
     - Creating instances of Electronics, Clothing, and Grocery.
     - Adding these instances to the ShoppingCart.
     - Iterating through the ShoppingCart to print product details and the total cost with discounts applied.

2. Library Management System

1. **Package Structure**:
   - Create a well-structured package hierarchy for the library management system. Suggested package structure:

```
library
├── books
│   ├── Book.java
│   ├── BookManager.java
├── members
│   ├── Member.java
│   ├── MemberManager.java
├── loans
│   ├── Loan.java
│   ├── LoanManager.java
├── reports
│   ├── ReportGenerator.java
└── main
    └── LibraryManagementSystem.java
```

2. **Package: library.books**:
   - **Book.java**: Create a Book class with attributes:
     - String isbn
     - String title
     - String author
     - int publicationYear
     - boolean isAvailable Include appropriate constructors, getters, setters, and a toString method.
   - **BookManager.java**: Create a BookManager class with methods to:
     - Add a new book
     - Remove a book
     - Search for a book by ISBN or title
     - Display all books
3. **Package: library.members**:
   - **Member.java**: Create a Member class with attributes:
     - String memberId
     - String name

        - String email
        - String phoneNumber Include appropriate constructors, getters, setters, and a toString method.
    - **MemberManager.java**: Create a MemberManager class with methods to:
        - Add a new member
        - Remove a member
        - Search for a member by ID or name
        - Display all members
4. **Package: library.loans**:
    - **Loan.java**: Create a Loan class with attributes:
        - String loanId
        - String memberId
        - String bookIsbn
        - String loanDate
        - String returnDate Include appropriate constructors, getters, setters, and a toString method.
    - **LoanManager.java**: Create a LoanManager class with methods to:
        - Issue a book to a member
        - Return a book
        - Display all active loans
5. **Package: library.reports**:
    - **ReportGenerator.java**: Create a ReportGenerator class with methods to:
        - Generate a report of all books
        - Generate a report of all members
        - Generate a report of all active loans
6. **Package: library.main**:
    - **LibraryManagementSystem.java**: Create a LibraryManagementSystem class with a main method to:
        - Demonstrate the functionality of the system
        - Allow interaction with the BookManager, MemberManager, LoanManager, and ReportGenerator classes

Lab 9: Exception Handling and Threads

PRE LAB PREPARATION

1. Can we use try instead of finally and catch blocks?
2. Name different types of exceptions in Java
3. What is the difference between exception and error in Java?
4. Can we throw an exception explicitly or manually?
5. Describe OutofMemoryError in exception handling.?
6. Is it illegal to keep an empty catch?
7. Define unreachable catch block error.
8. What is multithreading?
9. What is the purpose of wait() method in Java?
10. What is context switching?
11. should we interrupt a thread?
12. Can Java object be locked down for exclusive use by a given thread?

LAB PROGRAM

1. Create a Customer class with attributes:

- String customerId
- String name
- String email
- String phoneNumber

Include methods for:

- registerCustomer(String name, String email, String phoneNumber) - Registers a new customer and returns the customer ID.
- getCustomerDetails(String customerId) - Retrieves customer details by ID.

Handle exceptions such as InvalidCustomerException for invalid customer data (e.g., empty name or email).

2. **Online Payment System**

You are developing an online payment system where users can make payments for various services. The system should handle different types of exceptions to ensure smooth user experience and robust error management.

I. **User Class**:
   a. Attributes:
      i. String userId

ii. String name
iii. double accountBalance

b. Methods:
i. makePayment(double amount, String service) - Deducts the payment amount from the user's account balance and processes the payment for the specified service.

II. **Service Class**:
a. Attributes:
i. String serviceId
ii. String serviceName
iii. double serviceCost
b. Methods:
i. getServiceDetails() - Returns the service details.

III. **Custom Exceptions**:
a. Define custom exceptions:
i. InsufficientBalanceException (when a user tries to make a payment but has insufficient balance)
ii. InvalidServiceException (when a user tries to make a payment for an invalid or non-existent service)

IV. **PaymentService Class**:
a. Methods:
i. processPayment(User user, double amount, String serviceId) - Handles the payment process. Throws InsufficientBalanceException if the user does not have enough balance, and InvalidServiceException if the service does not exist.

V. **Main Class**:
a. Demonstrate the exception handling by simulating different payment scenarios, including successful payments, insufficient balance, and invalid service.

3. Write a JAVA program that creates threads by extending Thread class.First thread display "Good Morning "every 1 sec, the second thread displays "Hello "every 2 seconds and the third display "Welcome" every 3 seconds,(Repeat the same by implementing Runnable)

4. Multithreaded String Reversal

Create a program that uses multiple threads to reverse a list of strings. Each thread should handle the reversal of a single string from the list. Once all strings are reversed, print the reversed strings.

1. String List:
   - Use an array or list of strings that need to be reversed.
2. Thread Creation:
   - Create a separate thread for each string in the list.
   - Each thread should independently reverse its assigned string.
3. Output:
   - Print the reversed strings once all threads have completed their tasks.

4. Thread Management:
    - Ensure the main thread waits for all other threads to finish before printing the final output.

Example

For an input list of strings: ["hello", "world", "java", "multithreading"]

- Expected output: ["olleh", "dlrow", "avaj", "gnidaerhtitlum"]

Lab 10: Collections

PRE LAB PREPARATION

1. What is collections framework ?
2. Difference between collection, Collection and Collections in java?
3. List the interfaces which extends collection interface ?
4. Difference between Array and ArrayList ?
5. Difference between arraylist and vector ?
6. In which order the Iterator iterates over collection? .
7. When do we use HashSet over TreeSet?
8. Difference between HashMap and Hashtable?
9. What are the main interfaces of the Java Collections Framework?
10. List implementations of List Interface ?
11. Implementations of Set interface ?
12. Explain about Map interface in java?
13. Explain HashSet and its features ?
14. Explain the Comparable and Comparator interfaces.

LAB PROGRAMS

1. Find the Longest Subsequence

Problem Statement:

Given an ArrayList of integers, find the longest subsequence where elements are strictly increasing.

Requirements:

1. Use an ArrayList to store the values.
2. Implement a solution to find the longest increasing subsequence.

Example

For an ArrayList [10, 22, 9, 33, 21, 50, 41, 60, 80]:

- The longest increasing subsequence is [10, 22, 33, 50, 60, 80].

2. **Find Common Elements in Multiple Sets**

Problem Statement:

Given a list of sets of integers, find the common elements present in all the sets.

Requirements:

1. Use a List of Set to store the sets.
2. Find elements common to all the sets.

Example

For sets [{1, 2, 3}, {2, 3, 4}, {3, 5, 6}]:

- The common element is {3}.

3. **Find All Pairs with Given Sum**

Problem Statement:

Given a set of integers and a target sum, find all pairs of elements in the set that sum up to the target value.

Requirements:

1. Use a Set to store the integers.
2. Find all unique pairs that sum to the target value.

Example

For a set {1, 2, 3, 4, 5, 6} and target sum 7:

- The pairs are (1, 6), (2, 5), (3, 4).

4. Remove Duplicate Words from a Sentence

Problem Statement:

Given a sentence (string of words separated by spaces), remove all duplicate words and return the resulting sentence with the words in the order they first appeared.

Requirements:

1. Use a HashSet to keep track of words that have already been seen.
2. Use a List to maintain the order of words as they appear.

Example

For the sentence "the quick brown fox jumps over the lazy dog":

- The result should be "the quick brown fox jumps over lazy dog".

5. Top K Frequent Elements

Problem Statement:

Given a list of integers, find the top K most frequent elements in the list. If there are ties, the elements should be sorted in ascending order.

Requirements:

1. Use a HashMap to count the frequency of each element.
2. Use a PriorityQueue to maintain the top K frequent elements.

Example

For the list [1, 1, 1, 2, 2, 3] and K = 2:

- The result should be [1, 2] (since 1 is the most frequent, followed by 2).

6. Find and Sort Unique Words by Length

Problem Statement:

Given a list of sentences, extract all unique words, and sort them by their length in ascending order. If two words have the same length, sort them alphabetically.

Requirements:

1. **Unique Words:**
    - Use a HashSet to store unique words.
2. **Sorting:**
    - Use a PriorityQueue to sort the words by length and then alphabetically if lengths are equal.

Example

For an input list ["hello world", "world of programming", "hello java"]:

- The unique words sorted by length and then alphabetically are [of, java, hello, world, programming].

7. Group Anagrams

Description:

You are given an array of strings. Your task is to group the anagrams together. Anagrams are words or phrases that contain the same characters, but in a different order. The output should be a list of lists, where each list contains all the anagrams from the input array.

Requirements:

1. **Group Anagrams:**
    - Anagrams should be grouped together. For example, the words "eat", "tea", and "ate" are anagrams and should be grouped in the same sublist.
2. **Output Format:**
    - Return a list of lists where each sublist contains words that are anagrams of each other.
3. **Constraints:**
    - Assume the input list contains non-empty strings.
    - The output should preserve the relative order of the input strings as much as possible.

Input:

["eat", "tea", "tan", "ate", "nat", "bat"]

Output:

[

["eat", "tea", "ate"], ["tan", "nat"], ["bat"]

]

Lab 11: I/O Streams

PRE LAB PREPARATION

1. What are the differences between byte streams and character streams in Java, and when should each be used?
2. How does the **FileInputStream** and **FileOutputStream** work to handle raw file data, and what are their limitations?
3. Why and how does buffering improve I/O performance in file operations using **BufferedInputStream** and **BufferedOutputStream**?
4. What is **SequenceInputStream**, and how can it be used to read multiple files as one continuous stream?
5. How does a **ByteArrayOutputStream** allow in-memory data manipulation, and what are its benefits compared to traditional file streams?
6. What methods do **DataInputStream** and **DataOutputStream** provide for reading and writing primitive data types in binary format?
7. How do you safely write to and read from a text file using **FileWriter** and **FileReader**, including handling exceptions and character encoding?
8. What does Java's serialization mechanism do, and what must a class implement to allow its instances to be serialized and deserialized?
9. What are the security and compatibility considerations when using object serialization in distributed or long-term storage systems?
10. How can exception handling (using try-catch-finally or try-with-resources) be applied effectively during file and stream operations?

LAB PROGRAMS

Case Study 1: File Input and Output Streams for Log Archiving

Scenario: A small business needs to generate and store daily log reports from a billing system. The data is initially generated in real-time and needs to be preserved for audits.

Objective: Implement a Java program that reads data from a source file using `FileInputStream` and writes the formatted content to an archive using `FileOutputStream`.

Case Study 2: Buffered Streams for High-Performance File Transfers

Scenario: An e-learning app regularly transfers large text and binary content (like notes and PDFs). Basic file streams are too slow for processing gigabyte-sized material.

Objective: Create a Java program using `BufferedInputStream` and `BufferedOutputStream` to improve performance by minimizing disk I/O overhead.

Case Study 3: Sequence Input Stream for Merging Multiple Files

Scenario: A publishing team receives book chapters as separate files and wants to merge them into a single document seamlessly.

Objective: Use `SequenceInputStream` to read multiple input files and merge them into a single stream.

Case Study 4: ByteArray Input and Output Streams for In-Memory Processing

Scenario: A mobile app modifies image byte arrays (e.g., adding a watermark) before saving or uploading them, without using intermediate disk storage.

Objective: Build a Java application that manipulates byte data in memory using `ByteArrayInputStream` and `ByteArrayOutputStream`.

Case Study 5: DataInputStream and DataOutputStream for Binary Record Storage

Scenario: An HR system stores employee records as binary data—ID, salary, and performance score. Reading/writing accurate types (int, float, double) is critical.

Objective: Design a program using `DataInputStream` and `DataOutputStream` to write and read structured data types to a binary file.

Case Study 6: Serialization and Deserialization of Java Objects

Scenario: An e-commerce application must persist shopping cart data (complex objects with multiple fields) across sessions.

Objective: Create a Java program that serializes object data (like Cart or Product classes) and deserializes them during reload using `ObjectOutputStream` and `ObjectInputStream`.