Programme Name and Semester: MCA 2nd Semester
Object Oriented Design using Java (MCA27108)
Class: MCA
Academic Session: 2025-2026

Laboratory Manual

Object Oriented Design using Java Lab (MCA27108)

Table of Contents

Experiment/Lab Exercise/Activity No.	Name of the Experiment/Activity/Exercise	Page Number(s)
1	Design a Java program to manage basic student information using standard input and output operations.	
2	Create a Java program to calculate employee salary using modular methods.	
3	Develop a menu-driven calculator application using Java control structures.	
4	Create a Java program that analyzes a number entered by the user.	
5	Develop a Java program that stores the details of a student in suitable variables or data structures and then displays the stored information back to the user in a well-formatted manner. The program should accept details such as the student's name, roll number, course, and marks using standard input, store them temporarily in memory, and then retrieve and print the information to demonstrate input processing and structured output representation without using any file operations.	
6	Design and implement a Java program to generate a numeric series and perform basic analysis on it.	
7	Design a Java program that displays a simple personal profile on the console. The program should store basic information such as name, age, course, and institution in appropriate variables and display them using formatted output statements.	
8	Create a Java application that converts temperature from Celsius to Fahrenheit. The program should accept the temperature value as input, apply the appropriate conversion formula, and display the converted result.	
9	Develop a Java program that swaps two numeric values entered by the user. The program should demonstrate the swapping process using a	

	just two variable and display the values before and after swapping.	
10	Design a Java program that accepts three integer values from the user and determines the largest among them using conditional statements. The program should clearly display the input values and the computed result, thereby strengthening understanding of relational operators and decision-making logic.	
11	Create a Java program that generates a simple numeric pattern using iterative statements. The program should produce a structured output such as a right-angled number triangle based on a user-specified limit.	
12	Design a Java application that calculates the sum of the first N natural numbers using an iterative approach. The program should accept the value of N from the user and compute the sum using a loop construct, reinforcing the concept of accumulation and repetition in programming.	
13	Design a Java program to store and display student academic information using class and object concepts.	
14	Develop a Java program to calculate areas of different geometric shapes using method overloading.	
15	Design a Java program to calculate salary for different types of employees using inheritance.	
16	Create a Java program to demonstrate method overriding through a banking application.	
17	Design a Java program to compute area of different shapes using abstract classes.	
18	Create a Java program to demonstrate dynamic method dispatch and the use of final keyword.	
19	Write a Java program to demonstrate accessing class members using objects.	
20	Develop a Java program to initialize object data using a constructor and process it using methods.	
21	Create a Java program to demonstrate method overloading using different data types.	
22	Develop a Java program to demonstrate multilevel inheritance.	

23	Write a Java program to demonstrate runtime polymorphism using base class references.	
24	Design a Java program to demonstrate partial abstraction using abstract classes.	
25	Design and implement a Java application that analyzes the academic performance of a group of students. The program should store marks of multiple students using appropriate array structures and process the data to compute subject-wise averages, identify the highest scorer in the class, and determine the number of students who have passed or failed based on predefined criteria. The system should be modular, using separate methods for input, computation, and result display, and must handle numeric data using suitable wrapper classes where required.	
26	Develop a reusable Java application that functions as a matrix utility toolkit. The program should allow users to define and store matrices using two-dimensional arrays and perform operations such as matrix addition, transpose of a matrix, and calculation of diagonal elements' sum. Each matrix operation should be implemented as a separate method to promote modularity and reusability, and the program should clearly display the resulting matrices after each operation.	
27	Design a Java program that evaluates the strength of a user-entered password. The application should analyze the password string to verify compliance with basic security rules such as minimum length, presence of uppercase and lowercase letters, and inclusion of numeric characters. Based on these validations, the program should categorize the password strength and display appropriate feedback. String manipulation techniques and standard Java API methods must be used for character analysis.	
28	Create a Java application that performs statistical analysis on a block of text entered by the user. The program should process the input string to determine the total number of words, vowels, digits, and special characters present in the text. The design should emphasize efficient string traversal and proper use of Java's built-in string handling capabilities to generate accurate textual statistics.	
29	Develop a modular calculator system in Java using interface-based design principles. The application should define an interface that declares basic arithmetic operations and implement this interface through multiple concrete classes, each responsible for performing specific calculations. The program should provide a menu-driven interaction model that allows users to select operations dynamically,	

	demonstrating polymorphism through interface references.	
30	Design a Java application that simulates a simplified payment gateway system. The program should define multiple interfaces to represent payment processing and refund operations, and implement these interfaces through different payment modes such as card-based, digital wallet, or UPI-style transactions. Interface constants should be used where appropriate, and the application should demonstrate how multiple interfaces can be implemented within a single class to achieve flexible and extensible design.	
31	Create a user-defined Java package that groups together academic utility classes. The package should contain classes responsible for managing student information and calculating grades based on marks. A separate Java program should import this package and access its classes to perform academic computations, thereby demonstrating package creation, access, and code modularization across multiple source files.	
32	Develop a Java application that utilizes standard Java API packages to manage date and time operations. The program should display the current date and time, identify the day of the week, and calculate the number of days between two user-specified dates. The design should emphasize effective usage of built-in API classes rather than manual calculations, reinforcing the importance of library-based development.	
33	Design a Java program that functions as a data conversion utility using wrapper classes. The application should convert numeric and textual data between different formats, such as transforming strings into integers, integers into floating-point values, and numeric values back into string representations. The program should clearly demonstrate parsing techniques and object-to-primitive conversions using wrapper class methods.	
34	Create a Java application that performs statistical calculations using wrapper objects instead of primitive data types. The program should store numeric values as wrapper class objects, relying on Java's autoboxing and unboxing features to compute minimum, maximum, and average values. The design should highlight how automatic conversions simplify interaction between object-oriented structures and primitive operations.	
35	Develop a Java application that manages a dynamic student registry using the Vector class. The program should support runtime addition and removal of student records and provide search functionality based on roll numbers or identifiers. The design should demonstrate dynamic	

	data storage, wrapper class usage for numeric values, and menu-driven interaction for registry management.	
36	Design and implement an integrated Java application named University Resource Manager that consolidates multiple concepts learned so far. The system should manage students and courses using arrays or dynamic collections, provide interface-based service modules for operations, enable string-based searching and filtering of records, and organize code into appropriate packages. The project should demonstrate a well-structured, modular design suitable for extension in subsequent modules involving multithreading and exception handling.	
37	Create a system using Java where: • One thread generates numbers • Second thread checks prime numbers • Third thread computes factorial • Handle invalid input using throw	
38	Simulate file processing using Java threads where: • Thread 1: Read data (simulated) • Thread 2: Process data • Thread 3: Display results • Handle missing data scenario using Exception Handling	
39	Simulate an online ticket booking system using Java where Multiple users (threads) try to book tickets, but there are limited number of tickets. Also define a user-defined NoTicketAvailableException.	
40	Design a banking system where multiple transactions occur simultaneously. • Deposit and withdraw via threads • Balance shared across threads • Create InsufficientBalanceException and InvalidAmountException Classes to handle exceptional conditions	
41	Process student results using a Java Program implementing parallel threads: • Thread 1: Input marks • Thread 2: Calculate grade • Thread 3: Generate result summary • Create an InvalidMarksException exception to handle exceptional condition	
42	Design a producer–consumer system using Java: • Producer generates data	

	• Consumer processes data • Controlled execution count • Throw exception if buffer is empty/full	
44	Simulate a login system with concurrent login attempts in Java using Multiple login threads and Credential validation. Throw an InvalidLoginException if wrong credentials are provided.	
45	Design an online exam simulation system using Java. Create Thread for timer, Thread for answering questions, and the system should Auto-submit on timeout. Use an ExamTimeoutException for handling timeout exception.	
46	Create an inventory system where multiple threads update stock.	
47	Validate different types of data using separate Java threads. Create Thread for numeric validation, Thread for string validation, and Thread for range validation. Throw exceptions for invalid data.	
48	Design a Smart Service System using Java integrating: • Multithreading • User-defined exceptions • Modular design	
49	Design and implement a Java application that manages student records using an ArrayList. The program should allow dynamic insertion, deletion, and display of student objects at runtime, demonstrating ordered data storage and index-based access. Each student record should contain basic details such as roll number, name, and marks, and the application should provide a menu-driven interface to manage records efficiently.	
50	Develop a Java application that simulates an employee directory using the Vector class. The program should support concurrent-safe dynamic storage of employee records and allow operations such as adding new employees, removing existing ones, and displaying the complete directory. The design should highlight the difference between legacy collections and modern collection classes while maintaining a structured object-oriented approach.	
51	Create a Java application that manages course enrollment using the List interface. The system should store enrolled students in a list-based structure and allow operations such as adding students at specific positions, updating enrollment details, and iterating through the list to display enrollment order. The program should demonstrate polymorphic behavior by accessing an ArrayList object through a List	

	reference.	
52	Design a Java application that tracks student attendance using a Set implementation. The program should ensure that duplicate attendance entries are automatically prevented and should display the final attendance list after multiple insertion attempts. The design should clearly illustrate how sets differ from lists in handling uniqueness and ordering.	
53	Develop a Java application to manage a library book registry using HashSet. Each book should be represented as an object with attributes such as ISBN, title, and author. The system should store and manage books in such a way that duplicate book entries are not allowed, emphasizing object equality and hashing concepts.	
54	Design a Java application that maps student roll numbers to their marks using HashMap. The program should support insertion, update, search, and deletion of student records using keys, and should demonstrate efficient key-based data retrieval. The application should also display all entries using map traversal techniques.	
55	Create a Java application that analyzes a paragraph of text and calculates the frequency of each word using a Map implementation. The program should process the input string, tokenize it appropriately, and store word-frequency pairs efficiently. The output should display each word along with its occurrence count, demonstrating practical usage of maps in text analytics.	
56	Develop a Java application that manages product inventory using a combination of collection classes. The program should store product details dynamically, categorize products using appropriate data structures, and provide search and update functionalities. The design should integrate List, Set, and Map where appropriate to demonstrate real-world collection selection strategies.	
57	Design a Java program that connects to a relational database using JDBC and performs basic operations on a student table. The application should establish a database connection, insert new student records, retrieve existing records, and display them in a structured format. Proper separation of database logic and application logic should be maintained.	
58	Create a Java application that performs complete Create, Read, Update, and Delete operations on a database table using JDBC. The program should accept user input dynamically, execute corresponding SQL	

	queries, and display operation results clearly. The design should focus on reliable database interaction and resource management.	
59	Develop a Java application that synchronizes in-memory collection data with a database. The program should store records initially in Java collection objects, allow modifications at runtime, and then persist the updated data into a database table using JDBC. This application should demonstrate how collections act as temporary data holders before permanent storage.	
60	Design and implement a comprehensive Student Information Management System that integrates Java collections with database connectivity. The system should manage student details using appropriate collection classes for in-memory processing and use JDBC to store and retrieve records from a relational database. The application should provide a menu-driven interface, ensure data consistency, and demonstrate modular, extensible design suitable for real-world academic or enterprise systems.	