ADVANCED OBJECT-ORIENTED PROGRAMMING – LABORATORY WORKBOOK

23CS2103R

ACADEMIC YEAR: 2024-25

STUDENT ID:

STUDENT NAME:

Table of Contents

A.Y. 2024-25 LAB CONTINUOUS EVALUATION

S. No	Date	Experiment	Pre Lab (10M)	In-Lab (25M)			Post Lab (10 M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
		Introduction Session	-NA-							
1		**Creational Design Patterns**								
2		**Structural Design Patterns**								
3		**Behavioral Design Patterns I**								
4		**Behavioral Design Patterns II**								
5		**SOLID Design Principles and Test-Driven Development**								
6		**Generics with Classes and Interfaces**								
7		**Comparator and Comparable**								
8		**Sets and Maps**								

S. No	Date	Experiment	Pre Lab (10M)	In-Lab (25M)			Post Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
9		Nested Classes, Functional interfaces, Lambda Expressions and Stream API								
10		Threading in java								
11		Thread Synchronization and Coordination								
12		Java Database Connectivity								
13		JSP – Servlet and JDBC								

Experiment#		Student ID	
Date		Student Name	

1. Creational Design Patterns

Aim/Objective: To analyse the implementation of Singleton, Factory and Abstract Design Patterns for the real time scenarios.

Description: The student will understand the concept of Creational Design Patterns (Singleton, Factory and Abstract Factory)

Pre-Requisites: Classes and Objects in Java

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Draw the UML Relationship Diagram for Factory Design Pattern for any customized scenario.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Develop a game application with multiple levels and varying difficulty settings. Implement the following design patterns to manage different aspects of the game:

Singleton Pattern: Use this pattern to manage the game state, ensuring that there is only one instance of the game state throughout the application.

Factory Method Pattern: Apply this pattern to create different types of enemies for each level.

Abstract Factory Pattern: Utilize this pattern to create various types of weapons and power-ups based on the level and difficulty settings.

Ensure that the design of your game is flexible and can easily accommodate new levels, enemies, weapons, and power-ups in the future.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

VIVA-VOCE Questions (In-Lab):

1) Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?

2) Discuss the difference between Factory and Abstract Factory design patterns.

3) Mention the pros and cons of Factory Design pattern

4) Mention the cons of Singleton Design pattern.

Post-Lab:

1) Design and implement a ride-sharing application that allows users to request rides from various types of vehicles (cars, bikes, scooters). Utilize the Factory Method pattern to create vehicle instances, the Abstract Factory pattern to implement different payment methods (credit card, PayPal, cash), and the Singleton pattern to manage user authentication securely. Provide a detailed example demonstrating the interaction of these patterns within the application.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

2. Structural Design Patterns

Aim/Objective: To analyse the implementation of Adapter and Decorator Design Patterns for the real-time scenario.

Description: The student will understand the concept of Adapter and Decorator Design Patterns.

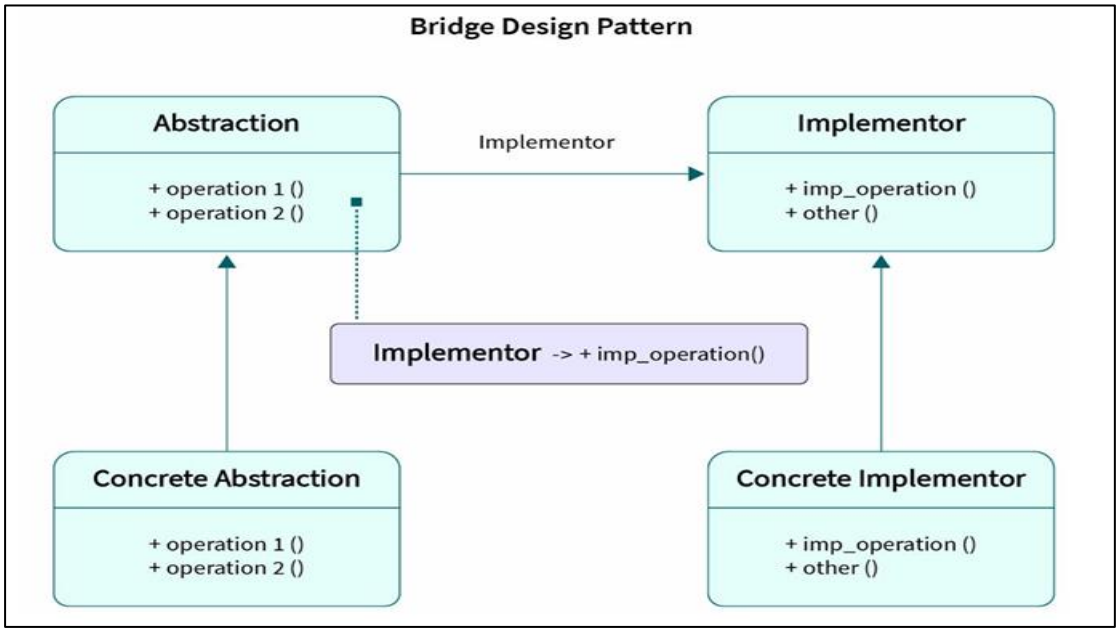
Pre-Requisites: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Draw the UML Relationship Diagram for Adapter Design Pattern for Mobile charger adapter scenario. Note: Mobile battery needs 3 volts to charge but the normal socket produces either 120V (US) or 240V (India). So the mobile charger works as an adapter between mobile charging socket and the wall socket.

2. Discuss about the below mentioned structure of Bridge Design Pattern.



Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Develop a music streaming application that can play music from various sources, such as local files, online streaming services, and radio stations. Implement the following design patterns to achieve this functionality:

Adapter Pattern: Adapt different music sources to a common interface.

Bridge Pattern: Decouple the music playback functionality from the music source.

Decorator Pattern: Add additional features (e.g., equalizer, volume control) to the music playback.

Your task is to design and implement the application using these design patterns to ensure flexibility, scalability, and maintainability.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab)

1) State at which situation that we are in need of Bridge Design Pattern.

2) Illustrate the difference between Decorator and Bridge pattern.

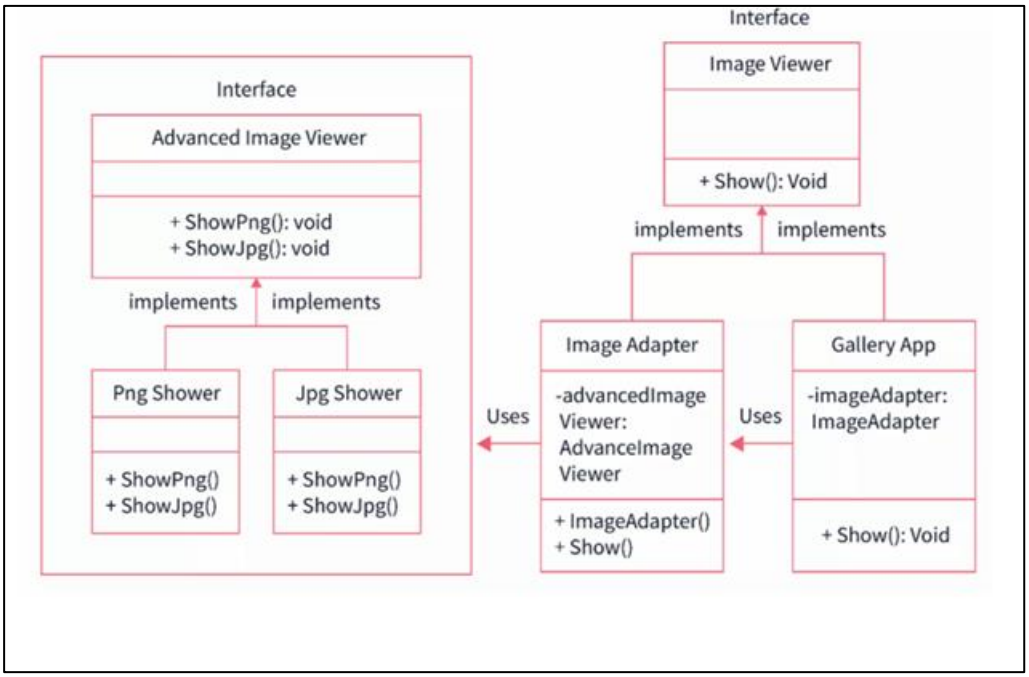
3) Discuss the Pros and Cons of Facade Design Pattern.

4) Discuss the Pros and Cons of Bridge Design Pattern.

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Implement the below depicted UML Diagram in Java Program with respective to Adapter Pattern



Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

3. Behavioural Design Patterns I

Aim/Objective: To analyse the implementation of Chain of Responsibility Design Pattern & Iterator Design Pattern and Command Design Pattern for the real-time scenario.

Description: To make student understand the application of behavioural design pattern in software applications.

Pre-Requisites: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Draw the UML Relationship Diagram for Iterator Design Pattern for customized Scenarios.

In-Lab:

1) You are required to design and implement a logging system in Java that processes log messages of different severity levels: INFO, DEBUG, and ERROR. The system should utilize the Chain of Responsibility, Command, and Iterator design patterns. Below are the detailed requirements:
   A. **Severity Levels**:
      a. INFO: General information about system operations.
      b. DEBUG: Detailed information typically used for diagnosing problems.
      c. ERROR: Error conditions indicating problems that need to be addressed.
   B. **Handlers**:
      a. Each handler is responsible for processing messages of a specific severity level.
      b. Handlers should be linked in a chain such that if a handler cannot process a message, it passes the message to the next handler in the chain.
   C. **Command Pattern**:
      a. Use the Command pattern to encapsulate the logging requests.
      b. Define a Command interface with an execute(String message) method.
      c. Implement a LogCommand class that executes logging requests using handlers.
   D. **Iterator Pattern**:
      a. Use the Iterator pattern to manage a list of commands.
      b. Create a Logger class that maintains a list of Command objects and processes them sequentially.
   E. **Implementation Steps**:
      a. Define an enum LogLevel to represent the severity levels.
      b. Implement the Command interface and LogCommand class.
      c. Create an abstract LogHandler class and concrete handler classes (InfoHandler, DebugHandler, ErrorHandler) for each severity level.
      d. Implement the Logger class that uses an iterator to process commands.
      e. Provide a client class to configure the chain of responsibility, create commands, and process log messages.

   Implement the following classes and interfaces to achieve the above requirements:

      a. LogLevel (enum): Represents the severity levels.
      b. Command (interface): Declares the execute(String message) method.
      c. LogCommand (class): Implements the Command interface.
      d. LogHandler (abstract class): The base class for log handlers.
      e. InfoHandler, DebugHandler, ErrorHandler (classes): Concrete handlers for each severity level.
      f. Logger (class): Uses an iterator to process a list of commands.
      g. Client (class): Configures and demonstrates the logging system.

Experiment#		Student ID	
Date		Student Name	

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) State at which situation that we need Chain of Responsibility Design Pattern.

2) Discuss the Pros and Cons of Iterator Design Pattern.

3) Discuss the Pros and Cons of Command Design Pattern

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

4. Behavioural Design Patterns II

Aim/Objective: To analyse the implementation of Observer Design Pattern & Template Design Pattern and Dependency Injection Design pattern for the real-time scenario.

Description: To make student understand the application of behavioural design pattern in software applications.

Pre-Requisites: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Draw the UML Relationship Diagram for Template Design Pattern for customized Scenarios.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Design an online auction system that utilizes the Observer and Template design patterns to manage auction events and bidding processes efficiently.

Requirements:

A. Observer Design Pattern:

- Use the Observer pattern to notify bidders about auction events such as item availability, bidding start, and bidding end.
- Bidders should be able to subscribe and unsubscribe to receive notifications.

B. **Template Design Pattern**:

- Implement the Template pattern to define the structure and steps of the auction process.
- Customize specific steps for different types of auctions (e.g., standard auction, reserve auction).

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) In a software development project, you're tasked with implementing a data import module that processes different types of data files (CSV, XML, JSON) and performs specific operations based on the file type. The Template Design Pattern is suitable for this scenario to provide a structured way to define the steps of data processing while allowing customization for each file type.

Requirements

A. **Define an Abstract Data Importer (Template)**:
   - Create an abstract class DataImporter that defines the template method importData() and other common methods shared by all data importers.
   - The importData() method should outline the sequence of steps required for data import, such as reading data, parsing it, and saving it.

B. **Implement Concrete Importers for Each File Type**:
   - Implement concrete subclasses (CSVImporter, XMLImporter, JSONImporter) that extend DataImporter.
   - Each subclass should override specific methods as needed to handle file-specific operations like parsing and validation.

C. **Client Code to Use the Template**:
   - Develop client code (e.g., a main method or another service) that uses the template method pattern to invoke data import operations.
   - Demonstrate how different file types are imported using their respective concrete importers.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) State at which situation that we need Observer Design Pattern.

2) Discuss the Pros and Cons of Template Design Pattern.

3) Discuss the Pros and Cons of Observer Design Pattern.

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Imagine you are developing a weather monitoring system that notifies various displays when the weather conditions change. The system involves multiple displays that need to stay updated with the latest weather data in real-time. Implement the Observer design pattern to achieve this functionality.

Requirements:

A. Subject Interface: WeatherStationSubject
   - **a.** Define an interface WeatherStationSubject that declares methods to register, remove, and notify observers.

B. Observer Interface: WeatherObserver
   - **a.** Define an interface WeatherObserver that declares a method for updating when notified by the subject.

C. Concrete Subject: WeatherStation
   - **a.** Implement the WeatherStationSubject interface in a class called WeatherStation. This class will maintain a list of observers and notify them when weather data changes.

D. Concrete Observers: Display Devices
   - **a.** Implement WeatherObserver interface in various display devices such as CurrentConditionsDisplay, StatisticsDisplay, and ForecastDisplay. These displays will update their information whenever the weather data changes.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) Design a scenario that incorporates the Template Method, Dependency Injection, and Observer patterns in Java. The scenario will be a notification system where different types of notifications (Email, SMS, and Push) are sent based on user events (e.g., user registration, password reset).

A. **Template Method Pattern**: Use this pattern to define the steps of sending a notification, allowing subclasses to implement specific steps for different notification types.
B. **Dependency Injection**: Use this pattern to inject the specific notification service into a notifier class.
C. **Observer Pattern**: Use this pattern to observe user events and trigger the appropriate notifications.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

5. SOLID Design Principles and Test-Driven Development.

Aim/Objective: To analyse the implementation of SOLID Principles & Test-Driven Development (TDD) for the real-time scenario.

Description: The student will understand the concept of SOLID Principles & Test-Driven Development (TDD).

Pre-Requisites: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Elaborate about each letter in SOLID Principles.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Develop a Student Information System (SIS) that adheres to SOLID design principles, where you need to manage the Student Information System (SIS) with the incorporation of the concepts of Classes, Objects, Constructors, Interfaces and inheritance.

Requirements

1. Student Management:
    - Manage student details such as name, ID, and courses enrolled.
2. Course Management:
    - Manage course details such as course ID, course name, and list of enrolled students.
3. Enrollment Management:
    - Handle the enrollment of students in courses.

Applying SOLID Principles

1. Single Responsibility Principle (SRP):
    - Each class should have one responsibility. For example, a Student class handles student details, and a Course class handles course details.
2. Open/Closed Principle (OCP):
    - The system should be open for extension but closed for modification. Use interfaces and abstract classes to allow for new types of students or courses without modifying existing code.
3. Liskov Substitution Principle (LSP):
    - Subtypes must be substitutable for their base types. Ensure derived classes can be used interchangeably with their base classes.
4. Interface Segregation Principle (ISP):
    - Create specific interfaces for different functionalities, ensuring clients only depend on the interfaces they use.
5. Dependency Inversion Principle (DIP):
    - High-level modules should depend on abstractions, not concrete implementations. Use dependency injection to manage dependencies.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) During an NCC parade, a large number of cadets participated, and the leader instructed them to stand in a line sorted alphabetically by their names for easier identification. After organizing the cadets in alphabetical order, the leader wants to verify whether they are indeed standing in the correct sorted order.

   You as the leader, construct a JUnit test program to validate the cadet's arrangement and ensure it aligns with the expected alphabetical sorting order. The test program should include various scenarios such as an empty list of cadets, a single cadet, multiple cadets with different names, and cadets with identical names and also assertions within the unit test to verify the correctness of the cadet's alphabetical arrangement.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) State about "Write Unit Tests" principle in Clean Coding Techniques.

2) Discuss about Single Responsibility Principle (SRP).

3) Illustrate the difference between Assertions and Annotations.

4) List the Assertions in JAVA Test Driven Development (TDD).

5) What is the problem of not having Interface segregation principle in SOLID principles>

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Write a Test-Driven Development program to accept the password when the length of it should be between 5 to 10 characters ("Password validator")

Input: Abc123

Output: Valid

password: accepted

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

6. Generics with Classes and Interfaces.

Aim/Objective: To analyse the implementation of the concept of Generics with Interfaces for the real time scenario.

Description: The student will understand the concept of Generics with Interfaces.

Pre-Requisites: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Discuss the necessity of Generics Interfaces.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Write a Java Program to identify the Maximum Value and Minimum Value in the arrays of different datatypes like Integer, String, Character & float by incorporating the concept of Generics with interfaces.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) List the benefits of Generics.

2) Discuss about the various types of Generics implementation in Java.

3) Illustrate about "Type Parameter Naming Conventions" in Generics

4) State the significance of diamond (<>) operator in generics.

5) List the limitations of generics.

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Create a generic method that sorts an array of objects using a bubble sort algorithm. Test the method with different types of objects such as integers, doubles, and strings.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

7. Comparator and Comparable

Aim/Objective: Analyse the practical application of the Comparator and Comparable interfaces in real-world scenarios, discussing their roles, advantages, and differences.

Description: Student will be able to understand and apply the concept of Comparator and Comparable Interfaces.

Pre-Requisites: A Strong knowledge on Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Discuss the differences between Comparator and Comparable by filling the below mentioned table.

S.no	Comparable	Comparator
1.		
2.		
3.		
4.		

Experiment#		Student ID	
Date		Student Name	

2) Write a Java program that sorts a Linked List using the Comparable interface.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Create a Java program that sorts a list of Movie objects by their year of release. Define the Movie class with attributes such as rating, name, and year. Implement the Comparable interface in the Movie class and override the compareTo() method to sort the movies based on their release year.

   Program:

Experiment#		Student ID	
Date		Student Name	

2) You are tasked with developing a system to manage employee records for a large corporation. The Employee class has attributes such as id, name, department, and salary. Different departments and teams need to sort employee records based on different criteria, such as salary, name, and department.

Implement a Java program that sorts a list of Employee objects using the Comparator interface. The program should allow sorting by multiple criteria: by salary (ascending and descending), by name (alphabetical order), and by department (alphabetical order).

Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) List the usage of comparable Interface.

2) List the usage of comparator interface.

3) What is the purpose of the compareTo method?

4) What happens if you do not override the compareTo method when implementing Comparable?

5) What is the difference between Comparable and Comparator?

Post-Lab:

1) Develop a Java program to compare movies by their ratings using a custom Comparator implementation. Your program should follow these steps: a. Implement a class that serves as a Comparator for Movie objects, providing the comparison logic based on movie ratings. b. Instantiate the Comparator class. c. Utilize the overloaded sort () method, passing both the list of movies and the instance of the Comparator class to perform the sorting.

   Sample Input:

   8.4 Return of the Jedi 1983

   8.8 Empire Strikes Back 1980

   8.3 Force Awakens 2015

   8.7 Star Wars 1977

   Sample Output:

   Sorted by rating

   8.3 Force Awakens 2015

   8.4 Return of the Jedi 1983

   8.7 Star Wars 1977

   8.8 Empire Strikes Back 1980

   Sorted by name

   Empire Strikes Back 8.8 1980

   Force Awakens 8.3 2015

   Return of the Jedi 8.4 1983

   Star Wars 8.7 1977

   Sorted by year

   1977 8.7 Star Wars

   1980 8.8 Empire Strikes Back

   1983 8.4 Return of the Jedi

   2015 Force Awakens

   Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

8. Implementation of Sets and Maps

Aim/Objective: To understand the concept and implementation of concepts of sets and maps.

Description: The student will understand the concepts of sets and maps.

Pre-Requisites: Classes and Objects, HashMap, TreeMap, HashSet and TreeSet in JAVA.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Explain the concept of a set in Java. What is the main characteristic of a set? Provide an example code snippet demonstrating the usage of a set.

Experiment#		Student ID	
Date		Student Name	

2) Draw a UML diagram showing the relationship between the Set interface, the Collection interface, and two implementations of Set: HashSet and TreeSet.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) You are tasked with developing a contact management application where you need to store a collection of contacts. Each contact has a name and a phone number. Implement a class called ContactManager that uses a map to store the contacts, with the contact's name as the key and the phone number as the value. Write code to demonstrate the implementation of the ContactManager class.

   Requirements

   1. ContactManager Class:
      - Use a map to store contacts.
      - The key of the map is the contact's name (String).
      - The value of the map is the phone number (String).
      - Provide methods to add, remove, and retrieve contacts.
      - Provide a method to list all contacts.
   2. Demonstration:
      - Create an instance of ContactManager.
      - Add several contacts.
      - Retrieve and display a contact's phone number.
      - Remove a contact.
      - List all contacts.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) You are working on a program that stores student grades. Each student has a unique ID assigned to them. Design a data structure using sets and maps to efficiently store and retrieve student grades based on their ID. Write the code for adding a student's grade to the data structure given their ID and grade.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) Explain the difference between a set and a map in programming.

2) What is the difference between a Set and a HashSet?

3) What is the difference between a Map and a HashMap?

4) How does a map differ from an array or a list?

5) What is the purpose of the key-value pairs in a map? Can the same key have multiple values?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Given a set of integers, write a code snippet to find the maximum value in the set.

Procedure/Program:

2) Write a Java program that uses a Set to store a list of fruits. The program should then iterate over the Set and print out the fruits in alphabetical order.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

9. Nested Classes, Functional interfaces, Lambda Expressions and Stream API

Aim/Objective: To implement the concepts of Lambda expression and Stream API to solve real world data through Collection classes.

Description: The student will understand the concepts of Nested classes, lambdas and stream api for efficient processing of data in a Collection.

Pre-Requisites: Classes and Objects

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) List some of the predefined functional interfaces available in java.util.function package and explain their uses.

Experiment#		Student ID	
Date		Student Name	

Write a lambda expression to sort a list of strings in descending order.

2) Write a stream pipeline that filters a list of integers to only even numbers, doubles them, and then collects them into a list.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Consider a Coffee shop which has staff member count of 5. Employer at the end of the month, before giving the salaries to the employees, employer asked them to stand in a queue where employee having more experience should stand first and later followed by less experience so on. Employee has attributes like name, age and experience. You as an employer should distribute salary along with bonus.
   Bonus should be given to the employees based on experience.
   Employee1 has experience 5 years
   Employee2 has 4 years
   Employee 3 has 3 years,
   Employee 4 has 1 year and
   Employee 5 is a fresher.
   Filter the employees who have experience more than 2 years should be given bonus. Make use of Predicate interface and construct the scenario.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

2) A company wants to perform various operations on the list of employees efficiently using Java Stream API.

   The employees have attributes like name, age, department, and salary. The operations include

   1.Filtering employees by Department.

   2.Sort employees by their names.

   3.Find the employee with the highest salary.

   4.Calculate average salary of employees.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is an anonymous inner class, and when would you use one?

2) How does the @FunctionalInterface annotation help in defining a functional interface?

3) How do lambda expressions relate to functional interfaces

4) Explain the difference between collect() and reduce() in the Stream API.

5) What is a parallel stream, and how do you create one?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) You are tasked with designing an Employee Management System for a company. The system needs to handle various operations on a list of employees using the Java Stream API. Each employee has attributes such as ID, name, department, salary, and age. The operations include filtering, sorting, grouping, and aggregation.
Requirements

1. **Data Model**:
    - Create an Employee class with attributes: id, name, department, salary, and age.
2. **Operations**:
    - **Filter** employees based on department.
    - **Sort** employees by salary in descending order.
    - **Group** employees by department.
    - **Find** the highest-paid employee.
    - **Calculate** the average salary of employees in a department.
    - **List** the names of employees who earn more than a specified amount.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

10. Threading in Java

Aim/Objective: To explore the basics of concurrent programming and understand the concepts of threading in Java.

Description: The student will gain an understanding of Java threading, encompassing the fundamentals of threads, their creation, synchronization, and communication, enabling concurrent execution of multiple tasks within a program.

Pre-Requisites: Knowledge on Classes and Objects, Understanding of threads and their execution model in JAVA.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) How can you implement five functions (printTwo, printThree, printFour, printFive, and printNumber) using multiple threads to print numbers from 1 to 15, where each function prints a message if the number is divisible by 2, 3, 4, or 5, and printNumber prints the number if none of these conditions are met?
   Program:

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Design the Account Class
    - Create a class named Account in the bank package.
    - The Account class should have the following attributes:
        - int accountNumber
        - double balance
    - Provide methods for deposit(double amount) and withdraw(double amount) that update the balance accordingly.
2) Create a Thread Using the Runnable Interface:
    - Create a class named "TransactionRunnable" in the bank package that implements the "Runnable" interface.
    - This class should take an "Account" object and an "amount" as parameters, and perform either a deposit or withdrawal in its run method.
3) Create a Thread by Extending the Thread Class:
    - Create a class named "TransactionThread" in the bank package that extends the "Thread" class.
    - This class should also take an "Account" object and an "amount" as parameters, and perform either a deposit or withdrawal in its run method.
4) Running the Experiment:
    - Create a Main class in the "bank" package with a "main" method to execute the experiment.
    - Instantiate an Account object.
    - Create multiple threads using both "TransactionRunnable" and "TransactionThread" to perform concurrent deposits and withdrawals.

Expected Results:

- The Account balance should be updated correctly by each thread.
- The synchronized methods in the Account class ensure thread safety, preventing race conditions.
- The final balance should reflect all deposits and withdrawals performed by the threads.

Experiment#		Student ID	
Date		Student Name	

Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is a thread? How does it enable concurrent execution in Java?

2) Describe the different ways to create a thread in Java.

3) What is thread safety? How do you achieve thread safety in Java?

4) explain the terms "synchronization" and "thread safety" in the context of threading.

5) How can you prevent race conditions in multithreaded programs?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Write a JAVA program which will generate the threads: -
    - To display 10 terms of Fibonacci series.
    - To display 1 to 10 in reverse order.

Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

11. Thread Synchronization and Coordination of Thread.

Aim/Objective: To understand the Thread Synchronization and Producer-Consumer thread coordination in a multi-threaded Java program.

Description: The student will understand the concepts of the Producer-Consumer pattern, which is used to solve the problem of synchronizing access to a shared resource between multiple threads.

Pre-Requisites: Classes, Objects, Understanding of multi-threading and synchronization in Java

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Explain the need of Synchronization in a multithreading environment?

2) How threads establish communication mechanism among them in a multithreading environment?

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) You are tasked with designing a Bank Account Management System where multiple users can perform transactions on a shared bank account simultaneously. To ensure the consistency and correctness of the account balance, synchronization is necessary.
Requirements
   A. **BankAccount Class**:
      a. Attributes: balance (double).
      b. Methods:

         deposit(double amount): Adds the specified amount to the account balance.

         withdraw(double amount): Subtracts the specified amount from the account balance if sufficient funds are available.

         getBalance(): Returns the current balance.

   B. **Thread Safety**:
      a. Use synchronization to ensure that deposit and withdrawal operations are thread-safe.
   C. **Operations**:
      a. Multiple threads will simulate users performing deposit and withdrawal operations concurrently.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) Implement a messaging application where a Producer class generates messages and a Consumer class consumes them. The communication between the producer and consumer will be synchronized to ensure proper message exchange without data loss or race conditions.

Requirements

A. **Producer Class**:

a. Generates messages and puts them into a shared buffer.
b. Uses synchronization to ensure thread safety.

B. **Consumer Class**:
   a. Consumes messages from the shared buffer.
   b. Uses synchronization to ensure thread safety

C. **Shared Buffer**:
   a. A thread-safe queue to store messages.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is the main challenge in implementing the producer-consumer problem?

2) How does the synchronized keyword ensure that only one thread can access a shared resource at a time?

3) How does the wait() and notify() methods facilitate inter-thread communication in the Producer-Consumer pattern?

4) How can you ensure mutual exclusion between the producers and consumers while accessing the shared buffer?

5) How does the Producer-Consumer pattern ensure thread safety and synchronization?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Write a code to implement a bounded buffer using the concepts learned in the experiment. Ensure that the buffer has a maximum capacity of 10 items, and the producer and consumer threads operate correctly while avoiding race conditions.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

12. Java Database Connectivity

Aim/Objective: To understand the how a Java application can be connected with any database in the world for data persistence.

Description: The student will understand the concepts of JDBC.

Pre-Requisites: Classes, Objects and database

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Explain the limitations of file system making us to use database?

2) Explain the concept of JDBC and its role in Java database connectivity. What are the main steps involved in performing database operations using JDBC?

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) You are developing a Student Information Management System where student details such as student ID, name, address, and program can be managed through a web interface. Implement a servlet called StudentDetailsServlet that accepts student details through a form submission and stores these details in a database using JDBC. Perform the following steps to create and manage the student database:
Requirements

A. **Create the Student Database**:
   a. Write a JDBC program to create a Student database.
B. **Create the Registration Table**:
   a. Write a JDBC program to create a Registration table inside the Student database with fields id, name, address, and program, where id is the primary key.
C. **Insert Records**:
   a. Write a JDBC program to insert four records into the Registration table.
D. **Display Records**:
   a. Write a JDBC program to display the records inserted into the Registration table.
E. **Update Records**:
   a. Write a JDBC program to update the program of students whose id values are 100 and 101.
F. **Delete Records**:
   a. Write a JDBC program to delete the student record whose id is 101.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is the role of JDBC in Java database connectivity?

2) What are the different types of JDBC drivers? Explain their differences and advantages.

3) How do you establish a database connection using JDBC?

4) Explain the difference between Statement and PreparedStatement in JDBC.

5) How do you handle exceptions related to database operations in JDBC? What are some common JDBC-related exceptions?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Create a JDBC program to connect to a PostgreSQL database. Specifically, you need to create a "test" database in the PostgreSQL server and write a JDBC program that checks whether the connection to the database is successful. If the connection is successful, the program should display the message "Connected to the database". If the username or password is incorrect, the program should display "Invalid username or password".
   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment#		Student ID	
Date		Student Name	

13. Handling Form Data in JSP, Processing with Servlet, and Persisting into Database

Aim/Objective: To understand the how to fetch form data in JSP and handle the data by using Servlet and persist the data into database.

Description: The student will understand creating a web application that collects user input through a form in a JSP page, processes the submitted data using a servlet, and persists the data into a database using JDBC.

Pre-Requisites: Classes, Objects and Databases

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) How does JSP differ from servlet?

2) Explain how a Servlet handle form data?

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Develop a web application that allows users to submit their contact details through a form on a JSP page. The form data will be processed by a servlet and stored in a PostgreSQL database.

A. Set Up the Database

a. Create a PostgreSQL database and a table to store the contact details.

B. **Create a JSP Form**:

a. Develop a JSP page with a form to collect user contact details.

C. **Create a Servlet to Handle Form Submission**:

a. Develop a servlet that processes the form data submitted from the JSP page.
b. Use the servlet to validate and prepare the data for persistence.

D. **Persist Data into the Database Using JDBC**:

a. Establish a connection to the PostgreSQL database.
b. Insert the submitted form data into the database.

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

2) A Programmer, is developing a web-based calculator application for his client using servlets, JSP (Java Server Pages), and integrating with a database. The application allows users to perform basic arithmetic operations such as addition, subtraction, multiplication, division, and modulus.

Requirements and Implementation Steps:

Step 1: Design UI (VIEW) Using HTML

Step 2: Create a Calculator (MODEL) Class

Step 3: Create a Servlet (CONTROLLER) to Handle Requests

Step 4: Integrate with JSP for Dynamic Views

Step 5: Deployment and Testing

Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is the role of a servlet in handling form data submitted by a user in a JSP page?

2) What is the difference between the request and response objects in servlets?

3) Explain the process of sending form data from a JSP page to a servlet for processing.

4) What is the purpose of the web.xml deployment descriptor file in a Java web application?

5) How can you access the request parameters sent from a JSP form in a servlet?

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Write a code snippet demonstrating the use of JSP and servlet to handle a login form. Use a scenario where a user submits a login form, and the servlet validates the credentials. Your code should include the following:
   a. Create a JSP page with a login form that collects username and password.
   b. Implement a servlet that receives the form data, validates the credentials, and redirects the user to a success or error page based on the validation result.
   c. Test your program by submitting different login credentials to ensure proper validation and redirection.

   Procedure/Program:

Experiment#		Student ID	
Date		Student Name	

Experiment#		Student ID	
Date		Student Name	

- ✓ **Data and Results:**

- ✓ **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.