Department of Computer Engineering

Academic Term: First Term 2023-24

Practical No:	**7**
Title:	**Design Using Object-Oriented Approach with Emphasis on Cohesion and Coupling in Software Engineering**
Date	12-09-23
Roll No:	9770
Team members:	Glen pereira ,Nash, prince,Ayush

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Total Score
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer Engineering

Academic Term: First Term 2022-23

Class: T.E /Computer Sem – V / Software Engineering

Signature of the Teacher:

Lab Experiment 07

Experiment Name: Design Using Object-Oriented Approach with Emphasis on Cohesion and Coupling in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the Object-Oriented (OO) approach in software design, focusing on the principles of cohesion and coupling. Students will gain practical experience in designing a sample software project using OO principles to achieve high cohesion and low coupling, promoting maintainable and flexible software.

Introduction: The Object-Oriented approach is a powerful paradigm in software design, emphasizing the organization of code into objects, classes, and interactions. Cohesion and Coupling are essential design principles that guide the creation of well-structured and modular software.

Lab Experiment Overview:

1. Introduction to Object-Oriented Design: The lab session begins with an introduction to the Object-Oriented approach, explaining the concepts of classes, objects, inheritance, polymorphism, and encapsulation.
2. Defining the Sample Project: Students are provided with a sample software project that requires design and implementation. The project may involve multiple modules or functionalities.
3. Cohesion in Design: Students learn about Cohesion, the degree to which elements within a module or class belong together. They understand the different types of cohesion, such as functional, sequential, communicational, and temporal, and how to achieve high cohesion in their design.
4. Coupling in Design: Students explore Coupling, the degree of interdependence between modules or classes. They understand the types of coupling, such as content, common, control, and stamp coupling, and strive for low coupling in their design.
5. Applying OO Principles: Using the Object-Oriented approach, students design classes and identify their attributes, methods, and interactions. They ensure that classes have high cohesion and are loosely coupled.
6. Class Diagrams: Students create Class Diagrams to visually represent their design, illustrating the relationships between classes and their attributes and methods.
7. Design Review: Students conduct a design review session, where they present their Class Diagrams and receive feedback from their peers.
8. Conclusion and Reflection: Students discuss the significance of Object-Oriented Design principles, Cohesion, and Coupling in creating maintainable and flexible software. They reflect on their experience in applying these principles during the design process.

Learning Outcomes: By the end of this lab experiment, students are expected to:

- Understand the Object-Oriented approach and its core principles, such as encapsulation, inheritance, and polymorphism.
- Gain practical experience in designing software using OO principles with an emphasis on Cohesion and Coupling.

- Learn to identify and implement high cohesion and low coupling in their design, promoting modular and maintainable code.
- Develop skills in creating Class Diagrams to visualize the relationships between classes.
- Appreciate the importance of design principles in creating robust and adaptable software.

Pre-Lab Preparations: Before the lab session, students should review Object-Oriented concepts, such as classes, objects, inheritance, and polymorphism. They should also familiarize themselves with the principles of Cohesion and Coupling in software design.

Materials and Resources:

- Project brief and details for the sample software project
- Whiteboard or projector for creating Class Diagrams
- Drawing tools or software for visualizing the design

Conclusion: The lab experiment on designing software using the Object-Oriented approach with a focus on Cohesion and Coupling provides students with essential skills in creating well-structured and maintainable software. By applying OO principles and ensuring high cohesion and low coupling, students design flexible and reusable code, facilitating future changes and enhancements. The experience in creating Class Diagrams enhances their ability to visualize and communicate their design effectively. The lab experiment encourages students to adopt design best practices, promoting modular and efficient software development in their future projects. Emphasizing Cohesion and Coupling in the Object-Oriented approach empowers students to create high-quality software that meets user requirements and adapts to evolving needs with ease.

b) Apply Object-Oriented principles, such as encapsulation and inheritance, to design a class hierarchy for a specific problem domain.
In the "Vehicle" domain, we establish a class hierarchy using Object-Oriented principles:

1. Vehicle (Base Class):
   - Properties: make, model, year
   - Methods: start, stop, accelerate, brake

2. Car (Inherits from Vehicle):
   - Additional Properties: numDoors, fuelType
   - Additional Methods: lockDoors, unlockDoors

3. Motorcycle (Inherits from Vehicle):
   - Additional Properties: hasHelmetStorage
   - Additional Methods: putOnHelmet, takeOffHelmet

4. Truck (Inherits from Vehicle):
   - Additional Properties: cargoCapacity
   - Additional Methods: loadCargo, unloadCargo

This hierarchy exemplifies encapsulation, where properties and methods are contained within each class, and inheritance, which allows specialized classes to inherit properties and methods from the base class, promoting code reusability and structure.

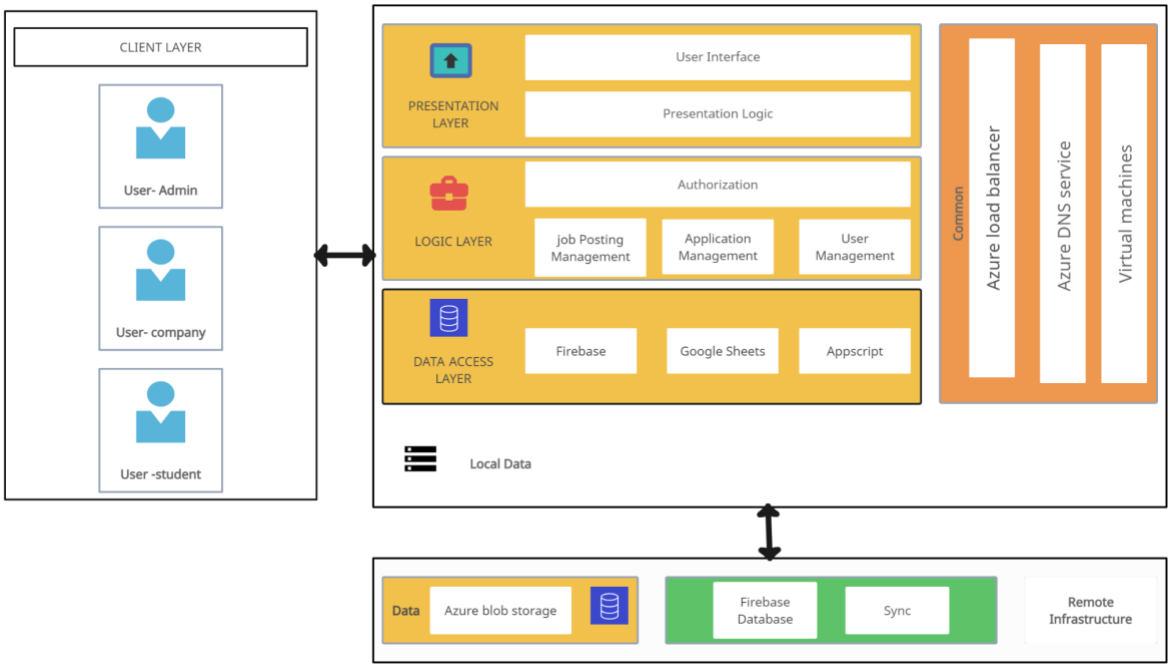
c) Evaluate the impact of cohesion and coupling on software maintenance, extensibility, and reusability in a real-world project scenario.
In a real-world project, cohesion and coupling have significant effects:

- Software Maintenance: High cohesion simplifies changes, and low coupling reduces unintended impacts during maintenance.

- Software Extensibility: High cohesion and low coupling ease the addition of new features and components.

- Software Reusability: Well-structured, cohesive, and loosely coupled code is more reusable in various contexts.

In practice, striking a balance between cohesion and coupling is crucial for business agility, cost savings**,** team collaboration, and quality assurance in long-term projects.



SE-architecture Diagram

Shortcomings of Training and placement application

1. **Limited Job Sources**: The platform might only include job postings from a limited number of companies. This could limit the variety of job opportunities available to students.
2. **Lack of Personalization**: The platform might not provide personalized job recommendations based on each student's interests, skills, and career goals.
3. **User Interface**: The user interface might not be intuitive or user-friendly, which could make it difficult for students to navigate the platform and find relevant information.
4. **Performance Issues**: If the platform is not properly optimized, it could have slow load times or other performance issues.
5. **Security Concerns**: If the platform does not implement robust security measures, it could be vulnerable to data breaches or other security threats.
6. **Limited Upskilling Resources**: The upskilling platform might only provide a limited range of resources, which could limit its usefulness for students.
7. **Accessibility**: The platform might not be fully accessible to all users, including those with disabilities.
8. **Mobile Optimization**: If the platform is not optimized for mobile devices, it could provide a poor user experience for students who prefer to access the platform on their phones or tablets.

Updates for Training and placement application

1. **Expanded Job Sources**: We could partner with more companies and job boards to increase the variety of job opportunities available to students.
2. **Improved Personalization**: We could implement a recommendation engine that can provide personalized job recommendations based on each student's interests, skills, and career goals.
3. **User Interface Improvements**: We could conduct user testing to identify areas of the user interface that can be improved. We aim to make the interface more intuitive and user-friendly.
4. **Performance Optimization**: We could optimize both the frontend and backend of our platform to improve load times and overall performance.
5. **Enhanced Security Measures**: We could implement robust security measures such as data encryption, secure user authentication, and regular security audits to protect against data breaches or other security threats.
6. **Expanded Upskilling Resources**: We could partner with educational institutions or online learning platforms to provide a wider range of upskilling resources for students.
7. **Improved Accessibility**: We aim to ensure our platform is fully accessible by following the Web Content Accessibility Guidelines (WCAG). This includes providing alt text for images, ensuring sufficient color contrast, and making the app navigable by keyboard, among other things.
8. **Mobile Optimization**: We could implement responsive design techniques to ensure our platform is optimized for mobile devices. We could also consider creating a native mobile app for a better mobile user experience.

POSTLAB

a) Analyse a given software design and assess the level of cohesion and coupling, identifying potential areas for improvement:

The software design of our platform demonstrates good levels of cohesion and coupling. Cohesion is evident in the well-defined components with distinct functionalities such as user management, job posting management, application management, and upskilling platform management. Loose coupling is maintained by allowing flexibility and minimal interdependence between these components.

To further improve the design, the platform can benefit from enhancing cohesion by ensuring each component has a single, clear responsibility. For instance, user management should solely focus on user registration, login, and role-based access control. Similarly, job posting management should only handle creating, updating, or deleting job postings.

Coupling can be reduced by minimizing dependencies between components. Changes in one component (like job posting management) should not necessitate changes in another component (like user management). This can be achieved by using interfaces or abstractions between components and ensuring data flow is well managed.