NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2002 OBJECT ORIENTED DESIGN & PROG

AY25/26 SEMESTER 1 GROUP ASSIGNMENT

Internship Placement Management System

Declaration of Original Work for SC2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	**Course**	**Lab Group**	**Signature/Date**
KEE KAI WEN (U2422273A)	SC2002	SCED	 18/11/2025
LIM JIA WEI JERALD (U2422241C)	SC2002	SCED	 18/11/2025
KELVIN TAY WEI JIE (U2422244J)	SC2002	SCED	 18/11/2025
KOAY JUN ZHI (U2240014J)	SC2002	SCED	 18/11/2025
TEO KAI JIE (U2420268G)	SC2002	SCED	 18/11/2025

Chapter 1: Intro **3**
Chapter 2: UML Class Diagram **3**
2.1 Core Class Structure 3
2.1.1 User Hierarchy 3
2.1.2 Entity 3
2.1.3 Data Management(Control) 4
2.1.4 Boundary 4
2.2 Some Key Relationships 4
Chapter 3: UML Sequence Diagram **5**
Chapter 4: Design Considerations **5**
4.1 OO Concepts 5
4.1.1 Abstraction 5
4.1.2 Polymorphism 6
4.1.3 Encapsulation 6
4.1.4 SOLID 6
4.2 Extensibility & Maintainability 7
4.2.1 New Filters/Field 7
4.2.2 New role 7
4.2.3 Shared patterns 7
4.2.3 User Inheritance 7
4.3 Additional design considerations 8
4.3.1 Password Security 8
4.3.2 Singleton Implementation 8
4.4 Reflection on Design Trade-offs 8
4.5 Alternative Design Considerations 8
Chapter 5: Implementation and Testing **9**
5.1 User Functionality 9
5.2 Student Functionality 10
5.3 Company Representative 11
5.4 Career Center Staff 12
Chapter 6: Reflection **12**

Chapter 1: Intro

The Internship Placement Management System (IPMS) is designed to act as a centralised hub for managing the internship process, and facilitating the interactions between the Students, Company Representatives, and Career Center Staff. To ensure that the system is maintainable, flexible and extensive, we designed the system using the SOLID guidelines and Boundary-Control-Entity (BCE) architectural pattern.

Chapter 2: UML Class Diagram

2.1 Core Class Structure

The UML Class Diagram **(Figure 2)** shows the breakdown of the structural design of the IPMS using the BCE architectural pattern.

The architecture is structured around key entities and their interactions, primarily organized by the principle of Inheritance.

2.1.1 User Hierarchy

- User (abstract): Serves as the base abstract class for the system users and holds the shared attributes like User ID, Name, and Password. It also contains the basic user management functions such as login and changePassword.
- Subclasses
    - Student: Additional attributes such as Year of Study and Major
    - CompanyRepresentatives: Additional attributes such as companyID, department and position.
    - CareerCenterStaff: Additional attributes like department

2.1.2 Entity

These represent the core data objects of the system and by employing encapsulation, the actual data is kept private and only accessed via public getter and setter methods.

- InternshipOpportunity: Stores all details pertaining to the list of internship opportunities. (internshipID, title, description, companyID, preferredMajor)
- InternshipApplication: Stores the details of a student's submission. (Student, InternshipOpportunity and Status)

- WithdrawalRequest: Stores the student's request for withdrawal before or after placement confirmation , which will be reviewed by the CareerCenterStaff.

2.1.3 Data Management (Control)

- User Managers: Manages the initialisation, persistence, and retrieval of all user and company records
    - StudentManager
    - CareerCenterStaffManager
    - CompanyRepresentativeManager
    - CompanyManager
- Listing Managers: Manages the collection lifecycle of the Internship entities
    - InternshipOpportunityManager
    - InternshipApplicationManager
    - InternshipWithdrawalRequestManager

2.1.4 Boundary

Role-Specific View Classes: Handles the detailed I/O and menu display in the CLI specific to each actor's functions.

- Examples: StudentView, CompanyRepresentativeView, CareerCenterStaffView

2.2 Some Key Relationships

Inheritance

- Establishes a clear hierarchy, with **User** serving as the abstract superclass for concrete types like **Student** and **CareerStaff**

Realization

- Promotes modularity, as nearly all Entity classes (e.g., **Company**) implement their corresponding interfaces (**ICompany**).

Dependency

- The majority of connections are **Dependencies**, illustrating how Boundary classes (**StudentView**) rely on Entity (**Student**) and Control (**StudentManager**) classes

Association and Aggregation

- Defines object multiplicity and ownership, such as a company offers up to 5 internships, and Control classes like **CompanyManager** managing multiple **Company** entities

Chapter 3: UML Sequence Diagram

The Sequence Diagram **(Figure 3)** shows the flow of the Company Representative's Role in Internship Management and Application Process. It demonstrates how the different classes interact with each other to show how the student applications are reviewed by representatives with approve/reject and slot fill.

Key Components:

- Interaction between Boundary, Control, Entity Classes (*selectApplication(applicationID)* → *getApplication(applicationID)* → *application*).
- The scenario involves essential business operations with complex validation requirements
- It showcases conditional business logic through explicit alternative flows and loops
- The diagram demonstrates complete user interactions with system-critical approval functions
- It highlights critical patterns like status management and automatic slot filling
- This workflow truly tests our system's ability to handle multi-object state transitions

Feature	**Implementation**	**Error Handling**	**Business Logic**
Status Transition Control	*setStatus(SUCCESSFUL/UNSUCCESSFUL)*	Choice validation (1 or 2 only)	Control Approve or Reject status
Automatic Slot Management	*updateFilledPlaces()* with status sync	Automatic FILLED status when slots full	Maintains real-time slot availability accuracy

Chapter 4: Design Considerations

4.1 OO Concepts

4.1.1 Abstraction

Shared state (UserID, name, email, password, filter settings) **(Figure 4.1.1a)** and behaviours are defined in the abstract 'User' class **(Figure 4.1.1b)**

Each role supplies its own implementation of the abstract methods, allowing us to reuse common logic while forcing role-specific behaviours in subclasses.

4.1.2 Polymorphism

After login and authentication, the system retains a 'User' reference and invokes 'displayHome' (**Figure 4.1.2)**. Dispatch is determined by the concrete role (Student, CompanyRepresentative,

CareerCenterStaff). This allows menus and actions to vary by role without changing the closer, improving flexibility

4.1.3 Encapsulation

Fields are private with public accessors. Hash/Verify operations are isolated in 'PasswordManager'. Manger would encapsulate persistence concerns and handle User ID. This helps to reduce accidental misuse of state and localises changes to storage or security.

4.1.4 SOLID

- **SRP**

Each class has one clear responsibility. Managers each look after their own data set. For example, PasswordManager only verifies and hashes **(Figure 4.1.4a),** and viewer only renders tables **(Figure 4.1.4b).** Hence when we change authentication, storage, or rendering, the adjustments stay within their classes and do not break other classes.

- **OCP**

We let the view and filter methods accept optional inputs **(Figure 4.1.4c)**. When a new filter or field is needed, we extend the filter object and the prompts while leaving the main query loop unchanged. This keeps the code open to extension without the need to modify code.

- **LSP**

We made use of LSP to program against the User base type and rely on each role (Student, CompanyRepresentative, CareerCenterStaff) to fulfil the base types without **adding stronger preconditions or weakening postconditions.** For example, Callers hold a User reference returned by login and invoke displayHome(), and the concrete role's implementation runs correctly without changing the caller.

- **ISP**

We follow the ISP by keeping interfaces small and focused on each role. The Callers only see the methods they need, which makes the code easier to change and test. For example, IStudent only has **student actions (Figure 4.1.4d)**, and ICgitareerCenterStaff only has **staff approval actions (Figure 4.1.4e).**

- **DIP**

To keep the code loosely coupled, we depend on interfaces and abstractions instead of concrete classes. Callers work with IStudent **(Figure 4.1.4d)**, ICareerCenterStaff, and ICompanyRepresentative rather than specific implementations, so higher-level logic stays

agnostic to how those roles are realised. This approach allows us to change or extend implementations without affecting the code that consumes them.

4.2 Extensibility & Maintainability

4.2.1 New Filters/Field

We can add new filter criteria by extending the filter object and manager query parameters. The existing prompts already accept optional input, so the core query loop does not need to change.

4.2.2 New role

When adding a new role, we **extend User**, implement the relevant role interface, add a manager if needed, and register a menu entry. As authentication and dispatch are driven by the base User, the login flow does not need to change.

4.2.3 Shared patterns

We follow the same patterns for persistence, ID generation, and table rendering, which keeps changes **predictable.** Managers own storage and ID logic, and formatting changes stay in Viewer, so updates in one area do not ripple through the rest of the code.

4.2.4 User Inheritance

The Student, CareerCenterStaff, and CompanyRepresentative classes **all extend User** and **implement** their respective role interfaces (e.g., IStudent, ICareerCenterStaff). This allows the system to invoke shared actions such as displayHome(Scanner) or viewProfile(Scanner) purely through a User reference. Because the login flow simply returns a concrete User, callers never need to know which specific role they are handling—polymorphism ensures the correct behaviour is dispatched. When introducing a new role, we only need to create a new subclass of User, define a small interface for its role-specific actions, and register its menu entry. The authentication and menu dispatch processes **remain untouched**, keeping the system **extensible** and easy to evolve.

4.3 Additional design Features

4.3.1 Password Security

We secured all credentials using the **SHA-256 hashing** and **verification** via PasswordManager. This ensures **no plaintext passwords** are stored or compared. This strengthens authentication integrity and reduces the risk of credential leakage.

4.3.2 Singleton Implementation

We used a Singleton for each manager so only **one in-memory instance exists**, preventing **duplicated** or **conflicting states**. Each getInstance() loads the serialized data into a static manager and returns that single object. When IPMS shuts down, close() serializes the in memory list back to the .dat file and resets the singleton **(Figure 5.3.3)**.

4.4 Reflection on Design Trade-offs

We aimed for a simple, lightweight console app, so we stayed with straightforward data managers and serialization.

A database or JSON backend would cope better if the data shape changes, but that was beyond this assignment. We kept some business checks in the menus because it made the flows **easy to follow**. In the future we would move those rules into dedicated services so they are easier to reuse and test.

We hashed passwords and set requirements for it to ensure **security** at a basic level. However in the future we should add extra safeguards like flagging out repeated log in and strengthening to strengthen the security even further.

Visibility and suitability are currently enforced through filters rather than hard rules. This works for now, but we could tighten it later to reduce mistakes.

4.5 Alternative Design Considerations

While designing IPMS, we examined several architectural options to improve coupling, testability, and maintainability, but scoped some of them out to meet assignment timelines and keep the console implementation lightweight.

Service layer with dependency injection. We considered adding a **dedicated service layer** and **using dependency injection** to decouple callers from concrete managers and utilities. This would let us swap implementations or inject test doubles without touching consumers, greatly improving configurability and automated testing. We kept the current static managers (E.g. StudentManager.getInstance()) to minimise plumbing in a console app.

Repository adapters vs serialization. We evaluated using CSV, JSON, or a database with repository adapters to improve portability and schema evolution. We chose Java serialization so as to keep the app offline and lightweight, accepting that a later migration would be needed to handle schema changes or multi-platform deployments.

Observer or event-driven updates. We looked at using an observer pattern to propagate status changes across managers. To preserve predictability and simplicity in the console flow, we retained direct manager calls and omitted event wiring.

Chapter 5: Implementation and Testing

5.1 User Functionality

No.	Test Case	Expected Behaviour	Failure Indicators	Pass/Fail
1	Valid User Login	User logs in with correct ID/password and sees the correct role menu	Cannot log in with valid credentials or sees the wrong menu. Incorrect error shown. Login bypasses approval checks.	Pass
2	Invalid ID	System rejects unknown ID and shows a clear error, stays at login prompt.	Unknown ID is accepted or no meaningful error is shown.	Pass
3	Incorrect Password	The system denies access, prompts again and does not login.	Logs in with the wrong password or shows no meaningful error.	Pass
4.	Password Change	Password hash updates. Subsequent login requires the new password.	Password not updated or old password still works	Pass
5.	Default Password Requirement	All users must be able to log in with default password “password” on first login.	Default password rejected	Pass
6.	Role menu dispatch	After login, displayHome() for the correct role runs (Student/Rep/Staff)	Wrong role menu or crash on dispatch	Pass

5.2 Student Functionality

No.	Test Case	Expected Behaviour	Failure Indicators	Pass/Fail
1	View/Filter internships with optional inputs	Blank inputs are accepted, only provided criteria filter results. No errors on blanks.	Blanks cause errors or filters ignore provided criteria.	Pass

2.	Visibility Filtering based on Profile	Students see only internships that match their major, level eligibility, and visibility on.	Students see internships of the wrong major, level or hidden ones.	Pass
3	Apply unique internship under cap	New application created with status PENDING when not already applied, status APPROVED/open, cap not exceeded.	Application not created in valid scenario or wrong status.	Pass
4	Apply over cap (max 3)	Rejection message. No new application added when already at 3.	Fourth application is accepted or no clear rejection.	Pass
5	Duplicate Application	Rejection when applying again to the same internship as the same student.	Duplicate application is created.	Pass
6	Apply to different internship under same company	Application is successfully created	Application is rejected from creating due applying to the same company	Pass
7	Application to intermediate or advanced as Y1-2	Rejection message, no application created for Advanced when student is Year 1 or 2.	Application is created or no warning shown	Pass
9	Withdraw Application	Withdrawal request added for ACCEPTED/SUCCESSFUL app not already requested, duplicates blocked.	No request created for valid case, or duplicate requests allowed	Pass
10	Accept successful offer	Status set to ACCEPTED. Other student applications removed, filled slots updated.	Status not changed, other apps not cleaned up, or slots not updated.	Pass
11	Reject successful offer	Status set to REJECTED. Other apps are unaffected.	Status unchanged or other apps wrongly removed	Pass
12	Visibility off but app retained	Students still see their own application status even if the internship visibility is turned off.	Students cannot see their own application after visibility is off.	Pass
13	Internship closing date passed	Students unable to apply for internship after closing date	Student still able to apply for internship after closing date	Pass

5.3 Company Representative

No.	Test Case	Expected Behaviour	Failure Indicators	Pass/Fail
1	Register new rep	New company rep saved with PENDING status. Cannot log in until approved	Company rep can log in without approval or status not set to PENDING	Pass
2	Create internship with validation	New opportunity created with PENDING status when fields are valid. Invalid dates prompt until corrected.	Opportunity created with bad dates or wrong status. No validation	Pass
3	Internship Edit allowed before Staff Approval	Company rep able to edit internship details while status is PENDING.	Company rep unable to edit internship details while status is PENDING	Pass
4	Approve/Reject Applicant	PENDING application for rep's company changes to SUCCESSFUL or UNSUCCESSFUL as chosen.	Status unchanged or wrong application updated.	Pass
5	Toggle visibility	Company rep able to turn visibility On/Off. Students no longer see it in listings, but their own applications remain visible.	Visibility does not change or the student's own application disappears.	Pass
6	Posting cap	Attempting to create a 6th posting is rejected if cap is enforced.	More than 5 postings allowed when a cap should apply.	Pass

5.4 Career Center Staff

No.	Test Case	Expected Behaviour	Failure Indicators	Pass/Fail
1	Approve/Reject company rep	PENDING rep status set to APPROVED or REJECTED. Approved reps can log in.	Status unchanged or rep login allowed while PENDING.	Pass
2	Approve/Reject	PENDING opportunity status set to	Status unchanged or not	Pass

	internship opportunity	APPROVED or REJECTED. Approved internships appear visible to students.	reflected in student views.	
3	Approve/Reject Withdrawal request	PENDING withdrawal status updated. Approved withdrawals are removed from the app and filled counts updates.	Status unchanged or app not removed/filled counts not updated.	Pass
4	Report Generation & Filtering	Reports filter correctly by major, level, status and company.	Filtering is incorrect	Pass

Chapter 6: Reflection

This assignment allowed us to apply Object-Oriented Design (OOD) principles, translating theoretical concepts into a functional system. One of the biggest challenges was maintaining consistency between our Class diagram, Sequence diagram, and the actual code. During implementation, we often had to make additional changes that needed to be reflected in our diagrams. We addressed this by continuously updating them to ensure alignment. The multiple interactions between different objects were complex, but breaking them down into smaller use cases and using a checklist helped us verify that we met all requirements.

Working on this assignment highlighted the importance of a structured workflow. Applying knowledge from the course, we learned to start by analyzing requirements for the different Boundary-Control-Entity (BCE) components before developing the class and sequence diagrams. Leveraging OOP concepts from the first half of the course, we implemented abstraction, polymorphism, encapsulation, and error handling. We also incorporated good design practices to ensure the system's flexibility, maintainability, and extensibility.

For future improvement, we believe that introducing these design practices earlier in the curriculum would be beneficial. This would allow students to incorporate them from the initial planning stages, reducing the need for frequent diagram revisions later on.

Overall, this assignment provided a valuable opportunity to deepen our understanding of the course concepts. It enabled us to put theory into practice and develop a maintainable and reliable system.

Github Link: https://github.com/pixelhypercube/sc2002-oop.git
Figure references can be found in Github Repo under **“Figures.md”**