SC2002 OBJECT ORIENTED DESIGN & PROGRAMMING
Internship Placement Management System Project Report
AY25/26 Sem 1 SCEC Group 1
Github Repository Link:
https://github.com/darthrevan030/TUI-Internship-Placement-Management-System

Declaration of Original Work for SC2002/CE2002/CZ2002 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 (SC2002/CE2002/CZ2002)	Lab Group	Signature /Date
MADHAV RAGHU ANANTHARAM	SC2002	SCEC	10 November 2025 [illegible]
LIM MING WEN	SC2002	SCEC	10 November 2025 [illegible]
SAMARTH BHATIA	SC2002	SCEC	10 November 2025 [illegible]
TAN CHOON YANG	SC2002	SCEC	10 November 2025 [illegible]
TOH JUN MENG	SC2002	SCEC	10 November 2025 [illegible]

1 DESIGN CONSIDERATIONS

The Internship Placement Management System (IPMS) is a Java console application designed with a focus on reusability, extensibility, and maintainability. It efficiently manages internship opportunities, student applications, and placement processes, accommodating three distinct user types (Students, Company Representatives and Career Center Staff) and their respective requirements, allowing for easy upgrades and future development.

1.1 Design Approach

The IPMS was designed with a focus on high cohesion and loose coupling, with classes separated into three categories: controllers (control package), boundaries (boundary package), and entities (entity package).

- Controllers include `ApplicationManager`, `InternshipManager`, `UserManager`, `AuthenticationManager`, and `ReportGenerator`
- Boundaries include `StudentUI`, `CompanyRepUI`, `StaffUI`, `MainUI`, and `InputValidator`
- Entities include `Student`, `CompanyRepresentative`, `CareerCenterStaff`, `InternshipOpportunity`, `Application`, `ApplicationStatus`, `InternshipLevel`, `OpportunityStatus`, `User`, `Withdrawal Status` and `WithdrawalRequest`.

When a user interacts with the system, they interact with the boundaries, which then call the controllers to perform requested operations like making changes to entities or retrieving information from entities to display. Each of these categories works together to complete our system while ensuring the dependency on each other is minimised. As such, our system is highly flexible, extendable, and easy to maintain. Minimum effort is required when extending our system, such as when a new function is introduced.

1.2 Highlights of Some Designs

Singleton Pattern: All manager classes (`UserManager`, `InternshipManager`, `ApplicationManager`, `AuthenticationManager`) implement the Singleton pattern to ensure only one instance manages the respective domain throughout the application lifecycle.

Strategy Pattern: The filtering system uses the Strategy pattern with a `FilterStrategy` interface and multiple implementations (`StatusFilter`, `LevelFilter`, `MajorFilter`, `VisibilityFilter`), allowing flexible and extensible filtering of internship opportunities.

Composite Pattern: The `CompositeFilter` class allows combining multiple filters to create complex filtering criteria, demonstrating the Composite design pattern.

Data Persistence with Serialisation: The system uses Java serialisation to persist data to `.dat` files, enabling data to survive across sessions. Initial data can be loaded from CSV files for easy bulk imports.

Input Validation Utility: The `InputValidator` class provides centralised, reusable validation methods for all user inputs, ensuring data integrity and improved user experience.

Inheritance and Polymorphism: The abstract `User` class serves as a base for `Student`, `CompanyRepresentative`, and `CareerCenterStaff`, with each subclass implementing role-specific behaviour while sharing common attributes and methods.

1.3 Applied Design Principle

1.3.1 Single Responsibility Principle (SRP)

The Single Responsibility Principle recommends that each class should have a clear and singular responsibility, avoiding unrelated tasks. By adhering to SRP, we minimise the ripple effect of changes, simplifying the process of modifying, testing, and reusing code, resulting in more maintainable and robust software design. For example,

- `AuthenticationManager` solely handles login/logout and session management
- `ApplicationManager` focuses only on application lifecycle management
- `InputValidator` handles only input validation logic
- `ReportGenerator` is responsible exclusively for generating reports

Each manager class has a single, well-defined responsibility, making the codebase easier to understand and maintain.

1.3.2 Open/Closed Principle (OCP)

The Open/Closed Principle states that classes should be open for extension but closed for modification, allowing for the addition of new functionality without changing existing code. OCP can be implemented through abstraction, inheritance, and polymorphism. For example,

- The `FilterStrategy` interface allows new filter types to be added without modifying existing filter code or the `ReportGenerator`
- The abstract `User` class can be extended to create new user types (e.g., Administrator) without modifying the authentication logic
- New internship levels can be added to the `InternshipLevel` enum without affecting the core business logic
- The `ApplicationStatus` and `OpportunityStatus` enums can be extended with new states without changing existing code

1.3.3 Liskov Substitution Principle (LSP)

To put it simply, the Liskov Substitution Principle states that subtypes must be substitutable for their base types. In IPMS, this principle is widely applied. For example,

- All `User` subclasses (`Student`, `CompanyRepresentative`, `CareerCenterStaff`) can be used wherever a `User` is expected
- The `AuthenticationManager` works with `User` objects without needing to know the specific subtype
- All `FilterStrategy` implementations can be used interchangeably in the filtering system
- The `displayMenu()` method is called polymorphically on any `User` object, and the correct implementation executes based on the actual user type

This ensures behavioural consistency across the inheritance hierarchy while allowing for type-specific implementations.

1.3.4 Interface Segregation Principle (ISP)

The Interface Segregation Principle refers to having many specific interfaces rather than one general interface. In other words, we should always avoid designing a 'fat interface'. When developing our system, we noticed that this is important to promote maintainability, flexibility, and modularity. For example,

- `FilterStrategy` is a focused interface with a single method `filter()`, rather than a bloated filtering interface
- The `Serializable` interface is used only where needed for data persistence
- Each entity class implements only the interfaces it needs, rather than forcing all entities to implement a monolithic interface

By keeping interfaces lean and focused, we ensure that classes don't implement methods they don't need, reducing coupling and improving maintainability.

1.3.5 Dependency Injection Principle (DIP)

The Dependency Inversion Principle suggests that high-level modules should not depend on low-level modules; both should depend on abstractions. Instead of directly depending on concrete classes, we depend on interfaces or abstract classes, which are less likely to change. For example,

- UI classes depend on manager interfaces/abstract classes rather than concrete implementations
- `ReportGenerator` depends on the `FilterStrategy` abstraction, not concrete filter classes
- `AuthenticationManager` works with the abstract `User` type rather than concrete user classes
- Controllers depend on entity abstractions, allowing entity implementations to change without affecting business logic

This principle allows us to add more user types or modify existing functionality with minimal impact on the system, making it easy to be extended.

1.4 Additional Features Implemented

Beyond the baseline requirements specified in the assignment, our system includes the following enhancements that demonstrate advanced OOP principles and improve system functionality:

1. **Automatic Slot Management**

When an internship opportunity reaches its maximum capacity (all slots filled), the system automatically updates its status to `FILLED`, preventing further applications and ensuring data integrity.

2. **Cascade Withdrawal on Placement Acceptance**

When a student accepts a placement, all their other pending and successful applications are automatically withdrawn, reducing manual processing, preventing double-bookings and ensuring "one placement per student" business rule compliance.

3. **User-Friendly Interface Design**

Every time a user is prompted to select from a list (internships, applications, etc.), the system first displays all available options with detailed information, then prompts for input. There is also a consistent UX pattern across all menus. This improves usability and reduces errors.

4. **Comprehensive Composite Filtering System**

Career Centre Staff can generate reports with complex, multi-criteria filters using the Composite pattern, enabling detailed analysis of internship opportunities and applications across various dimensions (status, level, major, visibility). This is a highly extensible design, a key property of object oriented design and principles.

5. **Withdrawal Request Approval System**

Students can request withdrawal from applications, but withdrawals require approval from Career Centre staff, ensuring proper oversight. The system distinguishes between withdrawals before and after placement acceptance, with different business implications. The system also tracks withdrawal reasons for audit purposes.

6. **Company Representative Approval Workflow**

New company representatives must be approved by Career Centre staff before they can log in and create internship opportunities, ensuring quality control and preventing unauthorised access.

These enhancements demonstrate our commitment to creating a production-ready system that goes beyond basic requirements while maintaining clean OOP design principles.

7. **Centralised Input Validation**

The system implements a centralised `InputValidator` utility class that provides reusable validation methods across all user interfaces. This includes email format validation, date format checking, integer and string validation, and confirmation prompts. By consolidating validation logic in a single location, the system improves data integrity and enhances user experience with consistent error messaging.

8. **Java Serialisation for Data Persistence**

Beyond the required CSV initialisation, our system implements full Java serialisation to `.dat` files for comprehensive data persistence. This approach preserves all system state, including users, internships, and applications, across sessions, enabling seamless recovery after a system restart. The serialisation mechanism ensures that all changes made during a session are retained, providing a robust foundation for production deployment.

9. **Application Statistics Report**

The system includes a dedicated statistical report that provides Career Centre staff with percentage breakdowns and trend analysis of application patterns, enabling comprehensive oversight of student engagement and internship popularity. The statistical insights support data-driven decision-making and help identify areas requiring attention or improvement.

10. **Real-Time Slot Availability Tracking**

All internship listings throughout the system display slot availability in an "X/Y filled" format, where X represents filled positions and Y represents total capacity. This dynamic calculation of available slots is updated in real-time and prevents students from applying to fully-booked internships. The transparency provided by this feature enhances the user experience for all stakeholders and reduces frustration from attempting applications to unavailable positions.

1.5 Reflection

This assignment underscored the critical role of design principles, as demonstrated through practical application. Initially, there were issues with the login system and we found that minor code alterations led to cascading changes throughout the project. By subsequently integrating design principles, we developed software exhibiting high cohesion and loose coupling, resulting in enhanced flexibility, maintainability, and extensibility. Concurrently, we gained insight into crafting software that precisely aligns with its intended functions and real-world use cases. Our design process involved continuous refinement, considering diverse user perspectives and potential scenarios to prevent conflicts and ensure robust functionality.

The implementation of design patterns (Singleton, Strategy, Composite) proved invaluable in creating a scalable system. The Singleton pattern for managers ensured data consistency, while the Strategy pattern for filtering provided flexibility without code modification. The separation of concerns through the three-layer architecture (boundary, control, entity) made testing and debugging significantly easier.

We also learned the importance of business rule enforcement at the entity level. By embedding constraints like "maximum 3 applications per student" and "maximum 5 internships per company representative" directly into the entity classes, we ensured that these rules are consistently applied regardless of how the entities are accessed.

2 DETAILED /UML CLASS DIAGRAM

2.1 Main Diagram

Found at: “.\docs\images\uml-diagrams\classdiagram.png”

2.2 Entity Sub Diagram

Found at: “.\docs\images\uml-diagrams\entity.png”

2.3 Controller Sub Diagram

Found at: “.\docs\images\uml-diagrams\control.png”

2.4 Boundary Sub Diagram

Found at: “.\docs\images\uml-diagrams\boundary.png”

3 DETAILED UML SEQUENCE DIAGRAM

3.1 Main Diagram

Found at: “.\docs\images\uml-diagrams\sequence-diagram.jpg”

3.2 Login & Authentication Sub Diagram

Found at: “.\docs\images\uml-diagrams\Login & Authentication Sub-Sequence Diagram.jpg”

3.3 Update Application Status Sub Diagram

Found at: “.\docs\images\uml-diagrams\Update Application Status Subsequence.jpg”

4 TESTING

NO.	Test Cases	Expected Behaviour: The image can be found in the submitted folder:	Failure Indicators
4.1	Welcome Page	“./docs/images/test-cases/4.1 Welcome Page.png”	Application fails to start or displays menu incorrectly.
4.1.1	Login	“./docs/images/test-cases/4.1.1 Login.png”	User cannot log in with valid credentials or receives incorrect error messages.
4.1.2	Register as Company Representative	“./docs/images/test-cases/4.1.2 Register as Company Representative.png”	Registration fails with valid data or succeeds with duplicate email.
4.1.3	Exit	“./docs/images/test-cases/4.1.3 Exit.png”	Application doesn't terminate properly or crashes on exit.
4.2	Student Menu	“./docs/images/test-cases/4.2 Student Menu.png”	Menu not displayed or invalid choices cause system crash.
4.2.1	View Internship Opportunities	“./docs/images/test-cases/4.2.1 View Internship Opportunities.png”	Students see internships not relevant to their profile (wrong major, wrong level, visibility OFF, or wrong status).
4.2.2	Apply for Internship	“./docs/images/test-cases/4.2.2 Apply for Internship.png”	Students can apply for more than 3 internships, apply for ineligible opportunities, or submit duplicate applications.
4.2.3	View My Applications	“./docs/images/test-cases/4.2.3 View My Applications.png”	Applications become inaccessible or show incorrect status information.
4.2.4	Accept Placement	“./docs/images/test-cases/4.2.4 Accept Placement.png”	Student can accept multiple placements or other applications remain active after accepting one.
4.2.5	Request Withdrawal	“./docs/images/test-cases/4.2.5 Request Withdrawal.png”	Withdrawal requests fail to submit or can be created for already withdrawn/unsuccessful applications.
4.3	Career Center Staff Menu	“./docs/images/test-cases/4.3 Career Center Staff Menu.png”	Menu not displayed or accessible by non-staff users.
4.3.1	Approve Company Representatives	“./docs/images/test-cases/4.3.1 Approve Company Representatives.png”	Approval doesn't enable representative login or changes aren't saved.
4.3.2	Approve Internship Opportunities	“./docs/images/test-cases/4.3.2 Approve Internship Opportunities.png”	Approved internships don't become visible to students or status doesn't update correctly.
4.3.3	Approve Withdrawal	“./docs/images/test-cases/4.3.3	Slot counts don't update properly or application

	Requests	Approve Withdrawal Requests.png”	status remains unchanged after approval.
4.3.4	Generate Reports	“./docs/images/test-cases/4.3.4 Generate Reports.png”	Report generation fails or menu doesn't display properly.
4.3.4.1	Internship Opportunities Report (with filters)	“./docs/images/test-cases/4.3.4.1 Internship Opportunities Report (with filters).png”	Report generation fails or menu doesn't display properly.
4.3.4.2	Application Statistics Report	“./docs/images/test-cases/4.3.4.2 Application Statistics Report.png”	Application counts or percentages are calculated incorrectly.
4.3.5	View All Internships	“./docs/images/test-cases/4.3.5 View All Internships.png”	Missing internships or displaying incorrect information (slots, status, application counts).
4.4	Company Representative Menu	“./docs/images/test-cases/4.4 Company Representative Menu.png”	Unapproved representatives can access menu or menu doesn't display.
4.4.1	Create Internship Opportunity	“./docs/images/test-cases/4.4.1 Create Internship Opportunity.png”	System allows creating more than 5 opportunities or accepts invalid data (closing date before opening, invalid slots).
4.4.2	View My Internship Opportunities	“./docs/images/test-cases/4.4.2 View My Internship Opportunities.png”	Shows other representatives' internships or displays incorrect slot/application counts.
4.4.3	View Application	“./docs/images/test-cases/4.4.3 View Applications.png”	Representative can view applications for other companies' internships or sees incomplete student information.
4.4.4	Review Applications	“./docs/images/test-cases/4.4.4 Review Applications.png”	Can review already processed applications or status updates don't persist correctly.
4.4.5	Toggle Visibility	“./docs/images/test-cases/4.4.5 Toggle Visibility.png”	Can toggle visibility for non-approved internships or changes don't reflect in student view.
4.5.1	Change Password	“./docs/images/test-cases/4.5.1 Change Password.png”	Accepts incorrect current password or new password doesn't work after change.
4.5.2	Logout	“./docs/images/test-cases/4.5.2 Logout.png”	User remains logged in or can access menu without re-authentication.