**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 honoured the principles of academic integrity and have upheld the 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 (Matriculation number)** | **Course** | **Lab group** | **Signature/ Date** |
| --- | --- | --- | --- |
| Amaravel Babu Arran  (U2423193C) | AISC | FCS6-G2 | 24/04/25 |
| Chia Jia Yuun  (U2421575D) | DSAI | FCS6-G2 | 24/04/2025 |
| Ivan Wong Jia Ming  (U2422775A) | CSC | FCS6-G2 | 24/04/2025 |
| Joey Tang See Ying  (U2422656K) | CSC | FCS6-G2 | 24/04/2025 |
| Khant Naing Lin  (U2421449K) | CSC | FCS6-G2 | 24/4/2025 |

**Chapter 1: Requirement Analysis & Feature Selection**

1.1 Understanding the Problem and Requirements

**Identification of the Main Problem Domain**

We began by reading through the assignment document, highlighting all use cases and system requirements. Based on this, we created a list of essential features and identified user roles and system entities. From this, we identified 3 key users: Applicants, HDB Officers and HDB Managers. Next, we extracted each stakeholder's specific capabilities, which were translated into use cases. By analysing these use cases, we identified the core domain entities of Users, Projects, Applications, Flats and Enquiries. Then we began piecing the business processes together, ie the flow of the system for the various processes: Application lifecycle , Project management , Enquiry handling. Finally, we zoomed in on the various constraints, such as eligibility requirements based on age and marital status. This is to ensure that we can implement additional core functions that help to further enhance the accuracy of the system.Through this analysis, we identified that the system's problem domain centres around the management of Build-to-Order housing projects in Singapore, encompassing five key areas: User management , Project management, Application processes, Flat allocation and booking, Enquiry handling. After identifying these areas and solidifying our problem domain, we began developing the class structure that would support these functionalities, as shown in our class and sequence diagrams.

**Explicit requirements**  
User authentication

1. Using Singpass
   1. NRIC-based login (S/T + 7 digits + letter)
   2. Password management and the ability to change passwords
2. Different access levels based on roles

User roles with distinct capabilities

* Applicants can view and apply for one project with specific qualities, track their applied project and its application status, book a flat via an HDB Officer if the status is ‘Successful,’ withdraw their BTO application before or after booking, and submit, view, edit, or delete their own enquiries.
* HDB Officers can register to join a project if not already assigned to another, tracking their application status via their profile interface. They manage applications, view details of their handled projects, respond to project enquiries, book flats for applicants with successful BTO applications, and generate receipts with booking details.
* HDB Managers can view, create, edit, and delete BTO project listings, information, and visibility for applicants, as well as view all projects regardless of visibility settings. They can view and approve HDB Officer applications and BTO applications/withdrawals within their projects, generate filtered applicant reports, and manage enquiries by viewing all projects’ enquiries and replying to those for their own project.
* The BTO Management System enforces business rules like age and marital status eligibility (singles 35+ for 2-Room, married 21+ for all types), one application per applicant, officers barred from applying to managed projects but allowed for others, project visibility controls, and application period enforcement, showing only relevant projects. It features project creation (name, neighborhood, flat types, dates), enquiry management, application tracking, report and receipt generation, using a CLI with menu-based navigation.

**Implicit requirements**

Authentication and Authorisation

1. The system needs robust user authentication since it is handling sensitive housing applications
2. There should be role-based access control implemented to restrict functionality based on user types (Applicant, HDB Officer, HDB Manager).

Transaction Management

1. If a flat booking fails, the system should roll back to the previous state
2. Database consistency should be ensured when updating flat availability count.

Exception Handling

1. Proper error messages for time-related operations (applications outside the valid period)

Notification System

1. Users would expect to be notified of application status changes

**Ambiguous or missing parts, and how we addressed them**

The document doesn't specify how enquiries are identified, stored, or associated with projects.

1. We implemented an EnquiryList with methods for adding, retrieving, and managing enquiries, assuming enquiries have unique IDs and are linked to both senders and projects.

How exactly flats are assigned to successful applicants isn't fully described in the document.

1. We created a FlatList class that contains methods like book\_2room() and book\_3room() that find the first available flat of the specified type, utilising a first-come-first-served approach to assign flats to the successful applicants.

1.2 Deciding on Features and Scope

We categorised features as core, optional, or excluded. Core features focused on showcasing strong OOP principles without overcomplicating the system. Optional features were non-essential and implemented if time allowed. Excluded features were either unnecessary or beyond the project scope.

1. User authentication (CORE) - login/logout, Role-based access control, Password changing functionality

Rationale: Role-based access is essential per requirements; the system depends on it to control user actions. Feasibility: Simple to implement using standard design patterns. Timeline: High-priority task, implemented early due to its system-wide impact.

1. Project management (CORE) - Create/edit/delete BTO projects, Toggle project visibility, View projects and Project details

Rationale: Central functionality that enables all other features; explicitly required for HDB Managers. Feasibility: Requires complex data structures but follows clear object-oriented patterns. Timeline: Must be implemented early, as projects are the foundation of the system.

1. Application processing (CORE) - Viewing application status, Approval/rejection of applications, Flat booking, Requesting and processing withdrawals

Rationale: Primary use case for both applicants and officers; directly fulfils the system's core purpose. Feasibility: Complex business rules, but clearly defined in requirements. Timeline: Dependent on project management implementation, but high priority.

1. Officer registration (CORE) - Register as an officer for a project, Approve/reject officer registrations, View registration status

Rationale: Critical workflow for system operation; explicitly required in the assignment. Feasibility: Moderately complex due to validation rules around date conflicts. Timeline: Must be implemented after user authentication but before application processing.

1. Enquiry system (CORE) - Submit enquiries, View and reply to enquiries

Rationale: Required communication channel is mentioned in the requirements for all user types. Feasibility: Straightforward CRUD operations. Timeline: Can be implemented in parallel with other core features.

1. Receipt Generation (CORE) - Detailed receipt format

Rationale: Required for HDBOfficers and applicants' referral. Feasibility: Simple string formatting but with attention to detail. Timeline: Can be implemented after the booking functionality works.

1. Advanced Report Generation (OPTIONAL, implemented) - Filter reports by various criteria (age, marital status, flat type), Export reports in different formats

Rationale: Basic reporting is required, but sophisticated filtering is an enhancement. Feasibility: Requires additional data manipulation but builds on existing structures. Timeline: Can be implemented after core functionality is stable.

1. Enhanced Enquiry Management (OPTIONAL, implemented) - Edit and delete enquiries, Categorise enquiries, Search functionality for enquiries

Rationale: Basic enquiry submission is essential, but editing/deleting are add-ons.

Feasibility: Extends basic enquiry system with moderate effort. Timeline: Enhancement that follows basic enquiry implementation.

1. User Interface Improvements (OPTIONAL, implemented) - Filtering and sorting project lists, Saved user preferences across sessions, Clear menu navigation

Rationale: Core functionality works with basic interfaces, but sorting/filtering improves the user experience. Feasibility: Depends on complexity; moderate challenge. Timeline: Can be addressed incrementally after core features work.

1. Time-based Validations (OPTIONAL, implemented) - Application period enforcement, Automatic status updates based on dates

Rationale: Needed for project application periods, but automated handling is an enhancement. Feasibility: Requires careful date handling with moderate complexity. Timeline: Basic validation early, advanced features later.

1. Advanced Security (EXCLUDED) - Password encryption, Session timeouts, Login attempt limitations

Rationale: Basic authentication satisfies requirements; encryption and advanced security exceed the scope. Feasibility: Requires significant additional expertise and testing. Timeline: Would extend beyond project deadline.

1. User Profile Management (EXCLUDED) - Update personal information, View application history across projects

Rationale: Basic user information is required, but comprehensive profile management exceeds requirements. Feasibility: Would require additional screens and data management. Timeline: Not prioritised due to time constraints

**Chapter 2: System Architecture & Structural Planning**

2.1 Planning the System Structure

In designing the BTO Management System, we divided it into logical components based on domain concepts and technical responsibilities:

* User Management Component: Manages authentication, authorization, and user operations, including the abstract User class and specialized classes (Applicant, HDBOfficer, HDBManager).
* Project Management Component: Handles project operations, property management, and listings, with classes like BTOProject, FlatList, Flat, and ProjectDatabase.
* Application Processing Component: Oversees the application lifecycle, featuring BTOApplication, ApplicationManager, and ReportGenerator.
* Registration Management Component: Manages officer registration and approval, with Registration and RegistrationManager.
* Enquiry Management Component: Handles enquiry creation, viewing, and responses, using Enquiry, EnquiryList, and manager classes.

**Interface Component:**We utilized interfaces like I\_manager and I\_applicant\_EnquiryM to achieve loose coupling through dependency inversion. This modular design separated concerns, facilitating parallel development, independent testing, and resulting in low coupling and high cohesion. We mapped key user flows—HDB Officer registration, BTO project application, and flat booking—to shape our class and sequence diagrams. For example, the registration flow prompted the creation of a dedicated Registration class for improved status tracking and validation.

2.2 Reflection on Design Trade-offs

**Trade-offs:**We considered merging the controller and logic layers for simplicity but kept them separate for better maintainability and testability. Instead of using large classes, we introduced focused manager classes (e.g., *ApplicationManager*, *ProjectManager*) to ensure clear responsibilities. Core features were fully implemented, while less critical ones like advanced reporting were kept minimal to meet deadlines. We also opted for smaller, single-purpose classes (e.g., *Enquiry* vs *EnquiryList*) for clarity and easier maintenance. Initially, we planned a single interface, but to align with SOLID principles (Single Responsibility, Interface Segregation), we created multiple specific interfaces tailored to each user role.

**3. Object-Oriented Design**

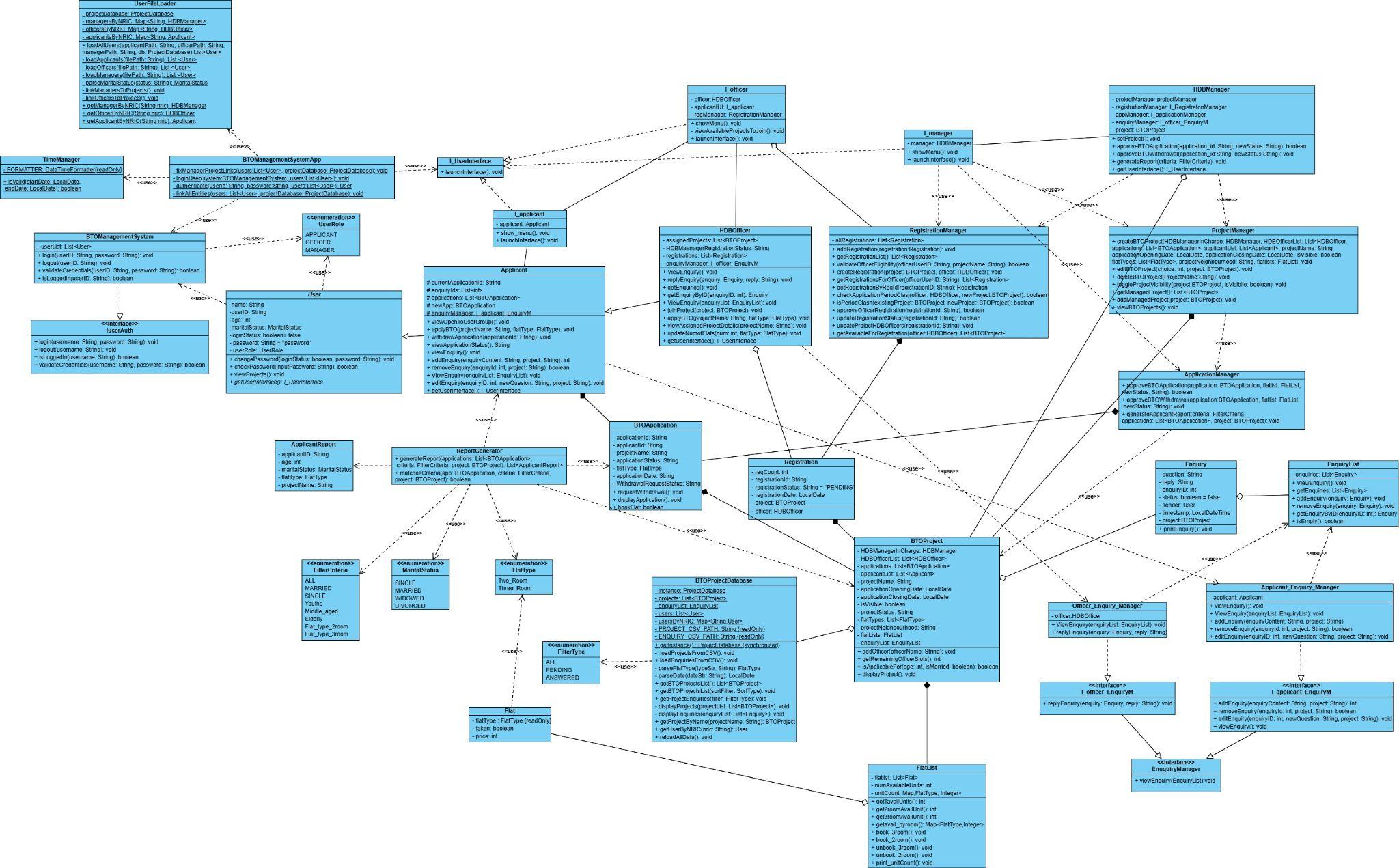
3.1 Class Diagram (with Emphasis on Thinking Process):

After reviewing the Project Requirement Specification, we identified key nouns like HDBOfficer, HDBManager, Applicant, User, BTOApplication, and BTOProject, focusing on core entities with direct roles. We analyzed their interactions to define methods (e.g., approveOfficerRegistration() for HDBManager, applyForProject() for Applicant).

We identified class relationships in two steps:

* Classified relationships as is-a (inheritance), has-a (association, aggregation, composition), or use-a (dependency).
* Determined composition (dependent lifecycle) versus association/aggregation (independent lifecycle).

Finally, we distinguished classes from attributes by assessing each noun’s complexity and lifecycle.

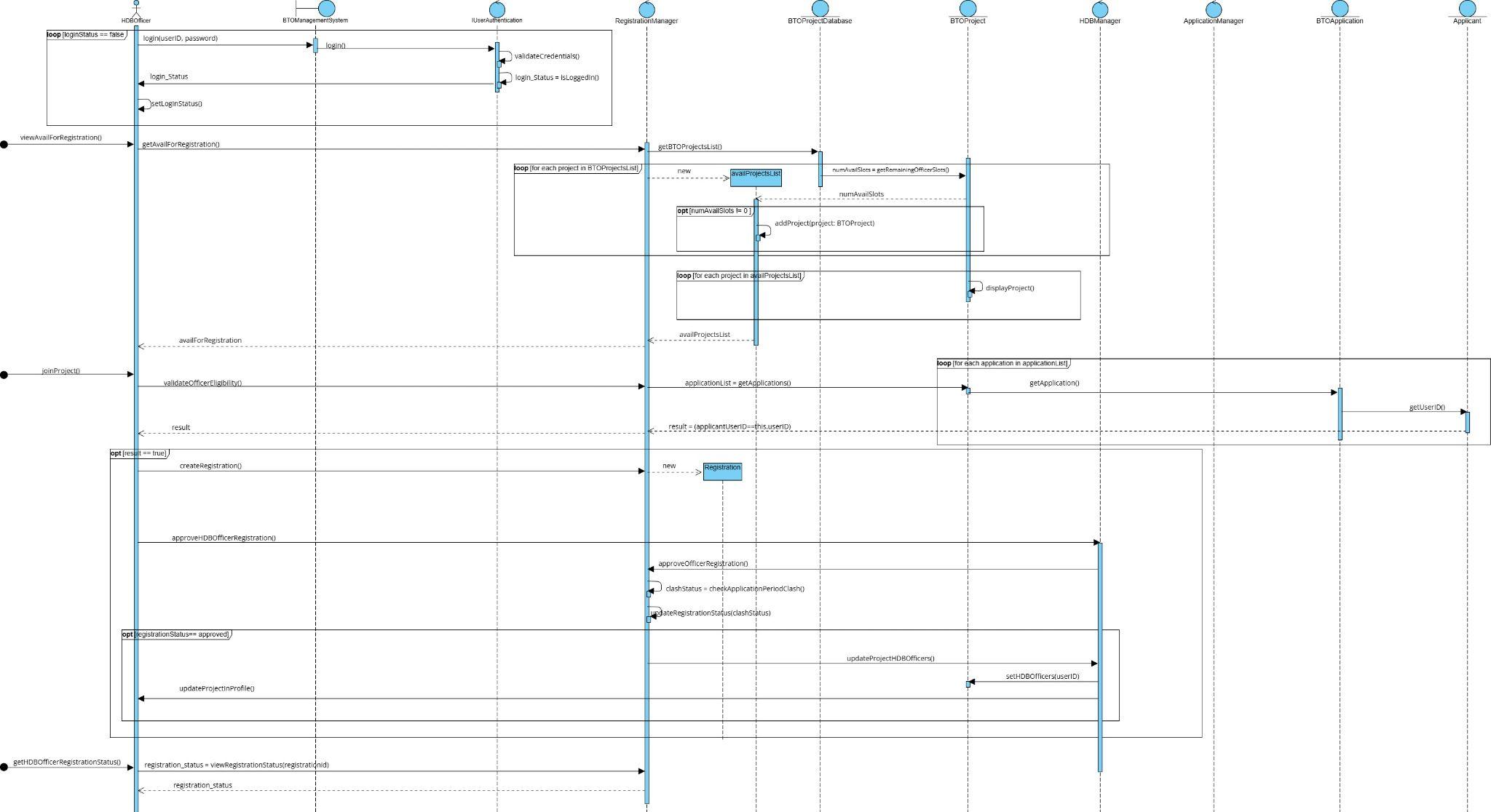


3.2 Sequence Diagrams (with Emphasis on Thinking Process)

We identified 3 critical use cases that demonstrate the system's complexity and core functionality as mentioned in section 2.1.

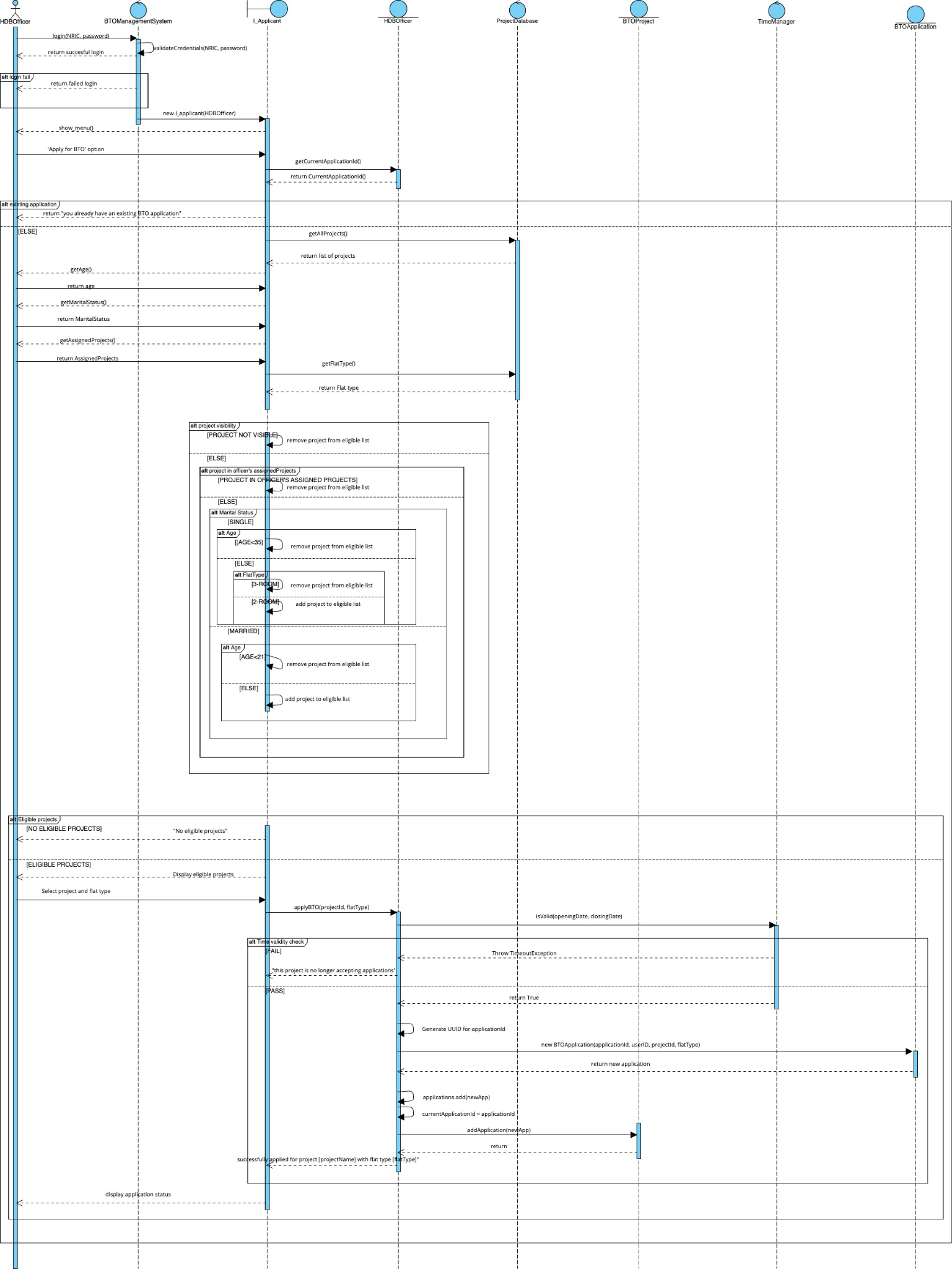
1**1. HDB Officer Registration Flow**

We chose this scenario because it demonstrates the interaction between different user types (Officer and Manager) and validates our role-based permission system. The registration process involves complex validation rules - checking for date conflicts, ensuring the officer isn't an applicant for the same project, and verifying available slots. This sequence helps validate that our system correctly implements these business constraints while showing how different objects collaborate to fulfil a critical administrative task.



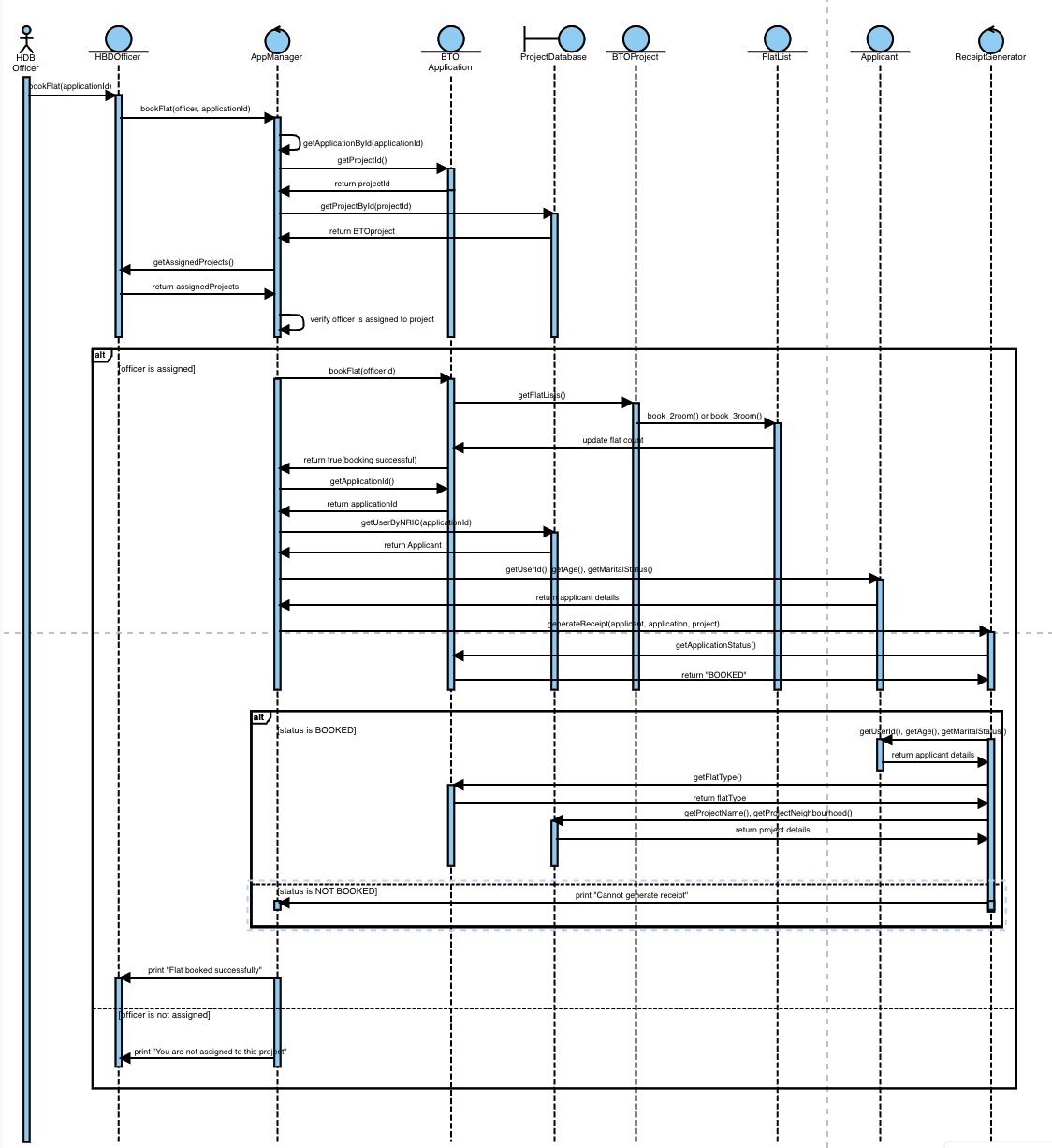
**2. HDB Officer Applying for a BTO Project**

This sequence diagram is valuable as it showcases our inheritance hierarchy, with HDB Officers inheriting from Applicants, sharing capabilities but with added constraints (e.g., officers cannot apply for projects they manage). It validates the inheritance model and business rules while illustrating cross-component collaboration between the user and application processing subsystems.



**3. Flat Booking Process**

We selected the flat booking process because it represents the culmination of the application lifecycle and exercises the inventory management aspects of our system. This diagram helps validate that our design properly handles resource allocation (flats), status transitions (from SUCCESSFUL to BOOKED), and role-based actions (only officers can book flats for applicants). It demonstrates how our object model maintains system integrity through encapsulation of critical operations.



3.3 Application of OOP Principles (SOLID)

**Single Responsibility Principle (SRP)**

The SRP states that a class should have one responsibility. RegistrationManager adheres to SRP by focusing solely on registration operations (e.g., createRegistration, getRegistrationsForOfficer) for HDB officers, excluding unrelated tasks like project management or authentication, handled by ProjectManager and BTOManagementSystem. Changes to registration logic, like adding deleteRegistration, only impact RegistrationManager, not ProjectDatabase or BTOManagementSystemApp, improving maintainability and testability.

**Open-Closed Principle (OCP)**

OCP states that classes should be open for extension but closed for modification. We defined separate interfaces for user roles to encapsulate their unique UI interactions. This allows us to add new roles or modify behaviour without changing the core User class or the login logic. This approach made the system highly extensible as when launching interfaces, we depend only on the I\_UserInterface abstraction, and role behaviour is encapsulated clearly

**Liskov Substitution Principle (LSP)**

LSP states that subclasses should be substitutable for their parent without breaking the program. HDBOfficer, Applicant and HDBManager classes extend from a common User superclass and override methods where needed, allowing them to be stored in a unified List<User> and authenticated uniformly. This enabled polymorphism in login and user management, keeping logic centralised.

**Interface Segregation Principle (ISP)**

ISP states that classes should not implement unneeded methods. We use focused interfaces like I\_projectManager, I\_officer\_EnquiryM, and I\_applicant\_EnquiryM. ProjectManager implements I\_projectManager for project tasks (e.g., createBTOProject), while OfficerEnquiryManager implements I\_officer\_EnquiryM for enquiry tasks (e.g., replyEnquiry). HDBOfficer only implements I\_officer\_EnquiryM, avoiding irrelevant methods. Adding features to I\_projectManager doesn’t affect I\_officer\_EnquiryM, enhancing modularity.

**Dependency Inversion Principle (DIP)**

DIP states that modules should depend on abstractions, not concrete classes. HDBOfficer and Applicant use interfaces like I\_officer\_EnquiryM and I\_applicant\_EnquiryM. In HDBOfficer, enquiryManager is an I\_officer\_EnquiryM, initialized with OfficerEnquiryManager. This allows swapping implementations (e.g., mocks for testing) without changing HDBOfficer. Similarly, ProjectManager depends on I\_projectManager, decoupling components and improving flexibility and testability.

**4. Implementation (Java)**

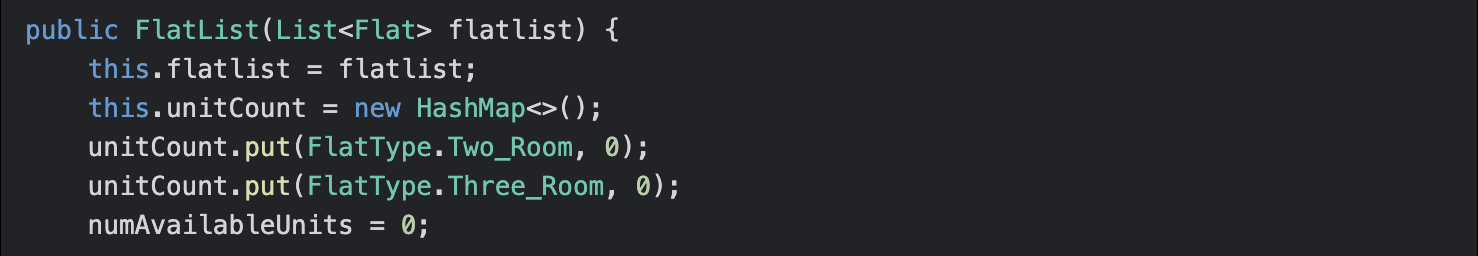
4.1 Tools Used:

* Java 17
* IDE: VSCode
* Version control: GitHub

4.2 Sample Code Snippets:

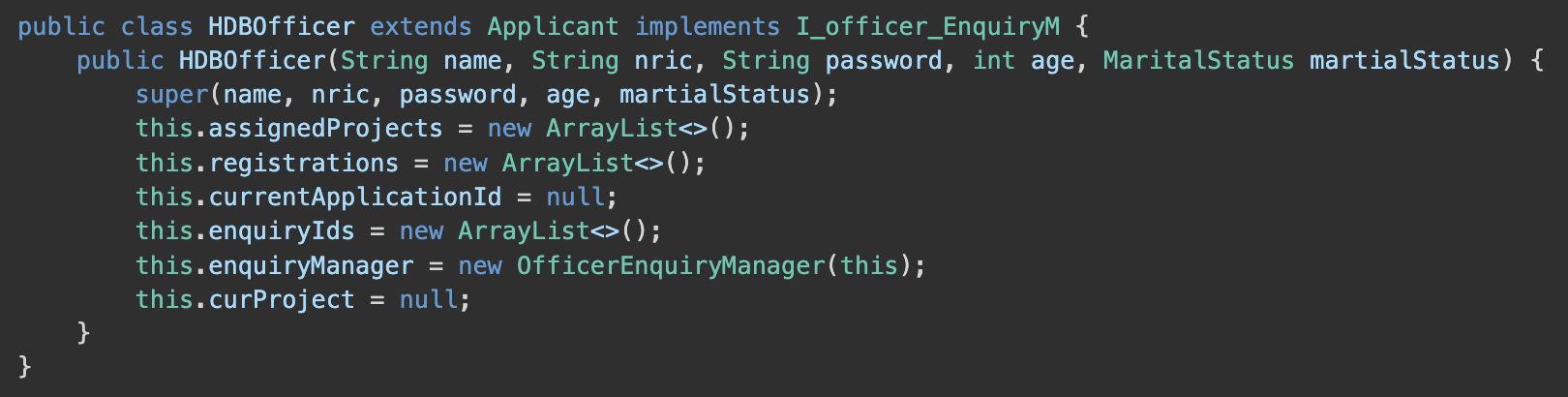
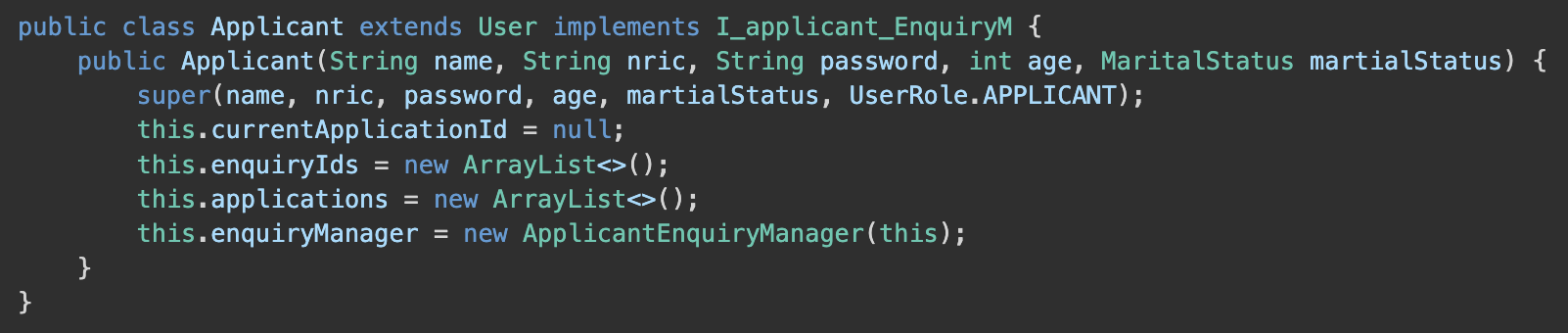
Show parts that demonstrate:

* Encapsulation

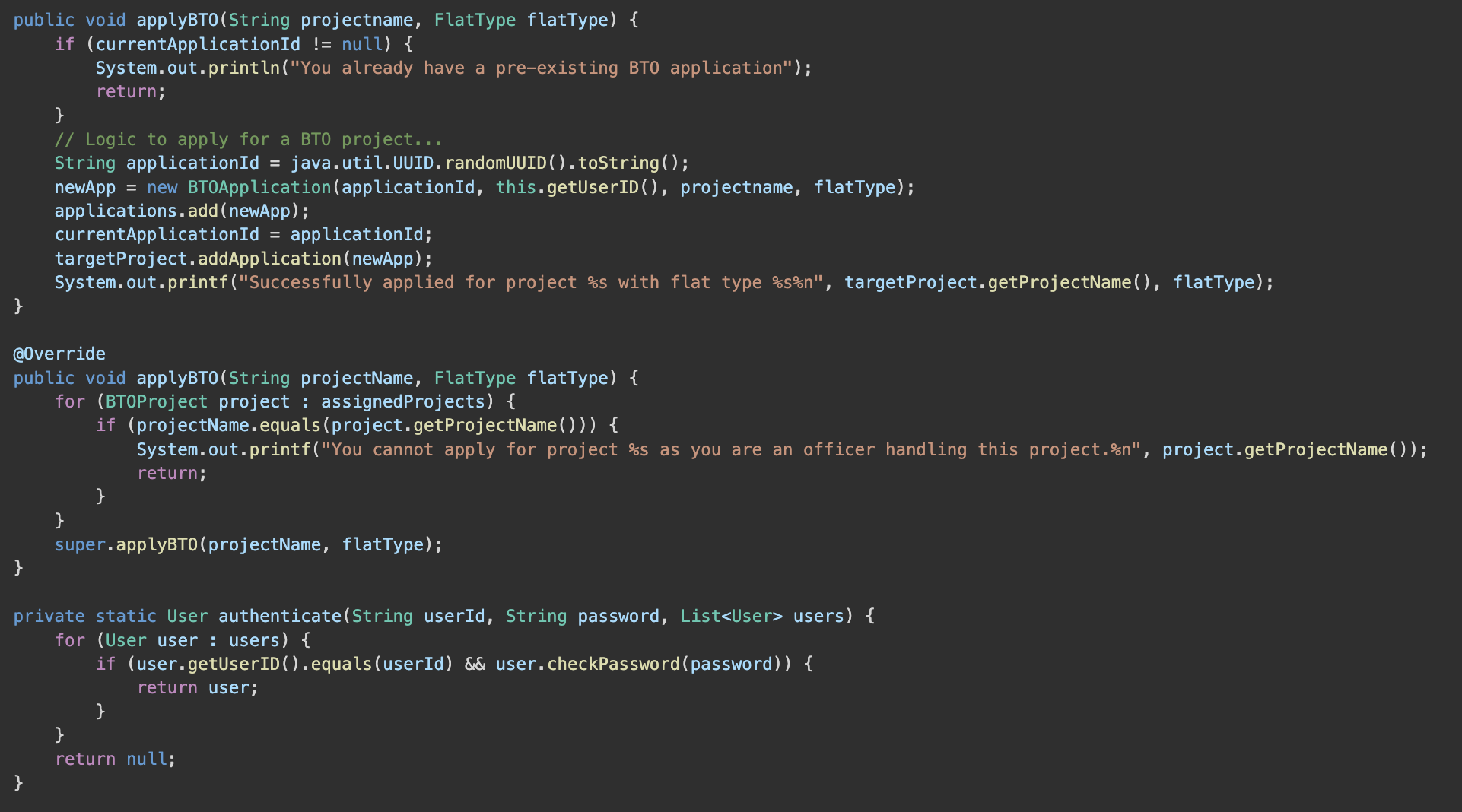


* Inheritance





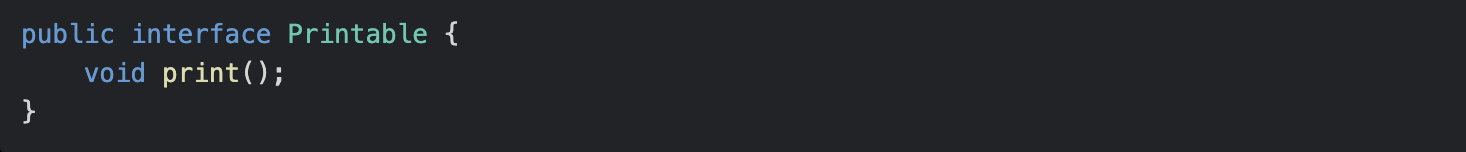
* Polymorphism

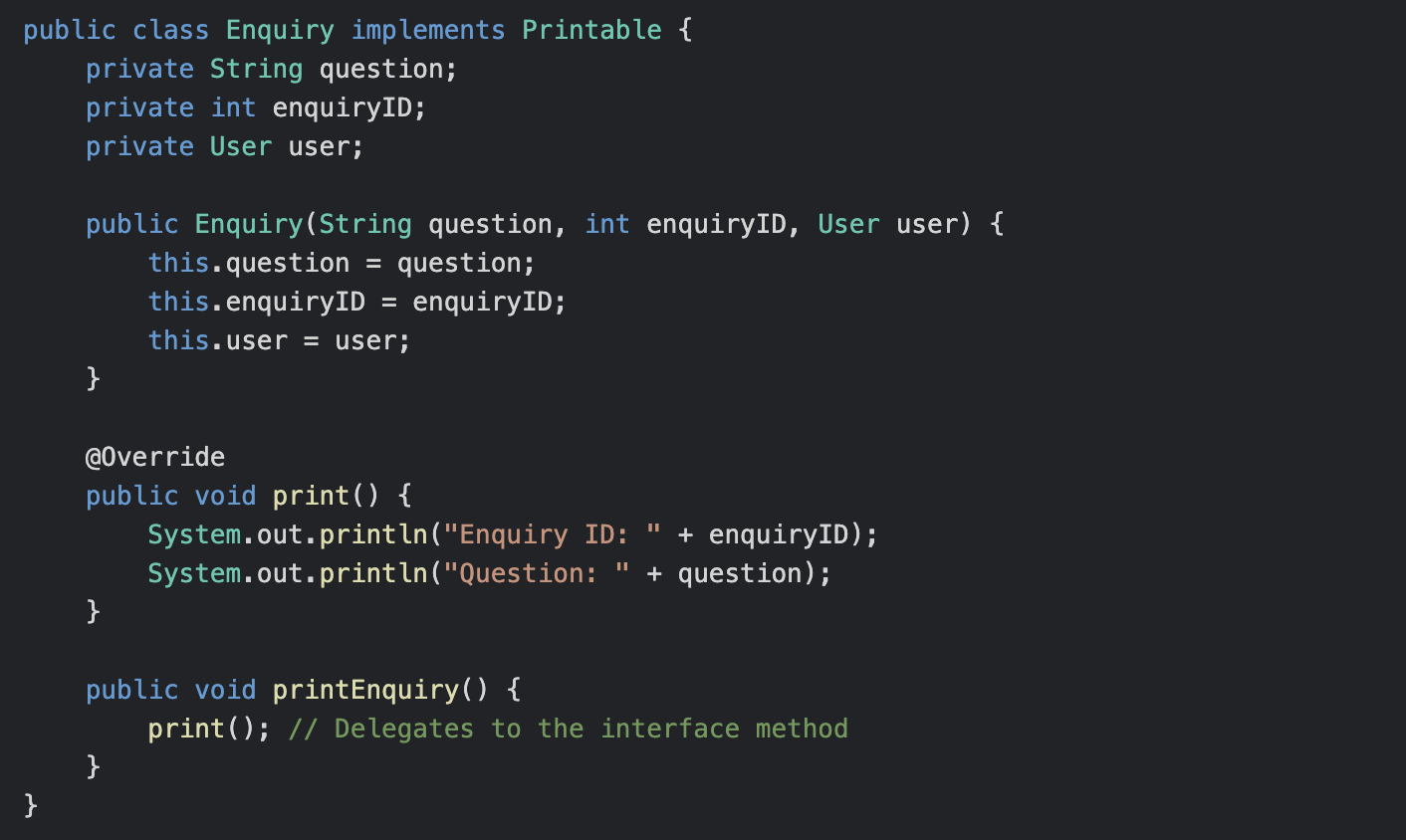


* Error handling



* Interface use





**Chapter 5: Testing**

5.1 Test Strategy

Black box testing (Equivalence class and defensive testing) and manual functional testing without the use of external tools will be used for this chapter. Since this system is mainly designed to be user-friendly and functional, we decided to focus on validating that the software meets requirements from a user's perspective by testing out the external interfaces rather than the internal code structure.

5.2 Test Case Tables

**Applicant**

| **Test Case** | **Description** | **Expected/Actual Behaviour** | **Test Case Passed** |
| --- | --- | --- | --- |
| Applicant Login  \*Applicable for Officer, Manager | Valid credentials:  NRIC - S1234567A  Password - password | Login successful | ✓ |
| Invalid credentials:  NRIC - S123A  Password - wrongpass | Login unsuccessful | ✓ |
| BTO Application  (Successful) | Format: (Age, Marital Status,  Project Name, Flat type, Existing BTO Application)  35, Single, , Acacia\_Breeze, 2, No  30, Married, Acacia\_Breeze, 2, No  37, Married, Acacia\_Breeze, 3, No | Successfully applied for project  Acacia\_Breeze with flat type | ✓ |
| BTO Application  (Unsuccessful) | Format: (Age, Marital Status,  Project Name, Flat type, Existing BTO Application)  35, Single, Acacia\_Breeze, 3, No | Singles are only eligible  to apply for 2-Room flats | ✓ |
| 25, Single, Acacia\_Breeze, 2, No  20, Married, Acacia\_Breeze, No | Ineligible Age | ✓ |
| 37, Married, Sea\_Estate, 3, No | Project not found | ✓ |
| 35, Single, Acacia\_Breeze, 2, Yes | Applicant has concurrent BTO application | ✓ |
| Add Enquiry (Successful) | Enquiry: How old is this flat? (Acacia\_Breeze) | Enquiry added successfully | ✓ |
| Add Enquiry (Unsuccessful) | Enquiry: How old is this flat? (Sea\_Estate) | Project not found | ✓ |
| Delete Enquiry (Successful) | Project Name: Acacia\_Breeze  Enquiry ID: 1 | Enquiry deleted successfully | ✓ |
| Delete Enquiry (Unsuccessful) | Project Name: Acacia\_Breeze  Enquiry ID: A1 | throw.InputMismatchException | ✓ |
| View Open Projects | No user input required | Available projects:  Project Name: Acacia\_Breeze | ✓ |
| Change Password  (Successful)  \*Applicable for Officer, Manager | password  newpassword  newpassword | Enter current password:  Enter new password:  Confirm new password:  Password successfully changed | ✓ |
| Change Password  (Unsuccessful)  \*Applicable for Officer, Manager | pass  password  newpassword  newpass | Incorrect current password.  Password change failed.  Passwords do not match.  Password change failed. | ✓ |

**Officer**

| **Test Case** | **Description** | **Expected/Actual Behaviour** | **Test Case Passed** |
| --- | --- | --- | --- |
| Applying for projects  (Successful) | Project Name: Acacia\_Breeze | Note: Officer “Name” will be assigned to  project Acacia\_Breeze after loading.  Registration request submitted for project: Acacia\_Breeze  Your request is pending approval from the HDB Manager. | ✓ |
| Applying for projects  (Unsuccessful) | Project Name: Sea\_Estate | Project not found. | ✓ |

**Manager**

| **Test Case** | **Description** | **Expected/Actual Behaviour** | **Test Case Passed** |
| --- | --- | --- | --- |
| Creating new BTO Project  (Successful) | Format:  (App. Opening date, Closing Date, Flat Type,  Visibility)  2025-07-23, 2027-07-23, Two\_Room, Three\_Room,  True  2025-07-23, 2027-07-23, Two\_Room, Three\_Room,  False | Project created | ✓ |
| Creating new BTO Project  (Unsuccessful) | 23-02-2025, 23-02-2025,  Two\_Room, Three\_Room,  True  2025-07-23, 2027-07-23  2, 3,  True | Invalid date format.  Please use YYYY-MM-DD format.  Invalid flat type: 2. Skipping this entry.  Valid flat types are: [Two\_Room, Three\_Room]  Invalid flat type: 3. Skipping this entry.  Valid flat types are: [Two\_Room, Three\_Room].  Project creation failed. | ✓ |
| Approving/  Rejecting Registrations | (Correct ID)  (Correct ID)  (Wrong ID) | Registration Approved  Registration Deleted  Registration not found: 1 | ✓ |
| Getting list of managed projects | 2 (Correct Project Number)  (Correct ID)  (Wrong ID) | Registration Approved  Registration Deleted  Registration not found: 1 | ✓ |

**7. Reflection & Challenges**

Our team (FCS6-G2) collaborated well by dividing work based on strengths—some focused on coding, others on documentation. Early diagram design helped clarify roles, though starting even earlier could’ve reduced stress during finals. Technically, we could’ve used more design patterns like Observer and Command to improve modularity and usability. Interfaces and manager classes helped us apply SOLID principles, making the system easier to extend and maintain. Incremental development and component testing proved effective, reinforcing that good design is about thoughtful modelling, not just patterns.

* **GitHub Repository**:<https://github.com/ivan0065/SC2002>