NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

**SC2002 : OBJECT ORIENTED DESIGN AND PROGRAMMING**

**AY24/25 SEMESTER 2 GROUP 3 ASSIGNMENT**

## 1. APPENDIX B:

### 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 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 | Course | Lab Group | Signature / Date |
|---|---|---|---|
| **Xu Wenbo** | **SC2002** | **FCSJ** | Xu Wenbo 23/4/2025 |
| **Aishik Kumar Sarkar** | **SC2002** | **FCSJ** | 23/04/2025 |
| **Yang Quan Kang, Sean** | **SC2002** | **FCSJ** | 23/04/2025 |
| **Jolin Chua Zhi Lin** | **SC2002** | **FCSJ** | 24/4/2025 |
| **Chen Ling Zheng** | **SC2002** | **FCSJ** | 24/4/2025 |

## 2. Design Considerations

### 2.1. Flow of project

The system starts at a central menu where users select their role and log in. Each role then sees a customized menu with functions relevant to their responsibilities. Applicants can view and apply for projects, manage their applications and enquiries, and book flats, interacting with classes like Project, Application, and Enquiry. HDB Officers have options to apply for officer roles, manage their officer applications, view assigned projects, respond to enquiries, accept bookings, and generate receipts. HDB Managers oversee the entire process, with abilities to manage projects, monitor all applications and officer registrations, generate reports, and handle enquiries. All actions are backed by CSV file storage, with loader classes managing data retrieval and saving.

### 2.2. Ambiguity

Our team had to decide how to store and manage system data like applications, enquiries, officer applications, and project details. With no requirement for a database management system, we opted for CSV files because they are simple to use, easy to edit both manually and programmatically, and widely supported by spreadsheet and text editors.

We also needed to determine how users would interact with the system and access its functions. To address this, we created separate menu classes for each user role—Applicant, HDB Officer, and HDB Manager—each offering functions relevant to that role. This role-based menu structure streamlines the user experience, reduces confusion, and enhances both maintainability and security by clearly defining permissions and workflows for each group.

### 2.3. Concepts

**Encapsulation**

Each class (such as Application, Enquiry, Project, and OfficerApplication) encapsulates its data and behavior, exposing only necessary methods for interacting with its internal state. For instance, fields are private and accessed or modified via getters and setters, ensuring controlled access and modification of data.

**Inheritance**

The Applicant class extends from a base User class, which allows reuse of common attributes like name, NRIC, and age.

**Polymorphism**

The Menu interface defines a run() method, which allows different types of menus to implement their own behavior while using a common method signature.
This is useful when displaying different menus based on user roles or system states.

**Abstraction**

The project hides complexity by abstracting away specific details. For instance:Applicants don't need to know how project data is saved—they just call apply() or bookFlat(). The use of enums (UserGroup, Visibility, etc.) abstracts internal logic into meaningful types, improving clarity.

**Additional Features**

Realtime Database – The project's functions access each .csv file in real-time, taking the current statuses. The CSV Loaders has on the spot modification of each .csv file for all its functionality. This improves the effectiveness of the BTO Project as users can instantaneously provide modifications and receive the results.

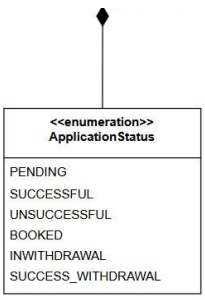
## 2.4 SOLID Principles

**Single Responsibility Principle (SRP):**

Classes such as Application, Project, OfficerApplication, and Enquiry each encapsulate a specific concept with their own data and related behaviors, adhering to SRP. For example, Enquiry handles enquiry data operations, while Project manages project attributes and logic. Menus are abstracted such that each menu is specific to either an Applicant, Officer or a Manager only.

**Open/Closed Principle (OCP):**

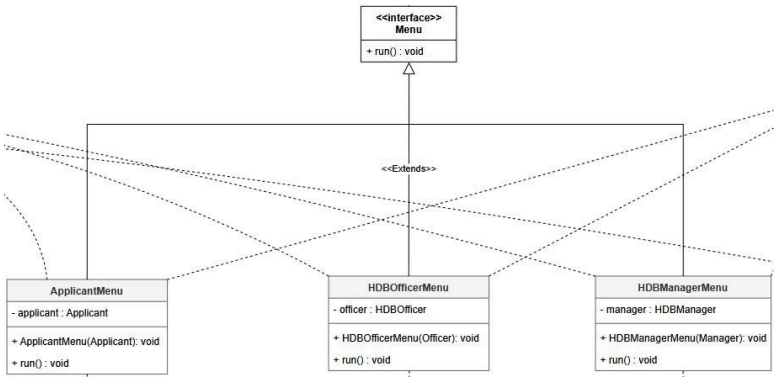
The use of enums for statuses (e.g., ApplicationStatus, OfficerApplicationStatus, Visibility) makes classes open for extension (by adding/removing statuses in the future)



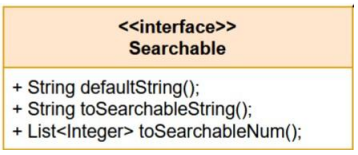
without modifying existing code. Each CSV loader also is specific to its own .csv file, hence future databases can be added without modifying the existing CSV loader classes.

**Liskov Substitution Principle (LSP):**

User roles such as Applicant, HDBOfficer, and HDBManager inherit from a common User base class and are handled polymorphically in the StartMenu's authentication logic. This design means any subclass of User can be used wherever a User is expected, supporting code reuse and flexibility. The use of interfaces like Searchable allows the system to be extended without modifying existing code.




**Interface Segregation Principle (ISP):**

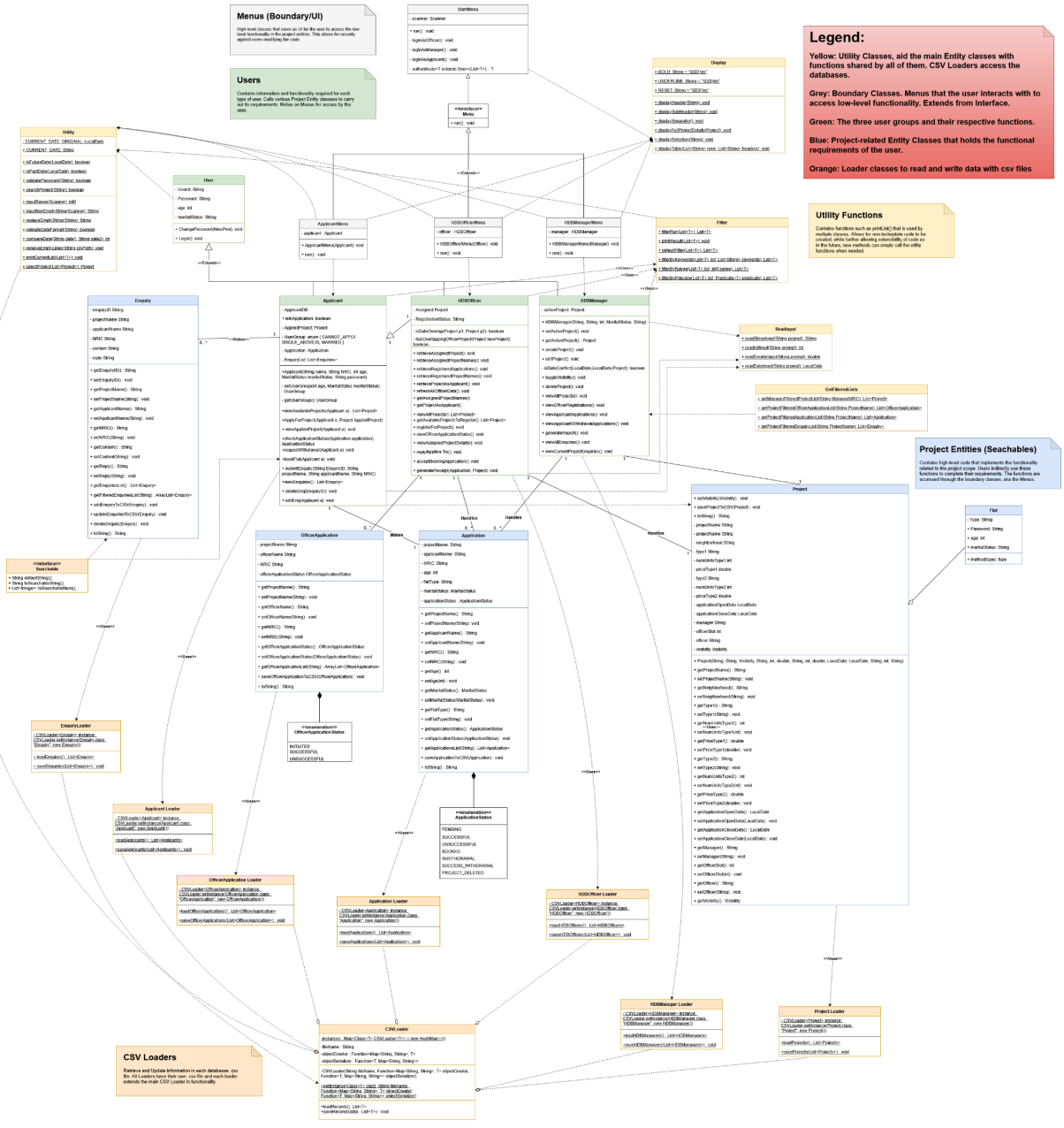



Interfaces, such as Searchable, are used that provide a contact for search-related methods. Every interface is specific and abstracted. This helps keep interfaces focused and relevant to implementing classes, supporting ISP.

**Dependency Inversion Principle (DIP):**

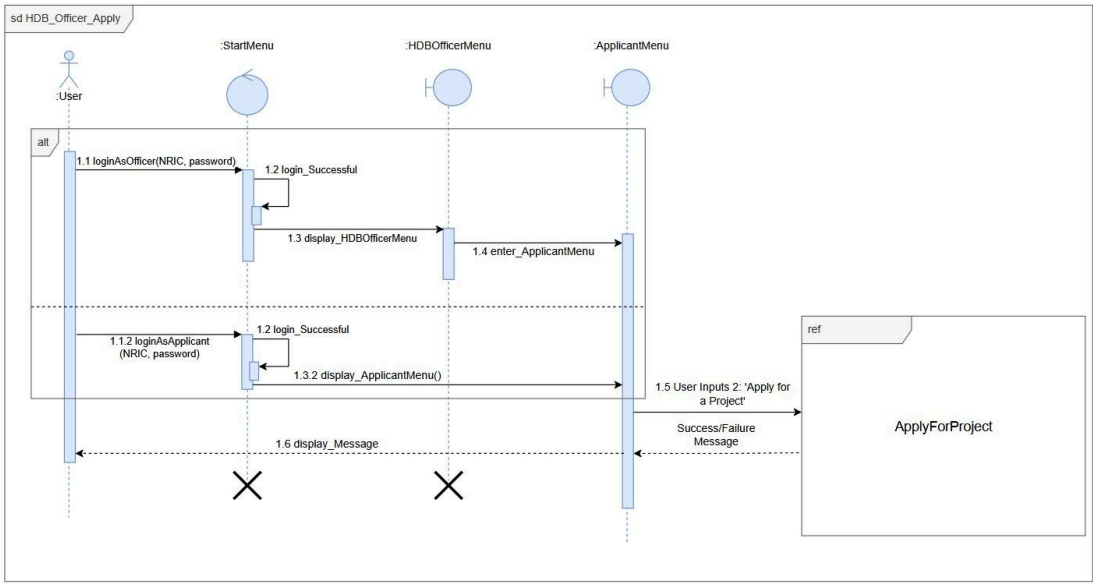
Classes depend on loader classes (e.g., ApplicationLoader, EnquiryLoader) for data persistence. Each high-level class depends on their corresponding user class rather than the low-level project entity classes.

# 3. UML Class Diagram




# 4. UML Sequence Diagrams

## 4.1. HDB_Officer_Apply Sequence Diagram

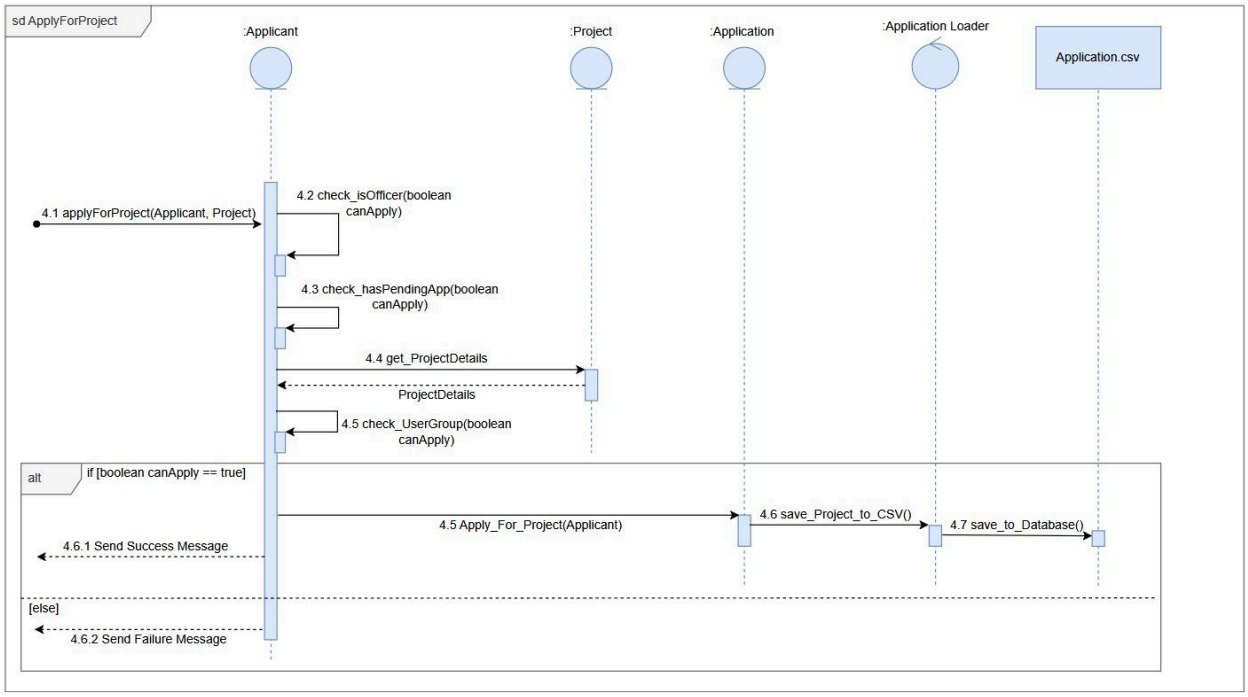



The HDB Officer Apply sequence diagram shows the flow where an Officer logs in and applies for a Project as an Officer, demonstrating the system's authentication process, hierarchical menu navigation, and collaboration between interface and backend logic.

There are two ways an Officer can log in and apply as an applicant, as illustrated by the alternate flows box. The Officer may LoginAsOfficer and then navigate to the ApplicantMenu, or the Officer can LoginAsApplicant and access the ApplicantMenu directly. This validates secure access, role-based functionality, allowing the user to access their corresponding menus depending on their role and access level, illustrating how core system components interact to complete a critical operation.

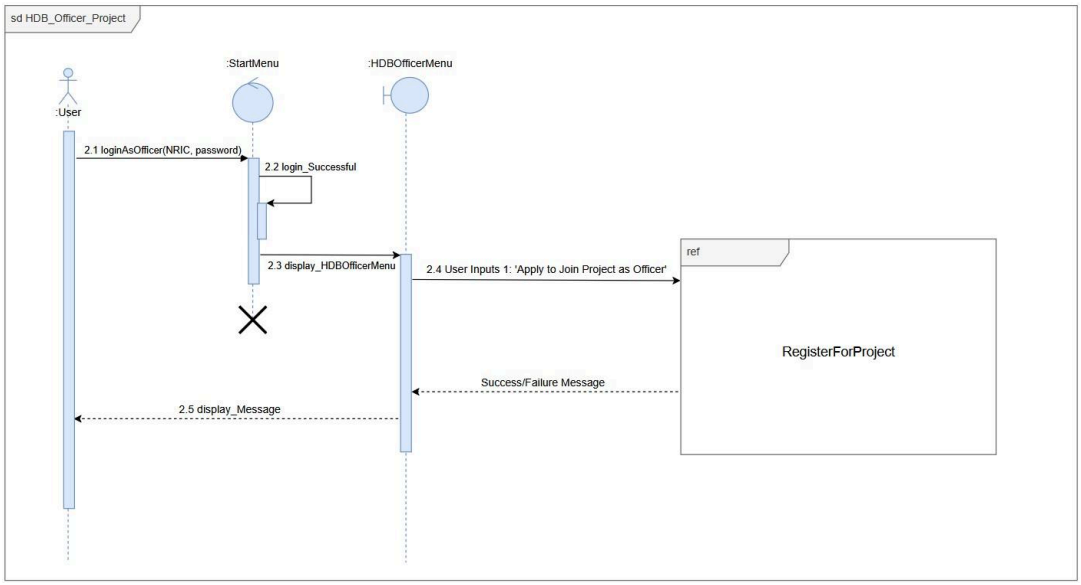
The flow ends with a reference to 4.2 ApplyForProject as continued below.

## 4.2. ApplyForProject Sequence Diagram




The ApplyForProject sequence diagram shows how the user is validated through multiple checkpoints (isOfficer, hasPendingApp, UserGroup) before they are able to apply as an Applicant to a Project. If the user is an Officer to the Project, has a pending Application, or is not in the correct user group, then the boolean canApply is set to false and they are unable to apply. Otherwise, canApply is set to true and a new application is created. The application is then saved to a csv file for other users like Managers to work on. This highlights how the system maintains data integrity through controlled operations and validates critical system capabilities in enforcing policies while demonstrating how different components collaborate to implement role-based access control.

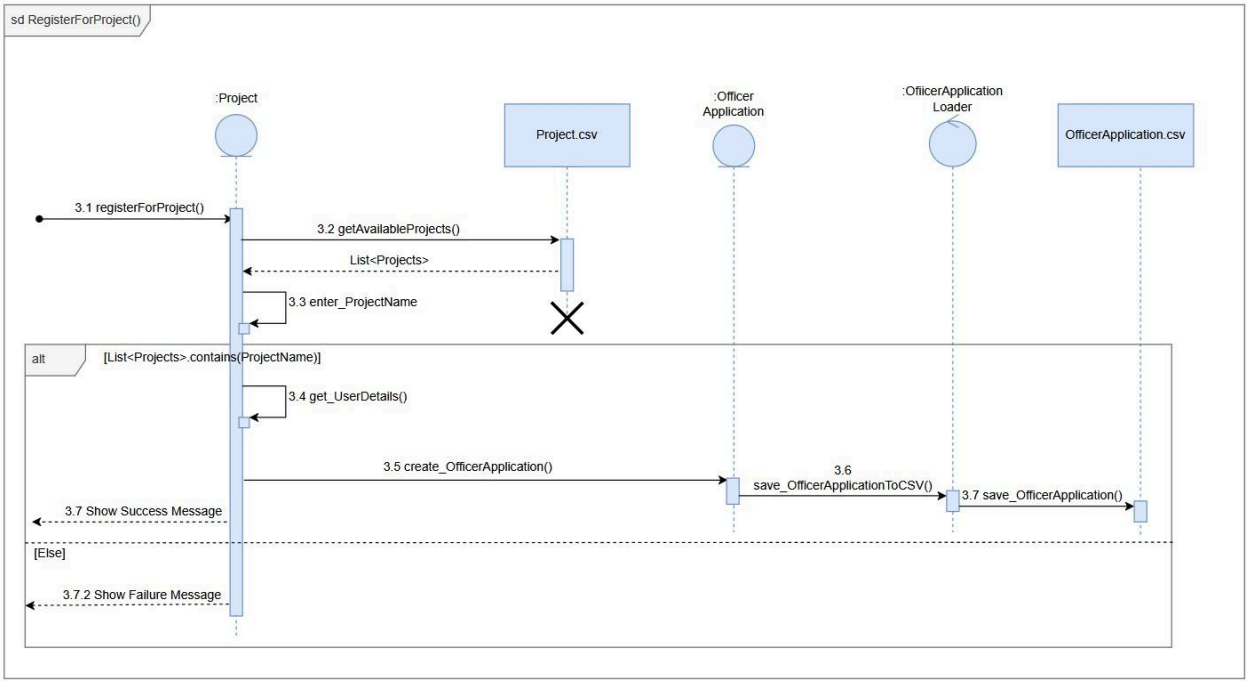
## 4.3. HDB_Officer_Project Sequence Diagram




The HDB_Officer_Project sequence diagram shows the workflow of an officer applying to join a project. Much with the applicant, the user must first log in with their credentials and select to login as an officer. Their credentials are then verified before they are brought to the HDBOfficer Menu, a boundary class that shows the functions the user can access. This demonstrates how the system ensures only authorized users can perform sensitive operations, maintaining system integrity and security.

The user hence accesses the system UI and must select amongst a few options. Selecting option 1, ‘Apply to Join Project as Officer’ then brings them to the process of registering for a project as an officer, which is shown in a separate sequence diagram 4.4. RegisterForProject.

## 4.4. RegisterForProject Sequence Diagram




The RegisterForProject sequence diagram is a helper function for 4.3 ApplyForProject, handling the process of registering an Officer to a Project through user and project validation and persistence workflow. The function creates a new entry in the OfficerApplication.csv file, showcasing how our system allows for data persistence across multiple users and multiple databases(Project.csv and OfficerApplication.csv), and how the system retrieves available projects, validates user selections, and creates new application records. The diagram illustrates how the Project class checks if the project exists and that project visibility is on before allowing an Officer to apply. Finally, the system returns a message that will be displayed to the user as shown in 4.3.

## 5 Testing

**Test Case 1: Incorrect password – Passed**

Expected Behaviour: System should deny access and alert the user to correct password.

Failure Indicator: User logs in successfully with a wrong password.

```
Enter NRIC: evoerg
Enter Password: fh35q
Invalid NRIC or Password. Try again.
```

Fig. 5.1. Test-Case 1

**Test Case 2: Applying for Project with Application Processing – Passed**

Expected Behaviour: System should deny access and alert the user that an application is already in process.

Failure Indicator: New application is created.

```
Enter Your Choice: 2
================================================================================
[SELECTION] Apply for a Project
================================================================================
You already have a application in process. You cannot apply to another application.
```

Fig. 5.2. Test-Case 2

**Test Case 3: Changing Password – Passed**

Expected Behaviour: System updates password, prompt re-login and allows login with new credentials.

Failure Indicator: System does not update the password or denies access with the new password.

```
[SELECTION] Change Password
============================================================
Enter 1 to proceed changing password or 0 to return: 1
Enter your new Password: 12345678
Confirm your new Password: 12345678
Confirm to change? (Y/n)
y
Password Successfully Changed!
```

Fig. 5.3. Test-Case 3

**Test Case 4: Submitting Enquiry – Passed**

Expected Behaviour: Enquiries can be successfully submitted, displayed, modified, and removed.

Failure Indicator: Enquiries cannot be submitted, edited, or deleted; or do not display correctly.

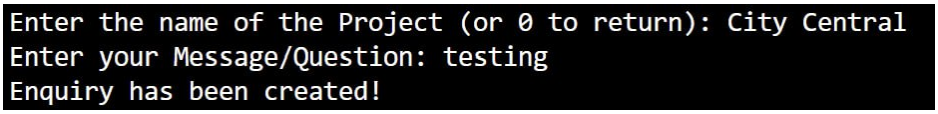



Fig. 5.4. Test-Case 4

**Test Case 5: Booking a Flat as Applicant when Application Unsuccessful – Passed**

Expected Behaviour: System should deny access and display an error message to the user that their application has not been approved and hence they cannot book a flat.

Failure Indicator: User is prompted to continue and flat is booked.


Fig. 5.5. Test-Case 5

**Test Case 6: Requesting Withdrawal – Passed**

Expected Behaviour: Application can be successfully withdrawn and labelled as WITHDRAWN in the database.

Failure Indicator: Application cannot be withdrawn or is not modified in the database.

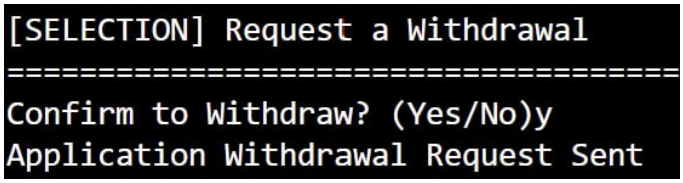



Fig. 5.6. Test-Case 6

**Test Case 7: Entering invalid choice – Passed**

Expected Behaviour: System prompts user to enter a valid choice.

Failure Indicator: System crashes and stops looping.

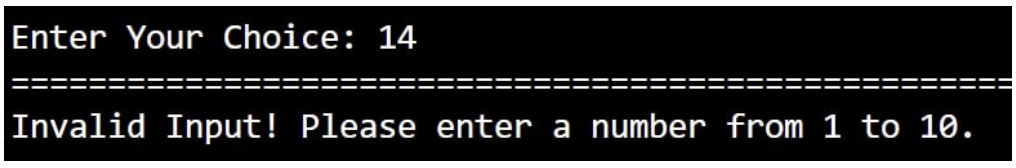



Fig. 5.7. Test-Case 7

**Test Case 8: Officer applying for unavailable project – Passed**

Expected Behaviour: System should not grant access to user and should display an error message that the project does not exist.

Failure Indicator: System crashes or new incorrect project is created.

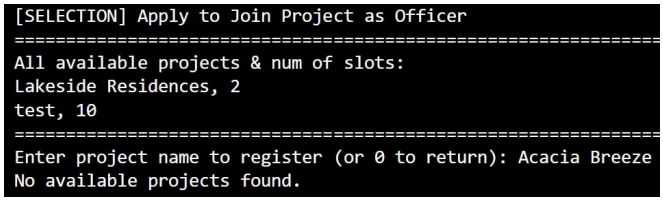



Fig. 5.8. Test-Case 8

**Test Case 9: Officer viewing application status – Passed**

Expected Behaviour: Officers can view their application status to become an officer.

Failure Indicator: Application status does not display correctly.

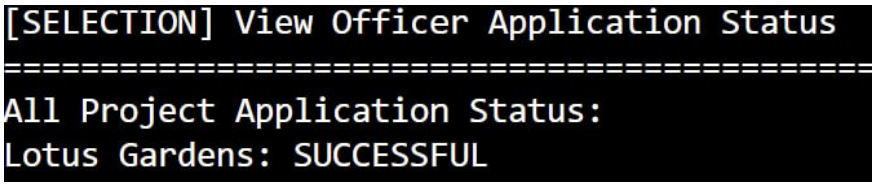



Fig. 5.9. Test-Case 9

**Test Case 10: Creating a project that already exists – Passed**

Expected Behaviour: System should not grant access to the user for the project function and displays an error message that the project already exists.

Failure Indicator: Duplicate project is created.

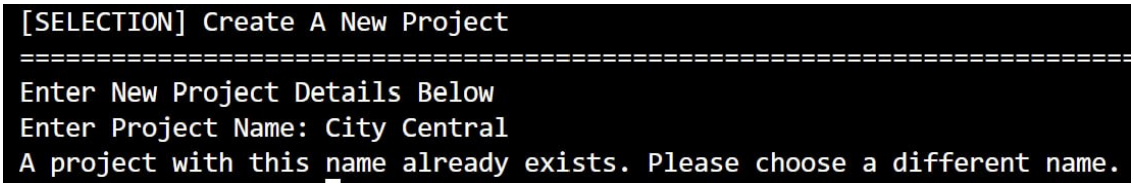



Fig. 5.10. Test-Case 10

# 6 Reflections

## 6.1. What went well

Our project clearly applies object-oriented principles with well-structured classes like Applicant and Project that encapsulate related data and behavior. Inheritance is used effectively, with Applicant extending User to promote code reuse and a logical hierarchy. Enums such as UserGroup and ApplicationStatus improve readability and structure. Responsibilities are well-separated—for example, Applicant handles application logic while Project manages

housing data. Eligibility logic is encapsulated in the setUserGroup method, making rule changes straightforward. This modular design also supports easy extensibility, allowing new roles to be added with minimal code changes.

### 6.1.2. What could be improved

a. Improve Input Validation and Error Handling:
Input handling in menus (such as in ApplicantMenu and StartMenu) is basic and could be made more robust by adding comprehensive validation, clearer error messages, and user guidance to prevent invalid operations and enhance reliability.

b. Optimize Search and Filtering:
The current search and filter features are functional but could be expanded to support more advanced queries, multi-criteria filtering, and better performance, especially as the dataset grows.

### 6.1.3. Lessons learned about OODP

Object-oriented design helped manage system complexity by dividing it into focused classes like Applicant, Project, and Application, each with clear responsibilities. Encapsulation made the system flexible and easier to modify, while inheritance and polymorphism enabled roles like Officer and Manager to extend the User class. Applying the Single Responsibility Principle highlighted the need to separate UI, logic, and data access for better maintainability. Enums such as UserGroup and Visibility also improved clarity and reduced bugs.