


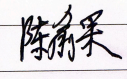



SC/CE/CZ2002 Object-Oriented Design & Programming Assignment Report

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
Amanda Rae Josephine	DSAI	FDAE	 24 April 2025
Chen Yu Guo	DSAI	FDAE	
Lee Teng Yee Sherilyn	CSC	FDAE	
Li Sihan	DSAI	FDAE	 24 April 2025
Tan Kai Hooi	DSAI	FDAE	

Important notes:

1. Name must EXACTLY MATCH the one printed on your Matriculation Card.
2. Student Code of Academic Conduct includes the latest guidelines on usage of Generative AI and any other guidelines as released by NTU.

CHAPTER 1: Design Considerations

1.1. Understanding the Problem and Requirements

To identify the main problem domain, our team began by analyzing the system's core purpose: to manage Build-To-Order (BTO) applications and administrative processes for different user types, namely Applicants, HDB Officers, and HDB Managers. By examining user roles and their corresponding functionalities, we recognized that the key domain revolves around housing application workflows, project visibility and management, flat allocation, and user enquiries. This helped us shape the primary entities and relationships in our object-oriented design.

The explicit requirements were clearly outlined in the assignment brief. These included features such as user login with NRIC format validation, role-specific permissions, flat booking mechanisms, enquiry handling, and project management (create, edit, toggle visibility, etc.). Each role was associated with strict eligibility criteria and workflow restrictions, which we carefully translated into class behaviours and control flows in our system.

In addition to these, our team inferred several implicit expectations. For example, we assumed the system must validate all user inputs and correctly handle any errors present in user inputs. We also inferred that the system should support system-like behavior to distinguish logged-in users, track their interactions, and save the data. Additionally, the mention of filters and sorted listings implied a need for dynamic filtering and data persistence during user navigation.

While most requirements were clearly defined, some aspects remained ambiguous. For instance, it was not specified if HDB Manager can generate reports for pending booking. Therefore, we interpreted this to mean that reports are only generated for successful and unsuccessful booking. In handling other uncertainties, we made reasoned assumptions based on real-world logic and ensured that our decisions remained consistent with the defined roles and intended functionality of the system.

Overall, our team's collaborative analysis and systematic breakdowns of use cases and constraints allowed us to construct a model of the system and align our implementation closely with both the given and implied requirements.

1.2. Deciding on Features and Scope

To determine the features to be implemented, our team first conducted a thorough analysis of the assignment brief and test cases. We compiled a comprehensive list of all functional requirements by identifying user roles and the actions associated with each. This allowed us to map out system capabilities and align them with real-world workflows.

We categorized the features based on their importance to the system's core functionality, the feasibility of implementation, and our project timeline:

- **Core Features (High priority):** These included all features explicitly required in the assignment brief and necessary to demonstrate the full range of role-based functionalities, such as project application, booking, registration, and enquiry handling.
- **Optional or Bonus Features (Lower priority):** While not excluded, features like password change, deleting enquiries, persistent filter settings, and saving data were initially treated as lower priority due to their less critical role in demonstrating the main system functions. These were implemented only after the core features were completed and tested.
- **Excluded Features:** We did not exclude any features from our implementation. All the required functionalities outlined in the project documentation were addressed in our system.

1.3. Additional Feature

Beyond the required features, we implemented a `saveData` method in all our database classes to ensure that any changes made during runtime are persistently stored. This method writes the updated data back into their respective CSV files. If a corresponding CSV file does not exist, the method will automatically generate a new one.

CHAPTER 2: System Architecture and Structural Planning

2.1 Planning the System Structure

After finalizing our features based on the BTO Management System requirements, we began designing the system's structure using an object-oriented approach. This involved identifying the key actors (Applicant, HDB Officer, HDB Manager) and modeling their responsibilities through logical components.

To promote reusability and flexibility in our system's user interaction layer, we created `BaseInterface`, which acts as a foundational template for the three role-specific interfaces:

ApplicantInterface, OfficerInterface, and ManagerInterface. This design supports polymorphism, allowing each user role to have its own menu flow while sharing common functionalities such as project filtering.

Our team grouped related features for each role into distinct controllers, boundaries, and entities. Controllers (e.g., ApplicantService, ManagerService, ProjectService) act as intermediaries between the user interface and the data layer. They handle input from the user interface, apply business logic, interact with the database and return the processed results back to the UI. Meanwhile, boundaries (e.g., ApplicantInterface, ManagerInterface, OfficerInterface) handle user interface input/output in the CLI environment. These classes interact with the service layer to collect input data, display results, and drive system flow based on user commands. Lastly, entities (e.g., Project, FlatType, Enquiry, Registration) encapsulate core attributes and behaviors. For instance, Application holds status for housing applications and links to Project and Applicant, and FlatType maintains availability logic and price checking mechanisms.

2.2 Reflection on Design Trade-offs

During the planning phase, we encountered several design trade-offs and made decisions to ensure our system was maintainable, modular, and aligned with OOP principles. Firstly, for user roles, we opted for an inheritance hierarchy where Applicant and HDBManager inherit from a base User class, to reflect "is-a" relationships. Although composition might have offered more flexibility, inheritance provided simpler role management and behaviour reuse. Next, we debated whether to use a single UserController with conditional role logic or multiple role-specific controllers but we ultimately chose multiple controllers (ApplicantService, OfficerService, ManagerService) to respect the Single Responsibility Principle. This made the logic easier to test and evolve independently. We chose to separate Project, FlatType, and Application entities rather than nesting them within one another. This improved clarity and enabled features like project filtering and flat availability tracking to be independently managed. While a single generic database class could have reduced code duplication, we opted for specialized database classes (e.g., ProjectDatabase, UserDatabase) to provide clearer APIs and encapsulate data-specific logic, such as parsing project details from CSV.

CHAPTER 3: Object-Oriented Design

3.1 Class Diagram

Based on our design thinking outlined in chapter two, we constructed a class diagram that represents the structure of our BTO Management System using the three-layered architectural pattern. The diagram is vertically layered: the model classes reside at the top to present core domain entities. These encapsulate data and behaviours that define the business logic of our system. The middle layer consists of controller classes which act as intermediaries. These classes apply system logic, interact with the model layer, and coordinate responses to user requests. The bottom layer consists of boundary classes which interact directly with the user. These classes extend a common BaseInterface to share reusable methods.

Our class diagram depicts all the relationships between the classes, such as inheritance, implementation, association, aggregation, composition, and dependency. In designing our class diagram, we prioritised modularity, maintainability, and adherence to OOP principles.

We also have a simplified class diagram to better illustrate the system layered architecture, namely the boundary, control, and entity classes.

3.2 Sequence Diagram

Our sequence diagrams illustrate two key scenarios: (1) an HDB Officer applying for a project, and (2) an HDB Officer registering to be an officer of a project. Each scenario was divided into two diagrams to reflect distinct responsibilities. For the first, one diagram captures the officer submitting an application with necessary validations, while the second shows the HDB Manager reviewing and approving or rejecting the application. Similarly, the second scenario is split into the officer's registration submission and the manager's subsequent approval process. This separation ensures clarity by isolating role-specific interactions and system responses.

3.3 Application of SOLID principles

1. Single Responsibility Principle

SRP states that a class should have one and only one reason to change, meaning it should have only one responsibility. Our codebase separates concerns into distinct layers: (1) Interface for user interaction, (2) Service for business logic, (3) Entity for data models, and (4) Database for data loading and persistence. This clear division ensures that each class has a focused responsibility, making the system easier to maintain and scale.

2. Open/Closed Principle

OCP states that software entities should be open for extension, but closed for modification. The use of InterfaceFactory in our system allows us to introduce new user roles without modifying existing logic. We can simply extend the factory to return a different interface implementation based on the user type, while keeping existing ApplicantInterface, OfficerInterface, and ManagerInterface classes untouched. Additionally, we also created an interface BaseInterface from which all the interfaces inherit. By doing so, we can introduce new role-specific interfaces by extending BaseInterface without modifying it.

3. Liskov Substitution Principle

LSP states that subtypes should be substitutable for their base types without affecting correctness. In our system, Applicant, HDBOfficer and HDBManager all extend the User class. This allows polymorphism through the system. For example, in the BaseInterface constructor, we pass in a User object, and it works whether the actual object is Applicant, HDBOfficer, or HDBManager. This means we can use any subclass of User wherever User is expected, without breaking the logic.

4. Interface Segregation Principle

ISP states that clients should not be forced to depend on methods they do not use. Instead of having one large service interface for all user roles, our system defines specific, role-focused interfaces like IApplicantService, IOfficerService, and IManagerService. Each role class only depends on the interface relevant to its needs. This keeps dependencies minimal and focused, making the system cleaner, more modular, and easier to maintain.

5. Dependency Inversion Principle

DIP states that high-level modules should not depend on low-level modules, and both should depend on abstractions. For example, in ApplicantInterface, we depend on the abstraction IApplicantService instead of the concrete ApplicantService class. The actual implementation (ApplicantService) is injected via the constructor. This means ApplicantInterface (high-level) does not rely on the service directly. Both depend on the interface, allowing the implementation to change without affecting the interface logic.

```

public class ApplicantInterface extends BaseInterface {
    //set final to maintain this thread
    // Declaration depends on abstraction
    private final IApplicantService applicantService;
    private final IProjectService projectService;

    public ApplicantInterface(User currentUser) {
        super(currentUser); //in this case, for currentUser: reference - User; object - roles
        // Initialization uses a concrete class, but through the interface
        this.applicantService = new ApplicantService((Applicant) currentUser);
        this.projectService = new ProjectService();
        super.setProjectService(this.projectService);
    }
}

```

Furthermore, our Main class depends on the abstraction BaseInterface rather than specific implementations like Applicant Interface or OfficerInterface. We used InterfaceFactory.getInterface(currentUser) to dynamically return the appropriate interface based on the user's role. As a result, new roles can be added in the future without modifying the core logic in Main.

CHAPTER 4: Implementation (Java)

Sample Code Snippets:

1. Encapsulation

Hiding internal data using private fields and exposing access through public methods.

```

private String name;
private String nric; //e.g. S1234567A
private String password; //default="password"
private int age;
private String maritalStatus; //"Single" or "Married"

public String getName() { return this.name; }
public String getNric() { return this.nric; }
public int getAge() { return this.age; }
public String getMaritalStatus() { return this.maritalStatus; }
public String getPassword() { return this.password; }

```

2. Inheritance

```

public class Applicant extends User {

```

3. Polymorphism

```

public ApplicantInterface(User currentUser) {
    super(currentUser); //in this case, for currentUser: reference - User; object - roles
    this.applicantService = new ApplicantService((Applicant) currentUser);
    this.projectService = new ProjectService();
    super.setProjectService(this.projectService);
}

```

4. Interface Use

```

public class ApplicantService implements IApplicantService {

```

5. Error Handling

```

if (application == null) {
    throw (new IllegalArgumentException(s:"Application not found.));
}

```

CHAPTER 5: Testing

5.1 Test strategy

We chose manual functional testing because we wanted to evaluate the system from the user's perspective. It is not just about whether the program works, but whether users can interact with it intuitively. While unit testing focuses solely on whether outputs match expected results, manual functional testing allows us to go a step further. It helps verify whether test cases are implemented correctly and ensures the application provides clear, appropriate information in response to user input. By combining correctness checks with a focus on usability, we aimed to make the system more user-friendly. This experience-oriented approach is why we relied on manual functional testing.

5.2 Test Case Table

No.	Type of test case	Description	Input	Expected output	Output
1	Valid User Login	Valid user login - Applicant	NRIC: S1234567A	Successful login - user dashboard	Successful login - user dashboard
		Valid user login - Manager	NRIC: T8765432F	Successful login - manager dashboard	Successful login - manager dashboard
		Valid user login - Officer	NRIC: T2109876H	Successful login – Officer dashboard	Successful login – Officer dashboard
2	Invalid NRIC Format	Wrong format (non-capital letters)	S1234567a	Invalid NRIC format, and ask to try again	Invalid NRIC format, try again
		Different combination of numbers	11234567a	Invalid NRIC format, and ask to try again	Invalid NRIC format, try again
		Various length	S123A	Invalid NRIC format, and ask to try again	Invalid NRIC format, and ask to try again
3	Incorrect Password	Valid username, wrong password	S1234567A 0123456	System should deny access and alert the user to incorrect password	Error: Invalid username or password
4	Password Change Functionality	Applicant - change password	Applicant: S1234567A Old password: password New password: password1	Updates password, prompt for re-login with new credentials	Successful login with new password Can not login with old password
		Manager – Change password	Manager: T8765432F Old password: password New password: password1	Updates password, prompt for re-login with new credentials	Successful login with new password Can not login with old password
		Officer – Change password	Officer: T2109876H Old password: password New password: password1	Updates password, prompt for re-login with new credentials	Successful login with new password Can not login with old password
5	Project Visibility Based on User Group and Toggle	Applicant: John:35, Single	Applicant: S1234567A	Should only be able to view 2-room rooms	Successful view houses/ only show 2-room houses
		Applicant(Manually added): Emily 21, Married	Applicant: S1111111A	Should be able to view both 2-room and 3-room options	Can see both 2 room and 3 room options
		Applicant(Manually added): Jack:22 Married	Applicant: S7654321S	Should be able to view both 2-room and 3-room options	Can see both 2 room and 3 room options
		Applicant(Manually added): Peter: 36 Single	Applicant: S1212121S	Should only be able to view 2-room option	Can only see 2 room options
		Applicant(Manually added): Alex:20, Married	Applicant: S1222222A	Should not be able to see any houses	Cannot see
		Applicant: Eric:34, Single	Applicant: S1256789A	Should not be able to see any houses	Cannot see
6	Project Application	Login as arbitrary user and apply for the same project again	S1234567A, password	Users can only apply for projects relevant to their group and when visibility is on	When user tends to apply a new project, prompt "You have already applied for a project."
7	Viewing Application Status after Visibility Toggle Off	Login as applicant Apply for project (NTU), then login as manager and toggle off visibility. Then re-login as applicant and check "My projects".	S1234567A, password S5678901G, password	Applicants continue to have access to their application details regardless of project visibility.	Applicant still can see applied projects.
8	Manager can change application state from pending to successful or unsuccessful	Login as applicant, then apply to any project Login as the respective manager and change application state	S1234567A, password S5678901G, password	Applicant can check the change of their application state	Can change status from pending to both successful and unsuccessful
9	Single Flat Booking per Successful Application	Apply for Acacia Breeze – 2 room Book again	Applicant: S1234567A Officer: T2109876H	System allows booking one flat and restricts further bookings	Work

10	Applicant's enquiries management	Enquiry: Project Name: Acacia Breeze Content: Toilet works?	Applicant: S1234567A Message: Good flat!	Enquiries can be successfully submitted, displayed, modified, and removed.	Successfully create new flat
11	HDB Officer Registration Eligibility	Apply without applying for the same project and not registered for a different project	Applicant: T2109876H	System allows registration only under compliant conditions	Can successfully submit registration/ Can not submit multiple registration/ Can not apply to houses that have already applied for registration/ Can not submit registration for houses that officer have applied
12	HDB Officer Registration Status	Officer applied for one project/ Manger approves the registration	Officer: T2109876H Manager: S5678901G Registered facility: NTU	Officers can view pending or approved status updates on their profiles.	Show approved
		Officer applied for one project/ Manger rejects the registration	Officer: T2109876H Manager: S5678901G Registered facility: NTU		Show rejected
		Manger rejects Officer's registration after approving it			Prompt error "Registration is already approved."

13	Project Detail Access for HDB Officer	Officer Age: 36 Single	Officer: T2109876H	Officers can always access full project details, even when visibility is turned off.	Can still see the project
14	Restriction on Editing Project Details	Login as officer	Officer: T2109876H	Edit functionality is disabled or absent for HDB Officers	Option does not exist for officer dashboard
15	Response to Project Enquiries	Login in as applicant& Edit as Manger or Officer	Applicant: S3456789E Officer: T1234567J Manager: S5678901G	Officers & Managers can access and respond to enquiries efficiently.	Can see the enquiry result
16	Flat Selection and Booking Management	Login as applicant. Apply NTU; Set to successful;	Applicant: S3456789E Officer: T1234567J Manager: S5678901G	Officers retrieve the correct application, update flat availability accurately, and correctly log booking details in the applicant's profile.	Can set state from successful to booked; available housing number updated Applicant can know whether they can book a flat now
17	Receipt Generation for Flat Booking	Login as user, apply for a house and approved by the manager Officer should be able to generate the respective receipt when set the house to booked	Applicant: S1234567 (apply to NTU) Officer: T1234567J Manager: T8765432F	Receipts are incomplete, inaccurate, or fail to generate	Can generate
18	Create, Edit, and Delete BTO Project Listings	Create project NTUS following instruction as a manager Try to delete the project Try to edit the project	(T8765432F,password)	Managers should be able to add new projects, modify existing project details, and remove projects from the system	Can edit Can create Can delete
19	Single Project Management per Application Period	Try to create projects with overlapping time	Project 1 period: 2026/12/01-2026/12/12 Try to create project 2: 2026/12/2 – 2027/1/1	System prevents assignment of more than one project to a manager within the same application dates.	Work
20	Toggle Project Visibility	Login as Jessica/ Check as any applicant	Officer: S5678901G Applicant: S1234567A	Changes in visibility should be reflected accurately in the project list visible to applicants	Can not see
21	View All and Filtered Project Listings (Manager)	Have option called "filter projects you created"	Manager: S5678901G	Managers should see all projects and be able to apply filters to narrow down to their own projects.	Can filter to created projects
22	Manage HDB Officer Registrations	Login as officer – register for NUS project/ Login as Manager – confirm or reject	Officer: T1234567J Manager: T8765432F	Managers handle officer registrations effectively, with system updates reflecting changes accurately	Work
23	Approve or Reject BTO Applications and Withdrawals	Login as applicant – apply for a house/ Manager should be able to reject or approve application	Manager: S5678901G Applicant: S1234567A (Apply NTU)	Approvals and rejections are processed correctly, with system updates to reflect the decision	Work/ Change application state from pending to successful and unsuccessful / Change withdraw state between true and false
24	Generate and Filter Reports	Simulate three successful applications from three applicants – Login as Manager to generate the report	Filter	Accurate report generation with options to filter by various categories.	Can generate report/ Filter behave normally

CHAPTER 6: Documentation

Developer Guide

Prerequisites:

- Java Development Kit (JDK) 17 or above.
 - Download and install from Oracle or OpenJDK.
- An IDE (recommended: IntelliJ IDEA, Eclipse, or VS Code with Java extensions).

Project Setup:

1. Clone or download the project.
 - a. If using git, type in *gh repo clone aflicuried/SC2002-FDAE-Project-Group-3* into the terminal.
 - b. Alternatively, download and extract the zip file.
2. Open the project in IDE.
 - a. Open the root folder (SC2002-FDAE-Project-Group-3) as a project.
 - b. If prompted, import as a Java project.
3. Check project structure.
 - a. Source code is in the `src/` directory.
 - b. Main entry point: `src/Main.java`
 - c. Supporting packages: Database, Entity, Interface, Service, View, util.
4. Configure the Java SDK.
 - a. In your IDE, ensure the project SDK is set to Java 17 or higher.
5. (If using Eclipse/IntelliJ) Import project files.
 - a. For Eclipse:
 - i. Use File > Import > Existing Projects into Workspace.
 - ii. Select the root directory.
 - b. For IntelliJ:
 - i. Open the project directory directly. IntelliJ will auto-detect `.iml` files.

Building and Running:

- From the IDE:
 - Right-click `Main.java` and select Run.
- From the Command Line:
 - Navigate to the `src` directory and run the following code in the terminal:
 - `cd src`
 - `javac Main.java`
 - `java Main`
- Data Files:
 - The application may read/write data from the `data/` directory. Ensure this directory exists in the project root.
- Troubleshooting:

- If you encounter issues with missing classes, ensure all subfolders in src/ are included as packages in your IDE.
- If you see file read/write errors, check that the data/ directory is present and accessible.

CHAPTER 7: Reflection & Challenges

The development of this project has presented both significant challenges and valuable learning opportunities. Initially, the team was confronted with the considerable scope of the system. However, by decomposing the project into smaller, manageable tasks, we were able to make steady progress and maintain clarity throughout the development process. The decision to structure the codebase into distinct packages—namely Entity, Service, View, and Interface—proved instrumental in maintaining organization and facilitating parallel workstreams. This modular approach minimized merge conflicts and enhanced overall team productivity.

A key strength of the project was the effective application of object-oriented design principles. The use of abstract classes and interfaces enabled the team to extend functionality and introduce new features with minimal disruption to existing code. Implementing the Singleton pattern for database classes ensures data consistency and integrity across the application. Furthermore, the clear separation between user interface components and business logic allowed for isolated testing and easier maintenance.

Team collaboration was another notable success. Regular communication, progress sharing, and mutual support in debugging complex issues contributed to a positive and productive working environment. This collaborative spirit not only accelerated problem-solving but also enhanced the overall learning experience for all team members.

Despite these achievements, the team encountered several challenges. One recurring issue was the lack of specificity in error messages, which occasionally complicated the debugging process. In future projects, greater emphasis will be placed on comprehensive exception handling and the implementation of automated tests to detect and resolve issues at an earlier stage.

Additionally, there was some duplication of code, particularly within the View and Service layers. Moving forward, the introduction of additional helper methods or abstract base classes would help to eliminate redundancy and promote code reuse. Documentation is another area

identified for improvement; more thorough commenting and detailed documentation would benefit both current and future contributors to the project.

This project has helped us to learn about the importance of OODP in developing scalable and maintainable software. The use of abstraction and interfaces facilitated flexibility and adaptability, while the separation of concerns contributed to a more organized and manageable codebase. The practical application of design patterns, such as Singleton, addressed real-world challenges encountered during development. Above all, the experience highlighted the critical role of effective communication and teamwork in the successful completion of complex software projects.

Individual Contributions

Amanda Rae Josephine: Class diagram, Sequence Diagrams, Project Report

Chen Yu Guo: Testing

Lee Teng Yee Sherilyn: Project Report

Li Sihan: Programmer

Tan Kai Hooi: Initial Class Diagram, Project Report

CHAPTER 8: Appendix

The complete source code and documentation for this project are available at:

<https://github.com/aflicuried/SC2002-FDAE-Project-Group-3>

- Java Standard Library: All core functionalities were developed using standard Java libraries provided by the JDK.
- IDE: Development was primarily conducted using IntelliJ IDEA and Eclipse.
- Version Control: Git and GitHub were used for source code management and collaboration.

(No third-party libraries or frameworks beyond the Java standard library were used.)