



### **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
Ang Wei Ming (U2321111B)	SC2002	FCS4	 / 24 Apr 2025
Aiyinikkunnathu Jigeendran Aiswarya (U2423361K)			 / 24 Apr 2025
Angella Feyne Neolani Nocon Pielago (U2422769L)			Angella Pielago / 24 Apr 2025
Aruldass Andrea Nicho (U2423021E)			Andrea Nicho / 24 Apr 2025

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.

# Table of Contents

<b>1 Requirement Analysis &amp; Feature Selection</b>	<b>3</b>
1.1 Understanding the Problem and Requirements	3
1.2 Deciding on Features and Scope	4
<b>2 System Architecture &amp; Structural Planning</b>	<b>5</b>
<b>3 Object-Oriented Design</b>	<b>6</b>
3.1 Class Diagram	6
3.2 Sequence Diagrams	7
3.2.1 HDB Officer	7
3.2.2 HDB Manager	8
3.2.3 Applicant	9
<b>Key Interactions:</b>	<b>10</b>
3.3 Application of SOLID Principles	10
<b>4 Implementation</b>	<b>12</b>
4.1 Tools Used	12
4.2 Code Highlights	12
<b>5 Testing</b>	<b>13</b>
<b>6 Documentation</b>	<b>13</b>
6.1 Javadoc	13
6.2 Developer Guide	13
<b>7 Reflection &amp; Challenges</b>	<b>14</b>
<b>Appendix</b>	<b>15</b>

The project revolves around a Build-To-Order (BTO) project management system, implemented as a command-line application.

# 1 Requirement Analysis & Feature Selection

## 1.1 Understanding the Problem and Requirements

Our team began by thoroughly analysing the BTO Management System requirements document. The main problem involves creating a centralised system for managing Singapore's BTO housing projects, supporting three distinct user roles:

- **Applicant:** Can view, apply for, and withdraw from projects; and submit, edit, or delete enquiries.
- **HDB Officer:** Inherits applicant functionalities and adds the ability to register for projects, book flats, and generate receipts.
- **HDB Manager:** Responsible for creating, editing, and deleting projects; toggling project visibility; managing officer registrations, applicant applications and withdrawals; generating reports; and viewing and replying to enquiries.

Explicit Requirements:

- User authentication via Singpass (simulated with NRIC and default password)

Implicit Expectations:

- Data persistence without the use of databases (user list initiated through a file uploaded into the system at initialisation)
- Proper validation of inputs
- Clear separation of concerns between user roles
- Comprehensive error handling

Resolving Ambiguities:

- Assumed verified user list given in data files
- Assumed only two flat types, strictly "2-Room" or "3-Room"
- Simplified date handling using strings rather than Date objects
- Used basic file I/O for data persistence

## 1.2 Deciding on Features and Scope

From our initial analysis, we arrived at the final list of core features we wanted to implement. These are crucial to addressing the problem outlined and providing functionality.

### Core Features Implemented:

- User authentication, initialisation and access control according to role assigned (Applicant, Officer or Manager)
- Applicant project application and status tracking
- Officer project registration, flat booking and receipt generation
- Manager project management, registration and application management, reporting
- Project creation, editing, and visibility toggling
- Enquiry management system

We also noted a few bonus features that would be a nice addition to the functionality.

### Optional Features:

- Search filtering capabilities and settings
- Comprehensive data validation
- Dedicated data access class

And the following features were ultimately discarded, reserving them for future enhancements to the system should we get a longer timeline to work with.

### Excluded Features:

- Advanced date-handling
- Appointment booking functionality between the Applicant and HDB Officer (post-flat booking success)
- More complex reporting – incorporating graphical representations and data analysis
- Multi-threaded operations

## 2 System Architecture & Structural Planning

The system is implemented using object-oriented programming principles and layered architecture, and is structured as a Maven project using JPMS (Java Platform Module System). Our code is organised into four main packages:

- **Control (edu.ntu.bto.control):** Contains control classes, each of which bundles appropriate actions together and facilitates execution.
- **Model (edu.ntu.bto.model):** Contains entity classes for the respective user types, projects, enquiries, applicant applications and officer registrations.
- **Service (edu.ntu.bto.service):** Contains the central control class, BTOManagementSystem. Executes business logic.
- **Utility (edu.ntu.bto.util):** Contains helper classes (i.e. DataLoader). Represents the data access layer.

The system is then presented on a CLI interface using Main.java.

### Key design decisions:

- Used inheritance for user roles, i.e. Applicant, HDB Officer, HDB Manager (such as Applicant → HDBOfficer)
- Dedicated classes for Projects, Enquiries, Applications, Registrations
- Singleton pattern for system instantiation (BTOManagementSystem), Factory pattern for user creation

### Reflection on Design Trade-offs:

For the implementation of the different roles, we considered interface-based implementation and inheritance, but ended up choosing inheritance. This is due to the commonalities and shared functionality between classes, such as Applicant and HDB Officer, allowing for more efficient code reuse and programming.

We also ended up implementing the Observer pattern for updates to projects, applications and so on, though it was complicated and less straightforward than direct modification. It helped us ensure smooth execution and data management.

## 3 Object-Oriented Design

### 3.1 Class Diagram

We identified our main classes by taking note of the roles involved (Applicant, HDB Officer, HDB Manager) and the entities the different users would interact with. (Project, Enquiry, Application, Registration).

We start with our User class, the base class for the different roles. It stores a user's age, NRIC, marital status, and password, and allows them to change their password. The Applicant class extends User, introducing functionality to apply for a project, check its status, and manage enquiries. HDBManager also extends User, separately introducing functionality to manage HDB projects, officer registrations, and applicant applications, generate reports and reply to enquiries. HDBOfficer extends Applicant, as it is meant to possess all the capabilities of Applicant. It allows registration as an officer for a project, checking of its status, viewing of handled project details regardless of visibility setting, enquiry reply and flat booking. All 3 subclasses of User implement displayMenu() accordingly.

The Project class allows us to create an instance of a project to store relevant project details, such as the number of flats available, with relevant setter methods, such as to decrement the number of officer slots. The Enquiry class similarly allows the instantiation of enquiries submitted by applicants. The Application class establishes links between the applicant who filed it and a project. It stores its status and the flat type chosen as attributes. The Registration class similarly establishes links between the officer who filed it and a project, and also has a status attribute for viewing.

Lastly, the BTOManagementSystem represents the entire system, serving as a platform for the interaction between users and projects and the boundary through which users log in. It is responsible for the initialisation of the system, with the DataLoader class acting as a helper class to read in data from the Excel files provided.

We implemented 5 control classes as well, each focusing on specific sets of actions for control by users. **ApplicationControl** deals with applications, allowing users (applicants and officers) to apply for projects and withdraw. Officers can also retrieve applications and book flats through it. **EnquiryControl** deals with enquiries, facilitating their creation, retrieval, and deletion, and allowing replies and edits. **ProjectControl** handles projects and provides viewing and filtering capabilities. Lastly, there is **OfficerControl** and **ManagerControl**, featuring more focused capabilities unique to each of the roles. **OfficerControl** allows officers to register for projects, while **ManagerControl** allows managers to create, edit and delete projects, handle applications, registrations and withdrawals, and generate reports.

### **Relationships:**

- Inheritance between User and its subclasses
- Association between Users and Projects
- Composition between Registration and HDBOfficer

## **3.2 Sequence Diagrams**

**We decided on 3 sequence diagrams to accurately describe the flow of our model.**

### **3.2.1 HDB Officer**

#### **Diagram Description:**

The UML Sequence Diagram details the interaction flow for the HDB Officer's role in applying for a BTO and register to handle a project:

#### **1. Login Phase:**

- The HDB Officer enters credentials through the CLI (Main/Scanner), and the request is sent to **BTOManagementSystem**.
- The system verifies the credentials and returns the appropriate **HDBOfficer** instance.

#### **2. Application Submission:**

- The officer can browse the open and visible projects and choose to apply for a project.

- The system filters available projects and sets the officer's application status to "Pending." The criterias for filtration include the visibility setting, the checking of if the officer has pending/approved registration and if the project itself has enough slots
- The application status is initially set to pending

### 3. Manager Approval Simulation:

- The HDB Manager will be able to view these application requests and approve/deny
- A simulated manager approval updates the officer's application status to "Successful."

### 4. Handling of the project:

- Once approved, an officer can handle a project
- The system queries and returns assigned projects to the officer that officer can choose from
- **OfficerControl** then updates the officer's registration and finalises the individual's management responsibilities.

### Key Interactions:

**HDBOfficer** → **BTOManagementSystem**: which controls the login and command input.

**BTOManagementSystem** → **OfficerControl / Registration**: which will coordinate the officer application and subsequent registration process

**OfficerControl** → **Project / Registration**: which will check and update which officer is assigned to which assignment

**CLI**: boundary interface for all user interaction

## 3.2.2 HDB Manager

### Diagram Description:

The UML Sequence Diagram details the interaction flow for the HDB Manager's role during the following process

#### 1. Login Phase:

- The HDB Manager enters credentials through the CLI (Main/Scanner), and the request is sent to **BTOManagementSystem**.
- The system verifies the credentials and returns the appropriate HDBManager instance.

#### 2. Project Creation and Management



- The manager chose the option to create a BTO project. Through the system, a new Project object is made with the details of the project name, neighbourhood, flat type, the number of units and application dates. The manager can edit and delete these posts and control who views them
3. **Handling Officer registration**
    - The manager can view the list of pending office registrations in a project and approve/reject them.
    - The officer's registration status is updated according to this
  4. **Managing the applicants' applications**
    - The manager can access the list of applicant submissions and approve or reject them. This approval depends on the unit's availability. When approval is sent, the applicant can book the flats
    - The manager can also choose to accept/reject the withdrawal requests given by the applicants
  5. **Generating reports**
    - Managers can generate reports with filter options such as marital status.
  6. **Handling enquiries**
    - The manager can view all enquiries and submit replies directly

#### **Key Interactions:**

**HDBManager** → **BTOManagementSystem**: To create new projects, approve matters and create reports

**BTOManagementSystem** → **Project**: For storing, editing, and retrieving project-related data.

**BTOManagementSystem** → **Applicant/HDBOfficer**: To update application/registration statuses.

**CLI (Main/Scanner)**: Provides input and output in sequence.

### 3.2.3 Applicant

#### **Diagram Description:**

The UML Sequence Diagram details the interaction flow for the Applicant's role during the following process:

1. **Login Phase:**
  - The HDB Officer enters credentials through the CLI (Main/Scanner), and the request is sent to **BTOManagementSystem**.
  - The system verifies the credentials and returns the appropriate HDBOfficer instance.
2. **Browsing the projects and submitting applications:**
  - Applicant can view open projects and filter available projects based on visibility settings and applicant groups(eg, single or married)
  - When an applicant does apply, **ApplicationControl** ensures the individual matches the criteria (such as being single to apply for a 2-Room flat)

- Only if the applicant passes the validation can they create an application to receive a pending status
- 3. **Viewing Application status**
  - It can be done at any time. **ApplicationControl** can fetch and display whether an applicant's application is pending, successful, unsuccessful or booked.
- 4. **Withdrawing application.**
  - If they wish to withdraw at any point, a request can be made to **ApplicationControl**. With this, the system will update the status to **Withdrawn**
- 5. **Enquiry Management**
  - Applicants can send an enquiry for a project, which is sent to **EnquiryControl**, which is then saved.
  - Applicant can view, edit and delete the enquiries that were sent
- 6. **Booking the flat.**
  - If their application status is Successful, an HDB officer can facilitate the process to book the flat.
  - **ProjectControl** checks if the unit is available, and if so, the booking is confirmed, and the application status changes to **booked**

### Key Interactions:

- **Applicant** → **BTOManagementSystem**: To log in and navigate the application, enquiry and booking menus.
- **BTOManagementSystem** → **ProjectControl**: To view projects and flat booking status.
- **BTOManagementSystem** → **ApplicationControl**: To create the application and fetch/update status
- **BTOManagementSystem** → **EnquiryControl**: For submission, retrieval, and modification of enquiries

### Annotations:

- Each message in the diagram is annotated to indicate where polymorphism is used (e.g., the overridden `displayMenu()` methods).

The diagram shows the clear separation of boundary (CLI), control (BTOManagementSystem), and entity (model classes) responsibilities..

## 3.3 Application of SOLID Principles

We capitalised on the 5 SOLID object-oriented design principles throughout our project, namely in the following examples:

### 1. Single Responsibility Principle:

The BTOManagementSystem class handles only core operations, acting as an initialisation point for the entire system and leaving more user-specific behaviours to the user-type and control classes. This prevents BTOManagementSystem from having to know excessive amounts of information on each user type, and just has it focus on managing the entire system. This improved cohesion in our classes, making it more maintainable.

## **2. Open-Closed Principle:**

We adhered to this principle by having a base User class that was extended appropriately according to the hierarchy by lower classes Applicant, HDB Officer and HDB Manager, leaving the user role system open to extension (i.e. new roles introduced) but closed for modification of existing User code. This will make future enhancements easier.

## **3. Liskov Substitution Principle:**

All User subclasses properly implement displayMenu(), an abstract method declared in User, not asking for any extra, unnecessary input and producing the required menu appropriately. Each subclass implements the method according to its needs and attributes, ensuring proper functionality and the right power given to the right user role without confusion.

## **4. Interface Segregation Principle:**

Similar to what was elaborated on for the Open-Closed principle, separate classes were defined for Applicant, HDB Officer, and HDB Manager to avoid a general User class. This ensures that each unique class only has the specific attributes and methods it needs and no more. This helped with programming and editing.

## **5. Dependency Injection Principle:**

High-level modules like BTOManagementSystem depend on the User class's abstraction at login. With the details strictly implemented by the User subclasses accordingly, this inversion of control achieves a more straightforward design and easier management of code.

## 4 Implementation

### 4.1 Tools Used

- Java 17
- IDE: Eclipse
- draw.io for UML
- GitHub for version control

### 4.2 Code Highlights

#### Encapsulation Example:

User attributes such as age and NRIC are protected, ensuring they belong to and are accessed by only the User class and its subclasses.

```
public abstract class User {  
    protected String nric;  
    protected int age;  
    protected String maritalStatus;  
    protected String password;
```

#### Polymorphism Example:

```
/**  
 * Display the CLI menu for the user.  
 */  
public abstract void displayMenu(Scanner scanner, BTOManagementSystem system);
```

```
@Override  
public void displayMenu(Scanner scanner, BTOManagementSystem system) {  
    boolean logout = false;  
    while (!logout) {  
        System.out.println(x:"\n=== Applicant Menu ===");  
        System.out.println(x:"1. View Projects");  
        System.out.println(x:"2. Apply for Project");  
        System.out.println(x:"3. View Application Status");  
        System.out.println(x:"4. Withdraw Application");  
        System.out.println(x:"5. Submit Enquiry");  
        System.out.println(x:"6. View Submitted Enquiries");  
        System.out.println(x:"7. Edit Submitted Enquiry");  
        System.out.println(x:"8. Delete Submitted Enquiry");  
        System.out.println(x:"9. Change Password");  
        System.out.println(x:"10. Logout");  
        System.out.print(s:"Select option: ");  
        int choice = Integer.parseInt(scanner.nextLine());
```

displayMenu(), an abstract method declared in the User class, is uniquely implemented in its subclasses, such as the Applicant class.

#### Inheritance Example:

```

/**
 * Represents an HDB Officer.
 * HDB Officers have all the capabilities of Applicants plus additional officer-specific operations.
 */
public class HDBOfficer extends Applicant {
    private String registrationStatus; // "Pending", "Approved", "Rejected"
    private Project handledProject; // The project for which the officer is registered
}

```

The HDBOfficer class extends the Applicant class.

### Error Handling:

Before a manager can successfully create a project, extensive validation is carried out to ensure there are no errors in each input field. Appropriate error messages are shown to get the correct input.

```

System.out.print(s:"Enter Type 1 (e.g., 2-Room): ");
String type1 = scanner.nextLine();
while (!type1.equalsIgnoreCase(anotherString:"2-Room") && !type1.equals("2-Room")) {
    System.out.println(x:"Invalid flat type entered. Try again.");
    System.out.print(s:"Enter Type 1 (e.g., 2-Room): ");
    type1 = scanner.nextLine();
}
System.out.print(s:"Enter number of units for Type 1: ");
int unitsType1 = Integer.parseInt(scanner.nextLine());
while (unitsType1 < 0) {
    System.out.println(x:"Invalid number entered. Try again.");
    System.out.print(s:"Enter number of units for Type 1: ");
    unitsType1 = Integer.parseInt(scanner.nextLine());
}
System.out.print(s:"Enter selling price for Type 1: ");
double priceType1 = Double.parseDouble(scanner.nextLine());
while (priceType1 < 0) {
    System.out.println(x:"Invalid number entered. Try again.");
    System.out.print(s:"Enter selling price for Type 1: ");
    priceType1 = Double.parseDouble(scanner.nextLine());
}

```

## 5 Testing

We employed manual functional testing with both valid and invalid inputs, as well as edge case testing of boundary conditions.

(refer to Appendix for test cases table)

## 6 Documentation

### 6.1 Javadoc

(included along with submission)

### 6.2 Developer Guide

1. Clone repository
2. Import as a Maven project
3. Ensure data files are in /data directory
4. Run Main.java

## 7 Reflection & Challenges

### **Successes:**

- Effective use of OOP principles
- Clean and clear separation of concerns
- Comprehensive error handling and case testing
- Meeting all core requirements of the system

### **Challenges Overcome:**

- File I/O Complexity: Faced difficulty reading the file into the system, implemented the DataLoader utility class to help
- Role segregation: Used inheritance with care and attention
- State Management: Carefully tracked and maintained application/project/registration statuses with the help of control classes
- Importing of JAR files: We overcame the challenge by using Maven to help with dependency management.

### **Lessons Learned:**

- Breaking down a detailed problem requirement document (obtaining names for entity classes, etc.), abstraction through extracting the critical features to implement in the solution
- Importance of appropriate UML design
- The value of testing properly with comprehensive test cases
- Benefits of clear role separation, inheritance

### **Possible Future Improvements:**

- Fully implement the edit project capability
- Enhanced data validation
- Enhanced manager reports
- Date-handling using LocalDate

# Appendix

- GitHub Repository: <https://github.com/angwm1/BTOManagementSystem2.git>

## Test Cases:

No.	Test Case	Details	Expected	Outcome
1	Valid User Login	Manager NRIC: S5678901G Applicant NRIC: S1234567A Officer NRIC: T2109876H	Users should be able to access their dashboard according to their assigned roles.	Pass
2	Invalid NRIC Format	NRIC: 123456789	User receives error message asking for re-entry	Pass
3	Incorrect Password Handling	NRIC: S1234567A	Access should be denied with an appropriate alert if the password is incorrect.	Pass
4	Password Change Functionality	NRIC: S1234567A	Upon changing the password, the system should update the credentials, prompt re-login, and accept the new password.	Pass
5	Project Visibility Based on User Group and Toggle	Manager NRIC: S5678901G Applicant NRIC: S1234567A T2345678D S3456789E	Projects should be visible only to users matching specific criteria such as age, marital status, and visibility toggle.	Pass
6	Project Application	Manager NRIC: S5678901G Applicant NRIC: S3456789E T2345678D	Users should only be able to apply for projects relevant to their group and when visibility is enabled.	Pass

7	Viewing of Application Status After Visibility Toggle Off	Manager NRIC: S5678901G Applicant NRIC: S1234567A	Applicants can view their application status regardless of project visibility	Pass
8	Single Flat Booking Allowed Per Successful Applicant	Applicant NRIC: S1234567A Manager NRIC: S5678901G	An error message is displayed when a successful applicant tries to book more than one flat	Pass
9	Applicant's Enquiry Management	NRIC: S1234567A	Applicant is able to display, modify and delete enquiries submitted	Pass
10	HDB Officer Registration Eligibility	Officer NRIC: T2109876H Manager NRIC: S5678901G	Error message displayed when an officer applies for a handled project as an applicant, and vice versa	Pass
11	HDB Officer Registration Status	NRIC: T2109876H	Officers can view their registration status on their profiles	Pass
12	Project Detail Access for HDB Officer	Manager NRIC: S5678901G Officer NRIC: T2109876H	Project officers can always access full project details, even when visibility is turned off	Pass
13	HDB Officer Restriction on Editing Project Details	NRIC: T2109876H	Project officers are unable to edit projects in any way	Pass
14	Response to Project Enquiries	NRIC: T2109876H	Officers & Managers can access and respond to enquiries	Pass



			efficiently (can view and reply to enquiries)	
15	Flat Selection and Booking Management	Applicant NRIC: S1234567A Manager NRIC: S5678901G Officer NRIC: T2109876H	Once an application is approved by a manager, officers retrieve the applicant's NRIC and flat type chosen to book a flat, with booking details correctly logged in the applicant's profile	Pass
16	Receipt Generation for Flat Booking	NRIC: T2109876H	Accurate and complete receipts are generated automatically for each successful booking	Pass
17	Create, Edit, and Delete BTO Project Listings	NRIC: S5678901G	Managers can add new projects, modify existing project details, and remove projects from the system	Pass
18	Single Project Management per Application Period	NRIC: S5678901G	The system prevents the assignment of more than one project to a manager within the same application period	Pass
19	Toggle Project Visibility	NRIC: S5678901G	Changes to visibility should be reflected in the project display accordingly	Pass
20	Viewing and Filtering Projects (Manager)	NRIC: S5678901G	The manager can view all projects and filter to see their own projects	Pass
21	Manage HDB Officer	Manager NRIC: S5678901G	Manager can manage	Pass

	Registrations	Officer NRIC: T1234567J	registrations, with changes to relevant details in the system reflected (eg number of officer slots)	
22	Manage Applications and Withdrawals	Manager NRIC: S5678901G Applicant NRIC: T7654321B	The manager can manage applications and withdrawals by applicants, with the status updated accordingly	Pass
23	Generate Report (includes Filter)	Manager NRIC: S5678901G	The report contains relevant details of applicants with bookings	Pass