BTO Housing Management System

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.

Names:	**Student ID:**	**Course / Lab Group**	**Signature / Date**
Jovan Ho Wei Quan	U2422097L	SC2002 / FCS3	[illegible]
Ethan Chuah Jhein Kiat	U2422252C	SC2002 / FCS3	[illegible]
Htet Moe Han	U2421958B	SC2002 / FCS3	[illegible]
Gay Jun Kai, Brendan	U2422686D	SC2002 / FCS3	[illegible]
Harish Raaj Sivakumar	U2420927C	SC2002 / FCS3	[illegible]

Chapter 1: Requirement Analysis & Feature Selection

1.1 Understanding the Problem and Requirements

Our group began by reading through the BTO document line-by-line and observed the capabilities and functionalities for different users. From our observations and understanding, the BTO Management System is a platform application for managing housing applications within 3 distinct user roles - Applicants, HDB Officers and HDB Managers. The main problem domain is a housing application system that serves multiple user positions with different permissions and requirements.

The key requirements outlined are:

- **User Authentication:** Users have to log in with their respective NRIC and password
- **Role-Based Access Control**: 3 distinct user roles with hierarchical access permissions
- **Project Management:** Creating, editing, deleting and visibility management of BTO projects
- **Application**: Allowing users to apply for eligible projects
- **Flat Booking:** Processes for successful applicants to secure specific housing units
- **Enquiry System:** Communication channel between applicants and HDB officers/managers
- **Generating Reports:** Generating reports based on application and booking data

Some of the implicit expectations includes:

- User-friendly interface when navigating through the application
- Supports case-sensitive user inputs (toLowerCase() method)
- Manages error handling

Ambiguous areas that required interpretation:

- Different categories of report generation:
    - Application Summary Report
    - Booking Details Report
    - Flat Type Preference Report
    - Remaining Units Report

    - Officier Performance Report
    - System Summary Report
- How different users navigate through the application
    - Created different User Interface (UI) for different user roles with respect to their methods/purposes

1.2 Deciding on Features and Scope

Our group categorized the features into three categories - core, optional, and excluded - and aim to ensure that the core features are implemented first and are able to work together with other classes, then followed by other features using the object-oriented principles we have learnt throughout the course.

The core features consist of:

- User Authentication
    - Login and logout system with the functionality to change his/her password
    - Role-based access control within the 3 user roles
    - The different user interface for each role
- Project Management
    - Creating, editing, and deleting of projects (For HDB Officers/Managers only)
    - Functionality of toggling visibility of projects
    - Application period management within an application period
    - Officer slot allocation and management
- Application Processing
    - Eligibility verification based on age, marital status, and flat type
    - Status tracking (Pending, Successful, Unsuccessful, Booked)
    - Application withdrawal handling
    - Flat booking for successful applicants
- Enquiry System
    - Submission of enquiries (For Applicants)
    - Functionality of responding to enquiries (For HDB Officers/Managers only)
    - View, edit ,and delete enquiries
- Report Generation
    - Report of a list of applicants with their respective flat booking and details

    - Functionality of filtering

The optional and bonus features are:

- Admin Role:
    - Receives a general report of all user roles when one enquires it
    - Administrators have the authority to reset a user's password in case it is forgotten
- Overwrite the .dat file:
    - To preserve program progress after actions (changing password, applying for project, etc)
    - Uses the recently saved progress when the application is launched again

The excluded features are:

- Graphical User Interface (GUI):
    - To offer users a visual and interactive experience, a GUI is recommended
- Supports multiple language::
    - To make it more user-friendly in real-life applications, adding multiple language settings helps users who are not fluent in one specific language

Chapter 2: System Architecture & Structural Planning

2.1 Planning the System Structure

This chapter explains how we translated our functional requirements into a clear, object‑oriented architecture. It lays out our four‑layer design, shows how we modeled user journeys before coding, and reflects on the trade‑offs we made along the way. Before writing a single line of code, we adopted a top‑down object‑oriented approach to decompose the Build‑To‑Order (BTO) Management System into cohesive, loosely‑coupled components:

Logical Component Breakdown

We organized the system into four layers to enforce separation of concerns:

- **Data Utility Layer**

  This layer is responsible for all file‑based initialization: loading users, application projects, enquiries and flat types from spreadsheets. We abstracted database operations to persist any updates made by the application to maintain data persistence upon exiting the program.

- **Boundary Layer**
  Serving as the external interface, it provides a Command-Line Interface (CLI) tailored to each profile (Applicant, HDB Officer and HDB Manager). Depending on the option the user selects from the text-menu, the boundary component validates their input and map it to corresponding methods which hands control to the next layer.

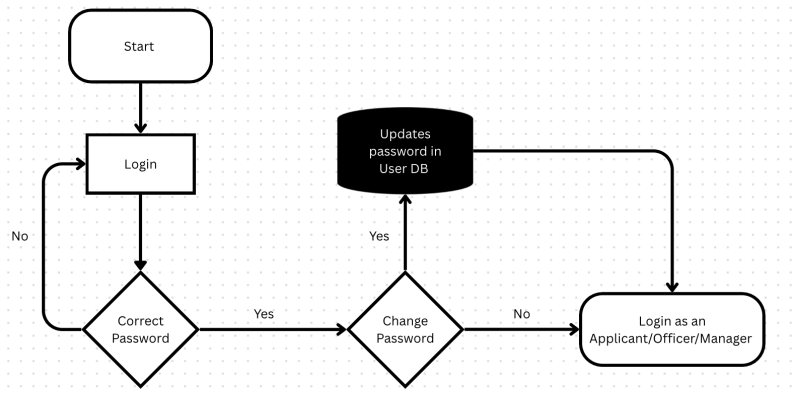
- **Control Layer**
  At the heart of our system architecture lies the Control Layer, where we encapsulate every distinct business flow. Services here handle authentication, eligibility checks, application submission, booking confirmation, enquiry routing, and report generation. They coordinate with entities and update the data utility layer if any, and ensure domain logic remains centralized.

- **Entity Layer**
  It houses the fundamental data models. These entities represent the fundamental elements of the program and encapsulate their attributes, properties and associated validations if any. It also contains a getter setter method to initialize the class. The following classes are as follows: User, Applicant, HDBOfficer, HDBManager, Project, Application, FlatBooking, OfficerRegistration,Enquiry, Receipt, Report. They live as an entity class in this layer. Apart from that, it also houses the enumerator which represents a group of constants that are unchangeable. They are as follows: UserRole, ApplicantStatus, FlatType, MaritalStatus.

Modeling User Flows & Early Visuals

Before coding, we captured every critical path in:

- **Flowcharts** illustrating step-by-step decision logic (e.g. Logging in as a User)


Start
Login
No
Correct Password
Yes
Change Password
Yes
Updates password in User DB
No
Login as an Applicant/Officer/Manager

- For this example, from the requirements of the report, we knew we needed to have data persistence and the choice to change password. Thus, through this simple flowchart presented above, it reflects the requirements needed to login as a user. Based on the credentials entered by the user, the system will log the user in one of the 3 roles which are Applicant, HDB Officer and HDB Manager.

2.2 Reflection on Design Trade-offs

Early in the design process, we faced several key trade-offs:

- **Controller + Logic vs. Strict Separation**
  We debated merging CLI input parsing with business logic, but chose to keep the Boundary Layer solely responsible for user interaction. This isolation made testing for our Control services predictable and straightforward. Thus we chose to adopt a Boundary-Control-Entity Layer, each tied to their own responsibility.
- **Generic vs. Specialized Classes**
  Combining Booking and WithdrawalRequest into a superclass would reduce duplication, but we kept them distinct so each can encapsulate its unique validation logic. This allows us to adhere to the Single Responsibility Principle.
- **Data Modeling Debates**

- Rather than splitting replies into a separate table, we opted to store Officer/Manager responses in the same Enquiry table. This simplified our schema and reduced the complexity of maintaining multiple entities.

By consciously documenting these compromises and sketching our boundary–control interactions alongside visual models, we established a maintainable, testable, and extensible architecture—ready to adapt as requirements evolve.

Chapter 3: Object-Oriented Design

3.1 Class Diagram

- **How were the main classes identified?**
  Other than the 3 obvious classes listed(Applicant, HDBofficer and HDBManager), and the generic User class, which serves as a super-class of these 3 classes. The main classes are:
    - Project(“Able to create, edit, and delete **BTO project** listings.”)
    - Enquiry(“applicants can submit **enquiries**”)
    - FlatBooking(“Applicant can **book one flat through the HDB Officer** (Only HDB Officer can help to book a flat”)
    - Application(“With Applicant’s successful BTO application…”)
    - FlatType(Enum)
    - ApplicationStaus(Enum)
    - MaritalStatus(Enum)

- **What are the responsibilities of each class?**

- User
    - Encapsulates common identity/auth data (NRIC, password, age, marital status) and login validation.
- Applicant
    - Tracks “currentApplicationId,” knows whether they’ve applied or need to withdraw.
- Project
    - Holds flat-type inventories, application window dates, visibility flag, and assigned officers.

- Application
    - Stores the status and timestamp of an applicant’s bid for a flat.
- FlatBooking
    - Created only once an application is successful; immutable record of who booked what.
- Enquiry
    - Has a question/response pair with a timestamp and answered flag.

- **How were relationships (inheritance, association, aggregation) determined?**

- Inheritance(Is-a)
    - Applicant, HDBManager and HDBOfficer **is‑a** User because they share identity/auth attributes and only differ in role‑specific methods.
- Aggregation/Composition(Has-a)
    - Application **has-a** FlatBooking once it is fulfilled, because a booking cannot live without its application\
- Association
    - Applicant holds only the currentApplicationId string rather than a full object reference to simplify persistence and avoid circular dependencies in the UI.

- **What trade-offs were considered?**
    - We used String IDs instead of object references because it simplifies serialisation, breaks cycles and aligns with our DB classes. However, there may be instances with typos that will not be caught automatically.

    - We chose file-based persistence over real databases as java serialization to “*.dat” files was quick to wire up and required no extra infrastructure. However, a real database would have given us more scalability and flexibility at the cost of extra setup and more complex code.

3.2 Sequence Diagrams

We decided to create sequence diagrams that address the most critical processes within the system, focusing on how users interact with key system components and how data flows across the architecture. These diagrams were chosen to highlight both functional and non-functional requirements of the system, ensuring that the design can handle real-world scenarios effectively.

Key objectives behind selecting these use cases include:

- **Demonstrating Conditional Decision Making**
  Our diagrams depict how business rules and conditions impact the flow of the system. The eligibility checks and role-specific actions highlight how decisions are made based on specific user conditions, ensuring that only eligible officers can apply or register for projects. This also demonstrates the flexibility of the system to adapt to varying user roles and permissions.
- **Mapping the Interaction between Layers**
  The diagrams map out how data is passed between the UI, controller, and model layers, ensuring that each component interacts efficiently. For example, the project management layer handles the eligibility check, while the application layer manages the officer's registration status. These layers interact to ensure data consistency and business rule enforcement.
- **Complex Workflow Integration**
  These scenarios show how different parts of the system (e.g., user interfaces, controllers, services) must work together seamlessly to perform complex workflows. For example, the officer registration process integrates user input, business rule validation, and approval workflows, which involves collaboration between the controller layer and the manager approval process.

Sequence Diagram 1: HDB Officer application for BTO as an Applicant
This scenario demonstrates an important flow in the system, from user authentication to application submission and flat booking. It involves key interactions such as role based access control, eligibility checks, and the management of objects like Project, Application, and FlatBooking. It also illustrates how the controller layers (ProjectController, ApplicantController, etc) interact with data models and the UI to ensure a seamless user experience.

Sequence Diagram 2: HDB Officer registers for Project Handling
This scenario illustrates the role based registration process for HDB Officers, focusing on the validation of eligibility and the approval process managed by the HDB Manager. The

diagram highlights the interaction between the controller, manager approval, and project management layers, as well as the various checks (such as confirming the officer isn’t already involved in another project). This scenario validates the system's ability to manage these conditional checks and role specific actions while ensuring seamless integration across the UI, business logic, and database models.

3.3 Application of OOD Principles (SOLID)

1. Single Responsibility
    - In ProjectsDB, its **sole** job is to load, save, and query Project objects from its backing file (projects.dat). We did not put any business rule code in it, making the persistence code easy to test and isolated and it will never change because our UI needs a new menu option.
2. Open/Closed
    - The FlatType enum drives available flat-type logic, and Project.decrementFlatTypeUnits(FlatType) works for any future flat type we want to add. By centralizing flat-type behaviors in the enum and generic map operations, adding a new FOUR_ROOM type requires **no** changes to Project methods, allowing it to be open for **extension** but closed for **modification**.
3. Liskov Substitution
    - HDBOfficer and HDBManager both subclass User and can safely be passed to any method expecting a User, such as LoginController.getCurrentUser(). We needed polymorphism so that authentication and role-checks work uniformly: any User can log in, and then .getRole() drives menu dispatch. This ensures that all subclasses of User **work seamlessly** with any methods that work with User.
4. Interface Segregation
    - Rather than having one giant SystemController interface, we have **multiple focused controllers**: LoginController, ProjectController, ApplicationController, etc. This is so that each UI class only depends on the controllers it needs
5. Dependency Inversion
    - ApplicationController depends on the **abstractions** ApplicationDB, ProjectDB, and UserDB rather than on concrete file-I/O code. This way we are

able to swap out our file-based DBs for a real database without changing business logic.

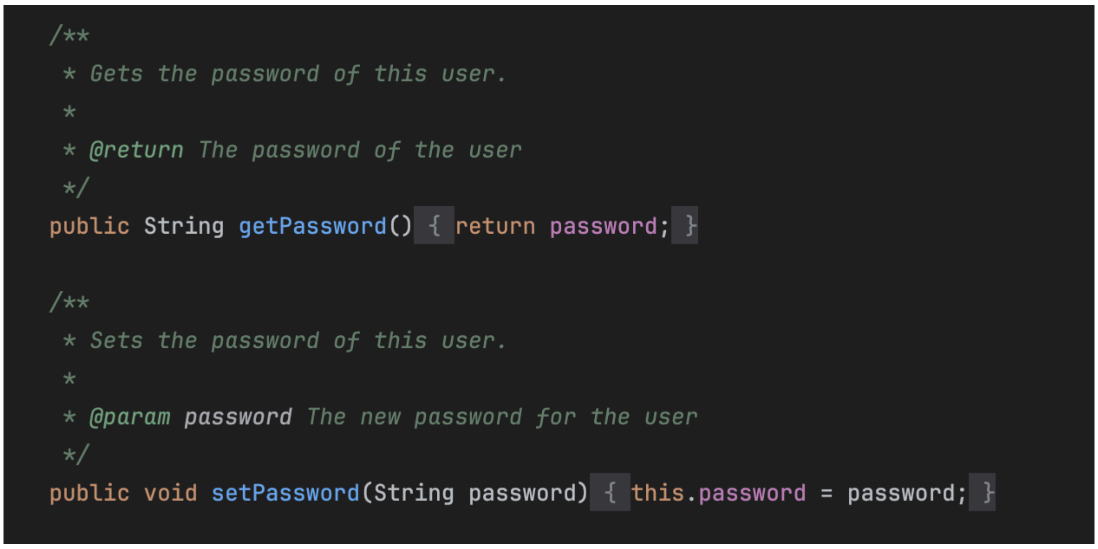
Chapter 4: Implementation

4.1 Tools used:

- Java 17
- IDE: IntelliJ / Eclipse
- Version control: GitHub
- Class & Use Case Diagram: Visual Paradigm
- Flowchart: Canva

4.2 Sample Code Snippets

- Encapsulation

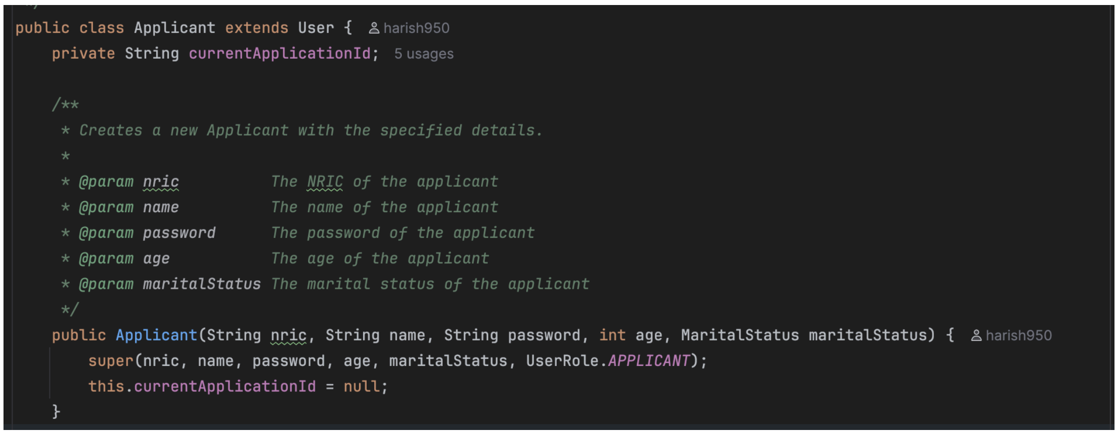


```java
/**
 * Gets the password of this user.
 *
 * @return The password of the user
 */
public String getPassword() { return password; }

/**
 * Sets the password of this user.
 *
 * @param password The new password for the user
 */
public void setPassword(String password) { this.password = password; }
```

As seen above, The getPassword() and setPassword() methods, known as getters and setters, provide controlled access to the Password attribute.

- Inheritance

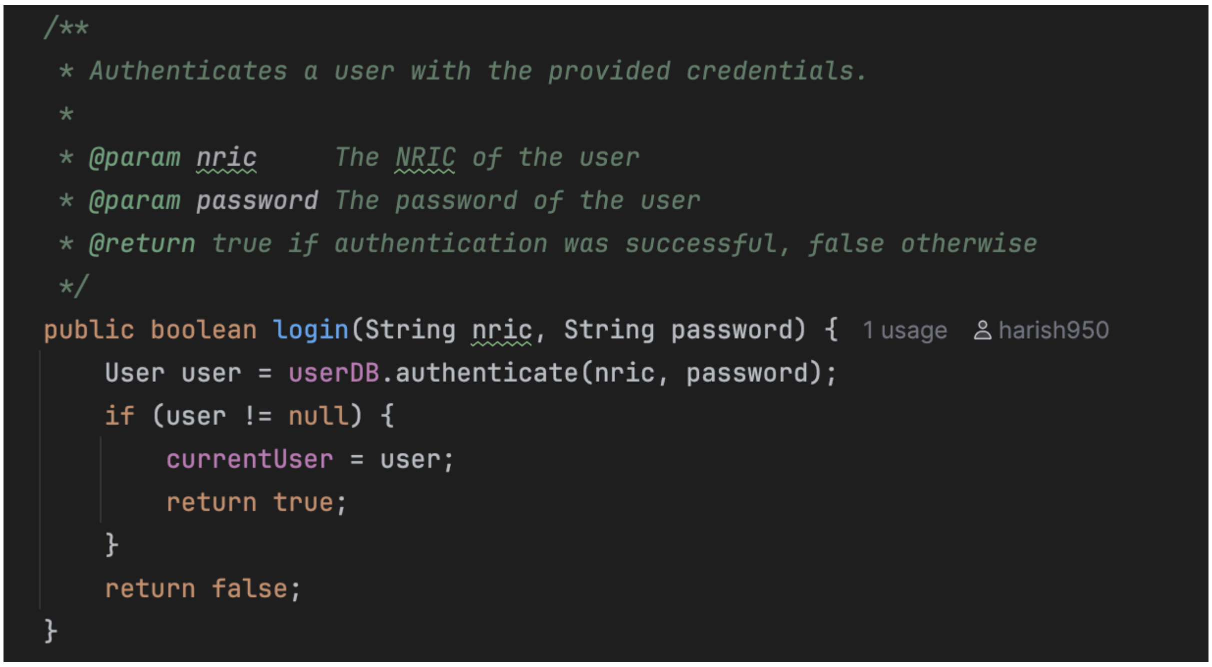


```java
public class Applicant extends User {  harish950
    private String currentApplicationId;  5 usages

    /**
     * Creates a new Applicant with the specified details.
     *
     * @param nric          The NRIC of the applicant
     * @param name          The name of the applicant
     * @param password      The password of the applicant
     * @param age           The age of the applicant
     * @param maritalStatus The marital status of the applicant
     */
    public Applicant(String nric, String name, String password, int age, MaritalStatus maritalStatus) {  harish950
        super(nric, name, password, age, maritalStatus, UserRole.APPLICANT);
        this.currentApplicationId = null;
    }
```

As seen above, it reflects Inheritance as the applicant inherits from the User, with User being the Superclass, and Applicant being the Subclass.

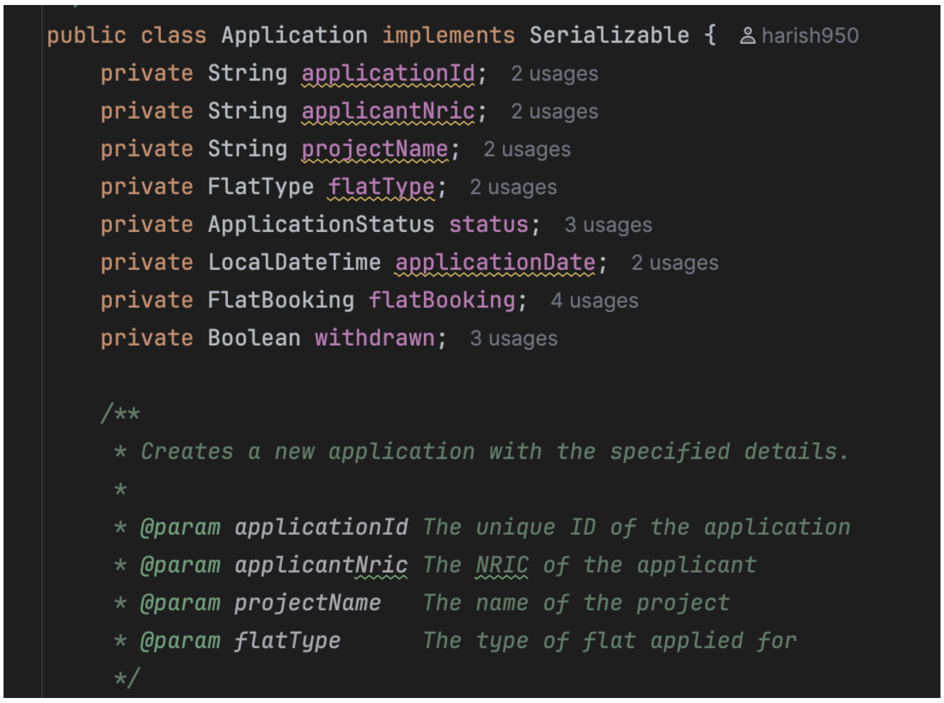
- Polymorphism



```java
/**
 * Authenticates a user with the provided credentials.
 *
 * @param nric     The NRIC of the user
 * @param password The password of the user
 * @return true if authentication was successful, false otherwise
 */
public boolean login(String nric, String password) {  1 usage  harish950
    User user = userDB.authenticate(nric, password);
    if (user != null) {
        currentUser = user;
        return true;
    }
    return false;
}
```

As seen above, it reflects Polymorphism, as based on the authentication, it returns the User reference which at runtime can be an Applicant, HDB Officer or HDB Manager.

- Interface usage

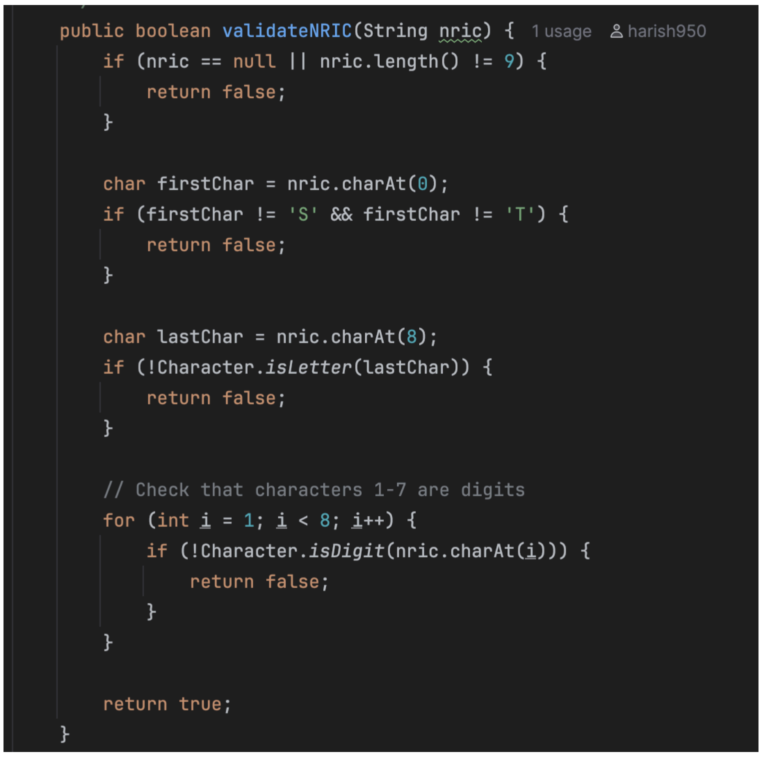


```java
public class Application implements Serializable {  harish950
    private String applicationId;  2 usages
    private String applicantNric;  2 usages
    private String projectName;  2 usages
    private FlatType flatType;  2 usages
    private ApplicationStatus status;  3 usages
    private LocalDateTime applicationDate;  2 usages
    private FlatBooking flatBooking;  4 usages
    private Boolean withdrawn;  3 usages

    /**
     * Creates a new application with the specified details.
     *
     * @param applicationId The unique ID of the application
     * @param applicantNric The NRIC of the applicant
     * @param projectName   The name of the project
     * @param flatType      The type of flat applied for
     */
```

As seen above, the Application implements the Serializable interface

- Error handling



```java
public boolean validateNRIC(String nric) {  1 usage  harish950
    if (nric == null || nric.length() != 9) {
        return false;
    }

    char firstChar = nric.charAt(0);
    if (firstChar != 'S' && firstChar != 'T') {
        return false;
    }

    char lastChar = nric.charAt(8);
    if (!Character.isLetter(lastChar)) {
        return false;
    }

    // Check that characters 1-7 are digits
    for (int i = 1; i < 8; i++) {
        if (!Character.isDigit(nric.charAt(i))) {
            return false;
        }
    }

    return true;
}
```

Above reflects the error handling when it comes to validating the NRIC. It handles returns false if any of the following requirements is not met, and returns true after checking the parameters of the NRIC.

Chapter 5: Testing

No.	Test Case	Expected Behaviour	Failure Indicators	Check
1 - Application Initialization				
1.1	No existing data	Application is able to initialise with data files 'usersInit.csv' and 'projectsInit.csv' Application is able to save data into the respective .dat files	1. Application cannot read csv files 2. Application cannot save data into .dat files	✅
1.2	Has existing data	Application is able to use data from the respective .dat files	1. Application cannot read .dat files	✅
1.3	Login Menu	Application shows the Login Menu	1. Login Menu not shown	✅
2 - Authentication				
2.1	Valid user login	User should be able to access their respective menu based on their roles	1. User cannot log in 2. User cannot view their respective menu	✅
2.2	User not found	Applications would deny access and notify user that account not found	1. User is able to log in with an invalid account	✅
2.3	Invalid NRIC format	Application would deny access and notify user about incorrect NRIC format	1. User is able to log in with an invalid NRIC	✅
2.4	Incorrect password	Application would deny access and notify user about incorrect password	1. User is able to log in with an incorrect password	✅
2.5	Default password	Application would prompt user to change password if it is the default 'password'	1. User is not prompted to change	✅
2.6	Change password	Application updates the new password, prompts re login	1. User is still able to login with old password 2. User is not able to login with new password	✅
3 - Applicant				
3.1	Project Visibility	Applicants can only see projects - Based on their age and marital status	1. Applicant is able to view projects that are not relevant to them 2. Applicant can view projects that are hidden	✅

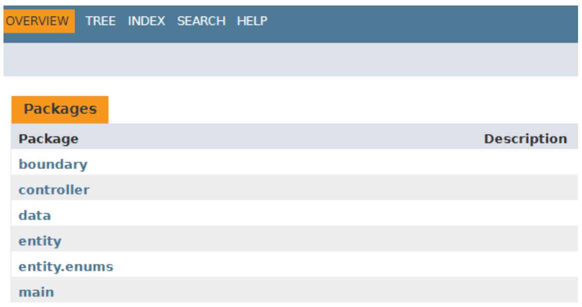
		- Based on the project's visibility status		
3.2	Project Application	Applicants can only apply to projects - Based on their age and marital status - Based on the project's visibility status	1. Applicant is able to apply for projects that are not relevant to them 2. Applicant can apply for projects that are hidden	✅
3.3	Single flat application	Applicants can only apply for 1 flat	1. Applicant can apply for more than 1 flat	✅
3.4	View Application Status after the project is hidden	Applicants can view status of their applications even if the project is hidden	1. Applicant cannot view application when the project is hidden	✅
3.5	Book flat	Applicant can only book a flat after successful application	1. Applicant can book flat regardless of application status	✅
3.6	Enquiries	Applicant can - Create an enquiry - View the enquiry - Edit the enquiry - Delete the enquiry	1. Applicant cannot create, view,edit or delete their enquiry	✅
4 - HDB Officer				
4.1	Has all capabilities of an Applicant	HDB Officer has access to all features of an Applicant	1. HDB Officer does not have applicant features such as applying for flat,etc	✅
4.2	Register to join a project	HDB Officer can register to join a project if - he has not applied for and will not apply for - has no conflicting projects dates	1. HDB Officer is able to register to join a projet he has applied as an applicant 2. HDB Officer is able to register to join a project with conflicting dates	✅
4.3	See status of project registration	HDB Officer can see the status for the project they have registered for	1. HDB Officer cannot see the status	✅
4.4	Profile shows list of projects	HDB Officer's profile shows all the projects he is registered for	1. HDB Officer's profile does not show the projects he is registered for	✅
4.5	View project details after the project is hidden	HDB Officer is able to view the details of his project even if it is hidden	1. HDB Officer is not able to view the details	✅
4.6	Cannot edit project details	HDB Officer should not be able to edit any project's details	1. HDB Officer is able to edit project's details	✅

4.7	View and Reply enquiries	HDB Officer is able to view and reply enquiries of projects that he is handling	1. HDB Officer is not able to view or reply	✅
4.8	Book flat	HDB Officer is able to process flat booking for an application and update booking status	1. HDB Officer is not able to process flat booking 2. HDB Officer is not able to update booking status	✅
4.9	Generate receipt	HDB Officer is able to generate receipt once flat is booked	1. HDB Officer is not able to generate receipt	✅
5 - HDB Manager				
5.1	Cannot apply for any BTO project as an Applicant	HDB Manager is not able to apply for project as an applicant	1. HDB Officer is able to apply for a project	✅
5.2	Manage project listings	HDB Manager is able to create (with all the required details), edit and delete project listings	1. HDB Officer is not able to create, edit or delete 2. HDB Officer does not include all the details needed	✅
5.3	Handle projects	HDB Manager can only handle projects with no conflicting projects dates	1. HDB Officer is able to handle conflicting projects	✅
5.4	Toggle project's visibility	HDB Manager can toggle project visibility	1. HDB Officer cannot toggle the visibility	✅
5.5	View projects	HDB Manager is able to view all created projects	1. HDB Officer is not able to view all projects (e.g. projects created by other HDB Officers)	✅
5.6	Filter projects	HDB Manager can filter projects that he is handling	1. HDB Officer cannot filter projects	✅
5.7	View HDB Officer Registrations	HDB Manager can view registrations of HDB Officers (both pending and approved)	1. HDB Manager cannot view the registrations	✅
5.8	Approve / Reject HDB Officer registrations	HDB Manager can approve or reject registrations of HDB Officers	1. HDB Manager cannot approve or reject	✅
5.9	Approve / Reject Applicants BTO Application	HDB Manager can approve or reject applicant's BTO applications	1. HDB Manager cannot approve or reject	✅
5.10	Approve Applicants BTO Application's withdrawal	HDB Manager can approve applicant's withdrawal of their BTO application	1. HDB Manager cannot approve	✅

5.11	Generate reports	HDB Manager can generate a report of all the applicants with their flat bookings (with filter)	1. HDB Manager cannot generate report 2. HDB Manager cannot filter	✅
5.12	View and reply to enquiries	HDB Manager can view and reply to all enquiries	1. HDB Manager cannot view or reply to enquiries	✅

6 - Admin				
6.1	Generate reports	Admin can only generate and view reports	1. Admin cannot generate reports 2. Admin cannot view reports	✅
6.2	Forgot password	Admin has the permission to reset one's password	1. Admin does not have the permission to reset password	✅
6.3	Cannot apply for any BTO Project	Admin cannot apply any project	1. Admin can apply for a BTO Project as an applicant	✅

Chapter 6: Documentation (Javadoc)



Package	Description
boundary	
controller	
data	
entity	
entity.enums	
main	

- Refer to javadoc folder in the zipped folder to view the javadoc

Chapter 7: Reflections & Challenges

What went well?

We have successfully implemented a Command Line Interface (CLI) based BTO Management System that supports complex user flows across three roles: Applicant, HDB Officer, and HDB Manager. Core functionalities such as BTO Application, officer registration, flat booking, enquiry handling, and report generation were implemented with modularity in mind.

Our use of the Model View Controller architecture also helped keep the code organised and extensible. The implementation of business rules (e.g., marital/age eligibility, officer registration conflicts) and our role-based access control also added robustness and realism to our system.

What could be improved?

- Time management
    - For our project, we have to rush the completion and leaving it to the last week to complete due to examinations.
- Learning Visual Paradigm
    - For our project, as instructed as parts of the assignment, we realise the difficulty in the learning curve and navigation of the platform. We felt that there could be an easier platform to use instead, such as LucidChart, draw.io

Lessons learned about OODP

- Code reusability
    - User class: Allows child classes (Applicant, HDB Officer and Manager) to inherit common functionality
    - Uses enum to manage the status system (Pending, Successful, Unsuccessful, Booked)
- Purpose of Class and Sequence Diagram
    - Creating a Class diagram facilitates communication between classes and provides a clear overview of the system - highlighting the concepts where inheritance or polymorphism should be applied
    - Creating a Sequence Diagram helps us to think of the entire process flow and identify which classes are needed to interact with one another. This prevents potential design errors from being implemented that would have been time-costly to fix.

Chapter 8: Appendix

- GitHub Link: https://github.com/harish950/SC2002_OOP_FCS3_GROUP_3.git
- References: NIL