NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC2002 OBJECT ORIENTED DESIGN & PROGRAMMING
BUILD-TO-ORDER (BTO) MANAGEMENT SYSTEM
Report of Project Structure Design & Functionality
AY24/25 Sem 2 | SDDA, Group 1

NAME	MATRIC NO.
AUSTIN TEO YUAN XUAN	U2310167H
MUHAMMAD NUR HAZIQ BIN ABDULLAH	U2310792G
NATASHA LYE SHI LING	U2340578H
SENGUTTUVAN NAVIENA	U2423943L
SINFUEGO SUMMER DAPHNE TAN	U2311061B

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 forthe assessed work. In addition, disciplinary actions may be taken.

Name	Course	Lab Group	Signature / Date
AUSTIN TEO YUAN XUAN	BCG	**SDDA**	23 April 2025
MUHAMMAD NUR HAZIQ BIN ABDULLAH	BACF	**SDDA**	23 April 2025
NATASHA LYE SHI LING	BCG	**SDDA**	**23 April 2025**

SENGUTTUVAN NAVIENA	DSAI	**SDDA**	 23 April 2025
SINFUEGO SUMMER DAPHNE TAN	BACF	**SDDA**	 23 Apr 2025

Chapter 1: Requirement Analysis & Feature Selection

1.1 Understanding the Problem and Requirements

Before getting started on the project, our team met up on Zoom to conduct a careful read-through and discussion of the document detailing the assignment. We read the specifications of the project line by line, noting down the requirements, user capabilities, constraints, and other important points.

Hence, we were able to identify the main problem domain: **a Build-To-Order (BTO) flat application management system via a command line interface (CLI) that must handle multiple user roles with different abilities and permissions, including the processes of application, approval, enquiries management, and more.**

Afterwards, we conducted a thorough requirement analysis outlining the explicit requirements mentioned and implicit requirements our team inferred from it:

1. **Non-Functional Requirements**

(i) Usability
- CLI must be intuitive and menu-driven where possible
- Clear prompts and error messages
- Clear confirmation messages
- Accessibility
- Input validation and help

(ii) Performance
- Fast response time for commands
- Efficient data retrieval for large numbers of applications
- Efficient data access
- Error and timeout handling
- Startup Performance

(iii) Security
- User authentication (Username-NRIC/Password-Default Password)
- Role-based access control

- Data Privacy
- Secure Storage
- Software Integrity

(iv) Scalability

- Design must allow for more flat types
- Command extensibility
- Scalable reporting
- Logging support

1.2 Deciding on Features and Scope

The BTO Management System (BTOMS) is a command-line-interface(CLI) based application designed to manage the end-to-end workflow to BTO flat applications.

The core features of this BTO flat application management system include:

- User registration and secure login with role-based access control for different users such as Applicants, BTO Officer and BTO Manager.
- Applicants can view available BTO projects, submit their application for a flat, check their application status, or withdraw their application anytime.
- The officers and managers can view and respond to enquiries made by the applicants and reject/approve the applications made.

The project scope focuses on:

- Building a functional CLI-based application that handles the core BTO processes and role - specific operations.
- The system is designed to be a lightweight yet secure, user-friendly source that helps with planning the project's roadmap.
- It extends to include CSV report exports, improved sessions and environment management,integration with national services like SingPass and region specific flat schemes.

Overall, BTOMS is designed to be modular and extensible, evolving from a secure and efficient CLI tool into a robust, enterprise-level BTO management solution.

Chapter 2: System Architecture & Structural Planning

2.1 Planning the System Structure

Planning the system structure is a crucial step for making the BTOMS clean, scalable, and maintainable. We started off by drawing class diagrams to get a better understanding of the relation between classes such as inheritance , dependency and the interactions between them. We finally decided to retain the Entity - Boundary - Controller pattern for our BTO management system project.

1. **Entity** : It encapsulates core business objects with clear data ownerships such as User, Application, Project and Enquiry. This includes characteristics such as identity, encapsulation and are designed to be saved via repositories.

2. **Boundary**: It is the layer that handles interactions between users and the system logic. It uses a structured CLI and is responsible for presenting prompts and reading user input. It also displays responses from the Controller layer(such as displaying project listings or application status) and routes the user to different parts of the system based on their roles and choices.

3. **Controller**: This layer coordinated between user interactions and data manipulation. It acts as the intermediary between Boundary (UI) and the Entity+Repository layers. It is mainly responsible for receiving requests from the CLI and validating the input data. It communicates with the repositories such as saving applications or fetching project files. It also performs validation to ensure the project exists before applying.

4. **Repository**: This is a design pattern that abstracts the data access logic. It helps to keep the entities and controlled agnostic of how and where the data is stored. It is mainly responsible for encapsulating data access logic, writes any changes to the CSV file, and maintains consistency of objects.

5. **Dependency**: A dependency is when one class relies on another function. It often practices to pass dependencies via constructors and it avoids tight coupling by not using globe state or static utility classes. It encapsulates external systems by ensuring repositories wrap the files and it shields other layers.

2.2 Reflection on Design Trade-offs

In designing the BTO Management System in the beginning, several design trade-offs were made to balance development with long term maintainability. Overtime, the design shifted towards dependency which offered lightweight storage without complexity. The Entity-Boundary-Controller (EBC) pattern was chosen over Model-View-Controller (MVC) pattern as it better suits a CLI based system as it provides clearer distinctions between user interactions and data models. Furthermore, controllers were split by domain (e.g ApplicationController, ProjectController etc) to enhance the coordination. These trade-offs reflect a thoughtful balance and lays a solid foundation for future enhancements and scalings.

Chapter 3: Object-Oriented Design

3.1 Class Diagram

(See Attached File)

3.2 Sequence Diagrams

(See Attached File)

3.3 Application of OOD Principles (SOLID)

3.3.1 Single Responsibility Principle

SRP states that a class should have only one reason to change, meaning it should only have one job or responsibility. This improves maintainability and keeps the code modular and focused. Adhering to SRP makes the system easier to maintain and test, reducing the risk of unintended side effects when modifying the code. This can be seen from the various ViewerClasses, whose single responsibility is to view the details of different entities, like Application, Project, Enquiries or OfficerRegistration.

3.3.2 Open/Closed Principle

A module should be open for extension but closed for modification, allowing for the addition of new functionality without changing existing code. OCP can be implemented through abstraction, inheritance, and polymorphism to enable safe scalability.

In our system design, the Open-Closed Principle (OCP) is applied through the use of an abstract class, FileBasedRepository<T>, which encapsulates common file-handling logic such as loading, saving, and directory management. Concrete repository classes like ApplicantRepository, HDBOfficerRepository, and ProjectRepository extend this base class to implement domain-specific methods without modifying shared functionality. This structure allows the system to be easily extended by introducing new repositories while keeping the core file logic unchanged.

3.3.3 Liskov Substitution Principle

LSP states that objects of a superclass should be able to be replaced with objects of a subclass without affecting the correctness of the programme. LSP is applied through the consistent use of ApplicationStatus enum and the behaviour of the Application class. Methods like updateStatus(AppilcationStatus) rely on any valid status values behaving predictably, allowing safe substitution without affecting correctness. Similarly, ApplicationRepository interacts with Application objects through a consistent interface, ensuring they can be used interchangeably in the system. This upholds LSP by allowing components to operate on abstractions without needing to know the specific internal state of each object.

3.3.4 Interface Segregation Principle

A class should not be forced to implement methods it does not need. A class should have small, focused interfaces rather than large, monolithic ones. This is evident from our PasswordPolicy and PasswordUI interfaces, segregating the requirements of the password and the interface it uses to read the password.

3.3.5 Dependency Injection Principle

High level modules should not depend on low-level modules- both should depend on abstractions. Instead of directly depending on the concrete class to perform some operation, we can depend on the interfaces, which are less likely to be changed.

DIP is applied through separation of controllers and repositories. Instead of having controllers depend directly on concrete repository implementations, each repository class (ie ApplicationRepository and OfficerRegistrationRepository) inherits from a shared abstract superclass or implements a common Repository interface. The abstraction layer ensures that high-level modules are not tightly coupled to specific low-level modules.

Chapter 4: Implementation

4.1 Tools Used

For the development of our programme, we used Java 17 and Visual Studio Code (VSC) as our IDE. The Version Control System we utilised is GitHub to facilitate collaborative coding.

4.2 Sample Code Snippets

4.2.1 Encapsulation

```java
public class Applicant implements User {
    private static final long serialVersionUID = 1L;

    /**
     * NRIC of the applicant (serves as unique user ID).
     */
    private String nric;

    /**
     * Name of the applicant.
     */
    private String name;

    /**
     * Password for this applicant's account.
     */
    private String password;
```

```java
     * Returns the NRIC of the applicant.
     * @return Applicant's NRIC
     */
    @Override
    public String getID() {
        return nric;
    }

    /**
     * Returns the name of the applicant.
     * @return Applicant's name
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * Returns the password for this applicant's account.
     * @return Applicant's password
     */
    @Override
    public String getPassword() {
        return password;
    }

    /**
     * Sets a new password for this applicant's account.
     * @param password The new password
     */
    @Override
    public void setPassword(String password) {
        this.password = password;
    }
```

Figure 4.2.1.1. Applicant.java

An example of how we demonstrated Encapsulation in our implementation is in the User classes such as Applicant, HDBOfficer, and HDBManager. In these classes, we declared certain **attributes** (such as nric, name, password, and others) as **private**, preventing direct access from outside the class. *Figure 4.2.1.1*, code from the Applicant class, shows an example of this.

We then employed **public getter and setter** methods that allow these private attributes to be read and modified, such as getID(), getName(), setPassword(), etc.

4.2.2 Inheritance

```java
public class Applicant implements User {
    private static final long serialVersionUID = 1L;
```

Figure 4.2.2.1. Applicant.java

```java
public class HDBOfficer extends Applicant {
    private static final long serialVersionUID = 1L;

    /** List of project names this officer is currently handling */
    private List<String> handlingProjects = new ArrayList<>();

    /** Registration status for project handling */
    private RegistrationStatus registrationStatus;

```

Figure 4.2.2.2. HDBOfficer.java

In the assignment brief, it was mentioned that HDBOfficer possesses all Applicant capabilities, hence we designed Applicant as a **superclass** and HDBOfficer as its **subclass**. Through this class inheritance, HDBOfficer inherits shared attributes and methods from Applicant such as nric, name, setPassword(), getAge(), etc. without having to redefine them, while still allowing for HDBOfficer to introduce officer-specific attributes and methods, e.g. registrationStatus.

4.2.3 Polymorphism

```java
public interface User extends Serializable {

    /**
     * Returns the unique identifier (NRIC) of this user.
     * @return the NRIC of the user
     */
    String getID();

    /**
     * Returns the name of this user.
     * @return the user's name
     */
    String getName();

    /**
     * Returns the password for this user's account.
     * @return the user's password
     */
    String getPassword();
```

Figure 4.2.3.1. User.java

```java
    /**
     * Returns the NRIC of the applicant.
     * @return Applicant's NRIC
     */
    @Override
    public String getID() {
        return nric;
    }

    /**
     * Returns the name of the applicant.
     * @return Applicant's name
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * Returns the password for this applicant's account.
     * @return Applicant's password
     */
    @Override
    public String getPassword() {
        return password;
    }
```

Figure 4.2.3.2. Applicant.java

We also applied polymorphism through our User interface which is implemented in the classes Applicant and HDBManager. The User interface contains **abstract methods** such as getID(), getName(), etc. (*Figure 4.2.3.1*) and each implementing class **overrides** with their **own specific implementations** of these methods as seen in *Figure 4.2.3.2*. Hence, at runtime, Java determines which method to call depending on the object's class.

4.2.4. Interface use

```java
public interface User extends Serializable {

    /**
     * Returns the unique identifier (NRIC) of this user.
```

Figure 4.2.4.1. User.java

```java
public class Applicant implements User {
    private static final long serialVersionUID = 1L;
```

Figure 4.2.4.2. Applicant.java

We also applied interface use in our program by **defining the User interface** and **implementing** it across the Applicant and HDBManager classes. As also mentioned in 4.2.3., the User interface declares abstract methods, requiring implementing classes to define them for their own uses. The use of this interface makes our code easily maintainable and scalable.

4.2.5 Error handling

```
151    public boolean batchUpdate(List<Application> applications) {
152        try {
153            applications.forEach(app -> {
154                entities.removeIf(a -> a.getApplicationId().equals(app.getApplicationId()));
155                entities.add(app);
156            });
157            return true;
158        } catch (Exception e) {
159            return false;
160        }
161    }
```

Figure 4.2.5.1. ApplicationRepository.java

Our use of error handling mechanisms is demonstrated in the ApplicationRepository class, with a **try/catch** block for the method batchUpdate() as shown in *Figure 4.2.5.1*. The method uses try to update the list of applications and if an exception occurs during the process, it catches the exception and returns false indicating the operation failed. This not only captures any potential errors in the code rather than letting the error crash during runtime, but also communicates the operation's success, or failure if an exception was caught.

Chapter 5: Testing

5.1 Test Strategy

We developed a comprehensive testing approach to ensure the correctness, functionality, and user experience of our BTO Management System. We mainly focused on **manual functional testing**. As our program is a CLI system, manual testing allowed us to simulate real user inputs through various test cases and observe outputs, from the perspective of each user type i.e. Applicant, HDBOfficer, and HDBManager (via test user accounts).

We derived test cases from the requirements identified in Chapter 1, taking into consideration expected or common user behaviours, as well as system constraints put into place. Moreover, we tested multiple edge cases to ensure data validation of our system. Lastly, since our system does not use a database backend, we manually verified data-related operations (e.g. user lists, enquiry lists) in the data files to ensure proper data retrieval and updates.

5.2 Test Case Table

TC	Scenario	Input	Expected Output	Actual Output
1	Valid user login	NRIC: T0222222A PW: password	"Login successful! Welcome, [user]." Directed to correct UI based on user type	As expected - Pass
2	Invalid user login (NRIC doesn't exist)	NRIC: S0000001A PW: password	"User not found. Please try again." Re-prompt for login details	As expected - Pass
3	Invalid user login (Incorrect password)	NRIC: T0222222A PW: wrongpw	"Incorrect password. Please try again." Re-prompt for login details	As expected - Pass
4	Change password (Incorrect)	Current PW: password New PW: pass2	"Password must have - 8+ characters - 1 uppercase	As expected - Pass

			- 1 lowercase - 1 digit -1 special symbol (@$!%*?&)" Re-prompt for new password	
5	Change password (correct)	Current PW: password New PW: Password123! Confirm PW: Password123!	"Password changed successfully!"	As expected - Pass
6	Project visibility based on user group/toggle (age, marital status, role)	User: Applicant T0222222A, Single, 23	Applicant is unable to view or apply to any projects (due to age and marital status)	As expected - Pass
7	Project application (eligible)	User: Officer T2109876H, Single, 36 Applying for 2-Room	"Application submitted successfully!" App can be approved/rejected from Manager's end Status: "Pending"	As expected - Pass
8	Project application (eligible)	User: Applicant T0444444A, Married, 21 Applying for 3-Room	"Application submitted successfully!" App can be approved/rejected from Manager's end Status: "Pending"	As expected - Pass
9	Project application (ineligible)	User: Manager, T8765432F, Single, 36 Applying for 2-Room	Applying to project as Applicant not available from Manager menu	As expected - Pass
10	Application withdrawal	User: Officer T2109876H, Single, 36 Withdrawal from 2-Room project	"Withdrawal request submitted successfully!" After manager approval, status: "Withdrawn"	As expected - Pass
11	Application for multiple projects	User: Applicant T0444444A, Married, 21 Applied for 3-Room, attempt to apply for 2-Room	Application not accepted "You already have an active application."	As expected - Pass
12	Applicant viewing capability with visibility toggle off	User: Applicant T0444444A, Married, 21 Applied for 3-Room, viewing same application	3-Room application details and status still accessible with visibility off	As expected - Pass
13	Project application (ineligible)	User: Officer, S5678901G Applying for 2-Room project Registered for same 2-Room	Applying to project registered to not available from Apply for BTO Project (as Applicant) menu	As expected - Pass
14	Officer project registration (eligible)	User: Officer, T1234567J Registering for 3-Room project	Registration accepted Registration can be approved/rejected from Manager end Status: "Pending"	As expected - Pass
15	Manager approving Officer registration	User: Manager, T8765432F Approving T1234567J's registration	"Registration successfully approved!" Updates project's Officer slots Registration status: "Approved" from Officer end	As expected - Pass
16	Officer edit capability	User: Officer, T1234567J Editing 3-Room project registered for	Unable to edit project details from menus	As expected - Pass
17	Officer viewing capability with visibility toggle off	User: Officer, T1234567J Viewing 3-Room project registered for	3-Room application details and status still accessible even with visibility off	As expected - Pass
18	Applicant enquiry capability	User: Applicant T0444444A, Married, 21 Submitting enquiry: "When will	Able to submit enquiry, modify it, and delete it	As expected - Pass

		construction begin?” Modifying enquiry, Remove enquiry		
19	Officer enquiry capability	User: Officer, T1234567J Accessing and responding to enquiry: “This incoming March.”	Able to view and respond to enquiry of registered projects	As expected - Pass
20	Manager enquiry capability	User: Manager, T8765432F Accessing and responding to enquiry: “Thank you for your patience.”	Able to view and respond to enquiry of managed projects	As expected - Pass
21	Manager approving application	User: Manager, T8765432F Approving T0444444A’s application	“Project successfully approved!” Application status: “Approved” from Applicant and Officer end	As expected - Pass
22	Applicant’s successful application (Officer responsibilities)	User: Officer, T1234567J Handling T0444444A’s successful 3-Room application	Update number of flat type to number remaining Update application status from “successful” to “booked” on Applicant end	As expected - Pass
23	Receipt generation for flat booking	User: Officer, T1234567J Generating receipt of T0444444A’s successful 3-Room application	Receipt generated, correctly displaying Applicant name, NRIC, age, marital status, flat type booked, project details	As expected - Pass
24	Manager project listing capabilities (create project - invalid date)	User: Manager, T8765432F Project: River Side, Sengkang, 3x 3-Room flats, 650000, 5 Officer slots Application date: 01/01/2025 - 01/06/2025	“Opening date must not be in the past.” Re-prompt for opening date	As expected - Pass
25	Manager project listing capabilities (create project - invalid date)	User: Manager, T8765432F Project: River Side, Sengkang, 3x 3-Room flats, 650000, 5 Officer slots Application date: 01/08/2025 - 01/06/2025	“Closing date must be after opening date.” Re-prompt for closing date	As expected - Pass
26	Manager project listing capabilities (create project - invalid price)	User: Manager, T8765432F Project: River Side, Sengkang, 3x 3-Room flats, 01/05/2025 - 01/12/2025, 5 Officer slots Price: -	“Invalid price” Re-prompt for price	As expected - Pass
27	Manager project listing capabilities (create project)	User: Manager, T8765432F Project: River Side, Sengkang, 3x 3-Room flats, 650000, 01/05/2025 - 01/12/2025, 5 Officer slots	“Project created successfully!” Project now visible to relevant group/users	As expected - Pass
28	Manager single project management	User: Manager, T8765432F Project: Breeze Residence, Novena, 2x 2-Room flats, 700000, 01/05/2025 - 01/05/2026, 3 Officer slots	Manager is unable to manage more than 1 project within same application dates “Failed to create project”	As expected - Pass
29	Manager project listing capabilities (modify existing project)	User: Manager, T8765432F Project: River Side, modifying date to 23/04/2025 - 01/12/202	“Project updated successfully!” Modified project details visible to relevant groups	As expected - Pass
30	Manager project listing capabilities (delete existing project)	User: Manager, T8765432F Deleting project: Bangers	“Project deleted successfully.” Project no longer visible to any group/user	As expected - Pass
31	Manager project listing	User: Manager, T8765432F	Project no longer visible except to Applicants	As expected -

	capabilities (toggle project visibility)	Toggle project “River Side” visibility to off	with ongoing applications and registered Officers	Pass
32	Manager project viewing capabilities	User: Manager, T8765432F	Can view all projects and project details	As expected - Pass
33	Manager report generation capability	User: Manager, T8765432F Project: River Side Filter: Marital Status = Married on Applicants Flat Type = 3-Room	Report generates all married applicants who applied for 3-room (for River Side) with correct information	As expected - Pass
34	Application for unavailable flat type	User: Applicant, T0111111A, Married, 25 Applying for 2-Room flat with 0 availability left	Application not accepted (due to 0 availability left) “Application failed. Check eligibility and availability.”	As expected - Pass

Chapter 6: Documentation

6.1 Javadoc

We utilised Javadoc to clearly and comprehensively document all classes, methods, attributes, and interfaces for enhanced readability and maintainability. Javadoc comments were added above each code definition to describe its functionality, purpose, and any other relevant details.

We generated the corresponding HTML Javadoc documentation and included these files in our deliverables folder. The HTML Javadoc can be used for easy browsing via web interface.

6.2 Developer Guide

Environment Setup:

- Java Development Kit: Java 17
- IDE: Visual Studio Code

1. Download the project repository (included in BTOMS main folder)
2. Open the project in Java IDE of your choice
    - Import as Java project
3. Compile all files within src folder
4. Run the Main class to start the application

Data files:

- users.dat: stores user information
- projects.dat: stores BTO project information
- applications.dat: stores application information

No external libraries, tools, etc. are needed for this application.

Chapter 7: Reflection & Challenges

All in all, our team was able to successfully implement the BTO Management System, directly adhering to our main problem domain and the requirements identified. We thoroughly planned our system architecture, structure, and design whilst balancing trade-offs. Early on, we effectively separated our system into boundary, controller, and entity layers, which made it much clearer and easier to work on.

We applied key Object-Oriented principles in our system, from the UML and Sequence diagrams, to maintaining SOLID principles in our design. In our code, we demonstrated OOP principles such as encapsulation, inheritance, polymorphism, and more. Our use of inheritance and interfaces was particularly invaluable, especially in the User classes given their shared and differing sets of attributes and methods. Also essential was testing with varied test cases covering key functionalities, allowing us to discover and rectify initial bugs.

Areas for improvement include our file-based data management system, which has limitations in data persistence, data integrity, and concurrent access that could be addressed with structured file storage solutions. Moreover, while we implemented error handling throughout, a more consistent approach with custom exception types would make error reporting easier.

Our team was collaborative from start to end, with consistent use of Github for collaborative code review. We assigned work based on each member's individual strengths, leading to meaningful individual contributions, including but not limited to: Austin led the coding, identifying and defining the project management system. Natasha created the Sequence Diagram. Summer created and carried out test cases for the code. Haziq created the UML Diagrams. Naviena worked on User classes.

This project has deepened our understanding of OODP principles beyond theory, seeing such principles come into play with BTOMS and its implementation. We learned the importance of UML and Sequence Diagrams for planning, as it greatly helped us in our execution. Another specific instance of what we learned is when to use class or interface inheritance, as we effectively utilised both in our system. We also bolstered our understanding of principles such as polymorphism and how it adds flexibility and extensibility to code. What we have learned is not limited to this, as this hands-on project has given us insight into OODP that will prove indispensable for future OODP projects. Nonetheless, beyond technical skills, this assignment has reinforced the value of collaborative coding and effective communication within a team.

Chapter 8: Appendix

GitHub Link: https://github.com/austintyx/SC2002-BTO-Management-System.git