SC2002 - Object Oriented Design & Programming

Semester 2, AY2023/2024

Lab Group: SDDB

GitHub Repository Link: https://github.com/Sai-Ajay/SC2002-OOP-FOMS-2.0

Declaration of Original Work for CE/CZ2002 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 honoured 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
Sally Ngui Yu Ying	SC2002	SDDB	Sally 26/4/24
Shingamu Sai Ajay	SC2002	SDDB	Ajay 26/4/24
Zhang Xinyang	SC2002	SDDB	Xinyang 26/4/24
Shen Jia Cheng	SC2002	SDDB	Jia Cheng 26/4/24
Sharanya Basu	SC2002	SDDB	Sharanya 26/4/24

1 Design Considerations

1.1 Overview

Our Fastfood Ordering Management System (FOMS) was designed with low coupling and high cohesion in mind. We utilised a modular approach that would adhere to the different Object-Oriented concepts and SOLID design principles. Our group also implemented MVC architecture by splitting the code into the **Model**, **View** and **Controller** packages with additional packages like **Stores**, **Services**, **Enums,** and **Interfaces**.

Through such a design, our code can be easily maintained, readable, and extensible. The adherence to OO concepts and SOLID design principles also ensures that the modules and classes in our FOMS can be developed, tested and debugged individually, which enhances the development efficiency and quality of the code.

1.2 Assumptions

We assumed that each branch's staff must strictly adhere to the specific quotas stated in the requirements. For example, if a branch has 9 to 15 staff, there must be 3 managers. This means that with 3 managers, the system will prevent the reduction of staff below 9. Similarly, with 9 to 15 staff, the system will also prevent the reduction of managers below 3. Conversely, if a branch wants to increase staff from 8 to 9, it must first add a third manager as well.

We also assumed that for every branch, if there are employees there, there must be at least one branch manager, but it is not necessary to have staff. This means that for every new branch created, a branch manager must be the first one to be added to that branch before other staff can be added. Similarly, if employees were transferred out of the branch, the last remaining employee has to be a branch manager.

1.3 OO Concepts

1.3.1 Encapsulation

Encapsulation is about bundling the attributes and methods into a single class, and protecting the internal state of the object. It does this through data hiding with access modifiers, where the inner workings of the object are hidden with keywords such as "private" and "protected". For example, the attributes of our classes are all made private or protected, and can only be accessed through methods of the class. This does not only enhance security and integrity, but also promotes a clear and maintainable code structure.

1.3.2 Abstraction

Abstraction simplifies the interaction with complex code by exposing only the essential features and concealing detailed implementations. Our group applied this by using abstract classes and interfaces, which hides the details of the concrete classes. All classes in the packages **Services** and **Views** implement specific interfaces to present only the necessary portions of our FOMS. Our abstract **User** class also serves as a foundational template that abstracts away the implementation specifics of the concrete **BranchUser** and **Admin** classes.

1.3.3 Inheritance

Inheritance allows a class to inherit the characteristics from their parent class, which promotes reuse of the code and enhances logical structure of the FOMS. **BranchUser** and **Admin** are inherited from the abstract **User** class. **Admin** inherits all the methods of the **User**, while **BranchUser** inherits the methods from **User** and has additional get and set methods for BranchID specific to branch users.

1.3.4 Polymorphism

Polymorphism allows the classes to be treated as instances of their parent classes instead of their actual class, resulting in multiple forms and behaviours. All the classes in the **Views** and **Services** packages implement specific interfaces and provide concrete implementations for the methods in the interface. For example, both **SerialDataService** and **CSVDataService** implements the **IFileDataService** interface, with each providing their own behaviours for the methods to import and/or export files based on the data type, which is an example of overriding.

1.4 Design Principles

1.4.1 Single Responsibility Principle (SRP)

To maintain high cohesion, each class is responsible for a specific task. This can be observed across all packages, with each package having a particular responsibility.

Stores	The classes handle the global state of the application, ensuring synchronisation of the data files stored across sessions and ensuring data persistence. For instance, **PasswordStorage**

	manages the global state of passwords and updates them after every change, and **BranchMenuItemStorage** maintains the global details of menu items specific to each branch.
Models	**Models** contain all entity classes and demonstrate their relationships. Each entity class, such as **BranchUser** and **Admin**, is solely responsible for its designated tasks.
Controller	The classes guide user actions based on their inputs, and manage the application's flow and user experience through **Services** and **Views**. Each class is responsible for a specific main page; for example, **LoginController** oversees the staff login process, and **CustomerController** controls customer experience.
Views	The classes present specific information requested by the user relevant to a particular aspect of the system. For instance, **BranchUserView** displays all the details of Staffs and Managers, such as their age and allocated branch, while **OrderStatusView** provides updates on the status of a particular order.
Services	The classes execute specific business logic functionalities by interacting with the database. **CSVDataService** manages the import and export of data to and from CSV files for various data models, and **LoginService** facilitates user authentication, login and password modifications.
Enums	The enumerations are sets of named constant categories standardised throughout the application. They include **Role**, **Gender**, **OrderStatus** and **OrderType**.
Interfaces	The interfaces are implemented by concrete classes and have responsibilities similar to the concrete classes that implement them. Some examples are **ICustomerService**, **IAdminService** and **IStaffService**.

1.4.2 Open-Closed Principle (OCP)

OCP ensures that classes are open for extension but closed for modification, allowing adding of new functionalities without changing the existing code. To maintain loose coupling and enhance extensibility, interfaces and abstract classes were used.

Interfaces	Interfaces ensure each component adheres to a defined contract, facilitating easy modifications, additions or removals of classes without significantly impacting the application. For example, **IFileDataService** is implemented by both **CSVDataService** and **SerialDataService** to handle

	the import and/or export of CSV and serialised data formats respectively. If the system needs to accommodate additional file types in the future, new classes such as '**XMLDataService**' for XML file type can be introduced, implementing the same interface. This is consistent across all classes in the packages **Views** and **Services**, each implementing a specific interface.
Abstract Classes	Abstract classes allow for extensibility through inheritance. For example, the **User** abstract class is extended to specific user types – **Admin** and **BranchUser**. This allows for creation of new subclasses for new user types, such as '**RegionalManager**', without altering the fundamental operations in User. This ensures new subclasses can be easily integrated into the system and maintain the integrity of the application.

1.4.3 Liskov Substitution Principle (LSP)

We ensured that derived classes must have pre-conditions no stronger and post-conditions no weaker than the base class method. This is so that the subclasses are substitutable for the base class without affecting the program's functionality. This is evident in how **Admin** and **BranchUser** subclasses provide specific implementations for methods like 'getName' and 'getRole' without introducing stricter pre-conditions not present in base class **User** or weaker post-conditions than **User** .

1.4.4 Interface Segregation Principle (ISP)

ISP is implemented by creating many client-specific interfaces instead of one general purpose one so that classes do not depend on interfaces they do not use. For example, each user role, Customer, Admin, Manager and Staff, has a specific dedicated interface, namely **ICustomerService**, **IAdminService**, **IManagerService** and **IStaffService** respectively. This ensures each role only has access to the methods they require for their designated tasks. This minimises unnecessary dependencies and enhances system security.

1.4.5 Dependency Inversion Principle (DIP)

We applied DIP to ensure that high-level modules do not depend on low-level modules but both depend on abstractions instead. This was achieved through the use of interfaces and abstract classes.

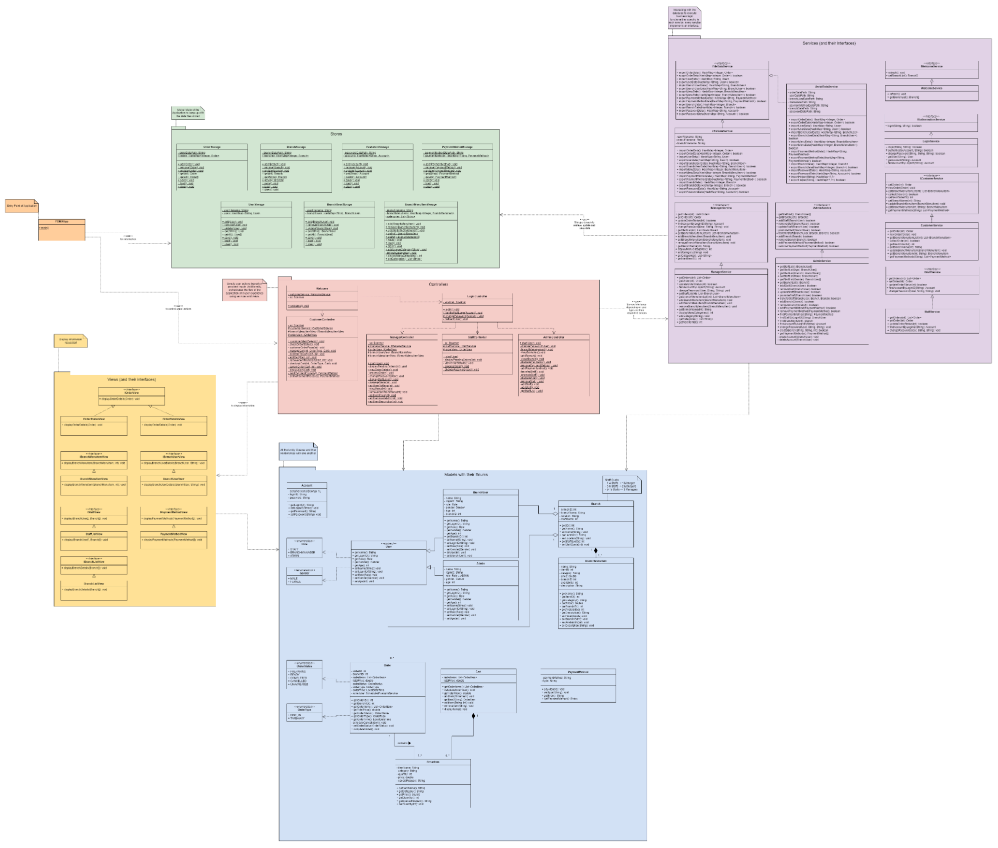
Interfaces	Each class in **Views** and **Services** implements a specific interface. For example, in the **Views** package, **IBranchUserView** is implemented by **BranchUserView**. In the **Services** package,

	interfaces like **ICustomerService** and **IAdminService** are implemented by **CustomerService** and **AdminService** respectively. This ensures the classes in **Controllers** and **Stores** interact with services solely through interfaces, decoupling the system's operations from specific implementations.
Abstract Classes	Similarly, by deriving specific subclasses **Admin** and **BranchUser** from base class **User**, we maintain a separation between the abstractions and concrete implementations. Other classes depend on abstract class **User** instead of the subclasses.

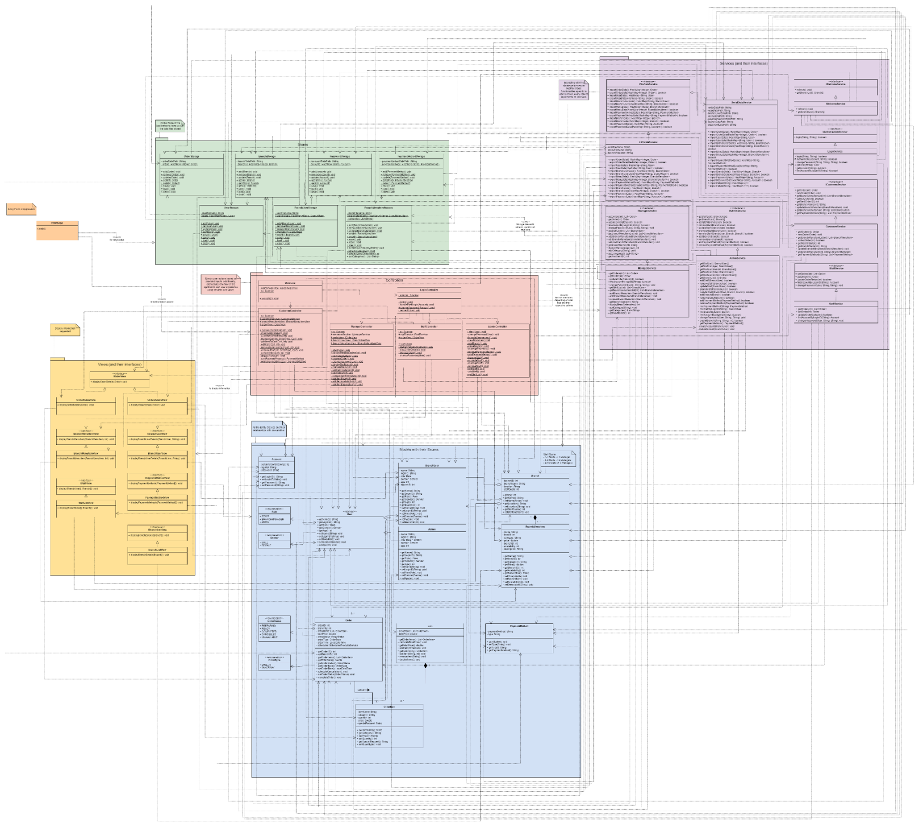
2 UML Class Diagram

Our UML Class Diagram for the FOMS Application illustrates the relationships between different Entity, Control and Boundary classes. The classes are grouped by their packages for easier understanding. We have identified the relationships at package level and class level respectively. Clearer images of our diagrams can be found in the folder.

2.1 Package-Level UML



2.2 Class-Level UML



3 Testing

Test Case	Test Description	Expected Outcome	P/F
1	Add a new menu item with a unique name, price, description, and category.	New menu item with the assigned name (ice cream), price (1.20), description (vanilla flavour), and category (dessert) is reflected under "View Menu"	P
2	Update the price and description of an existing menu item and verify that the changes are reflected in the menu.	Item "3PC set meal" with updated price (from $9.90 to $10) and description is reflected under "View Menu".	P
3	Remove an existing menu item and verify that the menu item is no longer available.	Removed "3PC set meal" and it is no longer shown in the Branch Menu on Manager's side and Customer's side.	P

4	Place a new order with multiple food items, customise some items, and choose takeaway option and verify that the order is created successfully.	Order placed with “FRIES” (with Special Request of “Less salt”) x3, and chicken nuggets x1, chose takeout option and checked out cart. “Order Status” is reflected as “PREPARING” under “Check Order Status” on Customer’s side and “Display pending orders” on Staff’s side.	P
5	Place a new order with the dine-in option and verify that the order is created with the correct preferences.	Order placed with “FRIES” (with Special Request of “Less salt”) x3, and chicken nuggets x1, chose dine-in option and checked out cart. “Order Status” is reflected as “PREPARING” under “Check Order Status” on Customer’s side and “Display pending orders” on Staff’s side. Order preferences of “Less salt” is also reflected.	P
6	Simulate a payment for using a credit/debit card and verify that the payment is processed successfully.	Receipt is shown after entering card payment details (Name, Card Number, Expiry Date and CVV) to indicate successful processing and order placement.	P
7	Simulate a payment using an online payment platform (e.g. PayPal) and verify that the payment is processed successfully.	Receipt is shown after entering online payment details (Email and Password) to indicate successful processing and order placement.	P
8	Track the status of an existing order using the order ID and verify that the correct status is displayed.	Order status is verified as one of the following: PREPARING/READY/COLLECTED/CANCELLED.	P
9	Login as a staff member and display new orders and verify that the staff can see all the new orders in the branch he/she is working.	After logging in as staff, new orders at the branch with “Order Status” as	P

		"PREPARING" can be reflected under "Display pending orders".	
10	Process a new order, updating its status to "Ready to pickup." and verify that the order status is updated correctly.	"Order processed successfully" displayed after clicking "Process order" with Order ID = 1. "Order Status" is now updated from "PREPARING" to "READY" under "Check Order Status" on Customer's side and "View order details" on Staff's side.	P
11	Login as a manager and display the staff list in the manager's branch and verify that the staff list is correctly displayed.	Displays list of all staff and managers in that branch under "Display Staff List in Branch".	P
12	A manager should be able to process order as described in Test Case 10	Same outcome as Test Case 10.	P
13	Close a branch and verify that the branch does not display in Customer's Interface	"Branch closed successfully" displayed when closing branch "JP". Branch does not show up on the Customer's side.	P
14	Login as an admin and display the staff list with filters (branch, role, gender, age) and verify that the staff list is correctly filtered.	Correctly displays when selecting each filter, shows no staff records if there is no staff with that particular age.	P
15	Assign managers to branches with the quota/ratio constraint and verify that managers are assigned correctly.	Managers are assigned correctly based on the quota/ratio previously determined. Error message shown and managers cannot be assigned to branches if it violates the quota constraint.	P
16	Promote a staff to a Branch Manager and verify that the staff is promoted successfully.	"Staff successfully promoted to manager" message shown and "Role" is displayed as "BRANCHMANAGER" under "Display Staff list"	P

17	Transfer a staff/manager among branches and verify that the transfer is reflected in the system.	"Staff transferred successfully" message shown and transfer is verified with "Display Staff list" with "Branch" filter.	P
18	Place a new order, check the order status using the order ID, and collect the food and verify that the order status changes from "Ready to pickup" to "Completed."	Order with ID 2 added to pending orders and is processed successfully by staff. Customer keys in Order ID = 2 under "Check Order Status" and collects order. "Order Status" is now updated from "READY" to "COMPLETED" under "Check Order Status" on Customer's side and "View order details" on Staff's side.	P
19	Attempt to add a menu item with a duplicate name and verify that an appropriate error message is displayed.	Error message "Item already exists.".	P
20	Attempt to process an order without selecting any items and verify that an error message prompts the user to select items.	Error message "Your cart is empty. Please add items to your cart before checking out.".	P
21	Add a new payment method and verify that the new payment method is successfully added.	New payment method Paylah is reflected under "View Payment Methods" on Admin's side and successfully used by customer to make payment.	P
22	Open a new branch and verify that the new branch is added without affecting existing functionalities.	New branch NUS is reflected under "View Branches" on Admin's side and successfully used by customers to make orders and branch staff to process orders. Other functionalities remain unaffected.	P
23	Place a new order and let it remain uncollected beyond the specified Timeframe and verify that	After the order is processed by staff and remains uncollected beyond the timeframe, when customer keys in Order ID under	P

	the order is automatically cancelled and removed from the "Ready to pick up" list.	"Check Order Status", they cannot collect order. "Order Status" is now updated from "READY" to "CANCELLED" under "Check Order Status" on Customer's side and "View order details" on Staff's side.	
24	Attempt to log in with incorrect credentials as a staff member and verify that an appropriate error message is displayed.	With either wrong LoginID or password, the error message "Login failed" is displayed.	P
25	Log in as a staff member, change the default password, and log in again with the new password and verify that the password change functionality works as expected.	Staff is automatically prompted to change default password upon first login. When logging in again, password change works, and is able to persist throughout sessions.	P
26	Upload a staff list file during system initialisation and verify that the staff list is correctly initialised based on the uploaded file.	Uploaded and verified through filtering of characteristics (No Filter, Role, Gender, Branch, Age).	P
27	Perform multiple sessions of the application, adding, updating, and removing menu items and verify that changes made in one session persist and are visible in subsequent sessions.	Changes, such as new password, payment methods and branches, persist in subsequent sessions of rerunning of code.	P

4 Reflection

4.1 Difficulties

One challenge we faced was adhering to OCP to ensure extensibility without modification of existing code. This challenge was managed by employing interfaces and abstract classes, allowing us to extend functionalities without changing the source code. Another difficulty faced was implementing effective error handling without compromising the readability of our code. We overcame this issue by creating custom exception classes tailored to handle specific errors, such as **PasswordIncorrectException**. This allowed for easier troubleshooting and greater clarity.

4.2 Knowledge Gained

Throughout our application's development, we delved deeper into OOP and the SOLID principles, gaining invaluable insights into software design and best implementation practices. By utilising packages and adopting the MVC architecture, we honed our understanding of code organisation and visualisation through UML class diagrams. Implementing SOLID principles allows us to extend and maintain our application, ensuring its scalability and robustness. Moreover, we recognised the importance of establishing clear naming conventions, to facilitate smoother collaboration among team members.

Addressing user requirements also emerged as a pivotal aspect of our development journey. By thoroughly examining and understanding these needs, we were able to tailor our application to meet user expectations effectively. To maintain readability, we learned how to handle exceptions with additional classes while complying with the principles.

4.3 Further Improvements

Currently, the system prompts users to input any special requests that they have in a free-text format under 'Special Requests'. This can lead to unstandardised messages and requests may not be fulfilled accurately. To further improve FOMS, we can consider implementing a more structured meal customisation option by allowing them to make selections of given choices such as "less salt" or "extra cheese". Furthermore, a user profile system can be created to give customers the option to create an account. This allows them to save their preferences and more importantly, conveniently view past orders and track new orders without needing to remember their OrderID. The system can also store their payment details securely, streamlining the checkout process by reducing the need to enter their personal details every time they order. This improves user experience by offering greater convenience and efficiency.