






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 (CE2002 or CZ2002) | Lab Group | Signature /Date |
|-----------------|------------------------------|--------------|--|
| Lee Choonggi | SC2002 | FDAD | 26 April 2024  |
| Chester Chiow | SC2002 | FDAD | 26 April 2024  |
| Lee Pei Xin | SC2002 | FDAD | 26 April 2024  |
| Eng Yi Xuan | SC2002 | FDAD | 26 April 2024  |
| Shem Lau Wei Yi | SC2002 | FDAD | 26 April 2024  |

1. Design Considerations

1.1 Approach Taken

To ensure data security and data integrity, as well as enhance the user-friendliness of the data, we have taken some approaches: encapsulation, low coupling and high cohesion, error handling, and data persistence.

Firstly, to ensure the security of the data, we apply the concept of information hiding which ensures that implementation details of a particular function are not exposed to users. The methods in Customer, StaffWorker, ManagerWorker, and Admin classes mainly interact with the user to ask for inputs while checking for input errors. The actual implementation will be done in another class that contains data information such as Order, WorkerList and Menu. For example, users will be prompted to select food items from the menu by calling the addFoodItemsToCart function in the Customer Class. The backend operations of adding food items to the cart will be done in the Order class instead. In this way, Customer does not need to be concerned with the inner workings of the Order class. The Order class is responsible for its own updates on the cart.

Secondly, to ensure the ease of modification of the system, we want to ensure low coupling and high cohesion of the classes. This minimises dependencies between different parts of the system, and it allows for easier modification to a single area. For example, we avoid branch class to initialise ArrayList of Orders, Workers, Payments and Menu. Instead, we created OrderList, WorkerList, PaymentList and Menu class each with its own inner workings. These classes can act as independent entities and are not concerned with any changes made to other parts of the branch class. This helps avoid complicated relationships between classes and their access patterns. Hence, prioritising flexibility and maintainability helps facilitate seamless modifications to our system.

Thirdly, to handle unseen errors that may interrupt user experience, we use try-catch blocks to validate users' input as requested by the system. 'InputMismatchException' and 'IndexOutOfBoundsException' were used to catch invalid errors and users will be prompted to re-enter their options. We have also simplified the instructions and minimised complex inputs from users. Each option is identified by an integer label, requiring users to only select and enter the corresponding integer.

Lastly, to ensure data persistence, we make use of serialisation to allow us to convert objects into a byte stream, allowing us to store them on disk or transmit them over a network.

1.2 Principles Used

1.2.1 Single Responsibility Principle

A class should be responsible for solely one aspect of the system's functionality. We have applied this principle to AdminWorker Class (Figure 1). We classified admin's methods on the branch list, payment list and workers list into the different operation classes, OperationsOnBranchList, OperationsOnWorkerList, OperationsOnPaymentList, and OperationsOnBranchStatus. With this, each operation class is responsible for an attribute that AdminWorker has access to and only has a reason to change if there are changes in the attribute.

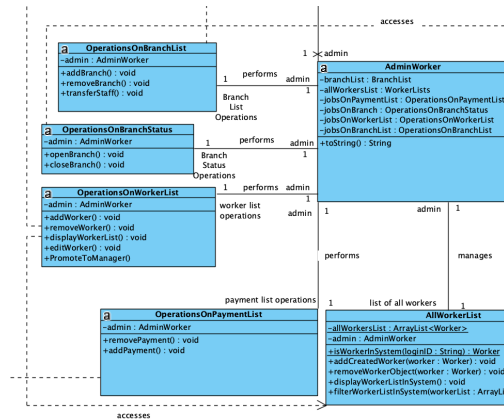


Figure 1. Relationships between AdminWorker Class and Operations classes

1.2.2 Open-Closed Principle

The features of entities being able to open for extension but closed for modification. We applied this principle in our FoodItem class. We created an abstract FoodItem class which consists of subclasses SetMeal, MainDish, Drink, and SideDish. The FoodItem class is closed for modifications but open for extension to the subclasses. For example, the Drink class extends and overrides the customiseFoodItem method. At the same time, if there are new types of food items offered in the branch, subclasses of FoodItem can be easily added by extending from FoodItem class, hence promoting extensibility. This principle is also applied in our filters and payment methods. If there are new filters and payment methods, new subclasses can easily be extended from the interfaces.

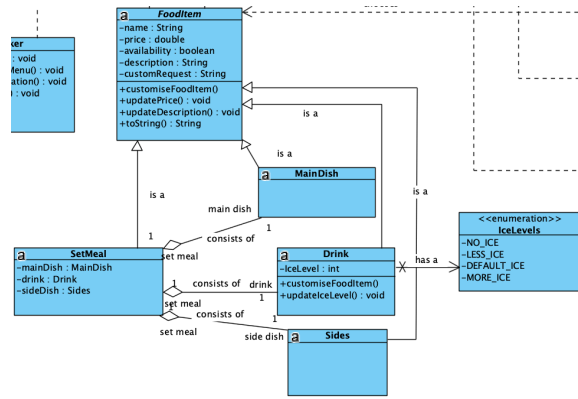


Figure 2. Relationships between FoodItem class, and its subclasses

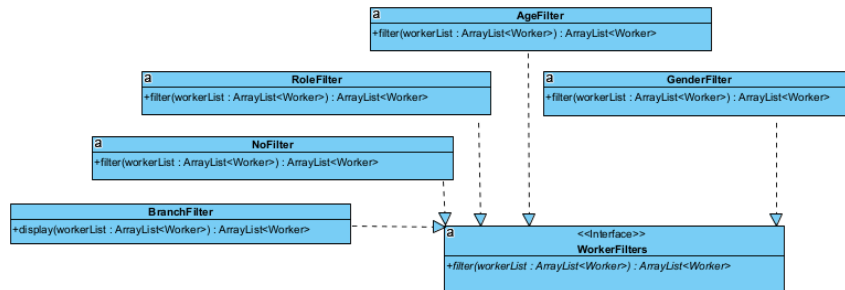


Figure 4. WorkerFilters

1.2.3 Liskov Substitution Principle

A subclass of a base class should continue to function properly when used in place of their base class. The StaffWorker class acts as the base class and the ManagerWorker class is the subclass (Figure 3). Noting that a manager should be able to perform all the tasks staff can do, we apply the Liskov Substitution Principle. The StaffWorker class can process and display orders. Using the concept of inheritance, the ManagerWorker has access to these functions as well. Hence, this means that a StaffWorker object can be substituted by a ManagerWorker object without causing any errors or unexpected behaviours.

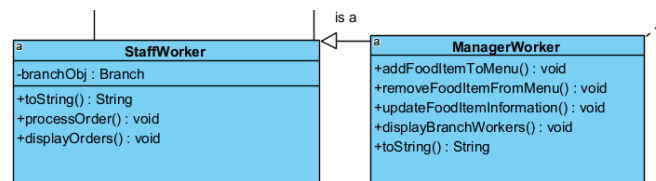


Figure 3. Relationships between ManagerWorker and StaffWorker

1.2.4 Dependency Injection Principle

High-level modules should not depend upon low-level modules, aiming to reduce coupling between classes. For example, to avoid direct dependency between CheckOutOrder class, a high-level module compared with Payments' subclasses, ShoppePay, PayWave, Paypal and NETS, low-level modules, we apply the principle by injecting a Payment interface. CheckOutOrder class now depends on the abstraction (interface), which has the lowest chance to be modified, instead of concrete classes.

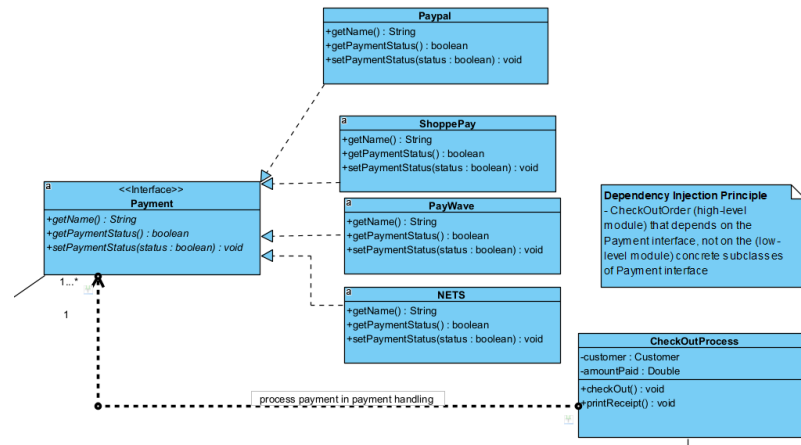


Figure 3. Relationships between CheckOutOrder, Payment, and its subclasses

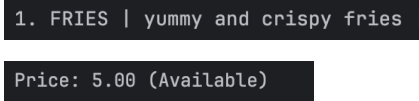
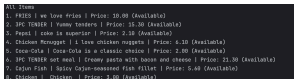
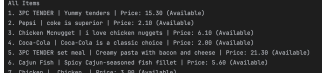
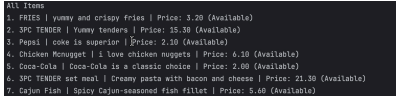
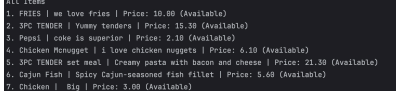
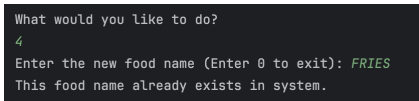
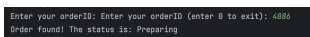
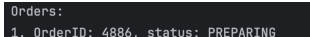
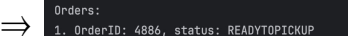
1.3 Assumptions Made


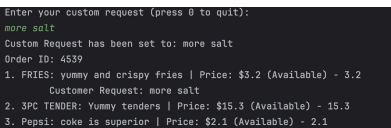
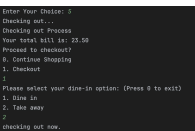
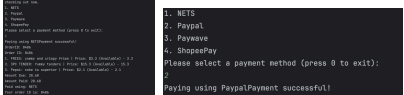
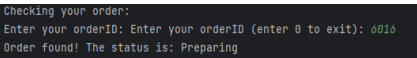
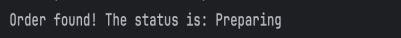
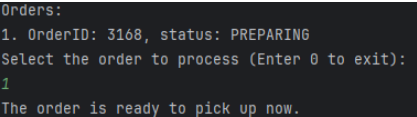
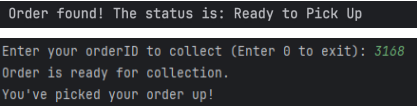
- One customer makes only one order – one order will be assigned with one unique order ID. This unique orderID allows customers to track their orders easily and avoid complexities during order processing.
- No time setting from the order being placed until it is ready to pick up – Staff workers are allowed ample time to prepare the order. Waiting times are not considered.
- Order is automatically cancelled and removed after a certain timeframe – applied a timer system to simulate real-time order handling. We assume that the customer will be able to collect the order before the timer runs out to prevent orders from stagnating indefinitely.

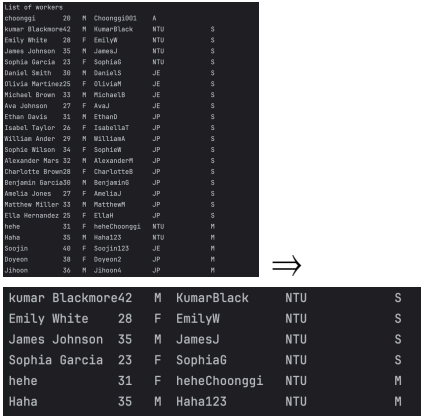
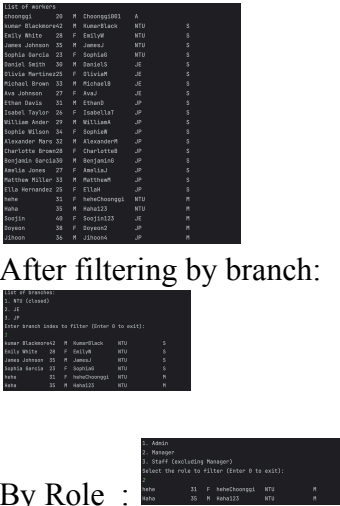
David Pridgen Marshall (University of Tennessee, Knoxville)



| | |
|--|--|
| | |
|--|--|

| | | | |
|-----|--|---|---|
| TC2 | Manager updates the price and the description of an existing menu item, verifying changes have been made to the menu. (repeat the steps before to login as a Manager) | <ol style="list-style-type: none"> Edit Item to Menu(Option 5) Edit Fries(Option1) Enter 1 to update(Enter 1) Update Price(Option 1) <ol style="list-style-type: none"> Enter 10.00 Repeat steps 1 -3 , then for step 4, enter 2 to Update Description. <ol style="list-style-type: none"> Enter “we love fries” Verify by displaying the “All in the menu” | <p>(Cut the image for easier readability)</p> <p>Before updating:</p>  <p>After updating:</p>  |
| TC3 | Manager removes the existing item and verifies that it is no longer available. (repeat the steps before to login as a Manager) | <ol style="list-style-type: none"> Remove Item to Menu(Option 6) Remove Fries(Option1) Enter 1 to remove(Enter 1) Verify by displaying the “All in the menu” | <p>Before removing:</p>  <p>After removing:</p>  |
| TC4 | Manager: Perform multiple sessions of adding, updating and removing menu items | <ol style="list-style-type: none"> Login into Manager Add(Option4) <ol style="list-style-type: none"> Enter “Chicken” .. Similar to TC 1 Updating Menu(Option 5) <ol style="list-style-type: none"> Follow TC 2 Remove item(Option 6) <ol style="list-style-type: none"> Follow TC 3 but change item to Coca-Cola | <p>Changes are made in one session and data changes are saved.</p> <p>Add - Item 7 : Chicken added</p> <p>Update - Fries price and description changed.</p> <p>Remove - Item 5 : Coca-Cola removed</p> <p>Before:</p>  <p>After:</p>  |
| TC5 | Manager: Attempt to add menu item with duplicate name | <ol style="list-style-type: none"> Login to Manager Add Item to Menu(Option 4) Enter Duplicate Item as on current menu: FRIES |  |
| TC6 | Manager: Process order | <ol style="list-style-type: none"> Login as manager Process Order(Option2) Select order to process(Option 1) | <p>Customer side :</p>  <p>Manager side:</p>  <p>⇒</p>  |

| | | | |
|------|---|---|--|
| TC7 | Customers place orders with multiple food items, customise and choose takeaway options, verify order is created. | <ol style="list-style-type: none"> Login as Customer: <ol style="list-style-type: none"> To start ordering(Option 1) Branch (Option 1) Place new order(Option1) Adding food to cart <ol style="list-style-type: none"> Placing new order(option 1) Filter the Menu (option 1) Add food items(option1, 2, 3) Exit to see cart items(enter -1) <ol style="list-style-type: none"> Placing new order – Check Cart to verify (option 4) Customise (option 2) <ol style="list-style-type: none"> Choose cart item to edit (Option 1) Input String for customisation: “more salt” Cart out (Option 5) <ol style="list-style-type: none"> Proceed to checkout (Option 1) Choose Takeaway(Option 2) |    |
| TC8 | Customer place a new order with dine in order | Repeat the same step as TC 6 but choose (Option 1) in step 5b. | System auto update to new cart once the previous customer has cart out. |
| TC9 | Customer: Payment using a debit/ credit (NETS) & Payment using online payment(Paypal) | <ol style="list-style-type: none"> Continuing from TC 7 NETS : Select Payment method (Option 1) Paypal : Select Payment Method(Option 2) | Receipt is printed out for verification.  |
| TC10 | Customer: Track the status of existing order using order ID. | <ol style="list-style-type: none"> Continuing from TC 6, orderId is XXXX Check for Order Status(Option 2) Enter orderID XXXX |  |
| TC11 | Customer: Place food order, check order status with the order ID and collect new food | <ol style="list-style-type: none"> Carry out TC 6-9 to check the order status. Collection of food is done after the staff at the branch has indicated that food is “READY TO PICK UP” (This process should be done fast to prevent the food from being cancelled). | <p>Customer:</p>  <p>⇒</p> <p>Staff Action:</p>  <p>⇒ Customer:</p>  |

| | | | |
|------|---|---|---|
| TC12 | Customer: Place a new order and let it be uncollected beyond timeframe | <ol style="list-style-type: none">Refer to TC 6 60s after staff processed the order, the order will be cancelled if not collected. | <p>Prints:</p> <p>Enter Your Choice: The order has been cancelled due to over time.</p> <p>Checking the order status of that food item using ID: Food Removed</p> <p>Enter your orderID: Enter your orderID (Enter 0 to exit): 4497 Order ID not found!</p> |
| TC13 | Admin: Display Staff list in the same branch. Verify. | <ol style="list-style-type: none">Similar to TC1 to login Admin Login<ol style="list-style-type: none">Login ID : Choonggi001Login Password: defaultNew Password: Choonggi001Display the staff list<ol style="list-style-type: none">Display staff list (Option 3)Want to filter: YFilter based on Branch (Option1) : NTU |  <p>⇒</p> <p>Filter based on NTU only</p> |
| TC14 | Admin : Close a Branch Ensure that it does not show in customer interface | <p>Admin Close the Branch</p> <ol style="list-style-type: none">Login as AdminChoose the close branch(Option10)Choose the branch to close : NTU Option 1 <p>Verifying through Customer Interface.</p> <ol style="list-style-type: none">Go to Management System, option 1 to go to customer | <p>Branch selection shown after going into customer should shows</p> <p>Select your branch (Enter 0 to exit): List of branches:</p> <ol style="list-style-type: none">NTU (closed)JEJP <p>If customer try to select the close branch:</p> <p>The branch is closed. Please select another branch. Select your branch (Enter 0 to exit): List of branches:</p> <ol style="list-style-type: none">NTU (closed)JEJP |
| TC15 | Admin: Display the staff list with filters Verify the list | <ol style="list-style-type: none">Login as AdminDisplay Staff List(Every staff) (Option 4)Filter? (Enter Y)Select the appropriate filters <p>Filters:</p> <ol style="list-style-type: none">RoleAgeGenderBranch <ol style="list-style-type: none">Step 4 and 5 will respect to allow further filtering with additional appropriate inputs.Verify using the general table to see if data syncs |  <p>By Role :</p> |

```
1. Role
2. Female
Select the gender to filter (Enter 0 to exit):
3
Haha          35   M   Haha123      NTU

By gender:

By age :
Enter age to filter (Enter 0 to exit):
35
Haha          35   M   Haha123      NTU
```

TC16 **Admin:** Assign managers to branches with quota and verify that managers are assigned correctly. Verify by printing the list of workers and see changes of their roles.

- 1. Login to Admin
- 2. Assign manager to branch(Option 5)
- 3. Given a list of worker, enter the staff worker index to promote (Choose 10) - Successful Case
- 4. Verify the role changes from Staff to Manager by displaying staff list again

```
Unsuccessful promotion
Enter the staff worker index to make manager (Enter 0 to exit):
24
Staff worker cannot be promoted because quota ratio is full.
Number of staff (excluding Manager):    5
Number of manager:                      2

Successful promotion
Enter the staff worker index to make manager (Enter 0 to exit):
10
JP Branch Manpower:
Number of staff (excluding Manager):    9
Number of manager:                      3
Quota ratio is met.

→ role: S(Staff) to M(Manager)

10: Isabel Taylor  26   F   IsabellaT      JP          S
24: Isabel Taylor  26   F   IsabellaT      JP          M
```

TC17 **Admin:** Transfer a staff/ manager among branches Verify it is in system

- 1. Login as Admin
- 2. Transfer a staff (Option 6)
- 3. Select worker to transfer to
- 4. Select the branch to transfer to
- 5. Verify by displaying staff list

```
Unsuccessful:
Currently, Isabel Taylor works at JP.
Enter the branch index to transfer to (Enter 0 to exit):
2
Manager cannot be assigned because quota ratio is full.
Number of staff (excluding Manager):    4
Number of manager:                      1

Successful:
Currently, William Ander works at JP.
Enter the branch index to transfer to (Enter 0 to exit):
2
William Ander transferred to JE.
JE Branch Manpower:
Number of staff (excluding Manager):    5
Number of manager:                      1
Note that quota ratio is not met.

Verify:

10: William Ander  29   M   WilliamA      JP          S
William Ander     29   M   WilliamA      JE          S
```

TC18 **Admin:** Upload a staff list during system initialization.

- 1. Login to Admin
- 2. Display staff list(Option 4) to verify that the staff list is correctly initialised based on the uploaded file.

```
Code:                                     Output:

[Image of terminal output showing staff list initialization]
```

TC19 **Admin:**Add payment method and verify it is added.

- 1. Login to Admin
- 2. Remove payment(option 12) as we have initially included all payment methods
 - a. Example removing NETS
- 3. Login as customer to verify whether payment is removed
 - a. Customer should not be seeing NETS payment in their payment methods
- 4. Return to Admin
- 5. Add a payment method (option 11)
 - a. Add NETS back

```
Remove a payment method

Payment methods available in the branch:
1. NETS
2. Paypal
3. Paywave
4. ShopeePay
Enter the payment method you want to remove from JE (Enter 0 to exit):
1
payment removed.

Customer side:
checking out now.
1. Paypal
2. Paywave
3. ShopeePay
Please select a payment method (press 0 to exit):

Add a payment method
```

| | | | |
|------|--|---|--|
| | | 6. Login as customer to verify payment is added back <ol style="list-style-type: none"> Customer should now see NETS payment in their payment methods | <pre>List of payments: 1. NETS 2. Paypal 3. PayWave 4. ShopeePay Enter the payment method you want to add to JE (Enter 0 to exit): 1 NETS is added to list of payment methods in JE.</pre> <p>Customer side:</p> <pre>checking out now. 1. Paypal 2. Paywave 3. ShopeePay 4. NETS</pre> |
| TC20 | Admin : Add a new branch without affecting existing functionalities | <ol style="list-style-type: none"> Login to Admin Remove a branch(option8) because we have included all branches initially Add the branch(option7) back | <p>Remove branch</p> <pre>Enter the branch index to remove (Enter 0 to exit): 1 Please confirm that you want to remove NTU. Enter 1 to remove (Enter 0 to exit): 1 Removed NTU. List of branches: 1. JE 2. JP</pre> <p>Add branch</p> <pre>Enter the new branch name (Enter 0 to exit): NTU Enter the location of the branch (Enter 0 to exit): West Enter the branch quota (Enter 0 to exit): 8 Please confirm that you want to add NTU. Enter 1 to add (Enter 0 to exit): 1 Added NTU. List of branches: 1. JE 2. JP 3. NTU</pre> |
| TC21 | Admin : Add a new worker | <ol style="list-style-type: none"> Login to Admin Add staff account (option 1) Enter new worker's loginID, new worker's name, new worker's age, new worker's gender, branch that new worker to work in, new worker's role Display staff list to ensure that new worker is added | <pre>Enter the new worker login ID (Enter 0 to exit): ABC123 Enter the new worker name (Enter 0 to exit): ABC Enter the new worker age (Enter 0 to exit): 20 1. Male 2. Female Select new worker gender (Enter 0 to exit): 1 List of branches: 1. NTU 2. JE 3. JP Select new worker branch (Enter 0 to exit): 1 1. Manager 2. Staff Select new worker role (Enter 0 to exit): 2 Added ABC. NTU Branch Manpower: Number of staff (excluding Manager): 5 Number of manager: 2 Quota is met.</pre> <pre>→ ABC 20 M ABC123 NTU S</pre> |
| TC22 | Staff : Change the default password for staff and re -ogin with new password Use the wrong password details and show error message | <ol style="list-style-type: none"> <u>Login as Staff (Option 2)</u> <ol style="list-style-type: none"> <u>Enter Login ID: EmilyW</u> <u>Password: default</u> <u>New Password : EmilyW</u> <p>Re Login with correct password :</p> <ol style="list-style-type: none"> Verify by relogin <ol style="list-style-type: none"> Exit (Enter 0), then enter (Enter 2) Enter Login ID and New Password <p>Re Login with wrong password:</p> <ol style="list-style-type: none"> Repeat steps in TC 22(Step 1) | <pre>Enter your login ID (press 0 to exit): EmilyW Enter your login Password: default Login successful. Enter your new password: EmilyW</pre> <pre>⇒ Enter your login ID (press 0 to exit): EmilyW Enter your login Password: EmilyW Login successful. Welcome Emily White</pre> <p>VS</p> <pre>Enter your login ID (press 0 to exit): EmilyW Enter your login Password: 123 Incorrect ID or Password! Enter your login ID (press 0 to exit):</pre> |

| | | | |
|------|--|--|---|
| | | <div>2. Verify by re-login<div>a. Exit (Enter 0), then enter (Enter 2)</div><div>b. Enter LoginID: EmilyW</div><div>c. Enter WRONG Password: 123</div></div> | |
| TC23 | Staff: View & process order before customer place order – shows error | <div>1. Login Staff before ordering anything<div>a. Process order (Option 2)</div></div> | <div>Welcome Emily White<div>1. Display orders</div><div>2. Process order</div><div>3. Exit</div><div>What would you like to do?</div><div>1</div><div>No order in the order list.</div><div>Select the order to process (Enter 0 to exit):</div><div>1</div></div> |
| TC24 | Staff: See new orders | <div>1. Login to staff.</div> <div>2. Display Order(option1) to view the new order added.</div> <div>3. Choose the order to display in details</div> | <div>Before:After:</div> <div><div>What would you like to do?</div><div>1</div><div>No order in the order list.</div></div> <div><div>What would you like to do?</div><div>2</div><div>Orders:</div><div>1. OrderID: 0010, status: PREPARING</div></div> |
| TC25 | Staff: Process a new order, update and verify that the order status | <div>Continuing from TC24</div> <div>1. Login to staff</div> <div>2. Process Order(option2) – updating status to ready to pick up</div> | <div>1. OrderID: 0010, status: PREPARING</div> <div>Select the order to process (Enter 0 to exit):</div> <div>1</div> <div>The order is ready to pick up now.</div> |

4. Reflection

Throughout the journey, we learnt the importance of designing well-structured systems by applying design principles. This has enabled us to envision effective solutions that are adaptable to future changes while maintaining data integrity. Moreover, we have embraced practices, such as encapsulation and abstraction to create codes that are easy to understand and debuggable by every team member.

We face many challenges like conflicting ideas within the team that lead to diverse approaches of the system design and implementation approaches. This results in inconsistency in our codes, as we approach the problems differently. Despite this, we overcame through active listening, and willingness to compromise to reach consensus. We see these challenges as opportunities for growth in gaining a deeper understanding of OODP, and they help strengthen our team’s cohesion. Recognising the need for improvement in our design process, we believe that an improvement that can be done is to prioritise the design phase first and allow more time to establish clear design patterns before moving to code. We believe this would reduce misunderstandings and increase the team’s productivity, delivering high-quality projects in the future.