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 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 beawarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course (CE2002 or CZ2002)	Lab Group	Signature /Date
ALLEN LU ZHAO QUAN	SC2002	FDAC	[illegible] 26/4/2024
GUAN XIAOXI	SC2002	FDAC	[illegible] 26/4/2024
KRITI RAJA	SC2002	FDAC	[illegible] 26/4/2024
YU FENGYUAN	SC2002	FDAC	Fengyuan 26/4/2024
ZHAO QIXIAN	SC2002	FDAC	[illegible] 26/4/2024

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.
2. Student Code of Academic Conduct includes the latest guidelines on usage of Generative AI and any other guidelines as released by NTU.

SC2002 FOMS Project Report

1. Design Considerations

The FOMS (Fastfood Ordering and Management System) is designed as a Java console application that enhances the efficiency of fast food restaurant operations. It adeptly manages human resource allocation, ordering, payment, and order tracking and processing with a structure that emphasizes modularity, reusability, and scalability. Supporting different user roles such as Customers, Staff, Manager and Administrators, the system assigns distinct capabilities to each.

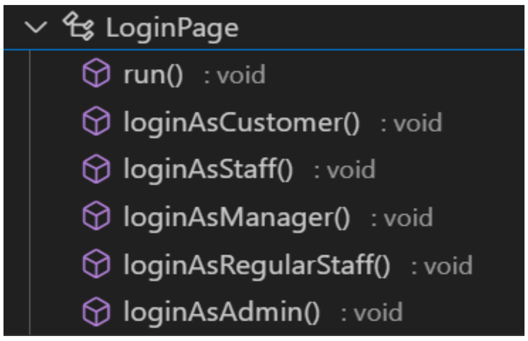
1.1 Our Approach

The FOMS architecture is designed with an emphasis on high cohesion and loose coupling, organized into three main categories: controllers, boundaries, and entities. Controllers, such as BranchManager and PaymentManager, handle requests from users, processing those requests, and then invoking changes. Boundaries, featuring interfaces like LoginPage , display data to users and interpret user commands to send to the controller. Entities like Customer, Manager, and Item constitute the essential data models that store and manage critical information. This structural setup ensures clear separation of responsibilities, minimal interdependencies among components, facilitating easy maintenance and future system enhancements.

1.2 Principles Used

1.2.1 Single Responsibility Principle (SRP)

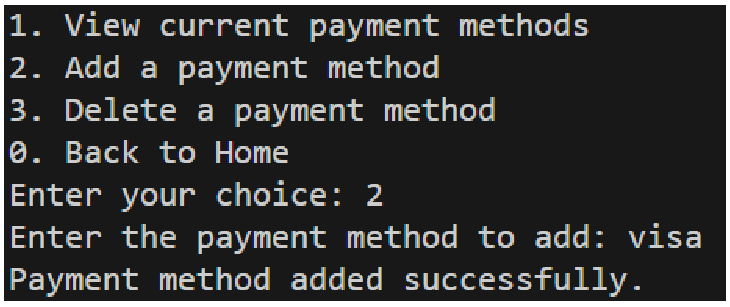
SRP advocates for each class to have a single responsibility, meaning it should only have one reason to change, enhancing maintainability and reducing coupling. One instance when we apply SRP is in LoginPage, where each method within the class focuses on a specific aspect of the login process, maintaining a clear separation of concerns.



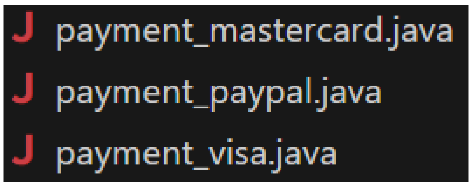
```
LoginPage
    run() : void
    loginAsCustomer() : void
    loginAsStaff() : void
    loginAsManager() : void
    loginAsRegularStaff() : void
    loginAsAdmin() : void
```

1.2.2 Open/Closed Principle (OCP)

OCP suggests software entities should allow extension for new functionality without modifying existing code, promoting adaptability and maintainability. We apply OCP when admin adds Payment Method, where changes can be made to new java files created, reducing the risk of introducing bugs.



```
1. View current payment methods
2. Add a payment method
3. Delete a payment method
0. Back to Home
Enter your choice: 2
Enter the payment method to add: visa
Payment method added successfully.
```



```
payment_mastercard.java
payment_paypal.java
payment_visa.java
```

1.2.3 Liskov Substitution Principle (LSP)

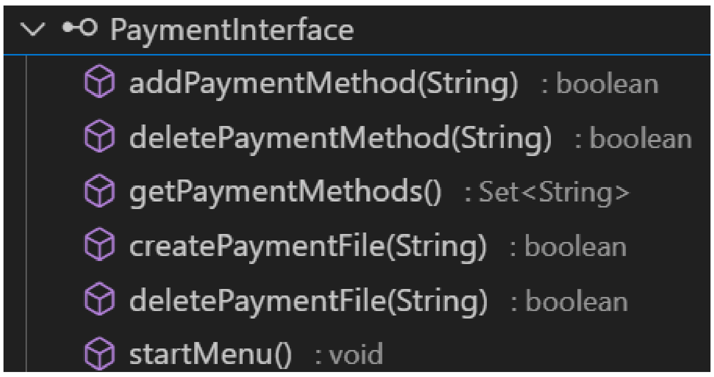
LSP states that objects of a superclass should be replaceable with objects of its subclasses without altering desired properties, ensuring behavioral compatibility. The Manager and Staff classes exemplify the Liskov Substitution Principle (LSP) by maintaining substitutability and behavior consistency within their inheritance hierarchy. Both Manager and Staff objects are interchangeable in contexts where a Staff object is expected, ensuring seamless functionality across the codebase. Despite Manager having additional functionalities specific to managerial roles, it extends Staff without violating the contracts established by its superclass.

1.2.4 Interface Segregation Principle (ISP)

ISP advises creating specific interfaces containing only methods relevant to their clients, avoiding imposing unnecessary dependencies. This promotes interface clarity, reduces coupling, and prevents classes from implementing methods they don't need. In our code, we have separated "LoginVerification" to three interfaces, "AdminattemptLogin", "ManagerattemptLogin" and "StaffattemptLogin", this would allow classes to implement only the interfaces relevant to their role, reducing unnecessary dependencies.

1.2.5 Dependency Injection Principle (DIP)

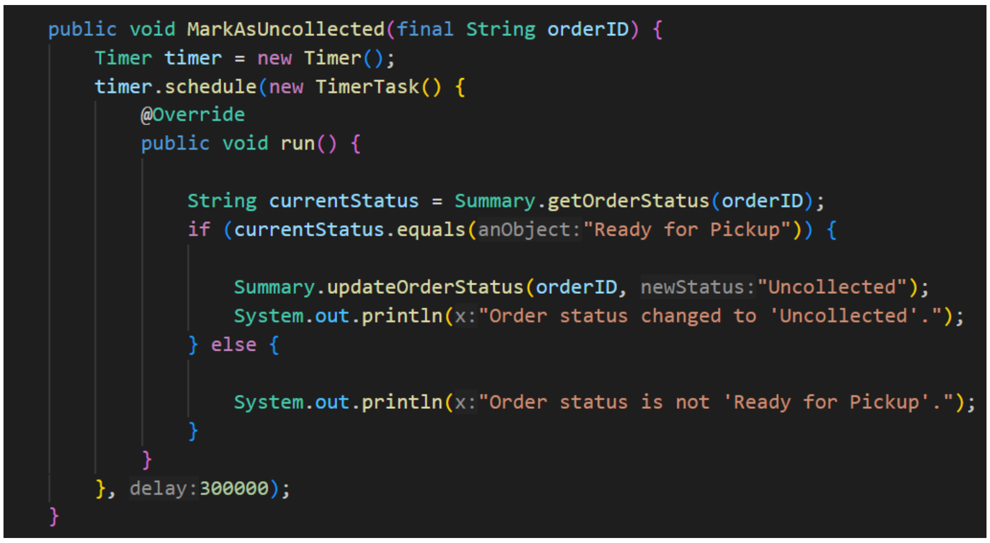
DIP advocates for abstraction layers between modules, promoting loose coupling and facilitating dependency management. High-level modules should depend on interfaces, not implementations, enabling flexibility and easier testing. This can be seen in our "PaymentManager" class, which implements an interface "PaymentInterface". By depending on this interface rather than concrete implementations, classes can be decoupled from specific payment method details, allowing for flexibility in implementation and facilitating dependency management.



```
PaymentInterface
    addPaymentMethod(String)  : boolean
    deletePaymentMethod(String)  : boolean
    getPaymentMethods()  : Set<String>
    createPaymentFile(String)  : boolean
    deletePaymentFile(String)  : boolean
    startMenu()  : void
```

1.3 Assumptions Made

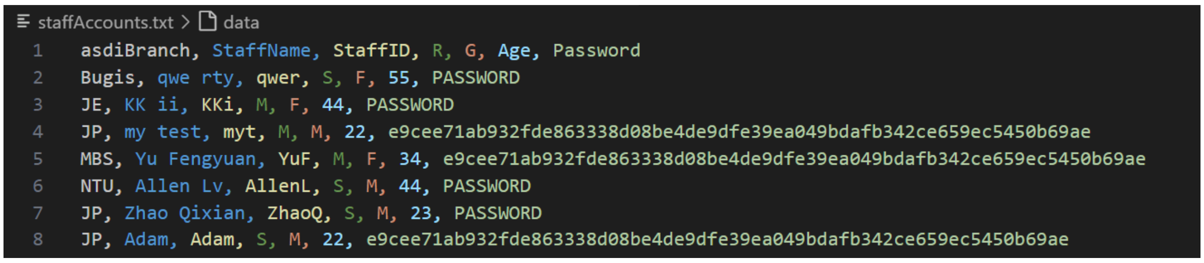
One assumption we have made is that staff and managers will keep the programme running after they have completed a task, for example view orders or process order. Whereas for customers, their programme will be automatically terminated once they have printed out the receipt. This aims to simulate the real world scenario, where staff and managers usually keep processing orders, while customers usually exit after making an order, until they check OrderStatus after some time. This logic also enables the countdown function that automatically marks the OrderStatus from Ready for Pickup to Uncollected, if order is not collected in 300 seconds. This function will not be possible if staff and managers terminate their console.



```java
public void MarkAsUncollected(final String orderID) {
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        @Override
        public void run() {

            String currentStatus = Summary.getOrderStatus(orderID);
            if (currentStatus.equals(anObject:"Ready for Pickup")) {

                Summary.updateOrderStatus(orderID, newStatus:"Uncollected");
                System.out.println(x:"Order status changed to 'Uncollected'.");
            } else {

                System.out.println(x:"Order status is not 'Ready for Pickup'.");
            }
        }
    }, delay:300000);
}
```

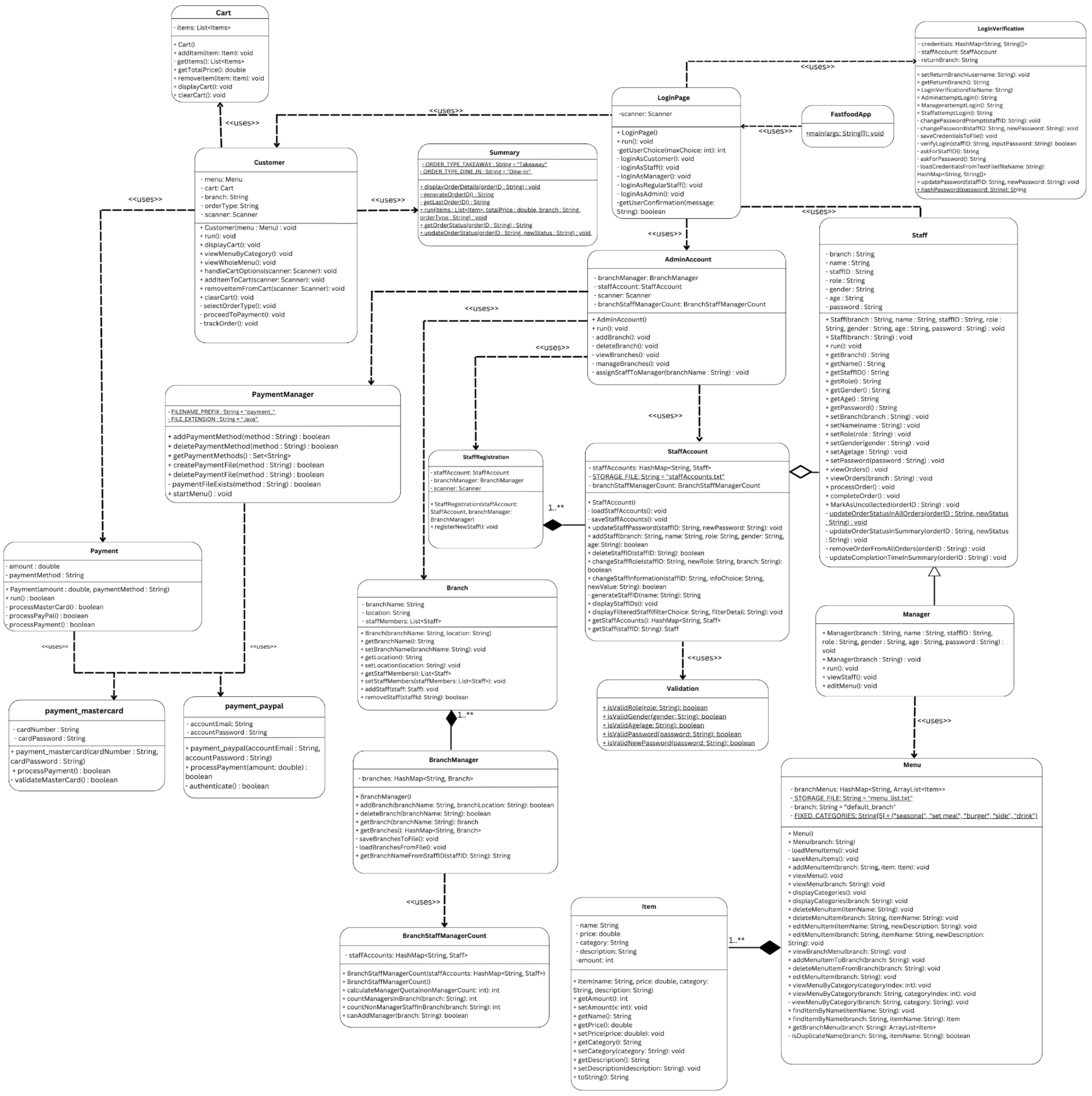
1.4 Innovative Features

Within the staffAccounts.txt file, housing credentials for all staff and managers, database owners wield access to its contents, including passwords. Upon initial staff or manager registration, a default password, "PASSWORD," is assigned. Subsequently, individuals may opt to change it. Nonetheless, this password's accessibility raises privacy concerns. Thus, we hash the new password via SHA-3 before storage. Such a measure upholds privacy standards, preventing unauthorized access to sensitive data and ensuring integrity of stored credentials.



```
staffAccounts.txt > data
asdiBranch, StaffName, StaffID, R, G, Age, Password
Bugis, qwe rty, qwer, S, F, 55, PASSWORD
JE, KK ii, KKi, M, F, 44, PASSWORD
JP, my test, myt, M, M, 22, e9cee71ab932fde863338d08be4de9dfe39ea049bdafb342ce659ec5450b69ae
MBS, Yu Fengyuan, YuF, M, F, 34, e9cee71ab932fde863338d08be4de9dfe39ea049bdafb342ce659ec5450b69ae
NTU, Allen Lv, AllenL, S, M, 44, PASSWORD
JP, Zhao Qixian, ZhaoQ, S, M, 23, PASSWORD
JP, Adam, Adam, S, M, 22, e9cee71ab932fde863338d08be4de9dfe39ea049bdafb342ce659ec5450b69ae
```

2. Detailed UML Class Diagram


Cart
- items: List<Items>
+ Cart()
+ addItem(item: Item): void
- getItems(): List<Items>
+ getTotalPrice(): double
+ removeItem(item: Item): void
+ displayCart(): void
+ clearCart(): void
<<uses>>
<<uses>>
LoginPage
-scanner: Scanner
+ LoginPage()
+ run(): void
-getUserChoice(maxChoice: int): int
- loginAsCustomer(): void
- loginAsStaff(): void
- loginAsManager(): void
- loginAsRegularStaff(): void
- loginAsAdmin(): void
-getUserConfirmation(message: String): boolean
<<uses>>
FastfoodApp
+main(args: String[]): void
<<uses>>
LoginVerification
- credentials: HashMap<String, String[]>
- staffAccount: StaffAccount
- returnBranch: String
+ setReturnBranch(username: String): void
+ getReturnBranch(): String
+ LoginVerification(fileName: String)
+ AdminattemptLogin(): String
+ ManagerattemptLogin(): String
+ StaffattemptLogin(): String
- changePasswordPrompt(staffID: String): void
- changePassword(staffID: String, newPassword: String): void
- saveCredentialsToFile(): void
- verifyLogin(staffID: String, inputPassword: String): boolean
- askForStaffID(): String
- askForPassword(): String
- loadCredentialsFromTextFile(fileName: String): HashMap<String, String[]>
+ updatePassword(staffID: String, newPassword: String): void
+ hashPassword(password: String): String
<<uses>>
Customer
- menu: Menu
- cart: Cart
- branch: String
- orderType: String
- scanner: Scanner
+ Customer(menu : Menu) : void
+ run(): void
+ displayCart(): void
+ viewMenuByCategory(): void
+ viewWholeMenu(): void
+ handleCartOptions(scanner: Scanner): void
+ addItemToCart(scanner: Scanner): void
+ removeItemFromCart(scanner: Scanner): void
+ clearCart(): void
- selectOrderType(): void
- proceedToPayment(): void
- trackOrder(): void
<<uses>>
<<uses>>
Summary
- ORDER_TYPE_TAKEAWAY : String = "Takeaway"
- ORDER_TYPE_DINE_IN : String = "Dine-in"
+ displayOrderDetails(orderID : String) : void
- generateOrderID() : String
- getLastOrderID() : String
+ run(items : List<Item>, totalPrice : double, branch : String, orderType : String) : void
+ getOrderStatus(orderID : String) : String
+ updateOrderStatus(orderID : String, newStatus : String) : void
<<uses>>
<<uses>>
AdminAccount
- branchManager: BranchManager
- staffAccount: StaffAccount
- scanner: Scanner
- branchStaffManagerCount: BranchStaffManagerCount
+ AdminAccount()
+ run(): void
- addBranch(): void
- deleteBranch(): void
- viewBranches(): void
- manageBranches(): void
- assignStaffToManager(branchName : String) : void
<<uses>>
<<uses>>
Staff
- branch : String
- name : String
- staffID : String
- role : String
- gender : String
- age : String
- password : String
+ Staff(branch : String, name : String, staffID : String, role : String, gender : String, age : String, password : String) : void
+ Staff(branch : String) : void
+ run(): void
+ getBranch() : String
+ getName() : String
+ getStaffID() : String
+ getRole() : String
+ getGender() : String
+ getAge() : String
+ getPassword() : String
+ setBranch(branch : String) : void
+ setName(name : String) : void
+ setRole(role : String) : void
+ setGender(gender : String) : void
+ setAge(age : String) : void
+ setPassword(password : String) : void
+ viewOrders() : void
+ viewOrders(branch : String) : void
+ processOrder() : void
+ completeOrder() : void
+ MarkAsUncollected(orderID : String) : void
- updateOrderStatusInAllOrders(orderID : String, newStatus : String) : void
- updateOrderStatusInSummary(orderID : String, newStatus : String) : void
- removeOrderFromAllOrders(orderID : String) : void
- updateCompletionTimeInSummary(orderID : String) : void
PaymentManager
- FILENAME_PREFIX : String = "payment_"
- FILE_EXTENSION : String = ".java"
+ addPaymentMethod(method : String) : boolean
+ deletePaymentMethod(method : String) : boolean
+ getPaymentMethods() : Set<String>
+ createPaymentFile(method : String) : boolean
+ deletePaymentFile(method : String) : boolean
- paymentFileExists(method : String) : boolean
+ startMenu() : void
<<uses>>
StaffRegistration
- staffAccount: StaffAccount
- branchManager: BranchManager
- scanner: Scanner
+ StaffRegistration(staffAccount: StaffAccount, branchManager: BranchManager)
+ registerNewStaff(): void
1..**
StaffAccount
- staffAccounts: HashMap<String, Staff>
- STORAGE_FILE: String = "staffAccounts.txt"
- branchStaffManagerCount: BranchStaffManagerCount
+ StaffAccount()
- loadStaffAccounts(): void
- saveStaffAccounts(): void
+ updateStaffPassword(staffID: String, newPassword: String): void
+ addStaff(branch: String, name: String, role: String, gender: String, age: String): boolean
+ deleteStaffID(staffID: String): boolean
+ changeStaffRole(staffID: String, newRole: String, branch: String): boolean
+ changeStaffInformation(staffID: String, infoChoice: String, newValue: String): boolean
- generateStaffID(name: String): String
+ displayStaffIDs(): void
+ displayFilteredStaff(filterChoice: String, filterDetail: String): void
+ getStaffAccounts(): HashMap<String, Staff>
+ getStaff(staffID: String): Staff
Payment
- amount : double
- paymentMethod : String
+ Payment(amount : double, paymentMethod : String)
+ run() : boolean
- processMasterCard() : boolean
- processPayPal() : boolean
- processPayment() : boolean
<<uses>>
<<uses>>
Branch
- branchName: String
- location: String
- staffMembers: List<Staff>
+ Branch(branchName: String, location: String)
+ getBranchName(): String
+ setBranchName(branchName: String): void
+ getLocation(): String
+ setLocation(location: String): void
+ getStaffMembers(): List<Staff>
+ setStaffMembers(staffMembers: List<Staff>): void
+ addStaff(staff: Staff): void
+ removeStaff(staffId: String): boolean
Manager
+ Manager(branch : String, name : String, staffID : String, role : String, gender : String, age : String, password : String) : void
+ Manager(branch : String) : void
+ run(): void
+ viewStaff(): void
+ editMenu(): void
<<uses>>
Validation
+ isValidRole(role: String): boolean
+ isValidGender(gender: String): boolean
+ isValidAge(age: String): boolean
+ isValidPassword(password: String): boolean
+ isValidNewPassword(password: String): boolean
payment_mastercard
- cardNumber : String
- cardPassword : String
+ payment_mastercard(cardNumber : String, cardPassword : String)
+ processPayment() : boolean
- validateMasterCard() : boolean
payment_paypal
- accountEmail: String
- accountPassword : String
+ payment_paypal(accountEmail : String, accountPassword : String)
+ processPayment(amount: double) : boolean
- authenticate() : boolean
1..**
BranchManager
- branches: HashMap<String, Branch>
+ BranchManager()
+ addBranch(branchName: String, branchLocation: String): boolean
+ deleteBranch(branchName: String): boolean
+ getBranch(branchName: String): Branch
+ getBranches(): HashMap<String, Branch>
- saveBranchesToFile(): void
- loadBranchesFromFile(): void
+ getBranchNameFromStaffID(staffID: String): String
<<uses>>
<<uses>>
Menu
- branchMenus: HashMap<String, ArrayList<Item>>
- STORAGE_FILE: String = "menu_list.txt"
- branch: String = "default_branch"
- FIXED_CATEGORIES: String[5] = {"seasonal", "set meal", "burger", "side", "drink"}
+ Menu()
+ Menu(branch: String)
- loadMenuItems(): void
- saveMenuItems(): void
+ addMenuItem(branch: String, item: Item): void
+ viewMenu(): void
+ viewMenu(branch: String): void
+ displayCategories(): void
+ displayCategories(branch: String): void
+ deleteMenuItem(itemName: String): void
+ deleteMenuItem(branch: String, itemName: String): void
+ editMenuItem(itemName: String, newDescription: String): void
+ editMenuItem(branch: String, itemName: String, newDescription: String): void
+ viewBranchMenu(branch: String): void
+ addMenuItemToBranch(branch: String): void
+ deleteMenuItemFromBranch(branch: String): void
+ editMenuItem(branch: String): void
+ viewMenuByCategory(categoryIndex: int): void
+ viewMenuByCategory(branch: String, categoryIndex: int): void
- viewMenuByCategory(branch: String, category: String): void
+ findItemByName(itemName: String): void
+ findItemByName(branch: String, itemName: String): Item
+ getBranchMenu(branch: String): ArrayList<Item>
- isDuplicateName(branch: String, itemName: String): boolean
Item
- name: String
- price: double
- category: String
- description: String
-amount: int
+ Item(name: String, price: double, category: String, description: String)
+ getAmount(): int
+ setAmount(x: int): void
+ getName(): String
+ getPrice(): double
+ setPrice(price: double): void
+ getCategory(): String
+ setCategory(category: String): void
+ getDescription(): String
+ setDescription(description: String): void
+ toString(): String
1..**
BranchStaffManagerCount
- staffAccounts: HashMap<String, Staff>
+ BranchStaffManagerCount(staffAccounts: HashMap<String, Staff>)
+ BranchStaffManagerCount()
+ calculateManagerQuota(nonManagerCount: int): int
+ countManagersInBranch(branch: String): int
+ countNonManagerStaffInBranch(branch: String): int
+ canAddManager(branch: String): boolean

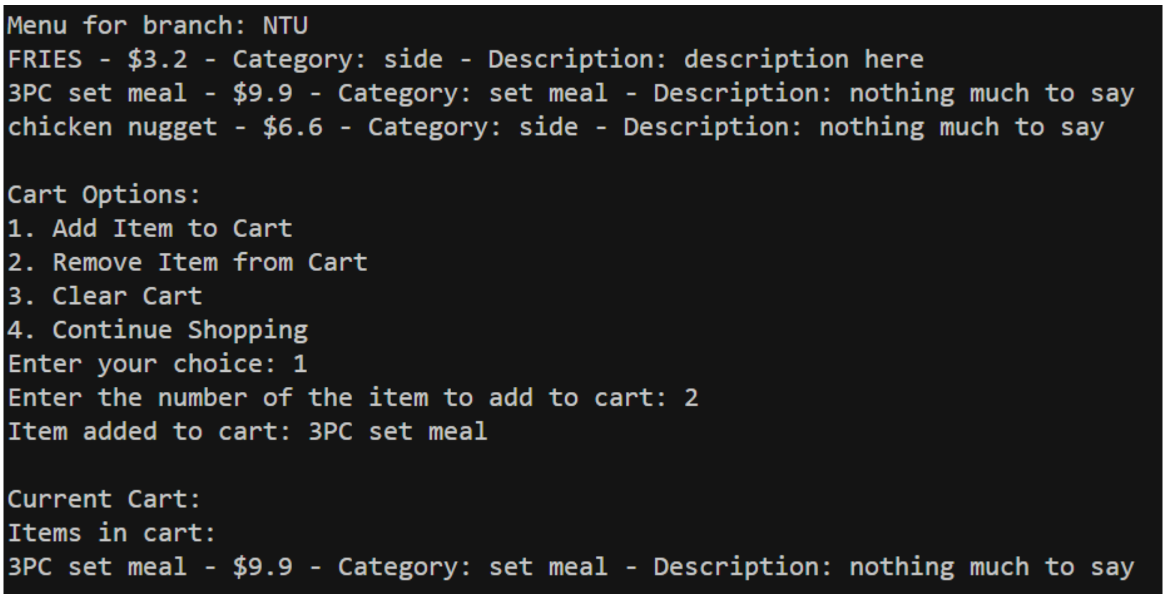
3.Testing

3.1 Customer Functions

3.1.1 Customer add Item to Cart

Choose Customer → Branch: “NTU” → Choose Add Item to Cart

→ Choose “3PC set meal” → Item added to Cart and Cart displayed automatically

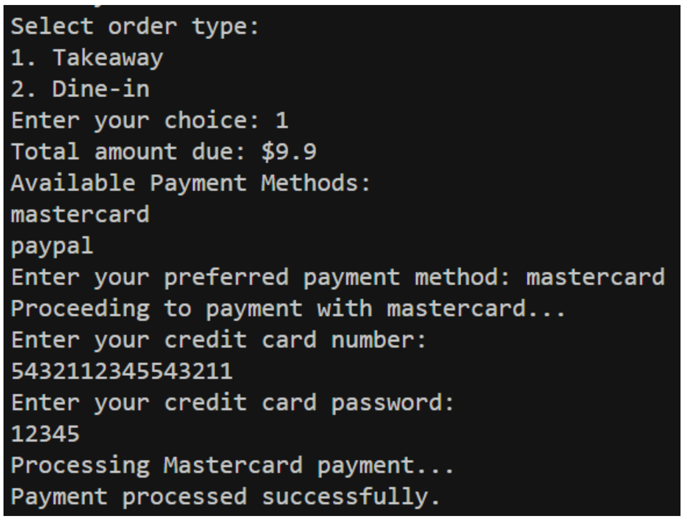


```
Menu for branch: NTU
FRIES - $3.2 - Category: side - Description: description here
3PC set meal - $9.9 - Category: set meal - Description: nothing much to say
chicken nugget - $6.6 - Category: side - Description: nothing much to say

Cart Options:
1. Add Item to Cart
2. Remove Item from Cart
3. Clear Cart
4. Continue Shopping
Enter your choice: 1
Enter the number of the item to add to cart: 2
Item added to cart: 3PC set meal

Current Cart:
Items in cart:
3PC set meal - $9.9 - Category: set meal - Description: nothing much to say
```

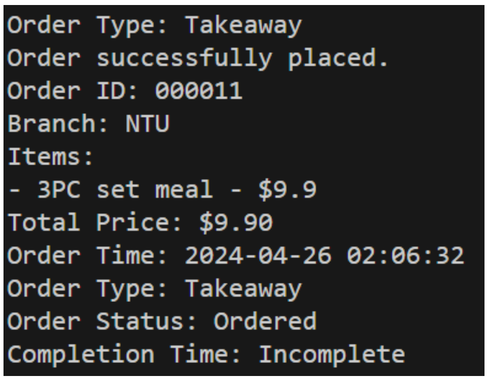
3.1.2 Customer make payments and place Order

Customer choose “Takeaway” → payment method: “mastercard”

→ key in card number and password → automatically display Receipt



```
Select order type:
1. Takeaway
2. Dine-in
Enter your choice: 1
Total amount due: $9.9
Available Payment Methods:
mastercard
paypal
Enter your preferred payment method: mastercard
Proceeding to payment with mastercard...
Enter your credit card number:
5432112345543211
Enter your credit card password:
12345
Processing Mastercard payment...
Payment processed successfully.
```

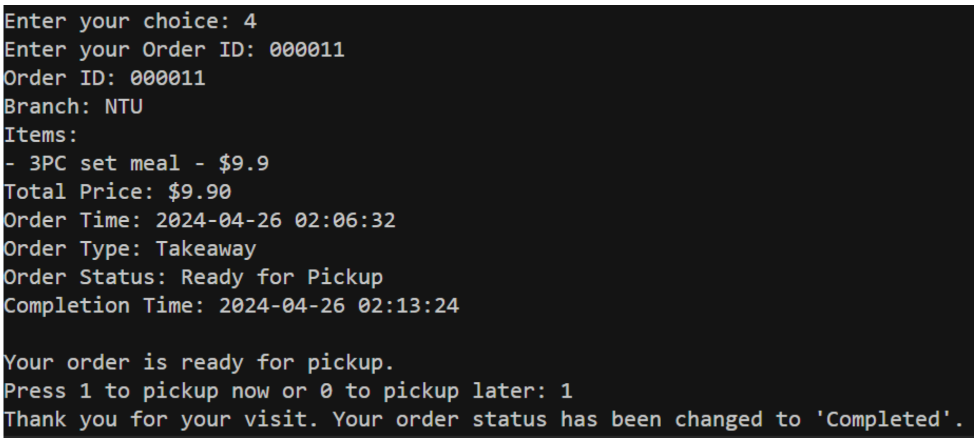


```
Order Type: Takeaway
Order successfully placed.
Order ID: 000011
Branch: NTU
Items:
- 3PC set meal - $9.9
Total Price: $9.90
Order Time: 2024-04-26 02:06:32
Order Type: Takeaway
Order Status: Ordered
Completion Time: Incomplete
```

3.1.3 Customer track Order Status and collect

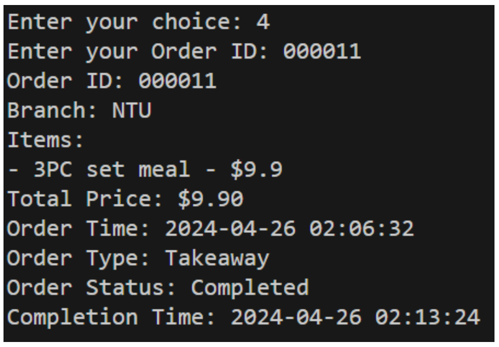
Customer choose track Order Status → If Status is Ready for Pickup

→ Customer choose to pickup



```
Enter your choice: 4
Enter your Order ID: 000011
Order ID: 000011
Branch: NTU
Items:
- 3PC set meal - $9.9
Total Price: $9.90
Order Time: 2024-04-26 02:06:32
Order Type: Takeaway
Order Status: Ready for Pickup
Completion Time: 2024-04-26 02:13:24

Your order is ready for pickup.
Press 1 to pickup now or 0 to pickup later: 1
Thank you for your visit. Your order status has been changed to 'Completed'.
```

After pickup, customer track order again and it is Completed

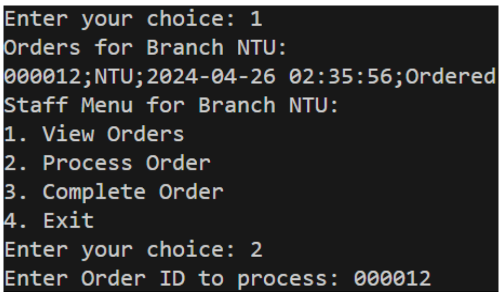


```
Enter your choice: 4
Enter your Order ID: 000011
Order ID: 000011
Branch: NTU
Items:
- 3PC set meal - $9.9
Total Price: $9.90
Order Time: 2024-04-26 02:06:32
Order Type: Takeaway
Order Status: Completed
Completion Time: 2024-04-26 02:13:24
```

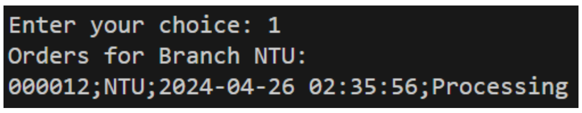
3.2 Staff Functions

3.2.1 Staff process Order

Staff Choose Process Order → Order Status change to Processing in AllOrders.txt and Summary.txt



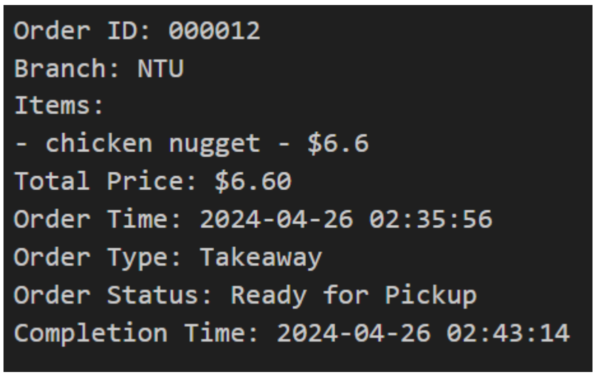
```
Enter your choice: 1
Orders for Branch NTU:
000012;NTU;2024-04-26 02:35:56;Ordered
Staff Menu for Branch NTU:
1. View Orders
2. Process Order
3. Complete Order
4. Exit
Enter your choice: 2
Enter Order ID to process: 000012
```



```
Enter your choice: 1
Orders for Branch NTU:
000012;NTU;2024-04-26 02:35:56;Processing
```

3.2.2 Staff complete Order

Staff choose Complete Order → Order removed from AllOrders.txt and Order Status change to Ready for Pickup in Summary.txt



```
Order ID: 000012
Branch: NTU
Items:
- chicken nugget - $6.6
Total Price: $6.60
Order Time: 2024-04-26 02:35:56
Order Type: Takeaway
Order Status: Ready for Pickup
Completion Time: 2024-04-26 02:43:14
```

3.3 Manager

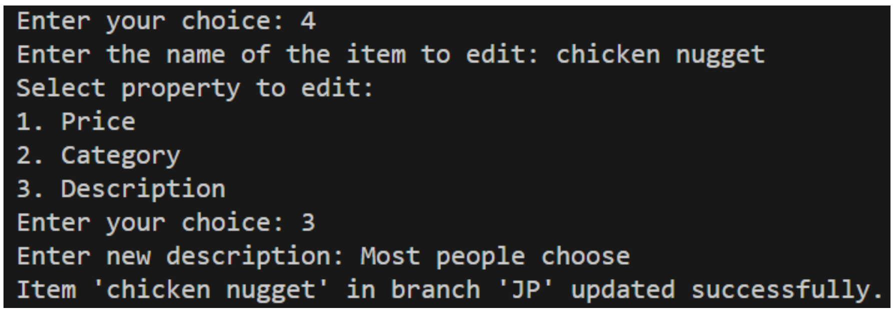
Manager extends Staff, this inheritance ensures that Manager (subclass) can perform all the methods of Staff (superclass), which fosters code reuse and extensibility by allowing subclasses to inherit properties and behaviors from their superclass, promoting modularity and facilitating efficient development.

Some of Manager unique methods are:

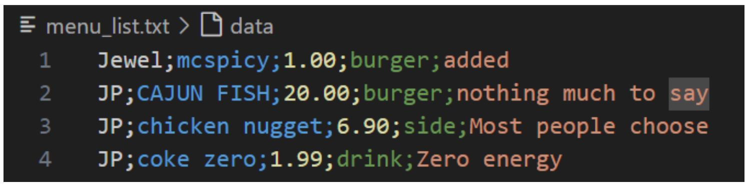
3.3.1 Manager Edit Menu Item Description

Manager choose edit “chicken nugget” → choose “Description”

→ enter “Most people choose” → Description edited in menu_list.txt



```
Enter your choice: 4
Enter the name of the item to edit: chicken nugget
Select property to edit:
1. Price
2. Category
3. Description
Enter your choice: 3
Enter new description: Most people choose
Item 'chicken nugget' in branch 'JP' updated successfully.
```



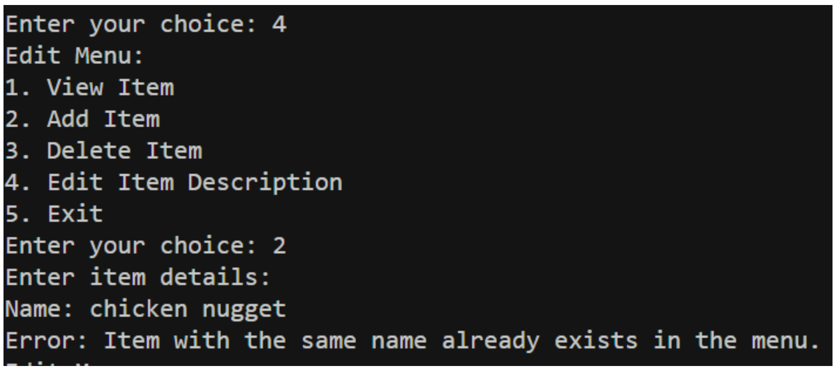
menu_list.txt > data

```
Jewel;mcspicy;1.00;burger;added
JP;CAJUN FISH;20.00;burger;nothing much to say
JP;chicken nugget;6.90;side;Most people choose
JP;coke zero;1.99;drink;Zero energy
```

3.3.2 Managers add new Item but same name

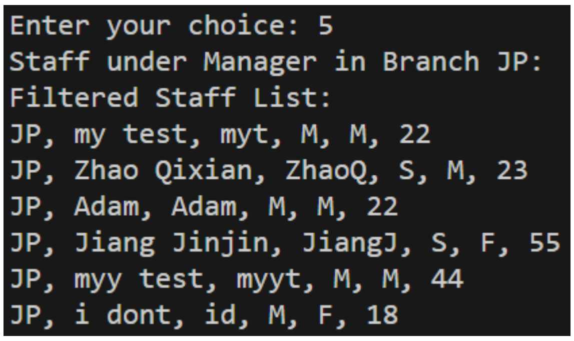
Admin choose edit Menu Item → add Item → add chicken nugget

→ failed as it is already in the menu



```
Enter your choice: 4
Edit Menu:
1. View Item
2. Add Item
3. Delete Item
4. Edit Item Description
5. Exit
Enter your choice: 2
Enter item details:
Name: chicken nugget
Error: Item with the same name already exists in the menu.
```

3.3.3 Managers view staff under their branch

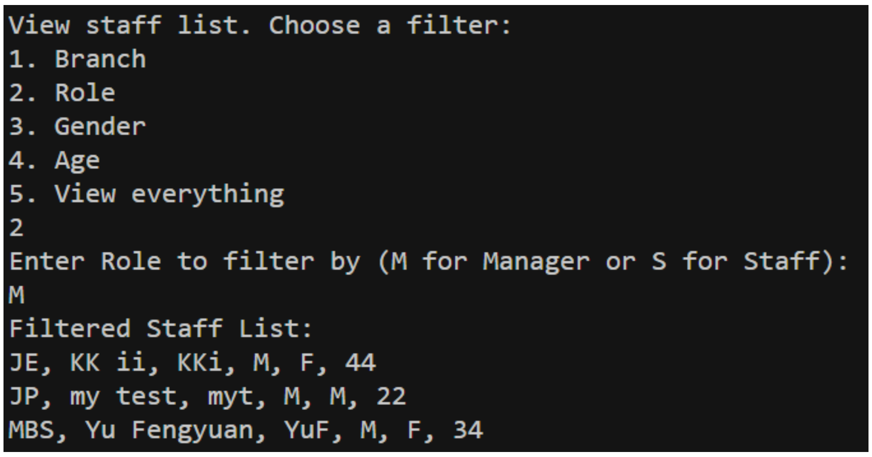


```
Enter your choice: 5
Staff under Manager in Branch JP:
Filtered Staff List:
JP, my test, myt, M, M, 22
JP, Zhao Qixian, ZhaoQ, S, M, 23
JP, Adam, Adam, M, M, 22
JP, Jiang Jinjin, JiangJ, S, F, 55
JP, myy test, myyt, M, M, 44
JP, i dont, id, M, F, 18
```

3.4 Admin

3.4.1 Admin view Staff filtered by Role

Admin choose view staff → filter by role → M for Manager → View all Managers

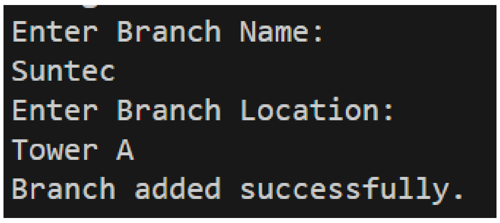


```
View staff list. Choose a filter:
1. Branch
2. Role
3. Gender
4. Age
5. View everything
2
Enter Role to filter by (M for Manager or S for Staff):
M
Filtered Staff List:
JE, KK ii, KKi, M, F, 44
JP, my test, myt, M, M, 22
MBS, Yu Fengyuan, YuF, M, F, 34
```

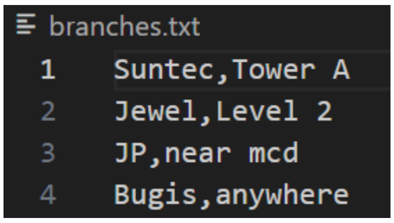
3.4.2 Admin add new branch

Admin choose add new branch → key in branch name and location

→ branch added to branches.txt

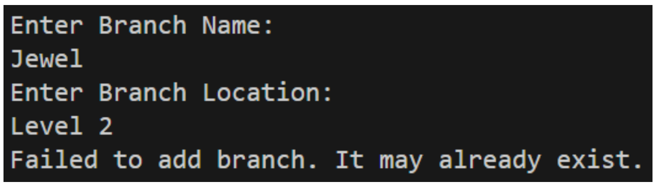


```
Enter Branch Name:
Suntec
Enter Branch Location:
Tower A
Branch added successfully.
```



```
branches.txt
Suntec,Tower A
Jewel,Level 2
JP,near mcd
Bugis,anywhere
```

If admin key in existing branch name and location → failed to add



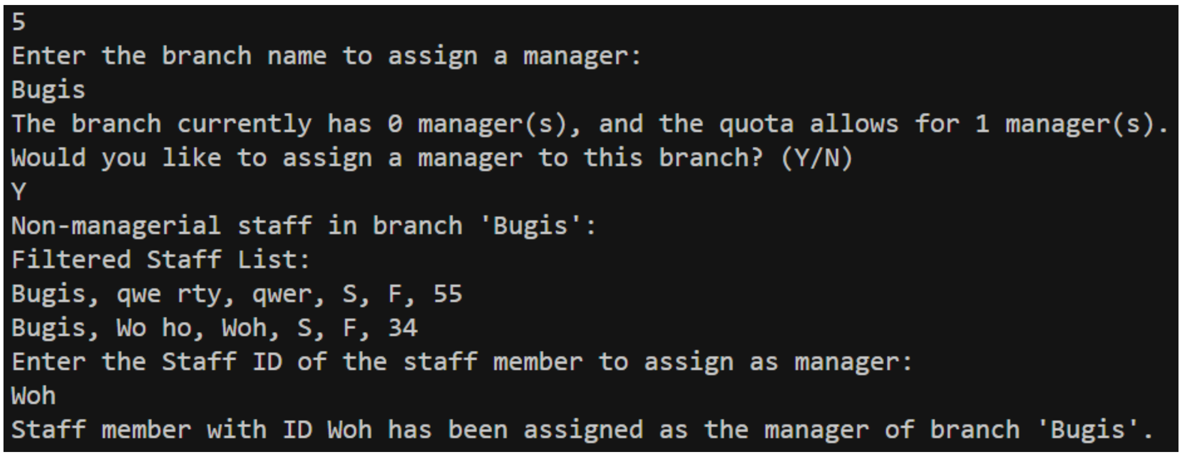
```
Enter Branch Name:
Jewel
Enter Branch Location:
Level 2
Failed to add branch. It may already exist.
```

3.4.3 Admin assigns a staff to manager

Admin chooses assign a staff to manager

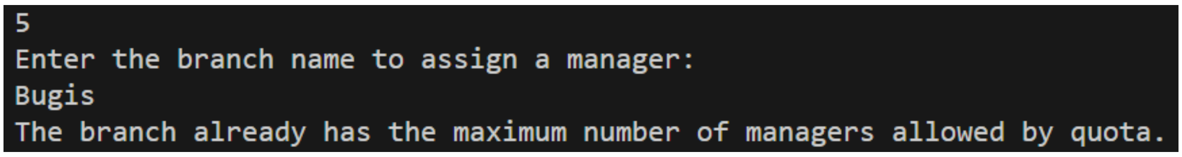
→ choose branch “Bugis” and quota displayed

→ choose staff “Woh” → Successfully assigned



```
5
Enter the branch name to assign a manager:
Bugis
The branch currently has 0 manager(s), and the quota allows for 1 manager(s).
Would you like to assign a manager to this branch? (Y/N)
Y
Non-managerial staff in branch 'Bugis':
Filtered Staff List:
Bugis, qwe rty, qwer, S, F, 55
Bugis, Wo ho, Woh, S, F, 34
Enter the Staff ID of the staff member to assign as manager:
Woh
Staff member with ID Woh has been assigned as the manager of branch 'Bugis'.
```

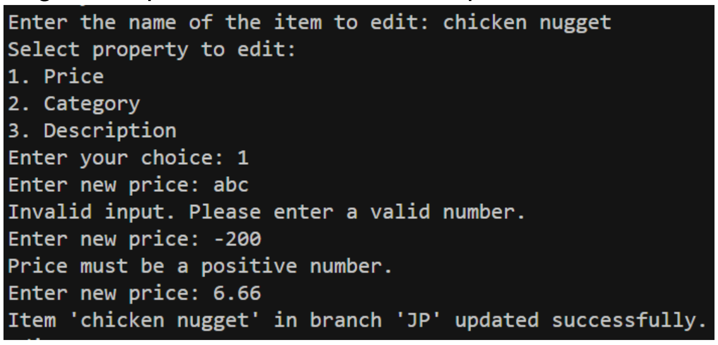
If admin assigns another staff to manager → failed due to quota



```
5
Enter the branch name to assign a manager:
Bugis
The branch already has the maximum number of managers allowed by quota.
```

3.5 Error handling

Manager edits Menu Item price → Non numeric input REJECTED
Negative input REJECTED → Valid price ACCEPTED



```
Enter the name of the item to edit: chicken nugget
Select property to edit:
1. Price
2. Category
3. Description
Enter your choice: 1
Enter new price: abc
Invalid input. Please enter a valid number.
Enter new price: -200
Price must be a positive number.
Enter new price: 6.66
Item 'chicken nugget' in branch 'JP' updated successfully.
```

4.Reflection

As we navigate our project, we come across the Single Responsibility Principle (SRP), a key concept in OODP SOLID principles. Initially, our classes handle multiple tasks, leading to much complexity and confusion. To improve on this, we split functions like "staffAccounts" into distinct units, isolating “Validation” and “staffRegistration” classes. This approach streamlines our codebase, making it easier to maintain and collaborate. With SRP, each component serves a specific purpose, enhancing our project's organization and efficiency.

Initially, collaboration posed challenges as different team members authored diverse classes, resulting in parameter mismatches. To address this, we standardized parameters and function formats, ensuring coherence across the project. We introduced MyLog.txt to track progress and facilitate communication, where members can update their actions. This experience underscored the significance of collaboration and communication in achieving project success.