CSE110: Object Oriented Programming Section: 15
Spring 24 Semester

Project Report
Grocery Deliver Management System

Course Code	: CSE110
Course Title	: Object Oriented Programming
Section	:15
Group No	:1
Group Name	:Clown Trio

Submitted by:

Student ID	Student Name	Contribution Percentage
2023-2-60-101	Raquib Uddin Sarkar	32%
2023-2-60-103	Md Sadik Shahriar	42%
2023-2-60-117	Md Sifat Bin Islam	26%

Submitted To:

Ahmed Abdal Shafi Rasel, Lecturer, Department of CSE East West University

Submission Date: 25.05.2024

Table of Contents

Introduction

The Crimson Grocery is a grocery management system that makes it easy for both customers and shop owners to buy and sell goods. The system provides different interfaces for customers and shop owners.

For Customers:

1.Account Creation: Customers must create an account before making any purchases.
2.Product Selection: Customers can choose products from various shops.
3.Shopping Cart: Customers add their desired products to the cart and confirm the purchase.
4.Automatic Stock Update: The stock levels are automatically updated after a purchase.
5.Order History: After placing order customers can see their order history.

For Shop Owners:

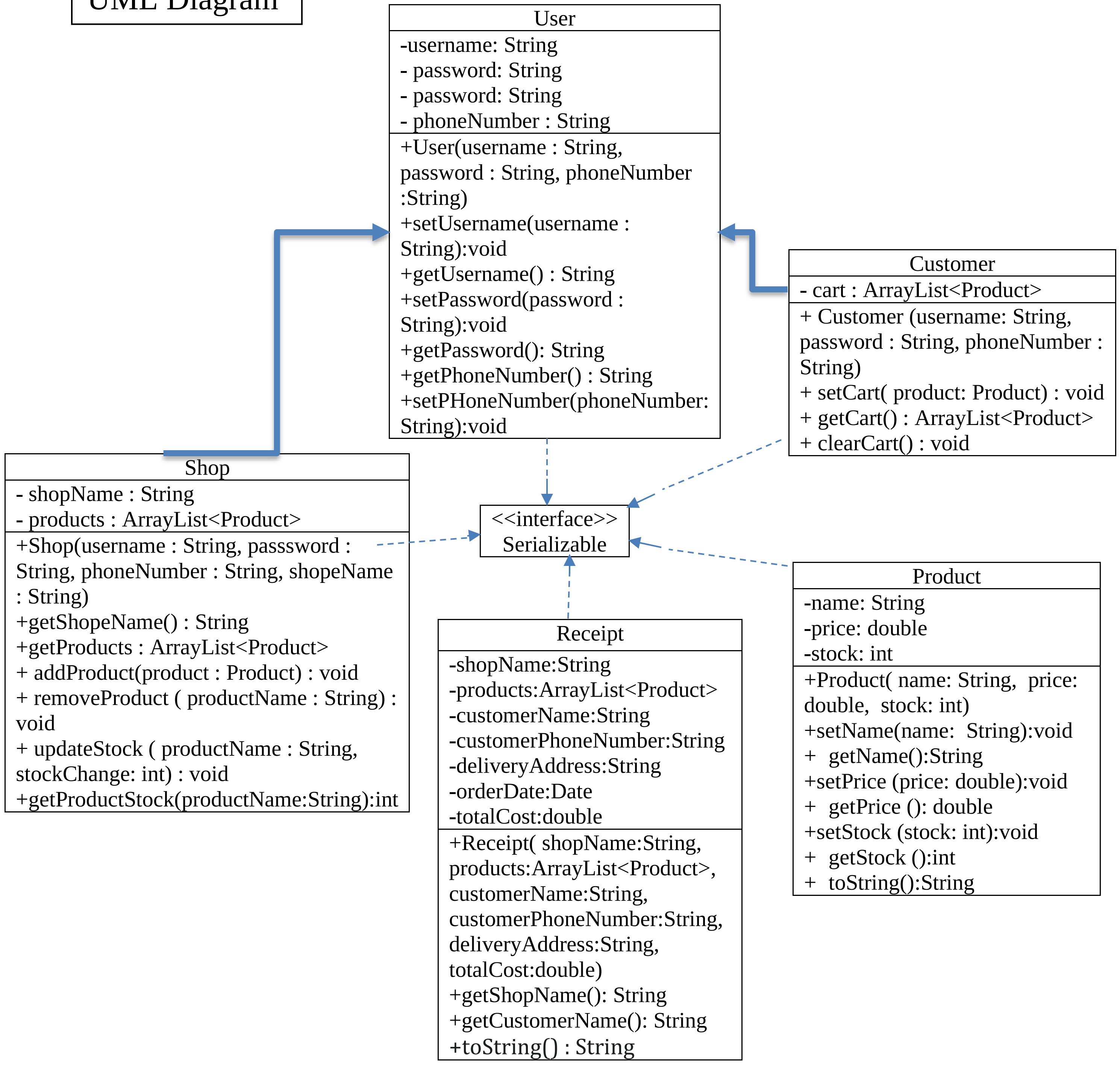
1.Account Creation: Shop owners must create an account to start selling.
2.Product Management: Shop owners can add products and update their stock levels.
3.**Receipt:** Shop owners can also see their sells through receipt.

Objective

The main objective of this project is to apply the fundamental concepts of Object-Oriented Programming (OOP) that we've learned throughout our CSE110 course. We aim to showcase our understanding and application of these principles through this project. Moreover, to design and implement an efficient, user-friendly platform that reflects the process of ordering and delivering groceries.

Methodology

UML Diagram


User
-username: String
- password: String
- password: String
- phoneNumber : String
+User(username : String, password : String, phoneNumber :String)
+setUsername(username : String):void
+getUsername() : String
+setPassword(password : String):void
+getPassword(): String
+getPhoneNumber() : String
+setPHoneNumber(phoneNumber: String):void
Customer
- cart : ArrayList<Product>
+ Customer (username: String, password : String, phoneNumber : String)
+ setCart(product: Product) : void
+ getCart() : ArrayList<Product>
+ clearCart() : void
Shop
- shopName : String
- products : ArrayList<Product>
+Shop(username : String, passsword : String, phoneNumber : String, shopeName : String)
+getShopeName() : String
+getProducts : ArrayList<Product>
+ addProduct(product : Product) : void
+ removeProduct (productName : String) : void
+ updateStock (productName : String, stockChange: int) : void
+getProductStock(productName:String):int
<<interface>>
Serializable
Receipt
-shopName:String
-products:ArrayList<Product>
-customerName:String
-customerPhoneNumber:String
-deliveryAddress:String
-orderDate:Date
-totalCost:double
+Receipt(shopName:String, products:ArrayList<Product>, customerName:String, customerPhoneNumber:String, deliveryAddress:String, totalCost:double)
+getShopName(): String
+getCustomerName(): String
+toString() : String
Product
-name: String
-price: double
-stock: int
+Product(name: String, price: double, stock: int)
+setName(name: String):void
+ getName():String
+setPrice (price: double):void
+ getPrice (): double
+setStock (stock: int):void
+ getStock ():int
+ toString():String

DataManager
- users : ArrayList<User> - shops : ArrayList<Shop> - receipts : ArrayList<Receipt>
+ DataManager() + addUser(user User) : void +loginShop(shop : Shop) : void +addShop(shop : Shop) : void +getShops() : ArrayList<Shop> +addReceipt (receipt: Receipt) : void +getReceiptsForShop(shopeName: String) : ArrayList<Receipt> +checkDuplicateAccount(username : String) +checkDuplicateShop(shopName : String) +getReceiptsForCustomer(customerName: String) : ArrayList<Receipt> +authenticate(username: String, password : String) : User - saveUsers() : void -saveShops() : void -saveReceipts(): void

CustomerUIController
- productListView: ListView<String> - receiptListView: ListView<String> - cartListView: ListView<String> - customer: Customer - shop: Shop - dataManager: DataManager - stage: Stage - loginStage: Stage - productList: ObservableList<String> - cartList: ObservableList<String> - receiptList: ObservableList<String> - totalAmount: double
+ initialize(): void + setCustomer(Customer): void + setShop(Shop): void + setDataManager(DataManager): void + setStages(Stage, Stage): void + handleAddToCart(): void + handlePlaceOrder(): void + handleLogout(): void - refreshProductList(): void - refreshCartList(): void + setTotalAmount(double): void + getTotalAmount(): double + show(): void - updateReceiptList(): void

FileManager
-SHOPS_FILE: String -USERS_FILE: String -RECEIPTS_FILE: String
+saveShops(shops:ArrayList<Shop>):void +loadShops():ArrayList<Shop> +saveUsers(users :ArrayList<User>):void +loadUsers(): ArrayList<User> +saveReceipts(receipts :ArrayList<Receipt>):void +loadReceipts(): ArrayList<Receipt>

AccountCreationUIController
- usernameField: TextField - passwordField: PasswordField |- confirmPasswordField: PasswordField - userTypeComboBox: ComboBox<String> - shopNameField: TextField - phoneNumberField: TextField - dataManager: DataManager - stage: Stage - loginStage: Stage
+ initialize(): void + setDataManager(DataManager): void + setStages(Stage, Stage): void + handleCreateAccount(): void + handleLogout(): void - showAlert(String): void

```
ShopUserUIController
---------------------------------------------------------
- nameField: TextField
- priceField: TextField
- stockField: TextField
- productListView: ListView<String>
- receiptListView: ListView<String>
- shop: Shop
- dataManager: DataManager
- stage: Stage
- loginStage: Stage
- productList: ObservableList<String>
- receiptList: ObservableList<String>
---------------------------------------------------------
+ initialize(): void
+ setDataManager(dataManager: DataManager): void
+ setStages(stage: Stage, loginStage: Stage): void
+ setUser(user: Shop): void
+ handleAddProduct(): void
+ handleRemoveProduct(): void
+ handleLogout(): void
+ show(): void
- refreshProductList(): void
- refreshReceiptList(): void
- showAlert(message: String): void
```

```
ShopSelectionUIController
---------------------------------------------------------
- shopListView: ListView<String>
- customer: Customer
- dataManager: DataManager
- stage: Stage
- loginStage: Stage
- shopList: ObservableList<String>
---------------------------------------------------------
+ initialize(): void
+ setCustomer(customer: Customer): void
+ setDataManager(dataManager: DataManager): void
+ setStages(stage: Stage, loginStage: Stage): void
+ handleSelectShop(): void
+ handleLogout(): void
- refreshShopList(): void
+ show(): void
```

```
LoginUIController
---------------------------------------------------------
- usernameField: TextField
- passwordField: PasswordField
- dataManager: DataManager|
- primaryStage: Stage
---------------------------------------------------------
+ setDataManager(dataManager: DataManager): void
+ setPrimaryStage(primaryStage: Stage): void
+ handleLogin(): void
+ handleCreateAccount(): void
```

Implementation

We have tried to implement all the concepts of Object-Oriented Programming throughout the project. Here are the key concepts that we have focused:

Encapsulation: We want to ensure our classes keep their data safe from outside interference. For example, a customer cannot access the data fields that have been reserved for the shop owner.

Inheritance: By establishing a class hierarchy, we can reuse code efficiently. The customer class in this program extends the user class by using the concept of inheritance.

Abstraction: We have also implemented the concepts of abstract class and abstract method in user class.

Interface: We tried to include the concept of interface by using serializable interface into the project.

Polymorphism: We have also covered the concept of Polymorphism so that a variable of super class can refer to its sub classes.

Exception Handling: We need to make sure our program can handle run time errors well. This means it should be able to deal with unexpected problems without crashing, so users have a good experience.

File I/**O Operations**: Our project will include reading from and writing to files. This allows us to persist data, making sure that our application can save and retrieve information as needed.

JavaFX: We also take the help of JavaFX to give a graphical structure to our project.

Reflection on Individual and Teamwork

All the members of our group (Raquib, Sadik and Sifat) have tried their best to contribute to the project. Their dedication and willing of work have reflected throughout the project.

Sadik plays an important role by creating a basic structure and implementing the classes of the project. He also has a contribution while debugging the code, implementing a Graphical User Interface, adding features, and modifying the code of our project.

Raquib plays a crucial role in designing and implementing the Graphical User Interface. He also reflects his contribution in debugging section, adding features to the project, implementing UML diagram, report writing and creating a PowerPoint presentation slide.

Sifat's contribution during this team-based project is highly appreciable. He reflects a significant impact in debugging, designing and establishing Graphical User Interface, debugging, implementing UML diagram, and helping during report writing.

Limitations and Future Work

Our program has a few limitations due to time constraints. We couldn't add some desired features like a "Forgot Password" option. Additionally, we didn't have enough time for thorough testing, so there might still be some bugs. Overall, there's a lot of room for improvement in this project.

References

We used ChatGPT to create the basic structure of our project. Later, we added all the necessary classes, methods, bug fixing, and features by ourselves to complete the project. We want to give an honorable mention to ChatGPT for its assistance with this project.