
SECJ2154 (OOP)

PROJECT REPORT

SECTION 01

MINI PROJECT TITLE: STOREMASTER RETAIL ORDERING SYSTEM

GROUP MEMBERS:

1. AISYAH BINTI MOHD NADZRI
2. AFIQAH IZZATI BINTI AZZEROL EFFENDI
3. MUHAMMAD IZAT BIN MD KAMIL
4. NURATHIRAH BINTI MUHAMAD ZAKI
5. MUHAMMAD HAZIQ BIN AZLI

TABLE OF CONTENT

1.0 Introduction	3
2.0 Problem Statement	4
3.0 Objectives	6
4.0 Project Design (Class diagram)	8
5.0 Project Output	9
6.0 Conclusion	13
7.0 Appendices	14

1.0 INTRODUCTION

StoreMaster is a software application that is used to manage the process of ordering items. It offers an online purchasing platform for the community which includes a range of features to support various aspects of the ordering process. The main objective of this report is to shed light on the importance and advantages of our retail ordering system. In this case, StoreMaster is able to improve efficiency of ordering items as well as provide varieties of items that can be selected in the inventory. By implementing the software, we can eliminate manual data entry as it is time consuming. This results in a significant boost of operational efficiency as well as reducing errors.

Furthermore, StoreMaster enables customers to place orders online conveniently as long as there is a good connection to the Internet. Purchases can be done online through mobile applications or any electronic devices like a laptop or a touch screen pad like an iPad. This offers flexibility and convenience to the user as purchases can be done anywhere and anytime. It is also worth mentioning that the real-time inventory visibility and accurate delivery estimation time contribute to a seamless customer experience, boosting satisfaction and loyalty of the customers.

A retail ordering system has several benefits to businesses looking to optimize their operations and provide excellent customer experiences in the increasingly competitive retail environment. Retailers can stay ahead of the curve and satisfy their consumers' changing expectations by automating operations, enhancing inventory management, and enabling data-driven decision making. Adopting a retail ordering system is now a requirement for shops who are serious about succeeding in the contemporary market.

2.0 PROBLEM STATEMENT

StoreMaster is a well-known store that is known for offering a diverse selection of apparel items. They have a high volume of customers and lots of inventory, which creates issues for them when it comes to handling their retail ordering process properly. StoreMaster wants to implement an ordering system so that they can better manage the many product orders coming in from customers. Just a few of the categories of goods that can be found at this retail store. The goal of this project is to construct a software application that will enable users to place orders, add things to their shopping carts, check the specifics of their orders, and figure out the total cost of their purchases.

The following are some of the current issues with the manual implementation of the existing system:

1. Processing orders in an inefficient way the manual installation of the retail ordering system in the clothes store results in processing orders in a manner that is both sluggish and inefficient. Recording client orders, determining totals, checking product availability, and keeping track of inventory all have to be done manually by the staff, which may be time consuming and leads to an increased risk of making mistakes.
2. When using a manual system, it might be difficult to monitor the current status of orders and communicate any revisions to clients. This can make order monitoring and communication more difficult. There is a lack of visibility into the progression of orders in real time.
3. Entering data manually increases the possibility of mistakes and can be inconsistent. Doing data input manually increases the possibility of mistakes, such as using the wrong product name, size, or quantity.

Functionalities suggested for implementation in the current system in order to fix the issues:

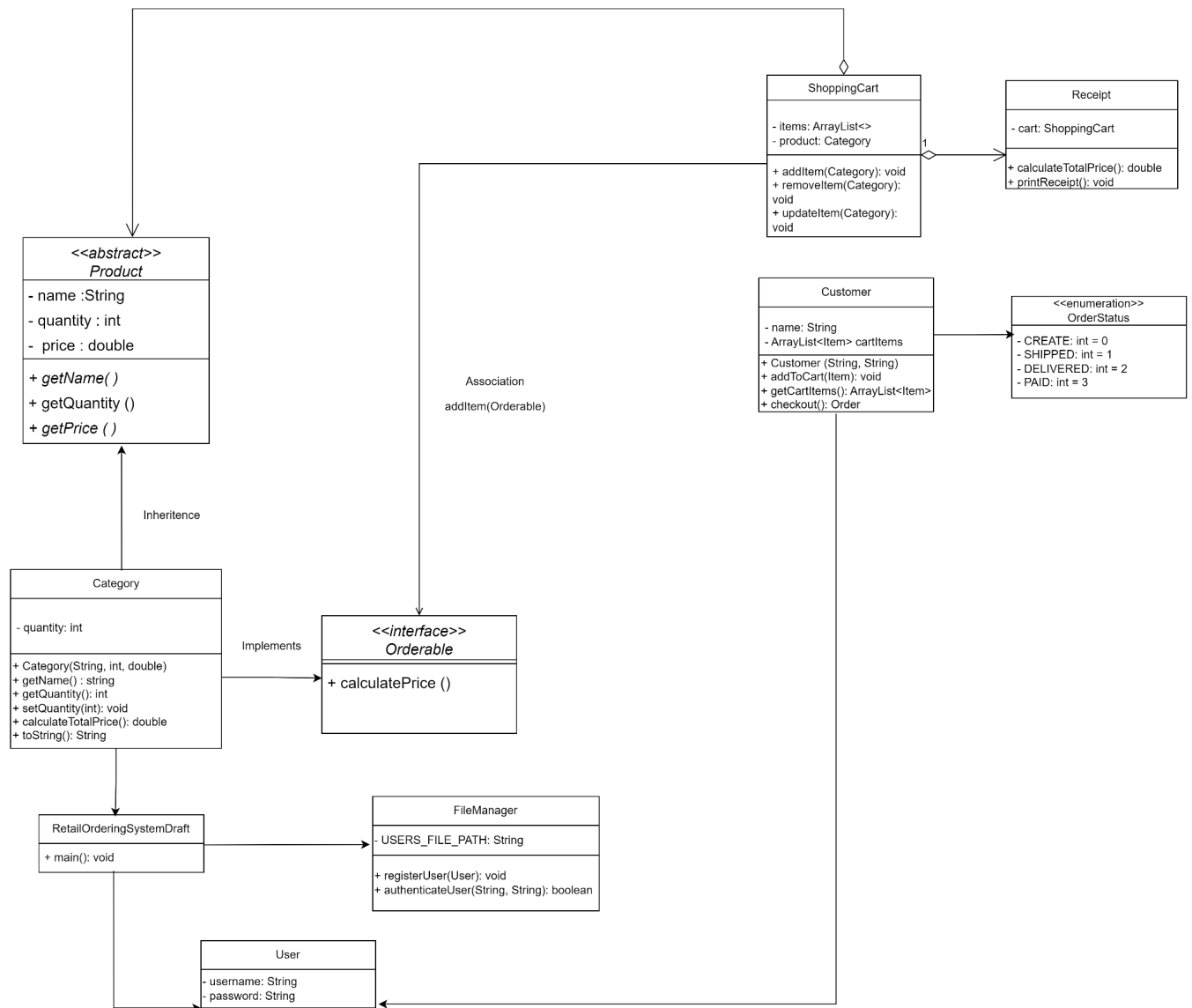
1. Order processing that simplified the system being suggested would automate the order processing, making it possible for employees to submit client orders in a timely and precise manner.
2. Communication and monitoring of the order the system allows for the tracking of orders throughout the entirety of the fulfilment process. The monitoring and transmission of orders This system allows for the tracking of orders during the full process of order fulfilment. As a result, the team of StoreMaster are able to seamlessly change order statuses, check progress, and communicate updates to clients order.
3. Validation of the data and consistency checks for data validity have been included in the system that is being proposed to ensure that data entry is accurate and consistent. It minimises the number of mistakes that might occur as a result of manually entering data by enforcing defined standards for product names, sizes, and quantities.

3.0 OBJECTIVES

1. Use a menu-driven interface that is easy for users to navigate so that customers can interact with the online retail system.
2. Enable customers to add items to their shopping cart, update quantities, and remove items from the cart.
3. Generate receipt that displays the order details, including item names, quantities, prices, and the total amount.
4. Enhance the display of item lists to include item numbers for easier selection by customers.
5. Implement error handling and validation using exception handling to handle invalid inputs and prevent program crashes.
6. Integrate a feature to display the available items with their respective prices before the menu is displayed.
7. Implement functionality to prevent customers from adding quantities greater than the available stock.
8. Develop a feature to calculate and display the subtotal for each item in the shopping cart.
9. Add a feature to display the quantity of each item in the shopping cart when listing the items.
10. Enhance the receipt format to include proper alignment and wider columns for better readability.
11. Implement input validation to ensure that item numbers entered by customers are within the valid range.
12. Develop a mechanism to handle empty shopping carts and display an appropriate message to customers.
13. Implement a feature to allow customers to view their current cart contents without making any changes.
14. Enhance the receipt generation feature to include the quantity of each item in the order details.
15. Implement a mechanism to handle item updates in the shopping cart, including changing quantities and updating item information.

16. Implement error handling for file I/O operations to provide meaningful error messages in case of failures.
17. Enhance the user experience by providing clearer prompts and error messages during registration and authentication.

4.0 PROJECT DESIGN (CLASS DIAGRAM)



5.0 PROJECT OUTPUT

Upon the execution of the program, these are the steps with their respective explanations. Users may also view all of the choices in the menu provided. They may add items, delete items, update items, view items, print receipt and make payment. They also may exit the system once they have completed their payment and order.

In figure 5.1, once the program has finished compiling, the output will display as below. Users will firstly have to register their details in which username and password before they are able to proceed to the next step. All the details will then be saved in the file as this part in the code is file implementation.

```
Enter username: Michael Jackson
Enter password: mj
User registration successful.
Enter username to authenticate: Michael Jackson
Enter password to authenticate: mj
Authentication successful.
```

Figure 5.1 User registration & authentication

In figure 5.2, the output shows the menu of the system. We have several choices that user can choose, namely add item to cart, delete item from cart, update item in cart, view cart, print receipt, make payment and exit.

```
Menu:
1. Add item to cart
2. Delete item from cart
3. Update item in cart
4. View cart
5. Print receipt
6. Make payment
7. Exit
```

Figure 5.2 User menu

In figure 5.3, we try out choice 1 which is to add an item to the cart. This part of the code, ArrayList is implemented. We use the add() method of ArrayList concept. Users may enter their desired items in the cart.

```
Enter your choice: 1

Available items:
1. Checkered Flannel (RM59.0)
2. Y2K Skirt (RM19.99)
3. Tank Top (RM10.49)

Enter item number: 2
Enter quantity: 1
```

Figure 5.3 User menu Choice - 1

In figure 5.4, we try out choice 2 which is to delete or remove an item from the cart. This part of the code, ArrayList is implemented. We use the remove() method of the ArrayList concept. Users may enter their desired items to remove in the cart.

```
Enter your choice: 2

Items in your cart:
1. Checkered Flannel x 2
2. Y2K Skirt x 1
3. Tank Top x 2

Enter item index to delete: 1
Item removed from the cart.
```

Figure 5.4 User menu Choice - 2

In figure 5.5, we try out choice 3 which is to update items in the cart. This part of the code, ArrayList is implemented. We use the set() method of ArrayList concept. Users may update their desired items in the cart to a new quantity.

```
Enter your choice: 3

Items in your cart:
1. Y2K Skirt x 1
2. Tank Top x 2

Enter item index to update: 1
Enter new quantity: 3
Item updated.
```

Figure 5.5 User menu Choice - 3

In figure 5.6, we try out choice 4 which is to view items in the cart. This allows users to view the items in the cart no matter the users only add items, delete items or even update items. All the changes made in the cart will be displayed when the users view the cart.

```
Enter your choice: 4

Items in your cart:
1. Y2K Skirt x 3
2. Tank Top x 2
```

Figure 5.6 User menu Choice - 4

In figure 5.7, we try out choice 5 which is to print the receipt. This allows users to view the items printed on the receipt. On the receipt, details of the item including name, quantity, order status and the price of items will be displayed. If the payment has not been made, order status will be printed pending otherwise if the payment has been completed, the order status will be updated to complete.

```
Enter your choice: 5

Details
=====
Item           Quantity  Price
-----
Y2K Skirt      3         RM19.99
Tank Top       2         RM10.49
-----

Total:                             RM80.95
=====
Order Status: PENDING
```

Figure 5.7 User menu Choice - 5

In figure 5.8, this choice allows users to make payment. Once users have done their payment, payment completed messages will be displayed to the user.

```
Enter your choice: 6

Payment completed. Order status: COMPLETED
```

Figure 5.8 User menu Choice - 6

In figure 5.9, if the users want to ensure the order status whether or not it has already updated from pending to completed, the users may enter choice 5 again to view the update. As we have made the payment, now the order status is updated to completed.

```
Enter your choice: 5

Details
=====
Item           Quantity  Price
-----
Y2K Skirt      3         RM19.99
Tank Top       2         RM10.49
-----

Total:                             RM80.95
=====
Order Status: COMPLETED
```

Figure 5.9 User menu Choice - 5

In figure 5.10, once the users have completed the order and payment, thus users now may enter choice 7 to exit the system.

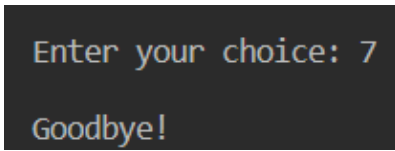
A screenshot of a terminal window with a dark background. The text 'Enter your choice: 7' is displayed on the first line, and 'Goodbye!' is displayed on the second line. The text is in a light-colored, monospaced font.

Figure 5.10 User menu Choice - 7

6.0 CONCLUSION

In conclusion, the implementation of the ordering system for StoreMaster has successfully addressed the challenges faced by the store in managing their retail ordering process. The software application developed enables users to conveniently place orders, add items to their shopping carts, and review the details of their orders. With this new system in place, StoreMaster can efficiently handle the high volume of customer orders and effectively manage their diverse inventory of apparel items.

Customers can now enjoy a seamless shopping experience, while the store benefits from improved order management and accurate cost calculations. Overall, the successful implementation of the ordering system has significantly enhanced StoreMaster's ability to serve their customers and streamline their retail operations.

7.0 APPENDICES

```
// SECJ2154-01 MINI PROJECT
// MINI PROJECT TITLE: STOREMASTER RETAIL ORDERING SYSTEM
//
// GROUP MEMBERS:
// 1. AISYAH BINTI MOHD NADZRI
// 2. AFIQAH IZZATI BINTI AZZEROL EFFENDI
// 3. MUHAMMAD IZAT BIN MD KAMIL
// 4. NURATHIRAH BINTI MUHAMAD ZAKI
// 5. MUHAMMAD HAZIQ BIN AZLI

import java.io.*;
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Scanner;

class FileManager {
    private static final String USERS_FILE_PATH = "users.txt";

    public void registerUser(String username, String password) {
        try {
            FileWriter writer = new FileWriter(USERS_FILE_PATH, true);
            writer.write(username + ":" + password + "\n");
            writer.close();
            System.out.println("User registration successful.");
        } catch (IOException e) {
            System.out.println("An error occurred while registering the
user.");
        }
    }

    public boolean authenticateUser(String username, String password) {
        try {
            File file = new File(USERS_FILE_PATH);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(":");
            }
        }
    }
}
```

```

        if (parts.length == 2 && parts[0].equals(username) &&
parts[1].equals(password)) {
            scanner.close();
            return true;
        }
    }
    scanner.close();
} catch (FileNotFoundException e) {
    System.out.println("User file not found.");
}
return false;
}
}

class User {
    private String username;
    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }
}

interface Orderable {
    double calculateTotalPrice();
}

abstract class Product {
    protected String name;
    protected double price;
}

```

```

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public abstract int getQuantity();

    public abstract void setQuantity(int quantity);
}

class Category extends Product implements Orderable {
    private int quantity;

    public Category(String name, double price) {
        super(name, price);
        this.quantity = 0;
    }

    @Override
    public int getQuantity() {
        return quantity;
    }

    @Override
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public double calculateTotalPrice() {
        return price * quantity;
    }
}

```



```

    @Override
    public String toString() {
        return name + " (RM" + price + ")";
    }
}

class ShoppingCart {
    private ArrayList<Product> items;

    ShoppingCart() {
        items = new ArrayList<>();
    }

    public ArrayList<Product> getItems() {
        return items;
    }

    public void addItem(Product item, int quantity) {
        item.setQuantity(quantity);
        items.add(item);
        //System.out.println("Item added to the cart.");
    }

    public void removeItem(int index) {
        if (index >= 0 && index < items.size()) {
            items.remove(index);
            System.out.println("Item removed from the cart.");
        } else {
            System.out.println("Invalid item index.");
        }
    }

    public void updateItem(int index, Product item, int quantity) {
        if (index >= 0 && index < items.size()) {
            item.setQuantity(quantity);
            items.set(index, item);
            System.out.println("Item updated.");
        } else {
            System.out.println("Invalid item index.");
        }
    }
}

```

```

    }
}

public void displayCart() {
    if (items.isEmpty()) {
        System.out.println("Your cart is empty.");
    } else {
        System.out.println("Items in your cart:");
        for (int i = 0; i < items.size(); i++) {
            Product item = items.get(i);
            int quantity = item.getQuantity();
            System.out.println(i + 1 + ". " + item.getName() + " x " +
quantity);
        }
    }
}

enum OrderStatus {
    PENDING,
    COMPLETED
}

class Receipt {
    private ShoppingCart cart;
    private OrderStatus orderStatus;

    public Receipt(ShoppingCart cart) {
        this.cart = cart;
        this.orderStatus = OrderStatus.PENDING;
    }

    public ShoppingCart getCart() {
        return cart;
    }

    public OrderStatus getOrderStatus() {
        return orderStatus;
    }
}

```

```

    public void setOrderStatus(OrderStatus orderStatus) {
        this.orderStatus = orderStatus;
    }

    public double calculateTotalPrice() {

        double totalPrice = 0.0;
        for (Product item : cart.getItems()) {
            //System.out.printf("%-25s %-10d RM%.2f\n", item.getName(),
item.getQuantity(), item.getPrice());
            totalPrice += item.getPrice() * item.getQuantity(); //!nanti
tukar guna Orderable class
        }

        return totalPrice;
    }

    public void printReceipt() {
        System.out.println("Details");

System.out.println("=====");
        System.out.printf("%-25s%-10s%s\n", "Item", "Quantity", "Price");

System.out.println("-----");
        for (Product item : cart.getItems()) {
            int quantity = item.getQuantity();
            double price = item.getPrice();
            System.out.printf("%-25s%-10dRM%.2f\n", item.getName(),
quantity, price);
        }

System.out.println("\n-----");
        System.out.printf("%-35sRM%.2f\n", "Total:",
calculateTotalPrice());

System.out.println("=====");
        System.out.println("Order Status: " + orderStatus);
    }

    public void makePayment() {

```

```

        orderStatus = OrderStatus.COMPLETED;
        System.out.println("Payment completed. Order status: " +
orderStatus);
    }
}

public class R11 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        FileManager fileManager = new FileManager();

        System.out.print("Enter username: ");
        String username = scanner.nextLine();

        System.out.print("Enter password: ");
        String password = scanner.nextLine();

        // User registration
        User user = new User(username, password);
        fileManager.registerUser(user.getUsername(), user.getPassword());

        // User authentication
        System.out.print("Enter username to authenticate: ");
        String authUsername = scanner.nextLine();

        System.out.print("Enter password to authenticate: ");
        String authPassword = scanner.nextLine();

        boolean isAuthenticated =
fileManager.authenticateUser(authUsername, authPassword);

        if (isAuthenticated) {
            System.out.println("Authentication successful.");
        } else {
            System.out.println("Authentication failed.");
        }

        // Create some sample products
        Category item1 = new Category("Checkered Flannel", 59.00);
        Category item2 = new Category("Y2K Skirt", 19.99);
    }
}

```

```

Category item3 = new Category("Tank Top", 10.49);

// Create a shopping cart
ShoppingCart cart = new ShoppingCart();

// Create a receipt
Receipt receipt = new Receipt(cart);

// Add sample products to the cart
cart.addItem(item1, 1);
cart.addItem(item2, 2);
cart.addItem(item3, 3);

//scanner.close();

// Main program loop
while (true) {
    System.out.println("\nMenu:");
    System.out.println("1. Add item to cart");
    System.out.println("2. Delete item from cart");
    System.out.println("3. Update item in cart");
    System.out.println("4. View cart");
    System.out.println("5. Print receipt");
    System.out.println("6. Make payment");
    System.out.println("7. Exit");

    try {
        System.out.print("\nEnter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.println("\nAvailable items:");
                System.out.println("1. " + item1);
                System.out.println("2. " + item2);
                System.out.println("3. " + item3);

                System.out.print("\nEnter item number: ");
                int itemNumber = scanner.nextInt();

```

```

        System.out.print("Enter quantity: ");
        int quantity = scanner.nextInt();

        switch (itemNumber) {
            case 1:
                cart.addItem(item1, quantity);
                break;
            case 2:
                cart.addItem(item2, quantity);
                break;
            case 3:
                cart.addItem(item3, quantity);
                break;
            default:
                System.out.println("Invalid item
number.");

                break;
        }
        break;

    case 2:
        System.out.println();
        cart.displayCart();

        System.out.print("\nEnter item index to delete:
");

        int index = scanner.nextInt();
        cart.removeItem(index - 1);
        break;

    case 3:
        System.out.println();
        cart.displayCart();

        System.out.print("\nEnter item index to update:
");

        int itemIndex = scanner.nextInt();

        System.out.print("Enter new quantity: ");
        int newQuantity = scanner.nextInt();

```

```

        Product itemToUpdate =
cart.getItems().get(itemIndex - 1);
        cart.updateItem(itemIndex - 1, itemToUpdate,
newQuantity);

        break;

    case 4:
        System.out.println();
        cart.displayCart();
        break;

    case 5:
        System.out.println();
        receipt.printReceipt();
        break;

    case 6:
        System.out.println();
        receipt.makePayment();
        break;

    case 7:
        System.out.println("\nGoodbye!");
        System.exit(0);
        break;

    default:
        System.out.println("\nInvalid choice. Please try
again.");

        break;
    }
} catch (InputMismatchException e) {
    System.out.println("\nInvalid input. Please enter a
number.");

    scanner.nextLine(); // Clear the input buffer
}
}
}
}

```

