OLAJARE OLABODE

FOOD ORDERING SYSTEM

DATA STRUCTURES & OBJECT-ORIENTED PROGRAMMING
420-202-RE

TABLE OF CONTENTS

PROJECT DESCRIPTION

I used IntelliJ to create this project which simulates the process of ordering food in a simple and interactive way. The system has three main roles: Admin, Customer, and Driver. The Admin is responsible for managing the menu by adding new food items or removing the ones that are no longer available. The Customer can view the menu, select the items they want, and place an order. Once an order is placed, Drivers can view the list of available orders and choose which ones to accept and deliver.

Working on this project helped me understand the application of some programming concepts. I used object-oriented programming (OOP) techniques such as inheritance, abstraction, and polymorphism to design the different types of users and their behaviors. Interfaces were also important in structuring the system and making it more efficient.

In addition, I made use of file input and output (I/O) to save and load menu items and order receipts, so that data can be reused even after the program is closed. I also made use of Collections (ArrayList, LinkedList, etc.), Stream, Lambda Expressions to filter, group, and manage data (food items, orders, etc.) more efficiently.

JUnit testing as well as Git repository were necessary in order to push a well functioning code with well-handled exceptions.

PROGRAM FEATURES & SCREENSHOTS

- **Admin**
  - Adding/Removing a food item by its name

```java
/**
 * Removing food item from menu
 * @param menu the menu to be changed
 * @param itemName the item to be removed
 */
public void removeFoodItem(Menu menu, String itemName) {  4 usages  olajireolabode
    if (itemName != null) {
        // finding the item in menu
        FoodItem toRemove = menu.findItemByName(itemName);
        if (toRemove != null) {
            // if found, remove from menu
            menu.removeItem(itemName);
            System.out.println(itemName + " removed.");
        } else {
            // item not found in menu
            System.out.println(itemName + " is not in the menu.");
        }
    } else {
        //invalid item name
        System.out.println("Invalid item name.");
    }
}
```

```java
/**
 * Adding food item to menu
 * @param menu the menu to be changed
 * @param item the item to be added
 */
public void addFoodItem(Menu menu, FoodItem item) {  3 usages  olajireolabode
    if (item != null && menu != null) {
        menu.addItem(item);
    } else {
        // notify if input is valid
        System.out.println("No item/menu.");
    }
}
```

  - Viewing the menu

```java
/**
 * Allows admin to view menu
 * @param menu the menu to be viewed
 */
@Override  2 usages  olajireolabode
public void viewMenu(Menu menu) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("\nAdmin - Menu:" );

    while (true) {
        System.out.println("1. Sort by Name  \n2. Sort by Price \n3. Update menu" );
        System.out.print("Choose an option: " );
        String choice = scanner.nextLine();

        // handling admins choices
        switch (choice) {
            case "1" -> {
                // if menu is empty, alert the admin
                if (menu.getItems().isEmpty()) {
                    System.out.println("The menu is empty!");
                } else { // sort menu alphabetically by item name
                    menu.sortByName();
                }
            }
            case "2" -> {
                // if menu is empty, alert the admin
                if (menu.getItems().isEmpty()) {
                    System.out.println("The menu is empty.");
                } else { // sort menu by price in ascending order
                    menu.sortByPrice();
```

```java
                    System.out.println("The menu is empty!");
                } else { // sort menu alphabetically by item name
                    menu.sortByName();
                }
            }
            case "2" -> {
                // if menu is empty, alert the admin
                if (menu.getItems().isEmpty()) {
                    System.out.println("The menu is empty.");
                } else { // sort menu by price in ascending order
                    menu.sortByPrice();
                }
            }
            case "3" -> System.out.println("Here's what's available:" );
            default -> {
                // if input is invalid, prompt this message
                System.out.println("Invalid option. Choose 1,2, or 3." );
                continue; // going back to the start of the loop
            }
        }
        break; // exiting loop after valid input
    }
    menu.getItems() // printing final menu
            .forEach( FoodItem item -> System.out.println(String.format("- %s: $%.2f", item.getName(), item.getPrice())));
}
```

- **Customer**
  - Viewing the menu

```java
/**
 * Allows customer to view menu
 * @param menu the menu to be viewed
 */
@Override  2 usages  olajireolabode
public void viewMenu(Menu menu) {
    System.out.println("\nCustomer - Menu:");
    menu.getItems()
            .forEach( FoodItem item ->
                    System.out.println(String.format("- %s: $%.2f ", item.getName(), item.getPrice())));
}
```

  - Placing an order

```
        case "remove" -> {
            System.out.print("Enter item to remove: ");
            String itemName = scanner.nextLine().trim();
            // finding item name in order before removing it
            FoodItem item = menu.findItemByName(itemName);

            if (item != null && order.getOrderedItems().containsKey(item)) {
                order.removeItem(item);
                System.out.println(item.getName() + " removed from your order.");
            } else {
                System.out.println("Item is not in your order.");
            }
        }

        case "done" -> ordering = false; // exiting the loop. customer is done

        default -> System.out.println("Invalid option. Enter add, remove, or done.");
    }
}
```

```
            if (item != null) {
                System.out.print("How many would you like? ");
                try {
                    int qty = Integer.parseInt(scanner.nextLine());
                    if (qty > 0) {
                        // adding the item to the order with the accurate quantity
                        for (int i = 0; i < qty; i++) {
                            order.addItem(item);
                        }
                        System.out.println(qty + "x " + item.getName() + " added to your order.");
                    } else {
                        System.out.println("Quantity must be at least 1.");
                    }
                } catch (NumberFormatException e) {
                    System.out.println("Invalid number.");
                }
            } else {
                System.out.println("Item is not in the menu.");
            }
        }

        case "remove" -> {
            System.out.print("Enter item to remove: ");
            String itemName = scanner.nextLine().trim();
            // finding item name in order before removing it
            FoodItem item = menu.findItemByName(itemName);

            if (item != null && order.getOrderedItems().containsKey(item)) {
                order.removeItem(item);
                System.out.println(item.getName() + " removed from your order.");
```

```
/**
 * Allows customer to place order
 * @param menu from where the order is placed
 */
@Override  1 usage  olajireolabode
public void placeOrder(Menu menu) {
    // if menu is empty, cancel order
    if (menu.getItems().isEmpty()) {
        System.out.println("\nThe menu is empty. Try again later.");
        return;
    }

    Scanner scanner = new Scanner(System.in);
    boolean ordering = true;

    System.out.println(); // skip a line

    // runs as long as customer is not done ordering
    while (ordering) {
        System.out.print("Do you wish to add or remove an item? (add/remove/done): ");
        String action = scanner.nextLine().toLowerCase();

        switch (action) {
            case "add" -> {
                System.out.print("Enter item name to add: ");
                String itemName = scanner.nextLine();
                // finding item name in menu
                FoodItem item = menu.findItemByName(itemName);
```

- Generating receipt

```java
//calculating and printing the total
double total = order.calculateTotal();
System.out.println("\nOrder complete!");
System.out.printf("Total: $%.2f ", total);
System.out.println("\nThanks for ordering! Come again soon.");

// printing receipt
Receipt.printReceipt(order);
```

- **Driver**
    - View available orders to be delivered

```java
/**
 * allows driver to view the orders to be delivered
 */
public void viewDeliveryOrders() {  1 usage   olajireolabode
    // print out message if there are no orders to be delivered
    if (deliveryOrders.isEmpty()) {
        System.out.println("\nNo orders to be delivered by " + username + ".");
        return;
    }

    // print out orders to be delivered
    System.out.println("\nOrders to be delivered by: " + username);
    for (int i = 0; i < deliveryOrders.size(); i++) {
        System.out.println("Order #" + (i + 1));
        Receipt.printReceipt(deliveryOrders.get(i));
    }
}
```

    - Accepting/Declining orders to be delivered

```
while (true) {
    System.out.print("Enter order number to accept or '0' to cancel: " );
    String input = scanner.nextLine();

    try {
        int selection = Integer.parseInt(input);

        if (selection == 0) {
            System.out.println("No order accepted." );
            break;
        } else if (selection >= 1 && selection <= availableOrders.size()) {
            Order selectedOrder = availableOrders.remove( index: selection - 1);
            driver.acceptOrder(selectedOrder);
            break;
        } else {
            System.out.println("Invalid order number. Enter 'yes' to try again or 'no' to exit: " );
            String answer = scanner.nextLine().toLowerCase();
            if (!answer.equals("yes" )) {
                System.out.println("Order selection cancelled." );
                break;
            }
        }
    } catch (NumberFormatException e) {
        System.out.println("Please enter a valid number." );
        System.out.println(" Enter 'yes' to try again or 'no' to exit: ");
        String answer = scanner.nextLine().toLowerCase();
        if (!answer.equals("yes" )) {
            System.out.println("Order selection cancelled." );
            break;
        }
```

```
/**
 * allows the driver to accept a delivery order
 * @param order the order to be accepted by the driver
 */
public void acceptOrder(Order order) {  5 usages  olajireolabode
    // ensuring the order is not null. can not accept a null order
    if (order != null) {
        deliveryOrders.add(order);
        System.out.println(username + " has accepted the order!");
    }
}
```

- **Hierarchy**

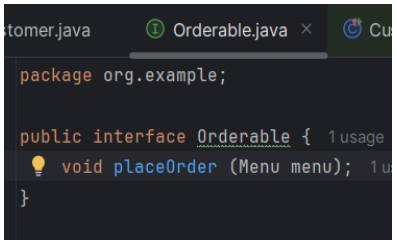
```
import java.util.Scanner;

public class Customer extends User
```

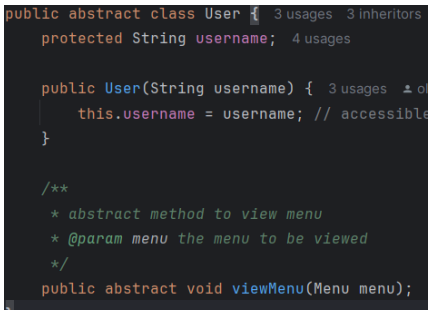
```
public class Admin extends User {  12 usages  olajireolabode
    public Admin(String username) {  8 usages  olajireolabode
```

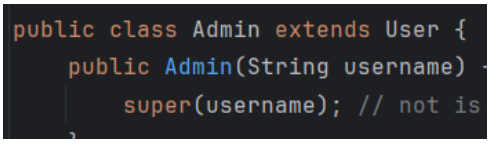
```
public abstract class User {  3 usages  3 inheritors  olajireolabode
    protected String username;  4 usages
```

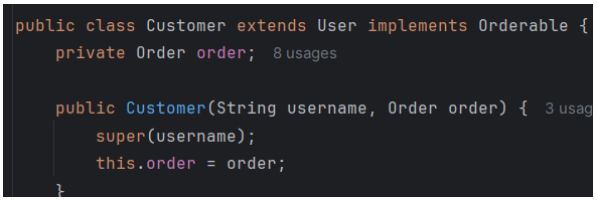
- **User-defined interface**

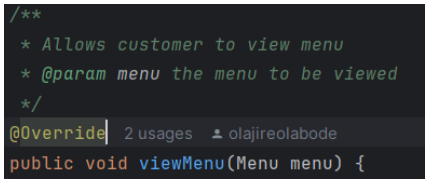


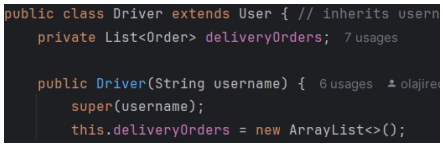
- **Polymorphism**











- **TextIO**

```java
public class FileManager {  no usages  olajireolabode
    // path to csv file
    private static final String MENU_FILE_PATH = "src/main/resources/menu.csv";

    /**
     * reading menu from a csv file
     * @return list of food items read from the file
     */
    public static List<FoodItem> loadMenu() {  no usages  olajireolabode
        List<FoodItem> items = new ArrayList<>();
        File file = new File(MENU_FILE_PATH);

        try (Scanner scanner = new Scanner(file)) {
            // reading each line from the file
            while (scanner.hasNext()) {
                String line = scanner.nextLine();
                String[] parts = line.split( regex: ",");
                // making sure the line has two parts
                if (parts.length == 2) {
                    String name = parts[0].trim();
                    double price = Double.parseDouble(parts[1].trim());
                    // creates and adds the food item to the list
                    items.add(new FoodItem(name, price));
                }
            }
        } catch (FileNotFoundException e) {
            // exception if file can't be found
            System.out.println("File not found.");
        }
```

```java
            }
        }
    } catch (FileNotFoundException e) {
        // exception if file can't be found
        System.out.println("File not found.");
    }

    return items; // returning the list
}

/**
 * Saves menu items to a csv file
 * @param items the food items to save
 */
public static void saveMenu(Collection<FoodItem> items) {  no usages  olajireolabode
    File file = new File(MENU_FILE_PATH);

    try (FileWriter fw = new FileWriter(file)){
        // writing each food item as a line in the file
        for (FoodItem item : items) {
            fw.write( str: item.getName() + "," + item.getPrice() + "\n");
        }
    } catch (IOException e) {
        // exception for writing error
        throw new RuntimeException("Failed to save menu.", e);
    }
}
```

```java
/**
 * Saves an order to a csv file
 * @param order the order to save
 */
public static void saveOrder(Order order) {  no usages  olajireolabode
    File file = new File(MENU_FILE_PATH);

    try (FileWriter fw = new FileWriter(file)) {
        System.out.println("Item,Quantity,Price:");
        // looping through the order and printing each entry in csv form
        for (Map.Entry<FoodItem, Integer> entry : order.getOrderedItems().entrySet()) {
            FoodItem item = entry.getKey();
            int quantity = entry.getValue();
            double price = item.getPrice();

            // printing order details to console
            System.out.printf("%s,%d,%.2f\n", item.getName(), quantity, price);
        }
    } catch (IOException e) {
        System.out.println("Failed to save order.");
    }
}
```

- **Comparable/Comparator**

```java
import java.util.Comparator;
import java.util.Objects;

public class FoodItem implements Comparable<FoodItem> {
```

```java
@Override  olajireolabode
public int compareTo(FoodItem other) { return this.name.compareTo(other.name); }
```

```java
class PriceComparator implements Comparator<FoodItem> {  1 usage  olajireolabode
    @Override  olajireolabode
    public int compare(FoodItem o1, FoodItem o2) { return Double.compare(o1.getPrice(), o2.getPrice()); }
}
```

- **JUnit testing**
  - **Admin**

    Example:

```java
/**
 * adding food item to menu
 */
@Test  olajireolabode
public void testAddFoodItem1() {
    Admin admin = new Admin( username: "Admin");
    Menu menu = new Menu();
    FoodItem burger = new FoodItem( name: "Burger",  price: 4.00);

    admin.addFoodItem(menu, burger);

    boolean contains = menu.getItems().contains(burger);
    Assertions.assertTrue(contains,  message: "Admin added burger to the menu");
}
```

  - **Customer**

    Example:

```java
/**
 * customer adds two items to order
 */
@Test  olajireolabode
void testAddItemToOrder() {
    Order order = new Order();
    Menu menu = new Menu();

    FoodItem burger = new FoodItem( name: "Burger",  price: 4.00);
    FoodItem fries = new FoodItem( name: "Fries",  price: 2.50);

    menu.addItem(burger);
    menu.addItem(fries);

    order.addItem(burger);
    order.addItem(burger);

    assertEquals( expected: 2, order.getOrderedItems().get(burger));
    assertEquals( expected: 8.00, order.calculateTotal(),  delta: 0.01);
}
```

  - **Driver**

    Example:

```java
/**
 * driver accepting an order
 */
@Test  olajireolabode
public void testAcceptOrder1() {
    Driver driver = new Driver( username: "Olajare");
    Order order = new Order();
    FoodItem burger = new FoodItem( name: "Burger",  price: 4.00);
    order.addItem(burger);

    driver.acceptOrder(order);

    List<Order> toDeliver = driver.getDeliveryOrders();

    Assertions.assertEquals( expected: 1, toDeliver.size(),  message: "Driver has 1 order to deliver.");
    Assertions.assertTrue(toDeliver.contains(order),  message: "Order to deliver is the accepted order.");
}
```

- **Menu**

  Example:

```java
/**
 * adding a single item
 */
@Test  olajireolabode
public void testAddItem1() {
    Menu menu = new Menu();
    FoodItem burger = new FoodItem( name: "Burger",  price: 4.00);

    menu.addItem(burger);

    boolean containsBurger = menu.getItems().contains(burger);
    assertTrue(containsBurger,  message: "Added burger."); // instead of assertEquals(true,..,..)
}
```

- **Order**

  Example:

```java
/**
 * ordering item that is on the menu
 */
@Test  olajireolabode
public void testAddItem1() {
    Order order = new Order();
    FoodItem burger = new FoodItem( name: "Burger",  price: 4.00);

    order.addItem(burger);

    boolean containsBurger = order.getOrderedItems().containsKey(burger);
    int quantity = order.getOrderedItems().get(burger);

    assertTrue(containsBurger,  message: "Burger added in order" );
    Assertions.assertEquals( expected: 1, quantity,  message: "Burger x1");
}
```

CHALLENGES FACED

1. Handling empty inputs was one of them. Input validation was required to prevent exceptions when blank names or prices were entered.
2. Also, avoiding duplicate items in order proved to be difficult. Initially, the same item was added multiple times instead of increasing its quantity. This was later resolved using a <Map<FoodItem, Integer>> for quantity tracking.
3. In addition, I had to ensure that once a user has made progress within a specific section of the menu, they are not redirected back to the main menu and forced to restart the process. I had to do work on this in all the User classes like Admin, Customer, and Driver. At some point, given that I'd have to add a loop in an already coded section, I was struggling to figure out where to place the curly brackets.
4. I had to decide whether to proceed with the Receipt class or not. I eventually gave in as it would help the efficiency of my code.
5. I also had to make a new or edit my JUnit testing whenever I made a major change to my code in a particular class.
6. All in all, I basically had to come up with solutions to issues I found along the way. It felt like the further I went, the more the issues were coming as well.

LEARNING OUTCOMES

- Stronger command of Java OOP concepts, particularly abstract classes, polymorphism, and class hierarchies.

- Experience with file I/O, using the Scanner for reading and FileWriter for saving structured data in .csv format.
- Getting more familiar with exception handling.

- Stream and collection usage, especially Java Streams, Maps, and Lists for filtering and grouping data.

- JUnit Testing principles, excluding simple getters/setters and focusing on more important logic testing.

- Basic Git/GitHub workflows, including committing, pushing code, and updating documentation. Also managing files like README.md.

This project really made me think harder than I normally would. I was bringing up problems with my own code and finding the solutions at the same time. It made me think of how to create a smooth running system with a wide variety of exceptions checked off the list.