IPRO001
Programming Project

: Scenario & Explanation of PizzaShop.java

Name: **Ina Song**

Student ID: **14556068**

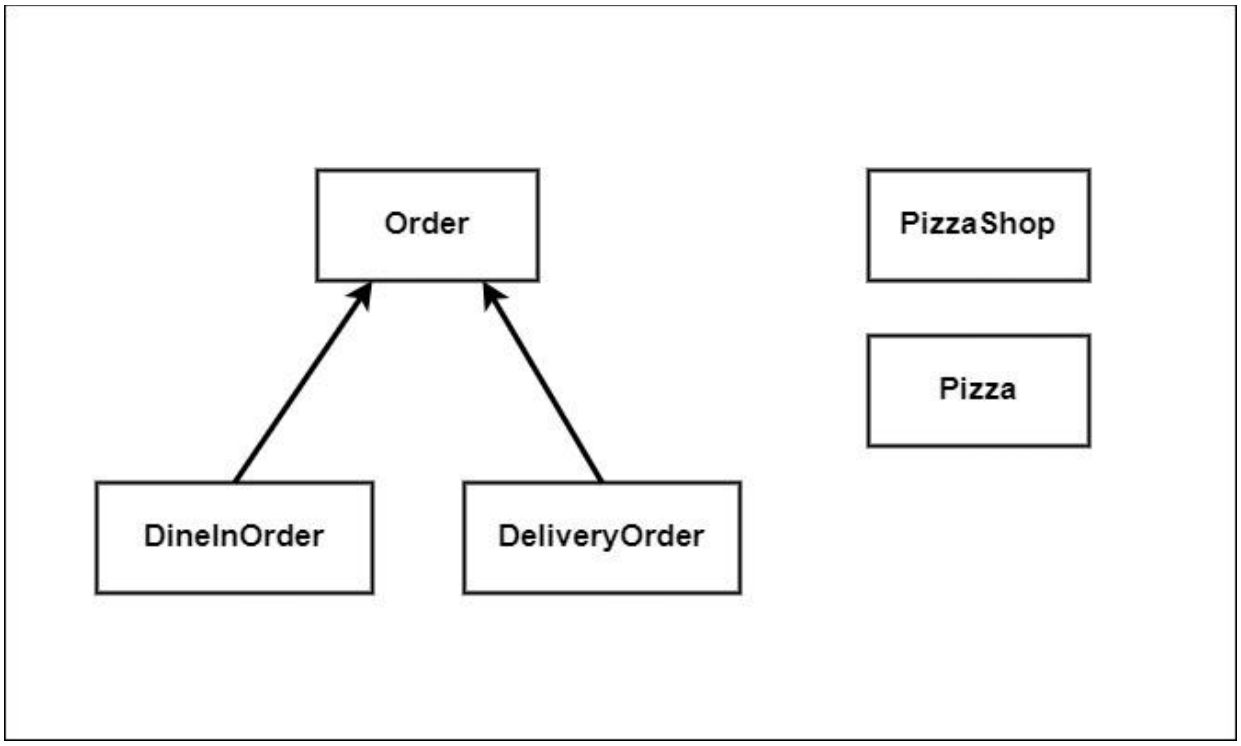
Tutor: **Hazel Kim**

Date: **26 / 04 / 2024**

Table of Contents

Scenario

This program simulates the ordering and management processes within a pizza shop. It features a parent 'Order' class that represents orders at a real-world pizza shop, containing multiple pizzas ordered. This superclass is extended by two subclasses, DineOrder and DeliveryOrder, each designed to handle specific types of orders. These subclasses inherit common attributes from the parent class while also introducing unique characteristics relevant to their respective order types. The system continuously runs until the user selects "Close system" from a home menu. Within this framework, users can either place new orders according to their preferences or, from an administrative perspective, manage and cancel orders by type. This setup efficiently mirrors the comprehensive operations in a pizza shop, integrating user interaction with backend order management.



Explanation

1. Basic Class Structure

1) **PizzaShop**: Main class handling the logic for the ordering platform.
2) **Order**: Superclass for different types of orders (represent the order of single or multiple pizzas made by the user).
3) **DineInOrder and DeliveryOrder**: Subclasses of **Order** for specific order types.
4) **Pizza**: Represents a pizza with specific attributes like name and price.

2. Attributes

1) **PizzaShop**
   - **HashMap<Integer, Pizza> pizzaMenu**: Stores pizzas available in the shop keyed by a menu number(Integer)
   - **ArrayList<DineInOrder> dineInOrders**: List of orders for customers dining in
   - **ArrayList<DeliveryOrder> deliveryOrders**: List of orders for customers requesting delivery
2) **Order**
   - **ArrayList<Pizza> pizzaOrder:** List of pizzas selected by the customer in one order
   - **double totalPrice:** Total price of all the pizzas added to the order
3) **DineInOrder**
   - **int tableNumber**: Specific to dine-in order to track table assignment
4) **DeliveryOrder**
   - **String address**: Specific to delivery order to track address for delivery
   - **double deliveryFee**: Specific to delivery order to add a charge for delivery based on total price
5) **Pizza**
   - **String name:** name(type) of the pizza
   - **double basePrice:** the base price of the pizza(medium size)
   - **char size:** size of the pizza which is set by user choice(input)

3. Constructor

1) **PizzaShop()**
   - Initialises **pizzaMenu(HashMap)** with predefined pizzas(representing menu options)
   - Instantiates **dineInOrders(ArrayList)** and **deliveryOrders(ArrayList)**

2) **Order()**
   - Instantiates **Pizzaorder(ArrayList)** and initialises totalPrice
3) **DineInOrder (int tableNumber)**
   - Calls the superclass(Order) constructor and initialises tableNumber attribute with the parameter
4) **DeliveryOrder (String address)**
   - Calls the superclass(Order) constructor and initialises address attribute with the parameter, and sets deliveryFee to be 5.0
5) **Pizza (String name, double basePrice)**
   - Initialises name and basePrice with the parameters and set the size to be 'M'

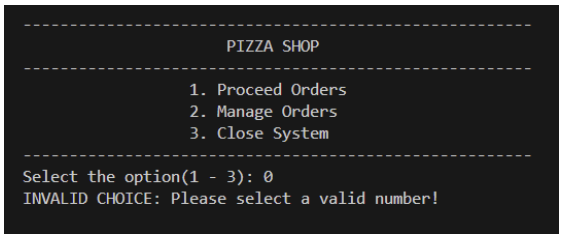
4. Important Methods

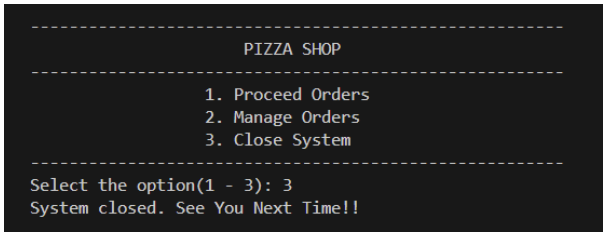
1) PizzaShop Class

(1) main method

The main method serves as the central control mechanism for the PizzaShop program.

It begins by creating an instance of the PizzaShop class and enters a loop that continuously prompts the user with the main menu, offering options to proceed with placing orders, manage the proceeded orders, or exit the program. If option 1 or 2 is selected, the proceedOrder() or manageOrder() method is called respectively. The loop ensures the program remains active and responsive until the user select option 3 to terminate the system.





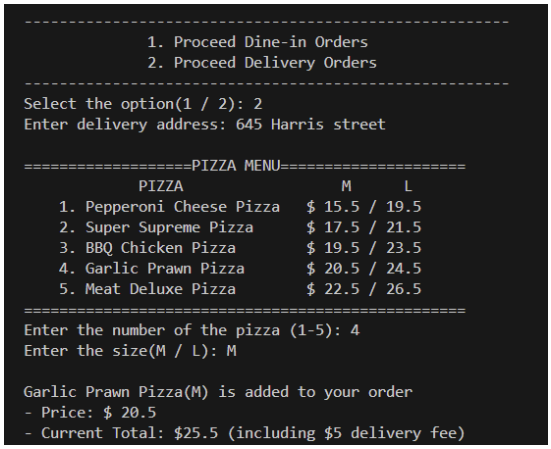
(2) proceedOrder()

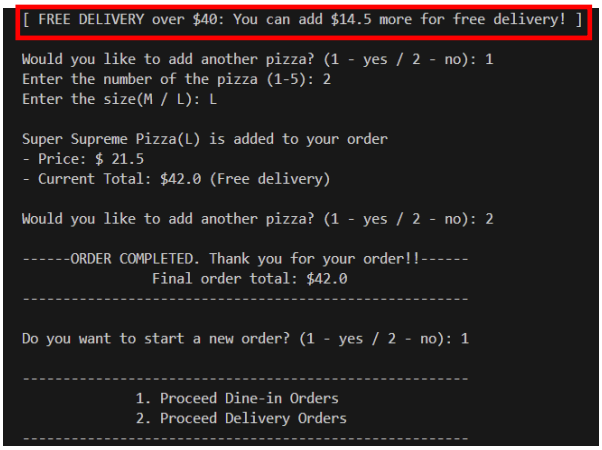
proceedOrder() method is used to handle the customer's entire ordering process within the pizza shop system.

It begins by prompting the user to select between dine-in and delivery options. Depending on the choice, it either requests a table number for dine-in orders or an address for delivery orders. Then, it displays a menu by calling showMenu() method, and the type and size of the pizza is selected by user input. The selected pizza is added to the ArrayList in respective order object, and its details with the current total price of the order are displayed.

For delivery orders, additional logic is included to handle a delivery fee, which is eliminated if the total price exceeds $40. Additionally, it offers promotional messages that tell users how much more

they need to spend to get free delivery. Users have the option to add more pizzas to their order or complete the order. Upon completion, the final total is shown, and the user is asked if they want to start a new order. This loop continues until the user decides not to initiate a new order, effectively making it possible to handle multiple orders in one session.

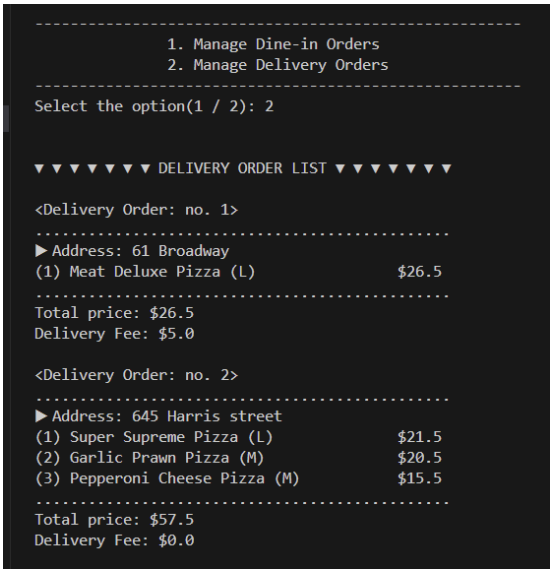


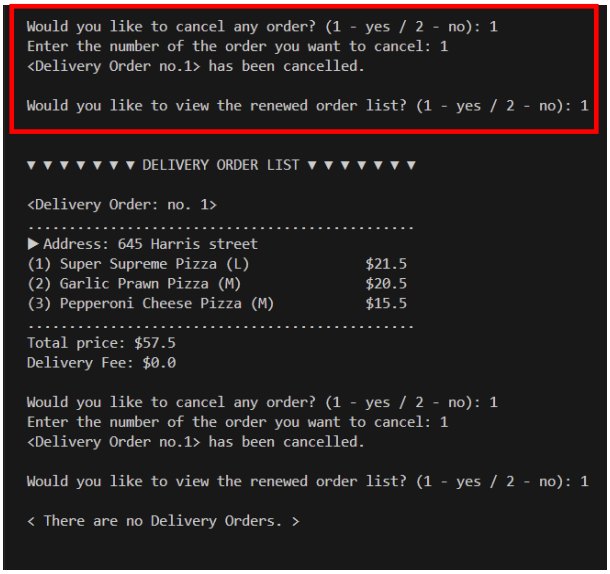


(3) manageDineInOrders() / manageDeliveryOrders()

The methods manageDineInOrders and manageDeliveryOrders are designed for managers to comprehensively check the order details and cancel orders if it is required.

The methods first checks if the ArrayList of respective orders is empty, and if so, it informs the user and exits the loop. If there are orders, it displays each order's details (table number, pizza name, size, and price) in sequence and prompts the user if they want to cancel any order. If the user chooses to cancel, they are asked to specify the order number to cancel. The method validates the input and removes the specified order from the list if valid, then allows the user to review the updated list or exit.





2) Order Class

(1) askSize()

The askSize() method is designed to prompt the user to enter a proper pizza size and return the char value, which will be used to set the pizza size.

When the user inputs a character, the method checks if it matches 'M' or 'm' for medium, or 'L' or 'l' for large. If a valid character is entered, the method returns the uppercase version of the size. This value will be an argument of setSize(char size) method to set the size and affect the price of the pizza.

(2) displayOrderDetails

The displayOrderDetails() method is designed to provide a comprehensive display of an order's details in a structured format, utilised in the order management process.

Initially, it calls displayPizzas(), which lists each pizza within the order, showing its sequence, name, size, and price. Following this, it calls the toString() method to append a summary line that includes the total price of the order.

This method is overridden in the DineInOrder and DeliveryOrder classes to include specific information: table numbers for dine-in and address with delivery fees for delivery orders.

3) Pizza Class

(1) generatePizza()

The generatePizza() method plays a critical role in creating new Pizza objects based on predefined templates and return a pizza object each time the user select pizza from the menu. This method ensures that every pizza order is treated as a unique instance, allowing for specific customisations such as size without altering the original template.

5. if Statements

In the PizzaShop.java program, if statements are extensively used to manage the flow of operations based on user inputs and system conditions. They perform essential roles such as validating inputs, directing the program's logic based on user choices, and enabling or disabling specific functionalities. This ensures that the program behaves correctly under various scenarios, maintains robustness, and provides a user-friendly experience by dynamically adapting to user interactions and system states.

6. Loops

1) Overall use of Loops

The program prominently uses while loops, which are used to continually prompt users until they provide valid inputs or choose to terminate certain processes. For instance, the main while(true) loop in proceedOrder() keeps the order process running until the user decides to exit. There are also loops for each input, such as choosing between dine-in and delivery, selecting pizzas, and deciding whether to add more items or complete the order. They check the user's input for validity, ensuring that only correct commands are processed, which helps maintain the integrity and smooth operation of the ordering system.

2) Loops iterating through a HashMap

In the showMenu() method of PizzaShop class, a for-each loop iterates through a HashMap called pizzaMenu, which stores integer keys and Pizza object values. The loop processes each HashMap.Entry<Integer, Pizza> to extract and display the menu number (key) and pizza details (value) including the name and prices for each size. The output is formatted into a structured menu layout. This approach ensures all pizzas are presented clearly and uniformly, making it easy for users to understand and make selections.

```
public void showMenu() {
    System.out.println(x:"\n====================PIZZA MENU====================");
    System.out.println(x:"             PIZZA                  M      L      ");
    for (HashMap.Entry<Integer, Pizza> entry : pizzaMenu.entrySet()) {
        Pizza pizza = entry.getValue();
        System.out.println("    " + entry.getKey() + ". " + pizza.getName() + " \t$ "
                           + pizza.getPrice(size:'M') + " / " + pizza.getPrice(size:'L'));
    }
    System.out.println(x:"==================================================");
}
```

3) Loops iterating through an ArrayList

The displayPizzas() method in Order class utilises a for-each loop to iterate through an ArrayList called pizzaOrder, which contains Pizza objects. Each iteration of the loop processes a Pizza object, extracting and displaying its name, size, and price. int pizzaNumber, which is a counter, is incremented with each iteration to sequentially number the pizzas as they are displayed. This method effectively showcases the details of each ordered pizza, facilitating a clear review of the order contents.

```
public void displayPizzas(){
    int pizzaNumber = 1;
    for (Pizza pizza : pizzaOrder) {
        System.out.print("(" + pizzaNumber + ") " + pizza.getName());
        System.out.println(" (" + pizza.getSize() + ") \t\t $" + pizza.getPrice(pizza.getSize()));
        pizzaNumber++;
    }
}
```

7. Inheritance Hierarchy

In the PizzaShop.java, the use of a parent(Order) class with child classes(DineInOrder and DeliveryOrder) is a strategic design choice that enhances code reusability, specialisation, and system scalability. The Order class encapsulates common features and functionalities which are required when processing both dine-in orders and delivery orders, reducing code duplication. Two child classes extends this common functionality with specific attributes and methods unique to different ordering scenarios. This structure allows for the easy addition of new order types as new subclasses(e.g. take away order), each tailored to particular needs without altering the existing system significantly. Moreover, it simplifies maintenance by localising common changes to the parent class and specific changes to respective subclasses, thus ensuring a clear and organised code.

8. Problem encountered and Solution

When I first design the code, 5 pizza objects corresponding to each menu item were instantiated within the constructor method and stored in a hashmap. During the order process, an object was retrieved from among five options in the hashmap by pizzaMenu.get(userInput) and added to the ArrayList of pizzas. While this approach initially appeared effective, persistent testing uncovered a critical issue. Specifically, when multiple orders require the same type of pizza with different sizes, the attributes of the already instantiated object had been altered due to previous orders, resulting in inaccurate pricing for subsequent orders. This issue not only affected the total price of each order, but also the pizza details in the order list when managing orders.

To correct this flaw, I designed a method named generatePizza() within the pizza class, which creates and returns a new Pizza object each time user selects a pizza. This approach thereby enhanced flexibility and efficiency in the pizza ordering process by minimising errors and maintaining consistency in the basic attributes of each pizza, such as name and price, before any customisations are applied.

```
public Pizza generatePizza() {
    Pizza newPizza = new Pizza(this.name, this.basePrice);
    return newPizza;
}
```

```
while (true) {
    System.out.print(s:"Enter the number of the pizza (1-5): ");
    int menuChoice = In.nextInt();

    if (pizzaMenu.containsKey(menuChoice)) {
        Pizza selectedPizza = pizzaMenu.get(menuChoice).generatePizza();
        if(orderType == 1){
            selectedPizza.setSize(dineInOrder.askSize());
            dineInOrder.addPizza(selectedPizza);
            System.out.println("\n" + selectedPizza.getName() + "(" + selectedPizza.getSize() + ") is added to your order");
            System.out.println("- Price: $ " + selectedPizza.getPrice(selectedPizza.getSize()));
            System.out.println("- Current Total: $" + dineInOrder.getTotalPrice());
```