### **1. Importing Required Packages**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.Scanner;**

* **import java.util.ArrayList;**: This imports the **ArrayList** class from the java.util package, which allows you to create dynamic arrays that can grow as needed.
* **import java.util.List;**: This imports the **List** interface, which represents an ordered collection (a sequence) of elements.
* **import java.util.Scanner;**: This imports the **Scanner** class, which is used to get user input.

### **2. Defining the Dish Class**

**class Dish {**

**String name;**

**String description;**

**double price;**

**// Constructor to initialize a Dish object**

**public Dish(String name, String description, double price) {**

**this.name = name;**

**this.description = description;**

**this.price = price;**

**}**

**}**

* **class Dish**: This defines a **Dish** class representing a menu item.
* **String name; String description; double price;**: These are the attributes of the **Dish** class. **name** is the dish's name, **description** is its description, and price is its price.
* **public Dish(String name, String description, double price)**: This is the constructor for the **Dish** class, which initializes the **name**, **description**, and price when a Dish object is created.

### **3. Defining the OrderItem Class**

**class OrderItem {**

**Dish dish;**

**int quantity;**

**// Constructor to initialize an OrderItem object**

**public OrderItem(Dish dish, int quantity) {**

**this.dish = dish;**

**this.quantity = quantity;**

**}**

**// Calculate the subtotal for this order item**

**public double getSubtotal() {**

**return dish.price \* quantity;**

**}**

**}**

* **class OrderItem**: This defines an **OrderItem** class representing an item that a customer orders.
* **Dish dish; int quantity;**: These are the attributes of the **OrderItem** class. **dish** is the **Dish** object that is ordered, and **quantity** is how many of that dish are ordered.
* **public OrderItem(Dish dish, int quantity)**: This is the constructor for the **OrderItem** class, which initializes the **dish** and **quantity** attributes.
* **public double getSubtotal()**: This method calculates the subtotal (cost) for this **OrderItem** by multiplying the dish's price by the quantity ordered.

### **4. Defining the RestaurantMenuSystem Class**

**public class RestaurantMenuSystem {**

**static Scanner scanner = new Scanner(System.in);**

**static List<Dish> appetisers = new ArrayList<>();**

**static List<Dish> mainDishes = new ArrayList<>();**

**static List<Dish> friedRice = new ArrayList<>();**

* **public class RestaurantMenuSystem**: This defines the main class for the restaurant menu system.
* **static Scanner scanner = new Scanner(System.in);**: This creates a **Scanner** object to get input from the user.
* **static List<Dish> appetisers = new ArrayList<>();**: This creates a list to hold **Dish** objects that are **appetisers**.
* **static List<Dish> mainDishes = new ArrayList<>();**: This creates a list to hold **Dish** objects that are **mainDishes**.
* **static List<Dish> friedRice = new ArrayList<>();**: This creates a list to hold **Dish** objects that are **friedRice** dishes.

### **5. The main Method**

**public static void main(String[] args) {**

**setupMenu(); // Initialize the menu with dishes**

**startOrdering(); // Handle the ordering process**

**}**

* **public static void main(String[] args)**: This is the entry point of the program. The code inside this method is executed when the program starts.
* **setupMenu();**: This calls the **setupMenu** method to initialize the menu with dishes.
* **startOrdering();**: This calls the **startOrdering** method to start the process of taking orders from the user.

### **6. Setting Up the Menu**

**private static void setupMenu() {**

**appetisers.add(new Dish("Spring Rolls", "Crispy rolls filled with vegetables and sweet chili sauce.", 6.00));**

**appetisers.add(new Dish("Thai Dumplings", "Steamed dumplings with chicken and vegetable filling.", 7.50));**

**mainDishes.add(new Dish("Pad Thai", "Stir-fried rice noodles with shrimp, tofu, peanuts, and bean sprouts.", 12.00));**

**mainDishes.add(new Dish("Basil Chicken", "Stir-fried chicken with Thai basil, chili, and vegetables, served with jasmine rice.", 12.00));**

**friedRice.add(new Dish("Thai Fried Rice", "Fried rice with chicken, shrimp, or vegetables, seasoned with Thai spices.", 11.00));**

**friedRice.add(new Dish("Pineapple Fried Rice", "Fried rice with pineapple, cashews, and choice of meat or tofu.", 12.50));**

**}**

* **private static void setupMenu()**: This method is used to set up the menu by adding various dishes to the **appetisers**, **mainDishes**, and **friedRice** lists.
* **appetisers.add(new Dish(...));**: This adds a new **Dish** to the **appetisers** list.
* **mainDishes.add(new Dish(...));**: This adds a new **Dish** to the **mainDishes** list.
* **friedRice.add(new Dish(...));**: This adds a new Dish to the **friedRice** list.

### **7. Starting the Ordering Process**

**private static void startOrdering() {**

**List<OrderItem> order = new ArrayList<>(); // List to hold the ordered items**

* **private static void startOrdering()**: This method handles the entire ordering process.
* **List<OrderItem> order = new ArrayList<>();**: This creates a list to hold the items that the customer orders.

### **8. The Main Ordering Loop**

**while (true) {**

**System.out.println("\nPlease choose an option:");**

**System.out.println("1. Show Menu");**

**System.out.println("2. Exit");**

**System.out.println();**

**System.out.println("Note: 1 For Menu and 2 For Exiting the menu");**

**String choice = scanner.nextLine();**

* **while (true)**: This creates an infinite loop that will keep running until the user chooses to exit.
* **System.out.println(...);**: These lines display options for the user to choose from (show the menu or exit).
* **String choice = scanner.nextLine();**: This reads the user's choice from the input.

### **9. Handling User Choices**

**if (choice.equals("2")) {**

**System.out.println("\nThank you for visiting!");**

**break;**

**} else if (choice.equals("1")) {**

**showCategories(); // Display categories of dishes**

**int categoryChoice = Integer.parseInt(scanner.nextLine());**

**List<Dish> selectedCategory = getCategoryByChoice(categoryChoice);**

**if (selectedCategory != null) {**

**showDishes(selectedCategory); // Display dishes in the selected category**

**System.out.println("\nEnter the number of the dish you want to order:");**

**int dishChoice = Integer.parseInt(scanner.nextLine());**

**if (dishChoice > 0 && dishChoice <= selectedCategory.size()) {**

**Dish dish = selectedCategory.get(dishChoice - 1);**

**int quantity;**

**while (true) {**

**System.out.println("How many would you like to order?");**

**quantity = Integer.parseInt(scanner.nextLine());**

**if (quantity > 0) {**

**break;**

**}**

**System.out.println("Quantity must be a positive number. Please try again.");**

**}**

**order.add(new OrderItem(dish, quantity)); // Add the ordered item to the list**

**} else {**

**System.out.println("\nInvalid dish selection. Moving to the next step.");**

**}**

**}**

**System.out.println("\nWould you like to continue ordering or finish?");**

**System.out.println("1. Continue Ordering");**

**System.out.println("2. Finish and Print Invoice");**

**String continueChoice = scanner.nextLine();**

**if (continueChoice.equals("2")) {**

**printInvoice(order); // Print the invoice and exit**

**break;**

**}**

**} else {**

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

**}**

**}**

**}**

* **if (choice.equals("2"))**: If the user chooses "2", the program exits the loop, thanking the user.
* **else if (choice.equals("1"))**: If the user chooses "1", the program shows the categories of dishes and allows the user to select a category and a dish.
* **showCategories();**: This displays the available categories.
* **int categoryChoice = Integer.parseInt(scanner.nextLine());**: This reads the user's category choice.
* **List<Dish> selectedCategory = getCategoryByChoice(categoryChoice);**: This retrieves the list of dishes in the selected category.
* **if (selectedCategory != null)**: This checks if a valid category was selected.
* **showDishes(selectedCategory);**: This displays the dishes in the selected category.
* **int dishChoice = Integer.parseInt(scanner.nextLine());**: This reads the user's choice of dish.
* **if (dishChoice > 0 && dishChoice <= selectedCategory.size())**: This checks if the dish choice is valid.
* **Dish dish = selectedCategory.get(dishChoice - 1);**: This gets the selected dish from the list.
* **int quantity;**: This variable will hold the quantity of the dish the user wants to order.
* **while (true)**: This loop ensures that the user enters a positive quantity.
* **order.add(new OrderItem(dish, quantity));**: This adds the ordered item to the list of orders.
* **else { ... }**: This handles invalid dish selections.
* **if (continueChoice.equals("2"))**: If the user chooses to finish, the program prints the invoice and exits.
* **else { ... }**: This handles invalid menu choices.

### **10. Displaying Categories**

**private static void showCategories() {**

**System.out.println("\nCategories:");**

**System.out.println("1. Appetisers");**

**System.out.println("2. Main Dishes");**

**System.out.println("3. Fried Rice");**

**System.out.println();**

**System.out.println("Enter the categories number:");**

**}**

* **private static void showCategories()**: This method displays the categories of dishes that the user can choose from.

### **11. Getting the Selected Category**

**private static List<Dish> getCategoryByChoice(int choice) {**

**switch (choice) {**

**case 1:**

**return appetisers; // Return appetisers if choice is**

**case 2:**

**return mainDishes; // Return main dishes if choice is**

**case 3:**

**return friedRice; // Return fried rice if choice is**

**default:**

**System.out.println("\nInvalid category selection.");**

**return null;**

**}**

**}**

* **private static List<Dish> getCategoryByChoice(int choice)**: This method returns the list of dishes for the selected category.
* **switch (choice)**: This is a **switch** statement that checks the user's category choice.
* **case 1:**: If the user chose "1", the method returns the **appetisers** list.
* **case 2:**: If the user chose "2", the method returns the **mainDishes** list.
* **case 3:**: If the user chose "3", the method returns the **friedRice** list.
* **default:**: If the user chose an invalid category, the method prints an error message and returns **null**.

### **12. Displaying Dishes**

**private static void showDishes(List<Dish> category) {**

**System.out.println("\nDishes:");**

**for (int i = 0; i < category.size(); i++) {**

**Dish dish = category.get(i);**

**System.out.printf("%d. %s: %s ($%.2f)%n", i + 1, dish.name, dish.description, dish.price);**

**}**

**System.out.println();**

**}**

* **private static void showDishes(List<Dish> category)**: This method displays the dishes in the selected category.
* **for (int i = 0; i < category.size(); i++)**: This loop goes through each dish in the category.
* **Dish dish = category.get(i);**: This gets the dish at index i.
* **System.out.printf(...)**: This prints the dish's number, name, description, and price.

### **13. Printing the Invoice**

**private static void printInvoice(List<OrderItem> order) {**

**System.out.println("\nInvoice:");**

**System.out.printf("%-20s%-10s%-10s%n", "Dish", "Quantity", "Subtotal");**

**double total = 0;**

**// Print each order item and its subtotal**

**for (OrderItem item : order) {**

**double subtotal = item.getSubtotal();**

**total += subtotal;**

**System.out.printf("%-20s%-10d$%-10.2f%n", item.dish.name, item.quantity, subtotal);**

**}**

**// Print the total amount**

**System.out.printf("\n%-20s%-10s$%-10.2f%n", "Total", "", total);**

**System.out.println();**

**}**

**}**

* **private static void printInvoice(List<OrderItem> order)**: This method prints the invoice with the ordered items and their subtotals.
* **System.out.println("\nInvoice:");**: This prints the header for the invoice.
* **System.out.printf("%-20s%-10s%-10s%n", "Dish", "Quantity", "Subtotal");**: This prints the column headers: "**Dish**", "**Quantity**", and "**Subtotal**".
* **double total = 0;**: This initializes the total amount to 0.
* **for (OrderItem item : order)**: This loop goes through each **OrderItem** in the order list.
* **double subtotal = item.getSubtotal();**: This calculates the subtotal for the OrderItem.
* **total += subtotal;**: This adds the subtotal to the total amount.
* **System.out.printf(...);**: This prints the dish name, quantity, and subtotal for each ordered item.
* **System.out.printf("\n%-20s%-10s$%-10.2f%n", "Total", "", total);**: This prints the total amount.
* **System.out.println();**: This prints an empty line after the invoice.

### **...args (Varargs) in Java**

**In Java, ...args is used to denote a variable-length argument list, also known as varargs. It allows you to pass an arbitrary number of arguments to a method. For example:**

**java**

**Copy code**

**public void exampleMethod(String... args) {**

**for (String arg : args) {**

**System.out.println(arg);**

**}**

**}**

**In this method, you can pass zero or more String arguments to exampleMethod. The args parameter is treated as an array within the method.**

### **printf Method**

**The printf method in Java is used to print formatted strings. The System.out.printf method takes a format string and a list of arguments. Here's your specific code:**

**java**

**Copy code**

**System.out.printf("\n%-20s%-10s$%-10.2f%n", "Total", "", total);**

* **\n - Newline character.**
* **%-20s - Formats the first argument ("Total") as a left-justified string in a 20-character wide field.**
* **%-10s - Formats the second argument ("", an empty string) as a left-justified string in a 10-character wide field.**
* **$%-10.2f - Formats the third argument (total) as a floating-point number with 2 decimal places, left-justified in a 10-character wide field, prefixed with $.**
* **%n - Platform-independent newline character.**

### **How args Relates to printf**

**The printf method effectively uses varargs to accept a variable number of arguments:**

**java**

**Copy code**

**public PrintStream printf(String format, Object... args)**

**This allows you to pass any number of arguments after the format string, which the method then formats according to the format string provided.**

**So, in your case:**

**java**

**Copy code**

**System.out.printf("\n%-20s%-10s$%-10.2f%n", "Total", "", total);**

**Here, "Total", "", and total are the arguments passed to printf, and the format string "\n%-20s%-10s$%-10.2f%n" dictates how they will be displayed.**