## Online Food Ordering System Project Documentation

#### **Overview**

The **Online Food Ordering System** is a Java-based application that allows users to browse restaurants, filter menus, place orders, and track deliveries. The project demonstrates the use of **Object-Oriented Programming (OOP)** concepts such as **inheritance**, **polymorphism**, **encapsulation**, **abstraction**, and **composition**.

#### **Features**

#### 1. User Management:

- Users can register with their name, email, phone, and address
- Users can view their order history and track orders

#### 2. Restaurant Management:

- Restaurants are categorized by cuisine and have menus with multiple items
- Users can filter restaurants by cuisine or price range

#### 3. Order Placement:

- Users can select items from a restaurant's menu and place an order
- Orders are saved to the user's order history

#### 4. Payment Processing:

• Supports multiple payment methods (Credit Card, Digital Wallet)

• Includes promo codes and reward points for discounts

#### 5. **Delivery Tracking**:

• Tracks the status of an order from placement to delivery

#### 6. File Management:

• Order history is saved and loaded from files based on the user's email

## **OOP Concepts Used**

## 1. Encapsulation

• **Definition**: Wrapping data (fields) and methods into a single unit (class) and restricting direct access to some components.

#### • Example:

- The User class encapsulates user details like name, email, and phone with private fields and public getter methods
- The Order class encapsulates order details such as orderId, orderedItems, and totalPrice

## 2. Inheritance

• **Definition**: A mechanism where one class acquires the properties and behaviors of another class.

#### • Example:

- CreditCardPayment and DigitalWalletPayment inherit from the abstract Payment class
- RestaurantSearchService implements multiple interfaces (CuisineFilter, PriceFilter)

## 3. Polymorphism

• **Definition**: The ability of a single interface to represent different types or behaviors.

#### • Example:

- The PaymentProcessor interface is implemented by both CreditCardPayment and DigitalWalletPayment, allowing different payment methods to be processed polymorphically
- Overloaded constructors in the User and Restaurant classes

#### 4. Abstraction

• **Definition**: Hiding implementation details and showing only the essential features of an object.

#### • Example:

- The PaymentProcessor interface defines methods like process() and refund() without specifying their implementation
- The DataManager interface abstracts file operations for saving and loading data

## 5. Composition

• **Definition**: A "has-a" relationship where objects are composed of other objects.

#### • Example:

- The User class has an Address object and a Rewards object
- The Restaurant class has a list of Menu objects, and each Menu has a list of MenuItem objects

## 6. Multithreading

• **Definition**: Running multiple threads concurrently to perform tasks.

#### • Example:

 The DeliveryTracker class uses a separate thread to update and display the delivery status of orders

## **Class Descriptions**

#### 1. User

- Represents a user in the system
- Fields: name, email, phone, address, orderHistory, rewards
- Methods:
  - addOrder(Order order): Adds an order to the user's history and updates reward points
  - trackOrder(int orderId): Tracks the status of a specific order
  - applyPromoCode(String promoCode, Order order): Applies a promo code to an order

### 2. Restaurant

- Represents a restaurant with menus and details
- Fields: name, location, cuisine, menus, isOpen, rating, averagePrice
- Methods:
  - addMenu(Menu menu): Adds a menu to the restaurant
  - getMenus(): Retrieves the list of menus

## 3. Order

- Represents an order placed by a user
- Fields: orderId, orderedItems, totalPrice, paymentMethod, isPaid, orderTime
- Methods:

- placeOrder(): Processes the payment and confirms the order
- assignPaymentMethod(PaymentProcessor paymentMethod):
  Assigns a payment method to the order

## 4. Payment (Abstract Class)

- Abstract class for payment processing
- Fields: amount, isProcessed
- Methods:
  - process(): Abstract method for processing payments
  - getReceipt(): Returns a receipt for the payment

## 5. CreditCardPayment (Extends Payment, Implements PaymentProcessor)

- Processes payments using credit cards
- Fields: cardNumber, cardHolderName, cvv
- Methods:
  - process(double amount): Processes the payment
  - refund(double amount): Refunds the payment

# 6. DigitalWalletPayment (Extends Payment, Implements PaymentProcessor)

- Processes payments using digital wallets
- Fields: walletId, provider
- Methods:
  - process(double amount): Processes the payment
  - refund(double amount): Refunds the payment

## 7. DeliveryTracker

- Tracks the delivery status of orders using multithreading
- Fields: orderStatus, isTracking
- Methods:
  - updateStatus(String orderId, String status): Updates the status of an order
  - startTracking(): Starts the tracking thread
  - stopTracking(): Stops the tracking thread

## 8. FileOrderHistoryManager

- Manages saving and loading order history to/from files
- Methods:
  - loadOrderHistory(String userEmail): Loads a user's order history from a file
  - saveOrderHistory(User user): Saves a user's order history to a file

## 9. RestaurantSearchService

- Filters restaurants based on cuisine and price range
- Implements: CuisineFilter, PriceFilter

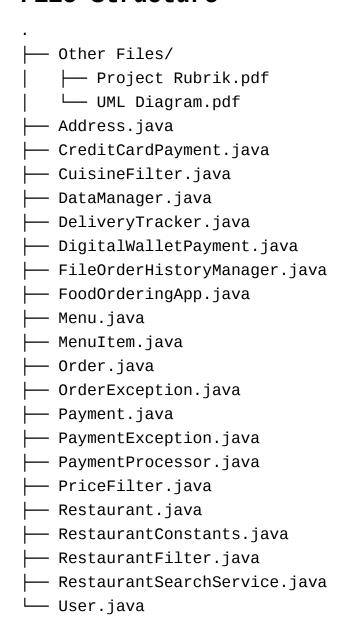
#### 10. Menu

- Represents a menu with multiple items
- Fields: items
- Methods:
  - addItems(MenuItem... items): Adds items to the menu
  - getItems(): Retrieves the list of menu items

#### 11. MenuItem

- Represents a single menu item
- Fields: itemId, name, price, category
- Methods:
  - getDetails(): Returns the details of the menu item

## File Structure



## How to Run

1. Compile the Project:

javac -d bin FoodOrderingApp.java

2. Run the Application:

java -cp bin FoodOrderingApp