CS 5800.01 - Advanced Software Engineering

Final - Project

Team Members:

1 - Chaitanya Nalage
2 - Hady Ziadeh
3 - Payton Perchez
4 - Shreyas Chaudhary

Github link:

https://github.com/chaitanyanalage/CS5800

Answers

To solve part a, we selected the following **5** design patterns:

1. **Singleton:** It ensures centralised management and control over the entire system. It is to ensure only one `CPPFoodDelivery` class that manages the entire food delivery platform. This is crucial for maintaining a single coordination point for registering users (customers, restaurants, and drivers) and handling orders.
2. **Factory:** To create different dietary restriction strategies. This pattern provides a way to encapsulate the instantiation logic for various dietary restrictions, making it easy to add new restrictions without modifying existing code. (`DietaryRestrictionFactory`)
3. **Observer:** To notify customers about the status updates of their orders. This pattern is useful for implementing a mechanism where the system can inform customers about changes to their order status (for example: when the order is picked up or delivered) in real-time. (`Order`, `OrderObserver`, `CustomerOrderObserver`)
4. **Decorator:** To add additional features (toppings) to food items dynamically. This pattern allows for flexible and reusable combinations of food items with various toppings without altering the underlying food item class. (`FoodItem`, `BasicFoodItem`, `FoodItemDecorator`, `ToppingDecorator`)
5. **Strategy:** To encapsulate various dietary restriction algorithms and apply them to customers. This pattern allows for the selection and execution of dietary restrictions at runtime, making it easy to support multiple types of dietary needs without hardcoding the logic. (DietaryRestrictionStrategy`, `NoRestriction`, `Paleo`, `Vegan`, `NutAllergy`)

Output:

```java
import java.util.*;

public class BasicFoodItem implements FoodItem {
    private String description;
    private double cost;

    public BasicFoodItem(String description, double cost) {
        this.description = description;
        this.cost = cost;
    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public double getCost() {
        return cost;
    }
}
import java.util.*;

public class CPPFoodDelivery {
    private static CPPFoodDelivery instance;
    private List<Customer> customers = new ArrayList<>();
    private List<Restaurant> restaurants = new ArrayList<>();
    private List<Driver> drivers = new ArrayList<>();

    private CPPFoodDelivery() {}

    public static CPPFoodDelivery getInstance() {
        if (instance == null) {
            instance = new CPPFoodDelivery();
        }
        return instance;
    }

    public void registerCustomer(Customer customer) {
```

```
        customers.add(customer);
        System.out.println(customer.getName() + " has been registered with CPPFoodDelivery.");
    }

    public void registerRestaurant(Restaurant restaurant) {
        restaurants.add(restaurant);
        System.out.println(restaurant.getName() + " has been registered with CPPFoodDelivery.");
    }

    public void registerDriver(Driver driver) {
        drivers.add(driver);
        System.out.println(driver.getName() + " has been registered with CPPFoodDelivery.");
    }

    public List<Driver> getAvailableDrivers(String county, Date time) {
        List<Driver> availableDrivers = new ArrayList<>();
        for (Driver driver : drivers) {
            if (driver.getCounty().equals(county) && driver.isAvailableDuring(time)) {
                availableDrivers.add(driver);
            }
        }
        return availableDrivers;
    }

    // Other mediator methods to handle interactions can be added here
}
import java.util.*;

// Customer Class with Dietary Restrictions
public class Customer extends User {
    private String address;
    private String county;
    private DietaryRestrictionStrategy dietaryRestrictionStrategy;

    public Customer(String name, String address, String county) {
        super(name);
```

```
        this.address = address;
        this.county = county;
    }

    public String getAddress() {
        return address;
    }

    public String getCounty() {
        return county;
    }

    public void setDietaryRestrictionStrategy(DietaryRestrictionStrategy
strategy) {
        this.dietaryRestrictionStrategy = strategy;
    }

    public DietaryRestrictionStrategy getDietaryRestrictionStrategy() {
        return dietaryRestrictionStrategy;
    }

    public void applyDietaryRestriction(List<String> carbs, List<String>
proteins, List<String> fats) {
        if (dietaryRestrictionStrategy != null) {
            dietaryRestrictionStrategy.applyRestriction(this, carbs,
proteins, fats);
        }
    }
}
import java.util.*;

// Class to represent the Customer's Order
class CustomerOrder {
    private List<FoodItem> foodItems = new ArrayList<>();
    private Customer customer;

    public CustomerOrder(Customer customer) {
        this.customer = customer;
    }
```

```
    public void addFoodItem(FoodItem foodItem) {
        foodItems.add(foodItem);
    }

    public double calculateTotalCost() {
        double total = 0;
        for (FoodItem foodItem : foodItems) {
            total += foodItem.getCost();
        }
        return total;
    }

    public List<FoodItem> getFoodItems() {
        return foodItems;
    }
}
import java.util.*;

public class CustomerOrderObserver implements OrderObserver {
    private String name;

    public CustomerOrderObserver(String name) {
        this.name = name;
    }

    @Override
    public void update(Order order) {
        System.out.println("Customer " + name + " received an update about
their order: " + order.getRestaurant().getName() +
                " has prepared " + ". It's on the way with " +
order.getDriver().getName() +
                ". Estimated delivery time: " +
order.getOrderDeliveredTime());
    }
}
public class DietaryRestrictionFactory {
    public static DietaryRestrictionStrategy createDietaryRestriction(String
restrictionType) {
        switch (restrictionType) {
```

```
            case "NoRestriction":
                return new NoRestriction();
            case "NutAllergy":
                return new NutAllergy();
            case "Paleo":
                return new Paleo();
            case "Vegan":
                return new Vegan();
            default:
                throw new IllegalArgumentException("Unknown restriction
type");
        }
    }
}
import java.util.List;

public interface DietaryRestrictionStrategy {
    void applyRestriction(Customer customer, List<String> carbs,
List<String> proteins, List<String> fats);
}

import java.util.*;

public class Driver extends User {
    private String address;
    private String county;
    private String shift;
    private int shiftStartHour;
    private int shiftEndHour;

    public Driver(String name, String address, String county, String shift,
int shiftStartHour, int shiftEndHour) {
        super(name);
        this.address = address;
        this.county = county;
        this.shift = shift;
        this.shiftStartHour = shiftStartHour;
        this.shiftEndHour = shiftEndHour;
    }
```

```
    public String getAddress() {
        return address;
    }

    public String getCounty() {
        return county;
    }

    public String getShift() {
        return shift;
    }

    public int getShiftStartHour() {
        return shiftStartHour;
    }

    public int getShiftEndHour() {
        return shiftEndHour;
    }

    public boolean isAvailableDuring(Date time) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(time);
        int hour = cal.get(Calendar.HOUR_OF_DAY);
        return hour >= shiftStartHour && hour < shiftEndHour;
    }
}
public interface FoodItem {
    String getDescription();
    double getCost();
}

import java.util.*;

public abstract class FoodItemDecorator implements FoodItem {
    protected FoodItem foodItem;

    public FoodItemDecorator(FoodItem foodItem) {
```

```
        this.foodItem = foodItem;
    }

    @Override
    public String getDescription() {
        return foodItem.getDescription();
    }

    @Override
    public double getCost() {
        return foodItem.getCost();
    }
}
import java.util.*;

public class NoRestriction implements DietaryRestrictionStrategy {
    @Override
    public void applyRestriction(Customer customer, List<String> carbs,
List<String> proteins, List<String> fats) {
        System.out.println(customer.getName() + "'s diet plan is No
Restriction. All food items are allowed.");
    }
}
import java.util.*;

public class NutAllergy implements DietaryRestrictionStrategy {
    @Override
    public void applyRestriction(Customer customer, List<String> carbs,
List<String> proteins, List<String> fats) {
        System.out.println(customer.getName() + "'s diet plan is Nut
Allergy. No Nuts.");
        carbs.remove("Pistachio");
        fats.remove("Peanuts");
    }
}
import java.util.*;

public class Order {
    private Restaurant restaurant;
    private Customer customer;
```

```
    private String dietaryRestriction;
    private List<FoodItem> foodItems;
    private Driver driver;
    private Date orderCreationTime;
    private Date orderPickUpTime;
    private Date orderDeliveredTime;
    private List<OrderObserver> observers = new ArrayList<>();

    public Order(Restaurant restaurant, Customer customer, String
dietaryRestriction, List<FoodItem> foodItems, Driver driver, Date
orderCreationTime) {
        this.restaurant = restaurant;
        this.customer = customer;
        this.dietaryRestriction = dietaryRestriction;
        this.foodItems = foodItems;
        this.driver = driver;
        this.orderCreationTime = orderCreationTime;
    }

    public void addObserver(OrderObserver observer) {
        observers.add(observer);
    }

    public void notifyObservers() {
        for (OrderObserver observer : observers) {
            observer.update(this);
        }
    }

    public void setOrderPickUpTime(Date orderPickUpTime) {
        this.orderPickUpTime = orderPickUpTime;
    }

    public void setOrderDeliveredTime(Date orderDeliveredTime) {
        this.orderDeliveredTime = orderDeliveredTime;
    }

    public Restaurant getRestaurant() {
        return restaurant;
```

```
    }

    public Customer getCustomer() {
        return customer;
    }

    public String getDietaryRestriction() {
        return dietaryRestriction;
    }

    public List<FoodItem> getFoodItems() {
        return foodItems;
    }

    public Driver getDriver() {
        return driver;
    }

    public Date getOrderCreationTime() {
        return orderCreationTime;
    }

    public Date getOrderPickUpTime() {
        return orderPickUpTime;
    }

    public Date getOrderDeliveredTime() {
        return orderDeliveredTime;
    }
}
import java.util.*;

public interface OrderObserver {
    void update(Order order);
}

import java.util.*;

public class Paleo implements DietaryRestrictionStrategy {
```

```
    @Override
    public void applyRestriction(Customer customer, List<String> carbs,
List<String> proteins, List<String> fats) {
        System.out.println(customer.getName() + "'s diet plan is Paleo. No
Carbs except pistachio, No Tofu, No Dairy.");
        carbs.removeIf(item -> !item.equals("Pistachio"));
        proteins.remove("Tofu");
        fats.removeAll(Arrays.asList("Cheese", "Sour cream"));
    }
}
import java.util.*;

public class Restaurant extends User {
    private String address;
    private String county;
    private String operatingHours;
    private String cuisineType;
    private Map<String, Double> menu;
    private Map<String, Double> toppings;
    private int openingHour;
    private int closingHour;

    public Restaurant(String name, String address, String county, String
operatingHours, String cuisineType, Map<String, Double> menu, Map<String,
Double> toppings, int openingHour, int closingHour) {
        super(name);
        this.address = address;
        this.county = county;
        this.operatingHours = operatingHours;
        this.cuisineType = cuisineType;
        this.menu = menu;
        this.toppings = toppings;
        this.openingHour = openingHour;
        this.closingHour = closingHour;
    }

    public String getAddress() {
        return address;
    }
```

```
    public String getCounty() {
        return county;
    }

    public String getOperatingHours() {
        return operatingHours;
    }

    public String getCuisineType() {
        return cuisineType;
    }

    public Map<String, Double> getMenu() {
        return menu;
    }

    public Map<String, Double> getToppings() {
        return toppings;
    }

    public boolean isOpenDuring(Date time) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(time);
        int hour = cal.get(Calendar.HOUR_OF_DAY);
        return hour >= openingHour && hour < closingHour;
    }

    public String getMenuWithPrices() {
        StringBuilder menuWithPrices = new StringBuilder();
        for (Map.Entry<String, Double> entry : menu.entrySet()) {
            menuWithPrices.append(entry.getKey()).append("
($").append(entry.getValue()).append("), ");
        }
        return menuWithPrices.toString();
    }

    public String getToppingsWithPrices() {
        StringBuilder toppingsWithPrices = new StringBuilder();
        for (Map.Entry<String, Double> entry : toppings.entrySet()) {
```

```
            toppingsWithPrices.append(entry.getKey()).append("
($").append(entry.getValue()).append("), ");
        }
        return toppingsWithPrices.toString();
    }
}
import java.util.*;

public class ToppingDecorator extends FoodItemDecorator {
    private String topping;
    private double toppingCost;

    public ToppingDecorator(FoodItem foodItem, String topping, double
toppingCost) {
        super(foodItem);
        this.topping = topping;
        this.toppingCost = toppingCost;
    }

    @Override
    public String getDescription() {
        return foodItem.getDescription() + ", " + topping;
    }

    @Override
    public double getCost() {
        return foodItem.getCost() + toppingCost;
    }
}
import java.util.*;

public abstract class User {
    protected String name;

    public User(String name) {
        this.name = name;
    }

    public String getName() {
```

```
        return name;
    }
}

import java.util.*;

// Singleton Pattern for managing user registration
class UserRegistry {
    private static UserRegistry instance;
    private List<User> users;

    private UserRegistry() {
        users = new ArrayList<>();
    }

    public static UserRegistry getInstance() {
        if (instance == null) {
            instance = new UserRegistry();
        }
        return instance;
    }

    public void registerUser(User user) {
        users.add(user);
        if (user instanceof Restaurant) {
            Restaurant restaurant = (Restaurant) user;
            System.out.println(restaurant.getName() + " has been registered
with CPPFoodDelivery.");
            System.out.println("Address: " + restaurant.getAddress());
            System.out.println("County: " + restaurant.getCounty());
            System.out.println("Operating Hours: " +
restaurant.getOperatingHours());
            System.out.println("Cuisine Type: " +
restaurant.getCuisineType());
            System.out.println("Menu: " + restaurant.getMenuWithPrices());
            System.out.println("Optional Meal Toppings: " +
restaurant.getToppingsWithPrices());
        } else if (user instanceof Driver) {
            Driver driver = (Driver) user;
```

```
            System.out.println(driver.getName() + " has been registered with
CPPFoodDelivery.");
            System.out.println("Address: " + driver.getAddress());
            System.out.println("Shift: " + driver.getShift() + " (" +
driver.getShiftStartHour() + ":00 - " + driver.getShiftEndHour() + ":00)");
            System.out.println("Operating County: " + driver.getCounty());
        } else if (user instanceof Customer) {
            Customer customer = (Customer) user;
            System.out.println(customer.getName() + " has been registered
with CPPFoodDelivery.");
            System.out.println("Address: " + customer.getAddress());
            System.out.println("County: " + customer.getCounty());
        } else {
            System.out.println(user.getName() + " has been registered with
CPPFoodDelivery.");
        }
    }
}
import java.util.*;

public class Vegan implements DietaryRestrictionStrategy {
    @Override
    public void applyRestriction(Customer customer, List<String> carbs,
List<String> proteins, List<String> fats) {
        System.out.println(customer.getName() + "'s diet plan is Vegan. No
Meat and No Dairy.");
        proteins.removeAll(Arrays.asList("Fish", "Chicken", "Beef"));
        fats.removeAll(Arrays.asList("Cheese", "Sour cream", "Tuna"));
    }
}
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // Singleton instance of CPPFoodDelivery
        CPPFoodDelivery cppFoodDelivery = CPPFoodDelivery.getInstance();

        // Registering restaurants with specific names
        List<Restaurant> restaurants = new ArrayList<>();
        Map<String, Double> menu1 = new HashMap<>();
```

```
        menu1.put("Cheese", 3.0);
        menu1.put("Chicken", 5.0);
        menu1.put("Avocado", 2.5);
        Map<String, Double> toppings1 = new HashMap<>();
        toppings1.put("Salsa", 1.0);
        toppings1.put("Cheese", 1.5);
        toppings1.put("Guacamole", 2.0);
        restaurants.add(new Restaurant("Mexican Fiesta", "123 Main St", "LA
County", "8AM - 4PM", "Mexican", menu1, toppings1, 8, 16));

        Map<String, Double> menu2 = new HashMap<>();
        menu2.put("Bread", 2.0);
        menu2.put("Beef", 6.0);
        menu2.put("Sour cream", 1.5);
        Map<String, Double> toppings2 = new HashMap<>();
        toppings2.put("Olives", 1.0);
        toppings2.put("Parmesan", 1.5);
        toppings2.put("Basil", 1.0);
        restaurants.add(new Restaurant("Italian Delight", "456 Oak St",
"Orange County", "4PM - 12AM", "Italian", menu2, toppings2, 16, 24));

        Map<String, Double> menu3 = new HashMap<>();
        menu3.put("Lentils", 3.5);
        menu3.put("Fish", 5.5);
        menu3.put("Tuna", 4.0);
        Map<String, Double> toppings3 = new HashMap<>();
        toppings3.put("Soy Sauce", 0.5);
        toppings3.put("Ginger", 0.5);
        toppings3.put("Scallions", 0.5);
        restaurants.add(new Restaurant("Chinese Garden", "789 Pine St", "San
Bernardino County", "12AM - 8AM", "Chinese", menu3, toppings3, 0, 8));

        Map<String, Double> menu4 = new HashMap<>();
        menu4.put("Pistachio", 4.0);
        menu4.put("Tofu", 3.0);
        menu4.put("Peanuts", 2.0);
        Map<String, Double> toppings4 = new HashMap<>();
        toppings4.put("Cilantro", 0.5);
        toppings4.put("Yogurt", 0.5);
```

```
        toppings4.put("Chutney", 1.0);
        restaurants.add(new Restaurant("Indian Spice", "101 Maple St", "LA County", "8AM - 4PM", "Indian", menu4, toppings4, 8, 16));

        for (Restaurant restaurant : restaurants) {
            cppFoodDelivery.registerRestaurant(restaurant);
        }

        // Registering drivers with specific names
        List<Driver> drivers = new ArrayList<>();
        drivers.add(new Driver("Walter White", "Driver Address 1", "LA County", "1st shift", 8, 16));
        drivers.add(new Driver("Jesse Pinkman", "Driver Address 2", "Orange County", "2nd shift", 16, 24));
        drivers.add(new Driver("Hank", "Driver Address 3", "San Bernardino County", "3rd shift", 0, 8));
        drivers.add(new Driver("Gus Fring", "Driver Address 4", "LA County", "1st shift", 8, 16));
        drivers.add(new Driver("Tuco", "Driver Address 5", "Orange County", "2nd shift", 16, 24));
        drivers.add(new Driver("Saul Goodman", "Driver Address 6", "San Bernardino County", "3rd shift", 0, 8));
        drivers.add(new Driver("Mike Ehrmantraut", "Driver Address 7", "LA County", "1st shift", 8, 16));
        drivers.add(new Driver("Skinny Pete", "Driver Address 8", "Orange County", "2nd shift", 16, 24));

        for (Driver driver : drivers) {
            cppFoodDelivery.registerDriver(driver);
        }

        // Registering customers with specific names
        List<Customer> customers = new ArrayList<>();
        for (int i = 1; i <= 10; i++) {
            Customer customer = new Customer("Customer " + i, "Customer Address " + i, "County " + ((i % 3) + 1));
            // Randomly assign a dietary restriction
            switch (i % 4) {
                case 0:
customer.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("NoRestriction"));
```

```
                    break;
                case 1:
customer.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("Paleo"));
                    break;
                case 2:
customer.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("Vegan"));
                    break;
                case 3:
customer.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("NutAllergy"));
                    break;
            }
            cppFoodDelivery.registerCustomer(customer);
            customers.add(customer);
        }

        // Macronutrient Food Options
        List<String> carbs = new ArrayList<>(Arrays.asList("Cheese", "Bread", "Lentils", "Pistachio"));
        List<String> proteins = new ArrayList<>(Arrays.asList("Fish", "Chicken", "Beef", "Tofu"));
        List<String> fats = new ArrayList<>(Arrays.asList("Avocado", "Sour cream", "Tuna", "Peanuts"));

        // Simulating customer ordering
        Random random = new Random();
        for (int i = 0; i < customers.size(); i++) {
            Customer customer = customers.get(i);
            Restaurant restaurant;
            Date orderCreationTime;

            if (i < 3) {
                restaurant = restaurants.stream().filter(r -> r.getName().equals("Indian Spice")).findFirst().get();
                orderCreationTime = new GregorianCalendar(2024, Calendar.MAY, 1, 9, 0).getTime(); // 9AM
            } else if (i < 6) {
```

```
                restaurant = restaurants.stream().filter(r ->
r.getName().equals("Chinese Garden")).findFirst().get();
                orderCreationTime = new GregorianCalendar(2024,
Calendar.MAY, 1, 1, 0).getTime(); // 1AM
            } else if (i < 8) {
                restaurant = restaurants.stream().filter(r ->
r.getName().equals("Mexican Fiesta")).findFirst().get();
                orderCreationTime = new GregorianCalendar(2024,
Calendar.MAY, 1, 10, 0).getTime(); // 10AM
            } else {
                restaurant = restaurants.stream().filter(r ->
r.getName().equals("Chinese Garden")).findFirst().get();
                orderCreationTime = new GregorianCalendar(2024,
Calendar.MAY, 1, 9, 0).getTime(); // 9AM for closed restaurant scenario
            }


System.out.println("=============================================================
=");
            System.out.println(customer.getName() + " is attempting to place
an order at " + restaurant.getName());


            // Check if restaurant is open
            if (!restaurant.isOpenDuring(orderCreationTime)) {
                System.out.println("Restaurant " + restaurant.getName() + "
is closed. Cannot place an order.");

System.out.println("=============================================================
=");
                continue;
            } else {
                System.out.println("Restaurant " + restaurant.getName() + "
is open.");
            }


            // Apply dietary restrictions
            List<String> availableCarbs = new ArrayList<>(carbs);
            List<String> availableProteins = new ArrayList<>(proteins);
            List<String> availableFats = new ArrayList<>(fats);
            customer.applyDietaryRestriction(availableCarbs,
availableProteins, availableFats);
```

```
            // Create Customer Order
            CustomerOrder customerOrder = new CustomerOrder(customer);

            // Select food items from the restaurant's menu
            List<String> menuItems = new
ArrayList<>(restaurant.getMenu().keySet());
            String selectedCarb =
menuItems.stream().filter(availableCarbs::contains).findAny().orElse(null);
            String selectedProtein =
menuItems.stream().filter(availableProteins::contains).findAny().orElse(nul
l);
            String selectedFat =
menuItems.stream().filter(availableFats::contains).findAny().orElse(null);

            // Add at least one main food item to the order
            boolean mainFoodItemAdded = false;
            if (selectedCarb != null) {
                customerOrder.addFoodItem(new BasicFoodItem(selectedCarb,
restaurant.getMenu().get(selectedCarb)));
                mainFoodItemAdded = true;
            }
            if (selectedProtein != null) {
                customerOrder.addFoodItem(new BasicFoodItem(selectedProtein,
restaurant.getMenu().get(selectedProtein)));
                mainFoodItemAdded = true;
            }
            if (selectedFat != null) {
                customerOrder.addFoodItem(new BasicFoodItem(selectedFat,
restaurant.getMenu().get(selectedFat)));
                mainFoodItemAdded = true;
            }

            if (!mainFoodItemAdded) {
                System.out.println("No suitable main food items available
for customer " + customer.getName() + " based on their dietary
restrictions.");
                continue;
            }

            // Add toppings randomly
            List<String> toppingItems = new
ArrayList<>(restaurant.getToppings().keySet());
```

```
            for (String topping : toppingItems) {
                if (random.nextBoolean()) {
                    customerOrder.addFoodItem(new ToppingDecorator(new
BasicFoodItem("Topping", 0), topping,
restaurant.getToppings().get(topping)));
                }
            }

            // Calculate total cost
            double totalCost = customerOrder.calculateTotalCost();

            // Find an available driver
            List<Driver> availableDrivers =
cppFoodDelivery.getAvailableDrivers(restaurant.getCounty(),
orderCreationTime);
            if (!availableDrivers.isEmpty()) {
                Driver driver =
availableDrivers.get(random.nextInt(availableDrivers.size()));
                Order order = new Order(restaurant, customer,
customer.getDietaryRestrictionStrategy().getClass().getSimpleName(),
customerOrder.getFoodItems(), driver, orderCreationTime);
                CustomerOrderObserver observer = new
CustomerOrderObserver(customer.getName());
                order.addObserver(observer);

                // Simulate order pickup time
                Date orderPickUpTime = new Date(orderCreationTime.getTime()
+ random.nextInt(3600000)); // Adding random time between 0 to 1 hour
                order.setOrderPickUpTime(orderPickUpTime);

                // Simulate order delivered time
                Date orderDeliveredTime = new Date(orderPickUpTime.getTime()
+ random.nextInt(7200000)); // Adding random time between 0 to 2 hours
                order.setOrderDeliveredTime(orderDeliveredTime);

                // Print order details
                System.out.println("----------------------------------------
----------------");
                System.out.println("Order Details:");
                System.out.println("Restaurant: " + restaurant.getName());
                System.out.println("Customer: " + customer.getName());
```

```
                System.out.println("Dietary Restriction: " +
customer.getDietaryRestrictionStrategy().getClass().getSimpleName());
                System.out.println("Food Items:");
                for (FoodItem item : customerOrder.getFoodItems()) {
                    System.out.println(" - " + item.getDescription() + ": $"
+ item.getCost());
                }
                System.out.println("Total Cost: $" + totalCost);
                System.out.println("Driver: " + driver.getName() + " (" +
driver.getShift() + ": " + driver.getShiftStartHour() + ":00 - " +
driver.getShiftEndHour() + ":00)");
                System.out.println("Order Creation Time: " +
order.getOrderCreationTime());
                System.out.println("Order Pick Up Time: " +
order.getOrderPickUpTime());
                System.out.println("Order Delivered Time: " +
order.getOrderDeliveredTime());

System.out.println("==========================================================
=");


                // Notify customer about order status
                order.notifyObservers();
            } else {
                System.out.println("No available driver for customer " +
customer.getName() + " at this time.");

System.out.println("==========================================================
=");
            }
        }
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=64821:/Applications/IntelliJ IDEA.app/Con
Mexican Fiesta has been registered with CPPFoodDelivery.
Italian Delight has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Indian Spice has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Jesse Pinkman has been registered with CPPFoodDelivery.
Hank has been registered with CPPFoodDelivery.
Gus Fring has been registered with CPPFoodDelivery.
Tuco has been registered with CPPFoodDelivery.
Saul Goodman has been registered with CPPFoodDelivery.
Mike Ehrmantraut has been registered with CPPFoodDelivery.
Skinny Pete has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Customer 3 has been registered with CPPFoodDelivery.
Customer 4 has been registered with CPPFoodDelivery.
Customer 5 has been registered with CPPFoodDelivery.
Customer 6 has been registered with CPPFoodDelivery.
Customer 7 has been registered with CPPFoodDelivery.
Customer 8 has been registered with CPPFoodDelivery.
Customer 9 has been registered with CPPFoodDelivery.
Customer 10 has been registered with CPPFoodDelivery.
=======================================================
Customer 1 is attempting to place an order at Indian Spice
Restaurant Indian Spice is open.
Customer 1's diet plan is Paleo. No Carbs except pistachio, No Tofu, No Dairy.
-------------------------------------------------------
Order Details:
Restaurant: Indian Spice
Customer: Customer 1
Dietary Restriction: Paleo
Food Items:
 - Pistachio: $4.0
 - Peanuts: $2.0
```

```
 - Peanuts: $2.0
Total Cost: $6.0
Driver: Mike Ehrmantraut (1st shift: 8:00 - 16:00)
Order Creation Time: Wed May 01 09:00:00 PDT 2024
Order Pick Up Time: Wed May 01 09:15:48 PDT 2024
Order Delivered Time: Wed May 01 10:38:23 PDT 2024
=======================================================
Customer Customer 1 received an update about their order: Indian Spice has prepared . It's on the way with Mike Ehrmantraut. Estimated delivery time: Wed May 01 10:38:23 PDT 202
=======================================================
Customer 2 is attempting to place an order at Indian Spice
Restaurant Indian Spice is open.
Customer 2's diet plan is Vegan. No Meat and No Dairy.
-------------------------------------------------------
Order Details:
Restaurant: Indian Spice
Customer: Customer 2
Dietary Restriction: Vegan
Food Items:
 - Pistachio: $4.0
 - Tofu: $3.0
 - Peanuts: $2.0
 - Topping, Chutney: $1.0
Total Cost: $10.0
Driver: Mike Ehrmantraut (1st shift: 8:00 - 16:00)
Order Creation Time: Wed May 01 09:00:00 PDT 2024
Order Pick Up Time: Wed May 01 09:09:20 PDT 2024
Order Delivered Time: Wed May 01 09:38:58 PDT 2024
=======================================================
Customer Customer 2 received an update about their order: Indian Spice has prepared . It's on the way with Mike Ehrmantraut. Estimated delivery time: Wed May 01 09:38:58 PDT 202
=======================================================
Customer 3 is attempting to place an order at Indian Spice
Restaurant Indian Spice is open.
Customer 3's diet plan is Nut Allergy. No Nuts.
-------------------------------------------------------
Order Details:
```

```
Order Details:
Restaurant: Indian Spice
Customer: Customer 3
Dietary Restriction: NutAllergy
Food Items:
 - Tofu: $3.0
 - Topping, Cilantro: $0.5
 - Topping, Chutney: $1.0
Total Cost: $4.5
Driver: Walter White (1st shift: 8:00 - 16:00)
Order Creation Time: Wed May 01 09:00:00 PDT 2024
Order Pick Up Time: Wed May 01 09:37:08 PDT 2024
Order Delivered Time: Wed May 01 10:05:44 PDT 2024
==========================================================
Customer Customer 3 received an update about their order: Indian Spice has prepared . It's on the way with Walter White. Estimated delivery time: Wed May 01 10:05:44 PDT 2024
==========================================================
Customer 4 is attempting to place an order at Chinese Garden
Restaurant Chinese Garden is open.
Customer 4's diet plan is No Restriction. All food items are allowed.
----------------------------------------------------------
Order Details:
Restaurant: Chinese Garden
Customer: Customer 4
Dietary Restriction: NoRestriction
Food Items:
 - Lentils: $3.5
 - Fish: $5.5
 - Tuna: $4.0
 - Topping, Scallions: $0.5
 - Topping, Soy Sauce: $0.5
Total Cost: $14.0
Driver: Hank (3rd shift: 0:00 - 8:00)
Order Creation Time: Wed May 01 01:00:00 PDT 2024
Order Pick Up Time: Wed May 01 01:21:16 PDT 2024
```

```
Order Pick Up Time: Wed May 01 01:21:16 PDT 2024
Order Delivered Time: Wed May 01 02:42:17 PDT 2024
==========================================================
Customer Customer 4 received an update about their order: Chinese Garden has prepared . It's on the way with Hank. Estimated delivery time: Wed May 01 02:42:17 PDT 2024
==========================================================
Customer 5 is attempting to place an order at Chinese Garden
Restaurant Chinese Garden is open.
Customer 5's diet plan is Paleo. No Carbs except pistachio, No Tofu, No Dairy.
----------------------------------------------------------
Order Details:
Restaurant: Chinese Garden
Customer: Customer 5
Dietary Restriction: Paleo
Food Items:
 - Fish: $5.5
 - Tuna: $4.0
 - Topping, Scallions: $0.5
 - Topping, Soy Sauce: $0.5
 - Topping, Ginger: $0.5
Total Cost: $11.0
Driver: Hank (3rd shift: 0:00 - 8:00)
Order Creation Time: Wed May 01 01:00:00 PDT 2024
Order Pick Up Time: Wed May 01 01:34:07 PDT 2024
Order Delivered Time: Wed May 01 01:41:38 PDT 2024
==========================================================
Customer Customer 5 received an update about their order: Chinese Garden has prepared . It's on the way with Hank. Estimated delivery time: Wed May 01 01:41:38 PDT 2024
==========================================================
Customer 6 is attempting to place an order at Chinese Garden
Restaurant Chinese Garden is open.
Customer 6's diet plan is Vegan. No Meat and No Dairy.
----------------------------------------------------------
Order Details:
Restaurant: Chinese Garden
Customer: Customer 6
```

```
Customer: Customer 6
Dietary Restriction: Vegan
Food Items:
 - Lentils: $3.5
 - Topping, Scallions: $0.5
 - Topping, Soy Sauce: $0.5
Total Cost: $4.5
Driver: Hank (3rd shift: 0:00 - 8:00)
Order Creation Time: Wed May 01 01:00:00 PDT 2024
Order Pick Up Time: Wed May 01 01:08:22 PDT 2024
Order Delivered Time: Wed May 01 03:06:16 PDT 2024
==========================================================
Customer Customer 6 received an update about their order: Chinese Garden has prepared . It's on the way with Hank. Estimated delivery time: Wed May 01 03:06:16 PDT 2024
==========================================================
Customer 7 is attempting to place an order at Mexican Fiesta
Restaurant Mexican Fiesta is open.
Customer 7's diet plan is Nut Allergy. No Nuts.
----------------------------------------------------------
Order Details:
Restaurant: Mexican Fiesta
Customer: Customer 7
Dietary Restriction: NutAllergy
Food Items:
 - Cheese: $3.0
 - Chicken: $5.0
 - Avocado: $2.5
 - Topping, Guacamole: $2.0
Total Cost: $12.5
Driver: Mike Ehrmantraut (1st shift: 8:00 - 16:00)
Order Creation Time: Wed May 01 10:00:00 PDT 2024
Order Pick Up Time: Wed May 01 10:56:28 PDT 2024
Order Delivered Time: Wed May 01 11:57:11 PDT 2024
==========================================================
Customer Customer 7 received an update about their order: Mexican Fiesta has prepared . It's on the way with Mike Ehrmantraut. Estimated delivery time: Wed May 01 11:57:11 PDT 2
```

```
==========================================================
Customer Customer 7 received an update about their order: Mexican Fiesta has prepared . It's on the way with Mike Ehrmantraut. Estimated delivery time: Wed May 01 11:57:11 PDT 2
==========================================================
Customer 8 is attempting to place an order at Mexican Fiesta
Restaurant Mexican Fiesta is open.
Customer 8's diet plan is No Restriction. All food items are allowed.
----------------------------------------------------------
Order Details:
Restaurant: Mexican Fiesta
Customer: Customer 8
Dietary Restriction: NoRestriction
Food Items:
 - Cheese: $3.0
 - Chicken: $5.0
 - Avocado: $2.5
 - Topping, Salsa: $1.0
 - Topping, Cheese: $1.5
Total Cost: $13.0
Driver: Gus Fring (1st shift: 8:00 - 16:00)
Order Creation Time: Wed May 01 10:00:00 PDT 2024
Order Pick Up Time: Wed May 01 10:24:38 PDT 2024
Order Delivered Time: Wed May 01 11:16:18 PDT 2024
==========================================================
Customer Customer 8 received an update about their order: Mexican Fiesta has prepared . It's on the way with Gus Fring. Estimated delivery time: Wed May 01 11:16:18 PDT 2024
==========================================================
Customer 9 is attempting to place an order at Chinese Garden
Restaurant Chinese Garden is closed. Cannot place an order.
==========================================================
==========================================================
Customer 10 is attempting to place an order at Chinese Garden
Restaurant Chinese Garden is closed. Cannot place an order.
==========================================================

Process finished with exit code 0
```

Here, we have successfully created the CPPFoodDelivery system, which has 4 restaurants, 8 drivers, and 10 customers. It shows interactions between registration with the platform, customer orders, and driver delivery.

Junit Testing

```
import org.junit.Before;
import org.junit.Test;


import java.util.*;
```

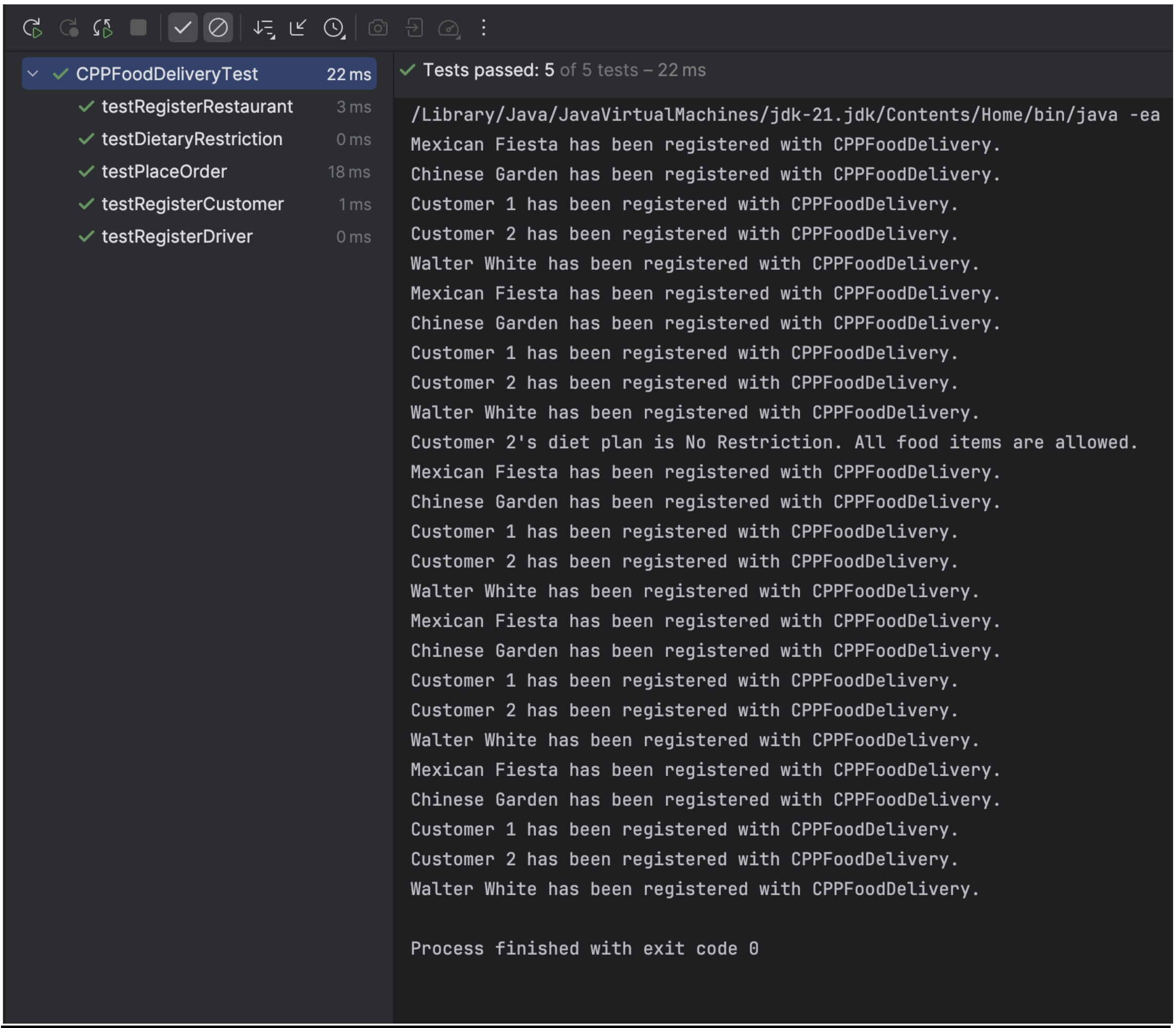
```
import static org.junit.Assert.*;

public class CPPFoodDeliveryTest {

   private CPPFoodDelivery cppFoodDelivery;
   private Restaurant mexicanFiesta;
   private Restaurant chineseGarden;
   private Customer customer1;
   private Customer customer2;
   private Driver driver1;

   @Before
   public void setUp() {
       // Initialize the singleton instance
       cppFoodDelivery = CPPFoodDelivery.getInstance();

       // Registering a restaurant
       Map<String, Double> menu1 = new HashMap<>();
       menu1.put("Cheese", 3.0);
       menu1.put("Chicken", 5.0);
       menu1.put("Avocado", 2.5);
       Map<String, Double> toppings1 = new HashMap<>();
       toppings1.put("Salsa", 1.0);
       toppings1.put("Cheese", 1.5);
       toppings1.put("Guacamole", 2.0);
       mexicanFiesta = new Restaurant("Mexican Fiesta", "123 Main St", "LA
County", "8AM - 4PM", "Mexican", menu1, toppings1, 8, 16);
       cppFoodDelivery.registerRestaurant(mexicanFiesta);

       Map<String, Double> menu2 = new HashMap<>();
       menu2.put("Lentils", 3.5);
       menu2.put("Fish", 5.5);
       menu2.put("Tuna", 4.0);
       Map<String, Double> toppings2 = new HashMap<>();
       toppings2.put("Soy Sauce", 0.5);
       toppings2.put("Ginger", 0.5);
       toppings2.put("Scallions", 0.5);
```

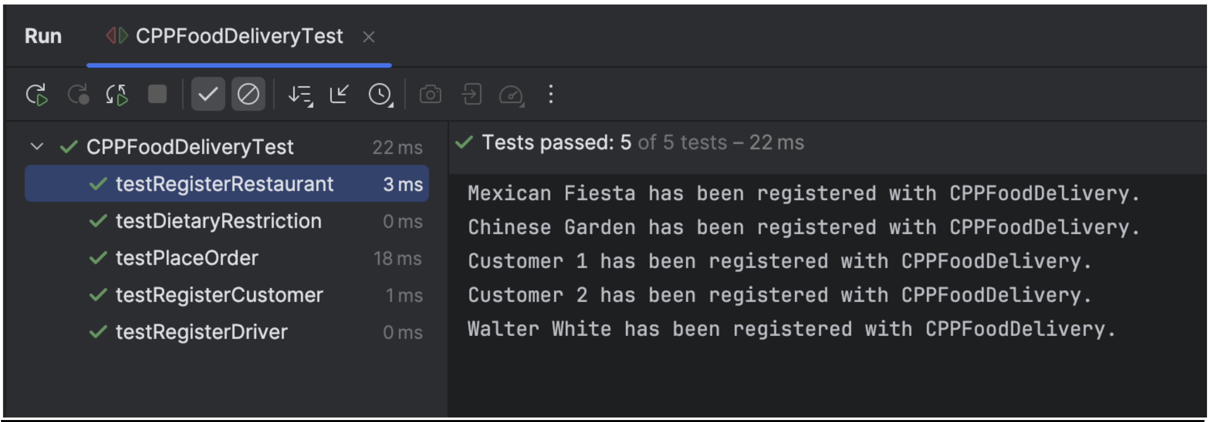
```
        chineseGarden = new Restaurant("Chinese Garden", "789 Pine St", "San Bernardino County", "12AM - 8AM", "Chinese", menu2, toppings2, 0, 8);
        cppFoodDelivery.registerRestaurant(chineseGarden);

        // Registering a customer
        customer1 = new Customer("Customer 1", "Customer Address 1", "LA County");
customer1.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("NoRestriction"));
        cppFoodDelivery.registerCustomer(customer1);

        customer2 = new Customer("Customer 2", "Customer Address 2", "San Bernardino County");
customer2.setDietaryRestrictionStrategy(DietaryRestrictionFactory.createDietaryRestriction("NoRestriction"));
        cppFoodDelivery.registerCustomer(customer2);

        // Registering a driver
        driver1 = new Driver("John Doe", "Driver Address 1", "LA County", "1st shift", 8, 16);
        cppFoodDelivery.registerDriver(driver1);
    }

    @Test
    public void testRegisterRestaurant() {
        List<Restaurant> registeredRestaurants = cppFoodDelivery.getRegisteredRestaurants();
        assertTrue(registeredRestaurants.contains(mexicanFiesta));
        assertTrue(registeredRestaurants.contains(chineseGarden));
    }

    @Test
    public void testRegisterCustomer() {
        List<Customer> registeredCustomers = cppFoodDelivery.getRegisteredCustomers();
        assertTrue(registeredCustomers.contains(customer1));
        assertTrue(registeredCustomers.contains(customer2));
    }
```

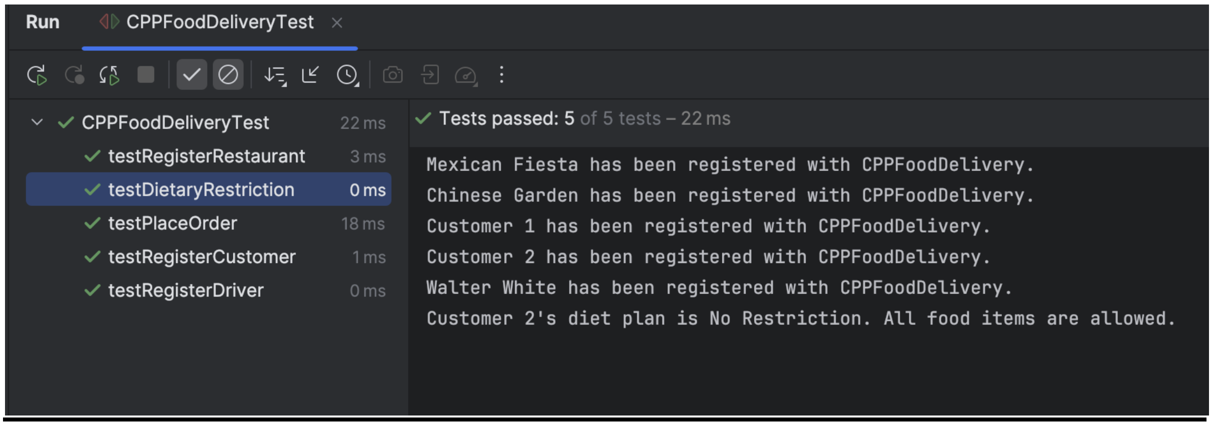
```
    @Test
    public void testRegisterDriver() {
        List<Driver> registeredDrivers =
cppFoodDelivery.getRegisteredDrivers();
        assertTrue(registeredDrivers.contains(driver1));
    }

    @Test
    public void testPlaceOrder() {
        // Customer 1 places an order from Mexican Fiesta within operating
hours
        Date orderCreationTime = new GregorianCalendar(2024, Calendar.MAY,
1, 10, 0).getTime(); // 10AM
        boolean isOpen = mexicanFiesta.isOpenDuring(orderCreationTime);
        assertTrue(isOpen);

        CustomerOrder customerOrder = new CustomerOrder(customer1);
        customerOrder.addFoodItem(new BasicFoodItem("Cheese", 3.0));
        customerOrder.addFoodItem(new BasicFoodItem("Chicken", 5.0));

        double totalCost = customerOrder.calculateTotalCost();
        assertEquals(8.0, totalCost, 0.01);

        List<Driver> availableDrivers =
cppFoodDelivery.getAvailableDrivers(mexicanFiesta.getCounty(),
orderCreationTime);
        assertTrue(availableDrivers.contains(driver1));

        Order order = new Order(mexicanFiesta, customer1,
customer1.getDietaryRestrictionStrategy().getClass().getSimpleName(),
customerOrder.getFoodItems(), driver1, orderCreationTime);
        CustomerOrderObserver observer = new
CustomerOrderObserver(customer1.getName());
        order.addObserver(observer);

        // Simulate order pickup and delivery times
        Date orderPickUpTime = new Date(orderCreationTime.getTime() +
3600000); // Adding 1 hour
        order.setOrderPickUpTime(orderPickUpTime);
```

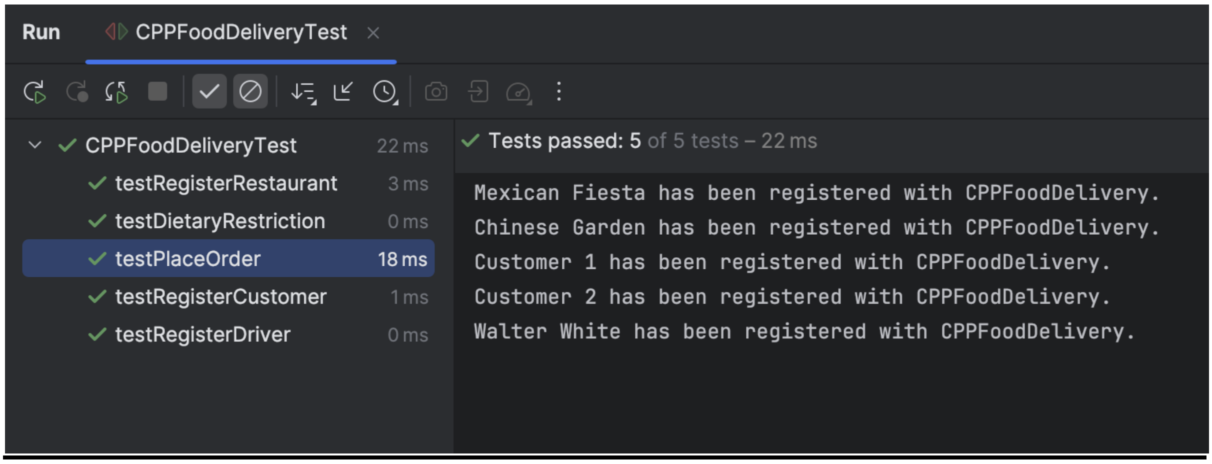
```
        Date orderDeliveredTime = new Date(orderPickUpTime.getTime() +
7200000); // Adding 2 hours
        order.setOrderDeliveredTime(orderDeliveredTime);


        assertEquals(orderPickUpTime, order.getOrderPickUpTime());
        assertEquals(orderDeliveredTime, order.getOrderDeliveredTime());
    }


    @Test
    public void testDietaryRestriction() {
        // Customer 2 (NoRestriction) orders from Mexican Fiesta
        List<String> availableCarbs = new
ArrayList<>(Arrays.asList("Cheese", "Bread", "Lentils", "Pistachio"));
        List<String> availableProteins = new
ArrayList<>(Arrays.asList("Fish", "Chicken", "Beef", "Tofu"));
        List<String> availableFats = new
ArrayList<>(Arrays.asList("Avocado", "Sour cream", "Tuna", "Peanuts"));


        customer2.applyDietaryRestriction(availableCarbs, availableProteins,
availableFats);


        // Checking that nothing gets removed
        assertTrue(availableCarbs.contains("Cheese"));
        assertTrue(availableProteins.contains("Chicken"));
        assertTrue(availableFats.contains("Sour cream"));
    }
}
```

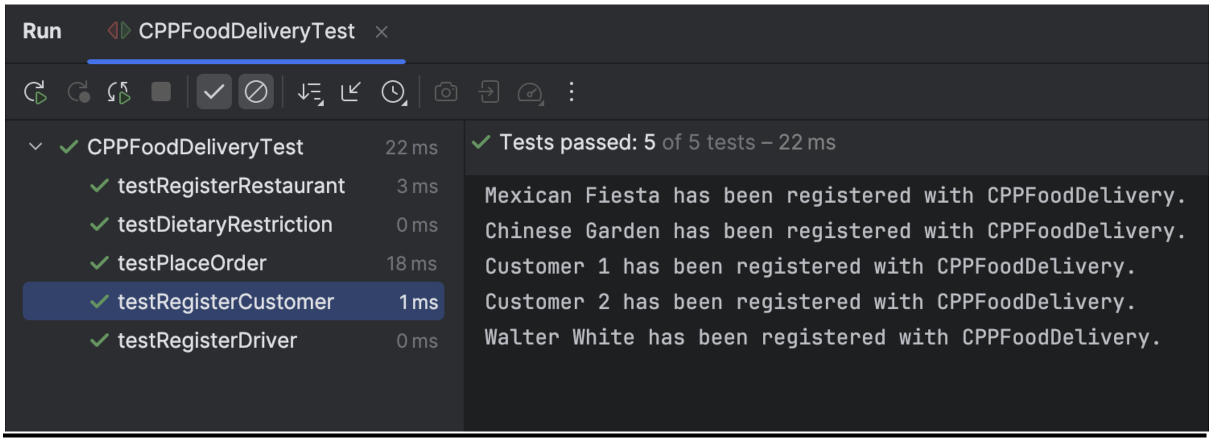
Output:

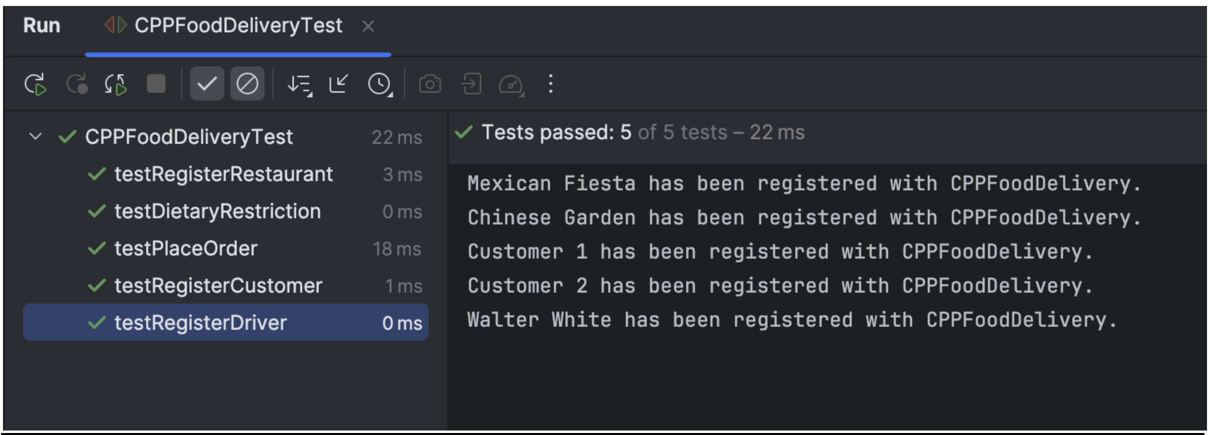

CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -ea
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Customer 2's diet plan is No Restriction. All food items are allowed.
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Process finished with exit code 0


Run
CPPFoodDeliveryTest
CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.


Run
CPPFoodDeliveryTest
CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.
Customer 2's diet plan is No Restriction. All food items are allowed.


Run
CPPFoodDeliveryTest
CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.


Run
CPPFoodDeliveryTest
CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.


Run
CPPFoodDeliveryTest
CPPFoodDeliveryTest 22 ms
testRegisterRestaurant 3 ms
testDietaryRestriction 0 ms
testPlaceOrder 18 ms
testRegisterCustomer 1 ms
testRegisterDriver 0 ms
Tests passed: 5 of 5 tests – 22 ms
Mexican Fiesta has been registered with CPPFoodDelivery.
Chinese Garden has been registered with CPPFoodDelivery.
Customer 1 has been registered with CPPFoodDelivery.
Customer 2 has been registered with CPPFoodDelivery.
Walter White has been registered with CPPFoodDelivery.

What's happening?

In this JUnit testing, we first initialized a `CPPFoodDelivery` system with restaurants, customers, and drivers. We then test various functionalities: `testRegisterRestaurant`, `testRegisterCustomer`, and `testRegisterDriver` verify the registration of restaurants, customers, and drivers, respectively. The `testPlaceOrder` method tests the order placement process, including checking restaurant operating hours, calculating the total order cost, and finding available drivers. The `testDietaryRestriction` makes sure that a customer with no dietary restrictions can access all food items, verifying the correct implementation of the `NoRestriction` dietary strategy.

PART - B

Requirements (OOA):

Functional Requirements

- Allow users to register an account with the service.
- Allow users to search for movies/shows by name.
- Allow users to browse for movies/shows by genre.
- Allow users to watch movies/shows.
- Allow users to pause a video recording.
- Allow users to resume a video recording.
- Allow users to fast forward a video recording.
- Allow users to rewind a video recording.
- Allow users to rate movies/shows.
- Allow users to indicate their preferred genres.
- Allow users to select different color schemes to accommodate for vision impairment.
- Allow users to log in and log out.
- Allow users to create and manage multiple profiles under one account.
- Implement password recovery and reset functionality.
- Provide personalized movie/show recommendations based on viewing history and ratings.
- Allow users to create and manage watchlists.
- Display recently watched items and continue watching from where the user left off.
- Allow users to share movie/show recommendations with friends.
- Allow users to follow friends and see their ratings and watchlists.
- Allow users to choose subtitles in various languages.
- Allow users to select different audio tracks for movies/shows.
- Implement parental control settings to restrict access to certain content based on ratings.
- Allow profile-specific content restrictions.
- Allow users to download movies/shows for offline viewing.
- Provide advanced search filters (e.g., release year, director, cast).

- Show trending and popular movies/shows in search results.
- Allow users to adjust video playback quality based on their internet connection.
- Allow users to choose subscription plans and manage billing information.
- Provide a detailed billing history and subscription status.
- Provide access to customer support through chat, email, or phone.
- Implement a help center with FAQs and troubleshooting guides

Use Cases (OOA):

Registering an Account

1. User opens app.
2. System prompts user for login credentials.
3. User selects button to register for a new account.
4. User inputs username and password.
5. System verifies the information with the user.
6. System creates the new account and logs the user into the service.

Logging into an Account

1. User opens app.
2. System prompts user for login credentials.
3. User inputs username and password.
4. If the given credentials are invalid, notify the user to try again.
5. If the given credentials are valid, log the user into the service.

Searching for a Movie/Show by Name

1. User navigates to the search bar.
2. User types in the name of the movie/show.
3. User selects button to search.
4. System returns a list of relevant movies/shows.

Browsing Movies/Shows by Genre

1. User navigates to the home page.
2. User scrolls through list of genres.
3. Under each listed genre, User traverses through a horizontal list of movies/shows.

Selecting a Movie/Show to Watch

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the thumbnail.
3. System displays description of the movie/show with reviews.
4. User selects the play button.
5. System begins playing the movie/show.

Pausing a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.

Resuming a Paused Video Playback

1. User selects the play button.
2. System resumes video playback.

Fast Forwarding a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.
3. User selects the fast forward button.
4. System displays available speeds.
5. User selects the desired speed.
6. System fast forwards through video playback at selected speed.
7. User presses the select button.
8. System stops fast forwarding and keeps the playback paused.
9. User selects the play button.
10. System resumes video playback where the fast forwarding stopped.

Rewinding a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.
3. User selects the rewind button.
4. System displays available speeds.
5. User selects the desired speed.
6. System rewinds through video playback at selected speed.
7. User presses the select button.
8. System stops rewinding and keeps the playback paused.
9. User selects the play button.
10. System resumes video playback where the rewinding stopped.

Rating a Movie/Show

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the thumbnail.
3. System displays description of the movie/show with reviews.
4. User navigates to the reviews section.
5. User selects the desired review from a list of available options.
6. System applies the review to the movie/show.

Indicating Preferred Genres

1. User opens the side panel.
2. User selects preferences.

3. System displays preferences page.
4. User scrolls to the genre section.
5. User selects check boxes of preferred genres from a list of available genres.
6. User selects button to save changes.
7. System updates user's genre preferences.

Changing UI Color Scheme

1. User opens the side panel.
2. User selects preferences.
3. System displays preferences page.
4. User scrolls to the accessibility section.
5. User selects the desired color scheme from the list of available schemes.
6. User selects button to save changes.
7. System updates user's UI to match selected color scheme.

Choosing Subtitles

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the subtitles button.
4. System displays available subtitle languages.
5. User selects the desired language.
6. System adds subtitles in the selected language to the video playback.

Logging out of an Account

1. User opens the app.
2. User navigates to the account settings.
3. User selects the logout option.
4. System logs the user out and returns to the login screen.

Creating a Profile

1. User logs into the account.
2. User navigates to the profile management section.
3. User selects the option to create a new profile.
4. User inputs profile name and other preferences.
5. System creates the new profile and associates it with the user's account.

Managing Profiles

1. User logs into the account.
2. User navigates to the profile management section.
3. User selects the profile to manage.
4. User updates the profile name, preferences, or deletes the profile.
5. System saves the changes.

Recovering Password

1. User opens the app.
2. User selects the "Forgot Password" option.
3. System prompts the user to enter the registered email address.
4. User inputs the email address.
5. System sends a password recovery email to the user.
6. User follows the link in the email to reset the password.
7. System prompts the user to enter a new password.
8. User inputs and confirms the new password.
9. System updates the user's password.

Viewing Recommendations

1. User logs into the account.
2. System analyzes the user's viewing history and ratings.
3. System displays personalized recommendations on the home page.

Creating a Watchlist

1. User navigates to a movie/show thumbnail.
2. User selects the "Add to Watchlist" option.
3. System adds the movie/show to the user's watchlist.

Managing Watchlists

1. User navigates to the watchlist section.
2. User selects a movie/show to remove or reorder.
3. System updates the watchlist accordingly.

Viewing Recently Watched

1. User logs into the account.
2. System displays a list of recently watched movies/shows on the home page.

Continue Watching

1. User selects a movie/show from the "Continue Watching" section.
2. System resumes playback from where the user left off.

Sharing a Movie/Show

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the "Share" option.
3. System provides options to share via social media, email, or messaging apps.
4. User selects the preferred sharing method.
5. System generates a shareable link or message.

Following Friends

1. User navigates to the friends section.
2. User searches for friends by username or email.
3. User selects the option to follow friends.

4. System updates the user's friend list.

Viewing Friends' Ratings

1. User navigates to the friends' activity section.
2. System displays a list of friends and their recent ratings and reviews.

Choosing Audio Tracks

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the audio track button.
4. System displays available audio tracks.
5. User selects the desired audio track.
6. System switches to the selected audio track.

Setting Parental Controls

1. User navigates to the settings section.
2. User selects the parental controls option.
3. System prompts the user to create a PIN.
4. User sets a PIN.
5. User selects content restrictions based on ratings.
6. System saves the settings.

Content Restrictions per Profile

1. User navigates to the profile management section.
2. User selects a profile to edit.
3. User sets content restrictions for the selected profile.
4. System saves the settings.

Downloading Content

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the "Download" option.
3. System downloads the movie/show to the user's device for offline viewing.

Using Advanced Filters

1. User navigates to the search bar.
2. User selects the advanced search option.
3. System displays available filters (e.g., release year, director, cast).
4. User applies the desired filters.
5. User selects the search button.
6. System returns a list of relevant movies/shows based on the applied filters.

Viewing Trending and Popular Content

1. User navigates to the home page.
2. System displays sections for trending and popular movies/shows.

Changing Playback Quality

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the playback quality button.
4. System displays available quality options.
5. User selects the desired quality.
6. System adjusts the playback quality accordingly.

Managing Subscription

1. User navigates to the account settings section.
2. User selects the subscription and billing option.
3. System displays available subscription plans.
4. User selects or changes the subscription plan.
5. System updates the subscription details.
6. User can also update billing information.
7. System saves the billing details.

Viewing Billing History

1. User navigates to the account settings section.
2. User selects the billing history option.
3. System displays a detailed billing history and current subscription status.

Accessing Customer Support

1. User navigates to the help section.
2. User selects the option to contact customer support.
3. System provides options for chat, email, or phone support.
4. User selects the preferred method and contacts support.

Using the Help Center

1. User navigates to the help section.
2. User selects the help center option.
3. System displays a list of FAQs and troubleshooting guides.
4. User searches or browses for relevant information.

Identified Objects & Methods (OOD):

Nouns

Verbs

Registering an Account

1. User opens app.
2. System prompts user for login credentials.
3. User selects button to register for a new account.
4. User inputs username and password.
5. System verifies the information with the user.
6. System creates the new account and logs the user into the service.

Logging into an Account

1. User opens app.
2. System prompts user for login credentials.
3. User inputs username and password.
4. If the given credentials are invalid, notify the user to try again.
5. If the given credentials are valid, log the user into the service.

Searching for a Movie/Show by Name

1. User navigates to the search bar.
2. User types in the name of the movie/show.
3. User selects button to search.
4. System returns a list of relevant movies/shows.

Browsing Movies/Shows by Genre

1. User navigates to the home page.
2. User scrolls through list of genres.
3. Under each listed genre, User traverses through horizontal list of movies/shows.

Selecting a Movie/Show to Watch

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the thumbnail.
3. System displays description of movie/show with reviews.
4. User selects the play button.
5. System begins playing the movie/show.

Pausing a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.

Resuming a Paused Video Playback

1. User selects the play button.
2. System resumes video playback.

Fast Forwarding a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.
3. User selects the fast forward button.

4. System displays available speeds.
5. User selects desired speed.
6. System fast forwards through video playback at selected speed.
7. User presses the select button.
8. System stops fast forwarding and keeps the playback paused.
9. User selects the play button.
10. System resumes video playback where the fast forwarding stopped.

Rewinding a Video Playback

1. User presses the select button.
2. System pauses video playback of the movie/show.
3. User selects the rewind button.
4. System displays available speeds.
5. User selects desired speed.
6. System rewinds through video playback at selected speed.
7. User presses the select button.
8. System stops rewinding and keeps the playback paused.
9. User selects the play button.
10. System resumes video playback where the rewinding stopped.

Rating a Movie/Show

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the thumbnail.
3. System displays description of movie/show with reviews.
4. User navigates to the reviews section.
5. User selects the desired review from a list of available options.
6. System applies the review to the movie/show.

Indicating Preferred Genres

1. User opens side panel.
2. User selects preferences.
3. System displays preferences page.
4. User scrolls to genre section.
5. User selects check boxes of preferred genres from list of available genres.
6. User selects button to save changes.
7. System updates user's genre preferences.

Changing UI Color Scheme

1. User opens side panel.
2. User selects preferences.
3. System displays preferences page.
4. User scrolls to accessibility section.
5. User selects desired color scheme from list of available schemes.
6. User selects button to save changes.

7. System updates user's UI to match selected color scheme.

Choosing Subtitles

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the subtitles button.
4. System displays available subtitle languages.
5. User selects the desired language.
6. System adds subtitles in the selected language to video playback.

Logging out of an Account

1. User opens the app.
2. User navigates to the account settings.
3. User selects the logout option.
4. System logs the user out and returns to the login screen.

Creating a Profile

1. User logs into the account.
2. User navigates to the profile management section.
3. User selects the option to create a new profile.
4. User inputs profile name and other preferences.
5. System creates the new profile and associates it with the user's account.

Managing Profiles

1. User logs into the account.
2. User navigates to the profile management section.
3. User selects the profile to manage.
4. User updates the profile name, preferences, or deletes the profile.
5. System saves the changes.

Recovering Password

1. User opens the app.
2. User selects the "Forgot Password" option.
3. System prompts the user to enter the registered email address.
4. User inputs the email address.
5. System sends a password recovery email to the user.
6. User follows the link in the email to reset the password.
7. System prompts the user to enter a new password.
8. User inputs and confirms the new password.
9. System updates the user's password.

Viewing Recommendations

1. User logs into the account.
2. System analyzes the user's viewing history and ratings.

3. System displays personalized recommendations on the home page.

Creating a Watchlist

1. User navigates to a movie/show thumbnail.
2. User selects the "Add to Watchlist" option.
3. System adds the movie/show to the user's watchlist.

Managing Watchlists

1. User navigates to the watchlist section.
2. User selects a movie/show to remove or reorder.
3. System updates the watchlist accordingly.

Viewing Recently Watched

1. User logs into the account.
2. System displays a list of recently watched movies/shows on the home page.

Continue Watching

1. User selects a movie/show from the "Continue Watching" section.
2. System resumes playback from where the user left off.

Sharing a Movie/Show

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the "Share" option.
3. System provides options to share via social media, email, or messaging apps.
4. User selects the preferred sharing method.
5. System generates a shareable link or message.

Following Friends

1. User navigates to the friends section.
2. User searches for friends by username or email.
3. User selects the option to follow friends.
4. System updates the user's friend list.

Viewing Friends' Ratings

1. User navigates to the friends' activity section.
2. System displays a list of friends and their recent ratings and reviews.

Choosing Audio Tracks

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the audio track button.
4. System displays available audio tracks.
5. User selects the desired audio track.
6. System switches to the selected audio track.

Setting Parental Controls

1. User navigates to the settings section.
2. User selects the parental controls option.
3. System prompts the user to create a PIN.
4. User sets a PIN.
5. User selects content restrictions based on ratings.
6. System saves the settings.

Content Restrictions per Profile

1. User navigates to the profile management section.
2. User selects a profile to edit.
3. User sets content restrictions for the selected profile.
4. System saves the settings.

Downloading Content

1. User navigates to the thumbnail of the desired movie/show.
2. User selects the "Download" option.
3. System downloads the movie/show to the user's device for offline viewing.

Using Advanced Filters

1. User navigates to the search bar.
2. User selects the advanced search option.
3. System displays available filters (e.g., release year, director, cast).
4. User applies the desired filters.
5. User selects the search button.
6. System returns a list of relevant movies/shows based on the applied filters.

Viewing Trending and Popular Content

1. User navigates to the home page.
2. System displays sections for trending and popular movies/shows.

Changing Playback Quality

1. During video playback, user presses the select button.
2. System pauses video playback.
3. User selects the playback quality button.
4. System displays available quality options.
5. User selects the desired quality.
6. System adjusts the playback quality accordingly.

Managing Subscription

1. User navigates to the account settings section.
2. User selects the subscription and billing option.
3. System displays available subscription plans.

4. User selects or changes the subscription plan.
5. System updates the subscription details.
6. User can also update billing information.
7. System saves the billing details.

Viewing Billing History

1. User navigates to the account settings section.
2. User selects the billing history option.
3. System displays a detailed billing history and current subscription status.

Accessing Customer Support

1. User navigates to the help section.
2. User selects the option to contact customer support.
3. System provides options for chat, email, or phone support.
4. User selects the preferred method and contacts support.

Using the Help Center

1. User navigates to the help section.
2. User selects the help center option.
3. System displays a list of FAQs and troubleshooting guides.
4. User searches or browses for relevant information.

CRC Cards (OOD):

LoginScreen	
Prompt User for credentials	User
Verify login information	User
Create new account	User
Log user into service	User
Manage password recovery	User

User	
Register for an account	LoginScreen
Store account credentials	LoginScreen
Login to the service	LoginScreen

View and update preferences	Preferences

HomePage	
Display movies/shows by section	GenreSection HistorySection RecommendationSection WatchlistSection ContinueWatchingSection FriendActivitySection PopularSection

GenreSection	
Display movies/shows by genre	HomePage Genre Movie

HistorySection	
Display previously watched movies/shows	HomePage History

PersonalizedSection	
Display movies/shows that match the user's preferences	HomePage Preferences Movie

WatchlistSection	
Display movies/shows manually added to watchlist by user	HomePage Watchlist Movie

ContinueWatchingSection

Display movies/shows that have been started but not completed by the user	HomePage Movie

FriendActivitySection	
Display recent ratings/reviews of user's friends	HomePage Review

PopularSection	
Display most popular movies/shows by recency and all time	HomePage Movie

SidePanel	
Display search bar Display user account related settings	SearchBar PreferencesSection ProfileSection FriendSection PaymentSection HelpSection LogoutButton

PreferencesSection	
Provide controls for managing user preferences	SidePanel Preferences

ProfileSection	
Provide controls for managing user profiles	SidePanel Profile

FriendSection	
Provide controls for managing friends	SidePanel User

PaymentSection	
Provide controls for managing subscription plans and billing info	SidePanel User

HelpSection	
Provide features for contacting customer support, FAQs, and troubleshooting guides	SidePanel

LogoutButton	
Log the user out of their account	SidePanel User LoginScreen

SearchBar	
Search for movies/shows by name Filter movies/shows by genre	Movie Genre

Movie	
Get movie/show details and reviews Add reviews to movies/shows Provide controls for sharing and downloading	Review Review

VideoPlayback

Play a movie	Movie
Pause a movie	Movie
Resume a paused movie	Movie
Fast forward a movie	Movie
Rewind a movie	Movie
Change speed	Movie
Add subtitles	Movie
Change playback quality	Movie

Review	
Add ratings to movies/shows	Movie
Get ratings for movies/shows	User

Genre	
List available genres	Preferences
Select preferred genres	Movie Preferences

Preferences	
Indicate preferred genres	Genre
Change UI color scheme	Profile

Watchlist	
Create watchlist	User
Add movies to watchlist	Movie

History	
Add watched movie/show to history	Movie User

Profile	
Create profile associated with user's account	User
Analyzes user's viewing history	User
Update UI according to user preferences	Preferences

Diagram (OOA):

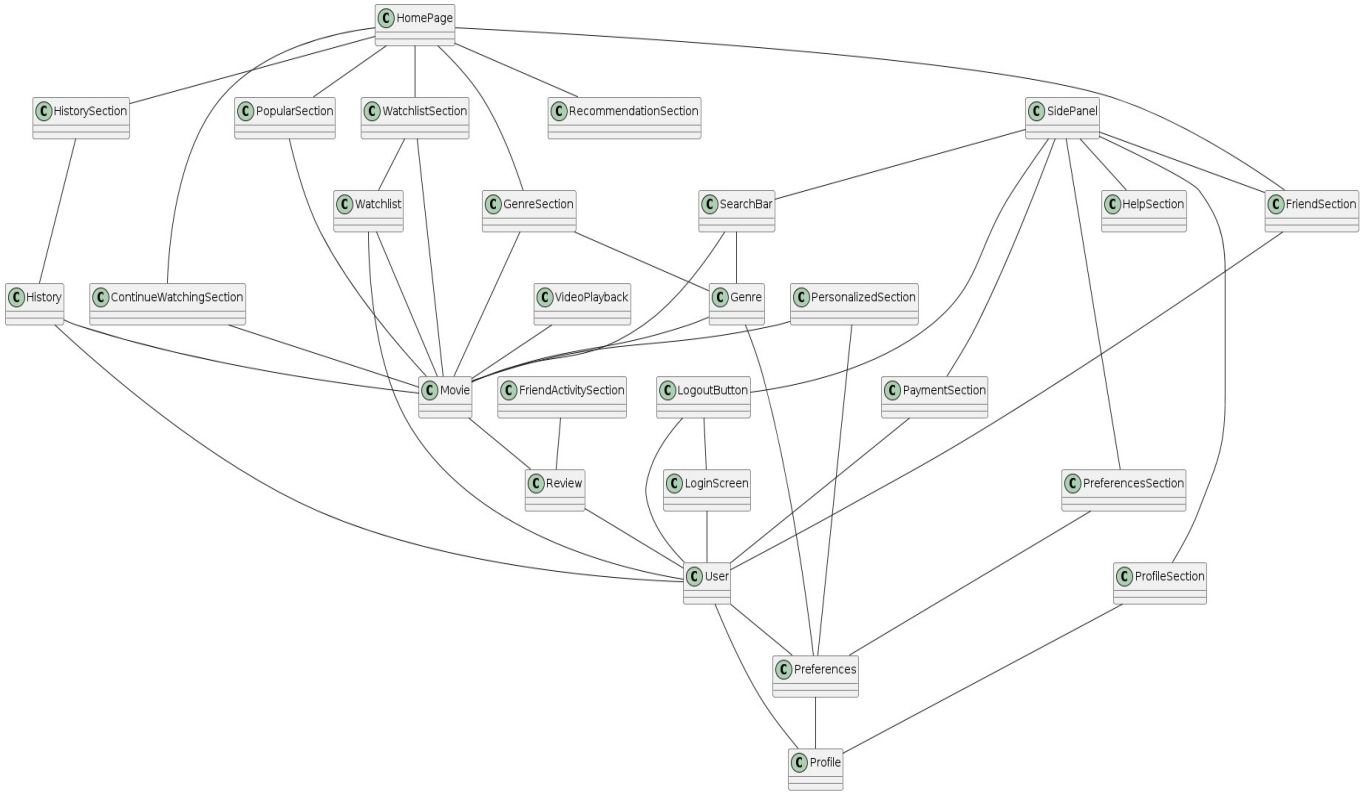
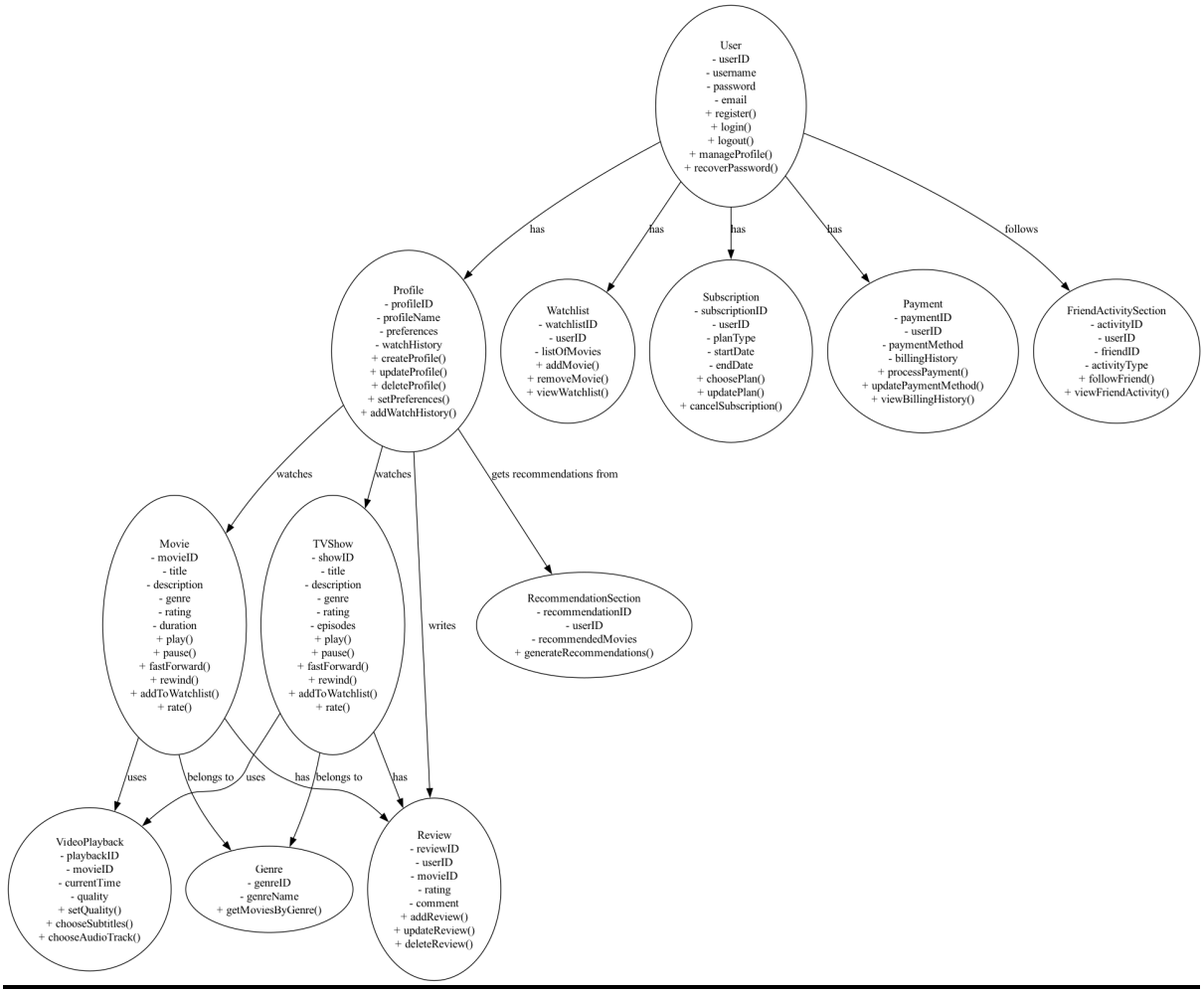


Diagram (OOD):


User
- userID
- username
- password
- email
+ register()
+ login()
+ logout()
+ manageProfile()
+ recoverPassword()
has
has
has
has
follows
Profile
- profileID
- profileName
- preferences
- watchHistory
+ createProfile()
+ updateProfile()
+ deleteProfile()
+ setPreferences()
+ addWatchHistory()
Watchlist
- watchlistID
- userID
- listOfMovies
+ addMovie()
+ removeMovie()
+ viewWatchlist()
Subscription
- subscriptionID
- userID
- planType
- startDate
- endDate
+ choosePlan()
+ updatePlan()
+ cancelSubscription()
Payment
- paymentID
- userID
- paymentMethod
- billingHistory
+ processPayment()
+ updatePaymentMethod()
+ viewBillingHistory()
FriendActivitySection
- activityID
- userID
- friendID
- activityType
+ followFriend()
+ viewFriendActivity()
watches
watches
gets recommendations from
writes
Movie
- movieID
- title
- description
- genre
- rating
- duration
+ play()
+ pause()
+ fastForward()
+ rewind()
+ addToWatchlist()
+ rate()
TVShow
- showID
- title
- description
- genre
- rating
- episodes
+ play()
+ pause()
+ fastForward()
+ rewind()
+ addToWatchlist()
+ rate()
RecommendationSection
- recommendationID
- userID
- recommendedMovies
+ generateRecommendations()
uses
belongs to
uses
has
belongs to
has
VideoPlayback
- playbackID
- movieID
- currentTime
- quality
+ setQuality()
+ chooseSubtitles()
+ chooseAudioTrack()
Genre
- genreID
- genreName
+ getMoviesByGenre()
Review
- reviewID
- userID
- movieID
- rating
- comment
+ addReview()
+ updateReview()
+ deleteReview()

THANK YOU!!!