オブジェクト指向設計 最終課題レポート

18B13863 前田 航希

2020 年 11 月 29 日

目次

1 設計概要 3

2 ユースケース記述 3

3 考察 4

3.1 設計順序 4

3.2 設計の変更 4

3.3 設計の難易度 4

3.4 設計の選択肢 4

3.5 要求の整理 5

3.6 デザインパターンの使用 5

3.7 単体テストの実行 5

3.8 エラーハンドリング 5

3.9 変更へのロバスト性 5

1　設計概要

予約・注文システムを CUI ベースで実装した。基本となる Shop クラスが在庫管理・予約管理・利用者管理を担い、客の入力を読み込んだ上で対応をする。

2　ユースケース記述

ユースケース名「商品提供システム」
概要：酒屋は客に商品を提供する一連のフローを行う。
アクター：システム・客
事前条件：

1. 客が十分に金を持っている
2. 酒の在庫がある

事後条件：

1. 客に酒が渡されているか、予約として次回入荷分に追加する。

基本フロー：

1. 客はシステムにログインする。
2. システムは客に利用可能なコマンドを表示する。
3. 客は要求コマンドとして注文を選択する。
4. システムは客に注文の入力を要求する。
5. 商品に酒類が含まれている場合、自己申告による年齢確認を行う。
6. システムは注文を履行するか予約するかを判断する。
7. それ以上の注文がない場合に、終了する。

代替フロー：

- 1. で入力された ID・名前・パスワードの組がデータベースに存在しなかった場合、入力が無効であることを通知する。
- 3. で選択されたコマンドが無効であった場合、無効であることを通知する。
- 4. で選択された商品がデータベースに存在しなかった場合、入力が無効であることを通知する。
- 5. で未成年が酒類を購入しようとしていた場合、注文が無効であることを通知する。
- 6. で予約された場合、翌日入荷分に追加する。

副シナリオ：

3. 客は予約の受け取りであることをシステムに命令する
4. システムは客の ID をキーとした予約があるかどうか判別する。
5. 予約が存在する場合、在庫にすべての商品があるかどうか判別する。
6. 注文を受けつけ、予約を破棄する

代替フロー (副シナリオ) :

- 4. で予約が存在しない場合、予約がないことを通知し、基本フローの 3 に戻る
- 5. で再び在庫がない場合、客は予約をキャンセルするかどうか選択する。キャンセルする場合、予約から消去する。キャンセルしない場合、再び予約として登録する

3 考察

3.1 設計順序

設計についてまず考えたことは, Shop という一つのクラスが全体を統括することである。主にこのクラスは店独自のものであり、コマンドラインインターフェイスとして実装されることを期待していた。したがって、何らかの出力をする場合はこのクラスに存在するものであるとした。

3.2 設計の変更

CUI の中で客と対話的に入力を受け取り処理を進めるものと、システムの起動時と終了時に行うバッチ処理ではエラーハンドリングの方法が違うため、CUI クラスは別途作成し、切り分けを行った。

また、要求を増やすにつれて Shop クラスが肥大化したため、適宜切り分けを行うことになった。結果的に Reservation のみを扱うことになり、他のクラスの中で扱うことが出来るようになった。このため、Shop クラス以外の CUI 開発の際にほかのクラスを移植することが容易になった。

3.3 設計の難易度

まず要件を定義したうえで小さなものから作っていったので、クラスの肥大化に対処するのが大変だった。

3.4 設計の選択肢

予約・取扱商品・会員など、システムを再起動したときに消えていて欲しくない情報が多く存在したので、テキストファイルとして管理する方法を選んだ。このときのエラー処理については定型文での処理のため行わなかった。

また、Order クラスに関しては、商品が複数あってそれを注文したのは一人であるために、これ

をどのように扱うか迷った。具体的には、商品一つと対応する注文者の集合とするか、商品の集合と注文者という形をとるかで迷い、最終的に直観的なわかりやすさから後者の設計とした。

3.5　要求の整理

システムは開店前に起動し、閉店後に終了するものとした。

予約は即時キャンセルされることがないものとし、システムを一度再起動したあと、すなわち翌日以降に予約の受け取りを行うものと考えた。

これに対して、入荷は毎日行うものとし、決まった数毎日入荷することとした。これはテキストファイルに保存されている。これを書き換えるか、新しくシステムを追加して注文ファイルを管理者権限によって書き換えることで、フレキシブルに注文を変更することができる。

それと同時に、予約されていた分を別途注文し入荷することとした。これにより、予約したものの商品の受け取りが容易になる。しかしながら、予約した商品をいつまでも取りに来ないというケースを考慮し、予約者専用の在庫とはせず、通常の在庫を増やす形で保管することとした。したがって、予約者よりも前に多くの注文をした場合、予約者は受け取りができないケースがある。

3.6　デザインパターンの使用

Product クラスにおいて Factory メソッドを使用した。その結果、適切なクラスの性質をもったオブジェクトを返すことができるようになったため、年齢確認などを適切に実施することができるようになった。

3.7　単体テストの実行

Customer クラスのテストとして単体テストを行い、CustomerTest クラスを作成した。これはそれぞれのメンバ変数が正しく取得できているかを確かめるテストである。

MemberList クラスの単体テストを行うために、MemberListTest クラスを作成した。メンバ変数の値が正しいかどうかに加えて、正しく会員が追加出来るかどうかもテストした。

さらに、CUI が正しく動作するかどうかを確かめるために、CUI クラスの読み込みのメソッドに限ってテストを行った。

3.8　エラーハンドリング

3.9　変更へのロバスト性