**DESIGN REVIEW OF AIMS PROJECT**

*Update 16/12/21:* Currently I’ve found no problem related to coupling in the current code.

Update 24/12/21: Currently I’ve found no cohesion problems. All the classes are functional related. The report has evaluation for the SOLID added.

Update 01/01/22: I’ve added some source codes and modified my analysis on the source code. The code is a little bit off-flow since I’m having struggle with managing the flow, and it does not have real functions for communicating with the bank yet (since I cannot use the PATCH method).

The code will be extended in the future, and this file will be updated accordingly for tracking the develop progress

These below tables are prepared for future use. Note that only problems are noted here. The goods achieved will not be mentioned

## **Couplings**

### Content coupling

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
| - | - | - | - |

### Common coupling

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
| - | - | - | - |

### Control coupling

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
| 1 | Calling controllers for placing normal order and rush order must depends on the selection of the checkbox | CartScreenHandler, PlaceNormalOrderController, PlaceRushOrderController, | Currently I’m using abstract class with some (complicated) nested include, which is not a good design. A better idea is to use a boundary class which implements an interface in order to freely call necessary functions without caring about the structure (this is the original idea of me but I’m struggling building it) |
| 2 | Payment for order must depends on the type of order placed (need the correct type of order controller) | PlaceNormalOrderController, PlaceRushOrderController, PaymentMethodScreenHandler |

### Stamp coupling

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
| 1 | Some functions of some class are passed some unnecessary data |  | Analyze types of information needed for the functions and changes where relevant |

### Data coupling

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Uncoupled

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

## **Cohesion**

### Coincidental

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Logical

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Temporal

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Procedural

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
| 1 | Constructors of place order controllers has nested relationships to serve the need of initializing the controller | PlaceOrderController, PlaceNormalOrderController, PlaceRushOrderController | Redesign the controller to have better inheritance as well as introduce interfaces to solve the problem of re-call unnecessary constructors |

### Communicational

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Sequential

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Informational

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

### Functional

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Description** | **Related modules** | **Proposed solutions** |
|  |  |  |  |

SOLID

The design follows SOLID rules well

- S: Each class is in charge of one responsibility (Controller resolves logics, Handlers resolves users’ interactions; there are also helpers responsible for database data)

- O:

+ All handlers and controllers are extended from a base class

+ Some entities are analyzed to make a concrete base abstract class (e.g. place order controllers, medias); further types of entities can extends this abstract class so as not to break down the original type

- L: Objects of child class can replace its parent class. No class has external or unwanted attribute/behavior that violates the normal actions of that line of class. Interfaces are introduced to resolve the problems of some classes have some same behaviors but different attributes and other behaviors (E.g Track, CD and DVD class implements Playable, but Track is not CD or DVD, so it does not extend Disc)

- I: Few interfaces are introduced; there are no obvious example in the project’s design

- D: Subsystems communicates via interfaces (E.g PlaceOrderController with Interbank subsystem via InterbankSubsystemInterface in function placeOrder()).

## **Design Patterns**

This project use Singleton, Façade and Strategy patterns.

* Singleton for some modules that only need one instance at runtime
* Façade for communicating between different interfaces
* Strategy pattern for building calculators for different types of orders

## **Evaluation**

Overall, most of the classes in the design are loose coupled and high cohesion related. All the functions are divided into classed with proper responsidbility according to each of which. However, the design for PlaceOrder classes need to be reconsidered to avoid circular dependency.

The design of the application can be extended more by further divide the user interfaces associate with the controllers according to different user flows for placing order. Another subsystem for place order can be used to avoid deep access to the place order logic from other subsystems.

The SOLID principles are followed well but not excellent. There are some other design patterns can be applied to further extend the application and better implement the rules.