Student Name: Nguyen Thuy Dung
Student ID: 20184244

*Supplement for design concepts and design principles*

# DESIGN CONCEPTS

## 1. Coupling

### 1.1. Content Coupling

| Related modules | Description | Improvement |
|---|---|---|
| `Order`, `RushOrder` | Attribute `deliveryInfo` is expose to other modules by function `getDeliveryInfo`(), which can be modified by calling different Hashmap's operations | Return just a value of the value for the HashMap, which still provide enough information, and prevent unexpected modifications. |

### 1.2. Common Coupling

| Related modules | Description | Improvement |
|---|---|---|
| `Cart` | Multiple modules (`CartScreenHandler`, `PlaceOrderController`, `PlaceRushOrderController`, …) have access or communicate with Cart, so we need to handle simultaneous accesses to assure data consistency. | Use singleton pattern for the class.<br>Use locking, semaphore mechanism to avoid simultaneous access or modification. |

### 1.3. Control Coupling

| Related modules | Description | Improvement |
|---|---|---|
| `CartScreenHandler` | Function `requestToPlaceOrder`(bool isRush) takes in control parameter to determine which controller to call | Separate it into 2 functions, each corresponds to a button on GUI. |

### 1.4. Stamp Coupling

| Related modules | Description | Improvement |
|---|---|---|
| `CartMedia` | The constructor of `CartMedia` take in `Cart` object, which is unnecessary and redundant | Discard cart from `CartMedia` instance, only take in necessary parameters |

### 1.5. Data Coupling

| Related modules | Description | Improvement |
|---|---|---|
| `PlaceOrderController`, `PlaceRushOrderController`, `Cart` | 2 controller classes share the singleton instance of `Cart`, modify that singleton separately with two different user threats, the data integrity is ensured. | None |

## 2. Cohesion

### 2.1. Coincidental Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| Utils Module | All utilities are placed under the same module | Some module-specific utilities can be placed in the same modules, while the general Utils module contain only general-purposed ones. |

### 2.2. Logical Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| Screen Module | All screen (GUI) handlers are placed under the same modules. | Separated each handler into its own module. |
| Order Module | Class `Order` and `RushOrder` are placed under the same module. | Separated function to handle two kinds of Order separately. |

### 2.3. Temporal Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| None | | |

### 2.4. Procedural Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| None | | |

### 2.5. Communication Cohesion

| Related modules | Description | Improvement |
|---|---|---|

| | | |
|---|---|---|
| `PaymentTransaction, InterbankSubsystem` | Work on the same input data of Card, amount; the output of `InterbankSubsystem` is provided to form Payment Transaction | None |

### 2.6. Sequential Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| `InterbankSubsystem, Payment Controller` | Work on the same input data of Card, amount; the output of InterbankSubsystem is provided to PaymentController tp form Payment Transaction | None |

### 2.7. Informational Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| None | | |

### 2.8. Functional Cohesion

| Related modules | Description | Improvement |
|---|---|---|
| ApplicationProgrammingInterface | Module to call API | None |

# DESIGN PRINCIPLES

## 1. Coupling Single Responsibility Principle

| # | Related modules | Description | Improvement |
|---|---|---|---|
| 1 | `InterbankSubsystemController` | Modules handle 2 tasks:<br>- Input data validation<br>- Data processing to pay, refund, … | Divide into 2 classes which are responsible for 2 tasks. |

| | | | |
|---|---|---|---|
| 2 | `PlaceOrderController` | Module handles 3 tasks:<br>- Create order<br>- Validate delivery information<br>- Calculate shipping fee | Divide into:<br>- 2 interfaces for creating order and validating delivery information<br>- 1 interface `ShippingFeeCalculator` |
| 3 | `PlaceRushOrderController` | Module handle 3 tasks:<br>- Create order<br>- Validate delivery information<br>- Calculate shipping fee | Divide into:<br>- 2 interfaces for creating order and validating delivery information<br>- 1 interface `ShippingFeeCalculator` |

## 2. Open/Closed Principle

| # | Related modules | Description | Improvement |
|---|---|---|---|
| 1 | `PlaceRushOrderController` | Function `calculateShippingFee`(): Modification on codebase for normal order, to calculate for rushOrder | Use additional interface `ShippingFeeCalculator` to separate the two logical pieces of code. |
| 2 | `PlaceRushOrderController` | Function `processDeliveryInfo`(): Modification the codebase in `PlaceOrderController` when changing the validation of delivery information | Use additional interface `DeliveryInfoValidation`. |

## 3. Liskov Substitution Principle

| # | Related modules | Description | Improvement |
|---|---|---|---|
| 1 | `Media` | Function `getAllMedia`():<br>return List<> but children classes override and return null | Remove redundant method in children classes |

## 4. Interface Segregation Principle

| # | Related modules | Description | Improvement |
|---|---|---|---|
| 1 | `InterbankSubSystem` | Payment Subsystem should have the two operations `payOrder`() and `refund`() | Put 2 methods into the same interface `InterbankInterface` and let two modules extend it. |

## 5. Dependency Inversion Principle

| # | Related modules | Description | Improvement |
|---|---|---|---|
| 1 | `PaymentTransaction, CreditCard` | Impossible to add new type of card without modifying `PaymentTransaction` | Make an abstract class as parent of all other types of payment card |
| 2 | `PlaceOrderController, PlaceRushOrderController and ShippingFeeCalculator` | Cannot change new formula to calculate shipping fee without modifying codes | Make an abstract class as parent of all types of calculating shipping fee |