

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

**TÀI LIỆU HƯỚNG DẪN THỰC HÀNH
THIẾT KẾ VÀ XÂY DỰNG PHẦN MỀM -
IT4490**

(LƯU HÀNH NỘI BỘ)

MỤC LỤC

1. GIỚI THIỆU	3
1.1. MỤC ĐÍCH VÀ PHẠM VI CỦA TÀI LIỆU	3
1.2. MỤC TIÊU THỰC HÀNH	3
1.3. THÔNG TIN CHUNG.....	3
2. CÁC QUY ĐỊNH ĐỐI VỚI SINH VIÊN	3
3. BÀI THỰC HÀNH SỐ 01 – THIẾT KẾ KIẾN TRÚC	3
3.1. MỤC ĐÍCH VÀ NỘI DUNG.....	3
3.2. CHUẨN BỊ.....	4
3.3. NỘI DUNG CHI TIẾT	5
3.3.1. Bắt đầu với Git/GitHub	5
3.3.2. Làm quen Astah UML.....	5
3.3.3. Thiết kế kiến trúc cho use case “Pay Order”	5
3.3.4. Biểu đồ lớp phân tích gộp	12
3.4. BÀI TẬP	13
4. BÀI THỰC HÀNH SỐ 02 – THIẾT KẾ GIAO DIỆN	13
4.1. MỤC ĐÍCH VÀ NỘI DUNG.....	13
4.2. CHUẨN BỊ	14
4.3. NỘI DUNG CHI TIẾT	14
4.3.1. Thiết kế giao diện người dùng (User Interface Design).....	14
4.3.2. Thiết kế giao diện hệ thống (System Interface Design)	20
4.4. BÀI TẬP	25
5. BÀI THỰC HÀNH SỐ 03 – THIẾT KẾ LỚP VÀ MÔ HÌNH HÓA DỮ LIỆU.....	25
5.1. MỤC ĐÍCH VÀ NỘI DUNG	25
5.2. CHUẨN BỊ	25
5.3. NỘI DUNG CHI TIẾT	25
5.3.1. Thiết kế lớp (Class Design)	25
5.3.2. Mô hình hóa dữ liệu (Data Modeling)	35
5.4. BÀI TẬP	45
6. BÀI THỰC HÀNH SỐ 04 – LẬP TRÌNH VÀ KIỂM THỬ ĐƠN VỊ.....	45
6.1. MỤC ĐÍCH VÀ NỘI DUNG	45
6.2. CHUẨN BỊ	46
6.3. NỘI DUNG CHI TIẾT	46
6.3.1. Kiểm thử đơn vị	46

6.3.2.	Lập trình.....	55
6.4.	BÀI TẬP	68
7. BÀI THỰC HÀNH SỐ 05 – CÁC KHÁI NIỆM VÀ NGUYÊN LÝ THIẾT KẾ PHẦN MỀM		69
7.1.	MỤC ĐÍCH VÀ NỘI DUNG.....	69
7.2.	CHUẨN BỊ.....	69
7.3.	NỘI DUNG CHI TIẾT	69
7.3.1.	Một số yêu cầu mở rộng.....	69
7.3.2.	Coupling và Cohesion: các khái niệm cơ bản trong thiết kế	70
7.3.3.	Nguyên lý thiết kế SOLID	74
7.4.	BÀI TẬP	80
7.4.1.	Coupling và Cohesion	80
7.4.2.	Nguyên lý thiết kế SOLID	81

1. GIỚI THIỆU

1.1. MỤC ĐÍCH VÀ PHẠM VI CỦA TÀI LIỆU

Đây là tài liệu hướng dẫn thực hành hướng dẫn sinh viên về toàn bộ các bước trong các quy trình thiết kế và xây dựng phần mềm, theo hướng tiếp cận use case và thiết kế hướng đối tượng.

1.2. MỤC TIÊU THỰC HÀNH

Các giờ THỰC HÀNH thuộc học phần IT4490 có mục tiêu giúp cho sinh viên có thể áp dụng được kiến thức trong các bài giảng trên lớp vào để thực hành, xây dựng một phần mềm theo đúng quy trình chuẩn: từ phân tích yêu cầu, thiết kế kiến trúc, thiết kế chi tiết, xây dựng, kiểm thử...

Đối với mỗi bài thực hành, sinh viên sẽ được hướng dẫn mẫu một số phần. Sau đó, dựa trên các hướng dẫn đã có, sinh viên sẽ tự thực hiện các phần nhiệm vụ tương tự được ghi cụ thể trong tài liệu hướng dẫn thực hành.

1.3. THÔNG TIN CHUNG

Thời lượng 5 buổi, Số tiết thực hành: 3 tiết/buổi

2. CÁC QUY ĐỊNH ĐỐI VỚI SINH VIÊN

Khi tham gia các buổi thực hành, sinh viên cần:

- Đi thực hành đúng giờ, giữ trật tự, không làm ảnh hưởng tới các sinh viên khác
- Tất cả các hoạt động trong buổi thực hành của em đều được thực hiện để phục vụ cho môn học. Ví dụ: Các em có thể tìm kiếm tài nguyên môn học trên Internet, nhưng không vào các trang web khác không liên quan tới môn học.
- Bài thực hành cần làm theo cá nhân. Trong buổi thực hành, thầy cô và anh chị trợ giảng (TA) có thể sẽ hỏi các em về nội dung các em đang làm, hoặc kiểm tra kết quả bài tập lab buổi trước của các em. Các bài tập buổi thực hành trước bắt buộc phải được hoàn thành hết, và các em phải giải thích được rõ ràng cho các thầy cô và các anh chị TA về các kết quả đã nộp trên github.

3. BÀI THỰC HÀNH SỐ 01 – THIẾT KẾ KIẾN TRÚC

3.1. MỤC ĐÍCH VÀ NỘI DUNG

Trước tiên, người học bước đầu làm quen với quản lý phiên bản sử dụng Git/GitHub.

Trong bài thực hành này, người học sẽ được hướng dẫn bước thiết kế kiến trúc (Architectural Design) sử dụng phần mềm Astah UML. Kết thúc bài thực hành, người học có thể nắm được cách phân tích từng use case, mỗi use case cần có biểu

đồ tương tác (interaction diagram), biểu đồ lớp phân tích (analysis class diagram) và biểu đồ lớp phân tích gộp.

Mô tả của Case Study AIMS Project được đưa trên Dropbox của môn học: https://www.dropbox.com/sh/mh890x6f0edixng/AAAAbc_8c0eTkE7ZNUANuT4Pfa?dl=0

3.2. CHUẨN BỊ

Người học cần tự hoàn thiện trước bước phân tích yêu cầu phần mềm (Requirement Analysis) trước buổi học. Kết quả của bước phân tích yêu cầu phần mềm SRS (Software Requirement Specification) sẽ là đầu vào cho bước thiết kế kiến trúc. SRS bao gồm: biểu đồ use case tổng quan, biểu đồ use case phân rã nếu có, đặc tả các use case nghiệp vụ, từ điển thuật ngữ (glossary), đặc tả phụ trợ. Ngoài ra, người học còn cần chuẩn bị trước phần mềm Astah UML. Để nhận bản quyền miễn phí cho sinh viên, truy cập đường dẫn <https://astah.net/products/free-student-license/>, cung cấp các thông tin cần thiết, và làm theo hướng dẫn của trang web để kích hoạt bản quyền.

Get a free student license

Students who have an academic email address are eligible to receive a **FREE license** for Astah UML.

- This is for a single student's personal use only.
- Instructors and teachers cannot use this student license.
- Instructors and teachers should purchase an [academic license](#).

We no longer accept applications with attachment files. This change had to be made due to a high number of dishonest applications. Please ask your teacher to [purchase a site-wide license](#).

By completing this form, you agree that you've read and agree to the Privacy Policy and consent to having Change Vision, Inc collects and store your information.

Academic Email Address*
email@sis.hust.edu.vn

School name*
Hanoi University of Science and Technology

First name*
Hieu

Last name*
Do Minh

I confirm that I entered my information correctly and I am a student at the school I entered above.*

I understand that fraudulent applications will result in permanent ineligibility to obtain a free student license.*

Apply now

Hình 1 - Đăng ký bản quyền phần mềm Astah UML miễn phí cho sinh viên

Cuối cùng, người học cần tạo tài khoản GitHub tại <https://github.com/> để quản lý phiên bản.

3.3. NỘI DUNG CHI TIẾT

3.3.1. Bắt đầu với Git/GitHub

GitHub là một trong những dịch vụ cung cấp kho lưu trữ cho phần mềm quản lý phiên bản phân tán Git (distributed version control).

Hướng dẫn sử dụng Git/GitHub:

<https://rogerdudler.github.io/git-guide/index.vi.html>

<https://o7planning.org/vi/10283/huong-dan-su-dung-github-voi-github-desktop>

<https://git-scm.com/about>

3.3.2. Làm quen Astah UML

- Hướng dẫn sử dụng Astah UML:

<https://astah.net/support/astah-pro/user-guide/>

- Hướng dẫn sử dụng Astah UML với biểu đồ trình tự:

<https://astah.net/support/astah-pro/user-guide/sequence-diagram/>

<https://www.youtube.com/embed/Qi2CsTY4LSk>

- Hướng dẫn sử dụng Astah UML với biểu đồ giao tiếp:

<https://astah.net/support/astah-pro/user-guide/communication-diagram/>

<https://www.uml-diagrams.org/communication-diagrams.html>

3.3.3. Thiết kế kiến trúc cho use case “Pay Order”

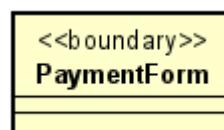
Phần này sẽ mô tả từng bước thiết kế kiến trúc cho use case “Pay Order”. Tất cả kết quả trong bài thực hành này được nộp vào thư mục “ArchitecturalDesign” trên repository cá nhân của sinh viên.

a) Phân tích lớp: Tìm các lớp (class) từ các hành vi trong use case.

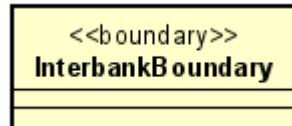
Bước 1. Tạo biểu đồ lớp mới trong Astah UML.

Bước 2. Tìm lớp boundary

- Lớp giao diện người dùng (user interface)



- Lớp giao diện hệ thống (system interface)



Bước 3. Tìm lớp entity

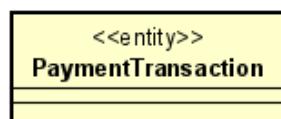
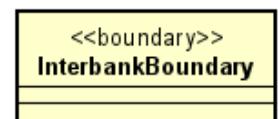


Bước 4. Tìm lớp control



Bước 5. Lưu lại biểu đồ

Kết quả thu được sau khi phân tích lớp:



b) Phân phối hành vi trong use case tới các lớp. Phân bổ trách nhiệm tới các lớp và mô hình hóa mối quan hệ giữa các lớp bằng cách sử dụng biểu đồ tương tác (interaction diagram). Chúng ta có thể sử dụng biểu đồ trình tự (sequence diagram) hoặc/và biểu đồ giao tiếp (communication diagram).

Biểu đồ trình tự

Bước 1. Tạo biểu đồ trình tự mới.

Bước 2. Kéo thả tất cả các lớp và actor liên quan ở structure tree vào khu vực vẽ biểu đồ.



: Customer



: PaymentForm



: PaymentController



: PaymentTransaction

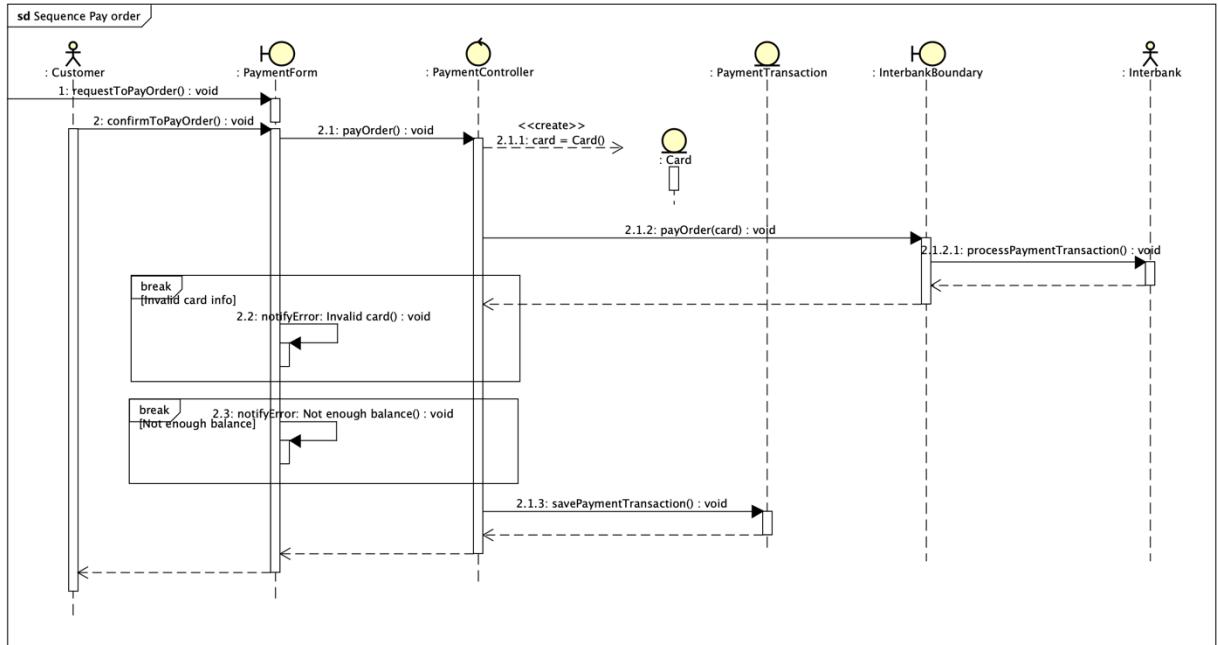


: InterbankBoundary



: Interbank

Bước 3. Phân bổ trách nhiệm tới các lớp.

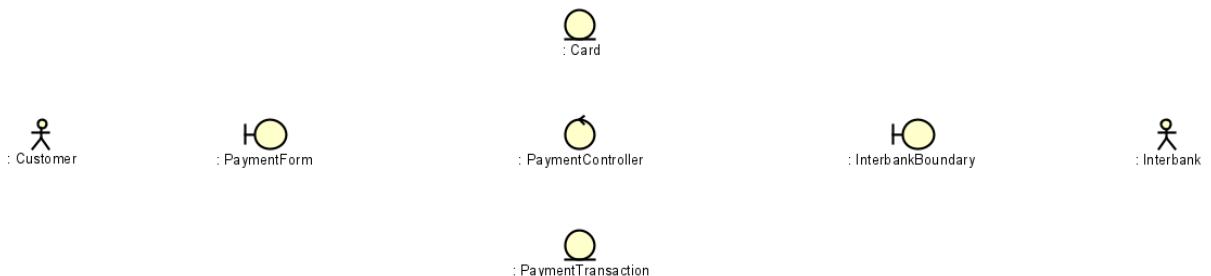


Bước 4. Lưu lại biểu đồ

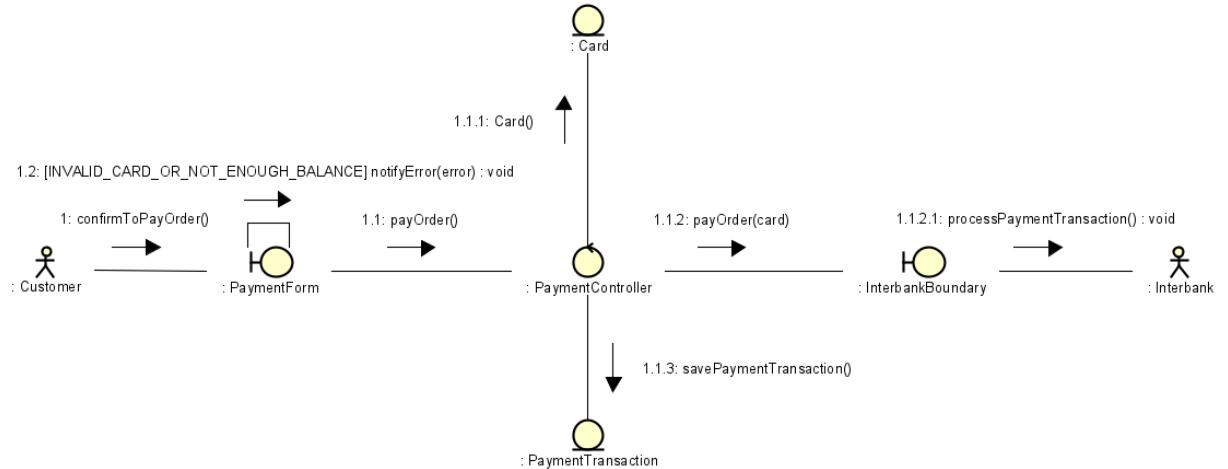
Biểu đồ giao tiếp

Bước 1. Tạo biểu đồ trình tự mới.

Bước 2. Kéo thả tất cả các lớp và actor liên quan ở structure tree vào khu vực vẽ biểu đồ.



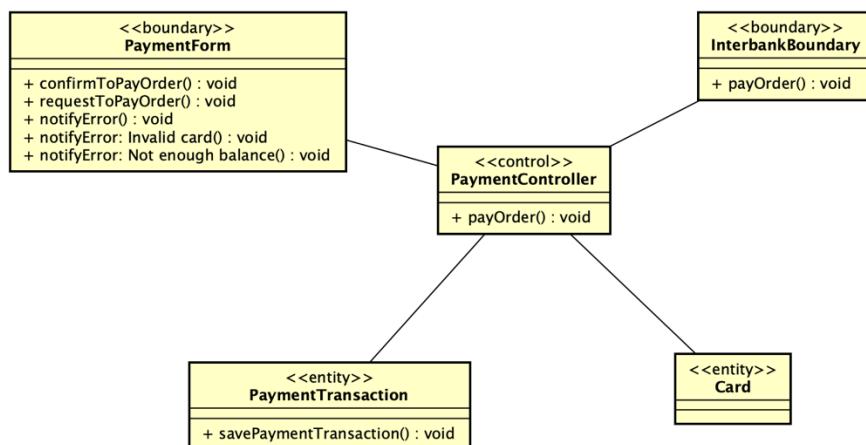
Bước 3. Phân bổ trách nhiệm tới các lớp



Bước 4. Lưu lại biểu đồ

c) Biểu đồ lớp phân tích (analysis class diagram)

Từ biểu đồ tương tác, chúng ta có thể dễ dàng vẽ được biểu đồ lớp phân tích cho use case “Pay Order.”



Phần này sẽ mô tả từng bước thiết kế kiến trúc cho use case “Place Order”. Tất cả kết quả trong bài thực hành này được nộp vào thư mục “ArchitecturalDesign” trên repository cá nhân của sinh viên.

a) Phân tích lớp: Tìm các lớp (class) từ các hành vi trong use case.

Bước 1. Tạo biểu đồ lớp mới trong Astah UML.

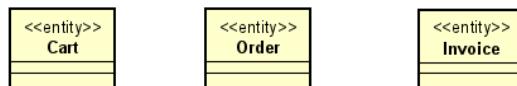
Bước 2. Tìm lớp boundary

- Lớp giao diện người dùng (user interface)



- Lớp giao diện hệ thống (system interface): Không

Bước 3. Tìm lớp entity

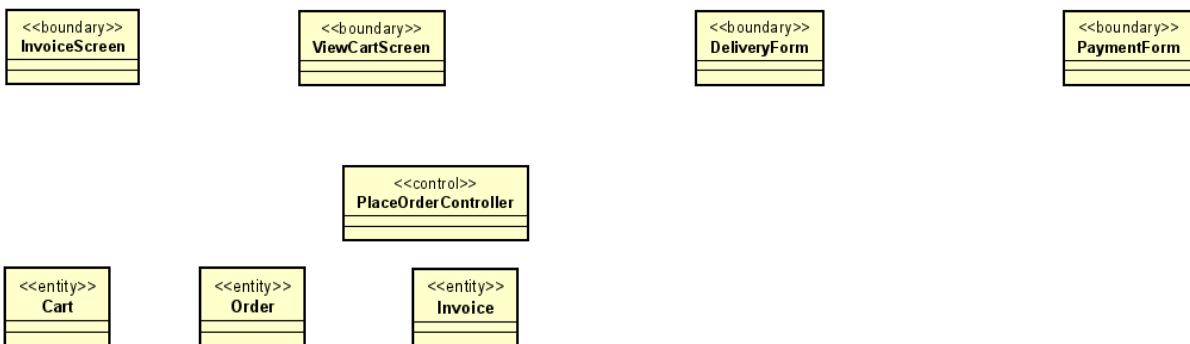


Bước 4. Tìm lớp control



Bước 5. Lưu lại biểu đồ

Kết quả thu được sau khi phân tích lớp:



b) Phân phối hành vi trong use case tới các lớp. Phân bổ trách nhiệm tới các lớp và mô hình hóa mối quan hệ giữa các lớp bằng cách sử dụng biểu đồ tương tác (interaction diagram). Chúng ta có thể sử dụng biểu đồ trình tự (sequence diagram) hoặc/và biểu đồ giao tiếp (communication diagram).

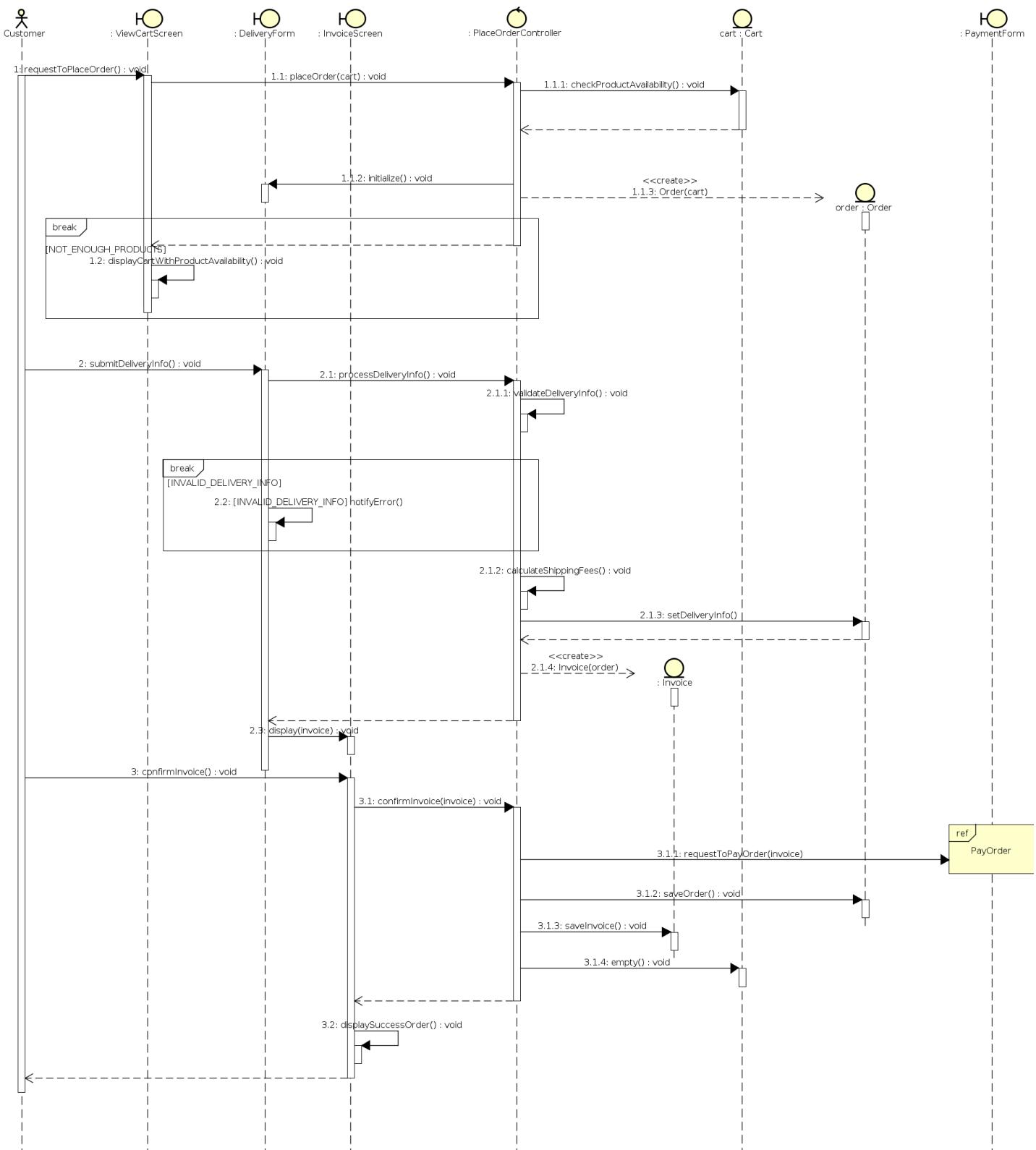
Biểu đồ trình tự

Bước 1. Tạo biểu đồ trình tự mới.

Bước 2. Kéo thả tất cả các lớp và actor liên quan ở structure tree vào khu vực vẽ.



Bước 3. Phân bổ trách nhiệm tới các lớp.



Bước 4. Lưu lại biểu đồ

Biểu đồ giao tiếp

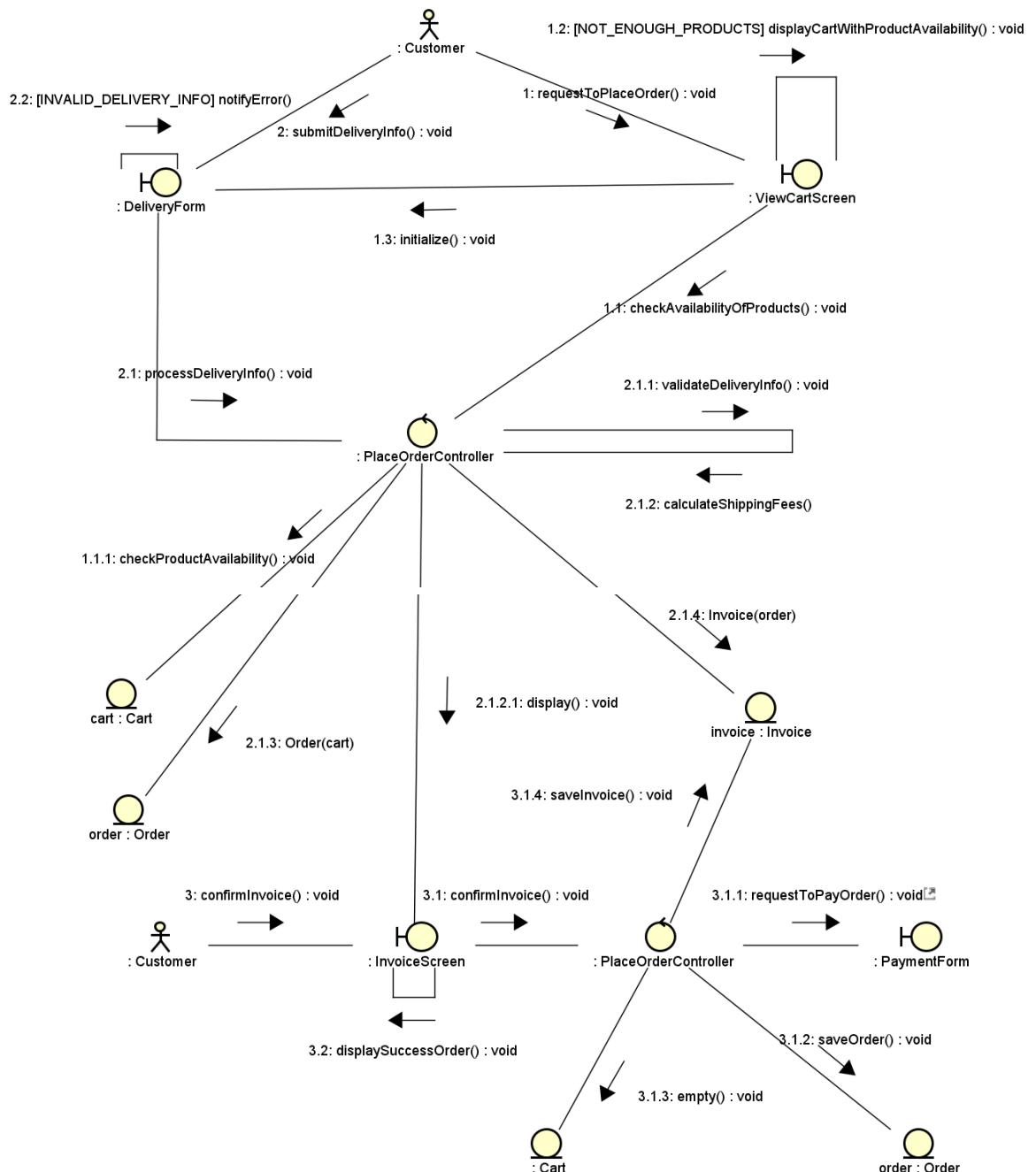
Bước 1. Tạo biểu đồ trình tự mới.

Bước 2. Kéo thả tất cả các lớp và actor liên quan ở structure tree vào khu vực vẽ.



Bước 3. Phân bổ trách nhiệm tới các lớp.

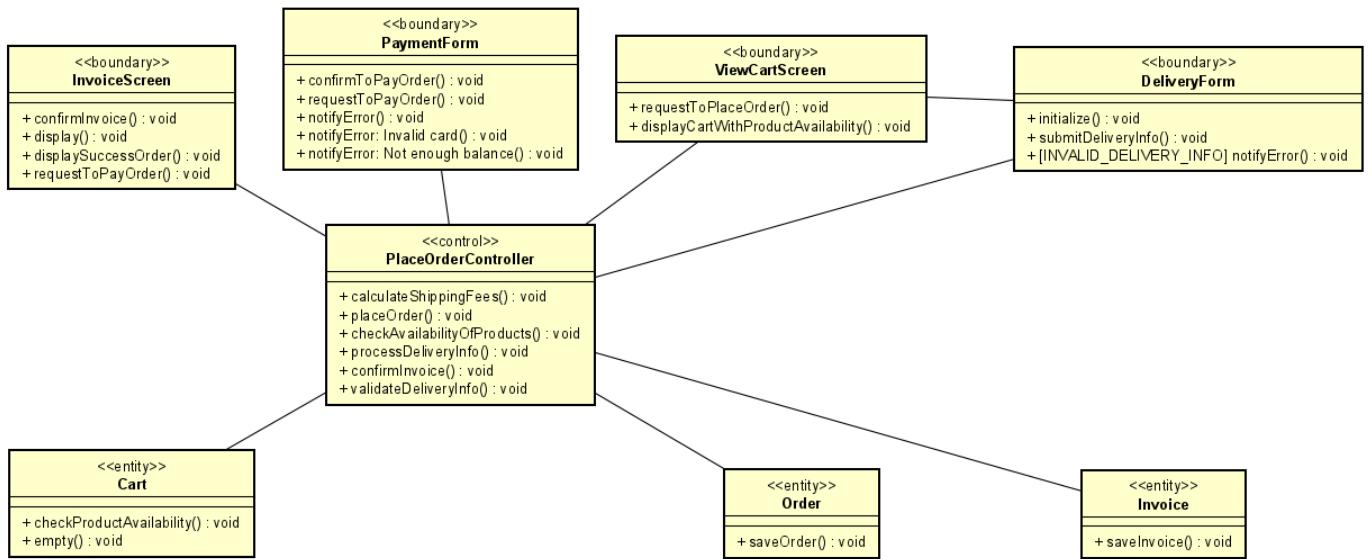
Communication Place order



Bước 4. Lưu lại biểu đồ

c) Biểu đồ lớp phân tích (analysis class diagram)

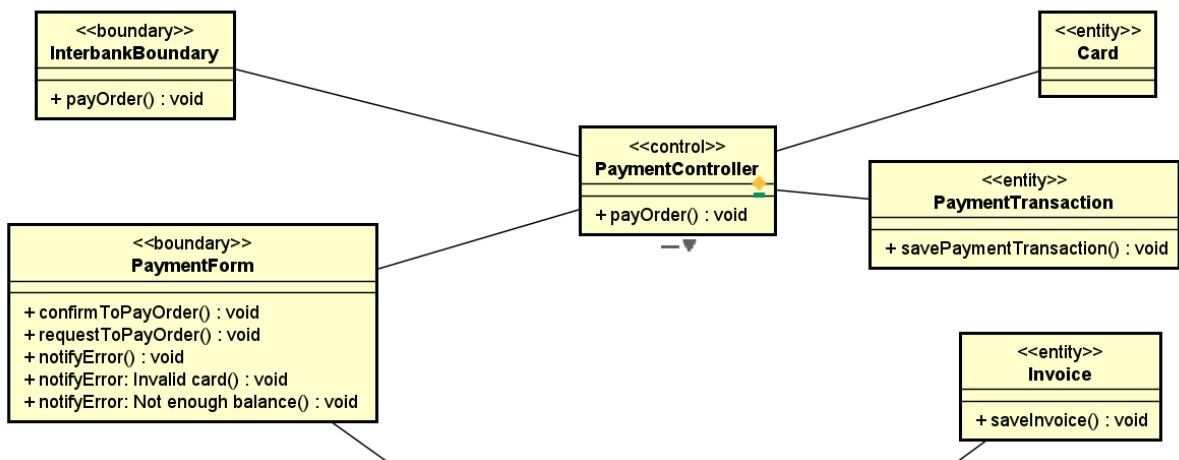
Từ biểu đồ tương tác, chúng ta có thể dễ dàng vẽ được biểu đồ lớp phân tích cho use case “Place Order”.

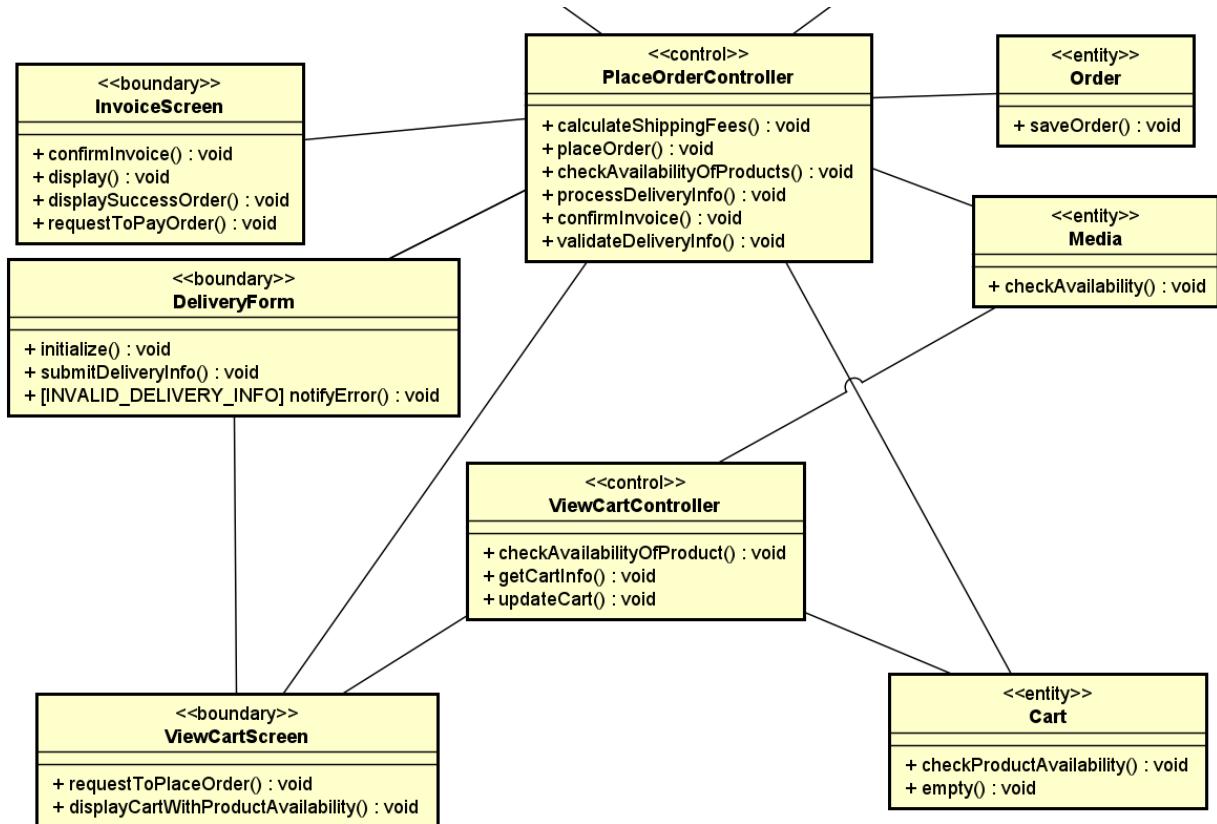


Khi hoàn thành các yêu cầu trong phần này, hãy xuất các biểu đồ ra các tệp ảnh PNG. Sau đó nộp cả tệp asta và tệp ảnh lên repository của cá nhân.

3.3.4. Biểu đồ lớp phân tích gộp

Dưới đây là một biểu đồ lớp phân tích gộp chưa hoàn thiện của use case “Place Order,” “View Cart”, “Pay Order”. Các bạn hãy cải tiến và hoàn thiện biểu đồ này.





3.4. BÀI TẬP

Hãy vẽ biểu đồ tương tác (cả biểu đồ trình tự và biểu đồ giao tiếp) của use case “Place Rush Order” và cải thiện bản thiết kế đã cho trước, trong đó có bổ sung mối liên hệ giữa Place Order và Place Rush Order trong biểu đồ tương tác.

Hãy chỉnh sửa biểu đồ lớp phân tích gộp theo những cập nhật của biểu đồ lớp phân tích cho use case “Place Order” đã làm phía trên và gộp cùng biểu đồ lớp phân tích của use case “Place Rush Order” đã làm trong phần trước.

4. BÀI THỰC HÀNH SỐ 02 – THIẾT KẾ GIAO DIỆN

4.1. MỤC ĐÍCH VÀ NỘI DUNG

Trong bài thực hành này, người học bước đầu làm quen với quá trình thiết kế chi tiết cho phần mềm (Detailed Design). Trước tiên chúng ta sẽ bắt đầu với **Interface Design** cho Case Study.

Đối với Interface Design, về cơ bản, đó là công việc thiết kế cho những lớp boundary đã có ở bước thiết kế kiến trúc (Architectural Design). Đối với các boundary class là External System Interface hoặc Device Interface thì nên xem xét chuyển thành 1 Subsystem vì nó độc lập, gắn chặt với actor.

Trong phần này, người học sẽ được hướng dẫn từng bước thiết kế Interface bao gồm thiết kế *User Interface* và thiết kế *Subsystem*.

Mô tả của Case Study AIMS Project được đưa trên Dropbox của môn học: <https://www.dropbox.com/sh/m8htvt3s9xao9f6/AAAVYjF-FDUyZkwblsOBaaTWa?dl=0>.

4.2. CHUẨN BỊ

Người học cần tự hoàn thiện trước thiết kế kiến trúc của phần mềm (Architectural Design) trước buổi học. Kết quả của bước thiết kế kiến trúc phần mềm sẽ là đầu vào cho bước thiết kế chi tiết. Các biểu đồ quan trọng cần có của bước thiết kế kiến trúc bao gồm: Biểu đồ tương tác (trình tự/giao tiếp), biểu đồ lớp phân tích.

4.3. NỘI DUNG CHI TIẾT

4.3.1. Thiết kế giao diện người dùng (User Interface Design)

Boundary Class được sử dụng để mô hình hóa tương tác giữa một hệ thống và môi trường xung quanh. Do đó, chúng có thể được sử dụng để nắm bắt các yêu cầu trên giao diện người dùng. Sự tương tác giữa con người và hệ thống có thể thông qua những loại User Interface (UI) khác nhau như Batch Interface, Command-line Interface (CLI) và Graphical User Interface (GUI).

Trong phần này, chúng ta sẽ sử dụng GUI để minh họa thiết kế UI từng bước.

4.3.1.1. Chuẩn hóa cấu hình màn hình

Display

Số lượng màu được hỗ trợ: 16,777,216 màu

Độ phân giải: 1366 x 768 pixels

Screen

Vị trí của của button: Ở dưới cùng (theo chiều dọc) và ở giữa (theo chiều ngang) của khung.

Vị trí của message: Ở giữa trung tâm khung màn hình

Vị trí của screen title: Title đặt ở góc trên bên trái của màn hình.

Sự nhất quán trong hiển thị chữ số: dấu phẩy để phân cách hàng nghìn và chuỗi chỉ bao gồm các ký tự, chữ số, dấu phẩy, dấu chấm, dấu cách, dấu gạch dưới và ký hiệu gạch nối.

Control

Kích thước text: medium size (24px). Font: Segoe UI. Color: #000000

Xử lý check input: Nên kiểm tra xem input có empty hay không. Tiếp theo, kiểm tra xem input có đúng format hay không.

Dịch chuyển màn hình: Không có các khung chồng lên nhau. Các màn hình được tách biệt. Tuy nhiên, hướng dẫn sử dụng được xem như là 1 popup message vì màn hình chính ở dưới sẽ không thể thao tác trong khi màn hình hướng dẫn sử dụng đang được hiển thị. Ban đầu khi app khởi chạy thì màn hình splash screen (màn hình chớp) sẽ được hiện lên và sau đó màn hình đầu tiên(Home Screen) sẽ xuất hiện

Thứ tự các màn hình trong hệ thống:

1. Splash screen (first screen)
2. Home screen
3. View cart screen – xem các sản phẩm trong giỏ hàng
4. Delivery form – Điền thông tin giao hàng
5. Invoice screen – Xem chi tiết order
6. Payment form – Điền thông tin thanh toán
7. Result screen

Nhập input từ bàn phím

Sẽ không có phím tắt. Có các button quay lại để quay lại các màn hình trước đó. Ngoài ra button “X” nằm ở thanh tiêu đề bên phải để đóng screen

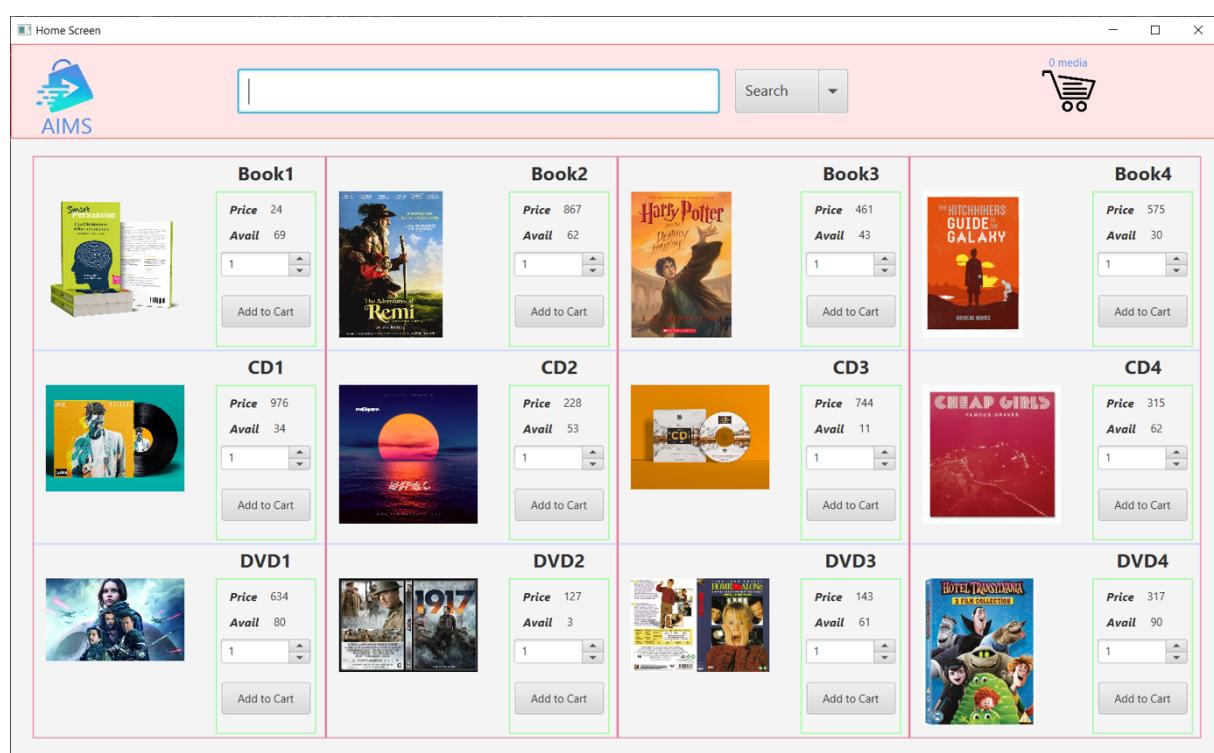
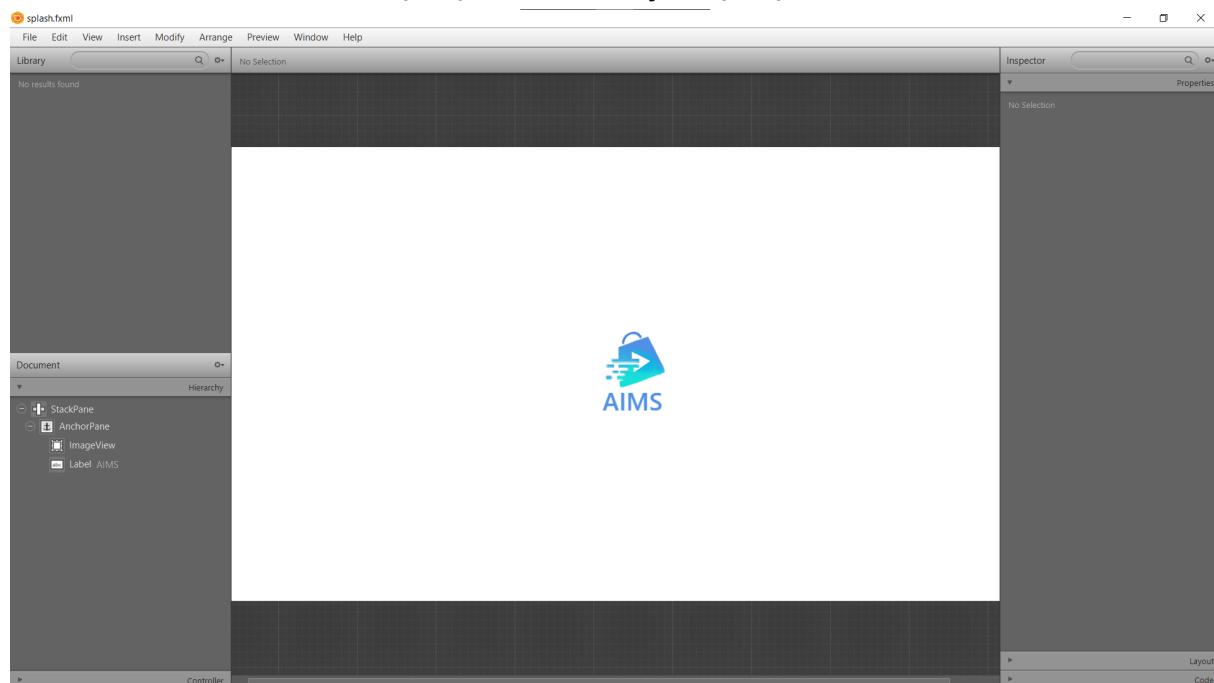
Error

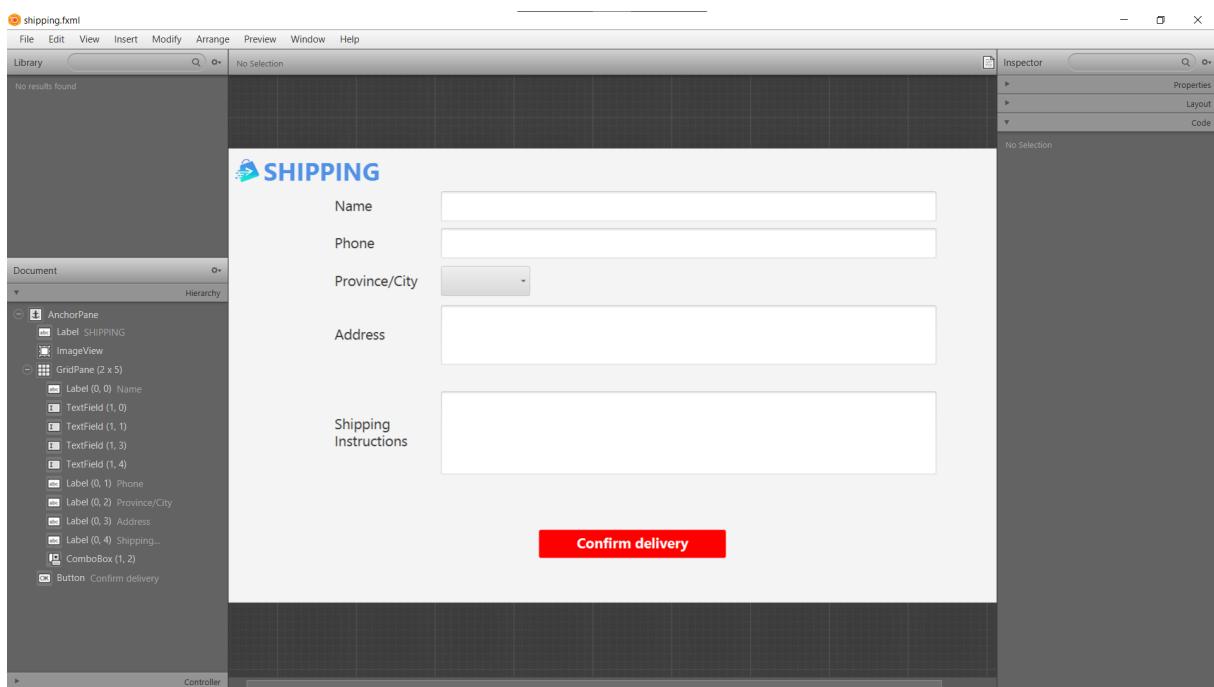
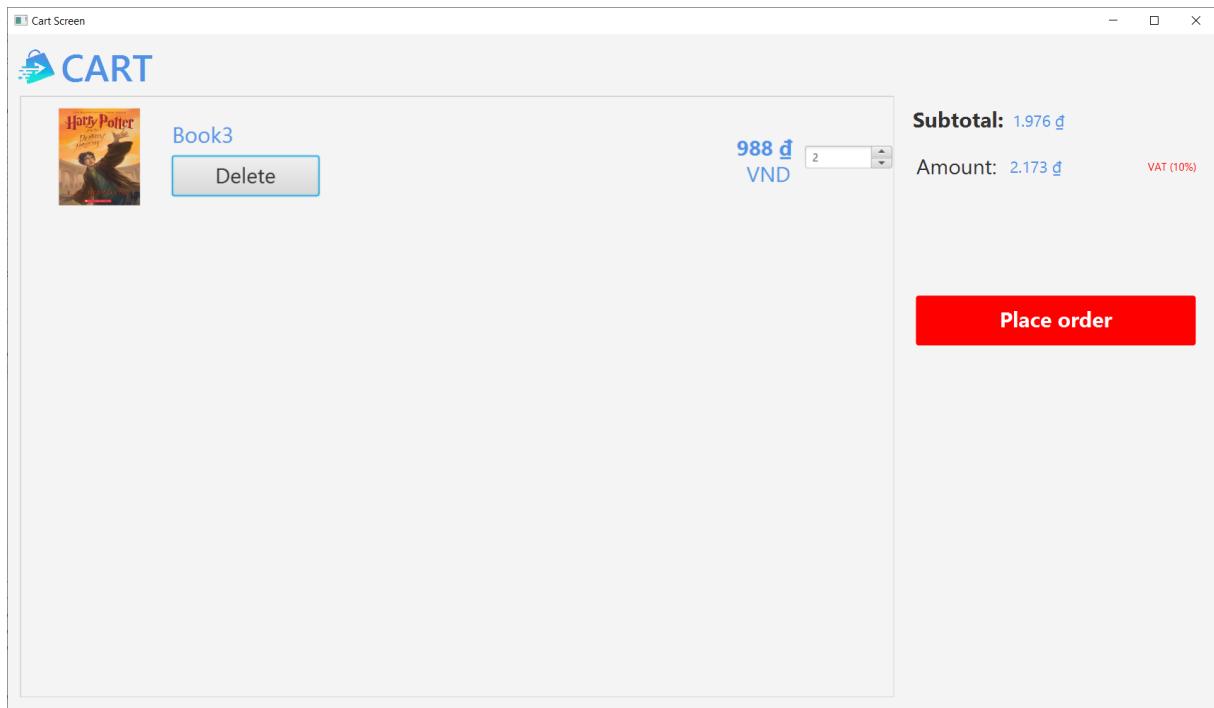
Một thông điệp sẽ được hiện lên để thông báo cho người dùng biết vấn đề đang gặp phải là gì.

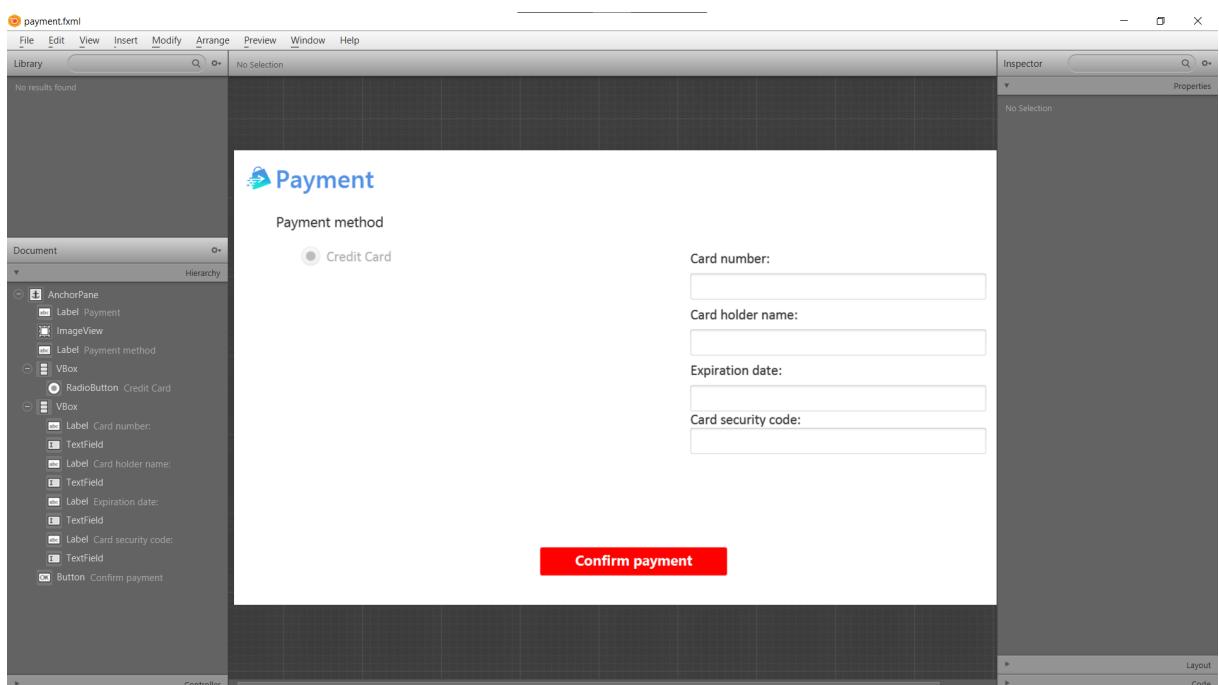
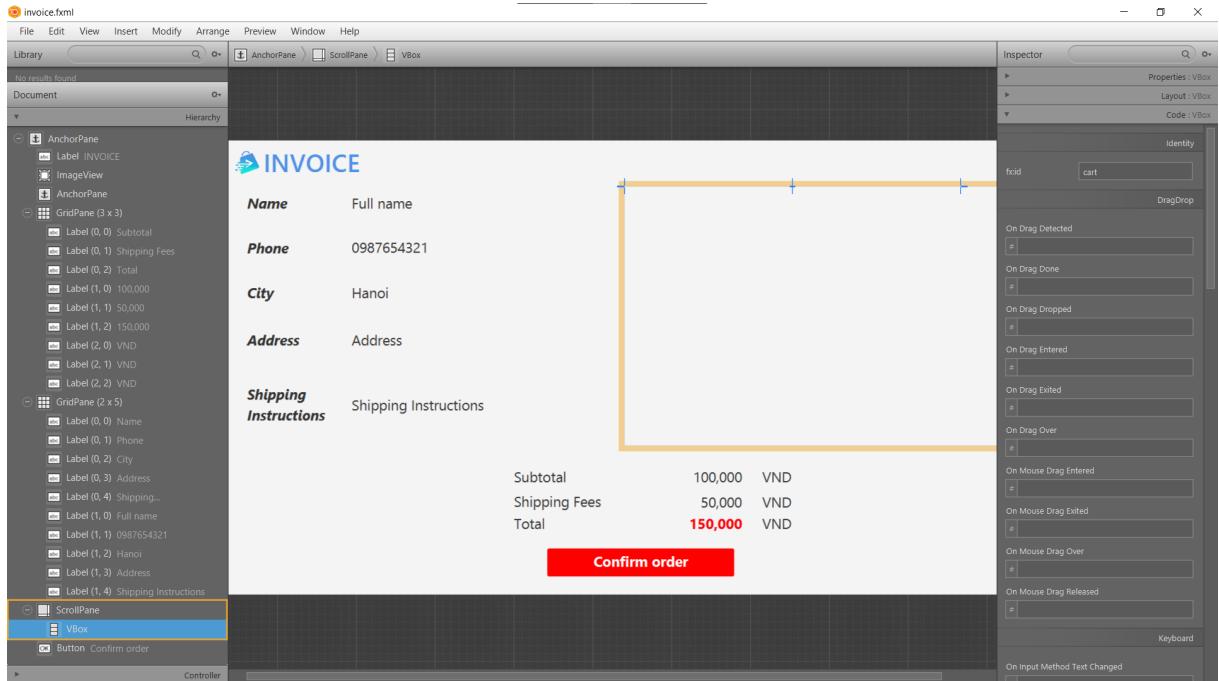
4.3.1.2. Tạo các ảnh màn hình

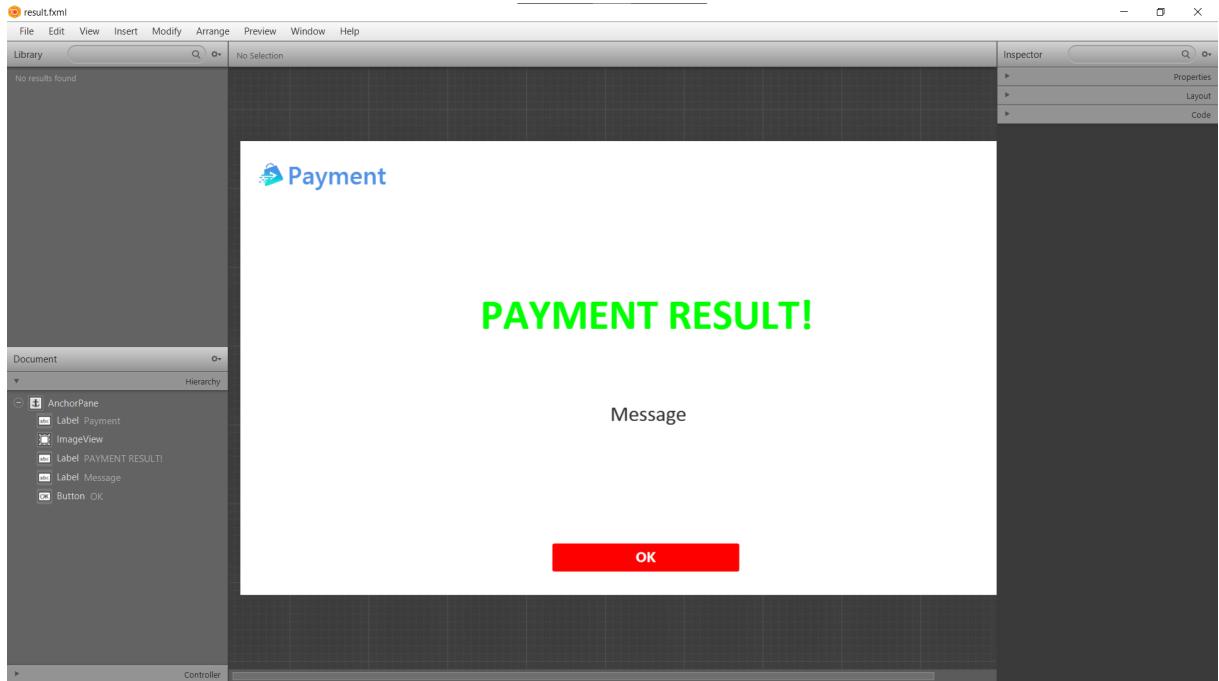
Tạo ảnh màn hình hoặc bản mockup có thể được thực hiện bằng cách sử dụng các ứng dụng như <https://moqups.com/>, Figma, InVision Studio, Paint, Adobe Graphic design software, Adobe XD hay Scene Builder...

Các hình ảnh màn hình được tạo ra dưới đây được tạo ra bởi Scene Builder

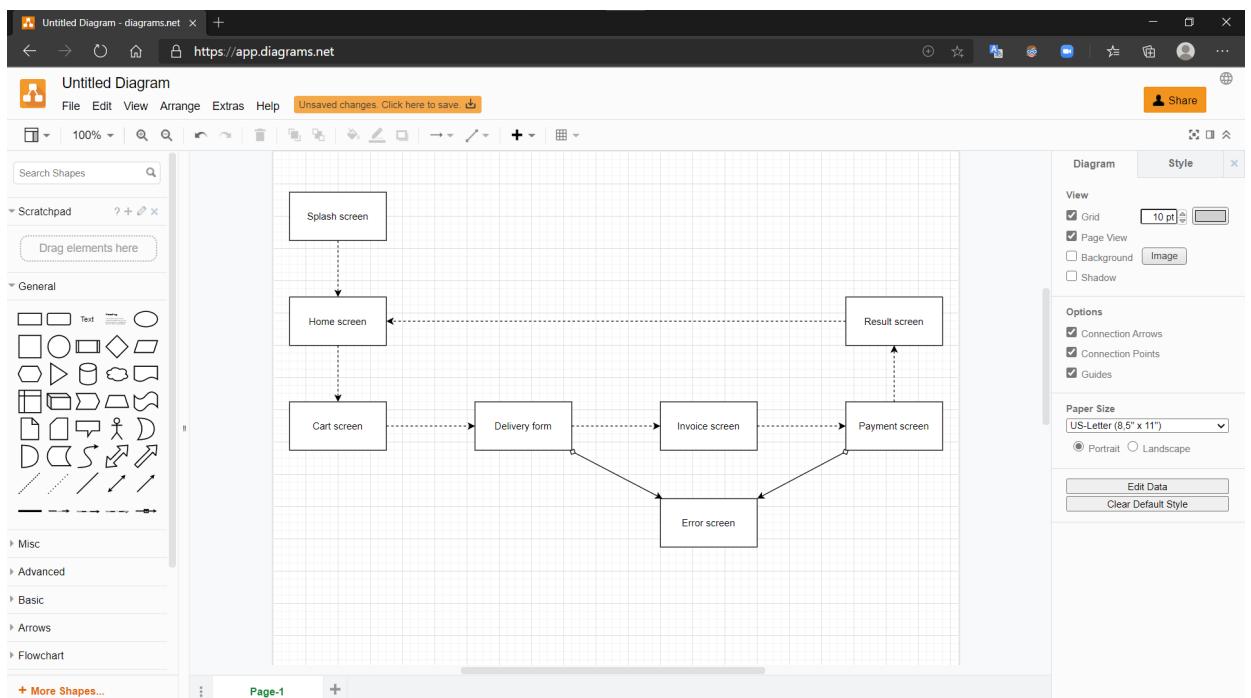




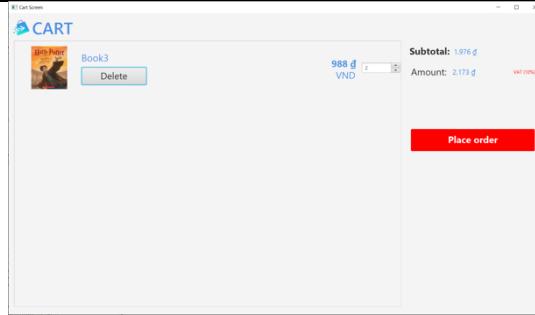




4.3.1.3. Tạo các dịch chuyển màn hình



4.3.1.4. Mô tả các màn hình

AIMS Software		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	View cart screen	30/10/2020			Đỗ Minh Hiếu
		Control	Operation	Function	
		Area for displaying the subtotal	Initial	Display the subtotal	
		Area for display items in the cart	Initial	Display the media with the corresponding information	
		Place order button	Click	Display the Delivery form	
		Delete button	Click	Remove the item from the cart	

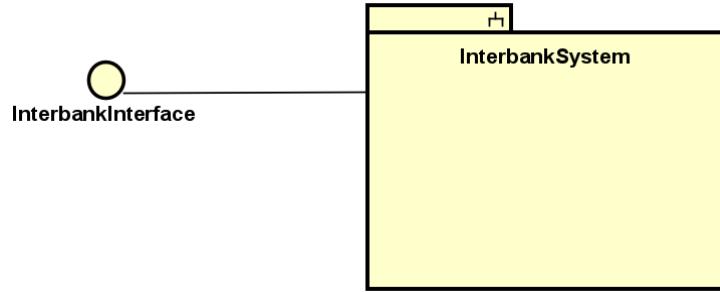
Định nghĩa các trường thuộc tính

Screen name	View cart			
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Media title	50	Numeral	Blue	Left-justified
Price	20	Numeral	Blue	Right justified
Subtotal	20	Numeral	Blue	Left-justified

4.3.2. Thiết kế giao diện hệ thống (System Interface Design)

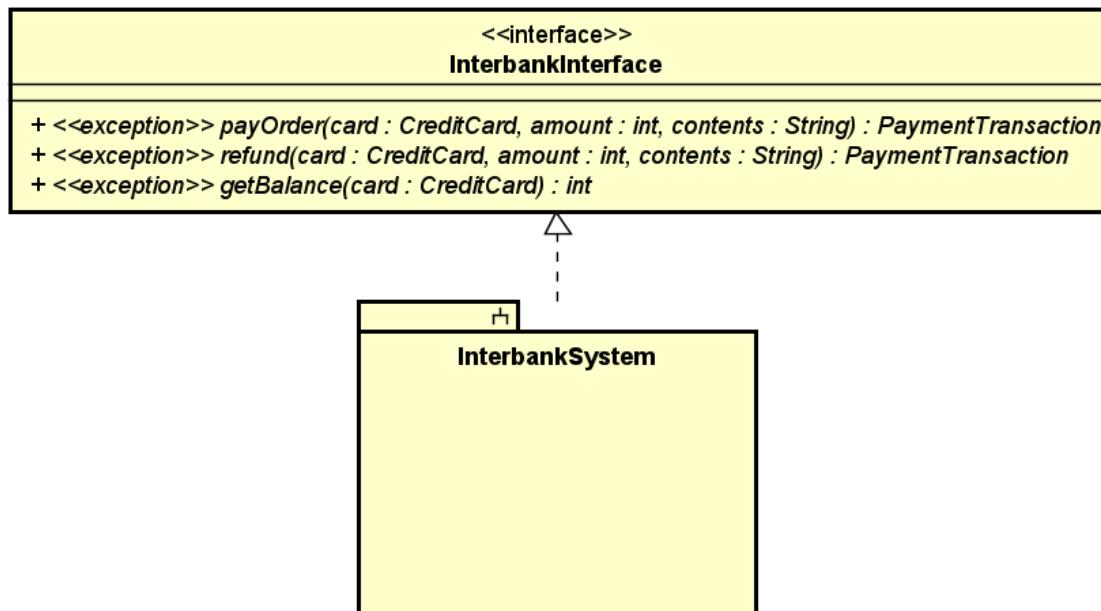
4.3.2.1. Tìm ra các subsystem

Có thể thấy, InterbankBoundary class trong Analysis Class Diagram cung cấp các dịch vụ phức tạp liên quan đến giao tiếp giữa AIMS Software và Interbank. Ngoài ra, nó cũng là một External System Interface, nó độc lập, gắn chặt với actor Interbank. Do đó, ta cần chuyển InterbankBoundary từ một Analysis Class thành một Subsystem.



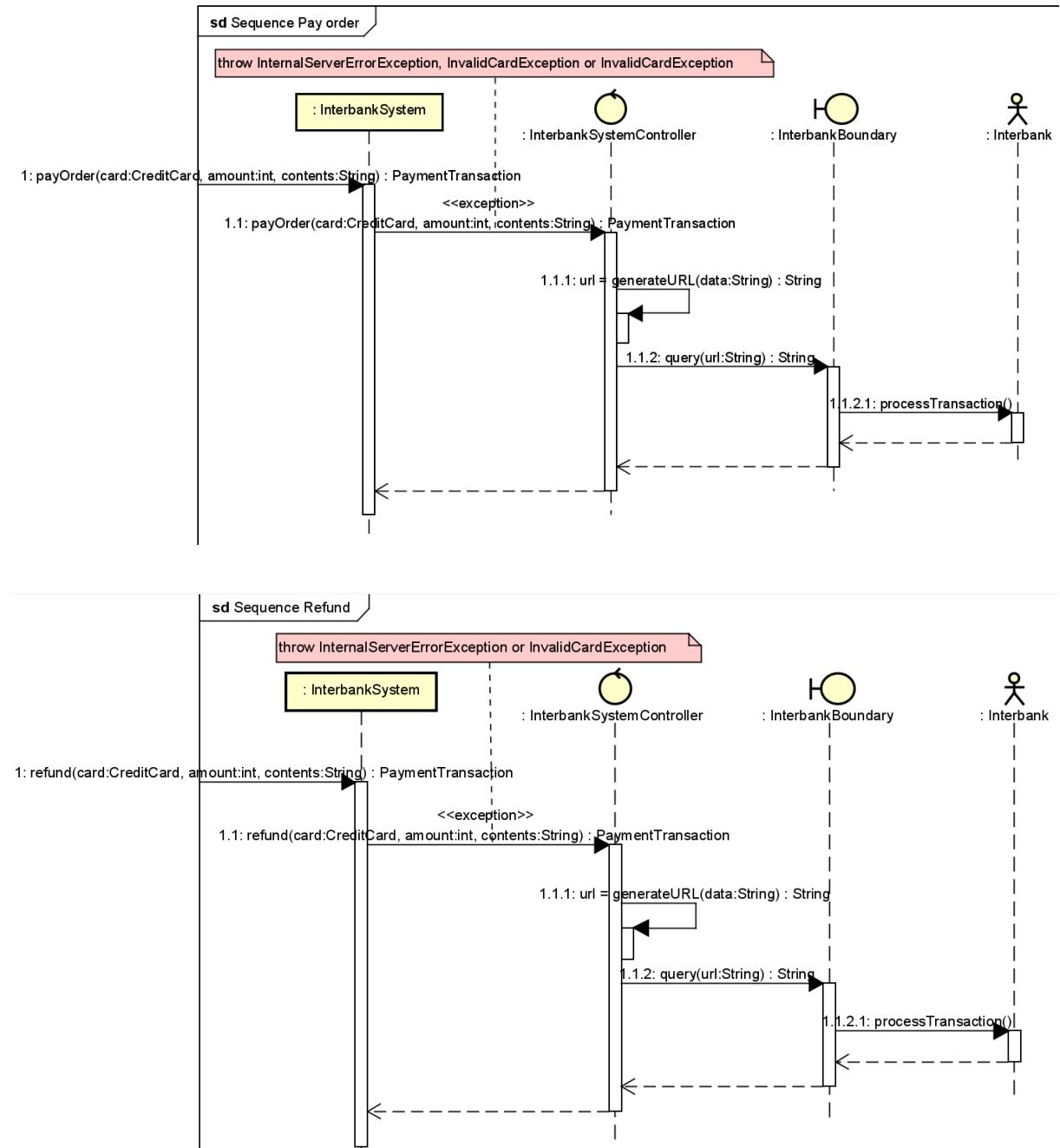
4.3.2.2. Thiết kế interface cho subsystem

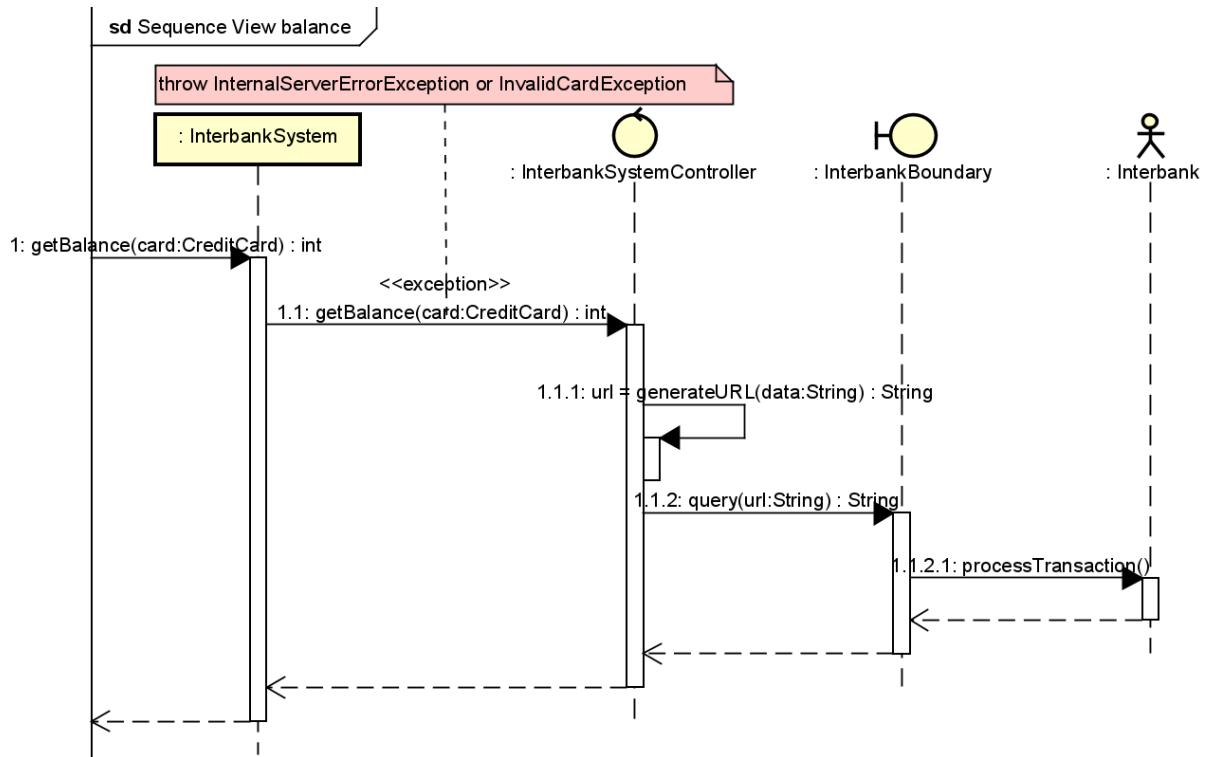
Dựa vào trách nhiệm chính của một hệ thống thanh toán, chúng ta có thể xác định được interface cho Subsystem như sau:



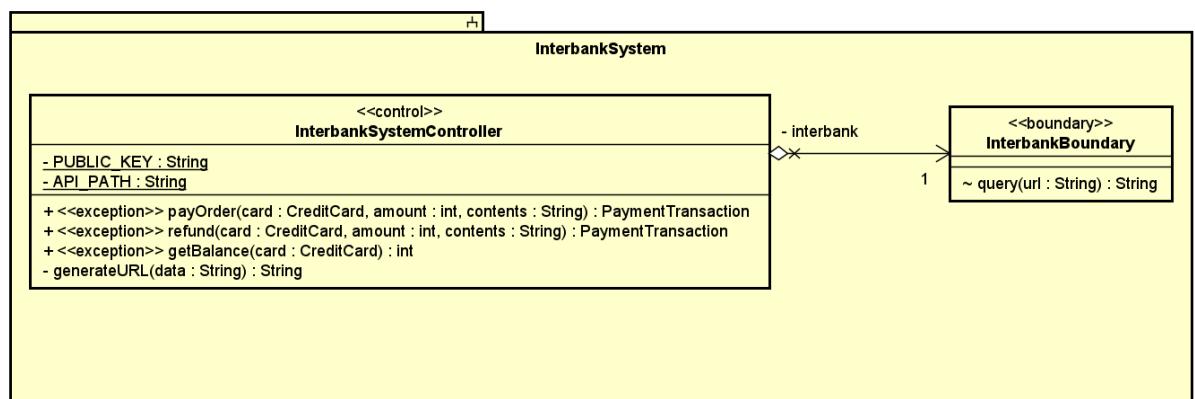
4.3.2.3. Thiết kế Subsystem

Distribute subsystem behavior to subsystem elements

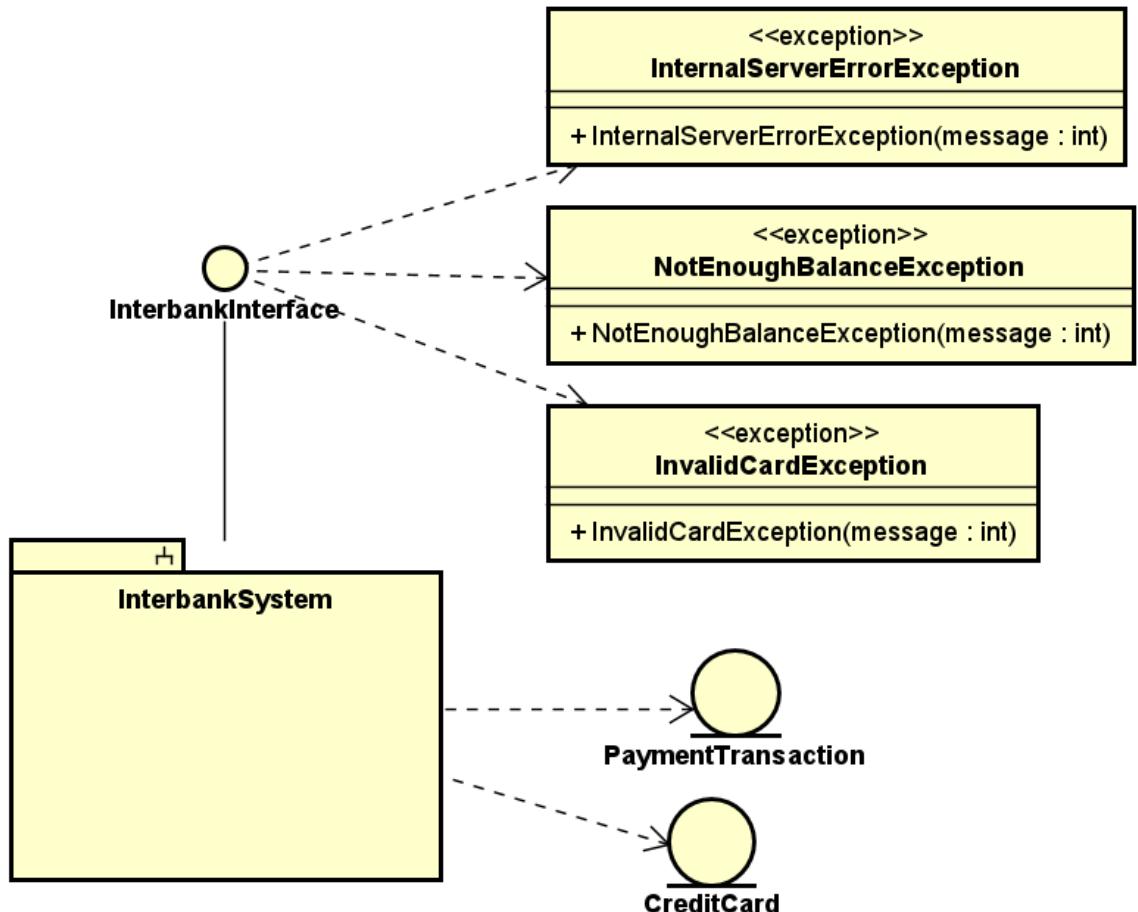




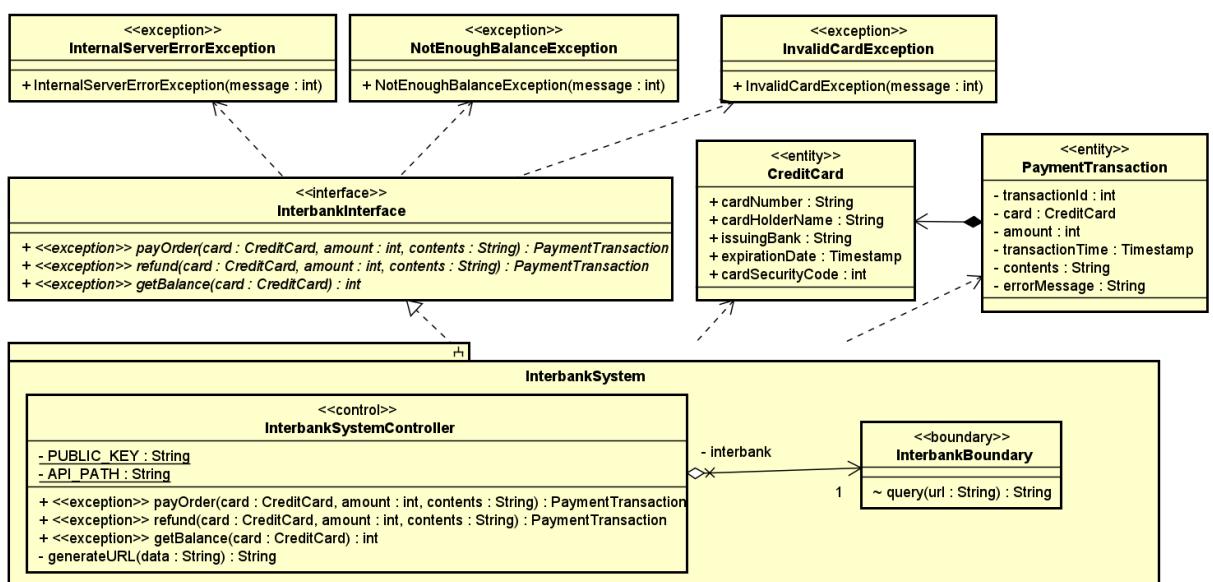
Document subsystem elements



Describe subsystem dependencies



Checkpoints



4.4. BÀI TẬP

Hãy thiết kế giao diện (Interface Design) chi tiết cho Use case “Place Rush Order”.

5. BÀI THỰC HÀNH SỐ 03 – THIẾT KẾ LỚP VÀ MÔ HÌNH HÓA DỮ LIỆU

5.1. MỤC ĐÍCH VÀ NỘI DUNG

Trong bài thực hành này, chúng ta sẽ làm quen với 02 bước còn lại trong thiết kế chi tiết (detailed design process), gồm có thiết kế lớp (class design) và mô hình hóa dữ liệu (data modeling). Kết thúc bài thực hành, người học có thể đạt được đủ kỹ năng để hoàn thiện Software Design Document (SDD).

5.2. CHUẨN BỊ

Người học cần tự hoàn thiện trước buổi học: bước thiết kế kiến trúc của phần mềm (Architectural Design) và thiết kế giao diện của bước đầu thiết kế chi tiết. Kết quả của bước thiết kế kiến trúc phần mềm sẽ là đầu vào cho bước thiết kế chi tiết. Đồng thời kết quả của thiết kế giao diện sẽ cần thiết để hoàn thiện thiết kế chi tiết.

5.3. NỘI DUNG CHI TIẾT

5.3.1. Thiết kế lớp (Class Design)

5.3.1.1. Bước đầu tạo các lớp thiết kế

Trong phần này, chúng ta sẽ ánh xạ các thành phần thiết kế (design elements, ví dụ: lớp – class, nhóm các lớp – group of classes, gói – package, subsystem) từ các lớp phân tích (analysis classes). Mỗi lớp thiết kế nên chỉ phục vụ tốt một mục đích duy nhất. Chúng ta sẽ xác định các lớp thiết kế dựa vào biểu đồ lớp kiến trúc và khuôn mẫu (stereotype) của lớp đấy. Lưu ý rằng chúng ta chưa ứng dụng các mẫu thiết kế (design patterns) trong bài thực hành này.

a) Thiết kế lớp boundary

Lớp boundary: Giao diện người dùng (User interface)

Trong Case study, chúng ta sử dụng JavaFX để xây dựng giao diện. Do đó, từ góc nhìn kiến trúc, mỗi lớp boundary giao diện người dùng tương đương với lớp thiết kế phụ trách xử lý sự kiện hoặc hành động của người dùng (được bắt bởi FXML tương ứng). Trong JavaFX, mặc dù các lớp thiết kế này thường được gọi là “controller” của các tệp FXML, chúng không thực sự đóng vai trò như lớp control trong UML. Do vậy, đa số các lớp xử lý sự kiện hiện tại đã khá đơn giản, và ánh xạ là 1-1.

Lớp boundary: Giao diện hệ thống (System/device boundary)

Trong bài thực hành trước, chúng ta đã “evolve”/ánh xạ lớp boundary cho liên ngân hàng Interbank thành một subsystem. Tuy nhiên, subsystem này thiết kế chưa được tốt: InterbankSubsystemController quá phức tạp, đồng thời một phần InterbankBoundary vẫn có thể tái sử dụng. Hiện tại trong mô tả của Case Study, chỉ có nhắc đến một hệ thống thông tin web bên ngoài (Interbank), nhưng thực tế có rất nhiều hệ thống như thế mà cần giao tiếp với REST APIs sử dụng các phương thức HTTP. Trong khi các hệ thống này có giao thức kết nối giống nhau, một trong số đó có thể là một hệ thống mà AIMS Software cần giao tiếp trong tương lai. Vì vậy, vì mục đích tái sử dụng, chúng ta có thể cần một lớp mới, ví dụ là API, để phụ trách giao tiếp API qua phương thức HTTP GET và HTTP POST. Ngoài ra, chúng ta sẽ nhận các vấn đề ở lớp control trong phần sau.

b) Thiết kế lớp entity

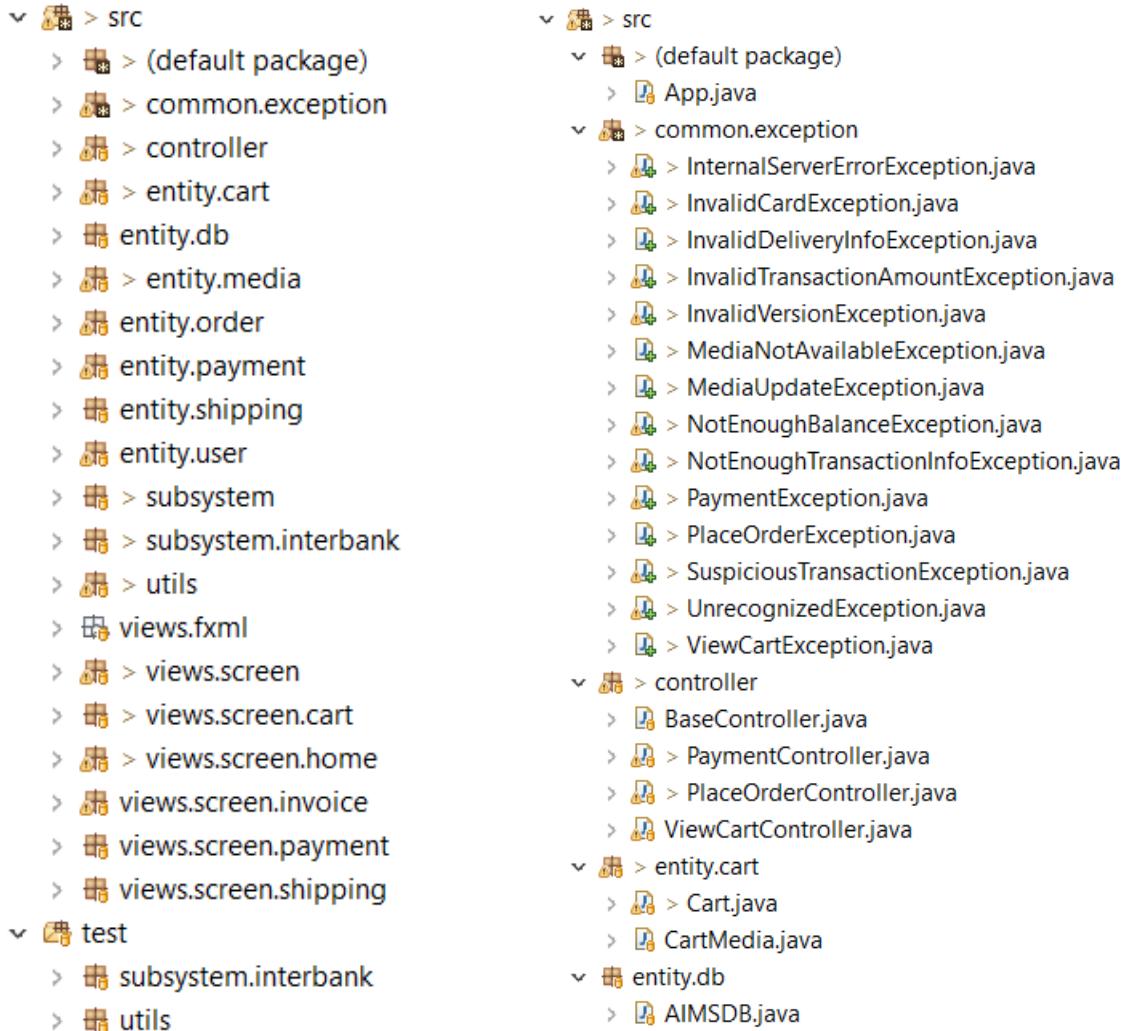
Với bản mô tả hiện tại và đối với 2 use case “Pay Order” và “Place Order”, đa số các lớp entity trong thiết kế kiến trúc đã đơn giản, và có thể ánh xạ 1-1 với lớp thiết kế.

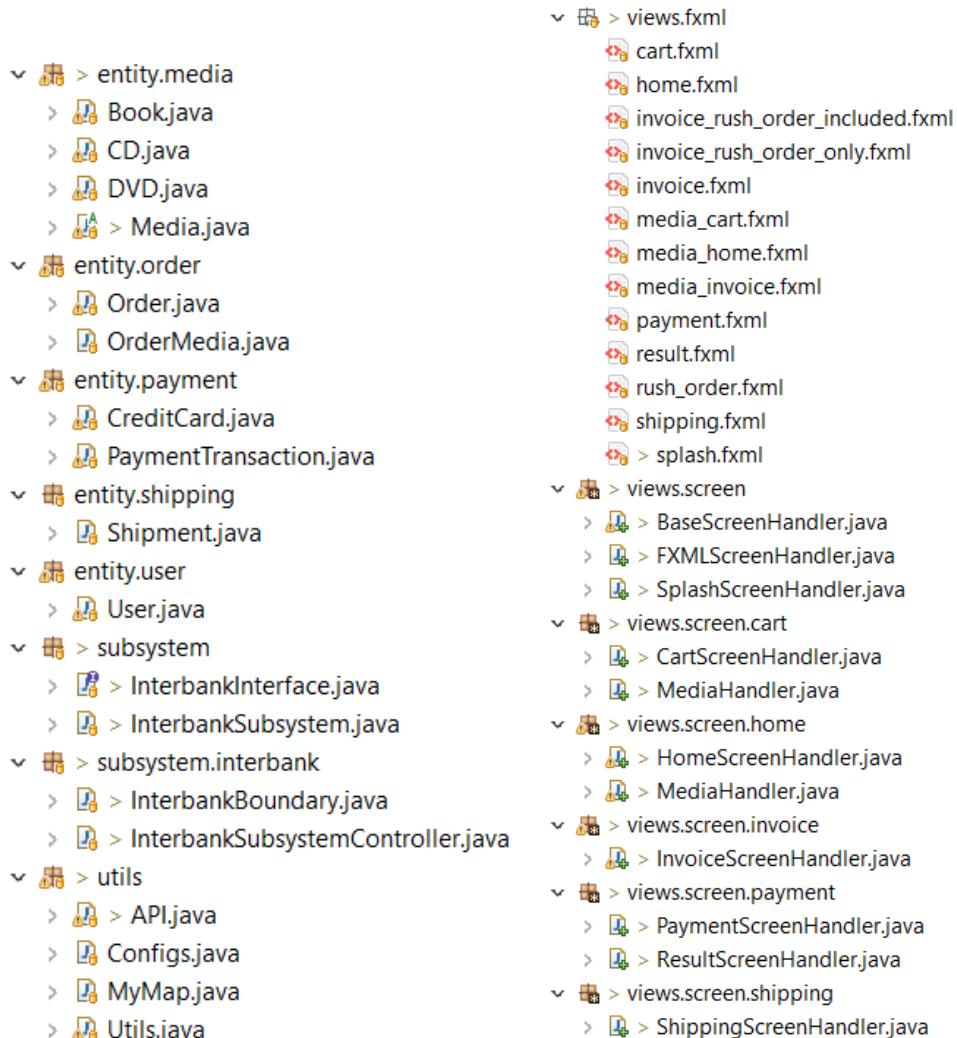
c) Thiết kế lớp control

Tương tự, đa số các lớp control trong thiết kế kiến trúc đã đơn giản, và có thể ánh xạ 1-1 với lớp thiết kế. Tuy nhiên, InterbankSubsystemController đang phụ trách 2 tác vụ: (1) điều khiển luồng dữ liệu và (2) chuyển đổi dữ liệu (chuyển đổi dữ liệu sang định dạng yêu cầu và xử lý kết quả trả về). Do đó chúng ta cần ít nhất một lớp khác để phụ trách chuyển đổi dữ liệu, ví dụ JSON hoặc MyMap (tùy thuộc vào thiết kế mà lớp này có thể tái sử dụng cho các hệ thống thông tin web khác).

d) Nhóm các lớp thiết kế

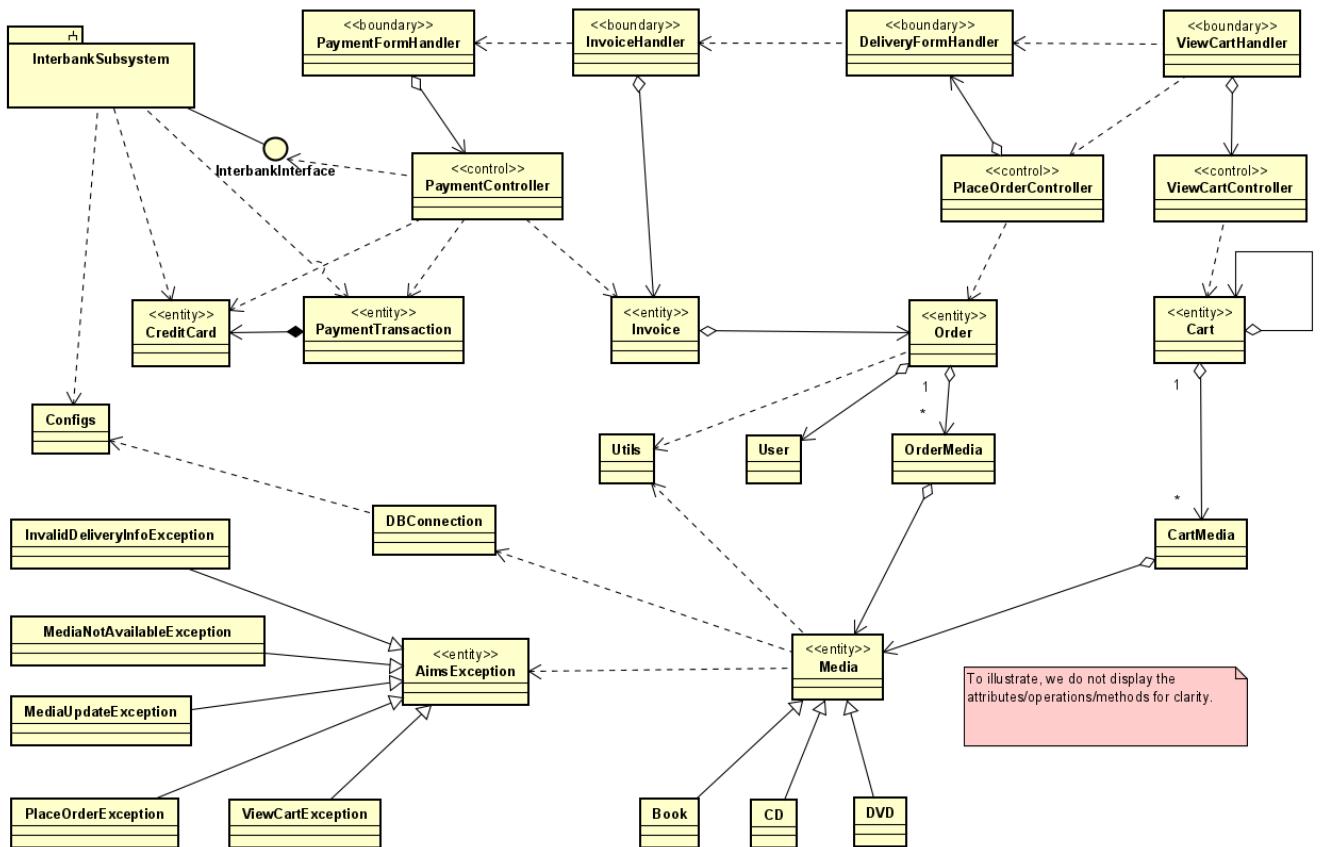
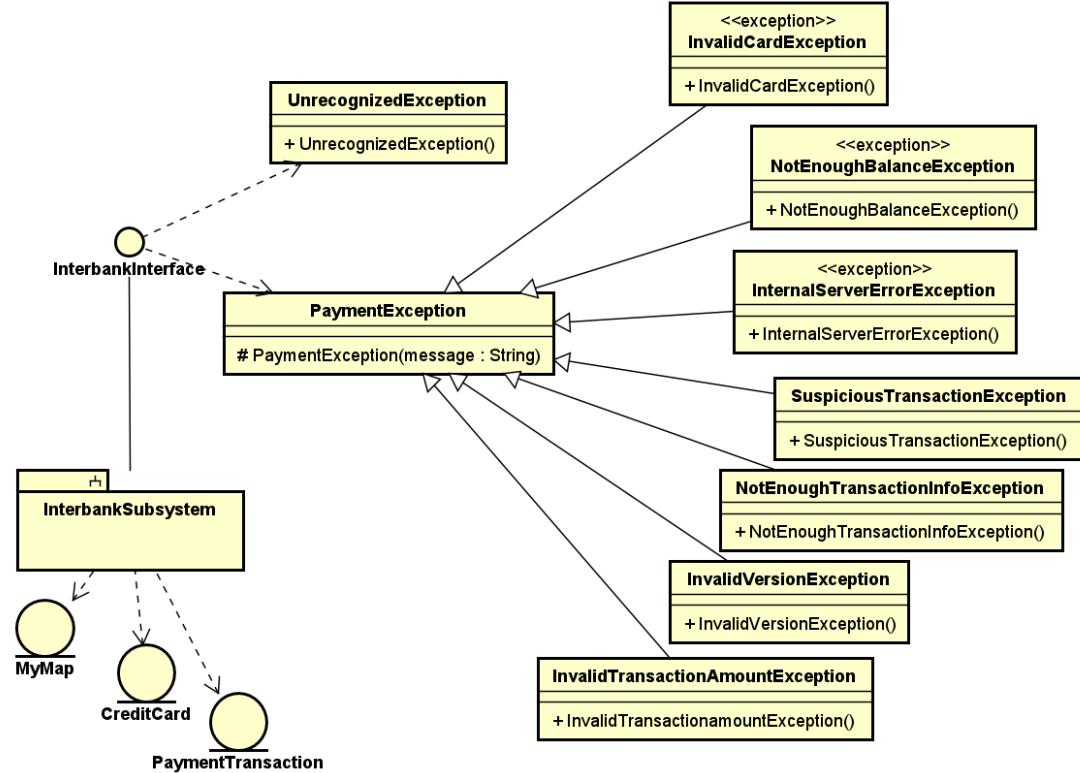
Sau đây là một cách để nhóm các lớp thiết kế vào các package.





5.3.1.2. Xác định mối quan hệ giữa các lớp

Để phục vụ mục đích minh họa mối quan hệ giữa các lớp được dễ nhìn và rõ ràng, các lớp ở các hình sau không bao gồm các attribute cũng như operation/method.



5.3.1.3. Lớp thiết kế

Trong phần này, các bước thiết kế lớp sẽ được minh họa theo từng bước.

a) Lớp “*InterbankInterface*”

<<interface>> InterbankInterface	
+ <<exception>> payOrder(card : CreditCard, amount : int, contents : String) : PaymentTransaction + <<exception>> refund(card : CreditCard, amount : int, contents : String) : PaymentTransaction	

Attribute

Không

Operation

#	Tên	Kiểu dữ liệu trả về	Mô tả (mục đích)
1	payOrder	PaymentTransaction	Thanh toán đơn hàng và trả về giao dịch thanh toán
2	refund	PaymentTransaction	Hoàn tiền và trả về giao dịch thanh toán

Parameter:

- card – thẻ tín dụng để giao dịch
- amount – số tiền giao dịch
- contents – nội dung giao dịch

Exception:

- PaymentException – nếu mã lỗi trả về đã biết
- UnrecognizedException – nếu không tìm thấy mã lỗi trả về hoặc có lỗi hệ thống

Method

Không

State

Không

Lưu ý khi lập trình, luôn cần chú thích các thành phần được công khai (public elements). Một ví dụ chú thích cho lớp này được thể hiện trong ảnh dưới đây.

```

1 package subsystem;
2
3@ import common.exception.PaymentException;
4 import common.exception.UnrecognizedException;
5 import entity.payment.CreditCard;
6 import entity.payment.PaymentTransaction;
7
8@ /**
9 * The {@code InterbankInterface} class is used to communicate with the
L0 * {@link subsystem.InterbankSubsystem InterbankSubsystem} to make transaction
L1 *
L2 * @author hieud
L3 *
L4 */
L5 public interface InterbankInterface {
L6
L7@ /**
L8 * Pay order, and then return the payment transaction
L9 *
L10 * @param card      - the credit card used for payment
L11 * @param amount    - the amount to pay
L12 * @param contents - the transaction contents
L13 * @return {@link entity.payment.PaymentTransaction PaymentTransaction} - if the
L14 *         payment is successful
L15 * @throws PaymentException      if responded with a pre-defined error code
L16 * @throws UnrecognizedException - if responded with an unknown error code or
L17 *         something goes wrong
L18 */
L19 public abstract PaymentTransaction payOrder(CreditCard card, int amount, String contents)
L20     throws PaymentException, UnrecognizedException;
L21
L22 /**
L23 * Refund, and then return the payment transaction
L24 *
L25 * @param card      - the credit card which would be refunded to
L26 * @param amount    - the amount to refund
L27 * @param contents - the transaction contents
L28 * @return {@link entity.payment.PaymentTransaction PaymentTransaction} - if the
L29 *         payment is successful
L30 * @throws PaymentException      if responded with a pre-defined error code
L31 * @throws UnrecognizedException - if responded with an unknown error code or
L32 *         something goes wrong
L33 */
L34 public abstract PaymentTransaction refund(CreditCard card, int amount, String contents)
L35     throws PaymentException, UnrecognizedException;
L36
L37 }
L38

```

Một số đường dẫn tham khảo:

<https://users.soe.ucsc.edu/~eaugusti/archive/102-winter16/misc/howToWriteJavaDocs.html>

<https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html>

b) Lớp “PaymentController”

<<control>> PaymentController	
- card : CreditCard	
- interbank : InterbankInterface	
+ payOrder(amount : int, contents : String, cardNumber : String, cardHolderName : String, expirationDate : String, securityCode : String) : Map<String, String>	
- getExpirationDate(date : String) : String	

Attribute

#	Tên	Kiểu dữ liệu trả về	Giá trị mặc định	Mô tả
1	card	CreditCard	NULL	Represent the card used for payment
2	interbank	InterbankInterface	NULL	Represent the Interbank subsystem

Operation

#	Tên	Kiểu dữ liệu trả về	Mô tả (mục đích)
1	payOrder	Map<String, String>	Thanh toán đơn hàng và trả về giao dịch thanh toán

Parameter:

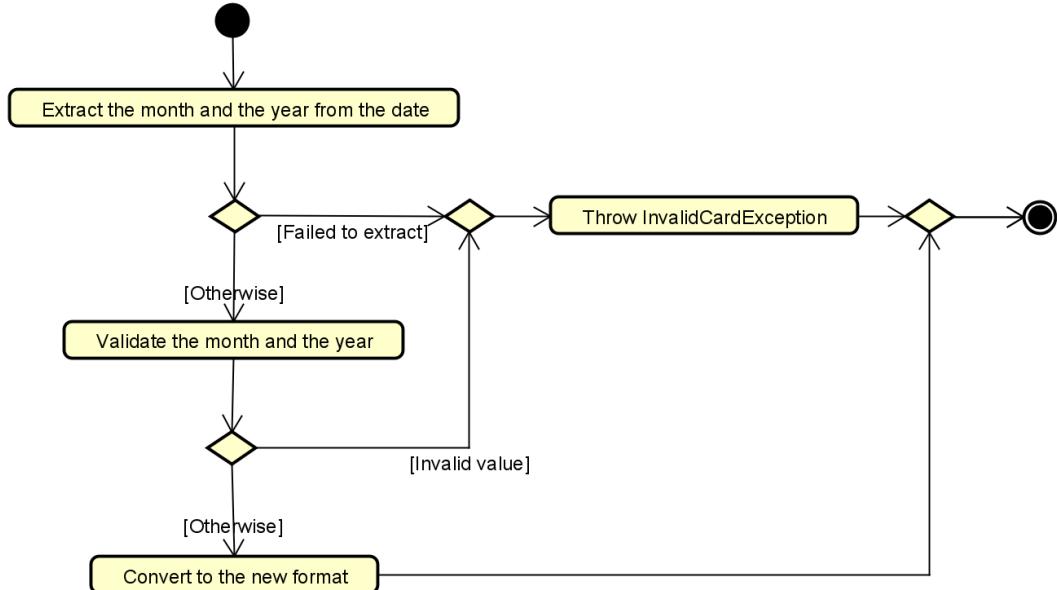
- amount – số tiền giao dịch
- contents – nội dung giao dịch
- cardNumber – số thẻ
- cardHolderName – tên chủ sở hữu
- expirationDate – ngày hết hạn theo định dạng "mm/yy"
- securityCode - mã bảo mật cvv/cvc

Exception:

- Không

Method

- getExpirationDate: Chuyển dữ liệu ngày từ định dạng "mm/yy" sang "mmyy".



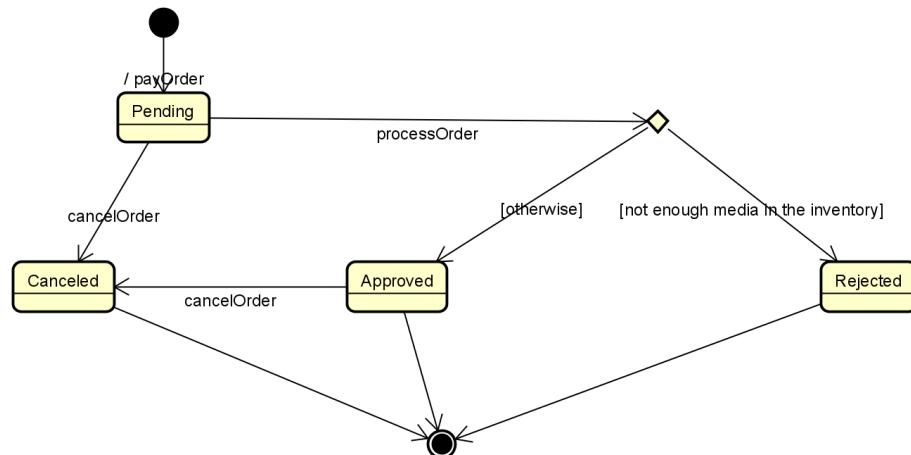
State

Không

c) Biểu đồ trạng thái (state machine) cho đối tượng “Order”

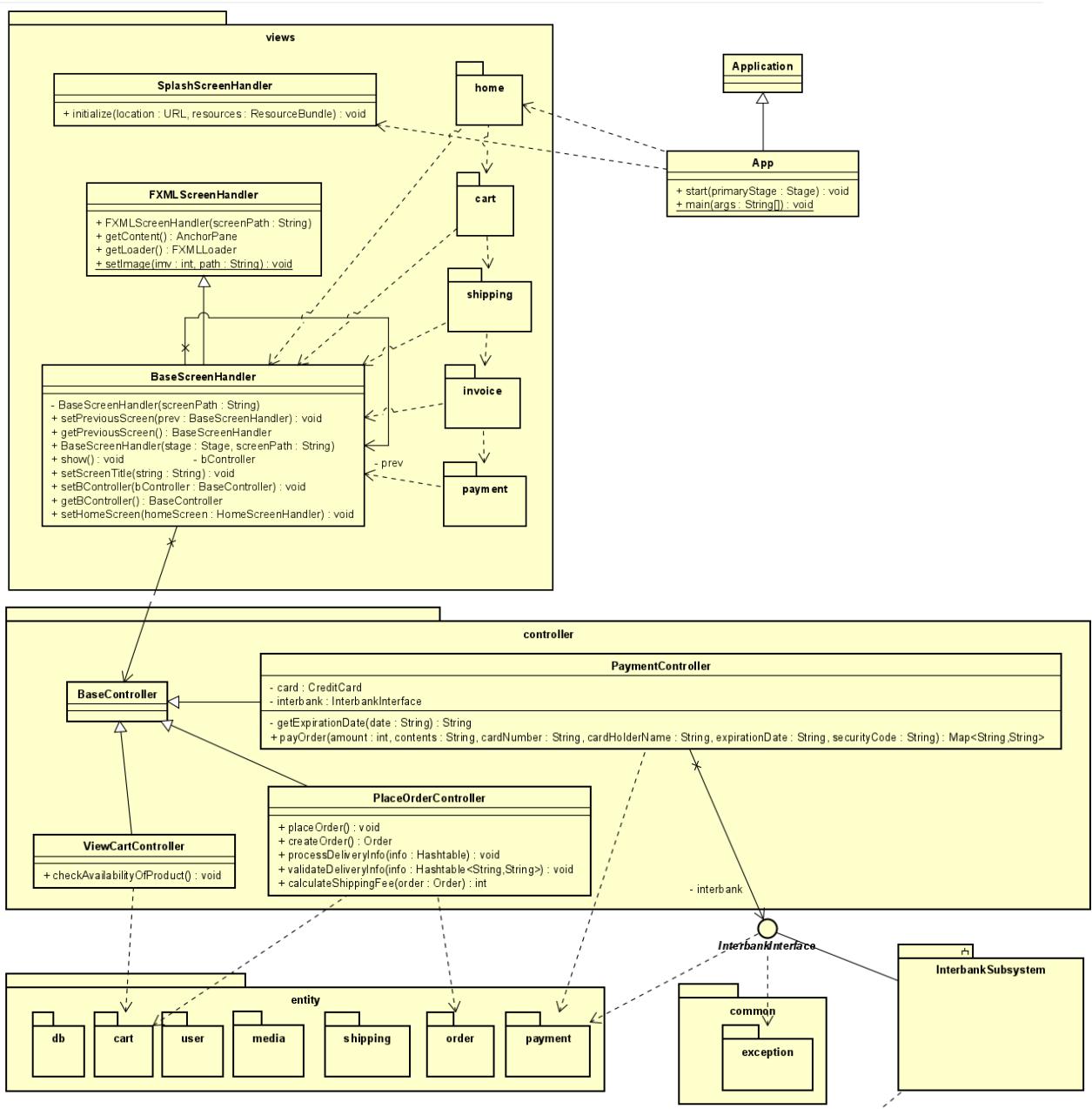
Chúng ta chỉ cần xác định các trạng thái được đặt tên (named states) của đối tượng, từ đó nắm được các trạng thái có thể có, và sự chuyển đổi giữa các trạng thái đó khi có các sự kiện tác động. Trước tiên, chúng ta tìm những lớp có các trạng thái được đặt tên. Sau đó, tìm các trạng thái có thể cho một đối tượng của lớp đó. Cuối cùng, mô hình hóa bằng biểu đồ trạng thái.

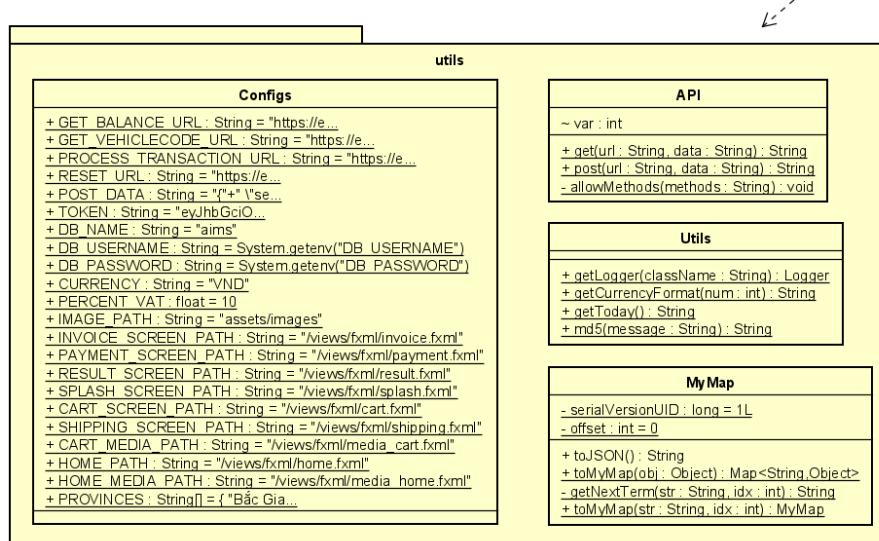
Dễ thấy, đối tượng “Order” là một trong những đối tượng có trạng thái được đặt tên. Dưới đây là một ví dụ minh họa biểu đồ trạng thái của đối tượng từ sau khi khách hàng đặt hàng cho tới khi một quản trị viên xử lý đơn hàng.



5.3.1.4. Biểu đồ lớp thiết kế

Cuối cùng, đặt tất cả các lớp thiết kế vào một biểu đồ lớp thiết kế tổng quan. Lưu ý không thể hiện chi tiết của subsystem trong biểu đồ này. Ngoài ra, nếu có quá nhiều chi tiết trong biểu đồ, có thể làm tương tự với các package và tạo biểu đồ lớp thiết kế cho từng package.



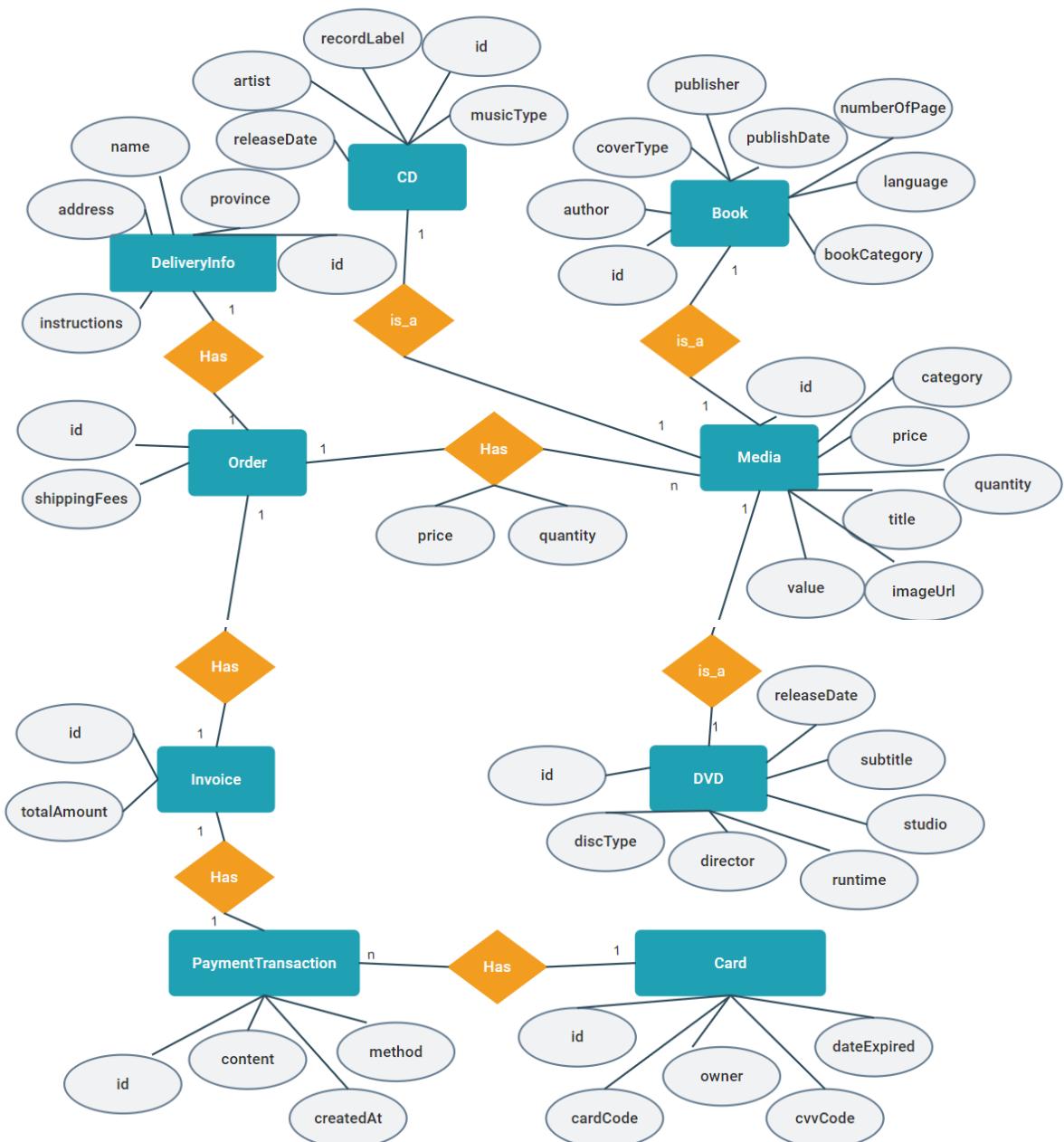


5.3.2. Mô hình hóa dữ liệu (Data Modeling)

5.3.2.1. Mô hình khái niệm (Conceptual Data Model)

Mô hình khái niệm là một mô hình dữ liệu có tính trừu tượng cao, trừu tượng hóa các biểu hiện tự nhiên (natural expression) mà không phụ thuộc vào bất kỳ hệ thống quản trị dữ liệu (database management system – DBMS) nào như PostgreSQL, SQLite, Microsoft Access, hoặc MongoDB. Một mô hình khái niệm có thể được thể hiện bởi biểu đồ thực thể-liên kết (Entity-Relationship diagram – ER diagram). Sau đây là một ví dụ cho AIMS Software.

AIMS System ERD



5.3.2.2. Thiết kế cơ sở dữ liệu

Trong phần này, ta cần phải xác định sự lựa chọn hệ thống quản trị dữ liệu (DBMS) và mô tả DBMS. Trong khuôn khổ của bài thực hành này, SQLite được chọn làm DBMS cho Case Study. Một số ưu điểm chính của SQLite:

- SQLite là một DBMS mã nguồn mở

- SQLite không chỉ nhẹ, nhanh, phổ biến, không cần cài đặt, có độ tin cậy cao, đầy đủ tính năng, và là một engine cơ sở dữ liệu SQL; ngoài ra còn ổn định, không phụ thuộc vào nền tảng, tương thích ngược và hỗ trợ lâu dài.
- Có thể kết nối với Java (<https://github.com/xerial/sqlite-jdbc>)

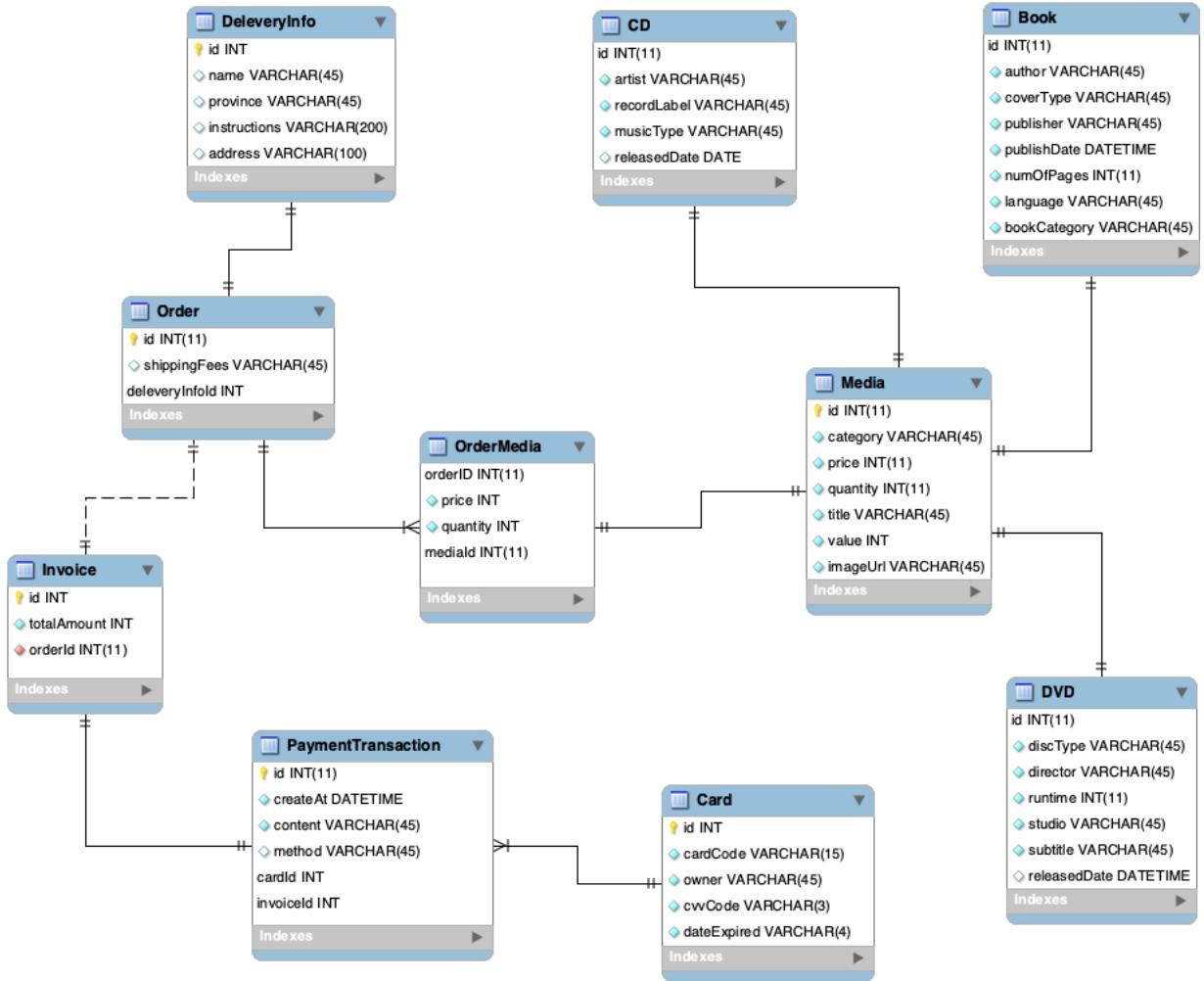
SQLite là một cơ sở dữ liệu local, thường được dùng để lưu trữ dữ liệu của người dùng trên chính thiết bị của người dùng. Vì vậy, SQLite không thể được cài đặt trên máy chủ để dùng chung và chia sẻ cho nhiều người dùng. Trong một hệ thống website giới thiệu và bán hàng như AIMS, khách hàng cần tiếp cận với các thông tin mới nhất của các mặt hàng, về số lượng còn剩下 của các mặt hàng... Các thông tin này cần được chia sẻ chung và cập nhật thường xuyên cho tất cả các khách hàng.

Vì vậy, thực ra SQLite hoàn toàn không phù hợp trong thực tế để sử dụng cho các hệ thống bán hàng như AIMS. Hệ quản trị CSDL có thể cài đặt tại server như MySQL hay PostgreSQL là phương án được lựa chọn để sử dụng trong thực tế cho những hệ thống như thế này.

Tuy nhiên, trong khuôn khổ bài thực hành, vì tính đơn giản, thuận tiện trong việc cài đặt và sử dụng, SQLite được lựa chọn để giúp mỗi sinh viên có riêng một cơ sở dữ liệu trên máy tính cá nhân của mình, có thể minh họa được các tính năng từ thiết kế đến các thao tác với cơ sở dữ liệu này. Sinh viên cần lưu ý rằng, SQLite chỉ được dùng để minh họa và cho sinh viên thực hành phần thiết kế, kết nối và thao tác với CSDL, chứ hoàn toàn không có ý nghĩa về tính ứng dụng trong thực tế với các hệ thống như AIMS.

a) Mô hình dữ liệu logic (Logical data model)

Từ mô hình khái niệm (được thể hiện bởi biểu đồ ER) trong phần trước, ta có thể thiết kế mô hình dữ liệu logic tương thích với DBMS đã chọn (SQLite). Sau đây là mô hình dữ liệu logic ví dụ.



b) Mô hình dữ liệu vật lý (Physical data model)

Trong phần này, chúng ta sẽ thiết kế chi tiết cho từng phần tử trong biểu đồ trên. Ví dụ, trong biểu đồ cơ sở dữ liệu quan hệ (RDBMS), ta thiết kế chi tiết cho dùng bảng và ràng buộc.

Chú thích:

PK: Primary Key

FK: Foreign Key

▪ Media

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	x		id	Integer	Yes	ID, auto increment
2.			category	VARCHAR(45)	Yes	Media type, e.g., CD, DVD

#	PK	FK	Column Name	Data type	Mandatory	Description
3.			price	Integer	Yes	Current price
4.			quantity	Integer	Yes	Number of products
5.			title	VARCHAR(45)	Yes	Product name
6.			value	Integer	Yes	Value of the product
7.			imageUrl	VARCHAR(45)	Yes	Product image path

▪ CD

#	PK	FK	Column Name	Data type	Mandatory	Description
1.		x	id	Integer	Yes	ID, same as ID of Media of which type is CD
2.			artist	VARCHAR(45)	Yes	Artist's name
3.			recordLabel	VARCHAR(45)	Yes	Record label
4.			musicType	VARCHAR(45)	Yes	Music genres
5.			releasedDate	DATE	No	Release date

▪ Book

#	PK	FK	Column Name	Data type	Mandatory	Description
1.		x	id	Integer	Yes	ID, same as ID of Media of which type is Book
2.			author	VARCHAR(45)	Yes	Author
3.			coverType	VARCHAR(45)	Yes	Cover type
4.			Publisher	VARCHAR(45)	Yes	Publishing house
5.			publishDate	DATETIME	Yes	Date of publishing
6.			numOfPages	Integer	Yes	Page number

#	PK	FK	Column Name	Data type	Mandatory	Description
7.			language	VARCHAR(45)	Yes	Language
8.			bookCategory	VARCHAR(45)	Yes	Book category

▪ DVD

#	PK	FK	Column Name	Data type	Mandatory	Description
1.		x	id	Integer	Yes	ID, same as ID of Media of which type is DVD
2.			discType	VARCHAR(45)	Yes	Disc type
3.			director	VARCHAR(45)	Yes	Director
4.			runtime	Integer	Yes	Duration
5.			studio	VARCHAR(45)	Yes	Manufacturer
6.			subtitle	VARCHAR(45)	Yes	Subtitles
7.			releasedDate	DATETIME	Yes	Release date
8.			filmType	VARCHAR(45)	Yes	Genres

▪ Card

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	x		id	Integer	Yes	ID, auto increment
2.			cardCode	VARCHAR(45)	Yes	Card code
3.			owner	VARCHAR(45)	Yes	Cardholders
4.			cvvCode	VARCHAR(3)	Yes	CVV code
5.			dateExpired	VARCHAR(4)	Yes	Expiration date

▪ DeliveryInfo

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	X		id	Integer	Yes	ID, auto increment
2.			name	VARCHAR(45)	Yes	Receiver name
3.			province	VARCHAR(45)	Yes	Provinces
4.			instructions	VARCHAR(200)	No	Delivery instructions
5.			address	VARCHAR(100)	Yes	Delivery address

▪ Order

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	X		id	Integer	Yes	ID
2.			shippingFees	VARCHAR(45)	Yes	Shipping fee
3.		X	deliveryInfoId	Integer	Yes	Delivery Info ID

▪ OrderMedia

#	PK	FK	Column Name	Data type	Mandatory	Description
1.		X	mediaID	Integer	Yes	Media ID
2.		X	orderID	Integer	Yes	Order ID
3.			price	Integer	Yes	Selling price
4.			quantity	Integer	Yes	Number

▪ Invoice

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	x		id	Integer	Yes	ID
2.			totalAmount	Integer	Yes	Total
3.		x	orderId	Integer	Yes	Order ID

▪ PaymentTransaction

#	PK	FK	Column Name	Data type	Mandatory	Description
1.	x		id	Integer	Yes	ID
2.			createAt	DATETIME	Yes	Date of creation
3.			content	VARCHAR(45)	Yes	Transaction contents
4.			method	VARCHAR(45)	Yes	Payment methods
5.		x	cardId	Integer	Yes	ID of used card
6.		x	invoiceId	Integer	Yes	Invoice ID

Cuối cùng, ta cần có database script. Với những công cụ thiết kế database chuyên nghiệp và các plugins, ta có thể tự động tạo database script trực tiếp từ mô hình dữ liệu logic.

```
BEGIN;
CREATE TABLE "aims"."Media"(
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "category" VARCHAR(45) NOT NULL,
    "price" INTEGER NOT NULL,
    "quantity" INTEGER NOT NULL,
    "title" VARCHAR(45) NOT NULL,
    "value" INTEGER NOT NULL,
    "imageUrl" VARCHAR(45) NOT NULL
);
```

```

CREATE TABLE "aims"."CD"(
    "id" INTEGER PRIMARY KEY NOT NULL,
    "artist" VARCHAR(45) NOT NULL,
    "recordLabel" VARCHAR(45) NOT NULL,
    "musicType" VARCHAR(45) NOT NULL,
    "releasedDate" DATE,
    CONSTRAINT "fk_CD_Media1"
        FOREIGN KEY("id")
        REFERENCES "Media"("id")
);
CREATE TABLE "aims"."Book"(
    "id" INTEGER PRIMARY KEY NOT NULL,
    "author" VARCHAR(45) NOT NULL,
    "coverType" VARCHAR(45) NOT NULL,
    "publisher" VARCHAR(45) NOT NULL,
    "publishDate" DATETIME NOT NULL,
    "numOfPages" INTEGER NOT NULL,
    "language" VARCHAR(45) NOT NULL,
    "bookCategory" VARCHAR(45) NOT NULL,
    CONSTRAINT "fk_Book_Media1"
        FOREIGN KEY("id")
        REFERENCES "Media"("id")
);
CREATE TABLE "aims"."DeleteeveryInfo"(
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "name" VARCHAR(45),
    "province" VARCHAR(45),
    "instructions" VARCHAR(200),
    "address" VARCHAR(100)
);
CREATE TABLE "aims"."Card"(
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "cardCode" VARCHAR(15) NOT NULL,
    "owner" VARCHAR(45) NOT NULL,
    "cvvCode" VARCHAR(3) NOT NULL,
    "dateExpired" VARCHAR(4) NOT NULL
);
CREATE TABLE "aims"."DVD"(
    "id" INTEGER PRIMARY KEY NOT NULL,
    "discType" VARCHAR(45) NOT NULL,
    "director" VARCHAR(45) NOT NULL,
    "runtime" INTEGER NOT NULL,
    "studio" VARCHAR(45) NOT NULL,

```

```

"subtitle" VARCHAR(45) NOT NULL,
"releasedDate" DATETIME,
CONSTRAINT "fk_DVD_Media1"
    FOREIGN KEY("id")
    REFERENCES "Media"("id")
);
CREATE TABLE "aims"."Order"(

"id" INTEGER NOT NULL,
"shippingFees" VARCHAR(45),
"deleveryInfoId" INTEGER NOT NULL,
PRIMARY KEY("id","deleveryInfoId"),
CONSTRAINT "fk_Order_DeleveryInfo1"
    FOREIGN KEY("deleveryInfoId")
    REFERENCES "DeleveryInfo"("id")
);
CREATE INDEX "aims"."Order.fk_Order_DeleveryInfo1_idx" ON "Order"
("deleveryInfoId");
CREATE TABLE "aims"."OrderMedia"(

"orderID" INTEGER NOT NULL,
"price" INTEGER NOT NULL,
"quantity" INTEGER NOT NULL,
"mediaId" INTEGER NOT NULL,
PRIMARY KEY("orderID","mediaId"),
CONSTRAINT "fk_ordermedia_order"
    FOREIGN KEY("orderID")
    REFERENCES "Order"("id"),
CONSTRAINT "fk_OrderMedia_Media1"
    FOREIGN KEY("mediaId")
    REFERENCES "Media"("id")
);
CREATE INDEX "aims"."OrderMedia.fk_ordermedia_order_idx" ON "OrderMedia"
("orderID");
CREATE INDEX "aims"."OrderMedia.fk_OrderMedia_Media1_idx" ON "OrderMedia"
("mediaId");
CREATE TABLE "aims"."Invoice"(

"id" INTEGER PRIMARY KEY NOT NULL,
"totalAmount" INTEGER NOT NULL,
"orderId" INTEGER NOT NULL,
CONSTRAINT "fk_Invoice_Order1"
    FOREIGN KEY("orderId")
    REFERENCES "Order"("id")
);

```

```

CREATE INDEX "aims"."Invoice.fk_Invoice_Order1_idx" ON "Invoice"
("orderId");
CREATE TABLE "aims"."PaymentTransaction"(
    "id" INTEGER NOT NULL,
    "createAt" DATETIME NOT NULL,
    "content" VARCHAR(45) NOT NULL,
    "method" VARCHAR(45),
    "cardId" INTEGER NOT NULL,
    "invoiceId" INTEGER NOT NULL,
    PRIMARY KEY("id","cardId","invoiceId"),
    CONSTRAINT "fk_PaymentTransaction_Card1"
        FOREIGN KEY("cardId")
        REFERENCES "Card"("id"),
    CONSTRAINT "fk_PaymentTransaction_Invoice1"
        FOREIGN KEY("invoiceId")
        REFERENCES "Invoice"("id")
);
CREATE INDEX "aims"."PaymentTransaction.fk_PaymentTransaction_Card1_idx" ON
"PaymentTransaction" ("cardId");
CREATE INDEX "aims"."PaymentTransaction.fk_PaymentTransaction_Invoice1_idx"
ON "PaymentTransaction" ("invoiceId");
COMMIT;

```

5.4. BÀI TẬP

Hãy thiết kế lớp chi tiết và mô hình hóa dữ liệu cho Use case “Place Rush Order”.

6. BÀI THỰC HÀNH SỐ 04 – LẬP TRÌNH VÀ KIỂM THỬ ĐƠN VỊ

6.1. MỤC ĐÍCH VÀ NỘI DUNG

- Trước tiên, người học được hướng dẫn thực hành sử dụng Eclipse IDE, JUnit Framework và thực hành thiết kế các Testcases. Sau đó, người học sẽ được học cách tạo và sinh Javadoc tự động với Eclipse
- Trong phần Unit test, này người học sẽ được hướng dẫn về:
 - Các bước thiết kế TestCase cho UnitTest
 - Tạo TestCase cho các class bằng Eclipse và Junit
 - Tiếp cận phương pháp Test Driven Development (TDD) trong quá trình xây dựng phần mềm

- Tiếp theo người học sẽ được học về cách refactoring code. Người học sẽ được hướng dẫn nhận diện một class hoặc method cần được refactoring và sau đó sẽ thực hành refactoring một class
- Sau bài học người học sẽ có thể áp dụng những kiến thức mình đã được học vào bài tập về nhà cũng như Capstone Project

6.2. CHUẨN BỊ

- Người học cần cài đặt sẵn Eclipse và môi trường Java11 trên máy cá nhân
- Clone project từ: <https://github.com/leminhnguyen/AIMS-Student>

6.3. NỘI DUNG CHI TIẾT

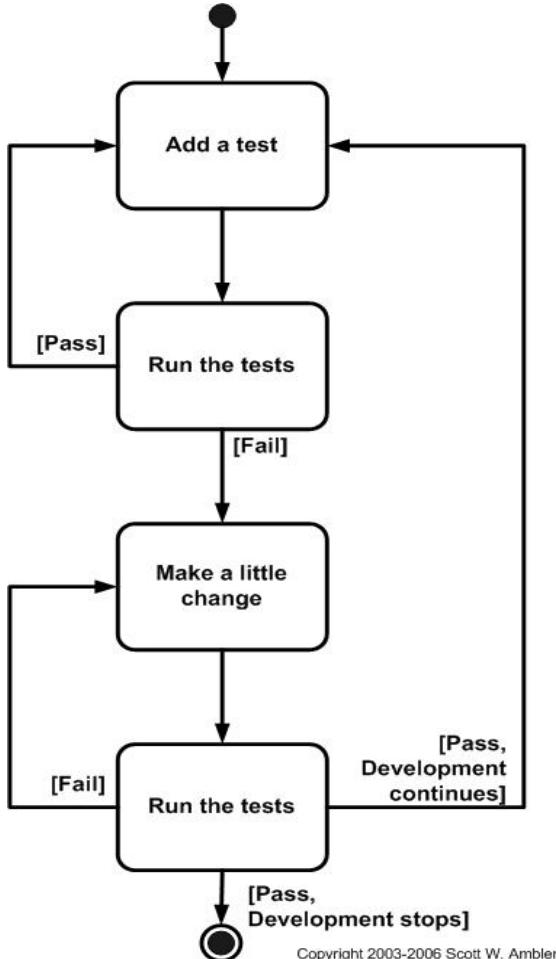
6.3.1. Kiểm thử đơn vị

6.3.1.1. *Bắt đầu với Kiểm thử đơn vị*

- Test là một phần nằm trong phần mềm phát triển, thực thi các phần khác nhau của phần mềm và xác nhận kết quả có đúng như mong đợi (expected result) hoặc thực thi đúng theo luồng các sự kiện mong muốn (behavior testing)
- Unit Test là các đoạn code hoặc chương trình được viết bởi lập trình viên nhằm kiểm tra hành vi hoặc trạng thái của một chức năng cụ thể trong PM
- Unit Test nhằm tới một đơn vị nhỏ của phần mềm như method hoặc class. Những phụ thuộc bên ngoài nên loại bỏ ra khỏi unit test, thay vào đó sử dụng mock được tạo ra bởi các test framework
- Unit Test không phù hợp cho việc testing cho những thành phần có giao diện người dùng phức tạp hoặc có sự giao tiếp giữa các thành phần khác nhau trong chương trình.

6.3.1.2. *Làm quen với TDD (test driven development)*

- TDD là quá trình phát triển phần mềm dựa trên việc các yêu cầu chức năng của phần mềm được chuyển thành các testcase trước khi phần mềm được phát triển hoàn thiện và theo dõi quá trình phát triển phần mềm bằng cách kiểm thử phần mềm với các testcase đó. Hay nói một cách đơn giản là chúng ta sẽ tiến hành viết test trước khi code và điều này trái ngược với việc một phần mềm được hoàn thiện rồi mới bắt đầu viết test



- Các bước thực hiện TDD
 - + **B1 - Add a Test:** bước đầu tiên trước khi bắt đầu phát triển một tính năng mới sẽ là viết test cho chức năng cần test (thông thường test ban đầu sẽ fail do có thể class cần Test chưa được viết)
 - + **B2 - Run the Tests:** chạy các đoạn test đã viết
 - + **B3 - Make A Little Change:** tiến hành viết code hoặc cập nhật chức năng để có thể vượt qua các Tests
 - + **B4 - Run the Tests:** thực hiện chạy lại các test, nếu như failed thì ta cần quay lại cập nhật code và chạy lại test cho đến khi pass. Sau khi vượt qua các test thì ta bắt đầu lặp lại quá trình từ đầu

6.3.1.3. *Làm quen với JUNIT5*

- Junit5 là phiên bản thứ 5 của Junit, một framework sử dụng annotation để nhận diện ra các phương thức test và Junit là một phần mềm mã nguồn mở
- Các phiên bản của Junit đi kèm theo với Eclipse, vì vậy chúng ta không cần phải cài đặt thủ công

- Chi tiết về bước add Junit5 vào trong Eclipse sẽ được trình bày chi tiết ở bước tiếp theo
- Junit nhận diện các phương thức cần test bằng các annotation (bắt đầu bằng @), một vài annotation được sử dụng phổ biến trong Junit5
 - + @Test: Biểu thị một phương thức test (Test Method)
 - + @DisplayName: Khai báo tên cho Test Class hoặc Test Method
 - + @BeforeEach: phương thức được thực thi trước khi bắt đầu mỗi Test Method
 - + @AfterEach: phương thức được thực thi sau khi chạy xong mỗi Test Method
 - + @BeforeAll: phương thức được thực thi trước khi tất cả các Test Method được thực hiện (VD: connect tới DB,...)
 - + @AfterAll: phương thức được thực thi sau khi tất cả các Test Method được thực hiện (VD: đóng connection tới DB,...)
- Để tìm hiểu chi tiết hơn và cụ thể hơn về Junit5, người học có thể tham khảo theo các link sau
 - + <https://www.journaldev.com/20834/junit5-tutorial>
 - + <https://junit.org/junit5/docs/current/user-guide/>

6.3.1.4. Thực hành thiết kế unit tests theo hướng TDD

a) Đặc tả yêu cầu

- Trong phần này chúng ta sẽ thực hành việc thiết kế các testcases cho phương thức *validateDeliveryInfo* nằm trong controller *PlaceOrder*
- Input đầu vào của phương thức *validateDeliveryInfo* sẽ là thông tin người dùng nhập vào như: *name, phone, address*
- Trong đó:

STT	Tên tham số	Yêu cầu
1	name	chỉ bao gồm chữ cái, không chứa ký tự đặc biệt, không được phép null
2	phone	chỉ bao gồm chữ số, độ dài 10 ký tự và bắt đầu là số 0
3	address	không được phép null, không chứa ký tự đặc biệt

- Công việc chúng ta cần phải thực hiện là thiết kế unit tests dựa trên đặc tả và xây dựng phương thức *validateDeliveryInfo* theo quá trình TDD

- Chúng ta có thể tách nhỏ *validateDeliveryInfo* thành 3 phương thức nhỏ hơn là *validateAddress*, *validateName* và *validatePhoneNumber* (ban đầu các phương thức này đều empty bởi vì chúng ta cần xây dựng testcase trước khi bắt đầu implement thực sự)

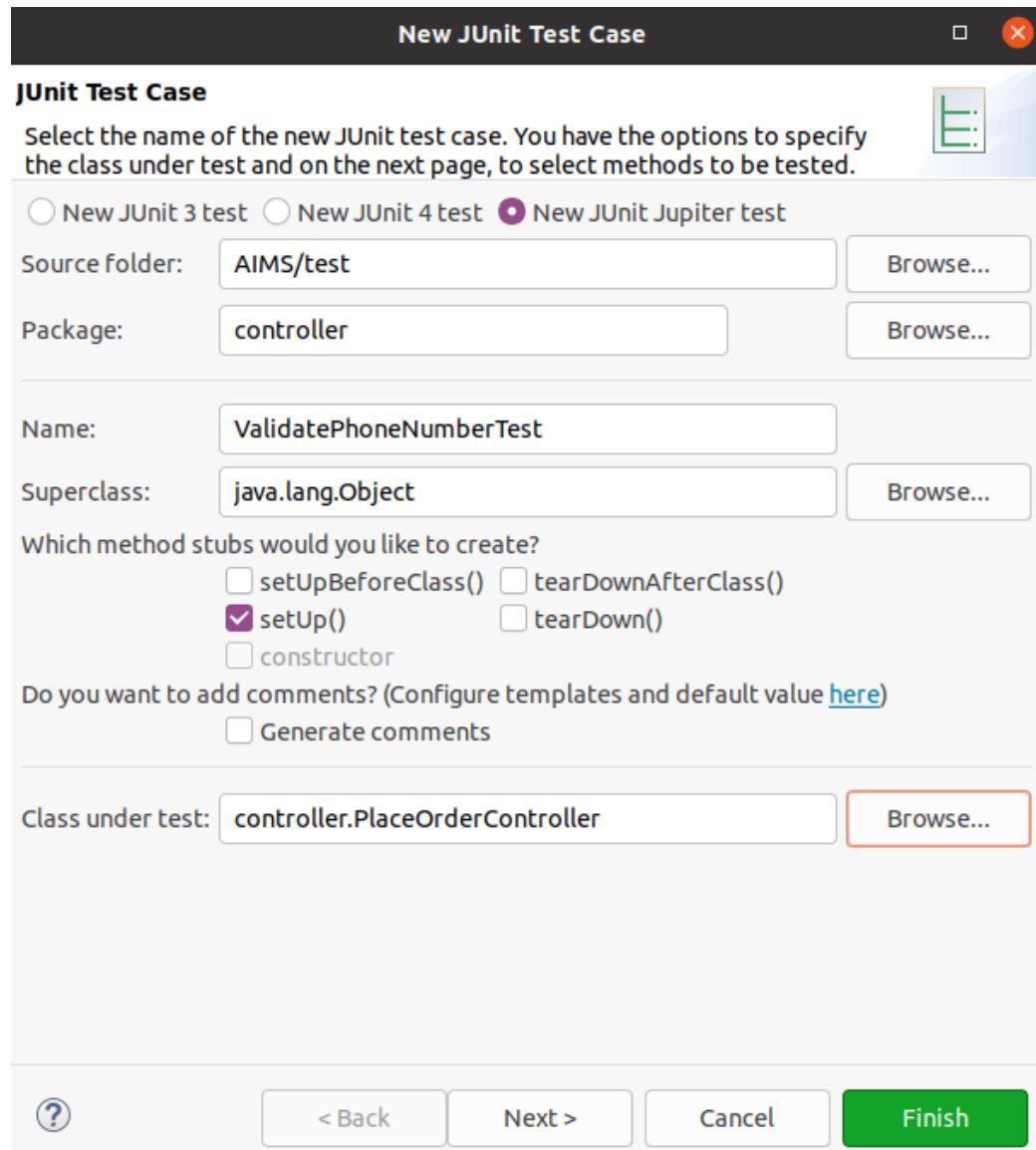
```

Java - AIMS/src/controller/PlaceOrderController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer  App.java AIMSDB.java PlaceOrderController.java BaseController.java CartScreenHandler.java PaymentCo
AIMS [AIMS-Student master 11]
src
controller
PlaceOrderController.java
entity
subsystem
utils
views
JRE System Library [java-11-openjdk-amd64]
JUnit 5
javafx
Referenced Libraries
assets
lib
test
66     * @throws InterruptedException
67     * @throws IOException
68     */
69     public void processDeliveryInfo(HashMap<String, String> info) throws InterruptedException, IOException{
70         LOGGER.info("Process Delivery Info");
71         LOGGER.info(info.toString());
72         validateDeliveryInfo(info);
73     }
74     /**
75      * The method validates the info
76      * @param info
77      * @throws InterruptedException
78      * @throws IOException
79      */
80     public void validateDeliveryInfo(HashMap<String, String> info) throws InterruptedException, IOException{
81     }
82     public boolean validatePhoneNumber(String phoneNumber) {
83         return false;
84     }
85     public boolean validateName(String name) {
86         return false;
87     }
88     public boolean validateAddress(String address) {
89         return false;
90     }
91 }
92
93
94
95

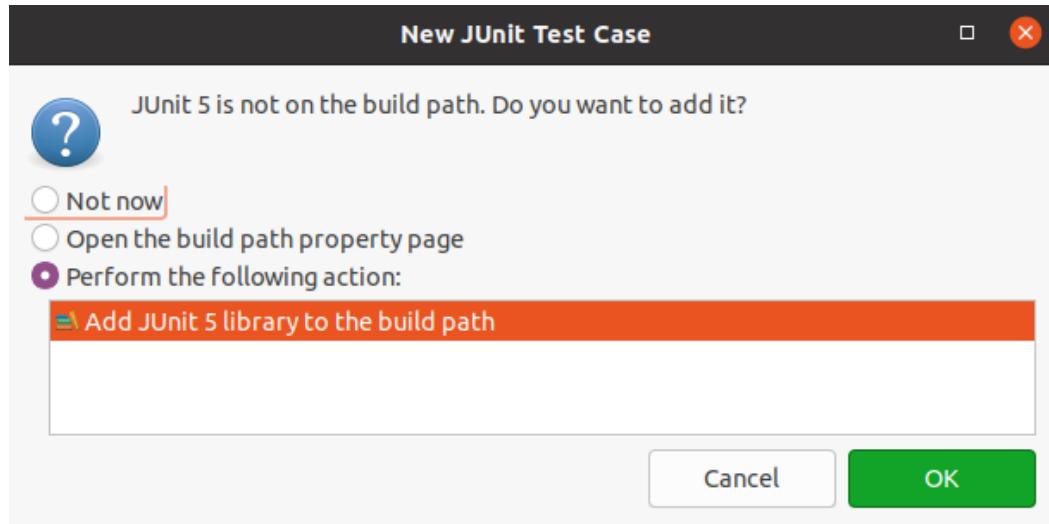
```

b) Tạo UnitTest bằng Eclipse

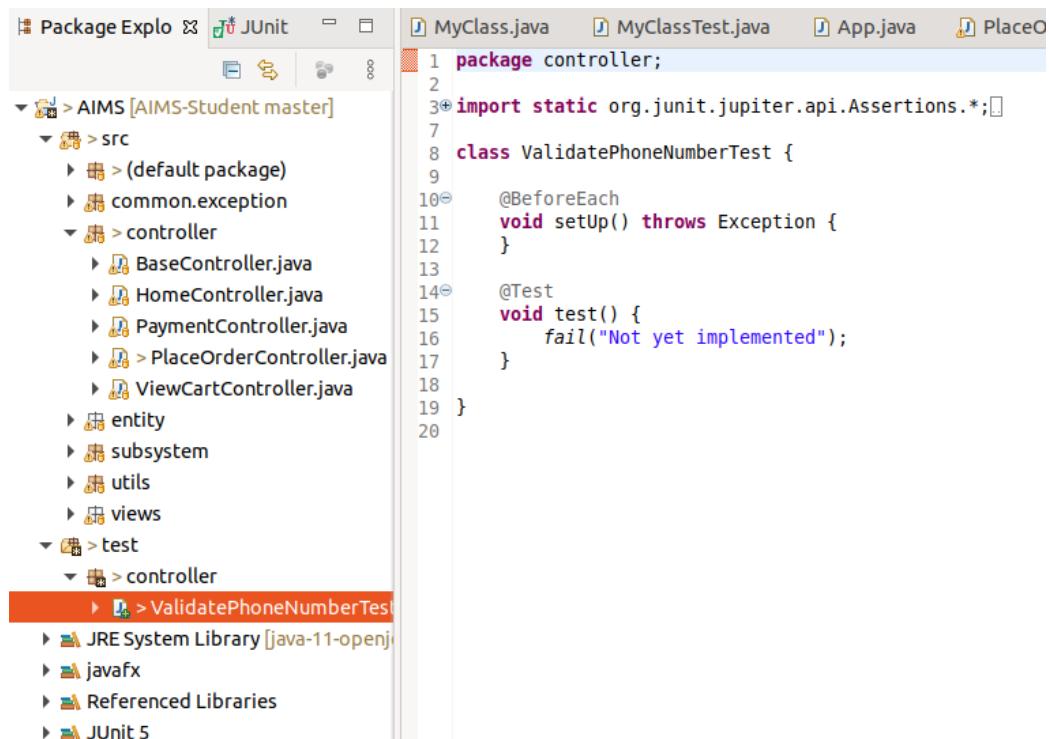
- Tạo một package tên controller ở trong folder test, sau này khi tiến hành tạo các testcase cho các class khác thì chúng ta nên tuân theo cấu trúc của class đó ở trong src folder
- Tiếp theo chúng ta sẽ tiến hành xây dựng các class Test cho các phương thức này bằng Junit5. Click chuột phải vào Project -> New -> Junit Testcase. Sau đó điền các thông tin cần thiết như:
 - Source folder:** là nơi các class Test được tạo ra, chúng ta sẽ đặt các testcase ở trong folder test
 - Package:** là package ở trong folder test mà chúng ta muốn đặt các test case
 - Name:** là tên của class Test
 - Class Under Test:** là tên class mà chúng ta đang cần test



- Sau đó click Finish, nếu như project chưa có Junit thì Eclipse sẽ hiện lên popup yêu cầu add Junit -> Click ok



- Sau khi hoàn thiện xong bước ở trên ta sẽ có kết quả như sau



- Đến đây chúng ta đã học được cách tạo thành công testcase cho một class, và nếu như bấm run thì tất nhiên là kết quả sẽ failed bởi vì chúng ta chưa implement bất kỳ phần code cần test nào

c) Thực hành xây dựng các phương thức theo TDD

- Bây giờ chúng ta sẽ bắt tay vào quá trình TDD, nếu như bây giờ chúng ta khởi tạo một đối tượng của PlaceOrderController ở trong class ValidatePhoneNumber và test phương thức validatePhoneNumber với một

số điện thoại hợp lệ (VD: 0123456789) thì chúng ta sẽ bị failed, lý do là code ban đầu của chúng ta đang mặc định return false)

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, which is currently empty. In the center is the JUnit view, showing a single test run: 'ValidatePhoneNumberTest [Runner]' with one test case 'test()' that took 0.039 seconds. Below the JUnit view is the code editor for 'ValidatePhoneNumberTest.java'. The code contains a JUnit test class with a single test method that asserts the validation of a phone number.

```

1 import static org.junit.Assert.assertEquals;
2
3 class ValidatePhoneNumberTest {
4     private PlaceOrderController placeOrderController;
5
6     @BeforeEach
7     void setUp() throws Exception {
8         placeOrderController = new PlaceOrderController();
9     }
10
11     @Test
12     void test() {
13         boolean isValid = placeOrderController.validatePhoneNumber("0123456789");
14         assertEquals(true, isValid);
15     }
16 }

```

- Vì vậy chúng ta sẽ quay lại class PlaceOrder và implement phương thức validatePhoneNumber. Số điện thoại chỉ được bao gồm 10 ký tự số vậy nên ta sẽ implement, và nếu chúng ta quay lại để chạy lại testcase thì chúng ta sẽ pass

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, showing the project structure with a file 'PlaceOrderController.java' selected. In the center is the code editor for 'PlaceOrderController.java', which contains several validation methods: validateDeliveryInfo, validatePhoneNumber, validateName, and validateAddress. Below the code editor is the JUnit view, showing a single test run: 'ValidatePhoneNumberTest [Runner]' with one test case 'test()' that took 0.011 seconds. The test case asserts the validation of a phone number.

```

1 /**
2 * The method validates the info
3 * @param info
4 * @throws InterruptedException
5 * @throws IOException
6 */
7 public void validateDeliveryInfo(HashMap<String, String> info) throws InterruptedException, IOException{
8
9 }
10
11 public boolean validatePhoneNumber(String phoneNumber) {
12     // check the phoneNumber has 10 digits
13     if (phoneNumber.length() != 10) return false;
14
15     // check the phoneNumber contains only number
16     try {
17         Integer.parseInt(phoneNumber);
18     } catch (NumberFormatException e) {
19         return false;
20     }
21
22     return true;
23 }
24
25 public boolean validateName(String name) {
26     return false;
27 }
28
29 public boolean validateAddress(String address) {
30     return false;
31 }

```

Package Explor JUnit

Finished after 0.011 seconds

Runs: 1/1 Errors: 0 Failures: 0

ValidatePhoneNumberTest [Runner]

- Chúng ta có thể tối ưu dữ liệu test bằng cách đưa vào một list các cặp dữ liệu - kết quả mong đợi (input - expected outcome). Chúng ta có thể làm được

điều này bằng cách sử dụng annotation `@Parameterized` và `@CsvSource`

```

1 package controller;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class ValidatePhoneNumberTest {
6
7     private PlaceOrderController placeOrderController;
8
9     @BeforeEach
10    void setUp() throws Exception {
11         placeOrderController = new PlaceOrderController();
12     }
13
14     @ParameterizedTest
15     @CsvSource({
16         "0123456789,true",
17         "01234,false",
18         "abc123,false",
19         "1234567890,false"
20     })
21     void test(String phone, boolean expected) {
22         // when
23         boolean isValid = placeOrderController.validatePhoneNumber(phone);
24
25         // then
26         assertEquals(expected, isValid);
27     }
28
29 }
30
31
32
33
34

```

- Từ kết quả trên ta có thể thấy chúng ta bị failed một testcase "1234567890,false", mong muốn của chúng ta này là phương thức validatePhoneNumber sẽ trả về false do số điện thoại này không bắt đầu bởi 0, tuy nhiên trong trường hợp này chúng ta đang bị sai và phải quay lại phương thức validatePhoneNumber để điều chỉnh lại. Sau đó ta sẽ có kết quả là cả 4 test case đều pass

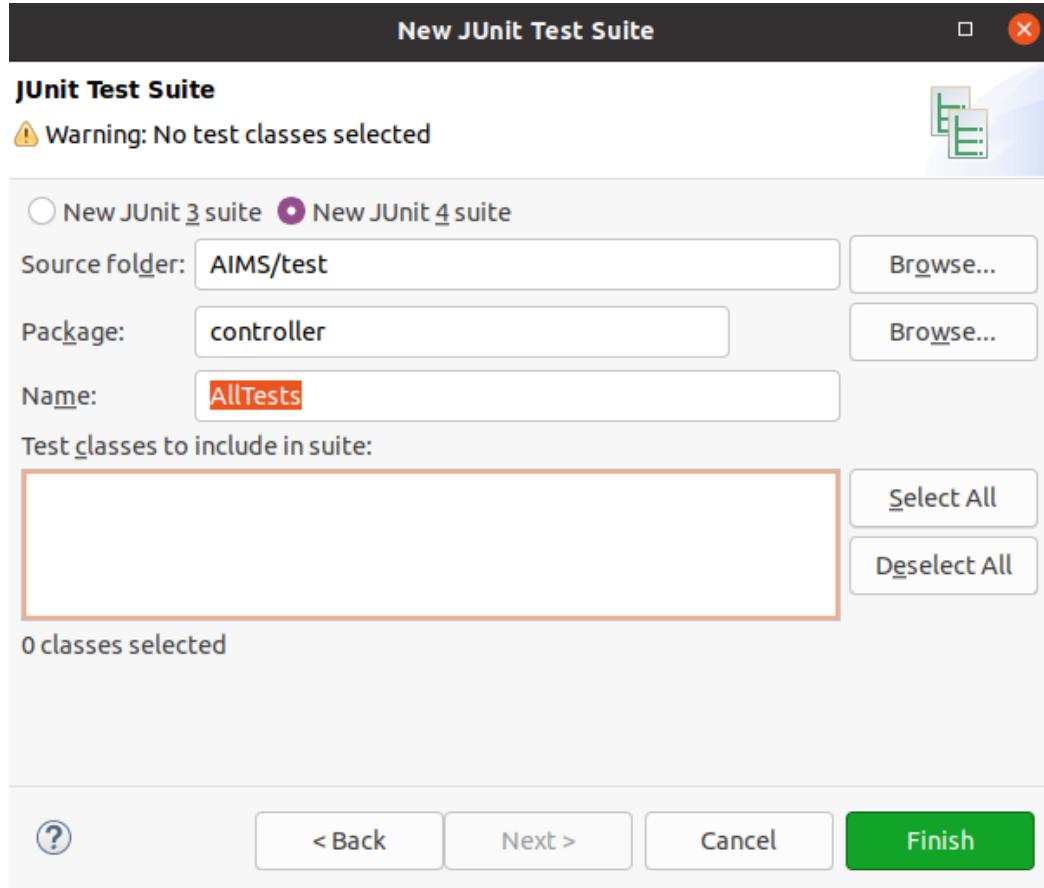
```

1 /**
2  * The method validates the info
3  * @param info
4  * @throws InterruptedException
5  * @throws IOException
6  */
7 public void validateDeliveryInfo(HashMap<String, String> in
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

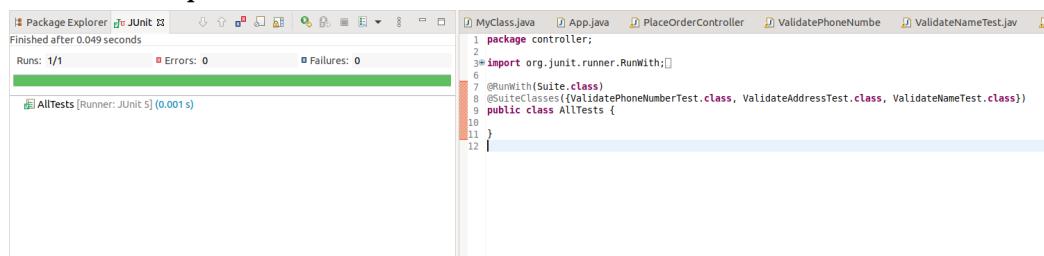
```

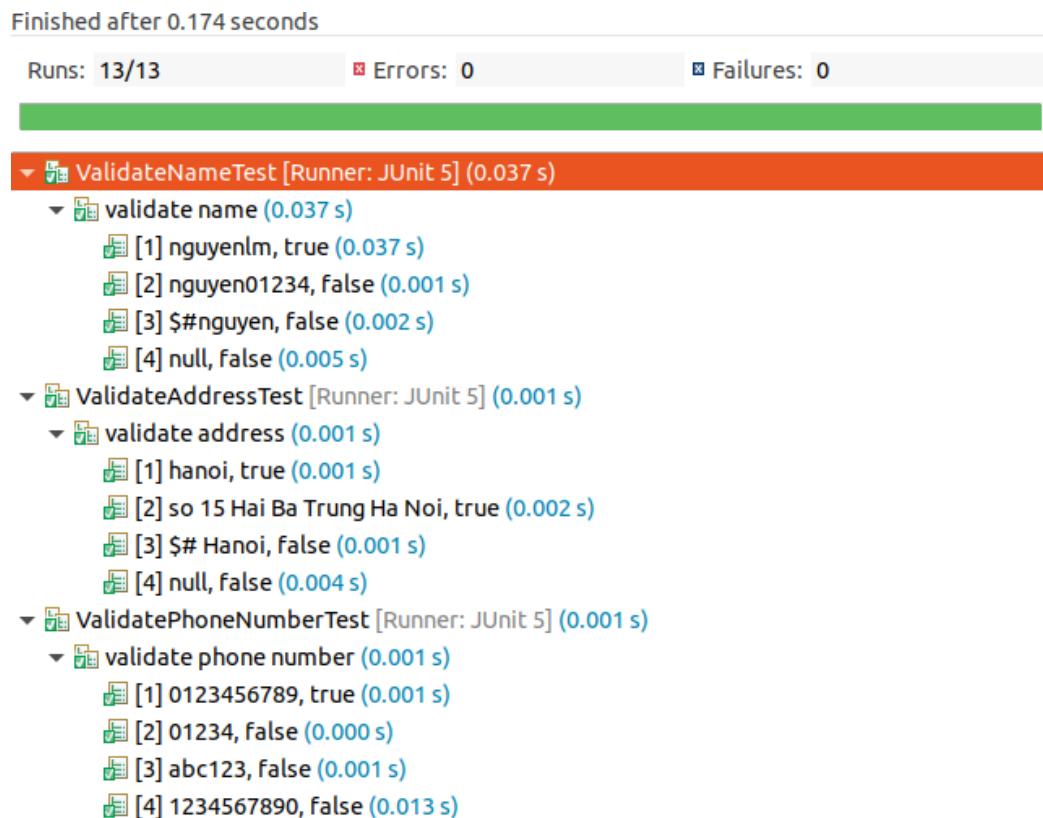
d) Thực hành tạo Test Suite

- Người học tiếp tục thực hành làm cho 2 phương thức còn lại là `validateAddress` và `validateName` dựa vào bảng đặc tả input đầu vào
- Sau khi hoàn thiện chúng ta sẽ có 3 class Test, và chúng ta có thể tiến hành tạo Test Suite. Test Suite là một tập các testcases có liên quan đến một nghiệp vụ nào đó. Click chuột phải vào project -> New -> Test Suite



- Sau đó chúng ta thêm các class cần test vào class AllTests, sau đó bấm Run và xem kết quả





6.3.2. Lập trình

6.3.2.1. Bắt đầu với code sample

- Trước khi đến với bài học các bạn hãy xem qua một class có tên **API.java** khoảng 15-20p và hãy suy ngẫm các câu hỏi:
 - Class này mục đích là gì ?
 - Class này do ai viết ?
 - Sử dụng class này như thế nào ?

```

1 package utils;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.OutputStreamWriter;
8 import java.io.Writer;
9 import java.lang.reflect.Field;
10 import java.lang.reflect.Modifier;
11 import java.net.HttpURLConnection;
12 import java.net.URL;
13 import java.text.DateFormat;
14 import java.text.SimpleDateFormat;
15 import java.util.Arrays;
16 import java.util.LinkedHashSet;
17 import java.util.Set;
18 import java.util.logging.Logger;
19
20 import entity.payment.CreditCard;
21 import entity.payment.PaymentTransaction;
22
23 public class API {
24
25     public static DateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
26     private static Logger LOGGER = Utils.getLogger(Utils.class.getName());
27
28     public static String get(String url, String token) throws Exception {
29         LOGGER.info("Request URL: " + url + "\n");
30         URL line_api_url = new URL(url);
31         HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
32         conn.setDoInput(true);
33         conn.setDoOutput(true);
34         conn.setRequestMethod("GET");
35         conn.setRequestProperty("Content-Type", "application/json");
36         conn.setRequestProperty("Authorization", "Bearer " + token);
37         BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
38         String inputLine;
39         StringBuilder response = new StringBuilder(); // using StringBuilder for the sake of memory and performance
40         while ((inputLine = in.readLine()) != null)
41             System.out.println(inputLine);
42         response.append(inputLine + "\n");
43         in.close();
44         LOGGER.info("Response Info: " + response.substring(0, response.length() - 1).toString());
45         return response.substring(0, response.length() - 1).toString();
46     }
47

```

```

47
48
49  public static String post(String url, String data) throws IOException {
50     allowMethods("PATCH");
51     URL line_api_url = new URL(url);
52     String payload = data;
53     LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");
54     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
55     conn.setDoInput(true);
56     conn.setDoOutput(true);
57     conn.setRequestMethod("PATCH");
58     conn.setRequestProperty("Content-Type", "application/json");
59     Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
60     writer.write(payload);
61     writer.close();
62     BufferedReader in;
63     String inputLine;
64     if (conn.getResponseCode() / 100 == 2) {
65         in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
66     } else {
67         in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
68     }
69     StringBuilder response = new StringBuilder();
70     while ((inputLine = in.readLine()) != null)
71         response.append(inputLine);
72     in.close();
73     LOGGER.info("Response Info: " + response.toString());
74     return response.toString();
75 }
76
77  private static void allowMethods(String... methods) {
78     try {
79         Field methodsField = HttpURLConnection.class.getDeclaredField("methods");
80         methodsField.setAccessible(true);
81
82         Field modifiersField = Field.class.getDeclaredField("modifiers");
83         modifiersField.setAccessible(true);
84         modifiersField.setInt(methodsField, methodsField.getModifiers() & ~Modifier.FINAL);
85
86         String[] oldMethods = (String[]) methodsField.get(null);
87         Set<String> methodsSet = new LinkedHashSet<>(Arrays.asList(oldMethods));
88         methodsSet.addAll(Arrays.asList(methods));
89         String[] newMethods = methodsSet.toArray(new String[0]);
90
91         methodsField.set(null/* static field */, newMethods);
92     } catch (NoSuchFieldException | IllegalAccessException e) {
93         throw new IllegalStateException(e);
94     }
95 }
96
97 }

```

- Nếu như sau khi xem xong và bạn đã hiểu hết những gì viết trong class thì chắc hẳn bạn phải là một lập trình viên Java nhiều kinh nghiệm
- Tuy nhiên đa phần các bạn sẽ bị choáng ngợp và cảm thấy khó hiểu ở class khoảng gần 100 dòng này. Bạn sẽ không hiểu class này mục đích là gì, dùng như thế nào và ai viết
- Không những class trên gây khó hiểu mà nó đang còn tồn đọng nhiều vấn đề như tái cấu trúc, tối ưu code,..
- Vậy bài học hôm nay sẽ giúp các bạn giải quyết từng vấn đề và bạn có thể áp dụng vào trong Project của mình

6.3.2.2. Làm quen với Javadoc

- Chắc hẳn trong chúng ta thì ai cũng đã nghe tới khái niệm như thêm comment hoặc documentation cho các class hoặc method mình viết
- Documentation là gì? Documentation đơn giản là các đoạn text được thêm vào trong mã nguồn của dự án phầm mềm với mục đích giải thích những cái bạn đang làm, thực hiện như thế nào và làm thế nào để sử dụng nó.
- Việc tạo thói quen thêm comment và documentation vào code thì có vai trò rất quan trọng, nó sẽ giúp cho người khác hiểu code của bạn, và cũng có thể là chính bạn sau này khi đọc lại code của mình. Hơn thế nữa việc thêm documentation vào code sẽ chứng tỏ bạn là một lập trình viên chuyên nghiệp.
- JAVADOC là documentation cho ngôn ngữ Java, mục đích chính của nó cũng là giúp cho bạn giải thích những đoạn code được viết. Việc thêm JAVADOC có thể thực hiện thủ công bằng cách gõ từng ký tự nhưng cũng có rất nhiều công cụ IDE hỗ trợ bạn tạo doc tự động như: Eclipse, intellij, VSCode,...
- Cú pháp của JAVADOC: các documentation của Java được đặt ở trong cặp /** */ và có thể thêm nhiều dòng trong giữa cặp dấu.
- JAVADOC thường được mô tả bởi các annotation (bắt đầu bởi @), một vài loại annotation phổ biến trong JAVADOC
 - **@author:** chỉ tên tác giả của đoạn code hoặc có thể là người đóng góp nhiều nhất. Thường được áp dụng cho các level: class hoặc package
 - **@param:** Mô tả tham số truyền vào một phương thức hoặc constructor
 - **@return:** mô tả giá trị trả về của một class hoặc phương thức
 - **@since:** phiên bản mà thuộc tính được thêm vào
 - **@throws:** loại exception mà phương thức có thể tung ra
 - **@deprecated:** chỉ cho người khác biết là phương thức hoặc class này không còn được sử dụng nữa
 - **{@link}:** tạo liên kết tới những phương thức hoặc phần nội dung khác
- Người học có thể tham khảo chi tiết thêm những loại annotation Java cung cấp ở link sau: <https://idratherbewriting.com/java-javadoc-tags/>

6.3.2.3. Thực hành tạo Javadoc với Eclipse

e) Import project vào Eclipse

- Mở Eclipse và import Project đã clone về máy, hoặc nếu bạn đã clone thì sử dụng git pull để lấy code mới nhất về nhánh master
- Mở class src/utils/API.java

- Tiếp theo chúng ta sẽ tiến hành thêm javadoc cho class và method ở trong class API.java này

f) Thực hành thêm doc cho API class

- Để thêm doc vào trong Java code thông qua Eclipse thì chúng ta có 3 cách:
 - + Gõ thủ công
 - + Dùng thanh công cụ
 - + Dùng các shortcuts
- Để thêm javadoc cho class, method hoặc attribute thông qua thanh công cụ thì ta sẽ click con trỏ chuột lên ngay bên trên class, method hoặc attribute đó. Sau đó click chuột phải -> Source -> Generate Element Comment
- Cách nhanh nhất để thêm java doc chính là thông qua shortcurt. Để thêm javadoc thì ta click chuột lên phía trên của class, method hoặc attribute sau đó gõ /** và ấn Enter, sau đó Eclipse sẽ tạo Javadoc tự động cho chúng ta
- Ví dụ với class API

```

12 import java.net.URL;
13 import java.text.DateFormat;
14 import java.text.SimpleDateFormat;
15 import java.util.Arrays;
16 import java.util.LinkedHashSet;
17 import java.util.Set;
18 import java.util.logging.Logger;
19
20 import entity.payment.CreditCard;
21 import entity.payment.PaymentTransaction;
22
23
24 /**
25 * @author nguyenlm
26 *
27 */
28 public class API {
29
30     public static DateFormat DATE_FORMATER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
31     private static Logger LOGGER = Utils.getLogger(Utils.class.getName());
32

```

- Việc thêm javadoc ở trên mới cho class API chỉ là bước đầu và khá sơ khai, vậy nên chúng ta cần bổ sung thêm các thông tin khác như: mô tả, @version, ngày viết chương trình
- Việc thêm mô tả cho một class, method hoặc attribute nên để lên đầu của documentation. Sau khi thêm các thông tin ta sẽ có thông tin documentation như sau:

```

24 /**
25 * Class cung cap cac phuong thuc giup gui request len server va nhan du lieu tra ve
26 * Date: 28/11/2020
27 * @author nguyenvn
28 * @version 1.0
29 */
30 public class API {
31
32     public static DateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
33     private static Logger LOGGER = Utils.getLogger(Utils.class.getName());
34

```

- Sau khi thêm documentation cho class API thì chúng ta đã có một cái nhìn tổng quan về mục đích của class này . Tiếp theo chúng ta sẽ tiếp tục thêm documentation cho các attribute và method

+ Attributes

```

30 public class API {
31
32 /**
33 * Thuoc tinh giup format ngay thang theo dinh dang
34 */
35 public static DateFormat DATE_FORMATTER = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
36
37 /**
38 * Thuoc tinh giup log ra thong tin ra console
39 */
40 private static Logger LOGGER = Utils.getLogger(Utils.class.getName());

```

+ get method

```

42 /**
43 * Phuong thuc giup goi cac api dang GET
44 * @param url: duong dan toi server can request
45 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
46 * @return response: phan hoi tu server (dang string)
47 * @throws Exception
48 */
49 public static String get(String url, String token) throws Exception {
50     LOGGER.info("Request URL: " + url + "\n");
51     URL line_api_url = new URL(url);
52     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
53     conn.setDoInput(true);
54     conn.setDoOutput(true);
55     conn.setRequestMethod("GET");
56     conn.setRequestProperty("Content-Type", "application/json");
57     conn.setRequestProperty("Authorization", "Bearer " + token);
58     BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
59     String inputLine;
60     StringBuilder response = new StringBuilder(); // using StringBuilder for the sake of memory and performance
61     while ((inputLine = in.readLine()) != null)
62         System.out.println(inputLine);
63     response.append(inputLine + "\n");
64     in.close();
65     LOGGER.info("Response Info: " + response.substring(0, response.length() - 1).toString());
66     return response.substring(0, response.length() - 1).toString();
67 }
68

```

+ post method

```

-- 70② /**
 * Phuong thuc giup goi cac api dang POST (thanh toan,...)
 * @param url: duong dan toi server can request
 * @param data: du lieu dua len server de xu ly (dang JSON)
 * @return respose: phan hoi tu server (dang string)
 * @throws IOException
 */
77② public static String post(String url, String data) throws IOException {
    allowMethods("PATCH");
    URL line_api_url = new URL(url);
    String payload = data;
    LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod("PATCH");
    conn.setRequestProperty("Content-Type", "application/json");
    Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
    writer.write(payload);
    writer.close();
    BufferedReader in;
    String inputLine;
    if (conn.getResponseCode() / 100 == 2) {
        in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    } else {
        in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
    }
    StringBuilder response = new StringBuilder();
    while ((inputLine = in.readLine()) != null)
        response.append(inputLine);
    in.close();
    LOGGER.info("Respone Info: " + response.toString());
    return response.toString();
}

```

+ allowMethods

```

105② /**
 * Phuong thuc cho phép gọi các loại giao thuc API khac nhau nhu PATCH, PUT,... (chi hoat dong voi Java 11)
 * @deprecated chi hoat dong voi Java <= 11
 * @param methods: giao thuc can cho cho phep [PATCH, PUT,...]
 */
110② private static void allowMethods(String... methods) {
    try {
        Field methodsField = HttpURLConnection.class.getDeclaredField("methods");
        methodsField.setAccessible(true);

        Field modifiersField = Field.class.getDeclaredField("modifiers");
        modifiersField.setAccessible(true);
        modifiersField.setInt(methodsField, methodsField.getModifiers() & ~Modifier.FINAL);

        String[] oldMethods = (String[]) methodsField.get(null);
        Set<String> methodsSet = new LinkedHashSet<>(Arrays.asList(oldMethods));
        methodsSet.addAll(Arrays.asList(methods));
        String[] newMethods = methodsSet.toArray(new String[0]);

        methodsField.set(null/* static field */, newMethods);
    } catch (NoSuchFieldException | IllegalAccessException e) {
        throw new IllegalStateException(e);
    }
}

```

g) Thực hành thêm comment cho các phương thức

- Để làm rõ hơn các câu lệnh trong từng method thì chúng ta có thể thêm comment (//) vào trong các method như sau

```

42 /**
43 * Phuong thuc giup goi cac api dang GET (lay so du tai khoan,...)
44 * @param url: duong dan toi server can request
45 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
46 * @return respone: phan hoi tu server (dang string)
47 * @throws Exception
48 */
49 public static String get(String url, String token) throws Exception {
50
51     // phan 1: setup
52     LOGGER.info("Request URL: " + url + "\n");
53     URL line_api_url = new URL(url);
54     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
55     conn.setDoInput(true);
56     conn.setDoOutput(true);
57     conn.setRequestMethod("GET");
58     conn.setRequestProperty("Content-Type", "application/json");
59     conn.setRequestProperty("Authorization", "Bearer " + token);
60
61     // phan 2: doc du lieu tra ve tu server
62     BufferedReader in;
63     String inputLine;
64     if (conn.getResponseCode() / 100 == 2) {
65         in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
66     } else {
67         in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
68     }
69     StringBuilder responce = new StringBuilder(); // su dung String Builder cho viectoi uu ve mat bo nho
70     while ((inputLine = in.readLine()) != null)
71         System.out.println(inputLine);
72     responce.append(inputLine + "\n");
73     in.close();
74     LOGGER.info("Responce Info: " + responce.substring(0, responce.length() - 1).toString());
75     return responce.substring(0, responce.length() - 1).toString();
76 }
77

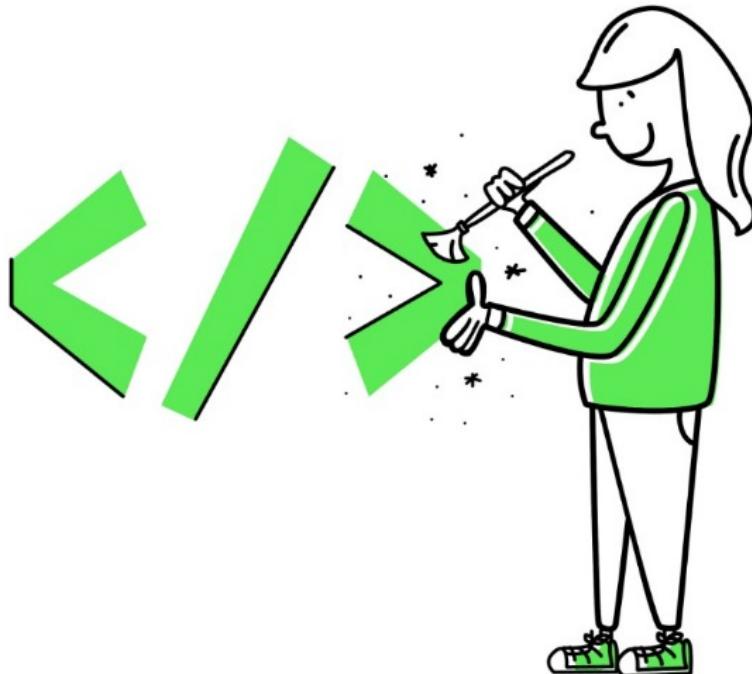
```

```

79⑨ /**
80 * Phuong thuc giup goi cac api dang POST (thanh toan,...)
81 * @param url: duong dan toi server can request
82 * @param data: du lieu dua len server de xu ly (dang JSON)
83 * @return responce: phan hoi tu server (dang string)
84 * @throws IOException
85 */
86 public static String post(String url, String data) throws IOException {
87     allowMethods("PATCH");
88
89     // phan 1: setup
90     URL line_api_url = new URL(url);
91     String payload = data;
92     LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");
93     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
94     conn.setDoInput(true);
95     conn.setDoOutput(true);
96     conn.setRequestMethod("PATCH");
97     conn.setRequestProperty("Content-Type", "application/json");
98
99     // phan 2: gui du lieu
100    Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
101    writer.write(payload);
102    writer.close();
103
104    // phan 3: doc du lieu gui ve tu server
105    BufferedReader in;
106    String inputLine;
107    if (conn.getResponseCode() / 100 == 2) {
108        in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
109    } else {
110        in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
111    }
112    StringBuilder response = new StringBuilder(); // su dung String Builder cho viectoi uu ve mat bo nho
113    while ((inputLine = in.readLine()) != null)
114        response.append(inputLine);
115    in.close();
116    LOGGER.info("Responce Info: " + response.toString());
117    return response.toString();
118 }

```

6.3.2.4. Làm quen với refactoring



a) Mục đích của refactoring

- Refactoring là việc thay đổi hoặc tái cấu trúc lại mã nguồn nhằm mục đích chuyển code rối rắm (mess code) về dạng code dễ hiểu hơn (clean code).
- Refactoring giúp cho việc bảo trì một cách dễ dàng, giảm bớt đi sự phức tạp rườm rà từ đó giúp chúng ta dễ quản lý mã nguồn và đưa được product của mình ra thị trường một cách nhanh chóng

b) Khi nào cần refactoring code

- Khi code trong project của mình bị lặp đi lặp lại nhiều lần
- Khi code quá rườm rà, dài dòng, khó hiểu và khó thêm thuộc tính mới
- Trong quá trình fix bug, khi chúng ta liên tục gặp phải bugs thì việc refactoring cho code cleaner hơn sẽ giúp chúng ta nhận diện ra những vấn đề trong code
- Trong quá trình review code trước khi đưa ra sản phẩm, đây có thể là cơ hội cuối cùng để tinh chỉnh code trước khi đưa ra sản phẩm

c) Một vài loại refactoring phổ biến

- **Extract method:** phương pháp này có nghĩa là nếu như bạn thấy đoạn code nào đó lặp đi lặp lại nhiều lần trong các phương thức khác thì hãy tách đoạn code đó ra thành một phương thức riêng
- **Extract class:** đây là cách làm tương tự như extract method nhưng ở mức độ class, những phương thức nào có liên quan tới nhau và hay được sử

dụng thì chúng ta có thể tách ra một class khác và tái sử dụng bằng cách kế thừa hoặc kết hợp

6.3.2.5. Thực hành refactoring code với class API

a) Nhận diện vấn đề

- Đây là thời điểm chúng ta sẽ quay lại với class API phía trên, nếu chúng ta để ý kỹ chúng ta sẽ thấy 2 phương thức get và post có nhiều phần chung được phân tách như sau:

```

42⊕ /**
 * Phuong thuc giup goi cac api dang GET (lay so du tai khoan,...)
44 * @param url: duong dan toi server can request
45 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
46 * @return responce: phan hoi tu server (dang string)
47 * @throws Exception
48 */
49⊕ public static String get(String url, String token) throws Exception {
50
51     // phan 1: setup
52     LOGGER.info("Request URL: " + url + "\n");
53     URL line_api_url = new URL(url);
54     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
55     conn.setDoInput(true);
56     conn.setDoOutput(true);
57     conn.setRequestMethod("GET");
58     conn.setRequestProperty("Content-Type", "application/json");
59     conn.setRequestProperty("Authorization", "Bearer " + token);
60
61     // phan 2: doc du lieu tra ve tu server
62     BufferedReader in;
63     String inputLine;
64     if (conn.getResponseCode() / 100 == 2) {
65         in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
66     } else {
67         in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
68     }
69     StringBuilder responce = new StringBuilder(); // su dung String Builder cho viectoi uu ve mat bo nho
70     while ((inputLine = in.readLine()) != null)
71         System.out.println(inputLine);
72     responce.append(inputLine + "\n");
73     in.close();
74     LOGGER.info("Responce Info: " + responce.substring(0, responce.length() - 1).toString());
75     return responce.substring(0, responce.length() - 1).toString();
76 }
77

```

```

79① /**
80 * Phuong thuc giup goi cac api dang POST (thanh toan,...)
81 * @param url: duong dan toi server can request
82 * @param data: du lieu dua len server de xu ly (dang JSON)
83 * @return responce: phan hoi tu server (dang string)
84 * @throws IOException
85 */
86② public static String post(String url, String data) throws IOException {
87     allowMethods("PATCH");
88
89     // phan 1: setup
90     URL line_api_url = new URL(url);
91     String payload = data;
92     LOGGER.info("Request Info:\nRequest URL: " + url + "\n" + "Payload Data: " + payload + "\n");
93     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
94     conn.setDoInput(true);
95     conn.setDoOutput(true);
96     conn.setRequestMethod("PATCH");
97     conn.setRequestProperty("Content-Type", "application/json");
98
99     // phan 2: gui du lieu
100    Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
101    writer.write(payload);
102    writer.close();
103
104    // phan 3: doc du lieu gui ve tu server
105    BufferedReader in;
106    String inputLine;
107    if (conn.getResponseCode() / 100 == 2) {
108        in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
109    } else {
110        in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
111    }
112    StringBuilder response = new StringBuilder(); // su dung String Builder cho viectoi uu ve mat bo nho
113    while ((inputLine = in.readLine()) != null)
114        response.append(inputLine);
115    in.close();
116    LOGGER.info("Responce Info: " + response.toString());
117    return response.toString();
118}

```

- Sau khi nhìn lại 2 methods và có sự phân cách giữa các dòng chúng ta sẽ thấy giữa 2 phương thức post và get này có các đoạn code chung như setup connection và đọc dữ liệu trả về từ server,
- Ta có thể tiến hành refactor đoạn mã nguồn này bằng cách chọn đoạn mã nguồn cần extract, ấn chuột phải, chọn Refactor → Extract method.

b) Thực hành refactoring

- Chúng ta sẽ tiến hành trích xuất 2 phương thức đặt tên là setupConnection và readResponse

- Code sau khi refactoring sẽ như sau:

- *setupConnection()*

```

44 /**
45 * Thiết lập connection tới server
46 * @param url: đường dẫn tới server cần request
47 * @param method: giao thức api
48 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
49 * @return connection
50 * @throws IOException
51 */
52 private static HttpURLConnection setupConnection(String url, String method, String token) throws IOException {
53     // phan 1: setup
54     LOGGER.info("Request URL: " + url + "\n");
55     URL line_api_url = new URL(url);
56     HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
57     conn.setDoInput(true);
58     conn.setDoOutput(true);
59     conn.setRequestMethod(method);
60     conn.setRequestProperty("Content-Type", "application/json");
61     conn.setRequestProperty("Authorization", "Bearer " + token);
62     return conn;
63 }
64

```

- *ReadResponse()*

```

65 /**
66 * Phương thức đọc dữ liệu trả về từ server
67 * @param conn: connection to server
68 * @return response: phần hỏi trả về từ server
69 * @throws IOException
70 */
71 private static String readResponse(HttpURLConnection conn) throws IOException {
72     // phan 2: đọc dữ liệu trả về từ server
73     BufferedReader in;
74     String inputLine;
75     if (conn.getResponseCode() / 100 == 2) {
76         in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
77     } else {
78         in = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
79     }
80     StringBuilder response = new StringBuilder(); // sử dụng String Builder cho việc tối ưu về mặt bộ nhớ
81     while ((inputLine = in.readLine()) != null)
82         System.out.println(inputLine);
83     response.append(inputLine + "\n");
84     in.close();
85     LOGGER.info("Response Info: " + response.substring(0, response.length() - 1).toString());
86     return response.substring(0, response.length() - 1).toString();
87 }
88

```

- Khi đó 2 phương thức get và post sẽ còn lại như sau:

```

89 /**
90 * Phuong thuc giup goi cac api dang GET (lay so du tai khoan,...)
91 * @param url: duong dan toi server can request
92 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
93 * @return respone: phan hoi tu server (dang string)
94 * @throws Exception
95 */
96 public static String get(String url, String token) throws Exception {
97
98     // phan 1: setup
99     HttpURLConnection conn = setupConnection(url, "GET", token);
100
101    // phan 2: doc du lieu tra ve tu server
102    String respone = readRespone(conn);
103
104    return respone;
105}
106
107 /**
108 * Phuong thuc giup goi cac api dang POST (thanh toan,...)
109 * @param url: duong dan toi server can request
110 * @param token: doan ma bam can cung cap de xac thuc nguoi dung
111 * @param data: du lieu dua len server de xu ly (dang JSON)
112 * @return respone: phan hoi tu server (dang string)
113 * @throws IOException
114 */
115 public static String post(String url, String data, String token) throws IOException {
116     // cho phep PATCH protocol
117     allowMethods("PATCH");
118
119
120     // phan 1: setup
121     HttpURLConnection conn = setupConnection(url, "GET", token);
122
123     // phan 2: gui du lieu
124     Writer writer = new BufferedWriter(new OutputStreamWriter(conn.getOutputStream()));
125     writer.write(data);
126     writer.close();
127
128     // phan 3: doc du lieu gui ve tu server
129     String respone = readRespone(conn);
130
131     return respone;
132 }

```

6.4. BÀI TẬP

- Thiết kế các testcase cho UC Place Rush Order bằng phương pháp TDD như đã học và được hướng dẫn thực hành
- Thực hành thêm javadoc cho các class, method và attribute cho UC Place Rush Order. Sau đó tiến hành review code và refactoring lại

những điểm code chưa hợp lý (lưu ý cần chỉ rõ cần refactoring ở điểm nào và tại sao).

7. BÀI THỰC HÀNH SỐ 05 – CÁC KHÁI NIỆM VÀ NGUYÊN LÝ THIẾT KẾ PHẦN MỀM

7.1. MỤC ĐÍCH VÀ NỘI DUNG

- Trong bài thực hành này, người học sẽ làm quen với các khái niệm thiết kế (Design Concepts) và các nguyên lý cơ bản trong thiết kế (Design Principles)
- Đối với Design Concepts, người học sẽ được tiếp cận với 2 khái niệm chính là cohesion và coupling.
- Đối với Design Principles, người học sẽ được làm quen với SOLID, đó là 5 chữ cái viết tắt của 5 nguyên lý cơ bản trong thiết kế: Single responsibility principle, Open/Closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle.
- Sau bài thực hành, người học sẽ có thêm những ý tưởng để tạo ra một bản thiết kế tối ưu, dễ bảo trì, mở rộng...

7.2. CHUẨN BỊ

- Người học cần hiểu được các kiến thức lý thuyết về Design Concept và Design Principle đã được học ở trên lớp trước khi bắt tay vào thực hành.
- Lưu ý các hướng dẫn dưới đây chỉ là 1 số gợi ý chứ không phải là tất cả những gì cần tối ưu cho bản thiết kế và mã nguồn.
- Bản thiết kế và mã nguồn được xem xét trong bài thực hành này có thể clone trên <https://github.com/leminhnguyen/AIMS-Student>

7.3. NỘI DUNG CHI TIẾT

7.3.1. Một số yêu cầu mở rộng

Dưới đây là danh sách các yêu cầu mở rộng:

a) Thay đổi phí ship:

- Phí vận chuyển hiện nay cũng phụ thuộc vào trọng lượng thực tế, khoảng cách và độ cồng kềnh của sản phẩm (tức là kích thước của sản phẩm: chiều dài, chiều rộng, chiều cao)
- Công thức chuyển đổi được đưa ra như sau:
- Alternative weight (kg) =Length (cm) × Width (cm) × Height (cm) / 6000

- Trọng lượng của sản phẩm bằng trọng lượng thực tế cộng với trọng lượng thay thế.

b) Thêm một cách tính tiền mới:

- Domestic debit card:
 - Issuing bank, e.g., VietinBank
 - Card number: 16 digits
 - Valid-from date, e.g., 12/33
 - Cardholder's name, e.g., VU DUY MANH
- Đối với các yêu cầu mở rộng này, giả sử API vẫn giữ nguyên, giả sử thông tin thẻ được thay đổi.

c) Sử dụng interbank khác

- Sử dụng một interbank khác với giao thức kết nối và API thay đổi.

7.3.2. Coupling và Cohesion: các khái niệm cơ bản trong thiết kế

7.3.2.1. Coupling

- Để có một bản thiết kế tốt thì cần phải thiết kế sao cho coupling lỏng để khi có sự thay đổi ở một module thì ảnh hưởng ít nhất có thể đến các module khác. Coupling càng lỏng lẻo, càng tốt. Trong phần này, chúng ta sẽ xem xét về các mức coupling giữa các module trong một project mẫu.

a) Content coupling

- May mắn thay, project của chúng tôi hiện tại chưa có loại mức độ coupling này.
- Để minh chứng cho điều này, hãy xem class Order trong package entity.order. Hiện tại, lớp này có thuộc tính deliveryInfo có kiểu HashMap và một getter public: getDeliveryInfo() cho thuộc tính này. Trong tương lai, có thể có một module quản lý để lấy đối tượng deliveryInfo bằng cách gọi getDeliveryInfo() của lớp Order và sau đó thay đổi deliveryInfo bằng cách gọi put() của lớp HashMap. Do đó, giá trị của deliveryInfo sẽ bị thay đổi trong khi object Order không biết gì về việc sửa đổi thuộc tính của nó từ bên ngoài.
- Để giảm mức độ ghép nối, chúng ta có thể:
 - Bỏ qua những accessors/getters dư thừa
 - Chọn chỉ định truy cập phù hợp.
 - Đóng gói dữ liệu bằng một lớp / đổi tượng chuyên biệt.
 - Sử dụng design pattern như builder patterns.
- Hãy nhớ nghĩ đến mức độ phụ thuộc giữa các mô-đun khi thiết kế.

b) Common Coupling

- Đây là trường hợp 2 module cùng chia sẻ chung dữ liệu, 2 global structure có thể cùng vào chỉnh sửa, truy cập hoặc có chung những khối mã nguồn thì sẽ vi phạm common coupling.
- Tuy nhiên, object oriented programming không có common data. Tất cả data đều thuộc về lớp. Ví dụ khi dùng C, bạn có data structure khai báo là global, sau đó sẽ có một số phương thức ghi vào structure, các phương thức khác đọc từ đó, trường hợp này sẽ vi phạm common coupling. Còn trong Object Oriented, không có bất cứ data nào mà không thuộc về lớp. Do đó, project hiện tại (JAVA) không vi phạm common coupling.

c) Control Coupling

- Một module vi phạm control coupling khi nó truyền tham số điều khiển cho các module khác thông qua việc gọi method. Điều này không tốt vì thành phần được gọi sẽ biết được cấu trúc bên trong thành phần gọi và khi cấu trúc này bị thay đổi thì thành phần được gọi sẽ phải thay đổi theo.
- Trong trường hợp sau đây, điều gì sẽ xảy ra nếu có nhiều phương thức thanh toán.
- Thông thường, chúng ta sẽ sử dụng control structure như if-else và sẽ gặp phải vấn đề về control coupling như sau:

```
if (paymentMethod == "NGAN_LUONG"){
    NGANLUONGSubsystem nganLuong = new NGANLUONGSubsystem();
    nganLuong.pay(args);
} else {
    OceanBankSubsystem oceanbank = new OceanBankSubsystem();
    oceanbank.pay(args);
}
```

- Cách xử lý:
 - Tách phương thức
 - Áp dụng các design patterns như strategy pattern hoặc factory pattern

d) Stamp coupling

- Hai lớp được coi là vi phạm stamp coupling nếu một lớp gửi một collection hoặc một object dưới dạng tham số và chỉ một vài phần data được sử dụng ở lớp thứ hai. Để giảm thiểu level coupling này, chúng ta truyền đủ những tham số cần thiết trong function gọi. Chú ý rằng, stamp coupling là mức có thể chấp nhận được.

e) Data coupling

- Nếu các module của bạn ở mức data coupling, thiết kế của bạn là thiết kế tốt. Đây là level mà chúng ta hướng đến.

f) Uncoupled

- Nếu các modules là uncoupled, vui lòng tách chương trình của bạn thành các module độc lập.

7.3.2.2. Cohesion

- Nhìn chung, để tăng mức độ (level) cohesion, chúng ta có thể thử đặt mỗi phần vào một module khác phù hợp hơn (tạo module mới nếu cần)

a) Coincidental cohesion

- Các sub component đặt trong 1 component vì tính ngẫu nhiên
- Rõ ràng, chúng ta có thể thấy loại cohesion này trong class Config hoặc class Utils trong package utils

b) Logical cohesion

- Các thành phần trong 1 module có liên quan đến nhau nhưng về mặt logic chứ không phải chức năng.
- Quan sát lại phần Control Coupling bên trên, nếu chúng ta chia đoạn code thành 2 methods và đưa chúng vào trong cùng một lớp, chúng ta có thể đổi mặt với logical cohesion. Do đó, chúng ta cần xem xét đưa chúng vào trong những class hoặc package khác.

c) Temporal cohesion

- Các sub component đặt trong một component vì chúng liên quan đến nhau về mặt thời gian chứ không phải về mặt chức năng.
- Thông thường, chúng ta cần loại cohesion này trong các module khởi tạo hoặc clean-up.
- Ví dụ: viết một phương thức khởi tạo tất cả các thành phần của hệ thống → vi phạm Temporal cohesion: thành phần này đi khỏi tạo dữ liệu cho thành

phần khác, thay vì vậy ta nên gọi đến thành phần khởi tạo của từng thành phần.

d) Procedural cohesion

- Các thành phần đặt trong 1 module vì nó có quan hệ chặt chẽ với nhau theo một thứ tự nào đó chứ không liên hệ với nhau về mặt chức năng.
- Chúng ta có thể thấy loại cohesion này ở trong class PlaceOrderController trong package control. Chúng ta validate các trường dữ liệu từng bước một với các phương thức validation.

```
public void validateDeliveryInfo(HashMap<String, String> info) throws InterruptedException, IOException {
}

public boolean validatePhoneNumber(String phoneNumber) {
    // TODO: your work
    return false;
}

public boolean validateName(String name) {
    // TODO: your work
    return false;
}

public boolean validateAddress(String address) {
    // TODO: your work
    return false;
}
```

e) Communicational cohesion

- Các thành phần đặt trong cùng một module vì cùng thực hiện trên cùng một dữ liệu.
- Các module theo kiểu cohesion này hoạt động trên cùng một đầu vào hoặc trả về cùng một đầu ra (ví dụ: các thành phần trong InterbankSubsystemController đều nhận các dữ liệu đầu vào giống nhau và dữ liệu đầu ra cùng trả về kiểu PaymentTransaction)

```
/**
 * @see InterbankInterface#payOrder(entity.payment.CreditCard, int,
 *      java.lang.String)
 */
public PaymentTransaction payOrder(CreditCard card, int amount, String contents) {
    PaymentTransaction transaction = ctrl.payOrder(card, amount, contents);
    return transaction;
}

/**
 * @see InterbankInterface#refund(entity.payment.CreditCard, int,
 *      java.lang.String)
 */
public PaymentTransaction refund(CreditCard card, int amount, String contents) {
    PaymentTransaction transaction = ctrl.refund(card, amount, contents);
    return transaction;
}
```

f) Sequential cohesion

- Trong một module, output của thành phần này là input của thành phần kia.
- Chỉ có một vấn đề nhỏ ở đây. Nếu chúng ta có 1 trình tự, chúng ta có thể chia nó thành nhiều phần khác nhau. Có nghĩa là các class được tạo ra có thể làm nhiều hơn một chức năng hoặc hoàn toàn ngược lại - chỉ một phần của chức năng.

g) Informational cohesion

- Các operation có tính độc lập (có input và output riêng nhưng chúng có thể thao tác trên một tập dữ liệu chung là attribute của lớp đó)
- Chúng ta có thể thấy rõ loại cohesion này ở trong các lớp entity như là Media hay Order.

h) Functional cohesion

- Mỗi một subcomponent thực hiện một công việc nào đó và hướng đến mục đích chung của component đó.
- Nhìn lại class API sau khi đã được refactor ở bài lab trước, bạn có thể thấy rõ được, đầu ra phương thức setUpConnection() là đầu vào cho phương thức get().

```

public static String get(String url, String token) throws Exception {
    LOGGER.info("Request URL: " + url + "\n");
    URL line_api_url = new URL(url);
    HttpURLConnection conn = setUpConnection(token, "GET", line_api_url);
    BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String inputLine;
    StringBuilder response = new StringBuilder(); // using StringBuilder for the sake of memory and performance
    while ((inputLine = in.readLine()) != null) {
        System.out.println(inputLine);
        response.append(inputLine + "\n");
    }
    in.close();
    LOGGER.info("Response Info: " + response.substring(0, response.length() - 1).toString());
    return response.substring(0, response.length() - 1).toString();
}

private static HttpURLConnection setUpConnection(String token, String method, URL line_api_url)
    throws IOException, ProtocolException {
    HttpURLConnection conn = (HttpURLConnection) line_api_url.openConnection();
    conn.setDoInput(true);
    conn.setDoOutput(true);
    conn.setRequestMethod(method);
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setRequestProperty("Authorization", "Bearer " + token);
    return conn;
}

```

7.3.3. Nguyên lý thiết kế SOLID

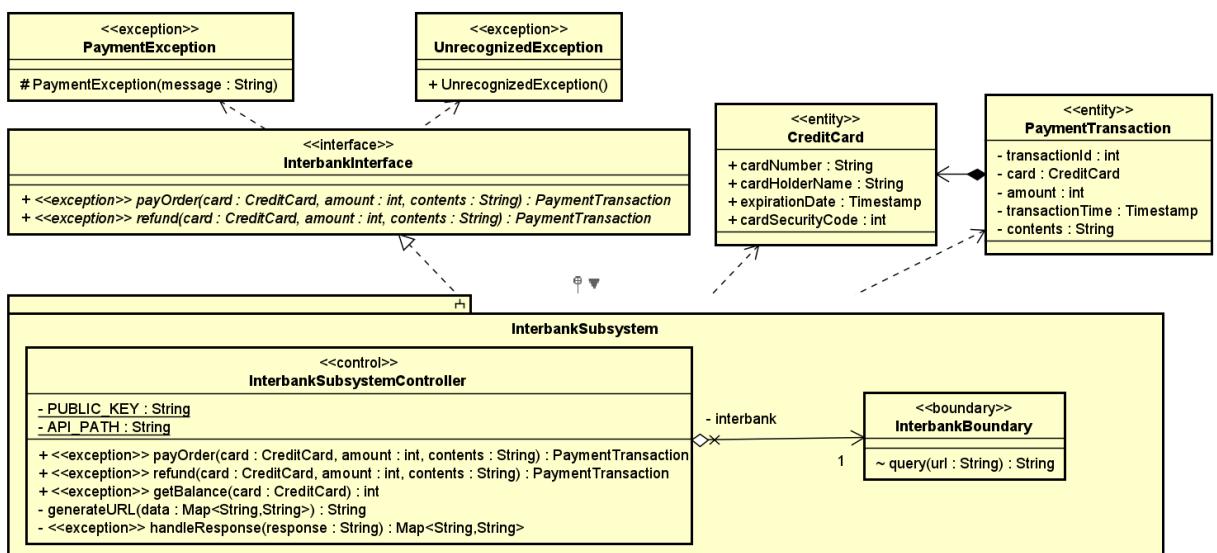
7.3.3.1. Single Responsibility

- Một lớp chỉ nên chịu trách nhiệm về một nhiệm vụ cụ thể nào đó mà thôi. Một lớp quá nhiều chức năng sẽ trở nên cồng kềnh, khó đọc, khó bảo trì, nên chỉ có duy nhất một lý do để thay đổi một lớp, không nên có 2 lý do trớ lén.

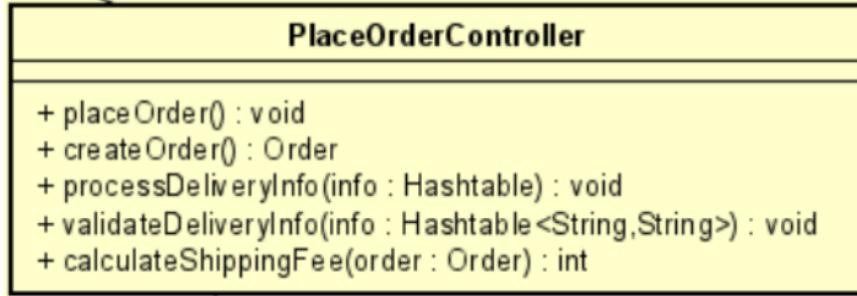
- Như đã nêu trong bài thực hành Class Design, InterbankSubsystemController chịu trách nhiệm cho 2 nhiệm vụ: (1) điều khiển luồng dữ liệu (2) chuyển đổi dữ liệu (chuyển đổi dữ liệu nhận về từ api sang dạng controller yêu cầu). Do đó, lớp này phải được thay đổi khi mà luồng dữ liệu thay đổi (ví dụ các tính năng mới được thêm vào) hoặc cách chuyển đổi dữ liệu thay đổi (ví dụ như thay đổi dữ liệu định dạng bắt buộc), đây là một chỉ báo của một bản thiết kế chưa tốt.
- Nguyên tắc này nghe có vẻ đơn giản nhưng việc phát hiện ra nó và thực hiện nó là rất khó và phức tạp.

7.3.3.2. Open/Closed

- Theo nguyên lý này, mỗi khi chúng ta thêm mới chức năng chúng ta nên viết class mới extend từ class đã có chứ không nên chỉnh sửa trực tiếp nội dung trên class đã viết.
- Sau khi đã hiểu về nguyên lý này, bạn dễ dàng nhận thấy chúng ta đã tuân thủ nguyên tắc này khi thiết kế subsystem cho interbank. Sau này mở rộng, chúng ta có thể sử dụng interbank khác với giao thức kết nối và API thay đổi thì chúng ta chỉ cần viết một subsystem khác implement các phương thức payOrder, refund ... Điều này đáp ứng được tính open/closed, tức là thay đổi yêu cầu mà không phải sửa lại thiết kế cũ.



- Ngoài ra, đối với thiết kế cũ, bạn có thể thấy lớp PlaceOrderController có một phương thức là calculateShippingFee như sau:



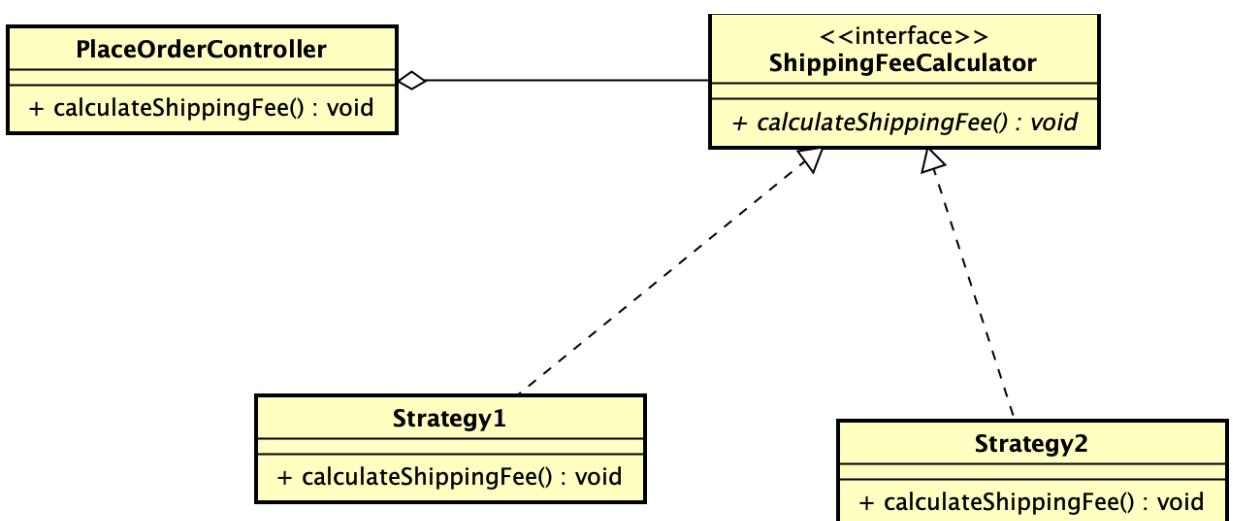
- Sau khi implement code, chúng ta có thể dễ dàng nhận thấy, sau này, nếu ta muốn tính phí ship theo kiểu khác, hoặc cần lưu nhiều kiểu tính phí ship khác nhau, thì chúng ta sẽ phải sửa đoạn code trên bằng một đoạn code với cách tính hoàn toàn khác. Điều này vi phạm nguyên tắc open/closed

```

public int calculateShippingFee(Order order){
    Random rand = new Random();
    int fees = (int)( (rand.nextFloat()*10)/100 ) * order.getAmount();
    LOGGER.info("Order Amount: " + order.getAmount() + " -- Shipping Fees: " + fees);
    return fees;
}

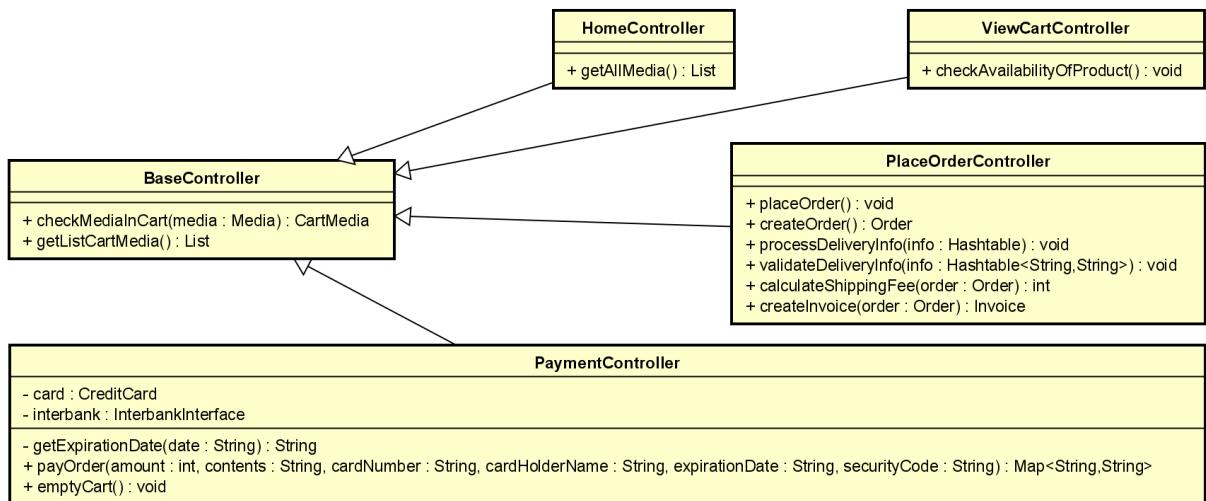
```

- Giải pháp: ta sẽ tạo ra một interface ShippingFeeCalculator với phương thức trừu tượng là calculateShippingFee. Khi chúng ta muốn thêm một hay nhiều cách tính phí ship thì chúng ta chỉ cần viết thêm một lớp mới implement interface trên. Và tại PlaceOrderController chúng ta khởi tạo một đối tượng là interface ShippingFeeCalculator với instance là loại chiến lược tính phí ship mà chúng ta muốn. Vậy là đã đáp ứng được nguyên lý open/closed.

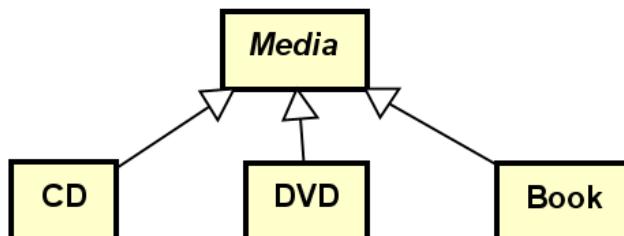


7.3.3.3. Liskov Substitution

- Nguyên tắc này nói rằng, các đối tượng của class con có thể thay thế cho lớp cha ở mọi tính huống mà không gây ra lỗi, hoặc nếu không, chúng ta có sự trùng tượng sai.
- Có thể thấy hệ thống phân cấp kế thừa ở class BaseController đã tuân theo nguyên lý này.



- Nay hãy nhìn vào mã nguồn và xem xét hệ thống phân cấp cây kế thừa của Media có vi phạm nguyên tắc này không?



- Ta có thể thấy phương thức Media.getAllMedia() được kì vọng trả về một List, tất cả các class con override lại phương thức này nhưng lại trả về null Class Media:

```

public List getAllMedia() throws SQLException{
    Statement stm = AIMSDB.getConnection().createStatement();
    ResultSet res = stm.executeQuery("select * from Media");
    ArrayList medium = new ArrayList<>();
    while (res.next()) {
        Media media = new Media()
            .setId(res.getInt("id"))
            .setTitle(res.getString("title"))
            .setQuantity(res.getInt("quantity"))
            .setCategory(res.getString("category"))
            .setMediaURL(res.getString("imageUrl"))
            .setPrice(res.getInt("price"))
            .setType(res.getString("type"));
        medium.add(media);
    }
    return medium;
}

```

Class DVD:

```

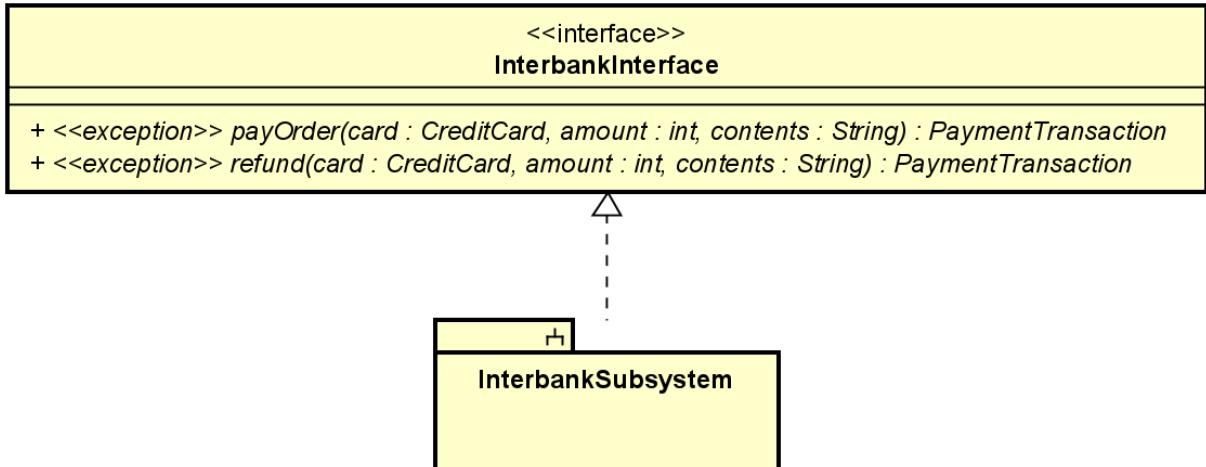
@Override
public List getAllMedia() {
    return null;
}

```

- Giải pháp: xoá đoạn code Override đi. Chương trình sau đó vẫn chạy bình thường vì phương thức getAllMedia() của lớp cha Media không phải phương thức abstract, không bắt buộc phải override.

7.3.3.4. Interface Segregation

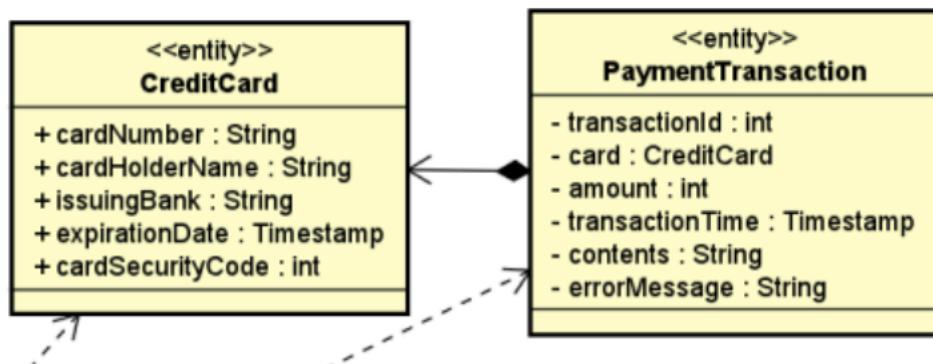
- Nguyên lý này nói rằng, thay vì một sử dụng một interface quá lớn, quá nhiều phương thức thì chúng ta sẽ tách nhỏ ra thành các interface con với mục đích cụ thể. Vì khi để một interface quá to, các lớp implement sẽ phải implement các phương thức mà bản thân nó không cần dùng đến.



- Thiết kế hiện tại về cơ bản đã đáp ứng được nguyên tắc này, ví dụ với InterbankInterface, cả 2 phương thức payOrder và refund đều được lớp InterfaceSystemController implement.
- Tuy nhiên, cũng cần phải cân nhắc, bởi trong tương lai, có thể có một số hệ thống interbank khác không hoàn tiền cho khách hàng mà chỉ thanh toán, lúc này phương thức refund của InterbankInterface trở nên dư thừa đối với interbanksubsystem đó → vi phạm Interface Segregation.

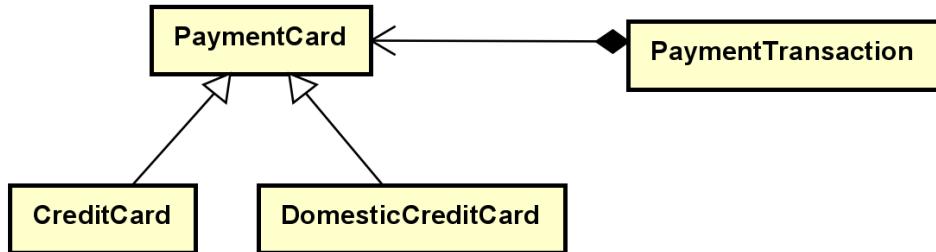
7.3.3.5. Dependency Inversion

- Có thể hiểu nguyên lý này như sau: những thành phần trong một chương trình chỉ nên phụ thuộc vào những cái trừu tượng. Những thành phần trừu tượng không nên phụ thuộc vào một thành phần mang tính cụ thể mà nên ngược lại.



- Hiện tại, PaymentTransaction đang phụ thuộc chặt chẽ vào lớp CreditCard, sau này giả sử không sử dụng CreditCard để thanh toán mà sử dụng một loại phương thức thanh toán khác, ví dụ như domestic debit card... như vậy thiết kế hiện tại đã vi phạm nguyên lý D trong SOLID.

- Giải pháp: tạo một lớp abstract là PaymentCard và lớp PaymentTransaction chỉ quan tâm đến lớp PaymentMethod này.



- Dễ thấy, giải pháp trên chính là một ví dụ cho cách áp dụng Strategy Pattern vào bản thiết kế.

7.4. BÀI TẬP

Review Design sau khi đã thêm UC “Place Rush Order”

7.4.1. Coupling và Cohesion

Trong phần này bạn cần:

- Review lại các mức coupling và cohesion và xác định các mức độ này trong modules của thiết kế cho “UC Place Rush Order”. Nếu thiết kế chưa tốt, hãy đề xuất giải pháp cải thiện.
- Viết báo cáo cho các vấn đề trên và sửa lại thiết kế và source code tương ứng với đề xuất.

Sau khi hoàn thành xong, nộp lại kết quả vào thư mục “GoodDesign/DesignConcepts.”

Dưới đây là một báo cáo mẫu:

1. Coupling

1.1. Content coupling

Related modules	Description	Improvement

1.2. ...

2. Cohesion

2.1. Coincidental cohesion

Related modules	Description	Improvement

2.2. ...

7.4.2. Nguyên lý thiết kế SOLID

Trong phần này, bạn cần thực hiện:

- Kiểm tra xem design của bạn sau khi đã thêm UC “Place Rush Order” xem có đáp ứng nguyên lý SOLID hay không trước và sau khi thêm các yêu cầu mở rộng. Nếu thiết kế chưa tốt, hãy đề xuất giải pháp giúp cho thiết kế tốt hơn để đáp ứng được với 5 nguyên lý SOLID.
- Viết báo cáo về các vấn đề trên và chỉnh sửa code để implement đề xuất của bạn. Chú ý rằng bạn không cần phải implement các yêu cầu mở rộng. Bạn cần cân nhắc việc có cần thiết kế để đảm bảo có các yêu cầu mở rộng thì không vi phạm các nguyên lý trong SOLID.

Sau khi hoàn thành xong, nộp lại kết quả vào thư mục “GoodDesign/DesignPrinciples.”

Dưới đây là một báo cáo mẫu:

1. Single Responsibility Principle

#	Related modules	Description	Improvement
1.1.			

2. Open/Closed Principle

#	Related modules	Description	Improvement
2.1.			

...