TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Thiết kế và xây dựng phần mềm EcoBike Rental

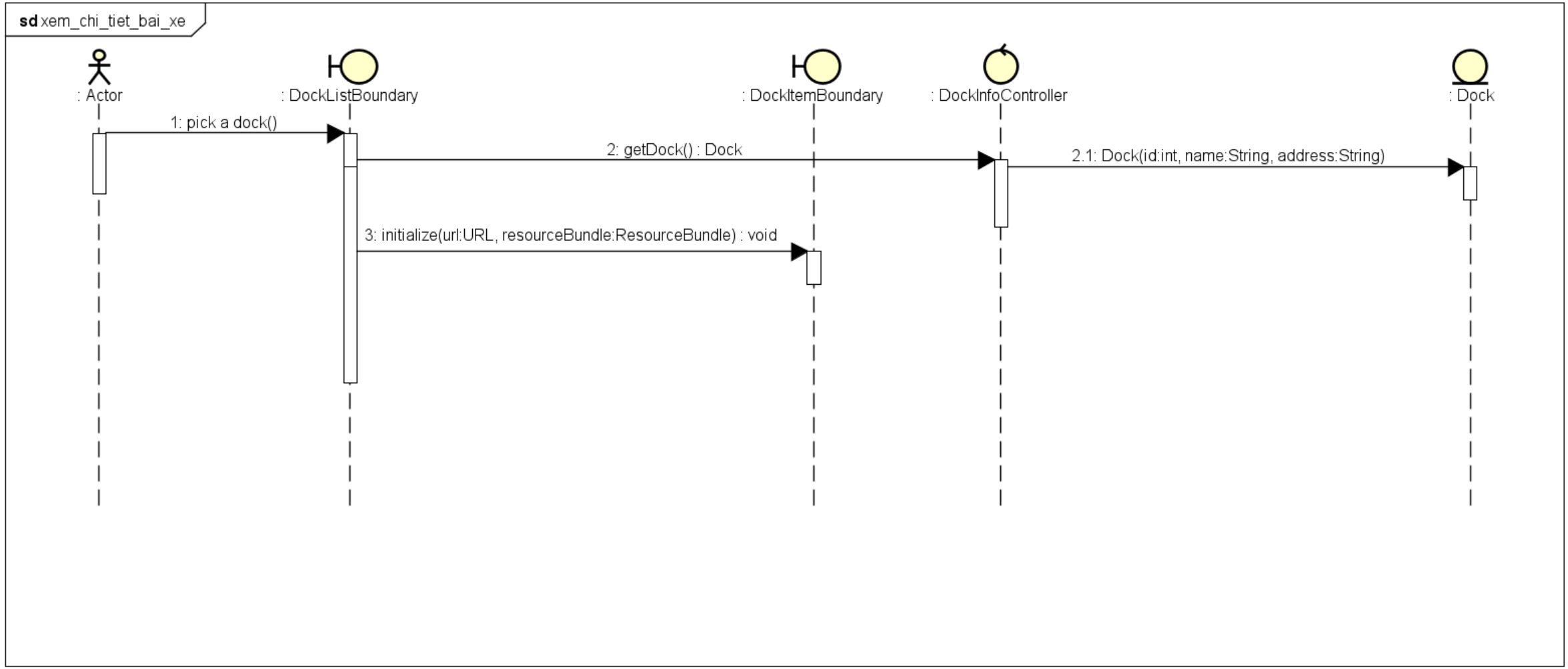
Group 10
Đặng Lâm San – 20170111
Nguyễn Mai Phương – 20170106
Dương Hồng Sơn - 20173347

Phân công công việc thành viên

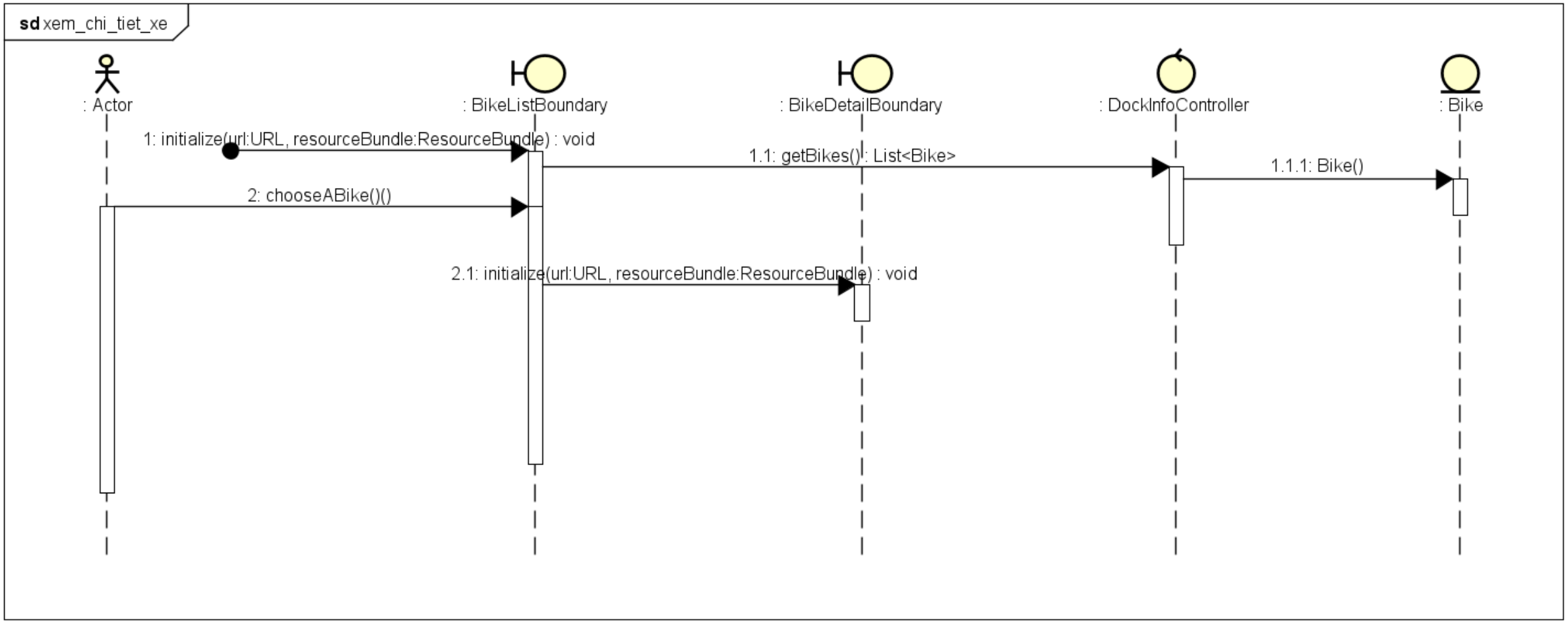
- Đặng Lâm San: SRS (Hoàn thành use case Trả xe, Chuyển đổi mã vạch, astah), SDD (Thiết kế subsystem, biểu đồ sequence), Demo, mã nguồn (package controller, subsystem, exception, database, entity), astah, thiết kế giao diện người dùng.
- Nguyễn Mai Phương: SRS (Hoàn thành use case Xem thông tin chi tiết bãi xe, Xem thông tin chi tiết xe, Xem thông tin chi tiết xe đang thuê, astah), SDD (Thiết kế giao diện người dùng, Design Considerations, Thiết kế chi tiết lớp), Test case spec, mã nguồn (package sample).
- Dương Hồng Sơn: SRS (Hoàn thành use case Thuê xe, Đặt cọc, astah), làm slide, SDD (Mô hình khái niệm dữ liệu, Thiết kế cơ sở dữ liệu, biểu đồ lớp), mã nguồn (DAO), Javadocs.

StudentId	StudentName	SRS	SDD	Programming	Testing	Summary
20170106	Nguyễn Mai Phương	Xem chi tiết bãi xe Xem chi tiết xe Xem xe đang thuê	Thiết kế giao diện người dùng Design Consideration.	Presentation Layer	Test case specification	35%
20170111	Đặng Lâm San	Trả xe Chuyển mã vạch	Thiết kế lớp, thiết kế Subsystem.	Business Layer, Data Layer, Subsystem	Test case implementation	35%
20173347	Dương Hồng Sơn	Thuê xe, Đặt cọc	Thiết kế mô hình dữ liệu. Thiết kế lớp, chi tiết lớp.	Data Layer		30%

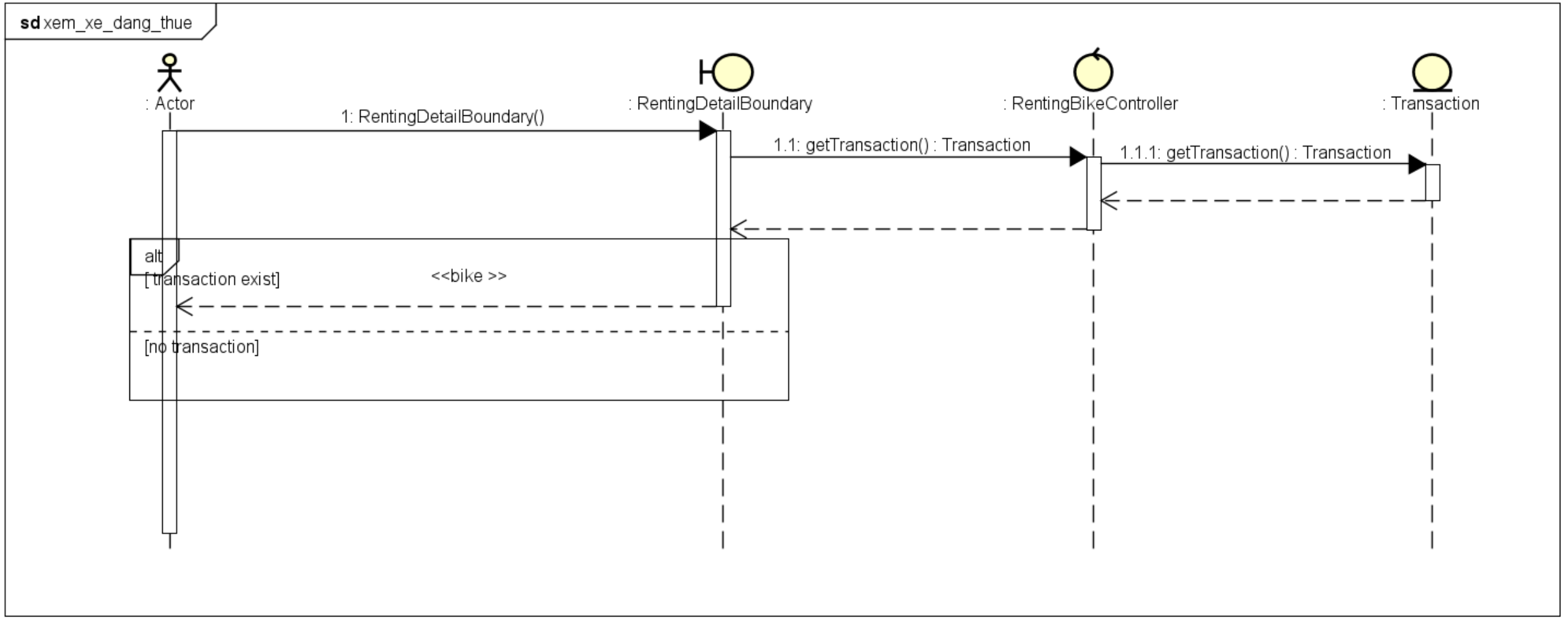
Biểu đồ tương tác UC Xem chi tiết bãi xe



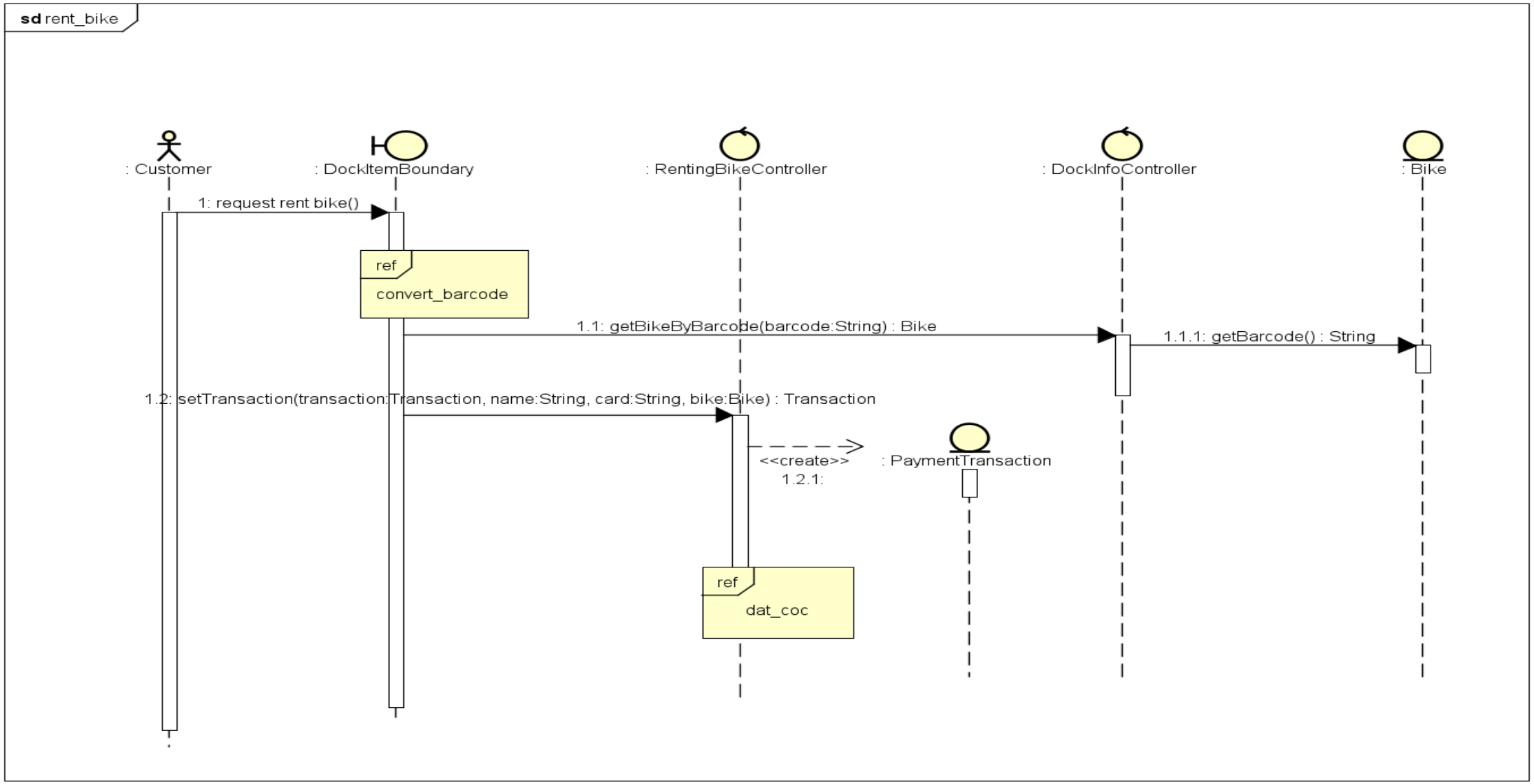
Biểu đồ tương tác UC Xem chi tiết xe



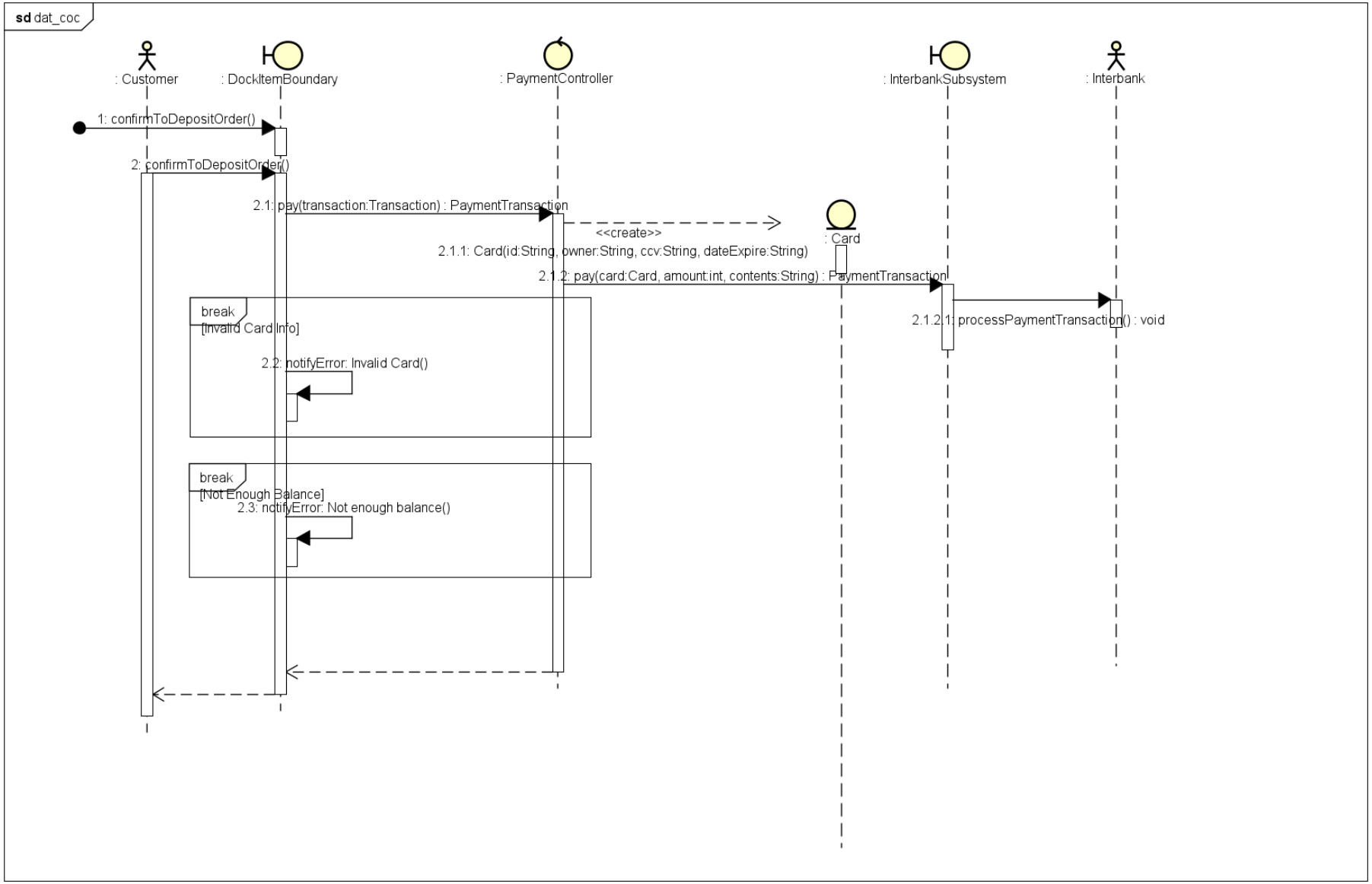
Biểu đồ tương tác UC Xem chi tiết xe đang thuê



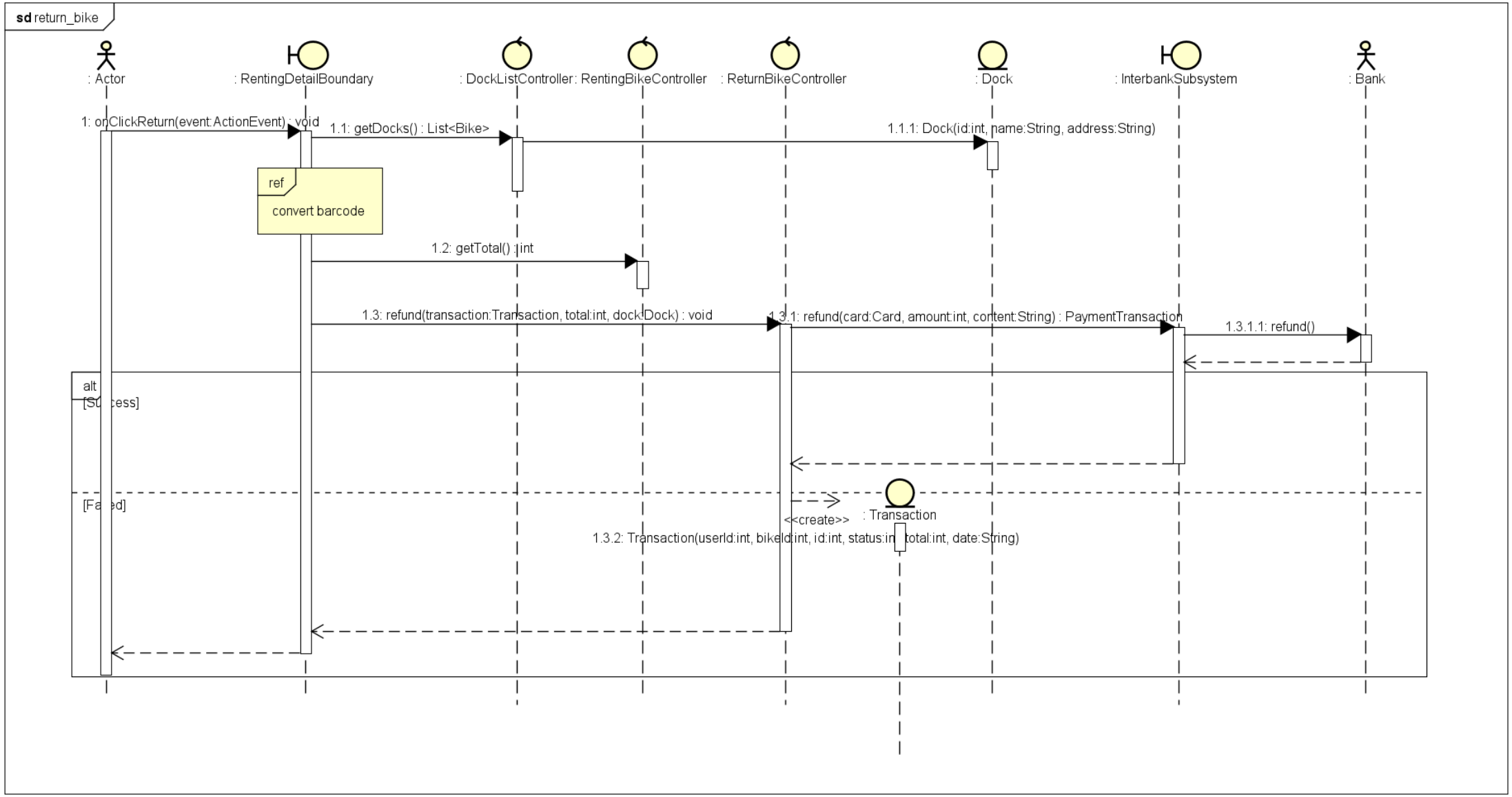
Biểu đồ tương tác UC Thuê xe



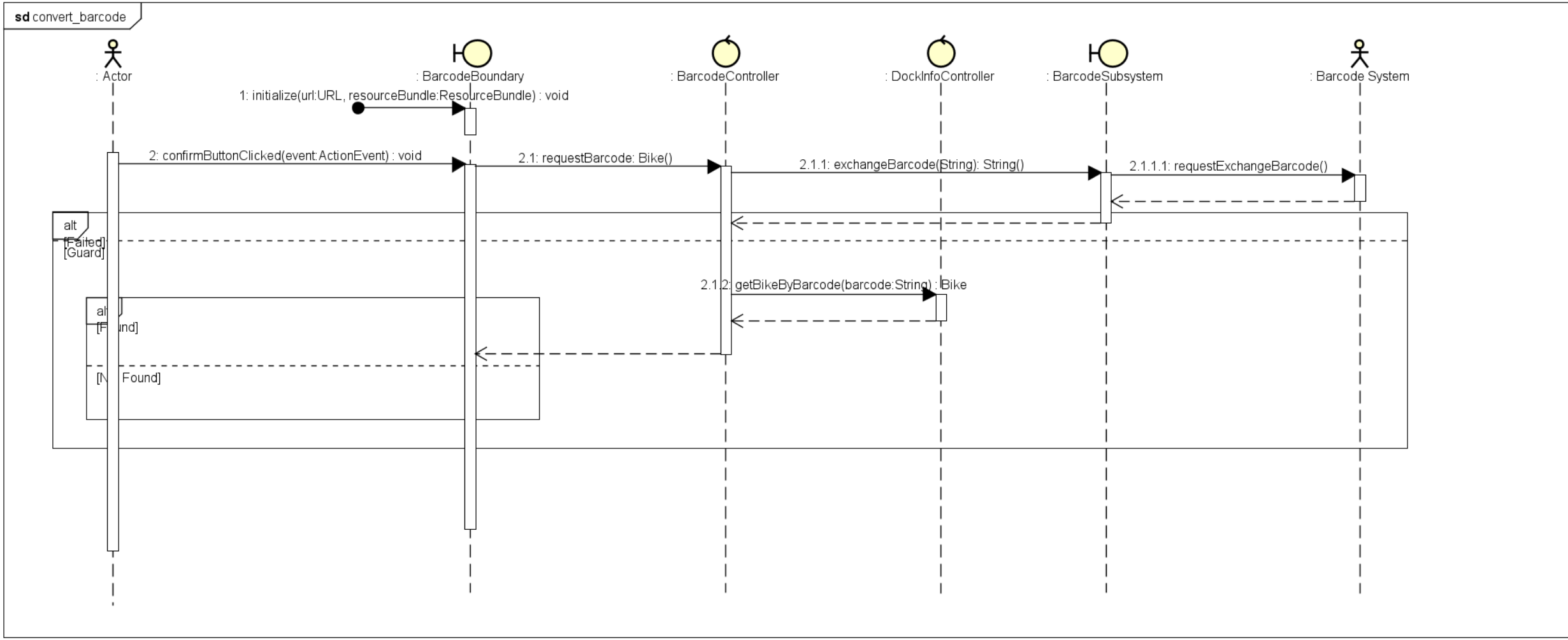
Biểu đồ tương tác UC Đặt cọc



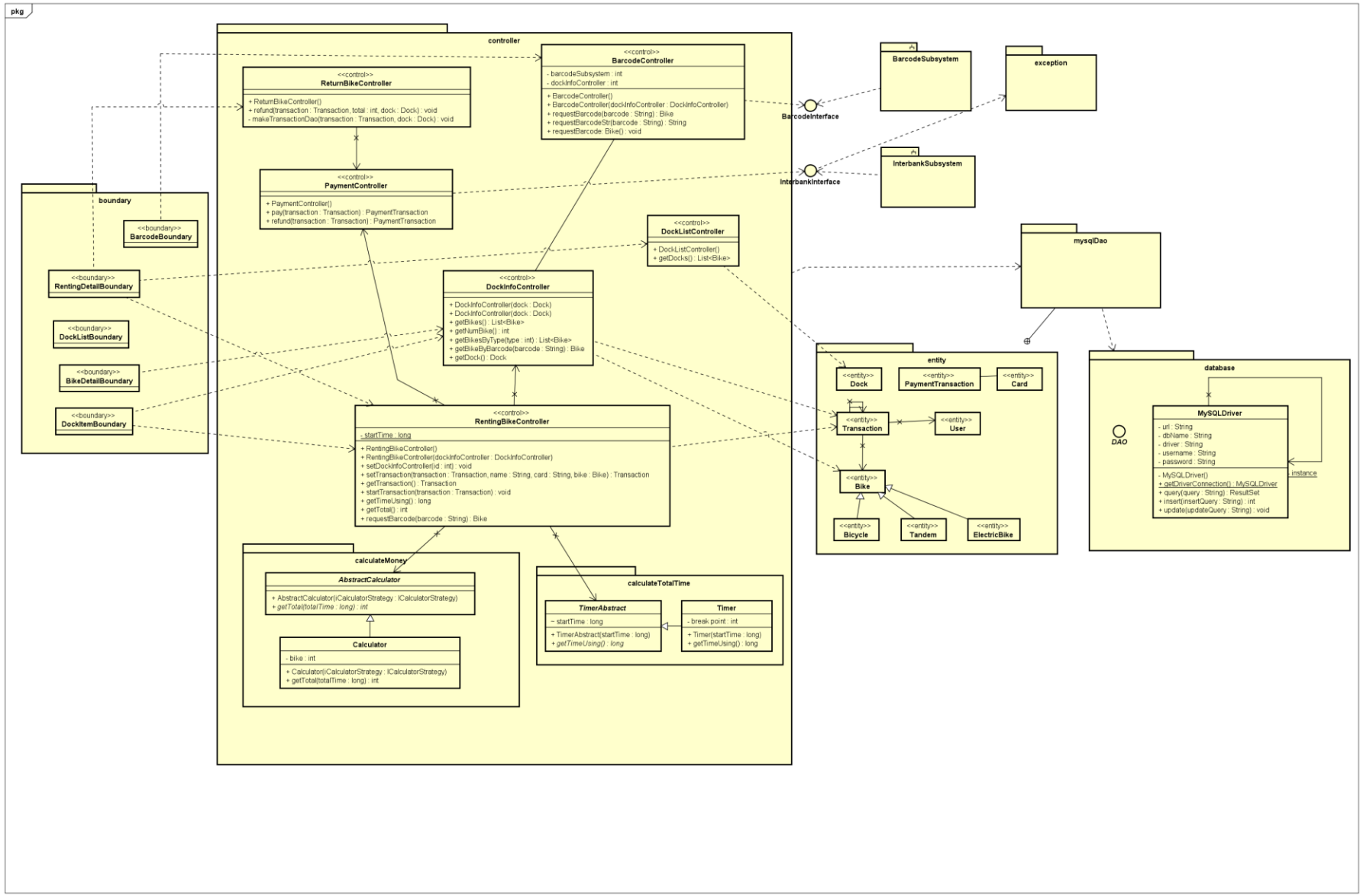
Biểu đồ tương tác UC Trả xe



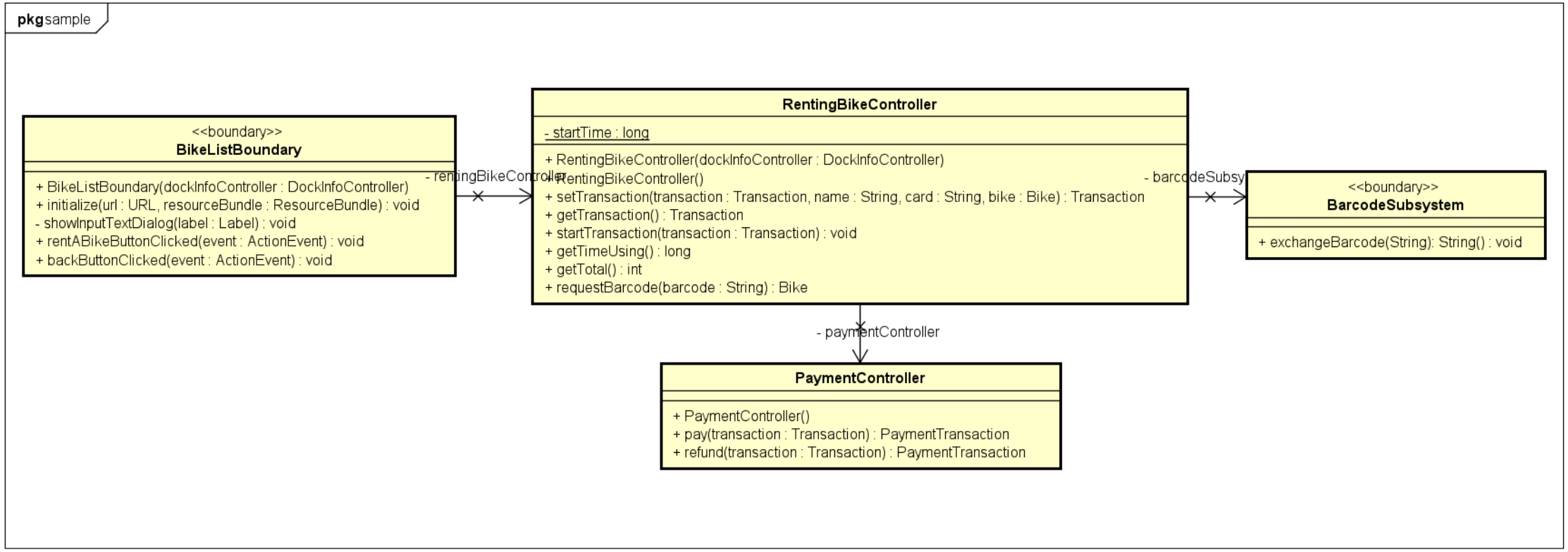
Biểu đồ tương tác UC Chuyển đổi barcode



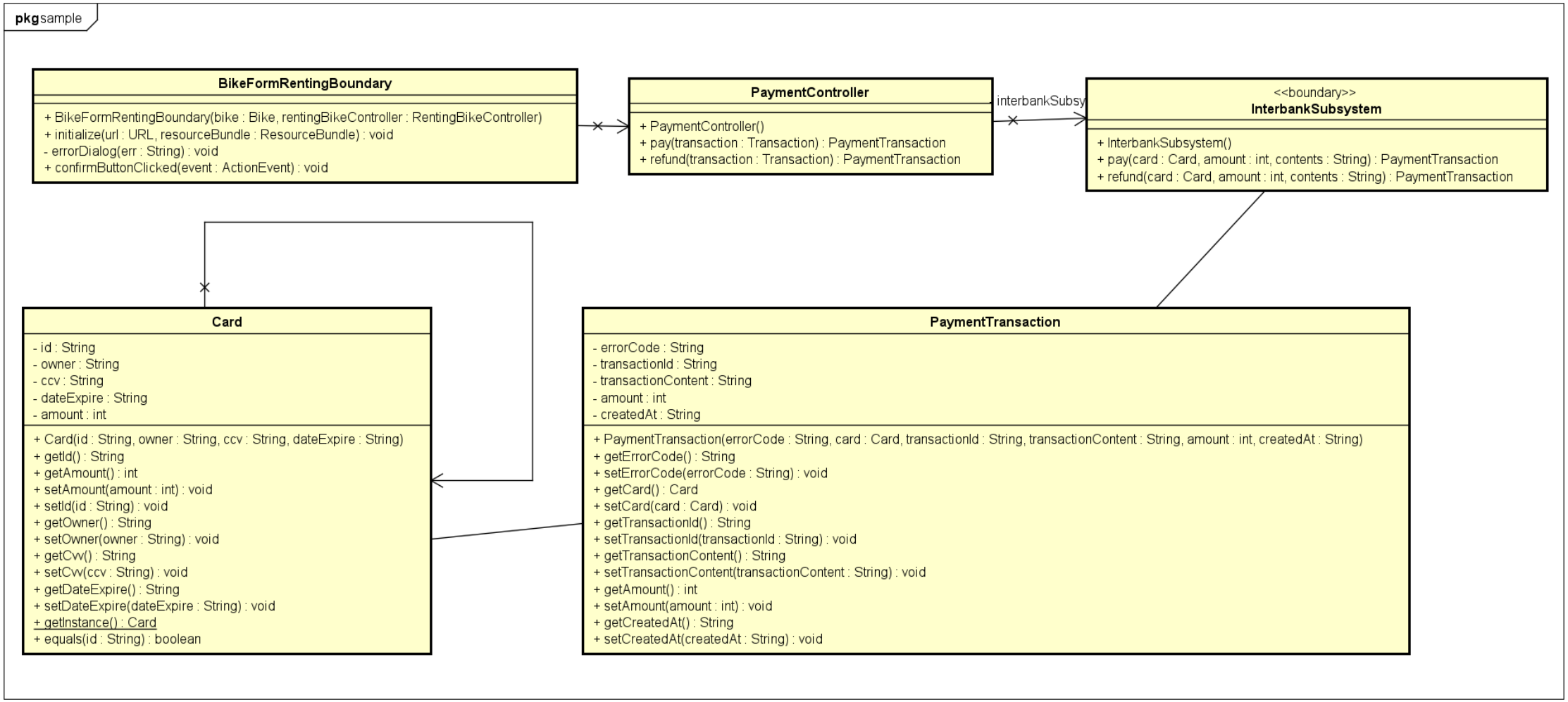
Biểu đồ lớp phân tích tổng quan



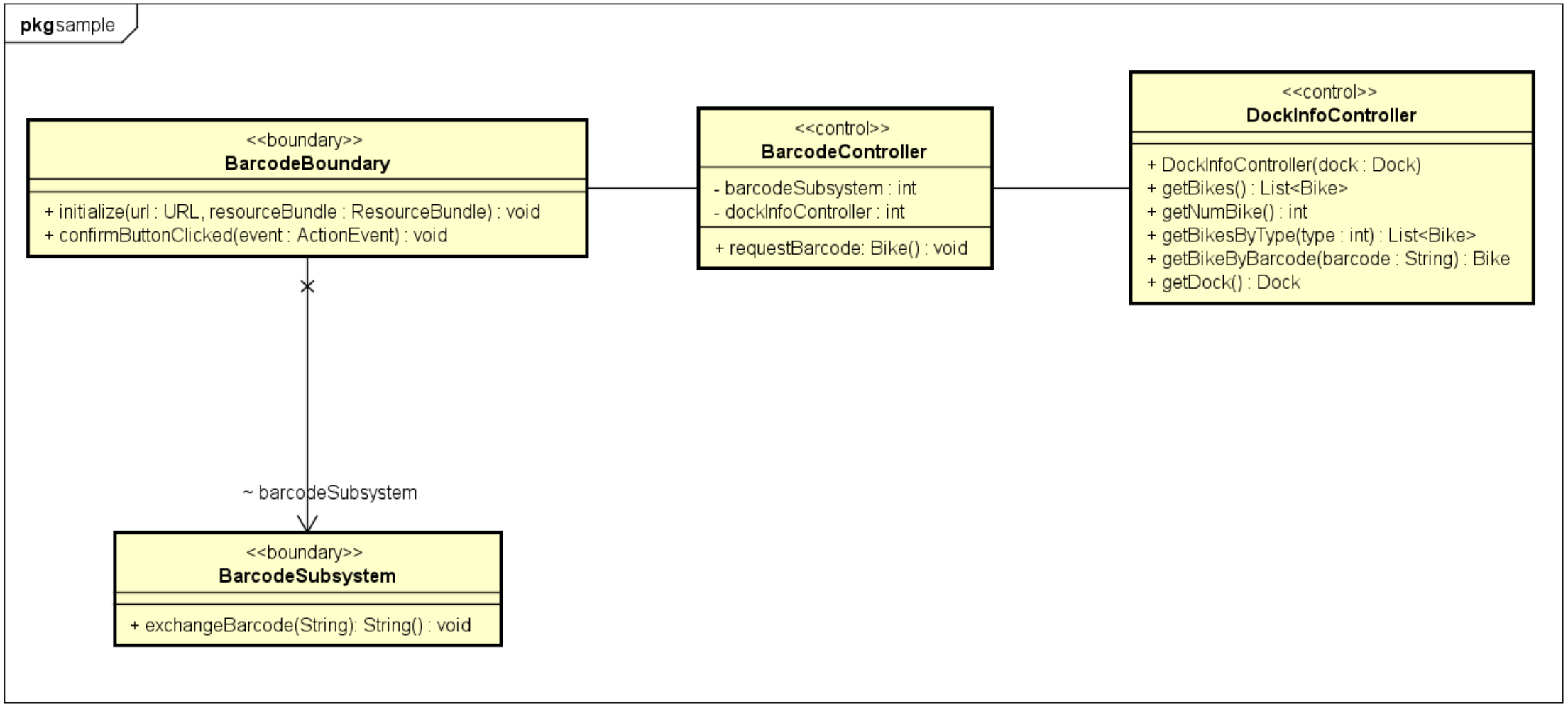
Class Diagram Rent Bike



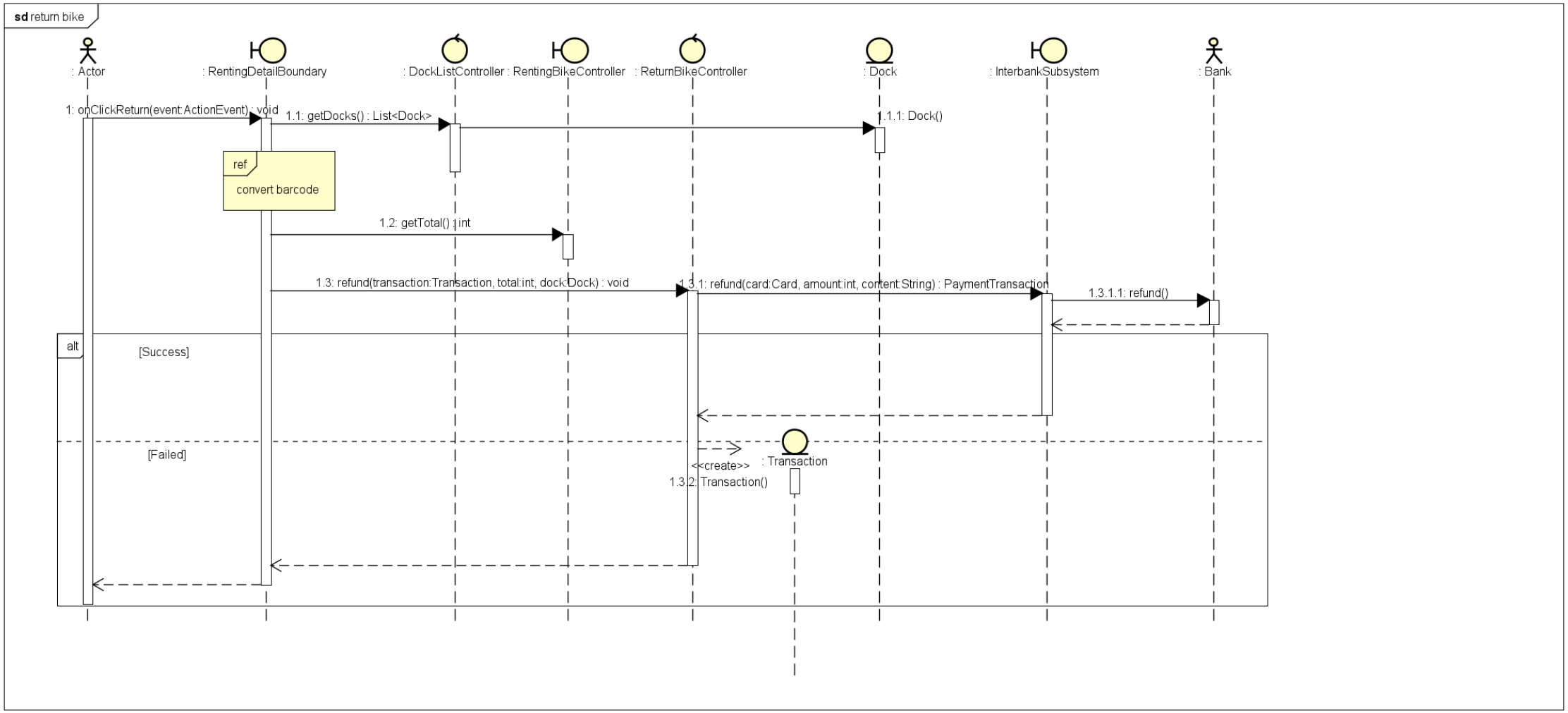
Class Diagram Deposit Order



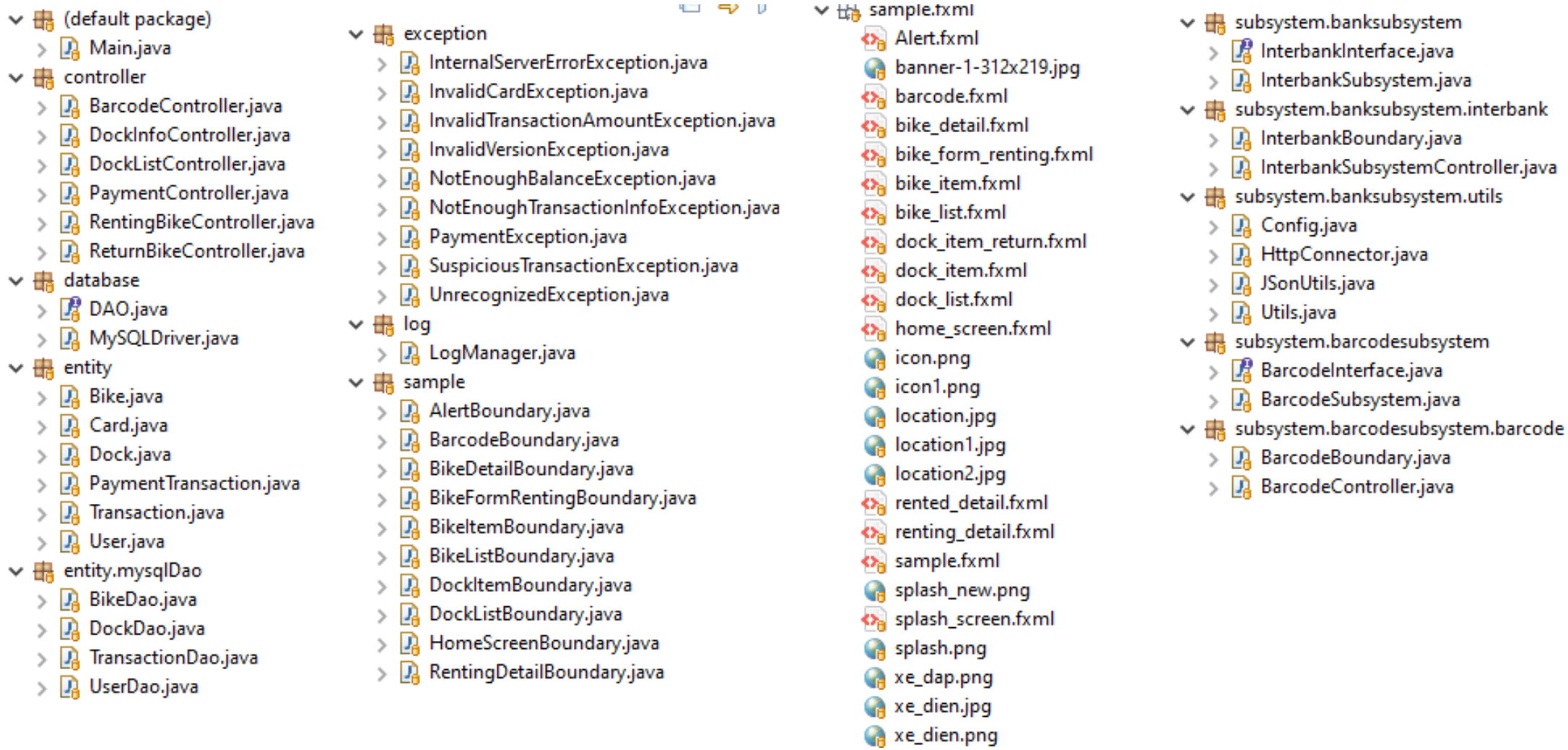
Class Diagram Convert Barcode



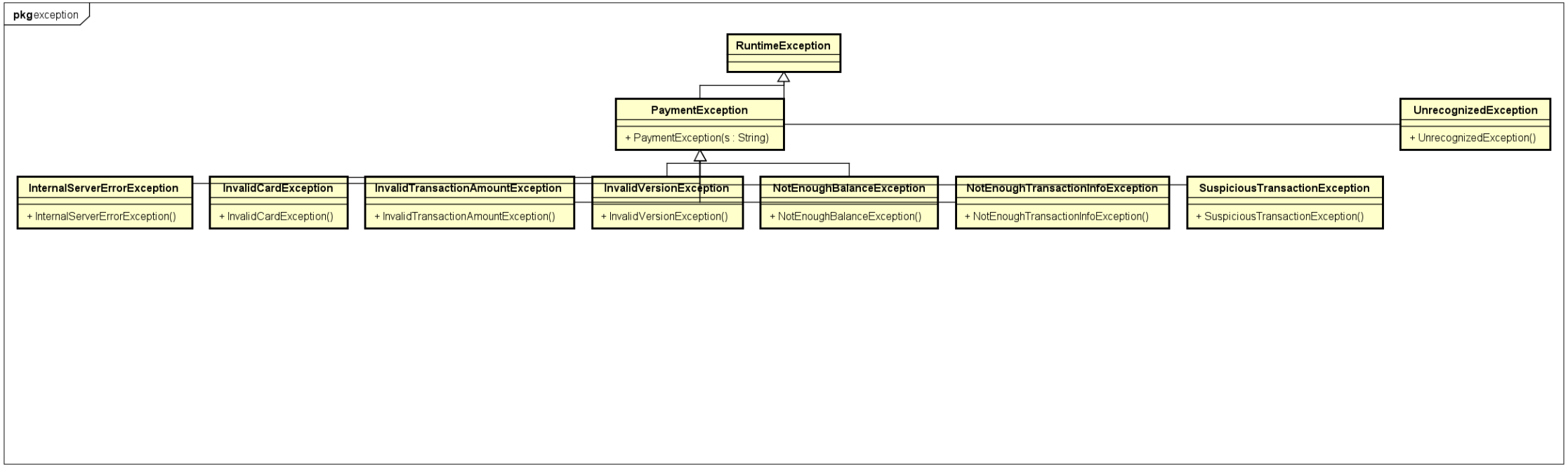
Class Diagram Return Bike



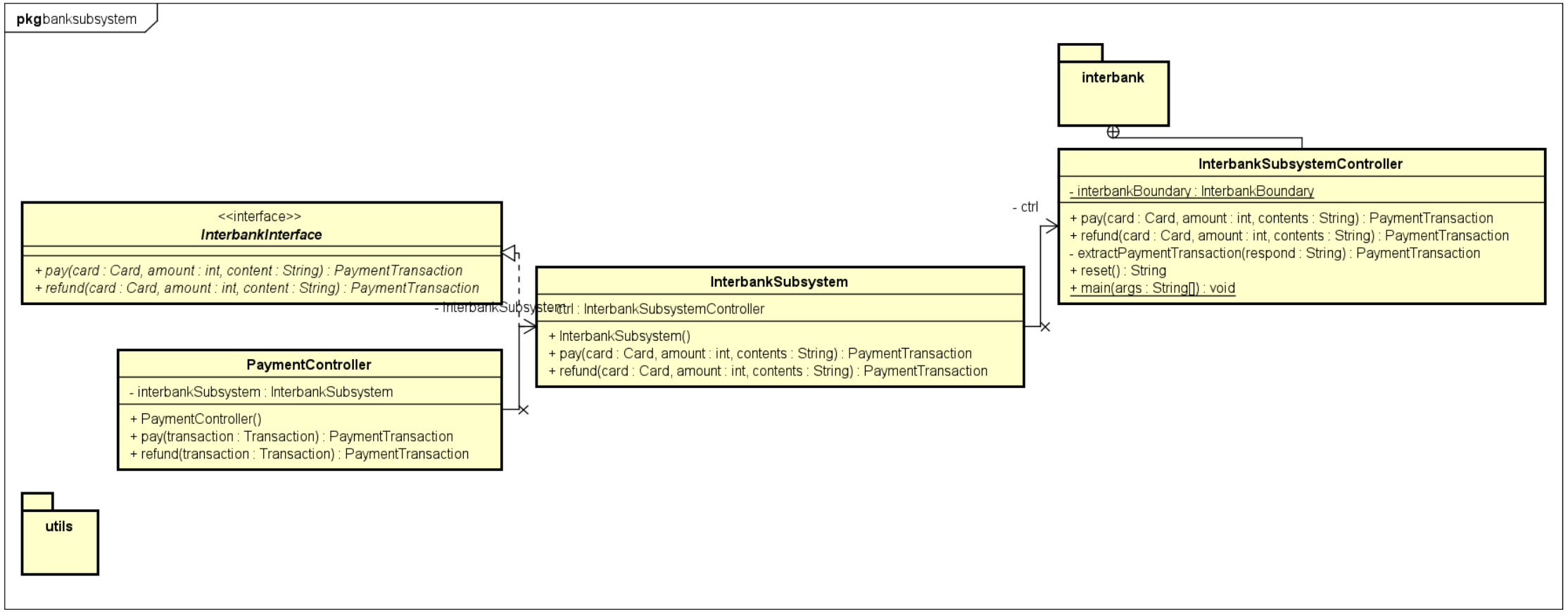
Nhóm các lớp thiết kế



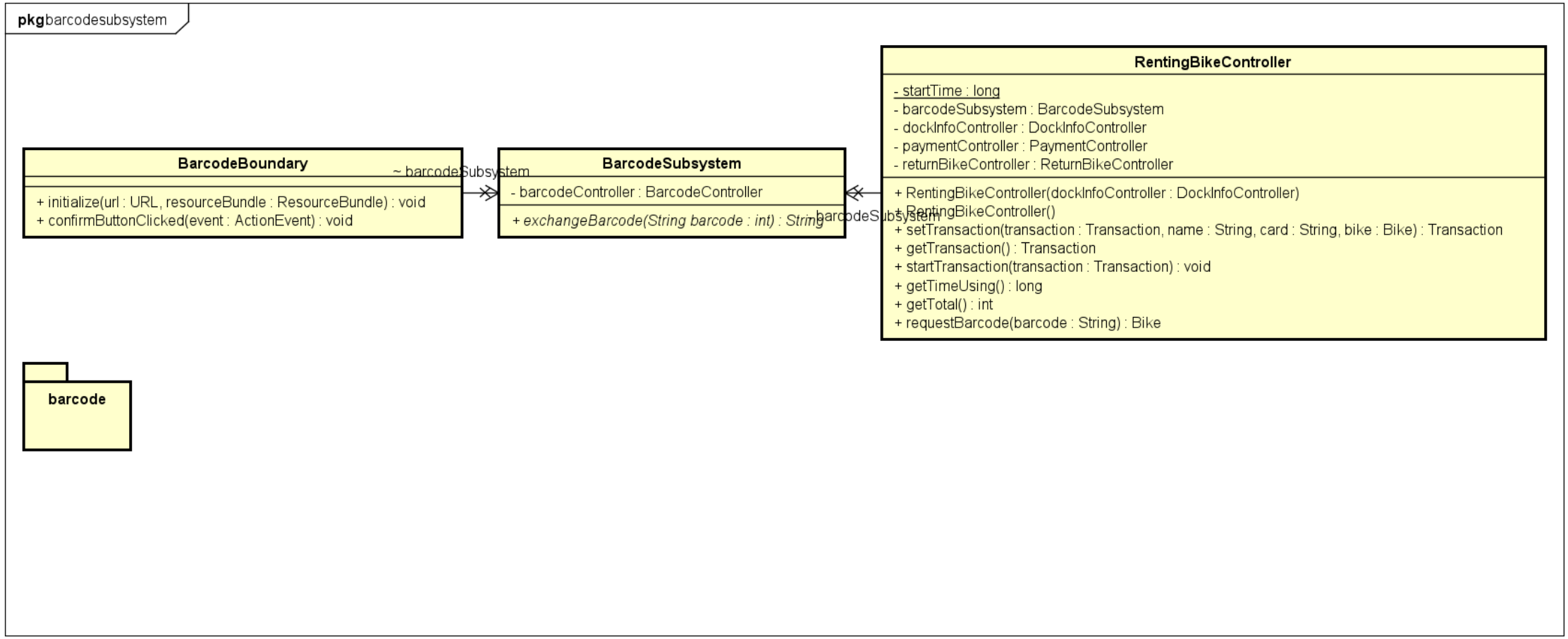
Biểu đồ lớp thiết kế của package exception



Biểu đồ lớp thiết kế của package bank subsystem



Biểu đồ lớp thiết kế của package barcode subsystem



Coupling/Cohesion

- Trong hệ thống này, tính kết dính - cohesion khá cao. Các module trong hệ thống được tách theo các vai trò mà nó quản lý, không có lớp nào mang nhiều hơn hai trách nhiệm nghiệp vụ. Cụ thể ví dụ như lớp ReturnBikeController chỉ có một phương thức là refund() và một phương thức phụ trợ việc lưu giao dịch vào cơ sở dữ liệu là makeTransactionDao()

- Bên cạnh đó, hệ thống chưa đạt được loose coupling. Các thành phần còn phụ thuộc vào nhau khá nhiều, ngoài ra một số module còn quản lý chung một số dữ liệu (Data Global) như PaymentTransaction. Cụ thể, RentingBikeController và ReturnBikeController đều có quyền chỉnh sửa trạng thái của lớp Singleton PaymentTransaction.

Design principles (1)

- Thiết kế của hệ thống đã cố gắng để tuân theo nguyên tắc trong SOLID. Tuy nhiên, do điều kiện kiến thức cũng như kỹ năng thực tế còn thiếu, nên chúng tôi phát triển và thiết kế phần mềm này cũng mắc phải một số lỗi. Các phần phía dưới bàn luận các thiết kế của hệ thống theo 5 nguyên tắc của SOLID.
- **Single Responsibility Principle:** Trách nhiệm của hệ thống được phân bổ tới các package, các subsystem và trong mỗi package, subsystem, trách nhiệm được chia nhỏ cho từng Class, mỗi Class đảm nhận một trách nhiệm duy nhất. Tuy nhiên, ta xem xét thử một lớp InterbankSubsystemController chịu trách nhiệm cho 2 nhiệm vụ: (1) điều khiển luồng dữ liệu (thanh toán, hoàn tiền), (2) chuyển đổi dữ liệu (chuyển đổi dữ liệu nhận về từ api sang dạng controller yêu cầu, hàm extractPaymentTransaction). 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 thayđổi (ví dụ như định dạng payment transaction thay đổi), có lẽ bản thiết kế nên được thay đổi để tốt hơn.

Design principles (2)

- **Open/Closed Principle**: Barcode subsystem implement các phương thức được định nghĩa trong Barcodeinterface. Các lớp của hệ thống chỉ phụ thuộc vào Barcode Interface chứ không phụ thuộc trực tiếp vào Barcode subsystem.

- Do đó, có thể dễ dàng thay thế subsystem sẵn có bằng một subsystem khác hoặc thêm một số phương thức khác cho Barcode interface và implement các phương thức này trong subsystem. Các thay đổi bên phía subsystem hoàn toàn trong suốt với các bên liên quan sử dụng giao diện của Barcode interface. Tương tự với Interbank subsystem.

Design principles (3)

- **Liskov substitution principle:** Không có
- **Interface segregation principle:** 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. Với BarcodeInterface cũng tương tự. Tuy nhiên, cũng cần phải cân nhắc một chút với InterfaceSystemController, 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.

Design principles (4)

- **Dependency Inversion principle:** 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 Card, sau này giả sử không sử dụng Card 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.

Design Pattern

- Singleton: Thiết kế áp dụng Singleton pattern cho lớp Transactionvà MySQLDriver.Với lớp Transaction, áp dụng thiết kế như vậy nhằm mục đích với mỗi khách hàng, cùng một thời điểm chỉ được đặt một xe. Điều này áp dụng tốt trong thực tế vì mỗi khách chỉ có thể dùng 1 xe tại một thời điểm, trách gây mất mát tài sản vì trong tương lai, hệ thống sẽ có thế có thểm chức nắng theo dõi vị trí của khách hàng khi thuê xe qua app điện thoại. Bên cạnh đó, lớp MySQLDriver chịu trách nhiệm quản lí liên kết với server database. Áp dụng pattern này cho MySQLDriver nhằm giúp hệ thống hoạt động tránh gặp lỗi hay xung đột nếu nhà phát triển chẳng may tạo nhiều thực thể MySQLDriver khác nhau tại nhiều vị trí trong phần mềm.

Design Pattern

- DAO - Data Access Object pattern: Data Access Object (DAO) Pattern là một trong những Pattern thuộc nhóm cấu trúc (Structural Pattern). Mẫu thiết kế DAO được sử dụng để phân tách logic lưu trữ dữ liệu trong một lớp riêng biệt. Theo cách này, các service được che dấu về cách các hoạt động cấp thấp để truy cập cơ sở dữ liệu được thực hiện. Nó còn được gọi là nguyên tắc Tách logic (Separation of Logic). Lớp Interface DAO là một interface định nghĩa các phương thức trừu tượng việc triển khai truy cập dữ liệu cơ bản cho BusinessObject để cho phép truy cập vào nguồn dữ liệu (DataSource). DockDAO, BikeDAO, TransactionDAO, UserDao cài đặt các phương thức được định nghĩa trong DAO, lớp này sẽ thao tác trực tiếp với nguồn dữ liệu (DataSource).
- Bridge Pattern: Được sử dụng cho phần nghiệp vụ tính giá tiền với đầu vào là thời gian sử dụng. Trong tương lai, cách tính tiền có thể thay đổi hoặc cách tính tiền của mỗi xe là khác nhau. Vì vậy áp dụng Bridge Pattern là một giải pháp.