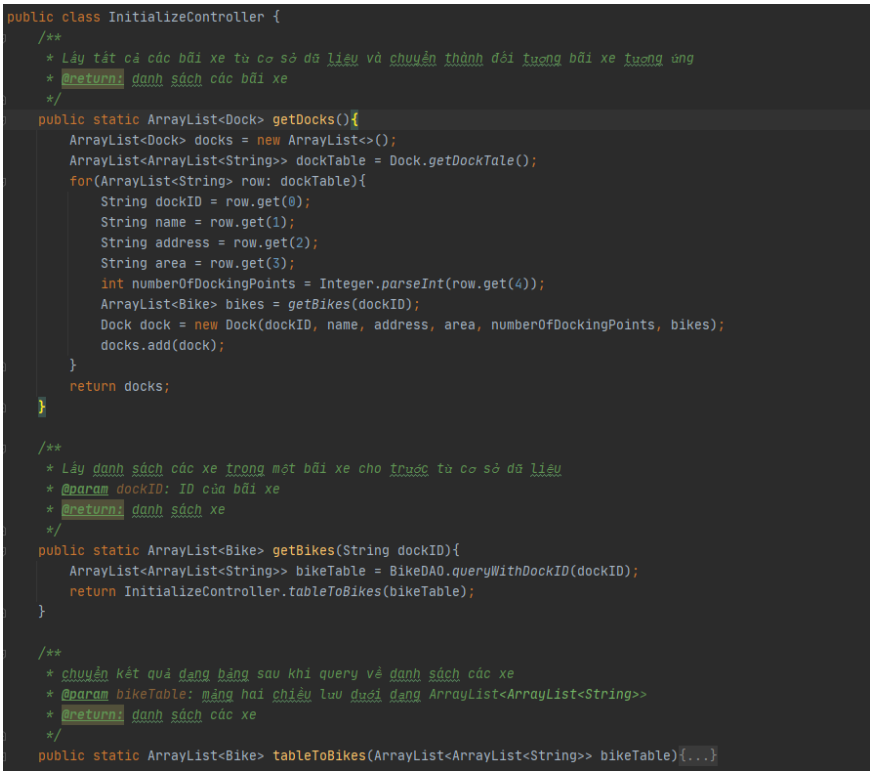
Phân tích tính Coupling và Cohesion

1 Phân tích tính Couping và Cohesion với gói “controller”

1.1 Phân tích tính Cohesion cho từng lớp

Lớp *InitializeController* thỏa mãn Functional Cohesion vì: Nhiệm vụ của class này là đọc dữ liệu từ database và trả về danh sách các bãi xe để *MainScreen* hiển thị khi chương trình bắt đầu. Để thực hiện nhiệm vụ này, lớp có một hàm chính là là *getDocks*(), hàm này có nhiệm vụ đọc dánh sách các bãi xe, để lấy danh sách xe của một bãi xe, *getDocks* gọi đến hàm *getBikes*, trong hàm *getBikes*, để chuyển dữ liệu dạng mảng hai chiều về danh sách các bãi xe, *getBikes* gọi đến hàm *tableToBikes*. Như vậy ba hàm *getDocks*, *getBikes* và *tableToBikes* có các chức năng riêng biệt nhưng cũng đồng thời đóng góp vào nhiệm vụ chung của lớp.



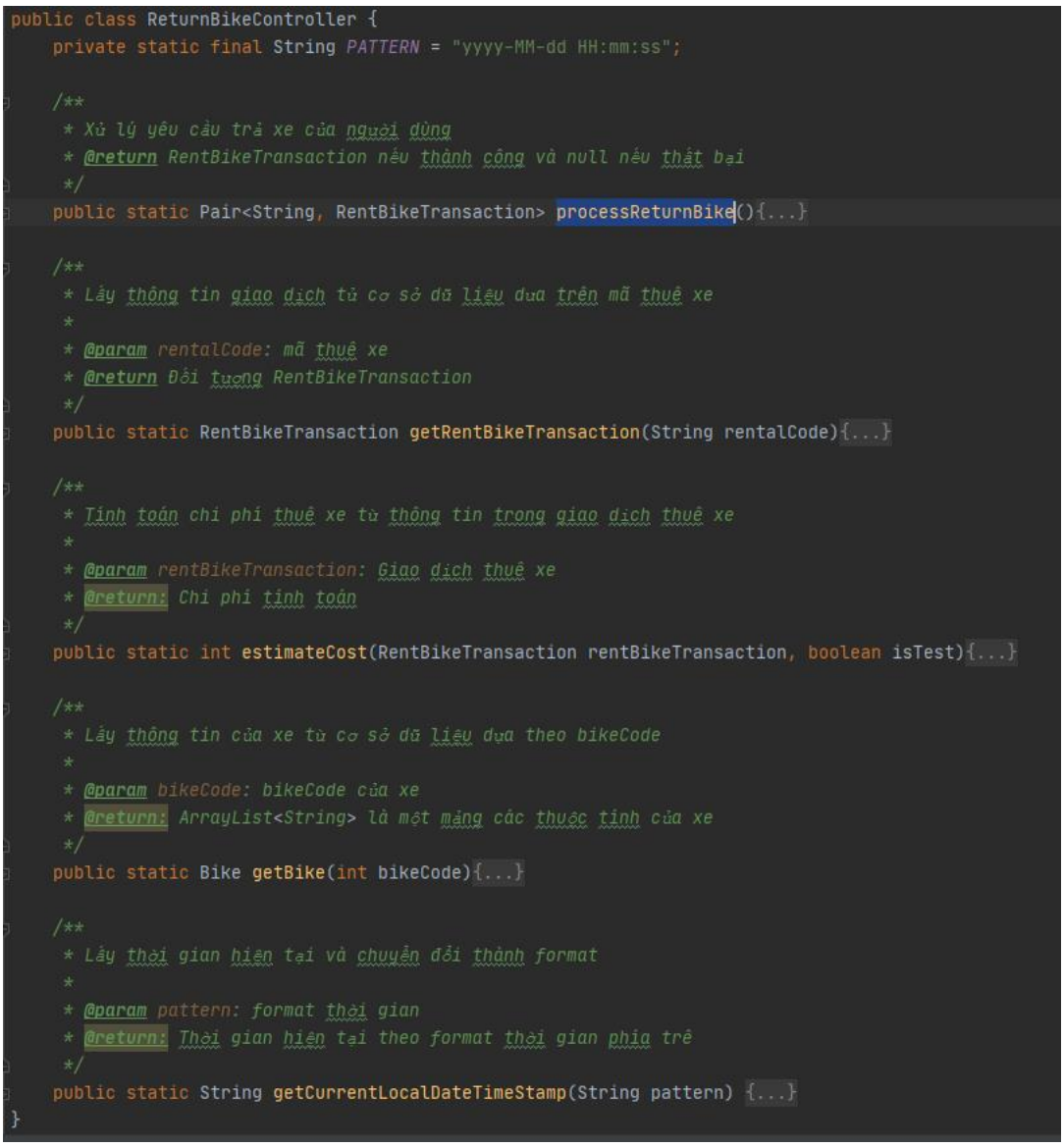
```
public class InitializeController {
    /**
     * Lấy tất cả các bãi xe từ cơ sở dữ liệu và chuyển thành đối tượng bãi xe tương ứng
     * @return: danh sách các bãi xe
     */
    public static ArrayList<Dock> getDocks(){
        ArrayList<Dock> docks = new ArrayList<>();
        ArrayList<ArrayList<String>> dockTable = Dock.getDockTale();
        for(ArrayList<String> row: dockTable){
            String dockID = row.get(0);
            String name = row.get(1);
            String address = row.get(2);
            String area = row.get(3);
            int numberOfDockingPoints = Integer.parseInt(row.get(4));
            ArrayList<Bike> bikes = getBikes(dockID);
            Dock dock = new Dock(dockID, name, address, area, numberOfDockingPoints, bikes);
            docks.add(dock);
        }
        return docks;
    }

    /**
     * Lấy danh sách các xe trong một bãi xe cho trước từ cơ sở dữ liệu
     * @param dockID: ID của bãi xe
     * @return: danh sách xe
     */
    public static ArrayList<Bike> getBikes(String dockID){
        ArrayList<ArrayList<String>> bikeTable = BikeDAO.queryWithDockID(dockID);
        return InitializeController.tableToBikes(bikeTable);
    }

    /**
     * chuyển kết quả dạng bảng sau khi query về danh sách các xe
     * @param bikeTable: mảng hai chiều lưu dưới dạng ArrayList<ArrayList<String>>
     * @return: danh sách các xe
     */
    public static ArrayList<Bike> tableToBikes(ArrayList<ArrayList<String>> bikeTable){...}
```

Lớp *ReturnBikeController* thỏa mãn tính Funcational Cohesion vì: Nhiệm vụ của class này là xử lý yêu cầu trả xe của người dùng, việc thực thi được chia làm các bước nhỏ:

- Lấy giao dịch thuê xe từ cơ sở dữ liệu
- Tính toán chi phí thuê xe
- Gửi yêu cầu interbanksubsystem để thực hiện giao dịch

- Nếu thành công:
    - Cập nhật chi phí thuê xe và thời gian trả xe vào giao dịch thuê xe
    - Cập nhật trạng thái của xe
    - Lưu lại giao dịch với ngân hàng vào cơ sở dữ liệu
    - Trả về giao dịch thuê xe và mã code tương ứng để hiển thị cho người dùng
- Nếu không thành công:
    - Trả về null và mã code tương ứng để hiển thị cho người dùng

Các bước trên đều nằm trong hàm chính *processReturnBike*. Để lấy giao dịch từ cơ sở dữ liệu, *processReturnBike* gọi đến hàm *getRentBikeTransaction*, để tính toán chi phí thuê xe gọi đến hàm *estimateCost*, để lấy thời gian trả xe theo đinh dạng gọi đến hàm *getCurrentLocalDateTimeStamp*, để cập nhật trạng thái xe thì cần tìm được xe tương ứng trong cơ sở dữ liệu, gọi đến hàm *getBike* để thực hiện điều này.



```
public class ReturnBikeController {
    private static final String PATTERN = "yyyy-MM-dd HH:mm:ss";

    /**
     * Xử lý yêu cầu trả xe của người dùng
     * @return RentBikeTransaction nếu thành công và null nếu thất bại
     */
    public static Pair<String, RentBikeTransaction> processReturnBike(){...}

    /**
     * Lấy thông tin giao dịch từ cơ sở dữ liệu dựa trên mã thuê xe
     *
     * @param rentalCode: mã thuê xe
     * @return Đối tượng RentBikeTransaction
     */
    public static RentBikeTransaction getRentBikeTransaction(String rentalCode){...}

    /**
     * Tính toán chi phí thuê xe từ thông tin trong giao dịch thuê xe
     *
     * @param rentBikeTransaction: Giao dịch thuê xe
     * @return: Chi phí tính toán
     */
    public static int estimateCost(RentBikeTransaction rentBikeTransaction, boolean isTest){...}

    /**
     * Lấy thông tin của xe từ cơ sở dữ liệu dựa theo bikeCode
     *
     * @param bikeCode: bikeCode của xe
     * @return: ArrayList<String> là một mảng các thuộc tính của xe
     */
    public static Bike getBike(int bikeCode){...}

    /**
     * Lấy thời gian hiện tại và chuyển đổi thành format
     *
     * @param pattern: format thời gian
     * @return: Thời gian hiện tại theo format thời gian phía trê
     */
    public static String getCurrentLocalDateTimeStamp(String pattern) {...}
}
```

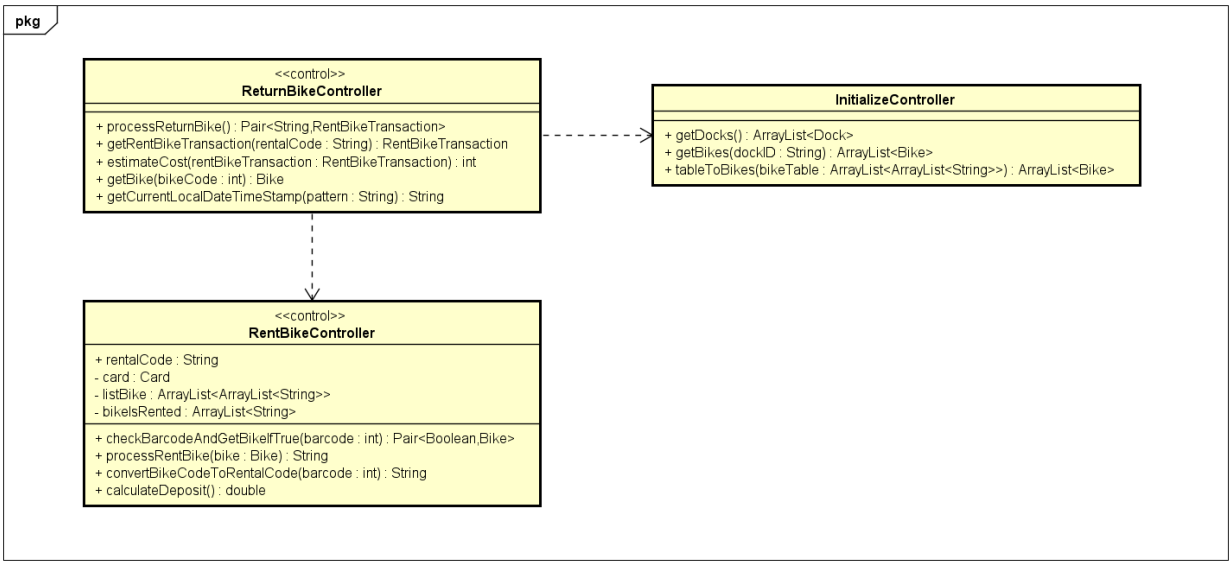
Lớp *RentBikeController* thỏa mãn tính Informational Cohesion vì: Nhiệm vụ của class là thực hiện xử lý yêu cầu thuê xe của người dùng. Việc thực thi được chia thành hai phase riêng biệt là kiểm tra barcode người dùng nhập vào có hợp lệ hay không lấy thông tin xe tương ứng nếu hợp lệ, nếu hợp lệ thì tiếp tục thực hiện phase thứ 2 để xử lý yêu cầu, cả hai phase đều tương tác với dữ liệu là các thuộc tính trong lớp *RentBikeController*. Cụ thể như sau:

- Hàm *checkBarcodeAndGetBikeIfTrue* kiểm tra *barcode* có hợp lệ hay không, nếu hợp lệ thì tìm xe tương ứng trong danh sách *listBike*.
- Hàm *processRentBike* thực hiện xử lý giao dịch sau khi đã kiểm tra *barcode* hợp lệ và có đối tượng xe tương ứng. Cụ thể gồm các bước:
    - Tính toán tiền đặt cọc
    - Chuyển *bikeCode* thành *rentalCode* và đặt lại giá trị thuộc tính *rentalCode*
    - Gửi yêu cầu thực hiện giao dịch trừ tiền đến *interbanksussystem*
    - Nếu thành công:
        - Tạo và lưu giao dịch thuê xe
        - Tạo và lưu giao dịch trừ tiền với ngân hàng
        - Cập nhật trạng thái xe
    - Nếu không thành công:
        - Đặt lại *rentalCode* về rỗng

Để tính toán tiền cọc, gọi đến hàm *calculateDeposit*, để chuyển *bikeCode* thành *rentalCode* gọi đến hàm *converterBikeCodeToRentalCode*, để chuyển *barcode* thành *bikeCode* gọi đến *barcodeconvertersubsystem*.

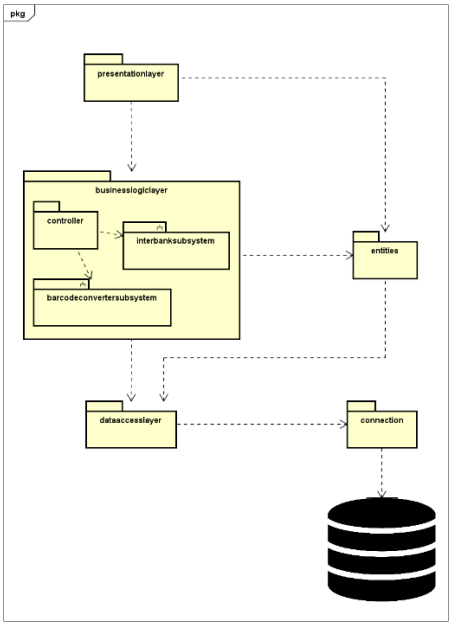
```java
public class RentBikeController {
    /**
     * rentalCode : Mã thuê xe được dùng cho toàn bộ những giao dịch và phiên thuê xe hiện tại.
     *              Phiên thuê xe khác nhau sẽ có rentalCode khác nhau.
     */
    public static String rentalCode = "";
    private static final Card card = Card.getInstance();
    private static ArrayList<ArrayList<String>> listBike = BikeDAO.getBikes();
    private static ArrayList<String> bikeIsRented;
    private static int bikeCode;

    /**
     *
     * @param barcode : mã xe
     * @return  <mã check bikeCode, thông tin xe (nếu bikeCode đúng)>
     */
    public static Pair<Boolean, Bike> checkBarcodeAndGetBikeIfTrue(int barcode){...}

    /**
     * Xử lý giao dịch thuê xe
     * Nếu giao dịch thành công thì sẽ tiến hành lưu lại giao dịch thanh toán,
     * thông tin phiên thuê xe và cập nhật xe thành đang sử dụng.
     *
     * Nếu giao dịch thất bại thì sẽ đưa ra thông báo lỗi và không lưu lại thông tin.
     */
    public static String processRentBike(Bike bike){...}

    /**
     *
     * @param bikeCode
     * Sinh ra rental code cho phiên thuê xe
     * @return rental code
     */
    public static String convertBikeCodeToRentalCode(int bikeCode){...}

    /**
     * @return giá trị tiền đặt cọc  = 40% giá trị xe
     */
    public static double calculateDeposit() { return Integer.parseInt(bikeIsRented.get(3)) * 0.4; }
}
```

1.2 Phân tích tính Coupling giữa 3 lớp controller



- Để tái sử dụng code, *ReturnBikeController* gọi hàm *tableToBikes* để chuyển danh sách các bãi xe dạng mảng hai chiều thành dánh sách đối tượng các xe. Đây là phụ thuộc data coupling do hai lớp chỉ giao tiếp với nhau thông qua dữ liệu
- Sự phụ thuộc giữa *ReturnBikeController* và *RentBikeController* là data coupling vì *ReturnBikeController* chỉ sử dụng biến toàn cục *rentalCode* của class *RentBikeController*

2 Phân tích tính Couping giữa các gói



Mỗi gói trong hệ thống có nhiệm vụ riêng biệt và việc phụ thuộc từ trên xuống dưới, không có gói ở tầng dưới phụ thuộc vào tầng trên, không có phụ thuộc vòng lặp.