VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
INFORMATION TECHNOLOGY - DEPARTMENT OF COMPUTER SCIENCE


## SOFTWARE ENGINEERING

# Assignment Project
# VINARENT CAR RENTAL SYSTEM

Lecturer: Dr.Nguyen Hong Quang
Team: HHTS-Solution

| No | Student name | Student ID | Responsible Task | Contribution |
|---|---|---|---|---|
| 1 | Nguyen Thi Mai Huong | ITITIU19128 | Task 3, 4, 5 | 50% |
| 2 | Mai Le Hung | ITITIU19125 | Task 1 | 20% |
| 3 | Nguyen Trung Truc | ITITIU19225 | Task 3 | 10% |
| 4 | Tran Thanh Son | ITITIU19201 | Task 2 | 20% |

Ho Chi Minh city, November 27th 2021

# Mục lục

# 1 Description for the whole system

## 1.1 Problem Statement

VinaRent is a company in **car rental business.**It has over 100 **branches**all over the country. It has about 5,000 **cars**and makes about 500,000 **rentals**per year. **The rentals spread across 100 branches**: branches in major airports, branches in major cities, and branches in local agencies such as hotels and garages. Each branch is identified by a branch number. VinaRent has several **IT systems.**We are concerned with one of them. This system is called **Vina Rental System.**Its main purpose is to**maintain information of the cars and the rentals of those cars.**It is also used to manage the car fleet, e.g. moving cars from branch to branch when necessary. It is decided that **the system**will be constructed **as a central system which has access to the information about every VinaRent branch** . Later on, when it is installed at a branch, it will be customized to restrict its access scope to an appropriate level.

***VinaRent Cars***

Each car has a model, identified by a model number. Useful information about a model includes a short description (which usually includes the model's marketing name), automatic or manual, petrol consumption (such as 1.5 liters or 6 cylinders), number of doors. The cars, on the basis of their model, are divided into five groups, group 'A' to group 'E', and all the cars in a group have the same rental price.

***VinaRent Customers***

VinaRent estimates that about 15,000 customers per year are served, of whom about 12% rent frequently, 40% rent between 3 to 5 times per year, and 50% are one-off renters in the sense that they use VinaRent once per year or less.

In fact, VinaRent shares a (much larger) customer information base with other businesses including various airlines and hotel providers. However, the sharing of customer information is done through a separate system. This is done so transparently (for both querying and updates) that we think of the shared customer base as part of the current system (for querying and adding customers).

*A blacklist of customers* is maintained by VinaRent. It is updated periodically based on the reports from the branches. The maintenance of the blacklist follows a separate process that we do not need to be concerned with. We can simply**assume that the list is available for querying.**

*A discount customer list* is also available. Customers on this list are given a 10%

discount for their car rents. The list is maintained by a separate system. We can simply**assume that the list is available for querying**.
Thus,**regarding the customers, a list of customers, a blacklist, and a list of customers entitled for discount are available for querying. In addition, the branches are also allowed to add new customers to the customer list.**

***VinaRent Car Rental Activities***

**The main activity at the branches is, of course, to rent cars on customers' requests. Customer can reserve a car.**They can also walk-in to request and pick up a car. Thus,the rentals involve the following activities: (1) answering customers' enquiries, (2) making reservations (3) recording walk-in rentals, (4) recording car pick-up, (5) recording returns of cars (may be at a different branch form the pick-up one, late returns possible), etc.

**The company's headquarter is responsible for purchasing and disposing off cars**. When a car is bought, its information (including its model if necessary) is entered into the system. Depending on their conditions and customer demand, cars may be removed from the fleet to be disposed off (e.g. though sales).

**HHTS-Solution decided to develop the VinaRent Car Rental System to manage the company base on business processes' discription will express in every tasks below.**

# 2 Task 1: Use cases

**Identify all the use cases, at the system level.**

## 2.1 Name of use cases

| Business Process | Step | Use case | Remarks |
|---|---|---|---|
| Reserving a Car (BP1.1a) | 2 | UC1: Gets the customer's requirements. | |
| | 3 | UC2: Search available cars | To find out if a car is available in this branch or need to get from a neighboring branch |
| | 5 | UC3: Update car's status | |
| | 6 | UC4: Check whether the customer is in the blacklisted or not | |
| | 7 | UC5: Add new customer | |
| | 8 | UC6: Creates a new car rental | |
| | | UC7: Record payment | |
| Picking-up a Car without Reservation (BP 1.1b) | 2 | Gets the customer's requirements. | Already had in BP1.1a |
| | | Search available cars | Already had in BP1.1a |
| | 3 | Check whether customer in the blacklisted or not | Already had in BP1.1a |
| | 4 | Creates a new car rental | Already had in BP1.1a |
| | 5 | Update car's status | Already had in BP1.1a |
| | ALTERNATIVE | Add new customer | Already had in BP1.1a |
| Moving Cars on Request (BP1.2) | 1 | UC8: Print out requested cars | |
| Picking-up a Reserved Car (BP1.3) | 1 | Update car's status | Already had in BP1.1a |
| | 3 | UC9: Check the license is match with license in rental | |
| | | Check whether customer in the blacklisted or not | Already had in BP1.1a |

| | | | |
|---|---|---|---|
| | | UC10: Change car rental information | |
| | 4 | UC11: Change payment information | |
| | 5 | Change car rental information | Already had in BP1.3 |
| | | Update car's status | Already had in BP1.1a |
| | ALTERNATIVE | Change car rental information | Already had in BP1.3 |
| | | Change car's status | Already had in BP1.1a |
| | | UC12: Make feedback | |
| Recording the Returning of a Car (BP1.4) | 1 | Change car rental information | Already had in BP1.3 |
| | 2 | UC13: Check whether the returned branch is the pick-up branch or not | |
| | 3 | Change car's status | Already had in BP1.1a |
| | | UC14: Update the car's residing branch | |
| Arranging for Car Maintenance (BP2) | 1 | UC15: Prints out the list of cars to be inspected | |
| | 2 | Update car's status | Already had in BP1.1A |
| | SUBSEQUENT ACTIVITIES | Update car's status | Already had in BP1.1A |
| Adding a New Car to the Active Pool (BP3) | 1 | UC16: Create a new model | |
| | 2 | UC17: Create a new car | |

## 2.2 Use case descriptions

**Use Case 1: Gets the customer's requirements.**
Triggers/ Goals: Clerk want to get the customer's requirements
Actors:Clerk, Operator
Main flow:

1. Clerk clicks the button on the screen to create a new reqirement
2. Clerk enters Pick-up Branch and Return Branch
3. System validates Pick-up Branch and Return Branch is valid
4. Clerk enters Pick-up Date, Time and Return Date, Time
5. System validates Pick-up Date, Time and Return Date, Time is valid
6. Clerk enters car is economy or not
7. System validates detail is valid
8. Clerk enters compact car

9. System validates detail is valid
10. Clerk enters the size of car
11. System validates the size is valid
12. Clerk enters car is station wagon or not
13. System validates detail is valid
14. Clerk enters car is van or not
15. System validates detail is valid
16. Clerk enters car is 4 wheels
17. System validates detail is valid
18. Clerk enters car is Sports or Luxury Car
19. System validates detail is valid
20. System stores the requirement details into database

Extension:

3a. If Pick-up Branch and Return Branch is invalid
1. System notifies clerk
2. Clerk checks and enters Pick-up Branch and Return Branch again or ABORT

5a. If Pick-up Date, Time and Return Date, Time is invalid
1. System notifies clerk
2. Clerk checks and enters Pick-up Date, Time and Return Date, Time again or ABORT

7a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

9a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

11a. If the size is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

13a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

15a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

17a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

19a. If the detail is invalid
1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

**Use Case 2: Search available cars.**

Triggers/ Goals: Clerk want to find out available cars for customer requirement.

Invocation Constraints: Clerk has already gotten customer's requirements

Actors: Clerk, Operator

Main flow:

1. Clerk enters car is economy or not
2. System validates detail is valid
3. Clerk enters compact car
4. System validates detail is valid
5. Clerk enters the size of car
6. System validates the size is valid
7. Clerk enters car is station wagon or not
8. System validates detail is valid
9. Clerk enters car is van or not
10. System validates detail is valid
11. Clerk enters car is 4 wheels
12. System validates detail is valid
13. Clerk enters car is Sports or Luxury Car
14. System validates detail is valid
15. System gets all the car of that type and has status is "RENT-READY" which are at the requested branch or a neighboring branch and the cost
16. System prints out the list of available cars for clerk with their cost

Extension:

2a. If the detail is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

4a. If the detail is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

6a. If the size is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

8a. If the detail is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

10a. If the detail is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

12a. If the detail is invalid
   1. System notifies clerk
   2. Clerk checks and enters detail again or ABORT

14a. If the detail is invalid

1. System notifies clerk
2. Clerk checks and enters detail again or ABORT

**Use Case 3: Update car's status.**
Triggers/ Goals: Clerk or Supervisor wants to update the status of a car.
Actors: Clerk, Supervisor, Operator
Main flow:
1. Clerk or Supervisor chooses a car on the screen
2. Clerk or Supervisor clicks on the edit button beside the status line
3. Clerk or Supervisor enters new status
4. System validates the status is valid
5. System stores the changes into database

Extension:
4a. If the new status is not valid
1. System notifies the clerk
2. System prints out all types of status that is valid for the clerk
3. Clerk or Supervisor enters the status again

**Use Case 4: Check whether customer in the blacklisted or not.**
Triggers/ Goals: Clerk wants to check if the customer in backlist or not.
Actors: Clerk, Operator
Main flow:
1. Clerk enters the customer's license number
2. System validates the customer's license number is valid
3. System confirms that the license is not in blacklist
4. System notifies for clerk that the license is not in blacklist

Extension:
2a. If the the customer's license number is invalid
1. System notifies clerk
2. ABORT

3a. If the license is already in the blacklist
1. System notifies to clerk that the license has been already in blacklist
2. ABORT

**Use Case 5: Add new customer.**
Triggers/ Goals: Clerk wants to add a new customer to the customer base.
Invocation Constraints: Customer has already ordered .
Actors: : Clerk, Operator
Main flow:
1. Clerk clicks the add new customer button on the screen 2. Clerk enters customer's driver license

3. System validates the driver license is valid
4. Clerk enters the customer's first name, last name
5. System checks that the name is not empty
6. Clerk enters customer's email
7. System validates the email is valid
8. Clerk enters customer's phone number
9. System validates the phone number is valid
10. System stores customer's information into the customer base

Extension:

3a. If the driver license is invalid
1. System notifies the clerk the license is invalid
2. Clerk enters the license again or ABORT

5a. If the name is empty
1. System notifies the clerk the name can not be empty
2. Clerk enters the information again or ABORT

7a. If the email is invalid
1. System notifies the clerk that the email is invalid
2. Clerk enters the email again or Email can be left blank

9a. If the phone number is invalid
1. System notifies the clerk that the phone nubmer is invalid
2. Clerk enters the phone number again or Phone number can be left blank

**Use Case 6: : Creates a new car rental.**

Triggers/ Goals: Clerk wants to create a new car rental.

Invocation Constraints: The customer was already checked that was not in blacklist and has already choosen a car that available for the rental.

Actors: : Clerk, Operator

Main flow:

1. Clerk click create a new rental on the screen
2. System prints out the retal screen, generates a unique rental number for this rental and asks Clerk to enter the information
3. Clerk enters the reserved car's number
4. System validates the reserved car'number is already existed
5. Clerk enters the pick-up and the return date, time
6. System validates the pick-up and the return date, time is valid
7. Clerk enters the pick-up and the return branch
8. System validates the pick-up, and the return branch is valid
9. Clerk checks the information again and presses enter
10. System sets the car rental's to "RESERVED"
11. System sets car's status to "RESERVED"
12. System stored the rental into database

Extension:

4a. If the reserved car's number is invalid

1. System notifies clerk
2. Clerk checks and enters the reserved car again or ABORT

6a. If the pick-up and the return date, time is invalid

1. System notifies clerk
2. Clerk checks and enters the pick-up and the return date, time again or ABORT

8a. If the pick-up, and the return branch is invalid

1. System notifies clerk
2. Clerk checks and enters the pick-up, and the return branch again or ABORT

**Use Case 7: : Record payment.**

Triggers/ Goals: Clerk wants to record the payment of customer.

Invocation Constraints: A new car rental has already been created and all information of the new car rental has already been entered to the system.

Actors: : Clerk, Operator

Main flow:

1. System caculates the payment amount
2. System prints out the payment amount for clerk
3. Customer pays the payment

—IF customer pays by credit card, do steps 4

4. Clerk makes a call to to another system to validate the payment
5. System records payment amount and payment method to the database

**Use Case 6: : Creates a new car rental.**

Triggers/ Goals: Clerk wants to create a new car rental.

Invocation Constraints: The customer was already checked that was not in blacklist and has already choosen a car that available for the rental.

Actors: : Clerk, Operator

Main flow:

1. Clerk click create a new rental on the screen
2. System prints out the retal screen, generates a unique rental number for this rental and asks Clerk to enter the information
3. Clerk enters the reserved car's number
4. System validates the reserved car'number is already existed
5. Clerk enters the pick-up and the return date, time
6. System validates the pick-up and the return date, time is valid
7. Clerk enters the pick-up and the return branch
8. System validates the pick-up, and the return branch is valid
9. Clerk checks the information again and presses enter
10. System sets the car rental's to "RESERVED"

11. System sets car's status to "RESERVED"
12. System stored the rental into database

Extension:

4a. If the reserved car's number is invalid
1. System notifies clerk
2. Clerk checks and enters the reserved car again or ABORT

6a. If the pick-up and the return date, time is invalid
1. System notifies clerk
2. Clerk checks and enters the pick-up and the return date, time again or ABORT

8a. If the pick-up, and the return branch is invalid
1. System notifies clerk
2. Clerk checks and enters the pick-up, and the return branch again or ABORT

**Use Case 8: Print out requested cars.**

Triggers/ Goals: Supervisor want to look the requested cars list at his/her branch to move to the requested branch.

Actors:Supervisor, Operator

Main flow:

1. System gets all "RESERVED" car in current branch which has the different pick-up branch
2. System adds list of cars into a datasheet
3. System prints out the list of cars for the supervisor

**Use Case 9: Check the license is match with license in rental.**

Triggers/ Goals: Clerk want to check if the license is match with the license used for the rental.

Actors: Clerk, Operator

Main flow:

1. Clerk enters the license number
2. System validates the license number is valid
3. System checks that the rental with that license number is existed
4. System prints out the rental for the clerk

Extension:

2a. If the license number is invalid
1. System notifies clerk
2. Clerk checks and enters the license number again or ABORT

3a. If the license is not match with any rental
1. System notifes clerk
2. Clerk checks and enters the license again or ABORT

**Use Case 10: Change car rental information.**
Triggers/ Goals: : Clerk want to edit car rental information.
Actors: Clerk, Operator
Main flow:
1. Systems prints out all car rental for clerk
2. Clerk clicks on the rental that needed to edit
3. Clerk chooses the line that need to edit
4. Clerk enters new information
5. System check that the new information is valid

—REPEAT steps 3-4-5 for each line's information that clerk wants to change

6. System validates all the information of the rental after edited is valid
7. System stored the changed rental into database

Extension:
5a. If the new information is not valid
1. System notifies clerk
2. Clerk checks and enters the information again or ABORT

6a. If any information is invalid
1. Sys notifies clerk
2. Clerk enters that information again or ABORT

**Use Case 11: Update payment status.**
Triggers/ Goals: : Clerk want to update payment status after customer paid the rest of the payment.
Invocation Constraints: Customer has been created a rental before.
Actors: Clerk, Operator
Main flow:
1. Customer pays the rest of the payment
2. Clerk enters the new payment status
3. System validates the new status is valid
4. System stores the change into database

Extension:
3a. If the new status is invalid
1. System notifies clerk
2. Clerk checks and enters the status again or ABORT

**Use Case 12: Make feedback.**
Triggers/ Goals: : Customer comes to pick-up the car but want to cancel .
Invocation Constraints:Customer has been created a rental before the retal for some reason and want to give feedback to the manager.
Actors: Customer, Operator
Main flow:

1. Customer enters his/ her information
2. System checks the information is valid
3. Customer enters the feedback
4. System validates the feedback is not empty
5. System sends feedback of the customer order to the manager
6. System notifies customer the feedback was recorded and send a sorry message to customer
7. System stored the feedback into database.

Extension:

2a. If the information is invalid
   1. System notifies customer
   2. Customer enters the information again or ABORT

4a. If the feedback is empty
   1. System notifies customer
   2. Customer enters feedback again or ABORT

**Use Case 13: Check whether the return branch is the pick-up branch or not.**

Triggers/ Goals: : Clerk wants to check whether the return branch is the pick-up branch or not.

Actors: Clerk, Operator

Main flow:

1. Clerk enters the car's license
2. System validates the car's license is valid
3. System checks that the license is match with a car in the database
4. System prints out the car information for the clerk
5. Clerk checks the pick-up branch of this car is the current branch or not

Extension:

2a. If the car's license is invalid
   1. System notifies clerk
   2. Clerk checks and enters the car's license again or ABORT

3a. If the license is not match with any car
   1. System notifies clerk
   2. Clerk checks and enters the license again or ABORT

**Use Case 14: Update the car's residing branch.**

Triggers/ Goals: :Clerk wants to update the car's residing branch .

Invocation Constraints: The car has been checked that the pick-up branch is not the current branch.

Actors: Clerk, Operator

Main flow:

1. Clerk enters the car's license
2. System validates the car's license is valid
3. System searchs for the car with that license in the database
4. System prints out the car information for the clerk
5. Clerk clicks the button to update the car's residing branch
6. Clerk enters the residing branch
7. System validates the residing brach is valid
8. System stores the modified car information into database

Extension:

2a. If the car's license is invalid
   1. System notifies clerk
   2. Clerk checks and enters the car's license again or ABORT

3a. If the license is not match with any car
   1. System notifies clerk
   2. Clerk checks and enters the license again or ABORT

7a. If the residing branch is invalid
   1. System notifies clerk
   2. Clerk checks and enters the residing branch again or ABORT

**Use Case 15: Prints out the list of cars to be inspected.**

Triggers/ Goals: :Supervisor wants to print the list of cars to be inspected if necessary.

Actors: Supervisor, Operator

Main flow:

1. System searchs all cars that have status is "RETURNED"
2. System adds all of them to a datasheet
3. System prints out the list for the supervisor

**Use Case 16: Create a new model.**

Triggers/ Goals: Clerk wants to create a new model for a new car.

Actors: Supervisor, Operator

Main flow:

1. Clerk enters the model number
2. System checks the model number is valid
3. Clerk enters a short description of the model
4. System checks the description is not empty
5. Clerk enters the model is automatic or manual
6. System checks that information is valid
7. Clerk enters the petrol consumption
8. System checks the petrol consumption is valid
9. Clerk enters the number of doors

10. System checks the number of doors is valid
11. Clerk enters the group of this model
12. System checks the group is valid
13. System saves the new model to database

Extension:

2a. If the model number is invalid
   1. System notifies clerk
   2. Clerk checks and enters the model number again or ABORT

4a. If the description is empty
   1. System notifies clerk
   2. Clerk checks and enters the description again or ABORT

6a. If the information is invalid
   1. System notifies clerk
   2. Clerk checks and enters the information again or ABORT

8a. If the petrol consumption is invalid
   1. System notifies clerk
   2. Clerk checks and enters the petrol consumption again or ABORT

10a. If the the number of doors is invalid
   1. System notifies clerk
   2. Clerk checks and enters the number of doors again or ABORT

12a. If the the group is invalid
   1. System notifies clerk
   2. Clerk checks and enters the group again or ABORT

**Use Case 17: Create a new car.**

Triggers/ Goals: Supervisor wants to create a new car to be added to the active pool.

Actors: Supervisor, Operator

Main flow:

1.Supervisor enters the registration number
2.System checks the registration number is valid
3.Supervisor enters the color
4.Supervisor enters the year of production
5.Supervisor enters the car's initial residing branch
6.System checks the car's initial residing branch is valid
7.Supervisor enters the car group
8.Supervisor validates the car group is valid
9.System sets the new car's status is "RENT-READY"
10.System stores the new car to the database

Extension:

2a. If the registration number is invalid
   1.System notifies clerk

2.Supervisor checks and enters the registration number again or ABORT

6a. If the car group is invalid

1.System notifies clerk

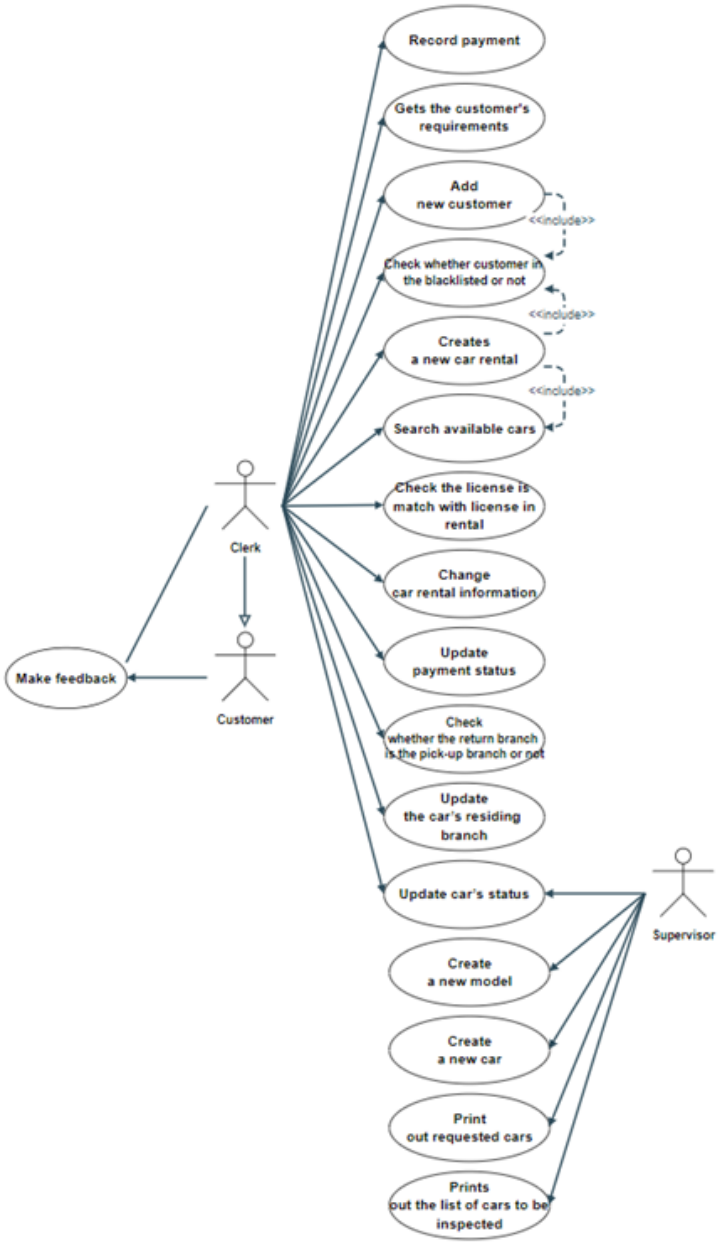
2.Clerk checks and enters the car group again or ABORT

8a. If the car's initial residing branch is invalid

1.System notifies clerk

2.Clerk checks and enters the car's initial residing branch again or ABORT

## 2.3 Use case diagram


Record payment
Gets the customer's requirements
Add new customer
<<include>>
Check whether customer in the blacklisted or not
<<include>>
Creates a new car rental
<<include>>
Search available cars
Check the license is match with license in rental
Change car rental information
Update payment status
Check whether the return branch is the pick-up branch or not
Update the car's residing branch
Update car's status
Create a new model
Create a new car
Print out requested cars
Prints out the list of cars to be inspected
Clerk
Customer
Supervisor
Make feedback

# 3 Task 2: Structural Domain Modeling

**Construct a structural domain model in terms of one or more (analysis) class diagrams.**

We use text analysis approach to identify classes and relationships for the diagram.

**Business Process: Renting a Car (BP1)**

Description:

1. A customer reserves a car with a branch. This is dealt with in the subprocess Reserving a Car (BP1.1a)

*We can see that the customer can rent a car at the branch. So, we obtain 4 classes: Customer, car, rental and branch.*

*Each car has a status, similarly each rental has a status. So, we have 2 class is CarStatus and RentalStatus. Alternatively, a customer can request and pick up a car without reservation. See subprocess Picking-up a Car without Reservation (BP1.1b)*

2. The branches must make sure that the car is available for pick-up. See subprocess Moving Cars on Request (BP1.2)

3. The customer picks up a car that has been reserved. See subprocess Picking-up a Reserved Cars (BP1.3)

4. The customer returns the car. See subprocess Recording the Returning of a Car (BP1.4)

=> *The 3 sentences above are the introduction for the BPs below.*

**Business Process: Reserving a Car (BP1.1a)**

Description:

1. A customer can rent a car by email, by phone or by walking into an office, making an enquiry or a request.

2. The booking clerk gets the customer's requirements. Typical requirements are as shown below:

*The clerk will receive and process the customer's car rental request. So, we have a class is clerk.*

*We also have a class is TypeOfCar for customer to choose the type of car.*

3. The clerk finds out from the system if an appropriate car is available at the requested branch or a neighboring branch and the cost (which is often of interest to the customer).

| Location of Travel | |
|---|---|
| Pick-up Branch: Tan Son Nhat Airport<br>Return Branch: Tan Son Nhat Airport | |
| When | |
| Pick-up Date: 20-AUG-2010<br>Return Date: 22-AUG-2010 | Time: AM 10.00<br>Time: AM 10.00 |
| Type of Car (Make one or more selections) | |
| Economy: ✓<br><br>Compact Car:<br>Medium Size: ✓<br>Full Size: | Station Wagon:<br>Van:<br>4 Wheel:<br>Sports/Luxury Car: |

Branches A and B are neighboring branches if any car at A can be requested to be moved to B overnight and vice versa (see subprocess BP1.2). Each branch has a list of neighboring branches.

To find out if a car is available is not a trivial task. We can apply simple search rules as well as quite sophisticated search rules. Currently, VinaRent uses the following simple search rule.

4. If no car is available, the clerk fills out a paper form which contains the requested pick-up branch, pick-up date and time, types of car wanted, and the customer contact phone number and/or email address. The supervisor may negotiate to have a car delivered to the requested branch and inform the customer about it. (It is not part of the responsibility of the system to support this sub-process.)

5. Otherwise (i.e. one or more cars are available), the clerk selects a car and put it on hold, that is sets it status to "HELD" (so that other operators will not reserve it).

NOTE ON ALTERNATIVE: If the deal does not go ahead as planned the clerk must release the hold on a car.

6. The clerk then asks for the customer's driver license to check if the customer is blacklisted or not.

*We have a class is blacklist to save the bad customer.*

7. If the customer is new, the clerk will enter the new customer to the customer base. The following information is entered: **the customer's first name, last name, driver license, email address** (if available), and **contact phone number** (if available).

*In here, we have attribute of Customer is FirstName, LastName, LicenseNumber, Email, PhoneNr*

8. If the customer is not black listed, a new transaction, which is called a rental, is

created. The new rental has a unique **rental number** (generated by the system) and has its status set to "RESERVED".

*In here, we have attribute of Rental is RentalNr.*

In addition to information about the reserved car, the pick-up and the return (date, time, branch), the clerk also enters information about payment.

*For the meaning of this sentence, we have a class is Payment to pay.*

A deposit of 10 % of cost (which may involves discount) has to be made by the customer. The payment can be made against the payment item "Deposit Payment" (the other payment item is "Cost Less Deposit Payment"). Payment amount and payment method (cash or credit card) are recorded. If the payment is made by credit card, the credit card's details are obtained and validated. The validation is performed by a call to another system.

*We have a class is PaymentMethod for the customer to choose. And a class is PaymentItem.*

The car assigned to this rental will have its status set to "RESERVED".

**Business Process: Picking-up a Car without Reservation (BP 1.1b)**

Description:

1. A customer walks into the office of a branch and makes a request.

2. The booking clerk gets the customer's requirements (as in subprocess BP1.1a) and see if a car is available at the branch.

3. If it is the clerk asks for the customer's driver license to check if the customer is blacklisted or not.

4. If the customer is not black listed, the clerk asks for payment and creates a new car rental. The same information recorded is as those described in the subprocess BP1.1a, except that the status of the new rental is marked as "PICKED-UP".

In addition, the car status is set to "PICKED-UP" and the mileage of the car is recorded for the rental (which is known as **start-mileage**).

*We have a attribute of Rental is startMileage.*

NOTE ON ALTERNATIVE: The customer may be a new one. In this case, his or her details are entered to the system as described in the previous process.

**Business Process: Moving Cars on Request (BP1.2)**

Description:

1. The supervisor prints out the list of cars at his/her branch which have been requested by other neighboring branches.

*We can see that the supervisor can check the information of the car in branch. So, we have a class is Supervisor.*

2. The supervisor arranges for the cars to be transferred. (It is not part of the responsibility of the system to support this activity.

**Business Process: Picking-up a Reserved Car (BP1.3)**

Description:

1. If the reserved car is not available for some reasons (e.g. overnight trip failed, which is rarely the case), the clerk will replace it by another car (if possible).

In this case, the previously reserved car will have its status set to "EXCEPTIONAL", and the substitute car will have its status set to "HELD" (so that no one else can put it on hold).

2. If a substitute cannot be made, the clerk sets the status of the reserved rental to "EXCEPTIONAL" and also ensures that the status of the reserved car is set to "EXCEPTIONAL"

The case is then referred to the supervisor to arrange for a refund. (It is not part of the responsibility of the system under study to support this process.)

3. Otherwise, the clerk checks the driver license to see if it matches the license used for the reservation. If a different driver license is used and the new driver is not blacklisted, a change of driver can be made.

4. The clerk asks for the rest of the payment and enters the payment details is entered. The payment item is "Cost Less Deposit Payment".

5. The clerk also enters the actual **pickup date** and **time**. The status of the rental is changed to PICKED-UP. The status of the car is also changed to PICKED-UP. The car's mileage is recorded again the rental record.

6. The customer picks up the car.

NOTE ON ALTERNATIVE: A customer may cancel a reservation or may simply not turn up. In either case, the rental will be cancelled and the customer losses their deposit. The rental's status is set to "EXCETIONAL", and the car's status set to "RENTREADY".

**Business Process: Recording the Returning of a Car (BP1.4)**

Description:

1. The clerk records **end mileage**, and the actual return date and time, and changes the status of the rental to "RETURNED".

*We have a attribute of Rental is endMileage.*

2. If the car is returned to different branch (i.e. not the "return" branch recorded at the time of reservation or pick-up), this has to be recorded against the rental record.

3. The clerk changes the status of the car to RETURNED, and updates the car's residing branch (i.e. where the car is).

NOTE ON SUBSEQUENT ACTIVITIES: The car will need to be inspected before returning to the "active" pool ready to be rented. The car can also be set aside for services or to be removed from the fleet..

**Business Process: Arranging for Car Maintenance (BP2)**

Description:

1. The supervisor prints the list of cars to be inspected if necessary (for example, the list of cars that were returned to the current branch).

2. The supervisor inspects the cars. The outcome for a car can be that it is OK, or that it needs servicing, or that it is to be removed, and the supervisor will have its status updated accordingly, i.e. RENT-READY, SERVICE NEEDED, or REMOVED respectively.

3. Supervisor arranges for necessary actions. (It is not part of the responsibility of the system under study to support this activity).

NOTE ON SUBSEQUENT ACTIVITIES: Later when a car has been serviced, and becomes available at a branch, the information about the car is updated.

**Business Process: Adding a New Car to the Active Pool (BP3)**

Description:

1. If the model is new, the clerk enter the model's details, which include (a) **the model number** and a **short description** of the model (which usually includes the model's marketing name), (b) **automatic or manual**, (c) **petrol consumption** (such as 1.5 liters or 6 cylinders) , and (d) **number of doors**. The clerks also specifies which group ("A" to "E") the model is classified into.

*We have a class Model to represent the model of the car. And the attribute of class Model is modelNr,shortDescription, numberOfDoor, control.*

*We also have a class is Group, which group the model is classified into. And ControlModel for the type of control(automatic or manual).*

2. The clerk enters details of the car, including registration number, color, year of production, and the car's initial residing branch. The car's status will be RENTREADY.

**Step 1: Identify candidates class and candidates relationships:**

16 candidate class: Customer, Car, Rental, Branch, CarStatus, RentalStatus, Clerk, TypeOfCar, Blacklist, Payment, PaymentMethod, PaymentItem, Supervisor, Model,

Group, ControlModel.

4 candidate relationship: Reserves, get, check, enter.

**Step 2: Identify class and relationships:**

13 class: Customer, Car, Rental, Branch, CarStatus, RentalStatus, TypeOfCar, Payment, PaymentMethod, PaymentItem, Model, Group, ControlModel.

4 relationship: Reserves, get, check, enter.

**Step 3: Identify multiplicity:**

Branch and Car: 1 – N. (It(branch) has about 5,000 cars and makes about 500,000 rentals per year. The branch includes many cars for customers to choose).

Car and CarStatus: 1 – 1. (Each Car has one RentalStatus to represent the status at the current moment).

Car and Model: 1 – 1. (Each car has a model).

Car and TypeCar: 1 – N. (a car can be of many different types).

Model and Group: N – 1. (The clerks also specifies which group ("A" to "E") the model is classified into).

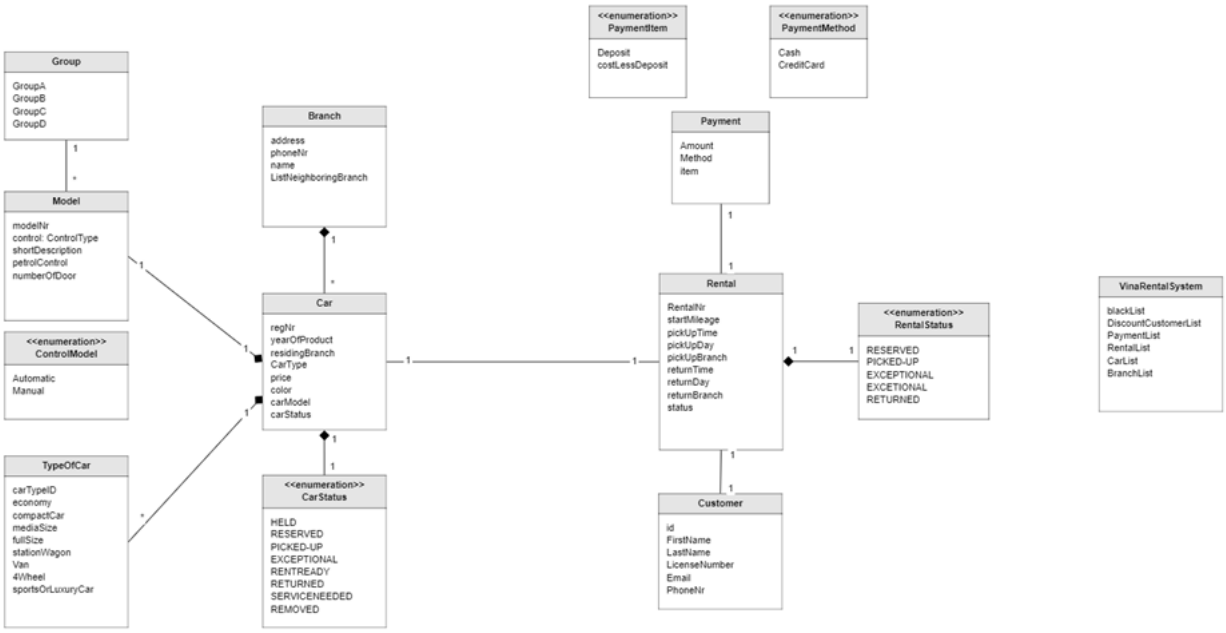
Car and Retal: 1 – 1. (For each car rented by a customer, a rental is generated)

Rental and Payment: 1 – 1. (Customers can pay the rental by a payment).

Rental and RentalStatus: 1 – 1. (The new rental has a unique rental number (generated by the system) and has its status set to "RESERVED". Each Rental has one RentalStatus to represent the status of this).

Rental and Customer: 1 – 1. (If the customer is not black listed, a new transaction, which is called a rental, is created).

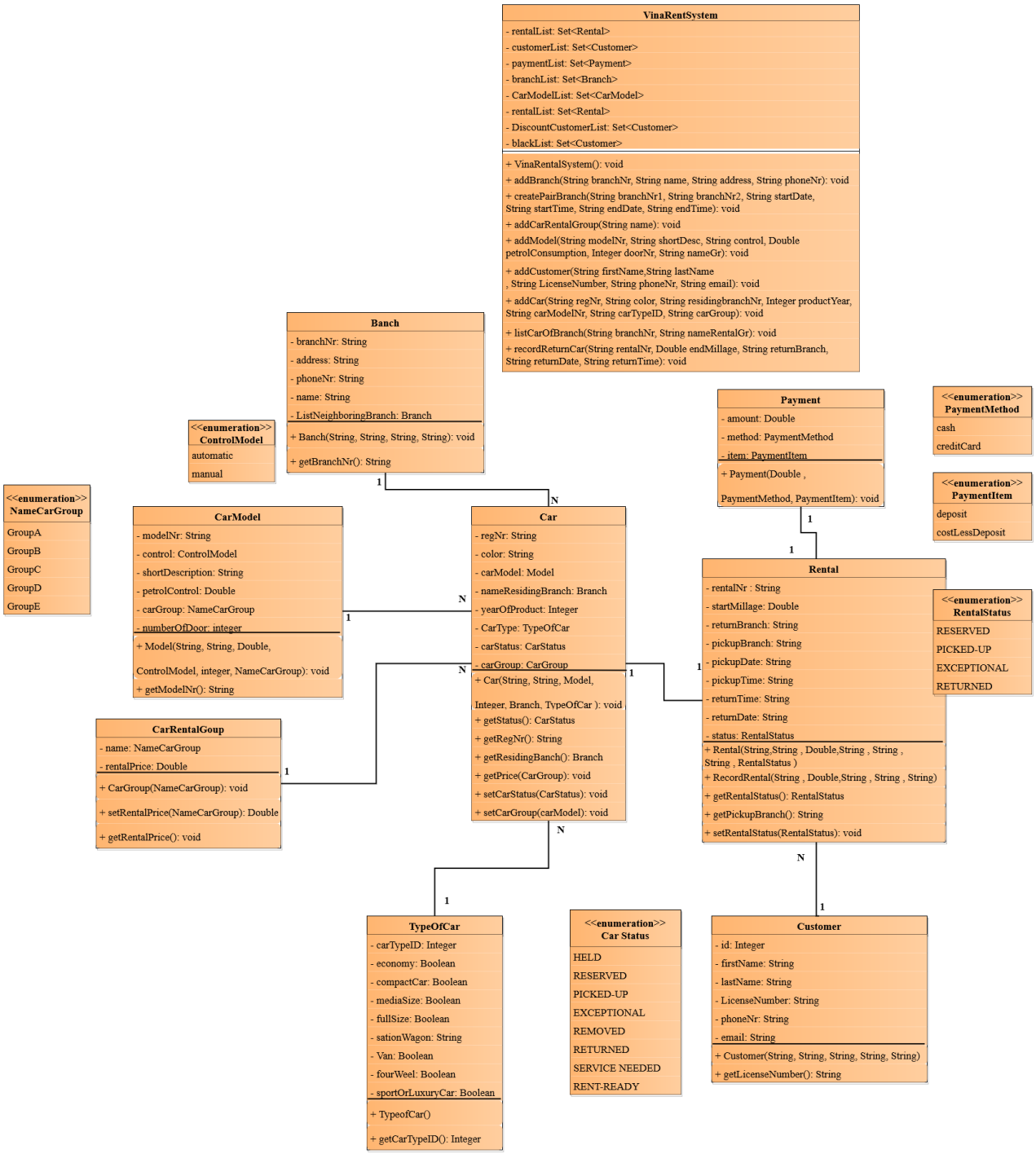
Base on these above analysis, we get the analysis class diagram:




# 4 Task 3: Design Class Diagram

**Base on the analysis class diagram in Task 2, the main task of this part is to construct the design class diagrams to show the complete design of the system.**

## 4.1 Design class Diagram.


VinaRentSystem
- rentalList: Set<Rental>
- customerList: Set<Customer>
- paymentList: Set<Payment>
- branchList: Set<Branch>
- CarModelList: Set<CarModel>
- rentalList: Set<Rental>
- DiscountCustomerList: Set<Customer>
- blackList: Set<Customer>
+ VinaRentalSystem(): void
+ addBranch(String branchNr, String name, String address, String phoneNr): void
+ createPairBranch(String branchNr1, String branchNr2, String startDate, String startTime, String endDate, String endTime): void
+ addCarRentalGroup(String name): void
+ addModel(String modelNr, String shortDesc, String control, Double petrolConsumption, Integer doorNr, String nameGr): void
+ addCustomer(String firstName,String lastName , String LicenseNumber, String phoneNr, String email): void
+ addCar(String regNr, String color, String residingbranchNr, Integer productYear, String carModelNr, String carTypeID, String carGroup): void
+ listCarOfBranch(String branchNr, String nameRentalGr): void
+ recordReturnCar(String rentalNr, Double endMillage, String returnBranch, String returnDate, String returnTime): void
Banch
- branchNr: String
- address: String
- phoneNr: String
- name: String
- ListNeighboringBranch: Branch
+ Banch(String, String, String, String): void
+ getBranchNr(): String
<<enumeration>> ControlModel
automatic
manual
Payment
- amount: Double
- method: PaymentMethod
- item: PaymentItem
+ Payment(Double , PaymentMethod, PaymentItem): void
<<enumeration>> PaymentMethod
cash
creditCard
<<enumeration>> PaymentItem
deposit
costLessDeposit
<<enumeration>> NameCarGroup
GroupA
GroupB
GroupC
GroupD
GroupE
CarModel
- modelNr: String
- control: ControlModel
- shortDescription: String
- petrolControl: Double
- carGroup: NameCarGroup
- numberOfDoor: integer
+ Model(String, String, Double, ControlModel, integer, NameCarGroup): void
+ getModelNr(): String
Car
- regNr: String
- color: String
- carModel: Model
- nameResidingBranch: Branch
- yearOfProduct: Integer
- CarType: TypeOfCar
- carStatus: CarStatus
- carGroup: CarGroup
+ Car(String, String, Model, Integer, Branch, TypeOfCar ): void
+ getStatus(): CarStatus
+ getRegNr(): String
+ getResidingBanch(): Branch
+ getPrice(CarGroup): void
+ setCarStatus(CarStatus): void
+ setCarGroup(carModel): void
Rental
- rentalNr : String
- startMillage: Double
- returnBranch: String
- pickupBranch: String
- pickupDate: String
- pickupTime: String
- returnTime: String
- returnDate: String
- status: RentalStatus
+ Rental(String,String , Double,String , String , String , RentalStatus )
+ RecordRental(String , Double,String , String , String)
+ getRentalStatus(): RentalStatus
+ getPickupBranch(): String
+ setRentalStatus(RentalStatus): void
<<enumeration>> RentalStatus
RESERVED
PICKED-UP
EXCEPTIONAL
RETURNED
CarRentalGoup
- name: NameCarGroup
- rentalPrice: Double
+ CarGroup(NameCarGroup): void
+ setRentalPrice(NameCarGroup): Double
+ getRentalPrice(): void
TypeOfCar
- carTypeID: Integer
- economy: Boolean
- compactCar: Boolean
- mediaSize: Boolean
- fullSize: Boolean
- sationWagon: String
- Van: Boolean
- fourWeel: Boolean
- sportOrLuxuryCar: Boolean
+ TypeofCar()
+ getCarTypeID(): Integer
<<enumeration>> Car Status
HELD
RESERVED
PICKED-UP
EXCEPTIONAL
REMOVED
RETURNED
SERVICE NEEDED
RENT-READY
Customer
- id: Integer
- firstName: String
- lastName: String
- LicenseNumber: String
- phoneNr: String
- email: String
+ Customer(String, String, String, String, String)
+ getLicenseNumber(): String
1
N

# 5 Task 4: Atomic Use Case Specifications

**Write atomic use cases for six main use cases.**

**1. Add a branch.**

```
        in:
            branchNr?: String
            name?:String
            address?: String
            phoneNr? : String
        out:
            NONE
        pre:
        // branchNr? is new
        not exists b in branchList| b.branchNr = branchNr?
      post:
          let b = new Branch(branchNr?, name?, address?, phoneNr?) |
              b.branchNr = branchNr?
              b.name = name?
              b.address = address?
              b.phoneNr = phoneNr?
              add b to branchList


```

**2. Make a pair of branches neighbors to each other.**

Branches A and B are neighboring branches if any car at A can be requested to be moved to B overnight and vice versa

As the definition, i assume that if time to move any car from branch A to branch B is less than 12 hours, A and B will be neighbor branch of each other. Assume The system already chose the longest distance to get the data to calculate the time

**Use Case 2: Make a pair of branches neighbors to each other.**

Triggers/ Goals: Supervisor wants to create a pair of branches neighbors to each other.

Actors: Operator

Main flow:

1.Operator enters the first branch number.

2.System validates the branch number is valid

3.Operator enters the remaining branch number.

4.System validates the branch number is valid and not similar with the branch number of the first branch

5.Supervisor enters the date and time when the cars start to move from one branch to other.

6.Supervisor enters the end date and end time when the cars already went to the

destination branch.

7.System validates that branch 2 not exist in the neighbor branch list of branch 1

8.System calculates whether time to move from one to another branch is less than 12 hours or not

9.Supervisor update the each branch into the neighbor list of each other.

10.System save the neighbor list for two branch into the database

Extension:

2a. If the first branch number is invalid

1.System notifies clerk

2.Supervisor checks and enters the branch number again or ABORT

4a. If the second branch number is invalid

1.System notifies clerk

2.Supervisor checks and enters the branch number again or ABORT

6a. If the time to move from one to another branch is greater or equals 12 hours

1.ABORT

6a. If the branch 2's number already exist in the list neighbor branch of branch 1.

1.ABORT

```
        in:
          branchNr1?: String
          branchNr2?: String
          startTime?: Date
          startDate?: Date
          endTime?: Date
          endDate?: Date
        out:
            NONE
      pre:
        // branchNr1? exists
        exists b1 in branchList| b1.branchNr = branchNr1?
        // branchNr2? exists and different with branchNr1?
        exists b2 in branchList| b2.branchNr = branchNr2? AND
  branchNr2? != branchNr1?
        // branch2 dose not exist in the neighborlist of branch 1
        let b1 = element in branchList| b1.branchNr = branchNr2?
        not exists b in b1.neighborBranchList | b.branchNr = empNo?
        let time = startTime.getTime() - endTime.getTime() + startDate
  .getTime() - endDate.getTime()
        then
            time < 12 hours
      post:
        let branch1 = element in branchList| b1.branchNr = branchNr1?
        let branch2 = element in branchList| b2.branchNr = branchNr2?
        branch1.addBranchNeighbor(branchNr2?)
        branch2.addBranchNeighbor(branchNr1?)

```

### 3. Add a car rental group.

```
        in:
          name? : String
      out:
        NONE
      pre:
        // NameCarGroup? exists
        exists n in NameCarGroup| NameCarGroup.n = NameCarGroup?
      post:
        let cRentalGr = new  CarRentalGroup(name?)|
            n.name = name?
            n.retalPrice = c.setRetalPrice()
                add n from carRentalGroupList

```

### 4. Add a model.

#### 4.Use Case: Add a model.

Triggers/ Goals: Operator wants to add a new model into the system.

Actors:Operator

Main flow:

1. Operator enters the model number.
2. System validates the model number is new.
3. Operator enters the control version(automatic/ manual) of the model.
4. System validates whether the control version is valid or not.
5. Operator enters the car group's type of the model.
6. System validates whether the car group's type is valid or not.
7. Operator enter the short description, the number of door, the petrol consumption level.
8. System save the information of the new car into the database.

Extension:

2a. If the model number already existed.
   1. System notifies operator.
   2. Operator checks and enters the model number again or ABORT .

4a. If the control version is invalid(not exist).
   1. System notifies operator.
   2. Operator checks and enters the control version again or ABORT .

6a. If car group's type of the model is invalid(not exist).
   1. System notifies operator.
   2. Operator checks and enters the car group again or ABORT .

#### **. The atomic use case specification:

```
        in:
          modelNr? : String
          control? : ControlModel
          shortDesc? : String
```

```
          petrolConsumption?: Double
          doorNr?: Integer
          nameGr?: NameCarGroup
      out:
        NONE
      pre:
        // modelNr? is new
        not exists m in modelList| m.modelNr = modelNr?
        // control? is valid
        exists c in ControlModel| ControlModel.c = modelNr?
        // nameGr? is valid
        exists n in NameCarGroup| NameCarGroup.n = nameGr?
      post:
        let m = new Model(modelNr?, control?, shortDesc?,
petrolConsumption?, doorNr?, nameGr?)|
          m.modelNr = modelNr?
          m.control = control?
          m.shortDesc = shortDesc?
          m.petrolConsumption = petrolConsumption?
          m.doorNr = doorNr?
          m.nameGr = nameGr?

          add m to modelList

```

**5. Add a car.**

this atomic use case specification is based on use case 17: Create a new car in task 1. Because i assume that adding a new car means the car already inspected to get ready for renting

```
      in:
          regNr?: Integer
          color?: String
          residingbranchNr?: String
        yearOfProduct?: Integer
        carModelNr?: String
        carTypeID?: String
        carGroup?: String
      out:
        NONE
      pre:
        // regNr? is new
        not exists c in CarList| c.regNr = regNr?
        // carGroup? exist
        exists n in NameCarGroup| NameCarGroup.n = carGroup?
        // residingbranch? exist
        exists b in BranchList| b.branchNr = residingbranchNr?
      post:
        let c = new Car(regNr?, color?, residingbranchNr?,
yearOfProduct?, carModelNr?, carTypeID?, carGroup? ) |
```

```
                c.regNr = regNr?
                c.color = color?
                c.residingbranchNr = residingbranchNr?
                c.yearOfProduct = yearOfProduct?
                c.carModelNr = carModelNr?
                c.carTypeID = carTypeID?
                c.carGroup = carGroup?

                add c to CarList

```

**6. Add a customer.**

```
        in:
            customerID? : String
            firstName?: String
            lastName? : String
            licenseCar? : String
            address? : String
            phoneNr? : String
            email? : String
        out:
            NONE
        pre:
            // licenseCar? not exist in the customer list
        not exists c in customerList| c.customerList = customerList?
         post:
            let c = new Customer(customerID?, firstName?, lastName?,
  licenseCar?, address?,phoneNr?,email?, discountCustomerList?) |
                c.customerID = customerID?
                c.firstName = firstName?
                c.lastName = lastName?
                c.licenseCar = licenseCar?
                c.address = address?
                c.phoneNr = phoneNr?
                c.email = email?
                c.discountCustomerList = discountCustomerList?
                add c to customerList


```

**7. List cars that are available at a specified branch and belong to a specified rental group.** **4.Use Case: Add a model.**

Triggers/ Goals: Operator wants to print out cars that are available at a specified branch and belong to a specified rental group.

Actors:Operator

Main flow:

1. Operator enters the branch number.
2. System validates the branch number exist in the branchList.

3. Operator enters the name of car rental group.
4. System validates whether the name of car rental group is valid or not.
5. Operator print out cars belong to the branch has input branch number and car rental group.

Extension:

2a. If the branch number does not exist.
1. System notifies operator.
2. Operator checks and enters the model number again or ABORT .

4a. If the name of car rental group is invalid.
1. System notifies operator.
2. Operator checks and enters the name of car rental or ABORT .

****. The atomic use case specification:**

```
        in:
          branchNr? : String
          nameRentalGr? : String
      out:
        result!
      pre:
        // branchNr? exists
        exists b in branchList| b.branchNr = branchNr?
        // branchNr? valid
        exists n in NameCarGroup|NameCarGroup.n  = nameRentalGr?
      post:
          // for each car return regNr, color, residingbranchNr,
 yearOfProduct, carModelNr, carTypeID and carGroupNr
          let result! == new List()
          for c in carList do
          {
              add(c.regNr, c.color, c.residingbranchNr, c.
 yearOfProduct, c.carModelNr, c.carTypeID,c.carGroupNr) to result!
          }


```

## 8. Record the return of a car.

***.Use Case: Record returning a car.**

Triggers/ Goals: Clerk saves the return car information into the system when a customer want to returns a car.

Innovation Constrain:The car must belong to a rental in the system.

Actors: Clerk, Operator

Main flow:

1. Clerk enters the rental number.
2. System validates the rental number is valid.
3. Clerk enters the return branch's address.

4. System check whether the return branch's address is similar with the pick-up branch recorded in the rental's information or not.

— If the return branch address of the car is different with the pick-up address in rental, do step 5.

5. Clerk update the residing branch of the car to the return branch address.

6. Clerk updates the status of car possess this rental into RETURNED.

7. Clerk enters the end millage, the actual date and time.

8. Clerk updates the status of rental into RETURNED.

9. System save the information of the new car into the database.

Extension:

2a. If the rental number is invalid.

1. System notifies clerk.

2. Clerk checks and enters the rental number again or ABORT .

****. The atomic use case specification:**

```
      in:
        rentalNr? : String
        returnBranch?: String
        endMillage? : Double
        returnDate?: Date
        returnTime?: String
      out:
        NONE
      pre:
      // rentalNr? exist
      exists r in rentalList| r.rentalNr = rentalNr?
    post:
        // if returnBranch? is different with the pickup address
        let r = element in rentalList | r.rentalNr = rentalNr? AND r
.pickUpBranch != returnBranch?
          // retrieve the car
              let c = element in CarList | c.regNr = r.regNr
              // update the cars' branch into the residing branch
              set c.branch = returnBranch?
              // update the car status into RETURNED
              set c.status = CarStatus.RETURNED

          // retrieve the rental
          let r = element in rentalList | r.rentalNr = rentalNr?
          // update information of returning car into the rental
information
              set r.returnBranch = returnBranch?
              set r.endMillage = endMillage?
              set r.returnDate = returnDate?
              set r.returnTime = returnTime?
              set r.status = RentalStaTus.RETURNED // update the
rental status into RETURNED
```

30
31

# 6 Task 5: Protype and Testing the Prototype

## 6.1 Protype and Testing for use case 1: Add a branch.

```
// build the prototype for use case 1: Add a branch to the branch list
public void addBranch(String branchNr, String name, String address, String phoneNr)  throws Exception {
  Branch branch = (Branch) Helper.search(branchList, branchNr);
  boolean pre = branch == null;
  if (!pre) {
    throw new Exception(" Cannot add the branch into the system since the  branch ID : " + branchNr + " already existed in the system!");
  }
  branch = new Branch(branchNr, name, address, phoneNr);
  branchList.add(branch);
  System.out.println( branch.toString() + " has been add successfully!!!");

}
```

Prototype of function add a branch

```
//  ================ TEST FOR USE CASE 1: ADD A BRANCH ==================
//  In this case, we must check the branch number is new or not
// Add some branchs to the branchList:
vnrs.addBranch("b1", "VinFast", "Thu Duc, HCM", "0369872830");
vnrs.addBranch("b2", "Hyundai Truong Chinh", "Thu Duc, HCM", "0369872834");
// try some invalid requests for adding new customer
// add customer with the existed branch number in the System
vnrs.addBranch("b1", "MazdaPMH", "Distric 7, HCM", "0369872830");
```

```
java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
   Branch[branchNr: b1, name: VinFast, address : Thu Duc, HCM, phoneNr: 0369872830] has been
 add successfully!!!

   Branch[branchNr: b2, name: Hyundai Truong Chinh, address : Thu Duc, HCM, phoneNr: 0369872
834] has been add successfully!!!
Exception in thread "main" java.lang.Exception:  Cannot add the branch into the system since
the  branch ID : b1 already existed in the system!
    at VinaRentalSystem.addBranch(VinaRentalSystem.java:39)
    at Main.main(Main.java:14)
```

## 6.2 Protype and Testing for use case 2: Make a pair of branches neighbors to each other.

```
// build the prototype for use case 2:  Make a pair of branches neighbors to each other.
public void createPairBranch(String branchNr1, String branchNr2, String startDate, String startTime, String endDate, String endTime)  throws Exception {
  Branch branch1 = (Branch) Helper.search(branchList, branchNr1);
  boolean pre = branch1 == null;
  //pre1: check whether the first branch number is new or not
  if (pre) {
    throw new Exception(" Cannot create a pair branch into the system since the  branch number : " + branchNr1 + " does not exist in the system!");
  }
  //pre2: check whether the second branch number is new and different with the first one
  Branch branch2 = (Branch) Helper.search(branchList, branchNr2);
  boolean pre2 = branch2 == null;
  if (pre2) {
    throw new Exception(" Cannot create a pair branch into the system since the second branch number : " + branchNr2 + " does not exist in the system!");
  }
  if(branchNr2.equals(branchNr1)){
    throw new Exception(" Cannot create a pair branch into the system since 2 branch number are similar!!!");
  }
  // pre3: Check that  branch2 dose not exist in the neighborlist of branch 1
  if(branch1.getlistNeibouringBranch().contains(branch2))
  {
    throw new Exception("Cannot  create a pair branch into the system since the second branch number " + branchNr2 + "already exist in the neighborlist of branch 1");
  }
  SimpleDateFormat sdf = new SimpleDateFormat("dd/M/yyyy HH:mm");
  long difference_In_Time = sdf.parse(endDate +" " + endTime).getTime() - sdf.parse(startDate+" " + startTime).getTime();
  int hours   = (int) ((difference_In_Time / (1000*60*60)) % 24);
  if(hours >= 12){
    throw new Exception("The time to move cars from branch " +branchNr1+" to branch " + branchNr2 + " is "+ hours + "=> Cannot create a pair branch into the system since the time to move cars from one to
    another is greater or equals 12 hours!!! ");
  }
  branch1.addNeighborBranch(branch2);
  branch2.addNeighborBranch(branch1);
  System.out.println("Makimg a pair of branches neighbors for branch " +branchNr1+" and branch " + branchNr2 +  " has been add successfully!!!");
}
```

check 4 pre conditions

update neighbor branch in to the list of each other branch

```
//  ======== TEST FOR USE CASE 2: MAKE A PAIR OF BRANCHES NEIGHBORS TO EACH OTHER ========
// Add some pair of branchs to the branchNeigborList of each branch:
vnrs.createPairBranch("b1", "b3", "15/10/2021", "16:30", "16/10/2021", "2:10");
vnrs.createPairBranch("b1", "b2", "22/10/2021", "7:50", "22/10/2021", "10:40");
vnrs.createPairBranch("b3", "b2", "29/12/2021", "7:50", "30/10/2021", "1:40");
// try some invalid requests for adding new branch
// Add some pair of branchs to the branchNeigborList of each branch with invalid branch number:
try
{
  vnrs.createPairBranch("b6", "b3", "15/10/2021", "16:30", "16/10/2021", "2:10");
}
catch(Exception e)
{
  System.out.println(e);
}
// Add a pair of branchs to the branchNeigborList of each branch with duplicate branch number:
try
{
  vnrs.createPairBranch("b3", "b3", "15/10/2021", "16:30", "16/10/2021", "2:10");
}
catch(Exception e)
{
  System.out.println(e);
}
// Add pair of branchs to the branchNeigborList of each branch while they already exist in the branchNeigborList of each other:
try
{
  vnrs.createPairBranch("b2", "b1", "15/10/2021", "16:30", "16/10/2021", "2:10");
}
catch(Exception e)
{
  System.out.println(e);
}
// Add pair of branchs to the branchNeigborList of each branch while the time to take car forwarding other branch is greater than 12 hours:
try
{
  vnrs.createPairBranch("b2", "b4", "15/10/2021", "8:30", "16/11/2021", "23:10");
}
catch(Exception e)
```

pre1

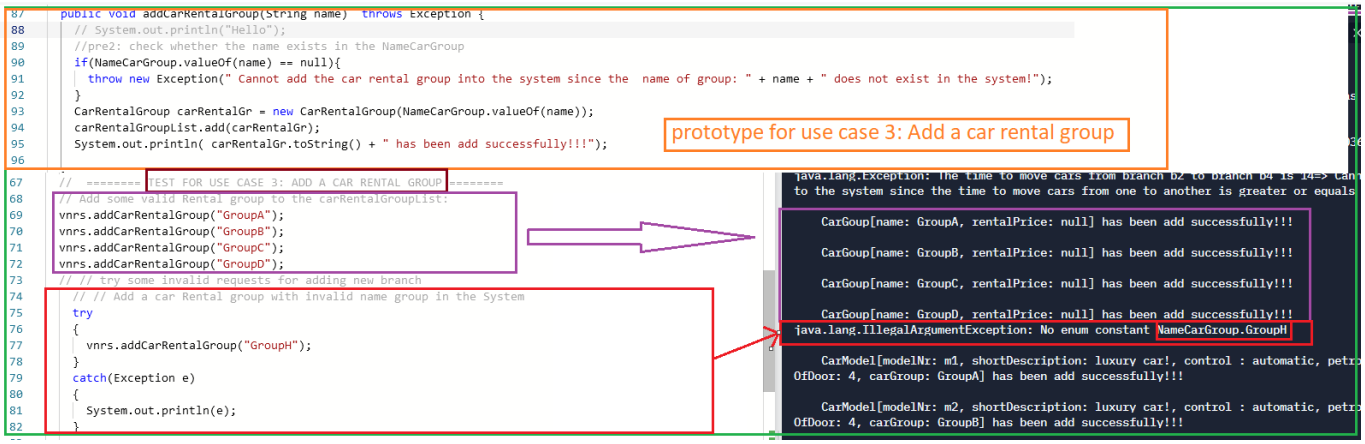
pre2

pre3

pre4

```
.java Helper.java Main.java NameCarGroup.java Rental.java RentalStatus.java SimpleKe
a TypeOfCar.java VinaRentalSystem.java
> java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main

    Branch[branchNr: b1, name: VinFast, address : Thu Duc, HCM, phoneNr: 0369872830] has
been add successfully!!!

    Branch[branchNr: b2, name: Hyundai Truong Chinh, address : Thu Duc, HCM, phoneNr: 036
9872834] has been add successfully!!!

    Branch[branchNr: b3, name: Hyundai Truong Chinh, address : Thu Duc, HCM, phoneNr: 036
9872874] has been add successfully!!!

    Branch[branchNr: b4, name: VinFast, address : Distric 7, HCM, phoneNr: 0369872874] ha
s been add successfully!!!
Makimg a pair of branches neighbors for branch b1 and branch b3 has been add successfully
!!!
Makimg a pair of branches neighbors for branch b1 and branch b2 has been add successfully
!!!
Makimg a pair of branches neighbors for branch b3 and branch b2 has been add successfully
!!!
java.lang.Exception:  Cannot create a pair branch into the system since the  branch numbe
r : b6 does not exist in the system!                                                    1
java.lang.Exception:  Cannot create a pair branch into the system since 2 branch number a
re similar!!!                                                                           2
java.lang.Exception: Cannot  create a pair branch into the system since the second branch
 number b1already exist in the neighborlist of branch 1                                 3
java.lang.Exception: The time to move cars from branch b2 to branch b4 is 14=> Cannot cre
ate a pair branch into the system since the time to move cars from one to another is grea
ter or equals 12 hours!!!                                                               4

    CarModel[modelNr: m1, shortDescription: luxury car!, control : automatic, petrolConsu
mption: 25.7, numberOfDoor: 4, carGroup: GroupA] has been add successfully!!!

    CarModel[modelNr: m2, shortDescription: luxury car!, control : automatic, petrolConsu
mption: 26.0, numberOfDoor: 4, carGroup: GroupB] has been add successfully!!!

    CarModel[modelNr: m3, shortDescription: sport car!, control : automatic, petrolConsum
ption: 27.9, numberOfDoor: 4, carGroup: GroupC] has been add successfully!!!

    CarModel[modelNr: m4, shortDescription: medium truck!, control : manual, petrolConsum
ption: 26.5, numberOfDoor: 4, carGroup: GroupE] has been add successfully!!!

    Car[regNr: x1, color: red, branch : b1, carStatus: null, carModel: m1, yearOfProd
: 2020, carTypeID: tc1, carGroup: null] has been add successfully!!!
```

## 6.3 Protype and Testing for use case 3: Add a car rental group.



## 6.4 Protype and Testing for use case 4: Add a model.

```java
// build the prototype for use case 4: Add a model to the model list
public void addModel(String modelNr, String shortDesc, String control, Double petrolConsumption, Integer doorNr, String nameGr)  throws Exception {
  CarModel model = (CarModel) Helper.search(carModelList, modelNr);
  boolean pre = model == null;
  //pre1: check whether the model number is new or not
  if (!pre) {
    throw new Exception(" Cannot add the model into the system since the  model number : " + modelNr + " already existed in the system!");
  }
  //pre2: check whether the control exists in the ControlModel
  if(ControlModel.valueOf(control) == null){
    throw new Exception(" Cannot add the model into the system since the  control type : " + control + " does not exist in the system!");
  }
  //pre3: check whether the the car group's type exists in the ControlModel
  if(NameCarGroup.valueOf(nameGr) == null)
  {
    throw new Exception(" Cannot add the model into the system since the car group : " + nameGr + " does not exist in the system!");
  }
  // Create an object model with inspected input parameters:
  model = new CarModel(modelNr,shortDesc, control, petrolConsumption, doorNr, nameGr);
  carModelList.add(model); // add object model into the carModelList
  System.out.println( model.toString() + " has been add successfully!!!"); // confirm success for user by the way print the model information in sreen

}
```

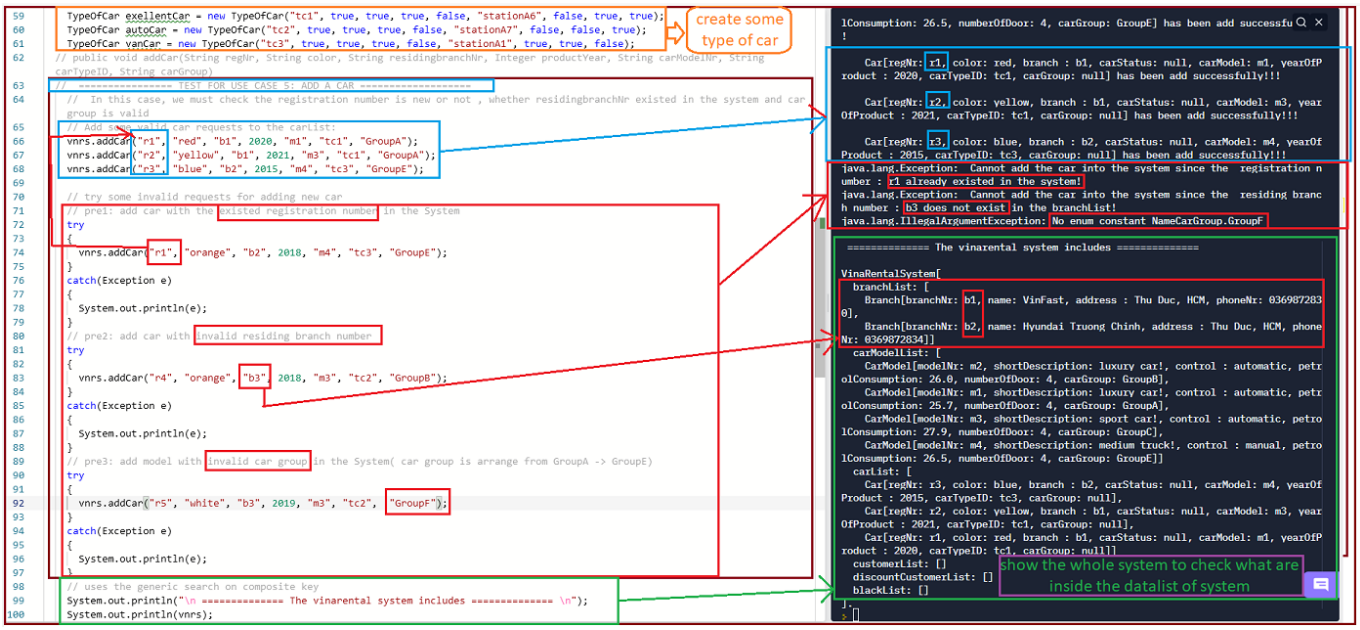
check pre-conditions

```java
// ================ TEST FOR USE CASE 4: ADD A MODEL ==================
// In this case, we must check the model number is new or not , whethername car group and type of control are valid
// Add some valid models to the branchList:
vnrs.addModel("m1", "luxury car!", "automatic", 25.7, 4, "GroupA" );
vnrs.addModel("m2", "luxury car!", "automatic", 26.0, 4, "GroupB" );
vnrs.addModel("m3", "sport car!", "automatic", 27.9, 4, "GroupC" );

// try some invalid requests for adding new model
// pre1: add model with the existed model number in the System
try
{
  vnrs.addModel("m1", "VinFast car", "manual",  27.9, 4, "GroupE");
}
catch(Exception e)
{
  System.out.println(e);
}
// pre2: add model with invalid control type in the System
try
{
  vnrs.addModel("m4", "VinFast car", "remotecontrol",  27.9, 4, "GroupC");
}
catch(Exception e)
{
  System.out.println(e);
}
// pre3: add model with invalid car group in the System( car group is arrange from GroupA -> GroupE)
try
{
  vnrs.addModel("m4", "VinFast car", "manual",  28.2, 4, "GroupF");
}
catch(Exception e)
{
  System.out.println(e);
}
// uses the generic search on composite key
System.out.println("\n ============== The vinarental system includes ============== \n");
System.out.println(vnrs);
```

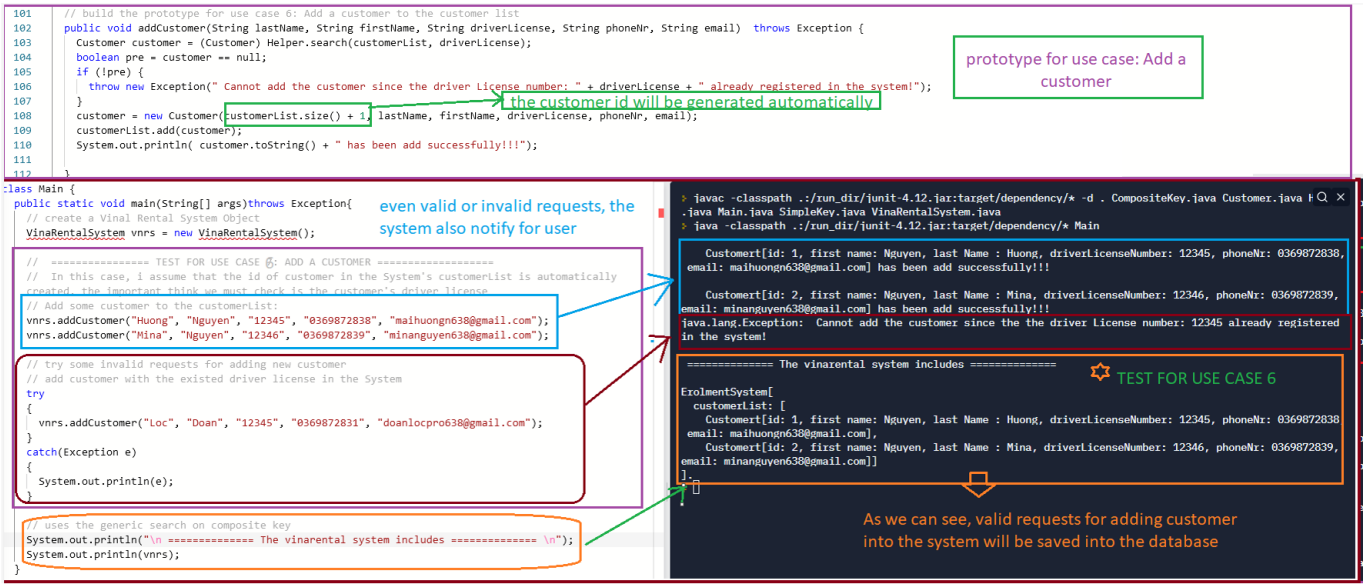
```
rModel.java CarStatus.java CompositeKey.java ControlModel.java Customer.java Helper.java Main.java NameCarGroup.java NeibouringBranch.java SimpleKey.java TypeOfCar.java VinaRentalSystem.java
> java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
    CarModel[modelNr: m1, shortDescription: luxury car!, control : automatic, petrolConsumption: 25.7, numberOfDoor: 4, carGroup: GroupA] has been add successfully!!!

    CarModel[modelNr: m2, shortDescription: luxury car!, control : automatic, petrolConsumption: 26.0, numberOfDoor: 4, carGroup: GroupB] has been add successfully!!!

    CarModel[modelNr: m3, shortDescription: sport car!, control : automatic, petrolConsumption: 27.9, numberOfDoor: 4, carGroup: GroupC] has been add successfully!!!
java.lang.Exception:  Cannot add the model into the system since the  model number : m1 already existed in the system!
java.lang.IllegalArgumentException: No enum constant ControlModel.remotecontrol
java.lang.IllegalArgumentException: No enum constant NameCarGroup.GroupF

 ============== The vinarental system includes ==============

VinaRentalSystem[
  branchList: []
  carModelList: [
    CarModel[modelNr: m2, shortDescription: luxury car!, control : automatic, petrolConsumption: 26.0, numberOfDoor: 4, carGroup: GroupB],
    CarModel[modelNr: m3, shortDescription: sport car!, control : automatic, petrolConsumption: 27.9, numberOfDoor: 4, carGroup: GroupC],
    CarModel[modelNr: m1, shortDescription: luxury car!, control : automatic, petrolConsumption: 25.7, numberOfDoor: 4, carGroup: GroupA]]
  customerList: []
  discountCustomerList: []
  blackList: []
].
> []
```

Valid requests will be accept and stored into the system

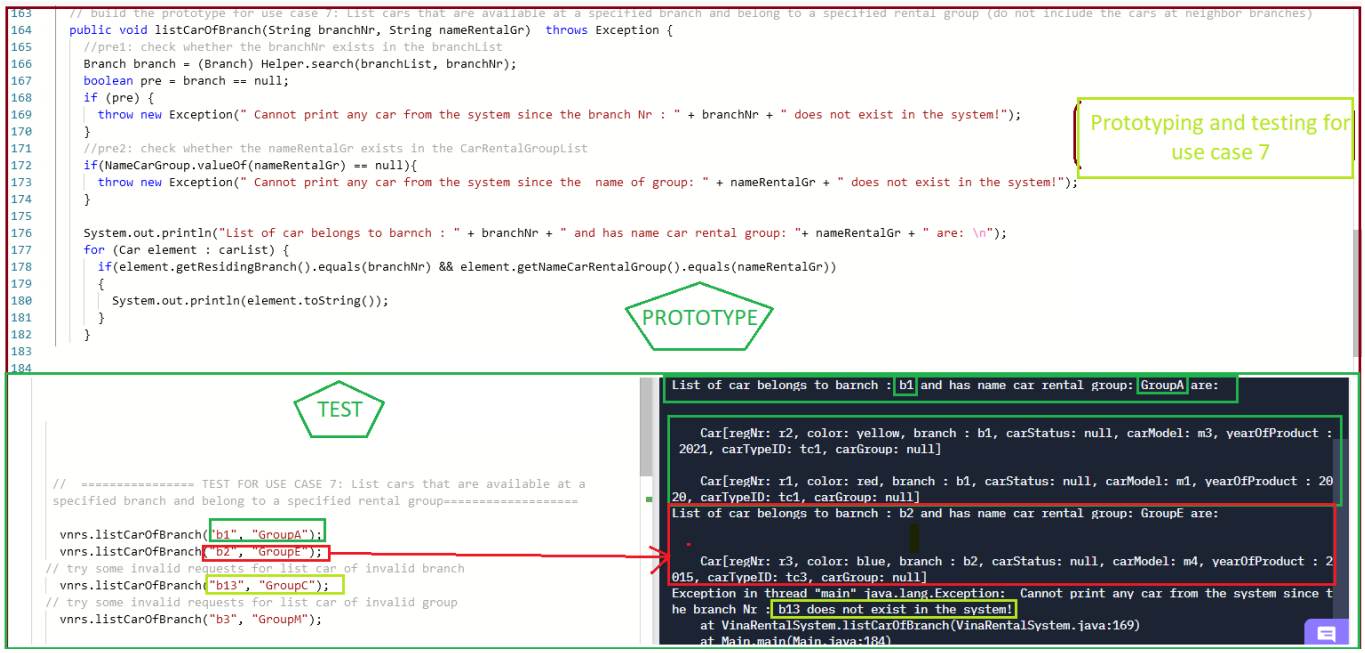
## 6.5 Protype and Testing for use case 5: Add a car.



## 6.6 Protype and Testing for use case 6: Add a customer.



## 6.7 Protype and Testing for use case 7: List cars that are available at a specified branch and belong to a specified rental group .



## 6.8 Protype and Testing for use case 8: Record the return of a car.

```
// build the prototype for use case 8: TEST FOR USE CASE 8: RECORD THE RETURN OF A CAR
// String rentalNr, Double endMillage,String returnBranch, String returnDate, String returnTime
public void recordReturnCar(String rentalNr, Double endMillage, String returnBranch, String returnDate, String returnTime)  throws Exception {
  Rental rental = (Rental) Helper.search(rentalList, rentalNr);
  boolean pre = rental == null;
  if (pre) {
    throw new Exception(" Cannot record the rental since the rental number: " + rentalNr + " does not exist in the system!");
  }

  // Check if the return branch is different with pickup branch, we will update residing branch of this car into return branch
  if(!returnBranch.equals(rental.getPickupBranch()))
  {
    Car car = (Car) Helper.search(carList, rental.getRegNr());
    car.setCarStatus(CarStatus.RETURNED);
    if(!car.getResidingBranch().equals(returnBranch))
    {
      car.setResidingBranch(returnBranch);
    }
  }
  rental.RecordRental(rentalNr, endMillage, returnBranch, returnDate, returnTime);
  rental.setRentalStatus(RentalStatus.RETURNED); // set status of rental become RETURNED
  System.out.println( rental.toString() + " has been record successfully!!!");

}
```

check pre-condition

update the residing branch of the car

```
//  ================ TEST FOR USE CASE 8: RECORD THE RETURN OF A CAR==================

  Rental rental1 = new Rental("rt1", "r1", 52.0, "b2", "15/10/2021", "17:30", RentalStatus.RESERVED);
  Rental rental2 = new Rental("rt2", "r2", 108.0, "b1", "15/11/2021", "14:30", RentalStatus.RESERVED);
  Rental rental3 = new Rental("rt3", "r3", 87.0, "b2", "27/12/2021", "8:10", RentalStatus.RESERVED);
  vnrs.addNewRental(rental1);
  vnrs.addNewRental(rental2);
  vnrs.addNewRental(rental2);

  // Record some returning car requests valid:
  vnrs.recordReturnCar("rt1", 176.5, "b2", "18/10/2021", "8:00");
  vnrs.recordReturnCar("rt2", 365.5, "b1", "21/11/2021", "14:20");

  // try some invalid requests for Recording some returning car requests
  // the request gives a rental number not exist in the retal list
  try
  {
    vnrs.recordReturnCar("rt10", 176.5, "b2", "18/10/2021", "8:00");
  }
  catch(Exception e)
  {
    System.out.println(e);
  }
  // uses the generic search on composite key
  System.out.println("\n ============== The vinarental system includes ============== \n");
  System.out.println(vnrs);

 }
}
```

adding some rental to the rental list

this is just the test case for UC8, we assume every input datas are correct

```
t in the branchList!
java.lang.IllegalArgumentException: No enum constant NameCarGroup.GroupF

    Rental[rentalNr: rt1, startMillage: 52.0, pickupBranch : b2, returnBranch: b2, pickupDate: Fri Oct 15 00:
00:00 UTC 2021, returnDate : Mon Oct 18 00:00:00 UTC 2021, pickupTime: Thu Jan 01 17:30:00 UTC 1970, returnT
me: Thu Jan 01 08:00:00 UTC 1970, status: RETURNED] has been record successfully!!!

    Rental[rentalNr: rt2, startMillage: 108.0, pickupBranch : b1, returnBranch: b1, pickupDate: Mon Nov 15 0
:00:00 UTC 2021, returnDate : Sun Nov 21 00:00:00 UTC 2021, pickupTime: Thu Jan 01 14:30:00 UTC 1970, return
ime: Thu Jan 01 14:20:00 UTC 1970, status: RETURNED] has been record successfully!!!
java.lang.Exception:  Cannot record the rental since the rental number: rt10 does not exist in the system!

 ============== The vinarental system includes ==============

VinaRentalSystem[
  branchList: [
    Branch[branchNr: b1, name: VinFast, address : Thu Duc, HCM, phoneNr: 0369872830],
    Branch[branchNr: b2, name: Hyundai Truong Chinh, address : Thu Duc, HCM, phoneNr: 0369872834]]
  carModelList: [
    CarModel[modelNr: m2, shortDescription: luxury car!, control : automatic, petrolConsumption: 26.0, number
OfDoor: 4, carGroup: GroupB],
    CarModel[modelNr: m1, shortDescription: luxury car!, control : automatic, petrolConsumption: 25.7, number
OfDoor: 4, carGroup: GroupA],
    CarModel[modelNr: m3, shortDescription: sport car!, control : automatic, petrolConsumption: 27.9, numberO
fDoor: 4, carGroup: GroupC],
    CarModel[modelNr: m4, shortDescription: medium truck!, control : manual, petrolConsumption: 26.5, numberO
fDoor: 4, carGroup: GroupE]]
  carList: [
    Car[regNr: r3, color: blue, branch : b2, carStatus: null, carModel: m4, yearOfProduct : 2015, carTypeID:
tc3, carGroup: null],
    Car[regNr: r2, color: yellow, branch : b1, carStatus: null, carModel: m3, yearOfProduct : 2021, carTypeID
: tc1, carGroup: null],
    Car[regNr: r1, color: red, branch : b1, carStatus: null, carModel: m1, yearOfProduct : 2020, carTypeID: t
c1, carGroup: null]]
  rentalList: [
    Rental[rentalNr: rt1, startMillage: 52.0, pickupBranch : b2, returnBranch: b2, pickupDate: Fri Oct 15 00:
00:00 UTC 2021, returnDate : Mon Oct 18 00:00:00 UTC 2021, pickupTime: Thu Jan 01 17:30:00 UTC 1970, returnTi
me: Thu Jan 01 08:00:00 UTC 1970, status: RETURNED],
    Rental[rentalNr: rt2, startMillage: 108.0, pickupBranch : b1, returnBranch: b1, pickupDate: Mon Nov 15 00
:00:00 UTC 2021, returnDate : Sun Nov 21 00:00:00 UTC 2021, pickupTime: Thu Jan 01 14:30:00 UTC 1970, returnT
ime: Thu Jan 01 14:20:00 UTC 1970, status: RETURNED]]
  customerList: [
```