# Systems Analysis and Design

Instructor : Huang, Chuen-Min

## Teamwork2 ver.1

## Group 10

| ID | Name |
|---|---|
| B10523016 | Frank |
| B10523044 | Nick |
| B10523011 | Lynn |
| B10523038 | Edward |
| B10523014 | Betty |
| B10523036 | Jeff |
| A10523031 | Asrock |
| B10523041 | Billy |

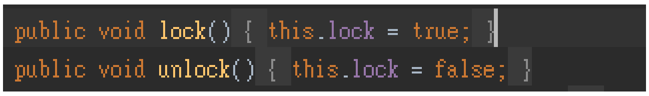
Date 2018/ 05 / 29

# CONTENT

## 1. Please explain the Law of Demeter (LoD) by using of your project.

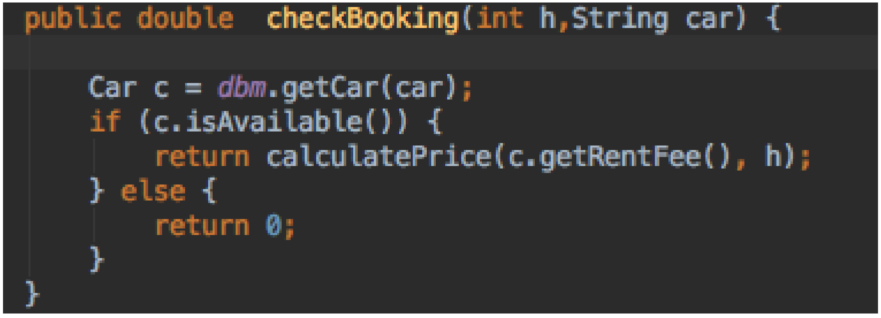
| Law of Demeter |
|---|
| (1) to itself (O itself) |
| (2) to objects contained in attributes of itself or a superclass (Any objects created/instantiated within M) |
| (3) to an object that is passed as a parameter to the method (M’s parameters) |
| (4) to an object that is created by the method (O's direct component objects) |
| (5) A global variable, accessible by O, in the scope of m |

(1) to itself (O itself)
Class Car



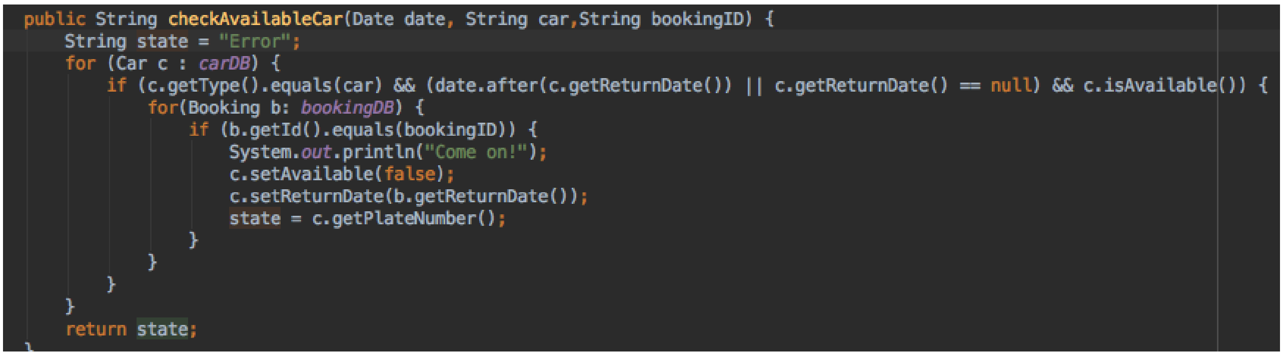
```
public void lock() { this.lock = true; }
public void unlock() { this.lock = false; }
```

(2) to objects contained in attributes of itself or a superclass (Any objects
Class BookingController



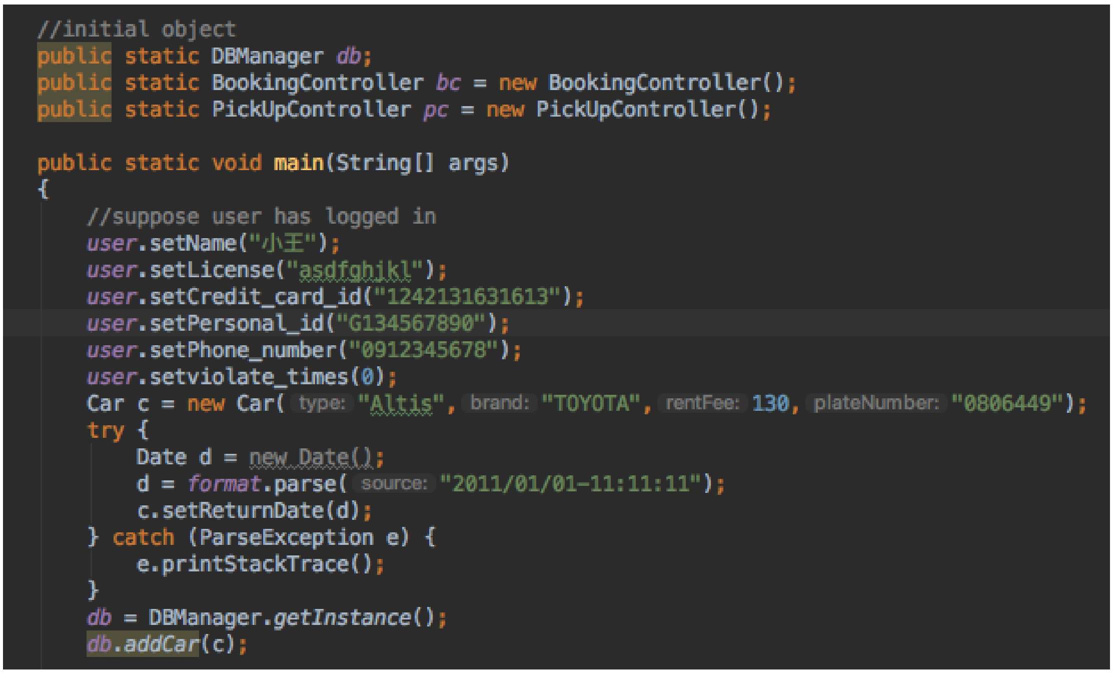
```
public double  checkBooking(int h,String car) {

    Car c = dbm.getCar(car);
    if (c.isAvailable()) {
        return calculatePrice(c.getRentFee(), h);
    } else {
        return 0;
    }
}
```

(3) to an object that is passed as a parameter to the method (M’s parameters)
Class DBmanager



```
public String checkAvailableCar(Date date, String car,String bookingID) {
    String state = "Error";
    for (Car c : carDB) {
        if (c.getType().equals(car) && (date.after(c.getReturnDate()) || c.getReturnDate() == null) && c.isAvailable()) {
            for(Booking b: bookingDB) {
                if (b.getId().equals(bookingID)) {
                    System.out.println("Come on!");
                    c.setAvailable(false);
                    c.setReturnDate(b.getReturnDate());
                    state = c.getPlateNumber();
                }
            }
        }
    }
    return state;
}
```

(4) to an object that is created by the method (O's direct component objects)
Class GUI

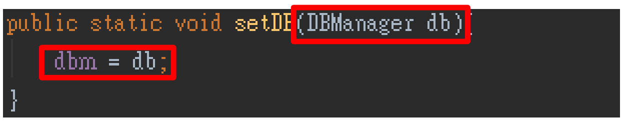


```
//initial object
public static DBManager db;
public static BookingController bc = new BookingController();
public static PickUpController pc = new PickUpController();

public static void main(String[] args)
{
    //suppose user has logged in
    user.setName("小王");
    user.setLicense("asdfghjkl");
    user.setCredit_card_id("1242131631613");
    user.setPersonal_id("G134567890");
    user.setPhone_number("0912345678");
    user.setviolate_times(0);
    Car c = new Car( type: "Altis", brand: "TOYOTA", rentFee: 130, plateNumber: "0806449");
    try {
        Date d = new Date();
        d = format.parse( source: "2011/01/01-11:11:11");
        c.setReturnDate(d);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    db = DBManager.getInstance();
    db.addCar(c);
```

## 2. Choose three pieces of your project to describe what types of the coupling they belong to.

■ **Data coupling**

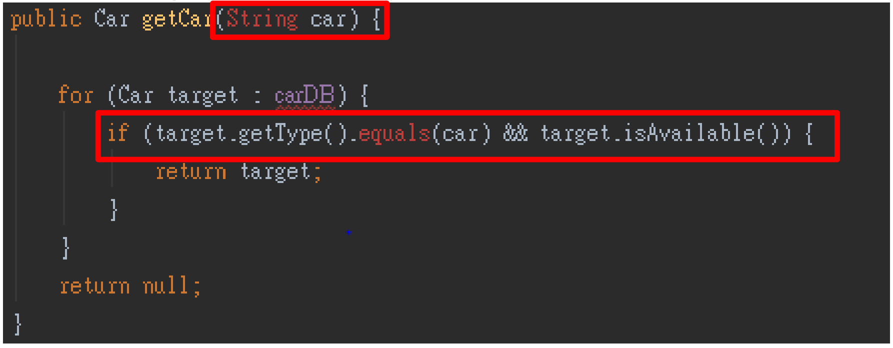
setDB() will get db and use it.
class BookingController



```
public static void setDB(DBManager db){
    dbm = db;
}
```

■ **Control coupling**

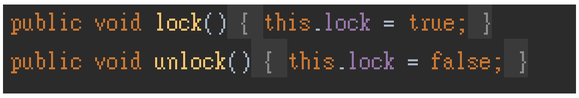
Parameter comes into getCar() will affect the next step of the program.
class DBManager



```
public Car getCar(String car) {

    for (Car target : carDB) {
        if (target.getType().equals(car) && target.isAvailable()) {
            return target;
        }
    }
    return null;
}
```

■ **Uncoupled**

These method od not related one another.
class Car



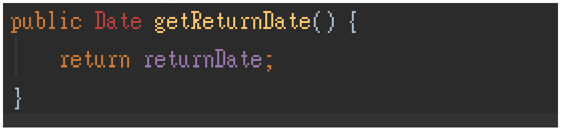
```
public void lock() { this.lock = true; }
public void unlock() { this.lock = false; }
```

## 3. choose three pieces of your project to describe what types of the cohesion they belong to.

### ■ Functional cohesion

The method performs a single problem-related task.

class Booking

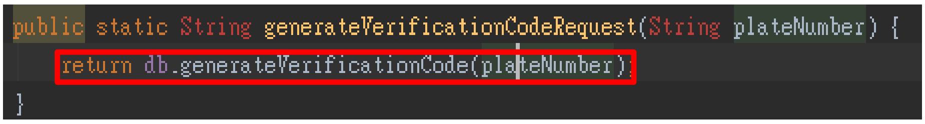


```
public Date getReturnDate() {
    return returnDate;
}
```

### ■ Sequential cohesion

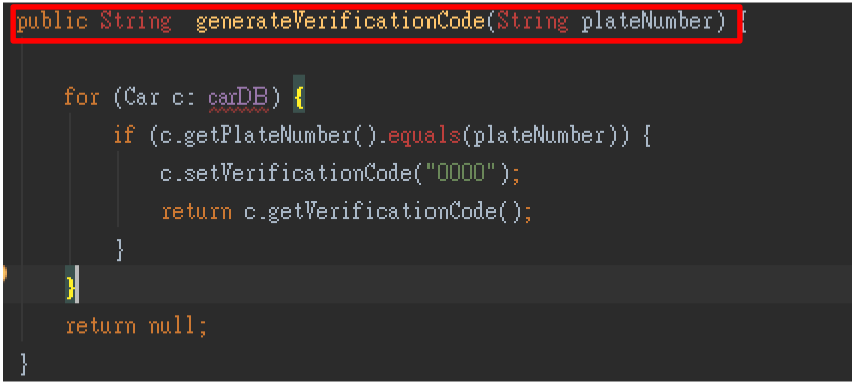
generateVerificationCodeRequest() will pass the plateNumber to generateVerificationCode() to perform its function.

Class GUI



```
public static String generateVerificationCodeRequest(String plateNumber) {
    return db.generateVerificationCode(plateNumber);
}
```

class DBManager

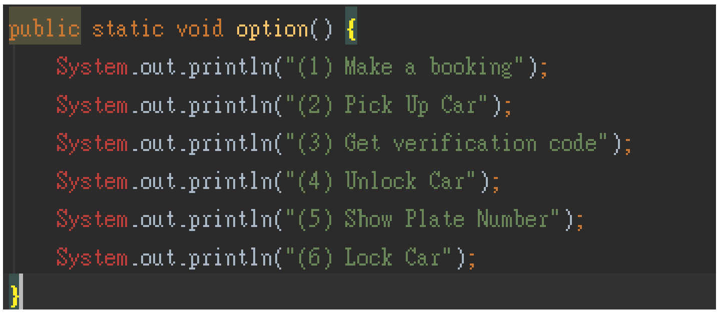


```
public String  generateVerificationCode(String plateNumber) {

    for (Car c: carDB) {
        if (c.getPlateNumber().equals(plateNumber)) {
            c.setVerificationCode("0000");
            return c.getVerificationCode();
        }
    }
    return null;
}
```

### ■ Temporal cohesion

The method lists the options at the same time.

Class GUI

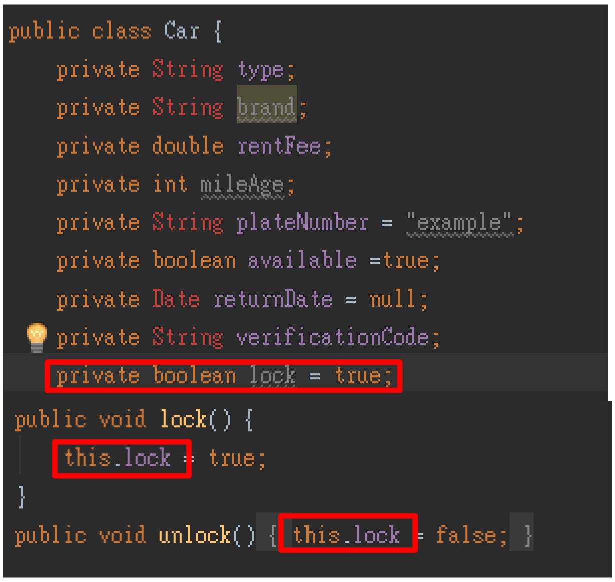


```
public static void option() {
    System.out.println("(1) Make a booking");
    System.out.println("(2) Pick Up Car");
    System.out.println("(3) Get verification code");
    System.out.println("(4) Unlock Car");
    System.out.println("(5) Show Plate Number");
    System.out.println("(6) Lock Car");
}
```

## 4. use three pieces of your project to describe what types of the connascence they belong to.

- **Connascence of Name:**

Class:Car

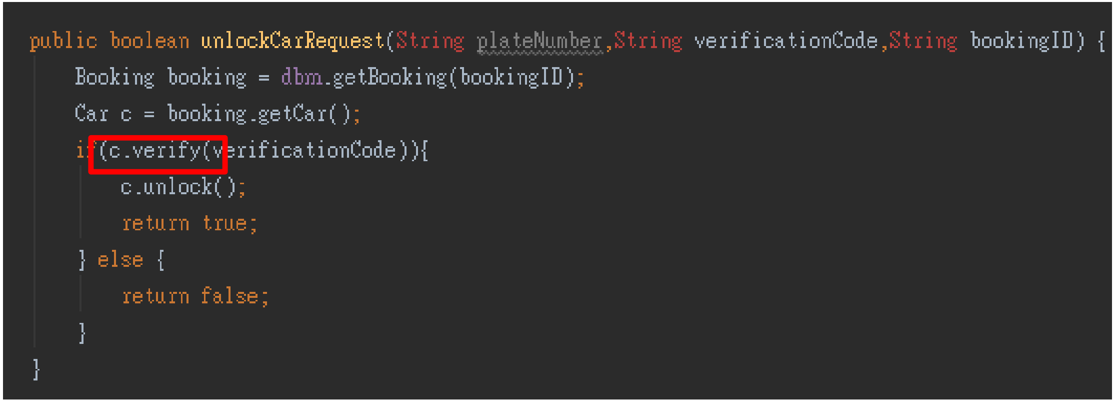


```
public class Car {
    private String type;
    private String brand;
    private double rentFee;
    private int mileAge;
    private String plateNumber = "example";
    private boolean available =true;
    private Date returnDate = null;
    private String verificationCode;
    private boolean lock = true;
public void lock() {
    this.lock = true;
}
public void unlock() { this.lock = false; }
```

If the name of lock has changed ,then these methods need to change to use the new name.

- **Connascence of Algorithm:**

Class:PickUpController

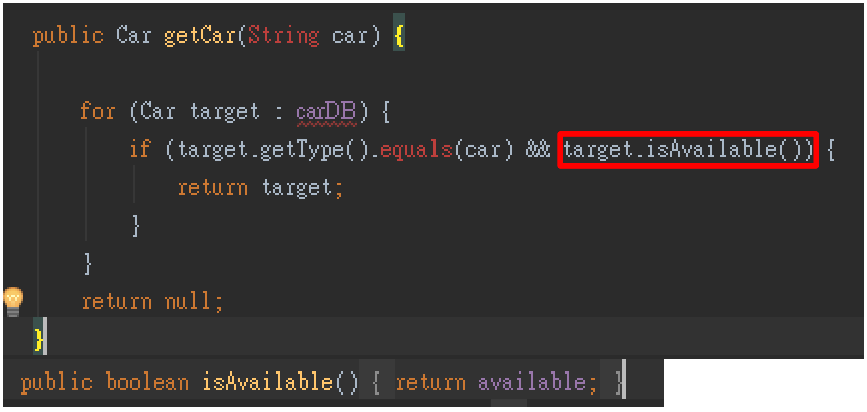


```
public boolean unlockCarRequest(String plateNumber,String verificationCode,String bookingID) {
    Booking booking = dbm.getBooking(bookingID);
    Car c = booking.getCar();
    if(c.verify(verificationCode)){
        c.unlock();
        return true;
    } else {
        return false;
    }
}
```

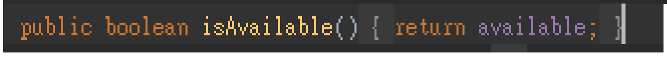
If the method of the Car:verify has changed , the result of unlockCarRequest will change.

- **Connascence of Convention:**

Class:DBManager



```
public Car getCar(String car) {

    for (Car target : carDB) {
        if (target.getType().equals(car) && target.isAvailable()) {
            return target;
        }
    }
    return null;
}
```

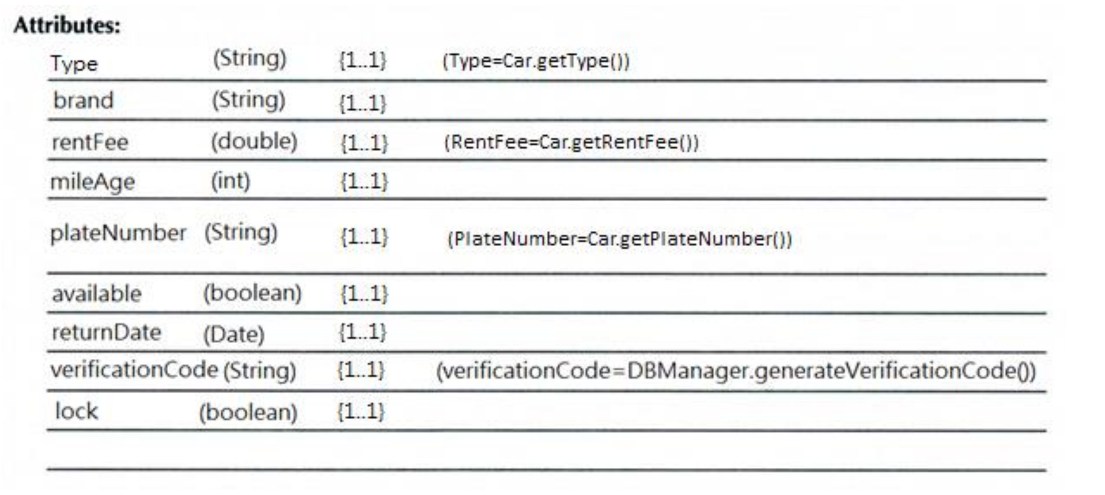


```
public boolean isAvailable() { return available; }
```

The method of DBManager:getCar(String car) need the value of car:available to return target or null.

## 5. Use one class from your project that can create a set of invariants and add them to the CRC card or the class diagram.

| | | |
|---|---|---|
| **Class Name:** Car | **ID:**6 | **Type:**Concrete,Domain |
| **Description:** Vehicles owned by the company | | **Associated Use Cases:**3,5 |
| **Responsibilities** | | **Collaborators** |
| setType | | |
| getType | | |
| isAvailable | | |
| setAvailable | | |
| getPlateNumber | | |
| getRentFee | | |
| lock | | |
| unlock | | |
| setReturnDate | | |
| getReturnDate | | |
| verify | | |
| setVerificationCode. | | |
| getVerificationCode | | |



**Attributes:**

| | | | |
|---|---|---|---|
| Type | (String) | {1..1} | (Type=Car.getType()) |
| brand | (String) | {1..1} | |
| rentFee | (double) | {1..1} | (RentFee=Car.getRentFee()) |
| mileAge | (int) | {1..1} | |
| plateNumber | (String) | {1..1} | (PlateNumber=Car.getPlateNumber()) |
| available | (boolean) | {1..1} | |
| returnDate | (Date) | {1..1} | |
| verificationCode | (String) | {1..1} | (verificationCode=DBManager.generateVerificationCode()) |
| lock | (boolean) | {1..1} | |

**Relationships:**

Generations(a-kind-of):

Aggregation(has-parts):

Other Associations: User,DBmanager

<<invariant>>

(verificationCode=DBManager.generateVerificationCode())

(Type=Car.getType())

(PlateNumber=Car.getPlateNumber())

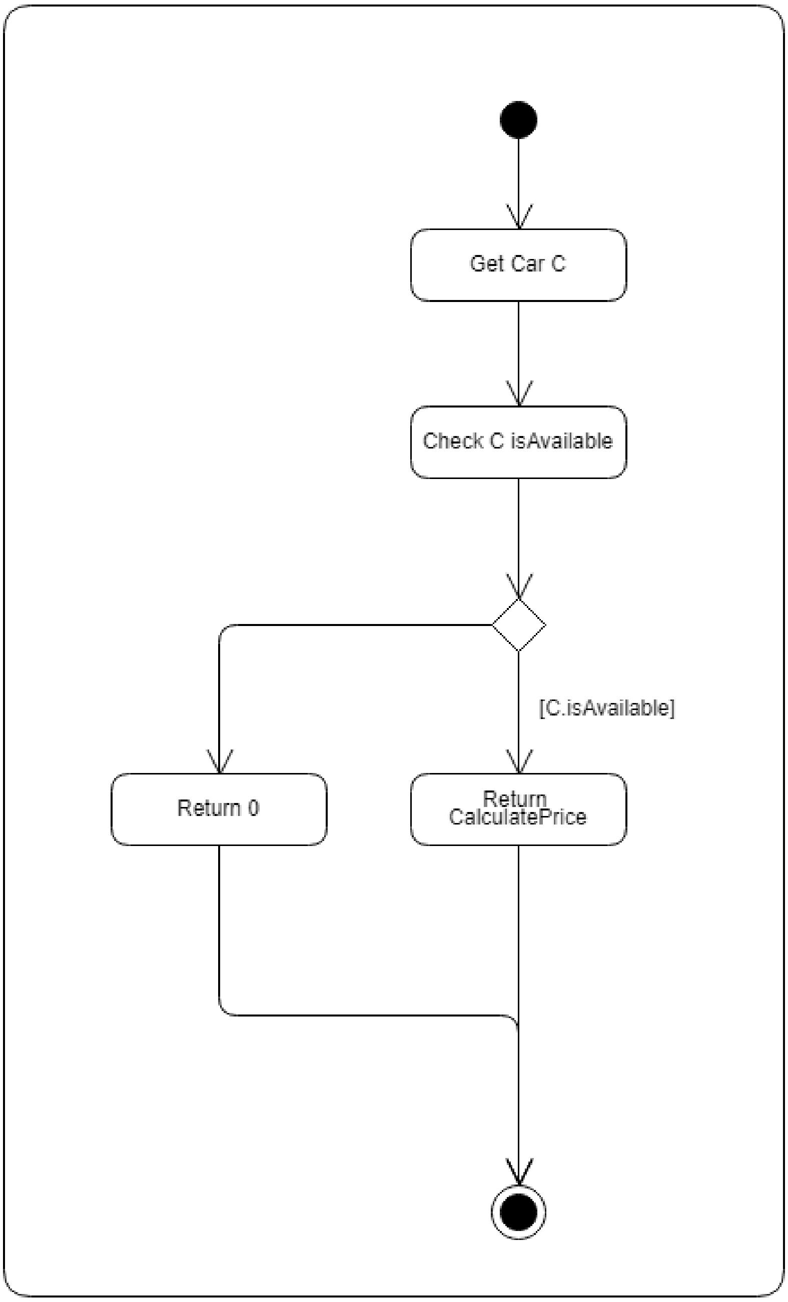
(RentFee=Car.getRentFee())

## 6. Use a method of a class from your project that can create a contract and describe its algorithm specification. Specify the pre- or post- condition and use both Structured English and an activity diagram to specify the algorithm.

| | | |
|---|---|---|
| Method Name: checkBooking | Class Name: BookingController | ID:1 |
| Clients (Consumers): GUI | | |
| Associated Use Case: Rent Car | | |
| Description of Responsibilities: Check whether the car is available | | |
| Arguments Received:<br>h int<br>car String | | |
| Type of Value Returned: double | | |
| Precondition: h && car != NULL | | |
| Postcondition: | | |

| | | |
|---|---|---|
| Method Name:<br>checkBooking | Class Name:<br>Booking Controller | ID:1 |
| Contract ID: | Programmer: | Data Due: |
| Programming Language: Java | | |
| Triggers/Events:<br>System wants to check booking if it is available | | |
| Arguments Received:<br>Data Type: | Notes: | |
| Int | rental hours | |
| String | car’s type | |
| Messages Sent & Argument Passed:<br>ClassName.MethodName: | Data Type: | Notes: |
| getCar(car) | String | |
| calculatePrice(RentFee,h) | double | |
| Arguments Returned:<br>Data Type: | Notes: | |
| Double | | |

| Algorithm Specification:<br><br>Get Car C with the car<br>IF C.isAvailable<br>        Return calculatePrice(C.getRentFee(),h)<br>ELSE<br>        Return 0 |
|---|
| Misc.Notes: |
| None |



## 7. Please evaluate any piece of your project in terms of cohesion, coupling, and connascence perspective.

According to question 2, 3, we can configure coupling and cohesion to help us evaluate our project below:

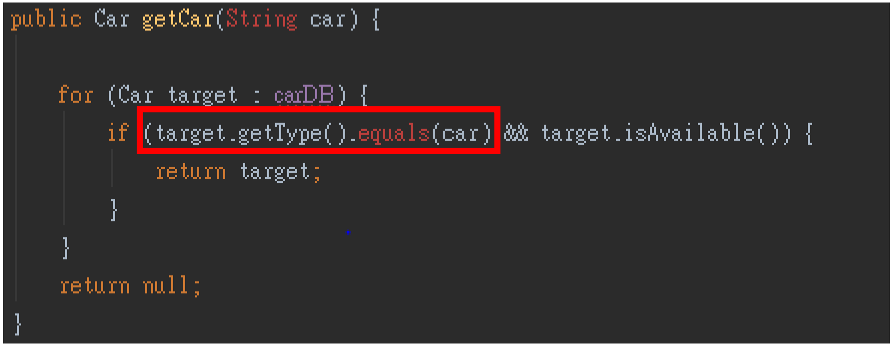
### ■ Coupling

In question 2, we find that the highest coupling of our project  is control, and this situation is not happened a lots in our java code, so we think our project has a medium level in coupling.

**Control coupling**

Parameter comes into getCar() will affect the next step of the program.

**class DBManager**



```
public Car getCar(String car) {

    for (Car target : carDB) {
        if (target.getType().equals(car) && target.isAvailable()) {
            return target;
        }
    }
    return null;
}
```

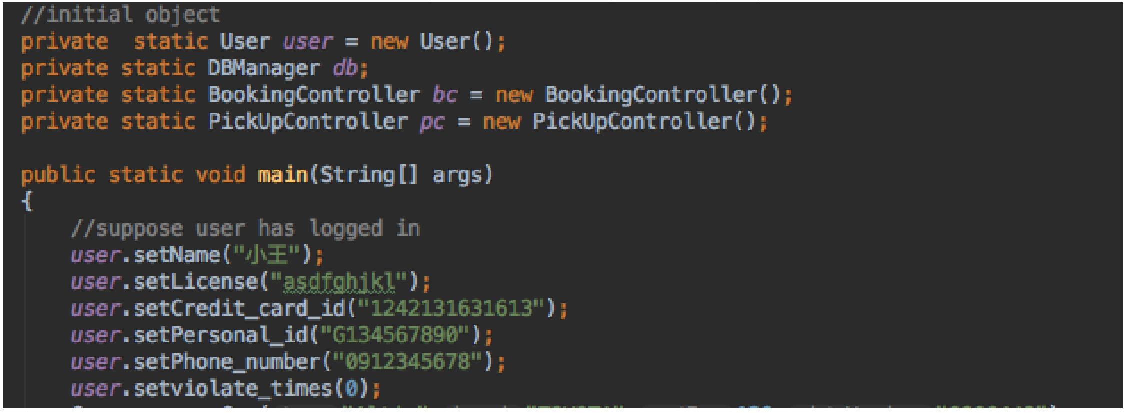
### ■ Cohesion

In question 3, the lowest cohesion is temporal , because we did not build the database practically, so we initialize the database in GUI, and cause this cohesion situation.

If we build the practical database will avoid this situation.

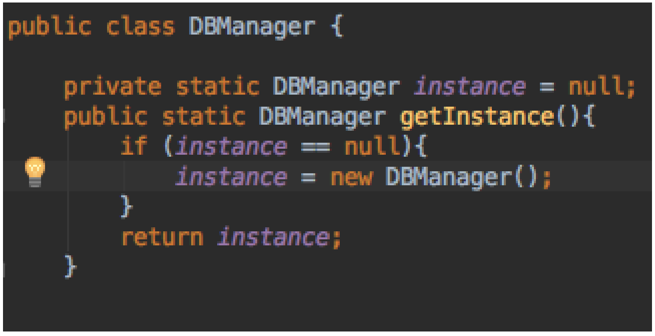
**Temporal Cohesion**

GUI will initialize a new DBManager when compile the project in main().



```
//initial object
private  static User user = new User();
private static DBManager db;
private static BookingController bc = new BookingController();
private static PickUpController pc = new PickUpController();

public static void main(String[] args)
{
    //suppose user has logged in
    user.setName("小王");
    user.setLicense("asdfghjkl");
    user.setCredit_card_id("1242131631613");
    user.setPersonal_id("G134567890");
    user.setPhone_number("0912345678");
    user.setviolate_times(0);
```

**class DBManager**



```
public class DBManager {

    private static DBManager instance = null;
    public static DBManager getInstance(){
        if (instance == null){
            instance = new DBManager();
        }
        return instance;
    }
```

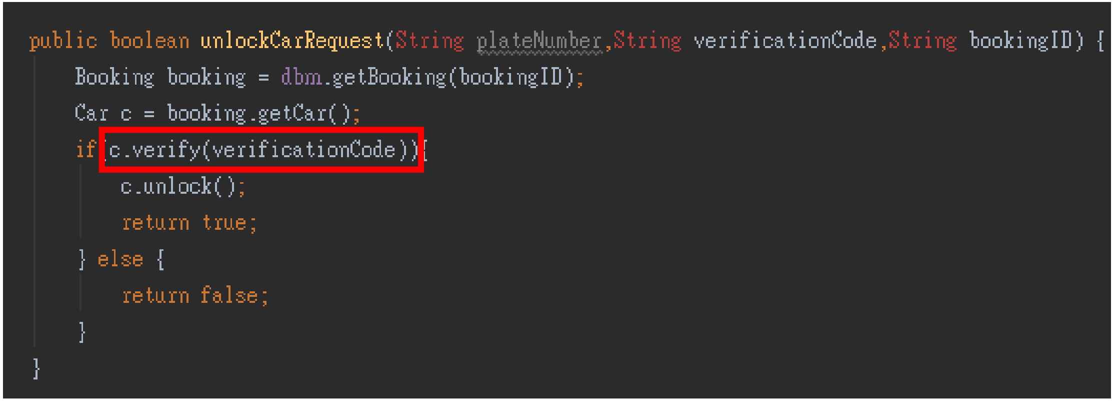
Otherwise, most methods in our project  are trend to high level(we classify functions in different method based on MVC development).

So that, combine class and method cohesion, our project has kind of a high level in cohesion.

- **Connascence**

In the question 4, we figure out some types of connascence.

Connascence of Algorithm:
Class PickUpController



```
public boolean unlockCarRequest(String plateNumber,String verificationCode,String bookingID) {
    Booking booking = dbm.getBooking(bookingID);
    Car c = booking.getCar();
    if(c.verify(verificationCode)){
        c.unlock();
        return true;
    } else {
        return false;
    }
}
```

If the method of the Car:verify has changed , the result of unlockCarRequest will change.

So if we take the verify()  in the unlockCarRequest() can solve the algorithm problem,
but it will cause the problem of cohesion.

## 8. Please evaluate any piece of your project in terms of cohesion, coupling, and connascence perspective.

When a foreign key value is used, it must reference a valid, existing primary key in the parent table. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.

The following relation schema of our project (Figure 1). To identify among user, car of rental, booking relationship in the Booking Table (Relation), existing user and car of rental in the User Table and Car of Rental must be referenced.

Thus, the Booking Table column (Attribute), also created in the Booking Table, that is a foreign key ("PERSONAL_ID", "PLATE_NUMBER"), these columns are special because its values are not newly created. Rather, these values must reference existing and identical values in the primary key column of another table. which are the "PERSONAL_ID" column of the User Table and "PLATE_NUMBER" column of the Car of Rental Table.

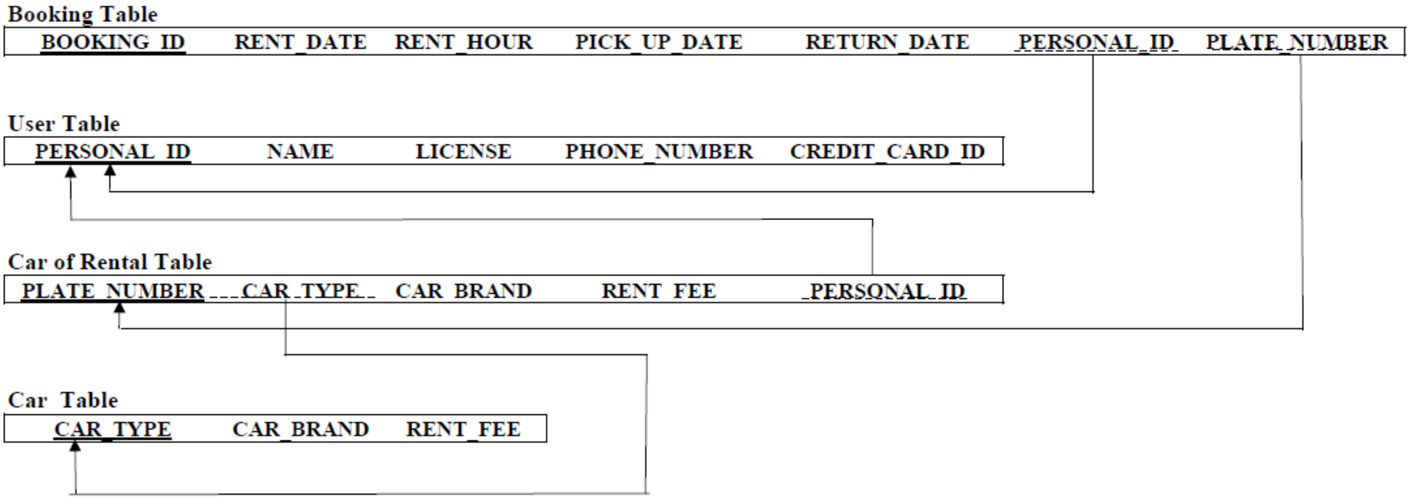
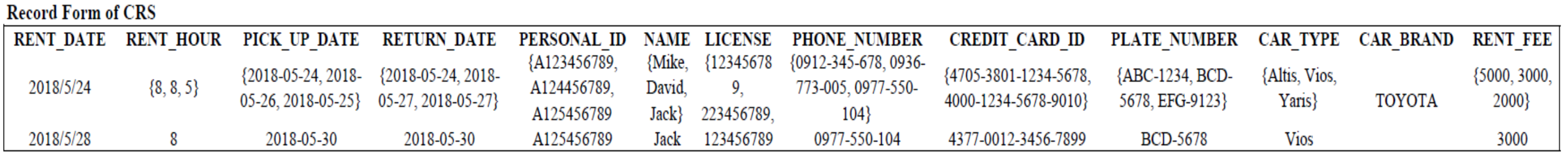


Figure 1 Relational Schema

## 9. Please evaluate any piece of your project in terms of cohesion, coupling, and connascence perspective.



Record Form of CRS

| RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | CREDIT_CARD_ID | PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2018/5/24 | {8, 8, 5} | {2018-05-24, 2018-05-26, 2018-05-25} | {2018-05-24, 2018-05-27, 2018-05-27} | {A123456789, A124456789, A125456789 | {Mike, David, Jack} | {123456789, 223456789, | {0912-345-678, 0936-773-005, 0977-550-104} | {4705-3801-1234-5678, 4000-1234-5678-9010} | {ABC-1234, BCD-5678, EFG-9123} | {Altis, Vios, Yaris} | TOYOTA | {5000, 3000, 2000} |
| 2018/5/28 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | Jack | 123456789 | 0977-550-104 | 4377-0012-3456-7899 | BCD-5678 | Vios | | 3000 |

0 Normal Form

Figure 2 Record Form of CRS (Unnormalized Form, UNF)

The form (Figure 1) has been repeated group; some records have a different number of columns from other records. So, this design that violates first normal form (1NF).

#First normal form enforces these criteria:

Remove the repeating fields.

Each table cell should contain a single value, each record needs to be unique.

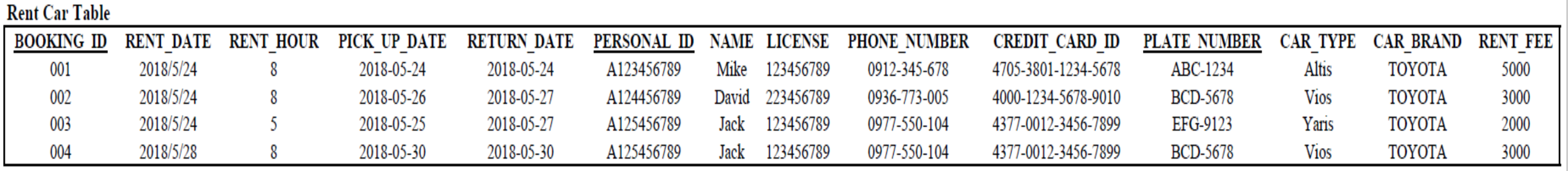
Identify each set of related data with a primary key (PK).

#A primary is a single column value used to identify a database record uniquely, it has the following attributes:

A primary key cannot be NULL.
A primary key value must be unique.
The primary key values cannot be changed.
The primary key must be given a value when a new record is inserted.

We break the values into atomic values, the following updated table (Figure 3) and it now satisfies the 1NF. By doing so, although a few values are getting repeated, but values for columns are now atomic for each row (record).

First Normal Form



Rent Car Table

| BOOKING_ID | RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | CREDIT_CARD_ID | PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 001 | 2018/5/24 | 8 | 2018-05-24 | 2018-05-24 | A123456789 | Mike | 123456789 | 0912-345-678 | 4705-3801-1234-5678 | ABC-1234 | Altis | TOYOTA | 5000 |
| 002 | 2018/5/24 | 8 | 2018-05-26 | 2018-05-27 | A124456789 | David | 223456789 | 0936-773-005 | 4000-1234-5678-9010 | BCD-5678 | Vios | TOYOTA | 3000 |
| 003 | 2018/5/24 | 5 | 2018-05-25 | 2018-05-27 | A125456789 | Jack | 123456789 | 0977-550-104 | 4377-0012-3456-7899 | EFG-9123 | Yaris | TOYOTA | 2000 |
| 004 | 2018/5/28 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | Jack | 123456789 | 0977-550-104 | 4377-0012-3456-7899 | BCD-5678 | Vios | TOYOTA | 3000 |

Figure 3 Rent Car Table in 1NF

Using the 1NF, data redundancy increases, as there will be many columns with the same data in multiple rows, but each row as a whole will be unique.
In Rent Car Table (Figure 2), “BOOKING_ID” is the primary key and will be unique for every row (record), hence we can use “BOOKING_ID” to fetch any row of data from this table.

When 003 is deleted, “PLATE_NUMBER”, “CAR_TYPE”, “CAR_BRAND”, “RENT_FEE”, etc. will also be deleted. As a result, the EFG-9123, “Yaris” is also deleted. The table (Figure 2) has some depend only on part of the primary key is a violation of second normal form (2NF).

Therefore, must transform the data in the first normal form (1NF) into the second normal form (2NF), to eliminate these problems.

#Second normal form enforces these criteria:

It should be in the first normal form (1NF).

Each column must depend on the primary key.

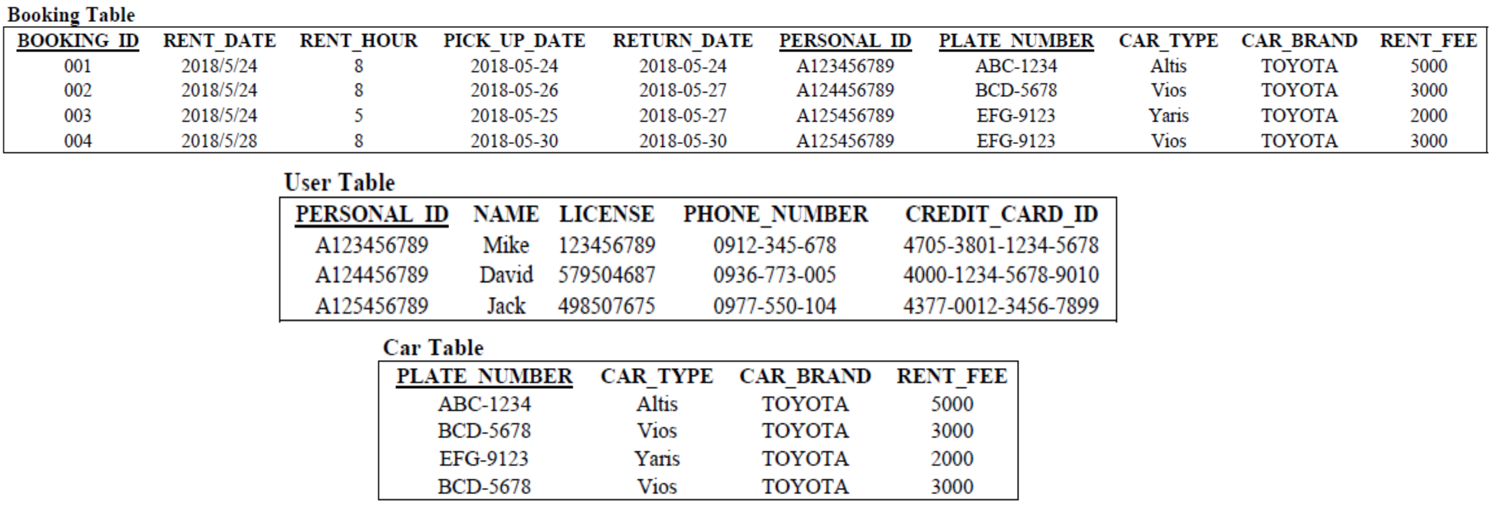
It should not have the partial dependency.

A functional dependency on the part of any candidate key is a violation of 2NF, in addition to the primary key.

We create another table for User and Car, which will have “PERSONAL_ID” for User Table and “PLATE_NUMBER” for Car Table will be the primary key.

After that, we also create another table for Booking, to store the information on rents obtained by Users in the respective booking. In the Booking Table, we are saving the “PERSONAL_ID” to know which user's rent records these are and “PLATE_NUMBER” to know for which car the rents are for, “PERSONAL_ID” + “PLATE_NUMBER” forms a candidate key for this table, which can be the primary key.

Second Normal Form



Booking Table

| BOOKING_ID | RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|---|---|---|---|---|---|
| 001 | 2018/5/24 | 8 | 2018-05-24 | 2018-05-24 | A123456789 | ABC-1234 | Altis | TOYOTA | 5000 |
| 002 | 2018/5/24 | 8 | 2018-05-26 | 2018-05-27 | A124456789 | BCD-5678 | Vios | TOYOTA | 3000 |
| 003 | 2018/5/24 | 5 | 2018-05-25 | 2018-05-27 | A125456789 | EFG-9123 | Yaris | TOYOTA | 2000 |
| 004 | 2018/5/28 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | EFG-9123 | Vios | TOYOTA | 3000 |

User Table

| PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | CREDIT_CARD_ID |
|---|---|---|---|---|
| A123456789 | Mike | 123456789 | 0912-345-678 | 4705-3801-1234-5678 |
| A124456789 | David | 579504687 | 0936-773-005 | 4000-1234-5678-9010 |
| A125456789 | Jack | 498507675 | 0977-550-104 | 4377-0012-3456-7899 |

Car Table

| PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|
| ABC-1234 | Altis | TOYOTA | 5000 |
| BCD-5678 | Vios | TOYOTA | 3000 |
| EFG-9123 | Yaris | TOYOTA | 2000 |
| BCD-5678 | Vios | TOYOTA | 3000 |

Figure 4 Rent Car Table in 2NF

Now we have a User Table with user information, Car Table with rents record and another table Booking for storing information of booking.

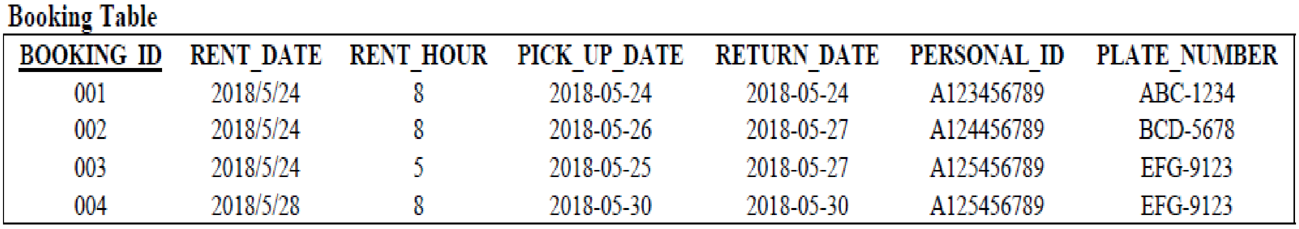
A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change.
In the Table (Figure 4). Changing the non-key column “CAR_TYPE” may change “RENT_FEE”. So, need to transform the table into third normal form (3NF).

#For a table to be in the third normal form (3NF), we enforce these criteria:
It should be in the second normal form (2NF).
And it should not have Transitive Dependency.

To move our 2NF table into 3NF, we need to again divide our table, we have a new table which stores “CAR_TYPE”.

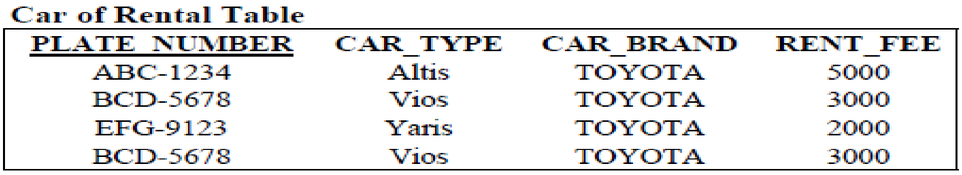
Third Normal Form

Figure 5 Rent Car Table in 3NF



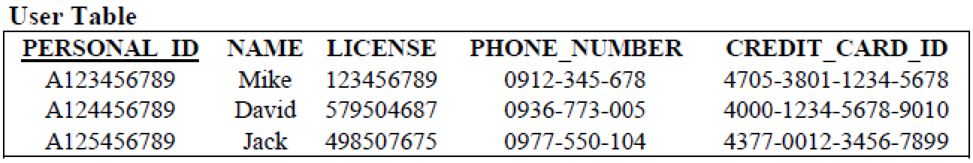
Booking Table

| BOOKING_ID | RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | PLATE_NUMBER |
|---|---|---|---|---|---|---|
| 001 | 2018/5/24 | 8 | 2018-05-24 | 2018-05-24 | A123456789 | ABC-1234 |
| 002 | 2018/5/24 | 8 | 2018-05-26 | 2018-05-27 | A124456789 | BCD-5678 |
| 003 | 2018/5/24 | 5 | 2018-05-25 | 2018-05-27 | A125456789 | EFG-9123 |
| 004 | 2018/5/28 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | EFG-9123 |



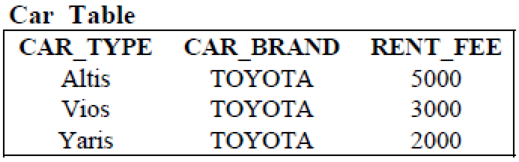
Car of Rental Table

| PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|
| ABC-1234 | Altis | TOYOTA | 5000 |
| BCD-5678 | Vios | TOYOTA | 3000 |
| EFG-9123 | Yaris | TOYOTA | 2000 |
| BCD-5678 | Vios | TOYOTA | 3000 |



User Table

| PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | CREDIT_CARD_ID |
|---|---|---|---|---|
| A123456789 | Mike | 123456789 | 0912-345-678 | 4705-3801-1234-5678 |
| A124456789 | David | 579504687 | 0936-773-005 | 4000-1234-5678-9010 |
| A125456789 | Jack | 498507675 | 0977-550-104 | 4377-0012-3456-7899 |

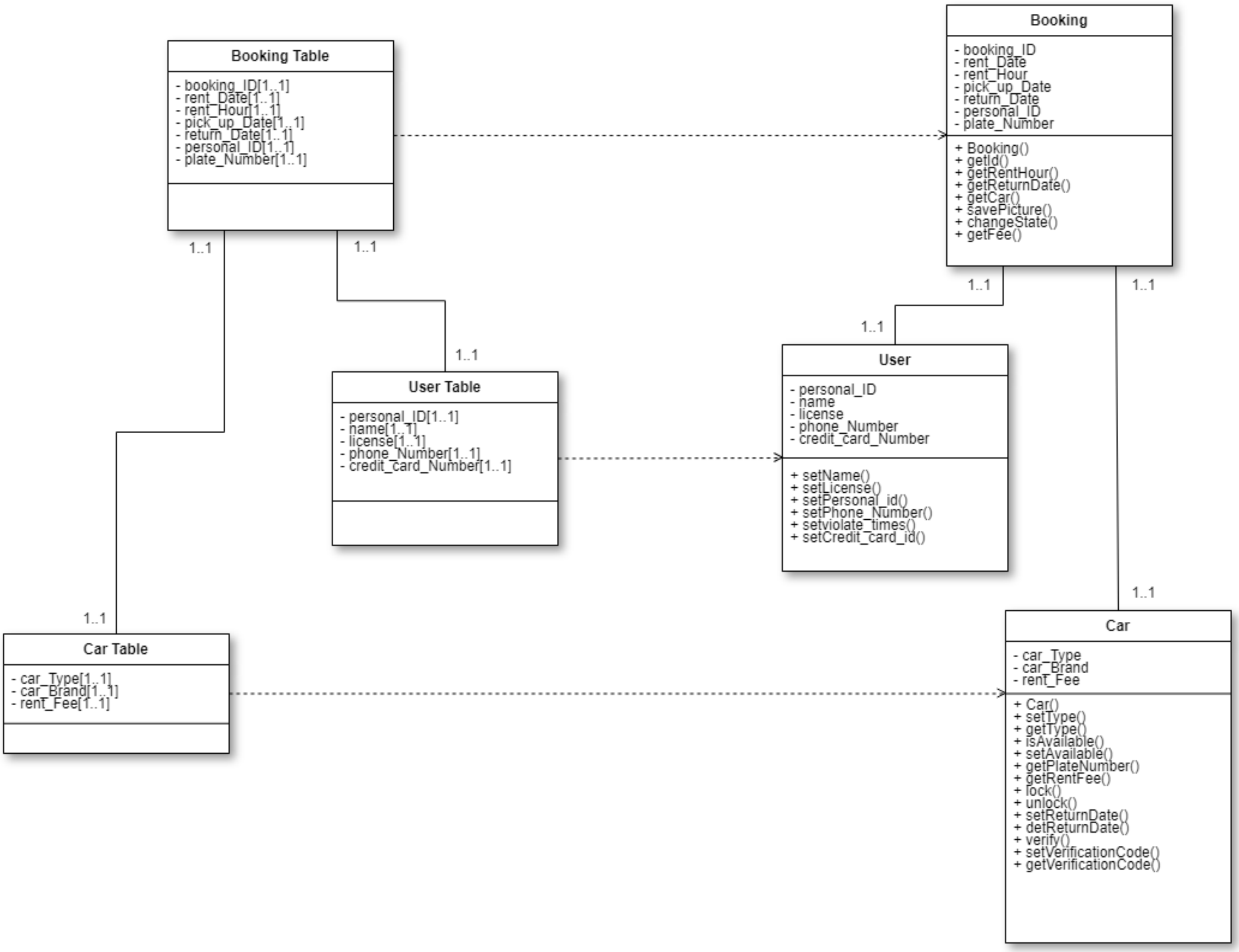


Car Table

| CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|
| Altis | TOYOTA | 5000 |
| Vios | TOYOTA | 3000 |
| Yaris | TOYOTA | 2000 |

There are no transitive functional dependencies, and hence our table is in 3NF, in the Car Table “CAR_TYPE” is the primary key, and in the Car of Rental Table “CAR_TYPE” is foreign to the primary key in Car Table.

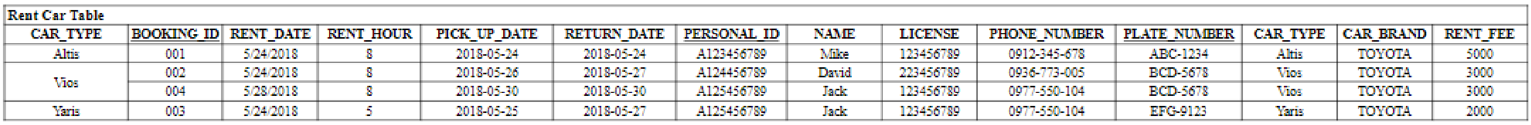
## 10. Please evaluate any piece of your project in terms of cohesion, coupling, and connascence perspective.



We denormalized Car of Rental .

Because when we search the booking, it can direct display what car I order and how much I have to pay.

## 11. Please evaluate any piece of your project in terms of cohesion, coupling, and connascence perspective.

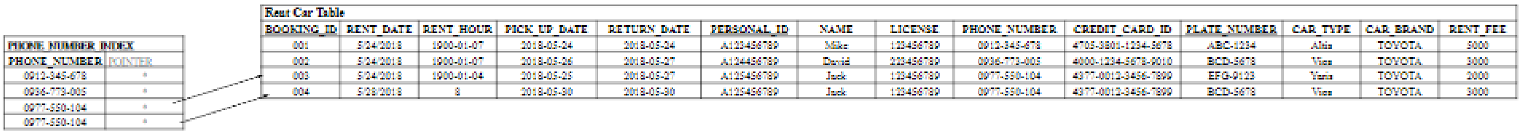


Rent Car Table

| CAR_TYPE | BOOKING_ID | RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Altis | 001 | 5/24/2018 | 8 | 2018-05-24 | 2018-05-24 | A123456789 | Mike | 123456789 | 0912-345-678 | ABC-1234 | Altis | TOYOTA | 5000 |
| Vios | 002 | 5/24/2018 | 8 | 2018-05-26 | 2018-05-27 | A124456789 | David | 223456789 | 0936-773-005 | BCD-5678 | Vios | TOYOTA | 3000 |
|  | 004 | 5/28/2018 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | Jack | 123456789 | 0977-550-104 | BCD-5678 | Vios | TOYOTA | 3000 |
| Yaris | 003 | 5/24/2018 | 5 | 2018-05-25 | 2018-05-27 | A125456789 | Jack | 123456789 | 0977-550-104 | EFG-9123 | Yaris | TOYOTA | 2000 |

We will create two index between Phone_Number and Booking_ID, because if we know the Phone_Number we can search the booking more quickly that we recently ordered.

Like we ordered a car, and final the GUI display the booking information, and we may not remember the info. If we want to search the booking and check the time, we can just input ourself phone_number, and can not to remember other info like personal_id.



PHONE_NUMBER_INDEX

| PHONE_NUMBER | POINTER |
|---|---|
| 0912-345-678 | • |
| 0936-773-005 | • |
| 0977-550-104 | • |
| 0977-550-104 | • |

Rent Car Table

| BOOKING_ID | RENT_DATE | RENT_HOUR | PICK_UP_DATE | RETURN_DATE | PERSONAL_ID | NAME | LICENSE | PHONE_NUMBER | CREDIT_CARD_ID | PLATE_NUMBER | CAR_TYPE | CAR_BRAND | RENT_FEE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 001 | 5/24/2018 | 1900-01-07 | 2018-05-24 | 2018-05-24 | A123456789 | Mike | 123456789 | 0912-345-678 | 4705-3801-1234-5678 | ABC-1234 | Altis | TOYOTA | 5000 |
| 002 | 5/24/2018 | 1900-01-07 | 2018-05-26 | 2018-05-27 | A124456789 | David | 223456789 | 0936-773-005 | 4000-1234-5678-9010 | BCD-5678 | Vios | TOYOTA | 3000 |
| 003 | 5/24/2018 | 1900-01-04 | 2018-05-25 | 2018-05-27 | A125456789 | Jack | 123456789 | 0977-550-104 | 4377-0012-3456-7899 | EFG-9123 | Yaris | TOYOTA | 2000 |
| 004 | 5/28/2018 | 8 | 2018-05-30 | 2018-05-30 | A125456789 | Jack | 123456789 | 0977-550-104 | 4377-0012-3456-7899 | BCD-5678 | Vios | TOYOTA | 3000 |

## 12. participate

| ID | Name | Participation | Main responsibility |
|---|---|---|---|
| B10523016 | Frank | 100% | Meeting together to discuss completion |
| B10523044 | Nick | 100% | |
| B10523011 | Lynn | 100% | |
| B10523038 | Edward | 100% | |
| B10523014 | Betty | 100% | |
| B10523036 | Jeff | 100% | |
| A10523031 | Asrock | 100% | |
| B10523041 | Billy | 100% | |