Inf2C Software Engineering 2019-20 Coursework 2

Capturing requirements for a federalised bike rental system

Group 100:
Xi Chen - s1817189
Jingwen Pan - s1827818

2.1-Introduction

The bike rental system is an easy-use, convenient and environmentally-friendly bike renting tool. It is designed for people who live in or visit the Scotland and helps them go out in a healthy and eco-friendly way. After a simple registration, users of this system can get quotes from different bike providers and choose the bike they like. After the secure payment, a confirmation message will be sent to user for checking the booking. Users can collect their bike in person or by delivery. Eventually, users need to return the bike on the same mode that they have chosen to collect the bike before the rental expiry date. If the users are bike providers, then after registration of their bike shops, users are required to register the bike including bike types, daily rental price and so on. Bike providers are allowed to set their own deposit policy and cooperate with each other as partners.

Self-assessment:

This introduction might not be the expected introduction. Also, it might be too long as an introduction.

2.2-Static model

2.2.2a High-level description

There are 11 classes (including the extension modulo) for main interactions inside the system; Quotes, BikeProvider, Bike, Depreciation, Customer, Controller, Driver, DeliveryServiceFactory, RentalNeeds, Booking and Confirmation. The other 5 classes; OpenHours, BikeShop, BikeType, DateRange, Location, are used inside that 11 classes being treated as attributes inside e.g. Each Bike-Provider has a list of Bike. Some classes have get or set or both method inside for their attributes to enable the future updates. In our design, the class Bike-Provider and class Depreciation implement the interface ValuationPolicy because these 2 classes contains attributes in type Bike or list of Bike and the replacement value of each bike type is calculated with the valuation policy defined in interface ValuationPolicy.

The controller takes responsibility of harmonizing interactions between different classes e.g. filterQuote() method in Controller class takes a parameter customerA whose type is Customer. The controller through customerA.getRental-Needs gets customer requirements in a type called RentalNeeds and based on these requirements, collects the available or suitable quotes from list of bike providers now registered on the system by returning an array list of quotes.

Also, for the relationship between a bike provider and same provider who is the original provider of a bike, the partnership is designed inside the BikeProvider class using HashMap type for one-to-one correspondence.(One-to-one correspondence means that each bike provider can only make 1 partnership agreement with another 1 bike provider at one time.) This avoids the ugly design of triple provider classes in the UML diagram(triple provider classes design means that define the type BikeProvider, OriginalBikeProvider, PartnerBikeProvider separately). What's more, it automatically updates the attributes if the attributes are changed in the related class. For example, a bike shop changes its name in BikeShop class, then in the partnership list of this bike shop, its name changes in consistence.

Each customer has 2 ways to book quotes, by methods in controller bookSingleQuote() and bookMultiQuote() separately. A list of bookings with allocated drivers(Each booking is randomly allocated with 1 driver) will be returned to Confirmation class such that a confirmation message will be generated and returned to the controller. The controller calls method sendConfirmation() to send out a confirmation message.

The controller calls the method deliveryBike() to simulate the action in a delivery. Inside the deliveryBIke() method, the attribute deliveryCompany in Controller class calls the method of its type setupMockDeliveryService() to actually perform the delivery. Once the bike is delivered to a shop, controller calls the method returnBike() which returns a boolean value to check if the bike is returned to the original bike provider. If not, then controller shall call this method several times until the bike is returned to the original bike provider.

Self-assessment:

UML class diagram notations are used. However, the high-level description might not be detailed enough because it doesn't explain the whole design throughly.

2.2.2b Design choices/resolution of ambiguities

*If customers don't want to pay anymore or the payment fails during the payment process, the payment should be interrupted and cancelled. In the design, the controller class takes charge of calling the method interrupt_payment() to interrupt the payment in this situation.

*If a customer wants to book multiple quotes but the quotes list returned available for each booking might contains quotes from different bike providers while it is expected that the customer can only book multiple quotes in one booking from same bike providers -
There is a method filterQuotesFromSameProvider() in the Controller class in charge of filtering quotes of the same provider once the customer decides to book a specific quote and wants to book more quotes, by taking 2 parameters; the available quotes list from different bike providers and also the specific quote the customer decides to book.

*If the customer want to book not only a single quote but multiple quotes (from the same bike provider.) -
As UML diagram shown, the Controller class has 2 methods; bookSingleQuote() and bookMultiQuote(). These two methods function as their name.

Self-assessment:
Resolutions of ambiguities are discussed.

2.3-Dynamic models

(see diagrams at the end.)
Self-assessment:
Basic notations of the sequence diagram and communication diagram are used. Diagrams might have some mistakes which we didn't found out.

2.4-Conformance to requirements

The system states stated in coursework1 file "report.pdf" are consistent to the attributes defined in each class in the UML diagram. What's more, the ambiguities found in coursework1 are discussed in this coursework section 2.2.2b.

Self-assessment:
The coursework1 file "report.pdf" is somehow consistent with this coursework. There might be some inconsistencies in between but we didn't noticed that.

2.5-Design extensions

2.5.1 Custom pricing
2.5.2 Custom deposits/valuation
2.5.3 Modified design
(see the diagram.)

self-assessment:
The appended interface diagrams might be incorrect.[3’]

2.6-Self-assessment

The self-assessment comments are not objective and over-estimated.[5’]

3.Justification of good Software Engineering practice

1. For each class, the attributes defined inside are allowed to be changed from outside because they have both set and get method for future updates so that people are possible to make changes during the use of this system e.g. A bike provider wants to change its shop name and the method setShopName() in class BikeShop enable he/she to do so.

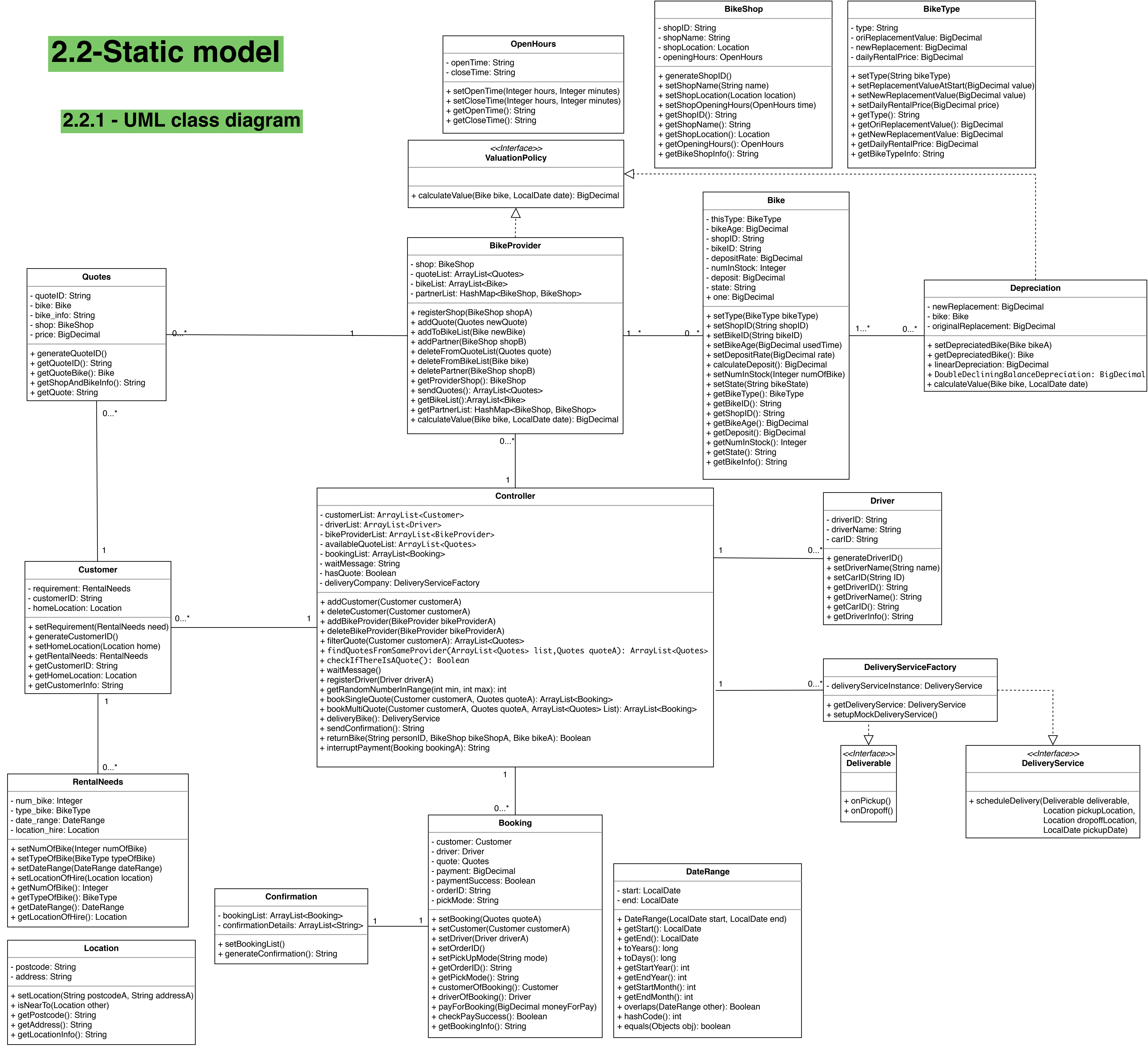
2.The coursework1 file is consistent with this coursework so that the system is designed consistently through the whole project.

3.Although we think that all the things in the design are necessary, it is possible that we don’t have a good consideration of some factors that are actually unnecessary for the design.

self-assessment:
The justifications based on designers themselves (which are us) are subjective.

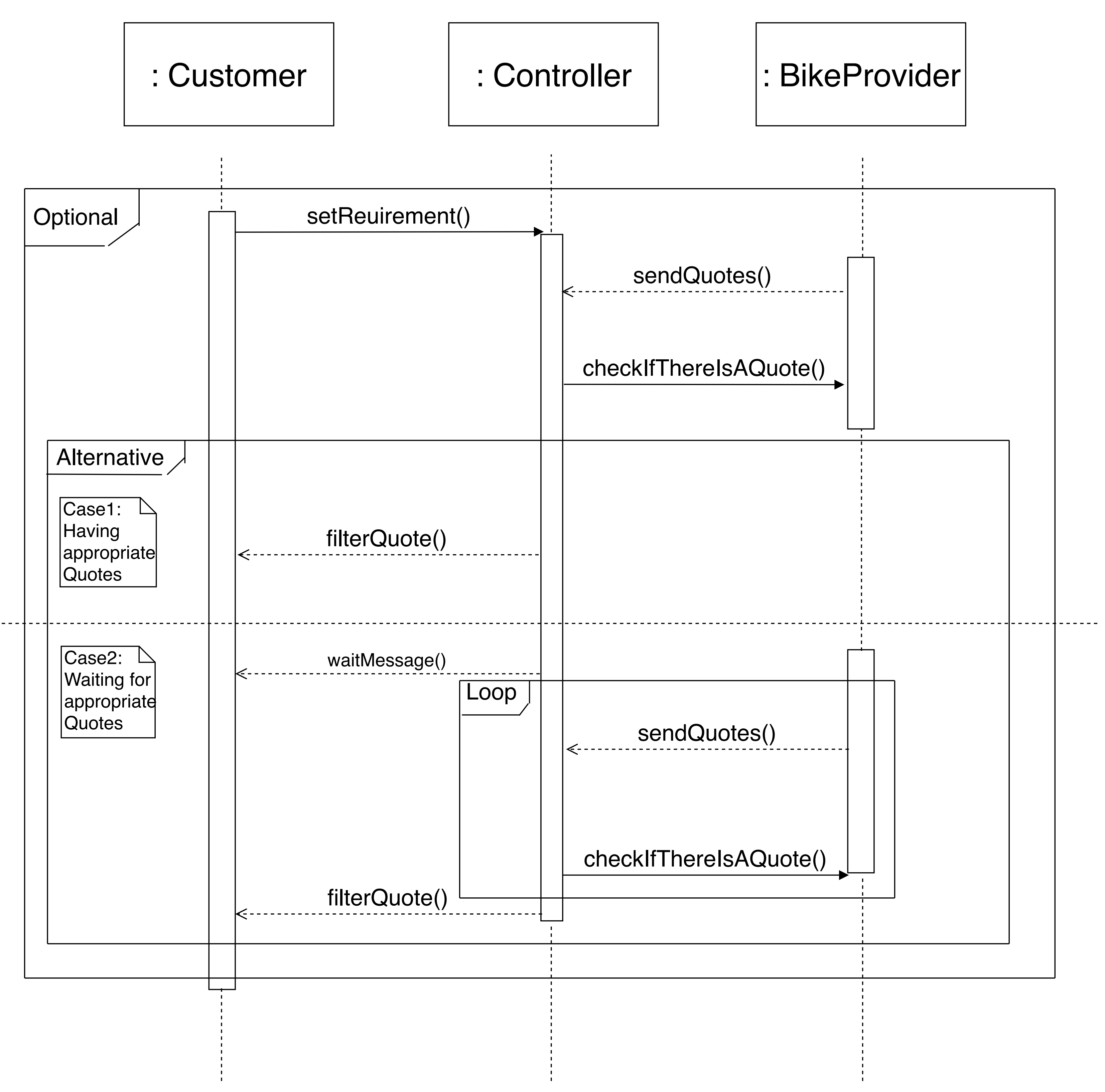
2.2-Static model

2.2.1 - UML class diagram


OpenHours
- openTime: String
- closeTime: String
+ setOpenTime(Integer hours, Integer minutes)
+ setCloseTime(Integer hours, Integer minutes)
+ getOpenTime(): String
+ getCloseTime(): String
BikeShop
- shopID: String
- shopName: String
- shopLocation: Location
- openingHours: OpenHours
+ generateShopID()
+ setShopName(String name)
+ setShopLocation(Location location)
+ setShopOpeningHours(OpenHours time)
+ getShopID(): String
+ getShopName(): String
+ getShopLocation(): Location
+ getOpeningHours(): OpenHours
+ getBikeShopInfo(): String
BikeType
- type: String
- oriReplacementValue: BigDecimal
- newReplacement: BigDecimal
- dailyRentalPrice: BigDecimal
+ setType(String bikeType)
+ setReplacementValueAtStart(BigDecimal value)
+ setNewReplacementValue(BigDecimal value)
+ setDailyRentalPrice(BigDecimal price)
+ getType(): String
+ getOriReplacementValue(): BigDecimal
+ getNewReplacementValue: BigDecimal
+ getDailyRentalPrice: BigDecimal
+ getBikeTypeInfo: String
<<Interface>>
ValuationPolicy
+ calculateValue(Bike bike, LocalDate date): BigDecimal
BikeProvider
- shop: BikeShop
- quoteList: ArrayList<Quotes>
- bikeList: ArrayList<Bike>
- partnerList: HashMap<BikeShop, BikeShop>
+ registerShop(BikeShop shopA)
+ addQuote(Quotes newQuote)
+ addToBikeList(Bike newBike)
+ addPartner(BikeShop shopB)
+ deleteFromQuoteList(Quotes quote)
+ deleteFromBikeList(Bike bike)
+ deletePartner(BikeShop shopB)
+ getProviderShop(): BikeShop
+ sendQuotes(): ArrayList<Quotes>
+ getBikeList():ArrayList<Bike>
+ getPartnerList: HashMap<BikeShop, BikeShop>
+ calculateValue(Bike bike, LocalDate date): BigDecimal
Bike
- thisType: BikeType
- bikeAge: BigDecimal
- shopID: String
- bikeID: String
- depositRate: BigDecimal
- numInStock: Integer
- deposit: BigDecimal
- state: String
+ one: BigDecimal
+ setType(BikeType bikeType)
+ setShopID(String shopID)
+ setBikeID(String bikeID)
+ setBikeAge(BigDecimal usedTime)
+ setDepositRate(BigDecimal rate)
+ calculateDeposit(): BigDecimal
+ setNumInStock(Integer numOfBike)
+ setState(String bikeState)
+ getBikeType(): BikeType
+ getBikeID(): String
+ getShopID(): String
+ getBikeAge(): BigDecimal
+ getDeposit(): BigDecimal
+ getNumInStock(): Integer
+ getState(): String
+ getBikeInfo(): String
Depreciation
- newReplacement: BigDecimal
- bike: Bike
- originalReplacement: BigDecimal
+ setDepreciatedBike(Bike bikeA)
+ getDepreciatedBike(): Bike
+ linearDepreciation: BigDecimal
+ DoubleDecliningBalanceDepreciation: BigDecimal
+ calculateValue(Bike bike, LocalDate date)
Quotes
- quoteID: String
- bike: Bike
- bike_info: String
- shop: BikeShop
- price: BigDecimal
+ generateQuoteID()
+ getQuoteID(): String
+ getQuoteBike(): Bike
+ getShopAndBikeInfo(): String
+ getQuote: String
0...*
1
1...*
0...*
1...*
0...*
0...*
1
0...*
1
Customer
- requirement: RentalNeeds
- customerID: String
- homeLocation: Location
+ setRequirement(RentalNeeds need)
+ generateCustomerID()
+ setHomeLocation(Location home)
+ getRentalNeeds: RentalNeeds
+ getCustomerID: String
+ getHomeLocation: Location
+ getCustomerInfo: String
0...*
1
Controller
- customerList: ArrayList<Customer>
- driverList: ArrayList<Driver>
- bikeProviderList: ArrayList<BikeProvider>
- availableQuoteList: ArrayList<Quotes>
- bookingList: ArrayList<Booking>
- waitMessage: String
- hasQuote: Boolean
- deliveryCompany: DeliveryServiceFactory
+ addCustomer(Customer customerA)
+ deleteCustomer(Customer customerA)
+ addBikeProvider(BikeProvider bikeProviderA)
+ deleteBikeProvider(BikeProvider bikeProviderA)
+ filterQuote(Customer customerA): ArrayList<Quotes>
+ findQuotesFromSameProvider(ArrayList<Quotes> list,Quotes quoteA): ArrayList<Quotes>
+ checkIfThereIsAQuote(): Boolean
+ waitMessage()
+ registerDriver(Driver driverA)
+ getRandomNumberInRange(int min, int max): int
+ bookSingleQuote(Customer customerA, Quotes quoteA): ArrayList<Booking>
+ bookMultiQuote(Customer customerA, Quotes quoteA, ArrayList<Quotes> List): ArrayList<Booking>
+ deliveryBike(): DeliveryService
+ sendConfirmation(): String
+ returnBike(String personID, BikeShop bikeShopA, Bike bikeA): Boolean
+ interruptPayment(Booking bookingA): String
1
0...*
Driver
- driverID: String
- driverName: String
- carID: String
+ generateDriverID()
+ setDriverName(String name)
+ setCarID(String ID)
+ getDriverID(): String
+ getDriverName(): String
+ getCarID(): String
+ getDriverInfo(): String
1
0...*
DeliveryServiceFactory
- deliveryServiceInstance: DeliveryService
+ getDeliveryService: DeliveryService
+ setupMockDeliveryService()
<<Interface>>
Deliverable
+ onPickup()
+ onDropoff()
<<Interface>>
DeliveryService
+ scheduleDelivery(Deliverable deliverable, Location pickupLocation, Location dropoffLocation, LocalDate pickupDate)
1
0...*
RentalNeeds
- num_bike: Integer
- type_bike: BikeType
- date_range: DateRange
- location_hire: Location
+ setNumOfBike(Integer numOfBike)
+ setTypeOfBike(BikeType typeOfBike)
+ setDateRange(DateRange dateRange)
+ setLocationOfHire(Location location)
+ getNumOfBike(): Integer
+ getTypeOfBike(): BikeType
+ getDateRange(): DateRange
+ getLocationOfHire(): Location
Location
- postcode: String
- address: String
+ setLocation(String postcodeA, String addressA)
+ isNearTo(Location other)
+ getPostcode(): String
+ getAddress(): String
+ getLocationInfo(): String
Confirmation
- bookingList: ArrayList<Booking>
- confirmationDetails: ArrayList<String>
+ setBookingList()
+ generateConfirmation(): String
1
1
1
0...*
Booking
- customer: Customer
- driver: Driver
- quote: Quotes
- payment: BigDecimal
- paymentSuccess: Boolean
- orderID: String
- pickMode: String
+ setBooking(Quotes quoteA)
+ setCustomer(Customer customerA)
+ setDriver(Driver driverA)
+ setOrderID()
+ setPickUpMode(String mode)
+ getOrderID(): String
+ getPickMode(): String
+ customerOfBooking(): Customer
+ driverOfBooking(): Driver
+ payForBooking(BigDecimal moneyForPay)
+ checkPaySuccess(): Boolean
+ getBookingInfo(): String
DateRange
- start: LocalDate
- end: LocalDate
+ DateRange(LocalDate start, LocalDate end)
+ getStart(): LocalDate
+ getEnd(): LocalDate
+ toYears(): long
+ toDays(): long
+ getStartYear(): int
+ getEndYear(): int
+ getStartMonth(): int
+ getEndMonth(): int
+ overlaps(DateRange other): Boolean
+ hashCode(): int
+ equals(Objects obj): boolean

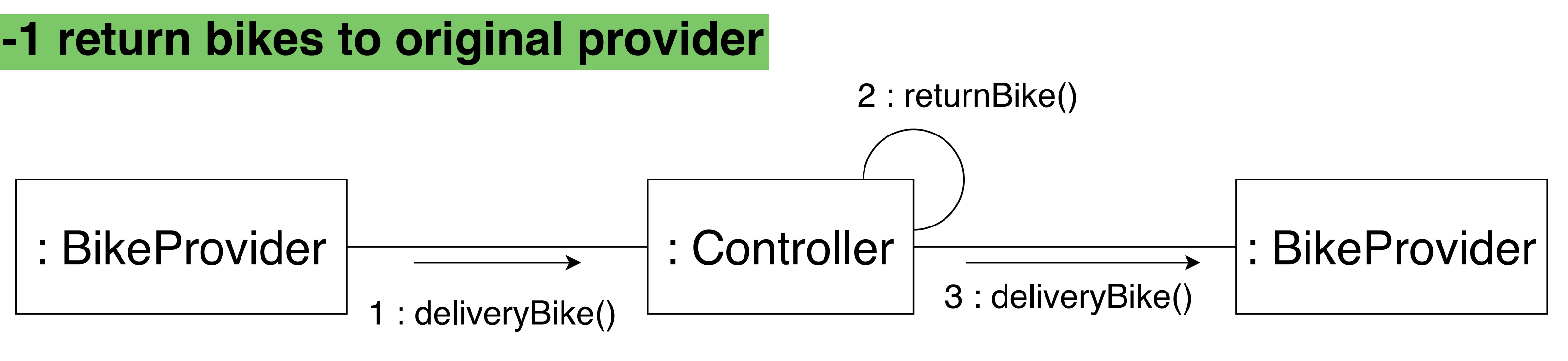
2.3-Dynamic models

2.3.1 - UML sequence diagram

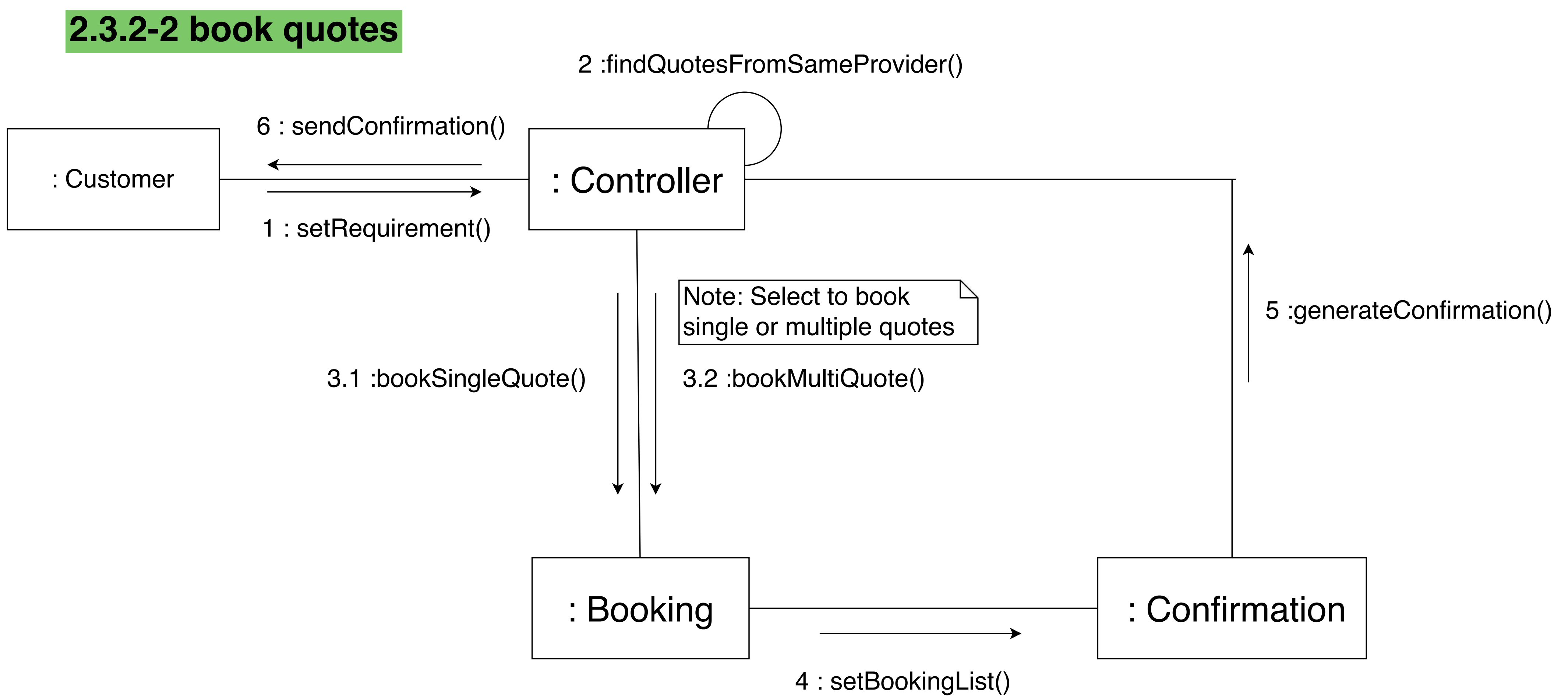


2.3.2 - UML communication diagram

2.3.2-1 return bikes to original provider



2.3.2-2 book quotes


2 :findQuotesFromSameProvider()
6 : sendConfirmation()
: Customer
: Controller
1 : setRequirement()
Note: Select to book single or multiple quotes
5 :generateConfirmation()
3.1 :bookSingleQuote()
3.2 :bookMultiQuote()
: Booking
: Confirmation
4 : setBookingList()