Informatics 2C – Introduction to Software Engineering Coursework 2

Team Member: Teh Min Suan (s1817967)
Michitatsu Sato (s1807428)

Introduction

This system is a platform for consumer to book bikes online. It includes features such as online payment, bike delivery, online chat and many more which will be introduced later on.

High-level description

- When a bike is being registered, it will be added to a list called bikeList.
- Location is a class that contains street address and postcode. There might be two different places with same street name but different postcode. This class is to prevent confusion in those cases.
- BikeProvider is an instance of bike provider actor which holds all the information of bike provider including all its partner, deposit rate and deposit policy.
- DoubleDecliningBalanceDepreciation and LinearDepreciation allow BikeProvider to set custom deposit policy.
- Customers get quotes by entering requirement which will then be converted to Condition class. The result is then returned to getQuotes() to search for bikes with matching condition. It returns a collection of bikes matching the attributes in Condition. The collection of bikes will then be wrapped as quotes within the method.

- Booking quotes can be done by calling bookQuote() method in Customer class. The method has 3 arguments which are a collection of quotes, a Boolean indicating method of collection and date. Date is automatically passed from condition (which previously entered by customer) to the method. So, customer doesn't have to reenter date of booking again. Then, system notifies both bike provider and customer and print the order summary to customer.
- ReturnBike() has to be called when customer returns bike to provider. First, it will check if the bike is returned to the correct bike provider or its partner. If bike is returned to partner, scheduleDelivery() will be called to record the delivery of bike back to the correct bike provider. Else, bike will be recorded as received by calling onPickup().
- If delivery is chosen as method of collection, scheduleDelivery() will be automatically called when booking is made.
- There is an additional class made which is OrderNumber. It is a generator which generates order number without duplication.

Reason as for why choices were made

- BikeList is created to keep track of all the bikes registered in our system. We considered implementing the system without bikeList, but it is hard to choose bikes matching condition entered by Customer without having a list.
- There are some class-specific methods in BikeProvider which are protected. This is to ensure that unauthorized user cannot access to the methods.
- Payment information of Customer is recorded within the Customer class. It cannot be accessed by unrelated classes so that this information will not be stolen. Additionally, payment information can be easily deleted when customer wishes to.
- We choose to create two separate class to hold information which are Condition and Invoice. This makes classifying information easier.
- Instead of using LocalDate to record date in our system, a new class called DateRange is created. As most of date datatype we used involves two LocalDate objects, it is more convenient to create a class that holds the information.

UML Class Model

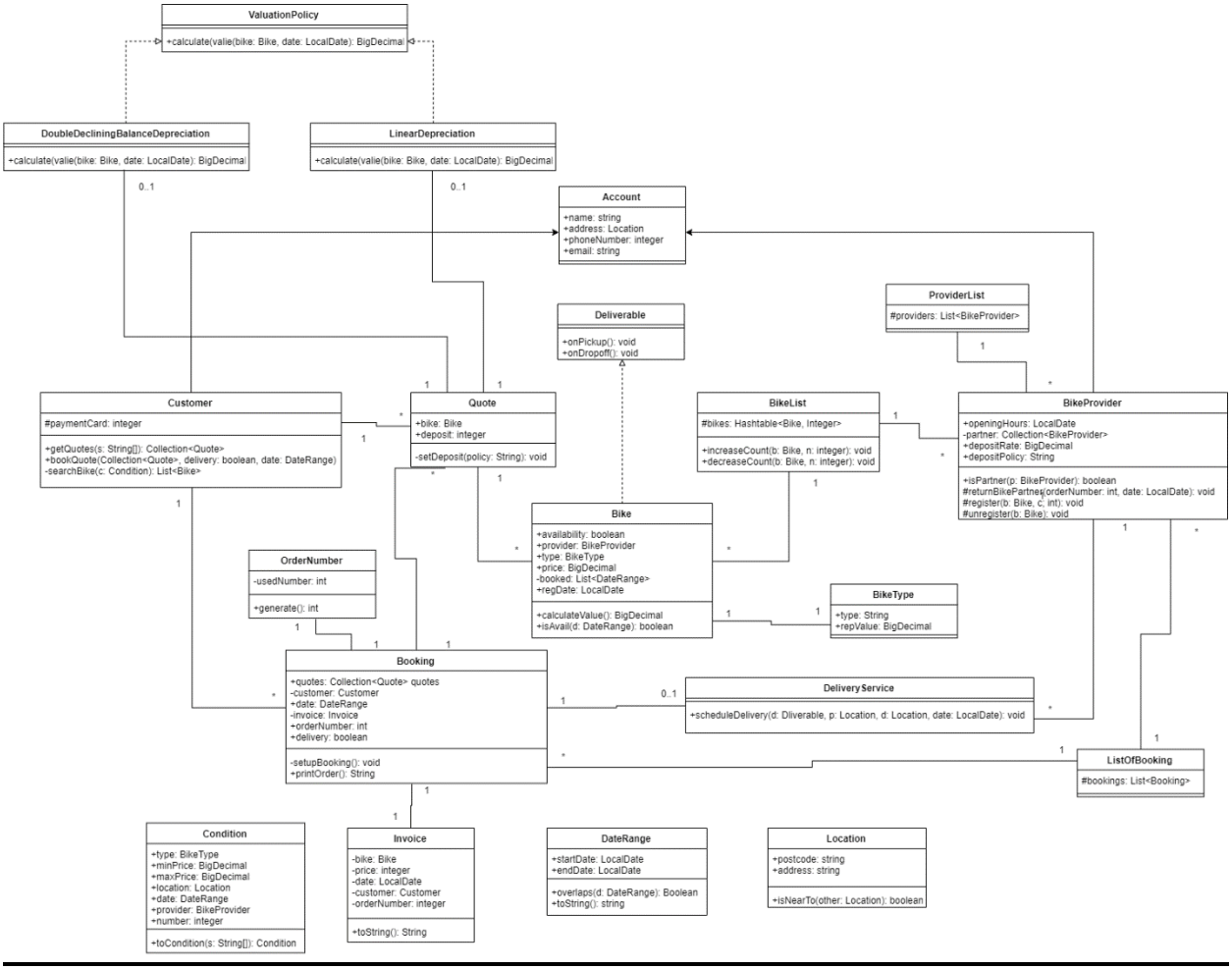

Figure 1: UML class diagram

UML Sequence Diagram

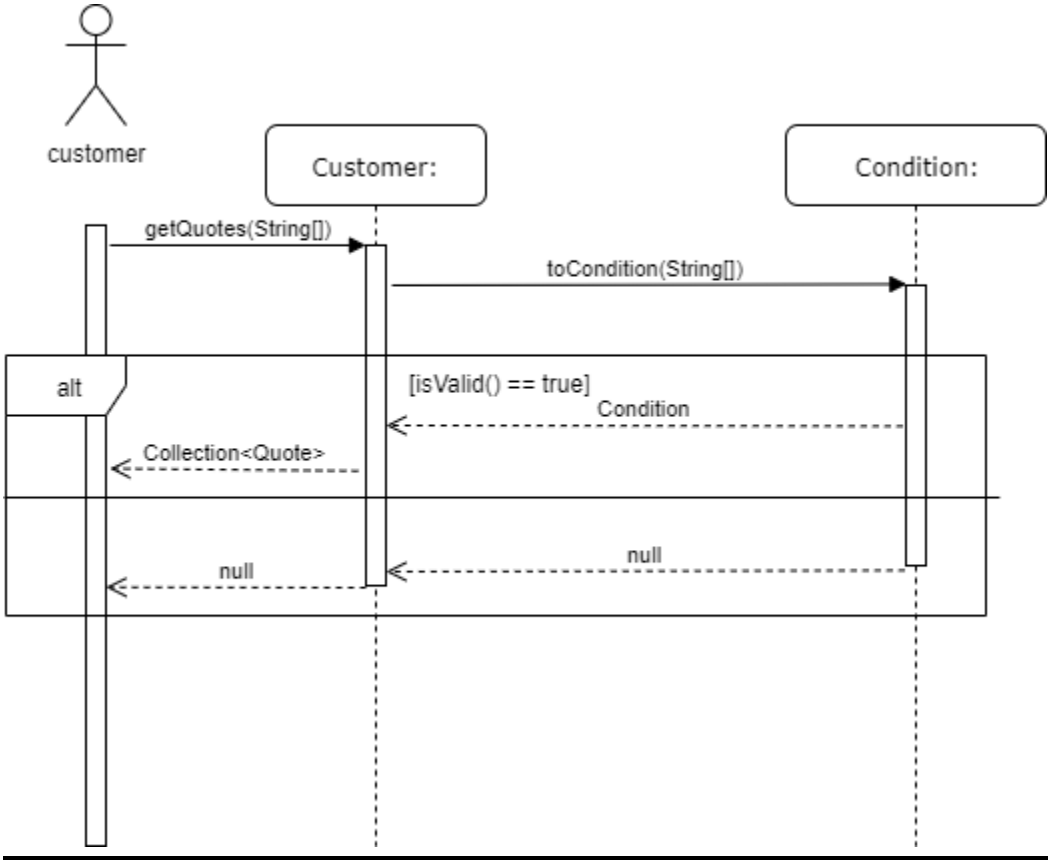

Figure 2: UML sequence diagram

UML Communication Diagram

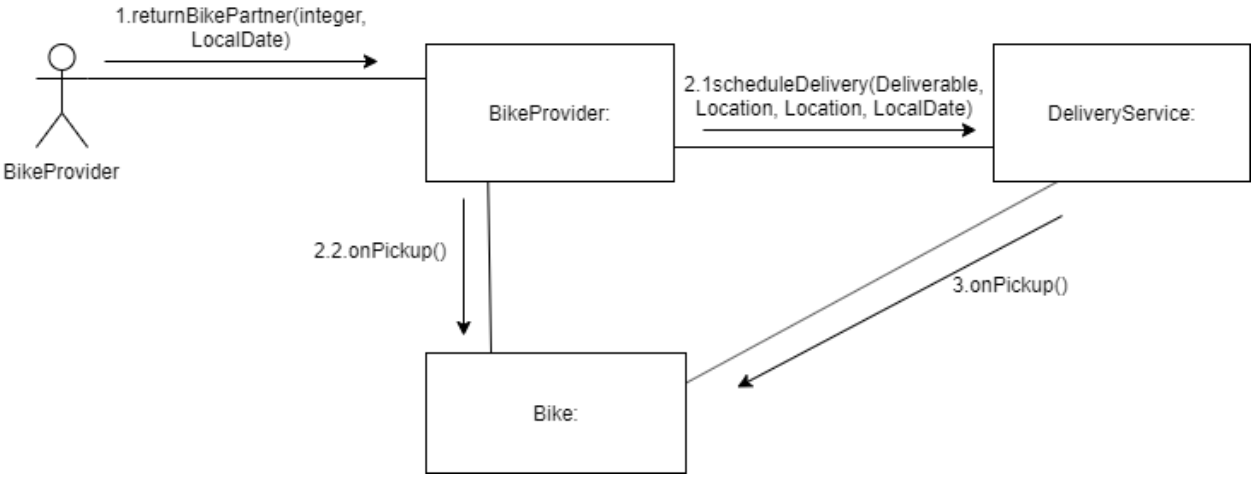

Figure 3: Return bikes to original provider

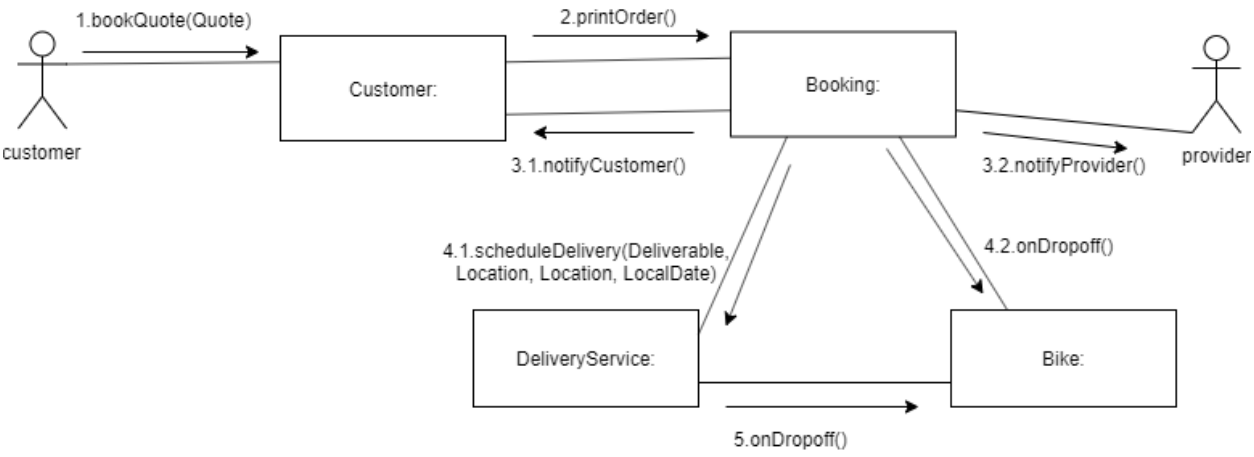

Figure 4: Book quote

Conformance to requirements

1. Originally, we planned to recognize each bike with a bikeId. But that doesn't have to be done as we can search for bikes using order number.
2. Deposit was originally being calculated only when making a booking but that doesn't give any flexibility to bike provider to change the rate. Thus, an interface to calculate deposit is created and the information will be held by Bike class.
3. Status of bike was not considered in our initial plan but getQuotes() has to filter out bikes that have been booked. Therefore, to be able to track the status of bike, all collection and return of bike must be recorded using onPickup() or onDropoff() method.
4. Initially, order summary was considered as a string that has all the information to be printed out. But it is hard to have a formatted string that holds all the booking information. Therefore, a new class, Invoice, is created to replace it.
5. Address was a string type but string can be easily modified. To ensure safety, a new class, Location, is created to save all the address information.

Self-assessment

Q 2.2.1 Static model 25%

- Make correct use of UML class diagram notation
  - Notation used should be all correct 5%
- Split the system design into appropriate classes
  - Several classes are created with important functionality 5%
- Include necessary attributes and methods for use cases
  - Each use cases should have suitable methods and attributes to be functional 5%
- Represent associations between classes
  - It should be done correctly 5%
- Follow good software engineering practices
  - Our design should be quite simple with multiple modules each having their own functionality 5%

Q 2.2.2 High-level description 15%

- Describe/clarify key components of design
  - Every part of our design is explained 10%
- Discuss design choices/resolution of ambiguities
  - We explained our choices of design as well as making sure ambiguities are solved 5%

Q 2.3.1 UML sequence diagram 20%

- Correctly use of UML sequence diagram notation

    - Notation used should be correct 5%
- Cover class interactions involved in use case
    - They should all be mentioned 10%
- Represent optional, alternative and iterative behavior where appropriate
    - We represented an alternative to getQuotes() when having no match in bike search 5%

Q 2.3.2 UML communication diagram 15%

- Communication diagram for return bikes to original use case
    - We should have correctly represented it in our UML communication diagram 8%
- Communication diagram for book quote use case
    - Same as above 7%

Q 2.4 Conformance to requirements 5%

- Ensure conformance to requirements and discuss issues
    - We brought up some issues we encountered and mentioned how we solved them 5%

Q 2.5.3 Design extensions 10%

- Specify interfaces for pricing policies and deposit/valuation polices
    - We did create 2 interfaces which are implemented by BikeProvider 3%
- Integrate interfaces into class diagram

- It is shown in our class diagram 7%

Q 2.6 Self-assessment

- Attempt a reflective self-assessment linked to the assessment criteria
  - We reviewed all of our works according to the assessment criteria at least once 5%
- Justification of good software engineering practice
  - Simple design. Only create class if it is necessary to do so. Every classes are created for its own functionality or, in another words, no overlapping functionality between classes. Design choices are explained clearly to ensure readers are able to understand the reason behind it.