INF2C SOFTWARE ENGINEERING

2019-20

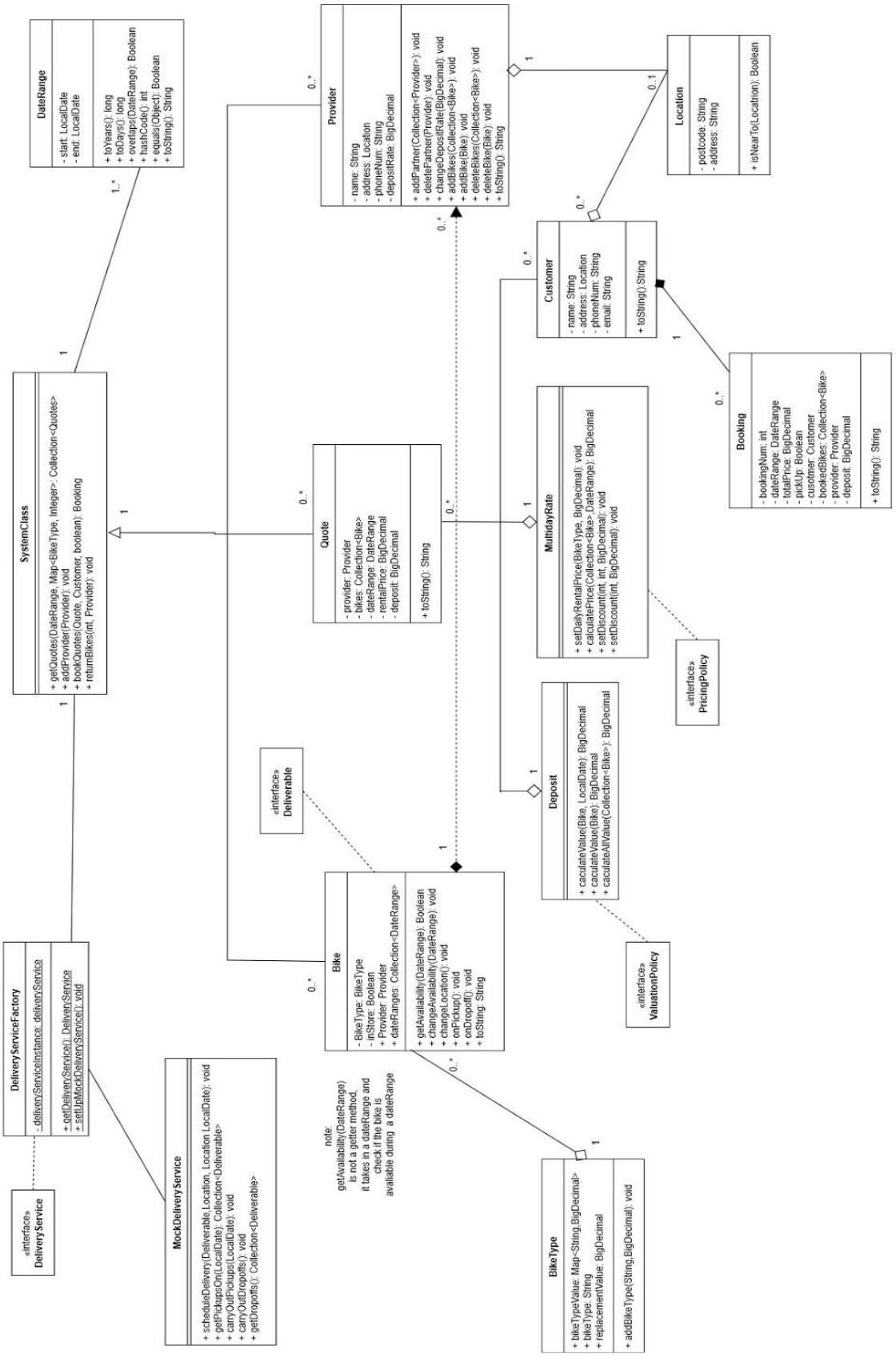
Coursework 2

Capturing requirements for a federalised bike rental system

Wini Lau	Kenza Amira
s1846175	s1813674

The coursework is entirely updated.

2.2.1 Static model



2.2.2 High-level description

In Our Class Diagram, we decided to add a System Class that is the center of the system. We made it so that there are less connections between classes to allow for easier future changes. It is designed as a super Class for 3 other classes: Quotes, Bike and Provider.

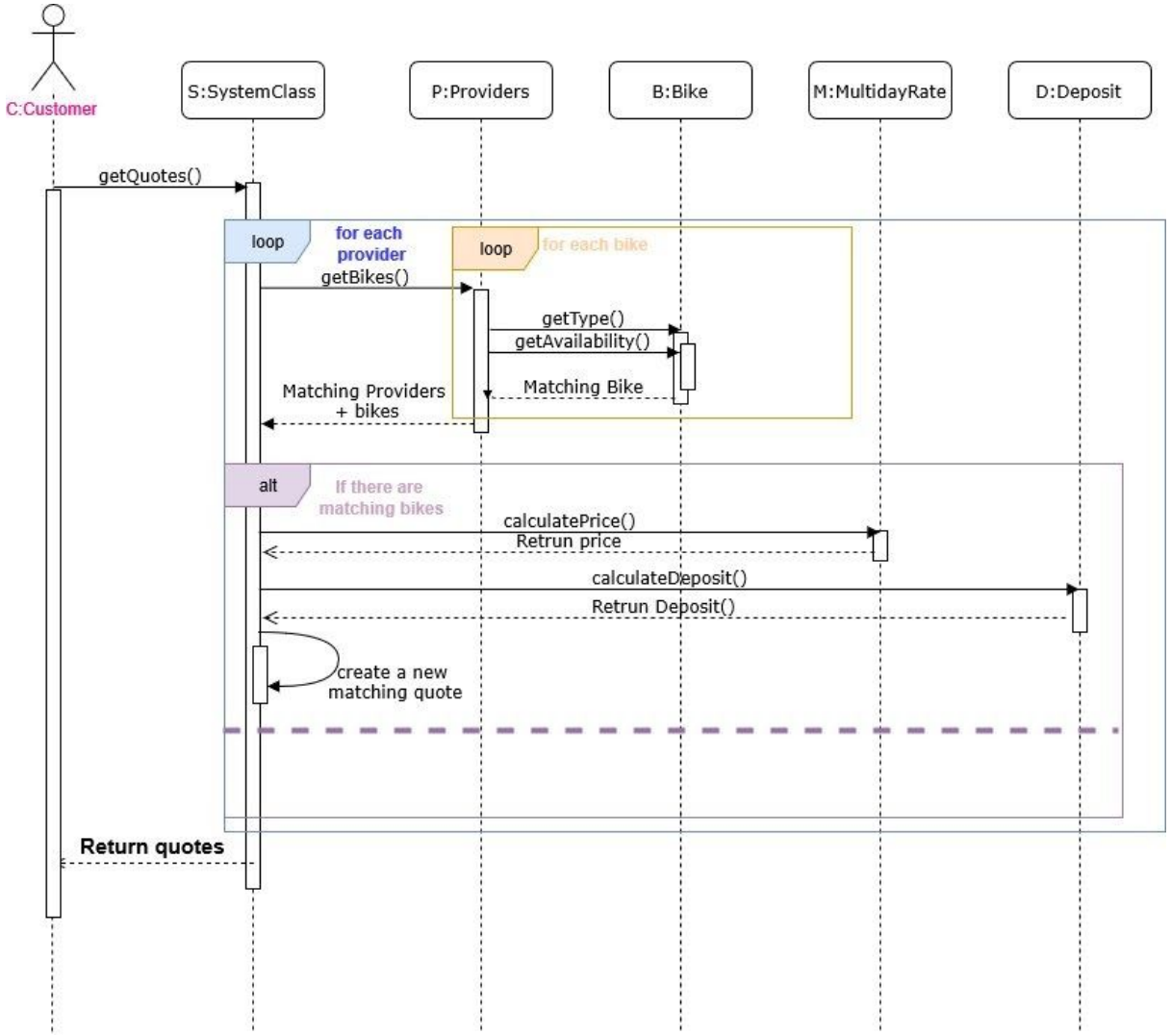
The system class contains a list of providers as it is better to store them like that. It also takes care of the getQuote(), bookQuote() and returnBikes() function (Our main use cases for this coursework).

Let's look into the details:

- The Bike class implements the Deliverable interface. That way it makes the bikes objects that can be delivered. It contains method that allow to access type, provider, price, availability, …
- The Quote class allows to create Quote objects so that we're able to generate quotes and book them.
- The Provider class contains all the information about the provider (name, address, partners, ..)
- The Deposit class implements the ValuationPolicy interface. It allows to calculate the deposit according to the bike replacementValue and the Provider's deposit rate.
- The Multi Day Rate class implements the PricingPolicy interface. It allows providers to set discounts (or not). When this class is used, the price is calculated taking discounts into consideration.
- The Customer class contains information about the Customer
- The Booking class allows to create a Booking when the Customer picks up the bike.
- The Location class is used to make creating addresses easier: a location contains an address and postcode. It is used by the Provider and the Customer classes (shop address and home address).
- BikeType is a class that allows for the creation of new types and its replacement values.
- Date Range is used to be able to handle Local Dates (like today's date) in Bookings, Quotes, Delivery and availability.
- The Delivery related classes are given classes that allow for Delivery scheduling.

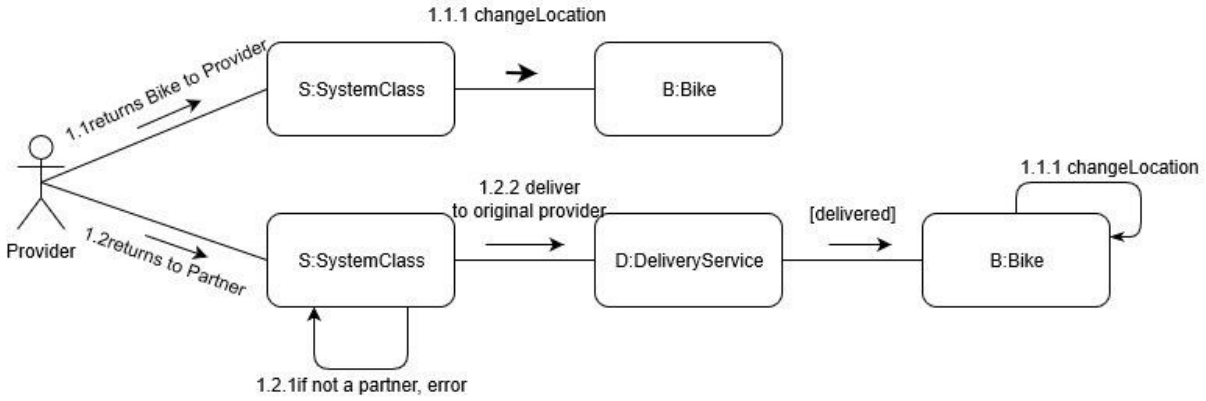
UML notation on the arrows explain the relationships between these classes. We tried to keep it to a minimum but it is hard to keep it very separate as each Class will logically need things from the other classes. However, the creation of the system class really makes it less intertwined. Even if a lot of classes are connected to others, changes are not too hard to make as they usually require just a little bit of work.

2.3.1 UML sequence diagram

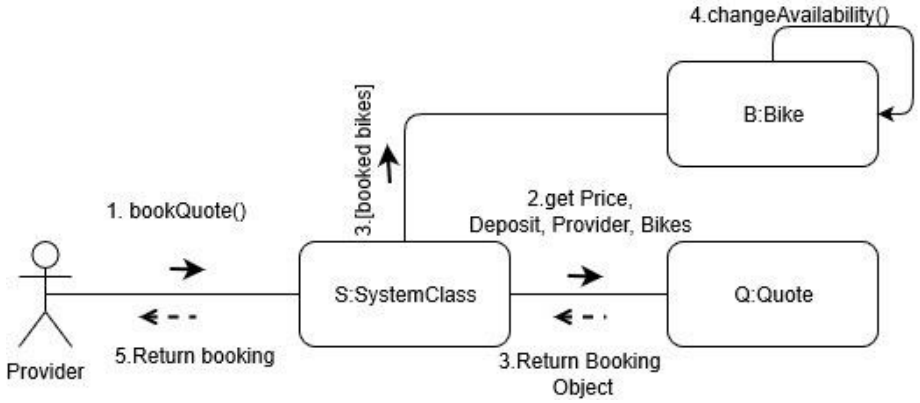


2.3.2 UML communication diagram

Communication diagram for record bike return to original provider use case:



Communication diagram for book quote use case:



2.4 Conformance to requirements

As required by coursework 3, we made quite a lot of changes to our second Coursework. We used the interfaces we needed to design in coursework 2 by implementing them in our 2 classes : Deposit and Multi day Rate. Also the Sequence Diagram changes a lot because it is clearer what needs to be done by the system, but also because we are not checking 3 days before and 3 days after in case there are no ùatching quotes for the date range given by the customer in his get a quote request. Other than that, it tries to follow all the other requirements listed in coursework 1. A provider can add a partner,set discounts, set deposits…

However, providers are now able to add a bike Type and set a replacement value for it. Hence we don't use set types anymore (Enum) but strings. We also now use BigDecimal instead of floats or doubles.

2.5.3 Design extensions* *(updated with the actual extensions)

```
public interface PricingPolicy {
        public void setDailyRentalPrice(BikeType bikeType, BigDecimal dailyPrice);
        public BigDecimal calculatePrice(Collection<Bike> bikes, DateRange duration);
}
public interface ValuationPolicy {
        public BigDecimal calculateValue(Bike bike, LocalDate date);
```

}

2.6 Self-assessment

Task	Predicted Marks	Met?
Q 2.2.1 Static model 25% • Make correct use of UML class diagram notation 5% • Split the system design into appropriate classes 5% • Include necessary attributes and methods for use cases 5% • Represent associations between classes 5% • Follow good software engineering practices 5%	21% 5% 5% 4% 3% 4%	Coursework 3 gave us much more information and allowed us to better see what was expected of the system. Hence, it makes the Class diagram clearer. However, there might still be some mistakes in cardinality.
Q 2.2.2 High-level description 15% • Describe/clarify key components of design 10% • Discuss design choices/resolution of ambiguities 5%	12% 9% 3%	We did describe the components and design of our use class however we might have missed some things that seemed clear to us but that are not necessarily straightforward.
Q 2.3.1 UML sequence diagram 20% • Correctly use of UML sequence diagram notation 5% • Cover class interactions involved in use case 10% • Represent optional, alternative, and iterative behaviour where appropriate 5%	18% 5% 9% 4%	We tried representing the use case with its extensions as much as we could, using loop and alt. It has also been made way easier as we now have less things to test for and have a better idea of the functions that need to be used
Q 2.3.2 UML communication diagram 15% • Communication diagram for record bike return to original provider use case 8%	12% 6%	recreated a simple communication diagram based on the use cases and the class diagram

• Communication diagram for book quote use case 7%	6%	
Q 2.4 Conformance to requirements 5% • Ensure conformance to requirements and discuss issues 5%	4% 4%	Tried to describe as much as possible. Our system should conform to the requirements introduced in cw3.
Q 2.5.3 Design extensions 10% •Specify interfaces for pricing policies and deposit/valuation policies 3% • Integrate interfaces into class diagram 7%	8% 3% 5%	The interface used here are just the interfaces that we had to implement. This is why we won't be changing the grade here. As the interfaces have been given.
Q 2.6 Self-assessment 10% • Attempt a reflective self-assessment linked to the assessment criteria 5% • **Justification of good software engineering practice 5%**	9% 5% 4%	**Use of meaningful words + No cryptic comments + Class Diagram made to make changes easier (System class and as little connections possible).**
Total	84%	