Apurv Mishra s1864480

Lorenzo Baldini s1853050

Inf2C Software Engineering 2019-20

Coursework 2:

Creating a software design for a federated bike rental system

2.1. Introduction

This document provides a software design for a federated bike rental system. The application allows the user to browse rent quotes, book a quote and return the bikes. Refer to the previous document for full specifications.

Implementation

We very successfully adhered to the other courseworks requirements. The main editing consisted in adding required methods that we didn't think of in the class diagram, such as some private methods.

We've implemented both submodules, and also use the default policy when the shop doesn't set a custom one. Testing has been added for all submodules and util classes.

The system is tested with a SystemTest jUnit class, that covers most edge cases, including but not limited to:

- Shop is in range when getting quotes;
- Same quote can't be booked twice;
- Only logged in users can return bikes;
- Bike return is handled correctly for return to both partner and original shop
- DeliveryService is called and works as expected.

Assertions are present in the code to check for correct internal logic in method calls, and if statements are used to parse user input in the controller, throwing Errors where appropriate.

2.2. Self-assessment

Extension submodules 10**/10**

Implementation of extension submodule 10/**10**

Should implement extension submodule

Should include unit tests for extension submodule
Extension submodule is implemented and unit testing included.

Peer review of other group's submodule 10**/10**
An extensive peer review is provided, covering positive and negatives, relative to this mark scheme.

Tests 35/**35**

System tests covering key use cases 20**/20**

Should have comments documenting how tests check the use cases are correctly implemented

Should cover all key use cases and check they are carrying out the necessary steps

Should have some variety of test data

Should use MockDeliveryService
System tests are provided and commented, all passing; each use case has its own test and varied test data is used, including well-known edge cases such as dates in the past, case-insensitive postcodes and empty shops.

Unit tests for Location and DateRange 5/**5**
Unit tests are provided for Location and DateRange

Systems test including implemented extension to pricing/valuation 5/**5**

Mock and test pricing/valuation behaviour given other extension (challenging) 5/**5**
All pricing and deposit submodules are tested and working, including a full implementation of the other submodule.

Code 45/**45%**

Integration with pricing and valuation policies 10**/10**

System should correctly interface with pricing and valuation policies

System should correctly implement default pricing/valuation behaviour
System correctly interfaces with custom and default submodules

Functionality and correctness 25**/25**

Code should attempt to implement the full functionality of each use case

Implementation should be correct, as evidenced by system tests
All use cases are correctly implemented as evidenced by the passing tests.

Quality of design and implementation 5**/5**

Your implementation should follow a good design and be of high quality

Should include some assertions where appropriate
All good practices from the Google Style Guide are followed

Readability 5**/5**

Code should be readable and follow coding standards

Should supply javadoc comments for Location and DateRange classes
Code is readable, commented where appropriate and javadocs are supplied when required plus for some extra methods.

Report 10**/10**

Revisions to design 5**/5**

Design document class diagram matches implemented system

Discuss revisions made to design during implementation stage
All previous coursework math this implementation

Self–assessment 5**/5**

Attempt a reflective self-assessment linked to the assessment criteria
A self-assessment is provided, commenting on the criterias in the mark scheme