Creating an Abstract Implementation of a Federalised Bike Rental System

Coursework 3

Informatics 2C - Introduction to Software Engineering

Justin Howe
s1840358

Rohan Nittur
s1803949

Contents

1. Changes

1.0.1 Requirements

There were no additional changes made to the system requirements documentation after the second submission of the document.

1.0.2 Design

Reflecting on the marker's comments, we decided to make significant multiple changes to our system design. The UML class diagram now correctly implements all of the classes required for the system description. The design embodies the idea of encapsulation. For example: We have a class called RequestedData which is sent from the Customer to the MainSystem. Our System class filters the quotes required for the `findQuotes()` function.

The sequence diagram now contains a label and a lifeline for the actor, and the erroneous `not update()` label has been removed. The initial loop has also been removed due to its lack of purpose. The messages have been ordered such that the condition is known before going into the alternative frame and the classes have been updated to reflect the new classes and method names and functions.

The UML Communication diagrams were entirely re-done, after re-examination of the methodology of these diagrams and the use cases they apply to.

2. Self Assessment

2.1 Assessment

§1 Extension submodules **10%**

- Implementation of extension submodule 10%
  - Should implement extension submodule
  - Should include unit tests for extension submodule

§2 Tests **25%**

- System tests covering key use cases 15%
  - 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
- Unit tests for Location and DateRange 5%
- Systems test including implemented extension to pricing/valuation 5%
- Mock and test pricing/valuation behaviour given other extension 0%

§3 Code **40%**

- Integration with pricing and valuation policies 8%
  - System should correctly interface with pricing and valuation policies
  - System should correctly implement default pricing/valuation behaviour
- Functionality and correctness 22%

  - Code should attempt to implement the full functionality of each use case
  - Implementation should be correct,as evidenced by system tests
- Quality of design and implementation 5%
  - Your implementation should follow a good design and be of high quality
  - Should include some assertions where appropriate
- Readability 5%
  - Code should be readable and follow coding standards
  - Should supply javadoc comments for Location and DateRange classes

§4 Report **10%**

- Revisions to design 5%
  - Design document class diagram matches implemented system
  - Discuss revisions made to design during implementation stage
- Self-assessment 5%
  - Attempt a reflective self-assessment linked to the assessment criteria

2.2 Justification

Reflecting on the journey that our team has made from creating the System Requirements document to coding an abstract model of a federal bike rental system, our team has certainly learned a lot throughout the whole journey. However, we knew time was certainly a issue for the team, while it took us a while to attempt to wholly understand the coursework description and refer back to the system requirements document and recreate our system design.

While we strongly felt that our extension submodule was fully implemented and integrated in the system, we felt we had very little time to conduct tests for the mock and test pricing/valuation behaviour given the other extension. Our code had plenty of useful comments and Javadoc documentation should be of use for the marker to understand.