# Coursework 3

## Creating a software design for a federated bike rental system

Group 46

Nathan Sharp s1869292

Yasmin Yang s1808941

1. Extension Modules (attached) and Peer Review (submitted in lab)
    LinearDepreciation.java
    DoubleDecBalanceDepreciation.java
    ValuationPolicy.java
2. Construct Code (attached)
    EntryPoint.java
    Bike.java
    BikeType.java
    BikeCategory.java (enum)
    BikeProvider.java
    Customer.java
    Order.java
    SystemDatabase.java
    DateRange.java
    Location.java
    Deliverable.java
    DeliveryService.java
    DeliveryServiceFactory.java
    MockDeliveryService.java
3. Create system-level tests (attached)
    SystemTests.java
4. Unit testing (attached)
    TestLocation.java
    TestDateRange.java
    ValuationPolicyTests.java
5. Integration and mocking of pricing and valuation
    The default pricing and valuation policies are implemented and integrated in system tests and so were tested.
6. Updates to design and requirements documents
    Summary of changes made and reason for them:
    OrderRequest : Removed - functionality replaced with EntryPoint class as requestQuotes method.
    BikeType : Split into;
        1) enum BikeCategory - to relate to immutable classes of bike eg. ROAD, MOUNTAIN etc.
        2) Class BikeType - to represent a class of bike in the bike providers inventory
    SystemDatabase : Added - to control global storage and access of information e.g list of BikeProviders

Additionally numerous changes were made to the specific functioning of every class as we found we often had different intuitions about how to implement a system when faced with the task in practice, also we found problems, loopholes and missing cases from our

plans which we had to overcome. Our new system specification is shown in the UML class diagram of of Coursework 2.

7. Self-assessment **{comments in bold}**

1. Extension submodules 10% **{10, we implemented a correctly working submodel}**

- Implementation of extension submodule 10%

Should implement extension submodule

Should include unit tests for extension submodule

- Peer review of other group's submodule (up to 10% bonus marks) **{10bonus, peer work was high quality and a good learning experience to spend time with another group and experience how they operate}**

2. Tests 35% **{30, did not get round to Mock testing pricing and valuation behavior}**

- System tests covering key use cases 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

• 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 (challenging) 5%

3. Code 45% **{42, well implemented at clean to read with some assertions, always room for improvement though}**

• Integration with pricing and valuation policies 10%

System should correctly interface with pricing and valuation policies

System should correctly implement default pricing/valuation behaviour

• Functionality and correctness 25%

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% **{10, analysis complete and UML class diagram reconfigured}**

• 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

**Overall we felt like we implemented a robust system with some clever design tricks such as the treemap search. Doing it again we would probably spend more time on our original UML class diagram as it became impossible to 'test first' one we deviated from our official plan. As a result many of the test were imcomplete in time so could not be submitted**