Coursework 3 – creating an abstract implementation of a federalised bike rental system

Maksymilian Mozolewski – s1751752

Lars Werne - s1824086

Design Changes induced by CW3

Our design went through a couple changes because of realisations we've had as we were trying to implement it. The biggest one, was the permament merging of the extension interfaces into the system, and the introduction of standard policies to capture default behaviour. Also due to ambiguity in the original design, our original system allowed each bike provider to set their own prices for each bike type. We've realised that the system was supposed to assume universal pricing amongst the different full replacement values. This was not clear to us at all, and so we did not even consider this as an ambiguity before(since it was obvious to us that the bike type prices would not be universal). The appearance of the deliverable interface also forced us to reconsider the way we have to update our bookings. Previously that was the job of the delivery system which would send messages to the bike rental system; now this had to be done by individual deliverable implementing objects. We dealt with that by storing references to relevant bookings in each bike making the bikes implement deliverable, and then adding a finite state machine to the booking class itself. We also removed the possibility of each bike provider sending multiple quotes, and decided to just use the cheapest quote-based on the individual and not group price of the bikes since there are a lot of possibiliities to check.

Anything else ?

Self-Assessment

1. Extension submodules 9/10
   a. Implementation of extension submodule 10/10:
      i. Should implement extension submodule:
         we have indeed implemented the correct extension submodule.
      ii. Should include unit tests for extension submodule:
         we included appropriate unit tests for the submodule. We believe the tests test passing should ensure the correct functionality of the submodule.
   b. Peer review of other group's submodule 9/10:
      We have carried out the peer review with another group, and we believe that our feedback was informative and of high quality.
2. Tests 35/35
   c. System tests covering key use cases 20/20
      i. Should have comments documenting how tests check the use cases are correctly implemented:
         We have used comments throughout our tests, which describe in good enough detail how the tests test the correct behaviour of the system
      ii. Should cover all key use cases and check they are carrying out the necessary steps :
         We have covered all of the provided use cases, and more. As well as covering all use cases We added a test which combines all of the use cases provided.
      iii. Should have some variety of test data:
         We have indeed provided key variations in the tests which check different flows of the code
      iv. Should use MockDeliveryService :
         Our tests do indeed use the mockdeliveryservice. And we believe this is done correctly.

d. Unit tests for Location and DateRange 5/5:
   We have provided suitable unit tests for Location and DateRange classes
e. Systems test including implemented extension to pricing/valuation 5/5:
   Our system tests include the implemented extension as a variation to tests.
f. Mock and test pricing/valuation behaviour given other extension (challenging) 5/5:
   We have suitably tested the other extension with a mock class, by seeing if it works with other classes.

3. Code 42/45
   g. Integration with pricing and valuation policies 10/10
      i. System should correctly interface with pricing and valuation policies:
         The system correctly interfaces with the pricing and valuation policies, as evident by the system tests we have provided, as well as the unit tests to some classes.
      ii. System should correctly implement default pricing/valuation behaviour:
         Our system provides standard implementations of pricing and valuation policies which represent the default pricing/valuation behaviour. We believe the tests we provided attest to its correctness
   h. Functionality and correctness 25/25:
      i. Code should attempt to implement the full functionality of each use case :
         We have attempted and succeeded in implementing the full functionality of each use case, if not more.
      ii. Implementation should be correct, as evidenced by system tests:
         The implementation is evidently correct as shown by our tests which we believe are correct.
   i. Quality of design and implementation 3/5:
      i. Your implementation should follow a good design and be of high quality:
         We have followed good software engineering practice as well as we could. We belive our attempt was pretty successful, as evident from the design of our system.
      ii. Should include some assertions where appropriate
         We have included a lot of assertions in our code.
   j. Readability 4/5
      i. Code should be readable and follow coding standards:
         We included comments, with explanations, and used good code formating standards.
      ii. Should supply javadoc comments for Location and DateRange classes :
         We have supplied javadoc comments for the location and dateRange classes.

4. Report 10/10
   k. Revisions to design 5/5
      i. Design document class diagram matches implemented system :
         We have adjusted the diagrams to match our implemented system.
      ii. Discuss revisions made to design during implementation stage:
         We have discussed any changes needed/made to the system at this stage.
   l. Self–assessment 5/5
      i. Attexmpt a reflective self-assessment linked to the assessment criteria:
         We believe our self-assesment is pretty accurate and objective, and follows the assesment criteria.