SWEA Iteration 5: Test Doubles and More Patterns

<<Group name>>
Computer Science, University of Aarhus
8200 Århus N, Denmark
<<Names>>

<<Date>>

1 EpsilonStone

1.1 Design and UML

[Short argumentation for your EpsilonStone design: Refer to the UML diagram, and emphasize how indirect input are encapsulated; which delegates are the Test Doubles and the real production implementation.]

[INCLUDE UML DIAGRAM HERE - A GOOD PICTURE OF HANDRAWN IS OK.]

1.2 JUnit test cases

[Write the full path of the JUnit test file, that configure Game with your test doubles(s), and tests the EpsilonStone behaviour.]

[Include screenshot/contents of the JUnit tests.]

2 ZetaStone

2.1 Design and UML

[Write a short argumentation for which roles (game/winner strategy/??) stores which state (hero health/attack output) and how this state is passed/access so your ZetaStone winner strategy can correctly determine the winner; and the collaborating roles have high cohesion.]

[Include a UML diagram that shows the design of the ZetaStone part of HotStone, with emphasis on the State pattern introduced and the existing winner strategies that are reused. (A good quality picture of a hand-drawn UML diagram is OK.)]

2.2 Unit/Integration Testing

[Include short argumentation for your decision to either TDD using unit testing the strategy using a Game test double; or by integration testing with your standard game implementation.]

2.3 State Selection Code

[Write the full path of the implementation file of your State pattern for ZetaStone.]

[Write a short argumentation for how state selection is made, refering to the code below.]

[A screenshot of the (JaCoCo-painted[1]) state selection code in the Context implementation of the State pattern, showing that correct behaviour of ZetaStone has been tested.]

3 EtaStone

3.1 Design and UML

[Short argumentation for your EtaStone design with reference to the UML diagram.]

[INCLUDE UML DIAGRAM HERE - A GOOD PICTURE OF HANDRAWN IS OK.]

4 Backlog

The following features and requirements are still not implemented in our HotStone software:

- ...
- ...

[1]It would be fine if you present JaCoCo or IntelliJ Code Coverage painted code to show your tests cover both states.