PV260 Software Quality

Assignment 3 - Static Code Analysis, Unit Testing

Spring 2015

1 General Information

1.1 Dates

- Assignment start: 7.4.2015
- Task 1 and 2 deadline: 21.4.2015 23:59
- Task 3 deadline: 28.4.2015 23:59

1.2 Submission

Please submit your solution as three .jar files containing source files to the Homework vault (Odevzdávarna) called *Assignment 3: Testing, SCA (Groups 01, 02).* Each jar should contain solution for one of the tasks described below. The name of the archives should be as follows: *lastname1-lastname2-assignment3-taskN.jar.* The tasks should be solved in the groups of two (exceptionally in groups of three). Include names of the group members in `@author` tag in every class. Only one solution per group should be submitted.

1.3 Evaluation

This assignment is split into three parts evaluated separately as follows:
Task 1 - max 8 points, Task 2 - max 8 points, Task 3 - max 9 points.
The points will be distributed based on both fulfilling the functional requirements and compliance of the code with SOLID and Clean Code principles.

2 Tasks

2.1 Task 1

- Using Checkstyle, write a Check module which will look for the *Brain Method* identity disharmony as described in `https://is.muni.cz/auth/el/1433/jaro2015/PV260/um/sem/54527454/55007209/identity-disharmonies.pdf?studium=675598` page 92
- Your module should be configurable through the standard Checkstyle xml analysis configuration file on all parameters mentioned in the Disharmony description:
  - How many LOC the method must have to be considered excessively large
  - How high cyclomatic complexity is allowed before the method can be considered a Brain Method
  - How deep nesting of control logic is allowed in the method before it can be considered a Brain Method
  - How many variables must be used inside the method before it can be considered a Brain Method

- It is not required that you write tests for the module. However, to prove your solution works, write a few example dummy methods on which your module can be run and gives correct results.
- In the .jar file with your module also include the test file mentioned above and anything else necessary for the module to be usable in Checkstyle.

2.2 Task 2

- Solve the following challenge: `http://www.reddit.com/r/dailyprogrammer/comments/3104wu/20150401_challenge_208_intermediate_ascii/`
- In addition to fulfilling the functional requirements of the challenge, provide unit test suite for your solution with at least 90% branch coverage.
- We do not enforce the TDD startegy of programming, so it is up to you whether you write your tests first or last. Nevertheless, the test-first strategy is highly recommended.
- You are free to use whichever tool you prefer for test coverage calculation while working on the solution. For evaluation JaCoCo will be used. If you decide to also use this tool, ant script to run the analysis can be found here: `https://is.muni.cz/auth/el/1433/jaro2015/PV260/um/sem/54527454/55345625/jacoco.xml?studium=675598`

2.3 Task 3

- Using the project ProductFilter: `https://is.muni.cz/auth/el/1433/jaro2015/PV260/um/sem/54527454/55345625/ProductFilter.zip?studium=675598`also available through IS/Project assignments. Write unit tests for classes AtLeastNOfFilter and Controller which will test the following behavior:
- AtLeastNOfFilter:
  - The constructor throws the exceptions as documented
  - The filter passes if at least exactly n child filters pass
  - The filter fails if at most n-1 child filters pass
  - Bonus: Try to only use the AtLeastNOfFilter class itself in the tests, do not depend on other project classes
- Controller:
  - The controller sends exactly the products selected by the provided filter to Output
  - The controller logs the message in documented format on success
  - If exception occurs when obtaining the Product data, Controller logs this exception
  - If exception occurs when obtaining the Product data, nothing is passed to the Output
  - Bonus: In the Logger tests don't depend on what the actual returned items are, e.g. check the format against a regular expression
- Use Mockito at least once to create one of the Test Doubles needed.
- Create at least one Test Double manually (without the use of any mocking framework).
- Do not modify the sources in any way. Only add your unit tests to the test folder.