HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

HBV205M: Prófun hugbúnaðar / Software Testing · Spring 2021
Dr. Helmut Neukirchen

Assignments 11/12 · Due 22.3.2021, 10:00 – Submit sources of test classes

Assignment 11

Objectives: Apply code coverage and mutation testing tools and improve test quality.
Check the quality of your solution of Assignment 10 (iff you have not solved Assignment 10, you can download a sample solution from Canvas) by using both a coverage tool and a mutation testing tool. Improve your tests until the tools tell you that your tests cover all lines of the implementation and kill all mutants. (You can ignore complaints concerning lines of the test cases themselves.)
Notes:

1. You need to install the PIT plugin for your IDE via `https://pitest.org/links/`.
2. For sure, the tools will complain about lines that you were never supposed to test in Assignment 10, therefore:
   - Remove `MoneyStackMain.java`,
   - Add from `hbv205m_08junit_example_junit4.zip` (in Canvas) tests for class `Money`, e.g., add `MoneyTest.java` (there, you need to remove lines 1 and 3 dealing with packages `junit4Demo.money` and `junit4Demo.moneyTest`).
   - Add these tests for class `Money` to your test suite.

As solution, only upload the final result of your source code containing the JUnit tests (and nothing else, e.g., do not upload the implementation under test, byte code, or whole Eclipse projects) – typically, you have to upload four `.java` files:

1. test suite class `AllTests`,
2. test cases for `Money`,
3. test cases for `MoneyStack.sum()`,
4. the other test cases for `MoneyStack`.

Assignment 12

Objectives: Apply Cucumber by extending a Gherkin feature file and adding Java step definitions using JUnit 4 assertions, and apply test-driven/behaviour-driven development.

As implementation under test, use the `IntStack` from the Cucumber lecture slides example:

1. Download `hbv205m_08cucumber6_example.zip` from Canvas.

   If you use Eclipse, you can import the zip file via: Import → General → Existing Projects into Workspace → Next → Select Archive File → Browse... → Finish. (This gives you a project that has the JUnit 4 library and the complete Cucumber 6 library with all its dependencies.)

   (If you rather use IntelliJ, you should be able to import the Eclipse project, but you probably need to add the jar files from directory `lib` manually.)

   The zip archive contains class `IntStack` and three files for testing: a feature file with one scenario using Gherkin syntax, Cucumber step definitions, and a JUnit 4 Java file for running the tests scripted in the feature file.

   As a sanity check, try to run class `RunCucumberTest` as JUnit test which should execute Cucumber with the stack passing the test case described by the provided scenario.

2. Add a scenario to the feature file that tests that a new stack is not full and add the needed step definitions to class `IntStackStepdefs`. *Hints:*
   (a) Running Cucumber on new steps in a feature file will give you the code snippets that you need to add to the Java step definitions.
   (b) Keep your step definitions as generic and re-usable as possible, e.g., re-use the existing field `actualBooleanAnswer` so that you can always use the same `itShouldAnswerYes` / `itShouldAnswerNo` method.

3. Add a scenario and needed step definitions to test that pop returns what has been pushed, e.g. when you push 42 and pop, then you get 42. *Hints:*
   (a) Cucumber will generate parameterised Java method snippets for you, however, it is recommended to replace there the generated reference to type `Integer` by `int`.
   (b) Use a generic approach comparable to Hint 2(b), just using `int` instead of `Boolean` to pass the result from method doing the pop to the method doing the `assert`.

4. Add a scenario and needed step definitions to test that after pushing the same value ten times, the stack is full and a pop from the full stack returns then that value. Take care that the number of pushes and the value to be pushed are two `int` parameters.

5. Use a test-driven (see slides 8-42/43)/behaviour-driven approach to add to `IntStack` a functionality to return the current size of the stack as integer, i.e. start with a minimal test (e.g. using an empty stack), have then a minimal implementation that fails that test to be sure that your test can fail at all; only after that, make your implementation pass your first assertion by returning a hard-coded value). Then, add a further scenario, to test more thoroughly than just your minimal test and extend your implementation to make both scenarios pass. (Always have the simplest possible implementation that just passes your tests. If you have at the end the feeling that your implementation is too simple, you did not add enough tests.)

As solution, upload three files:

1. feature file,
2. Java source code of your step definitions,
3. Java source code of your modified `IntStack` class.

To keep in mind during flipped classroom sessions

- First: Solution of last week's assignment 10 presented.
- Next: Questions concerning slides/videos (Coverage, Mutation testing, Cucumber).
- Then: Outlook
  - Next week (22.3.2021) is last regular class (test tools, test management);
  - 29.3.2021: start (2 weeks not counting Easter) project work (allowed as pair of two) on testing something bigger of your choice (counts 20%, assignments: 20%, exam: 60%), e.g. add tests for your Hugbúnaðarverkefni project. If you do not have a project, Helmut will give you one: write autograders that test student submissions. Encouraged to try new test tools. Last teaching week (19.4.2021): short 5 minute presentations of project results. 21.4.2021: info an exam and ISTQB certification.
  - Exam: planned oral, but if Corona situation allows, might be written on campus.
- After outlook: New assignments 11/12 (EclEmma/PIT, Cucumber) introduced.
- Finally: Work on assignments in breakout rooms: by default random allocation of 2 students each, but: **If you want to work in the breakout rooms with your favourite buddy, this is possible** as you can join on your own to whatever breakout room. But it is recommended to **let Helmut know via the chat** to coordinate who will go to which breakout room.
  - Breakout rooms do not get recorded.
  - First thing to do:
    1. Remember breakout room number: in case you need to reconnect, Helmut can then add you again to your old breakout room.
       - **if Helmut remembers to enable, you can also join breakout rooms yourself** / if you join late, you can add yourself to an empty breakout room or one that has only one student.
    2. Exchange contact details (email/phone), e.g. via chat in each breakout room, so that you can continue in case of technical problems/after class finishes.
  - While there are Eclipse plug-ins for shared editing, I suggest to have one student editing and sharing the screen to the other student.
  - Use "Ask for help" button (question mark icon) to make Helmut join (but may take time if he is busy in another breakout room).
    - Redo "Ask for help" if Helmut does not come after a few minutes (the requests do not queue up, but Helmut sees them only displayed once). In case of being idle, Helmut will come along each breakout room.
  - Submit source code file to Gradescope as a team (unless you disagree on solution): use Gradescope's group submission feature. Gradescope is reachable via Canvas. You have until Monday for submission (re-submission possible).
  - In case of some announcement for all: Helmut sends a chat message to all or ends all breakout room sessions for a video session with all. (If Helmut does not mess it up, should be possible to continue in the old breakout room.)
- There is no wrap-up at end of class, i.e. if you are finished with your assignment, there is no need to wait – you are welcome to leave the class.