TESTS NAMING, STRUCTURE AND GUIDELINES

Test class name: CustomerShould (with Should suffix)

Test methods: create\_transaction\_when\_deposit\_placed (underscore instead of cammelCase, first part is the THEN of the test, second part is the WHEN part)

Test structure:

- GIVEN (state of the class)/WHEN (action performed)/THEN (side effect expected)

- extract all test irrelevant data (vars, initializations, etc) outside of the test method and name them to be readeable

- try to extract more complex object creations to one place (could be builder) in order to change in one place if needed

- when you want to test legacy code, identify the "seam" points. Those are hard wired dependencies like DB invokation, sessions, caches, web services, etc. Extract them from the method to package protected methods. Then in your test extend the class, override those seams and mock them (if you want to eliminate them). Also start testing the legacy method from the most external exit point and go deeper.

- TDD could be applied for explolatory style of design (starting from input and output without going for the solution first)

Usefull tips:

- when you have multiple examples of the same behaviour (multiple different inputs), use a parameterised test: https://github.com/Pragmatists/JUnitParams

- the proper testing pyramid is

    1) a little acceptance E2E tests (just happy business cases), cus they are slow and hard to maintain

    2) a little more integration/API tests which are slower, since they communicate with infrastructure

    3) big (enough) amount of unit tests which are fast

- commit frequently, this enables you to revert to particular state of your code easily

- when an idea is discussed, if we do not have an aggreement within 30 seconds to 1 minute, each one of us forks a branch and starts typing something real to show (prototype, pseudo code, etc)