<epam>

Unit testing

Ákos Osvald
Budapest 2019

About Unit Testing

WHAT IS IT?

- Unit test is for testing the code on the lowest level
- In Java this means testing through – public – methods
- Quickest way to get feedback about the code
- White-box testing
- The wiedes range of tests

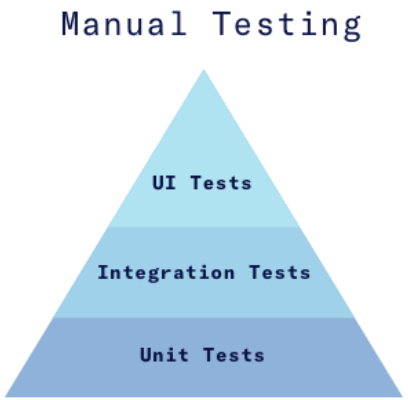
GOALS

- Uncovering implementation problems (mostly in TDD)
- Documenting code
- Detecting possible design problems in class
- Acting as part of regression test

A good unit test is:

F.I.R.S.T. PRINCIPLES OF UNIT TESTING

- **F**ast – should be the fastest tests you write, around milisec runtime
- **I**ndependent – tests must not depend on each other
- **R**epeatable – multiple runs should yield the same results
- **S**elf-validating – test should be able to tell if it is successful or not
- **T**imely – tests should be written the same time as the implementation



Construction of a test

RULES OF CODE

- Testing Absolute class
- Test code should be separated from production code
- Still should go into the same package
- Class and test name should follow naming convention
    - <class name>Test
    - <method name>Test
- A unit test can have multiple asserts, but
- One test should test one aspect of the behavior only, don't be afraid having multiple unit tests
- Test method should be public with no return type (`void`) and cannot have any parameters
- Test method should be annotated with `@Test` (org.junit.Test)

AAA PATTERN

- Unit tests should follow the AAA pattern
    - Arrange
    - Act
    - Assert
- Most commonly known as GIVEN/WHEN/THEN
- In each test an 'A' can happen only once, if you need repeat that probably should be multiple tests

Frameworks

Rest-Assured

- Always use framework
- Don’t mix them
- Know their behavior
- Most common:
    - Junit (we will be using this)
    - TestNG
- These are implementing the xUnit pattern

TEST CASES

Test cases

WHAT TO TEST?

- Proper amount and scenarios are come with practice
- But you should test:
    - Happy path
    - Edge cases
- Test coverage can lead you

RUNNING A TEST

- In IDE
    - Right click -> run test
    - It collects results
    - Easier to debug
- Command line:
    - In Maven: mvn test
    - It runs all the test
    - There are parameters to limit it, but we will be using IDE for now

Test cases

TEST STAGES

- Another reason for using framework is that they support before/after setup
- `@Before, @BeforeClass, @After, @AfterClass` methods
- Helps maintaining independent tests and reducing code duplication

ASSERTIONS

- Testing frameworks offer multiple choices
- There are also 3rd party libraries like Truth or AssertJ, both has fluent API
- You can also implement your own
- Try using the most precise assertion possible, don’t solve everything with assertTrue/assertFalse
- Think about test output, if not that clear you should use assertion message

EXCEPTIONS

Exceptions

- Whenever an unexpected condition occurs, an exception can be thrown with an exception object as a parameter
    - The normal program control flow stops and the search for a catch block begins
    - Either a catch been found, or the exception handled by JVM and program terminates
- Can happen even to the best of code
- Many times it is intentional and part of the solution
- We must test them!

Let’s assume there is a method that throws exception if the argument is negative

OPTION 1	OPTION 2	OPTION 3

`@Test(expected = IllegalArgumentException.class)`
• + framework supported
- cannot check for exception details
	`try{`
	`  undertest.call(-1);`
	`} catch (Exception exc){`
	`  //assertions here`
	`}`
	• What is wrong with this?
		`try{`
		`  undertest.call(-1);`
		`  fail(”error”);`
		`} catch (Exception exc){`
		`  //assertions here`
		`}`
		• Close enough for our needs

MOCKING

Mocking

PURPOSE

- Unit test is for testing a single method in a class
- That means external dependencies must be mocked
- We can use a fake object to mimic the behavioir of the original ones

HOW TO DO?

- Frameworks:
  - **Mockito – we will use this**
  - EasyMock
  - PowerMock

Test doubles

- Dummy
  - Empty call without logic, like a model class with only getter setter. For those using mocking is acceptable, but a bit of overkill
- Stub
  - A fake class created in the background, does not keep original logic, offers behavior manipulation (we are not using this in unit testing)
- Mock
  - Same as a stub, but it also keeps the records of the method calls for later verifications
- Spy(in Mockito)
  - It’s a mock, but it uses logic from original class that can be changed but it’s not necessary (we are not using this in unit testing)

Mocking in practice

MOCKITO USAGE

- RandomString *stringGeneratorMock* = Mockito.mock(RandomString.class);
- Creates a mock object
- Right now it's empty, method calls will return default value for the return type
- We can setup behavior like:
    - *when(stringGeneratorMock.createString(10)).thenReturn("abcde");*
    - *when(stringGeneratorMock.createString(-1)).thenThrow(**new IllegalArgumentException());***

MOCKS USED FOR VERIFICATIONS

- *verify(stringGeneratorMock,times(1)).createString(10);*
- *checks if the createString method was called once with the parameter 10*
- *acts same way as an assert method would*
- *can have multiple verification*

THANK YOU FOR YOUR ATTENTION!