UNIT TESTING

WHAT IS UNIT TEST

- Unit test is for testing the code on the lowest level
- In Java this means testing through – public – methods
- It is quickest way to get feedback about the code
- It is white-box testing

GOALS OF UNIT TESTING

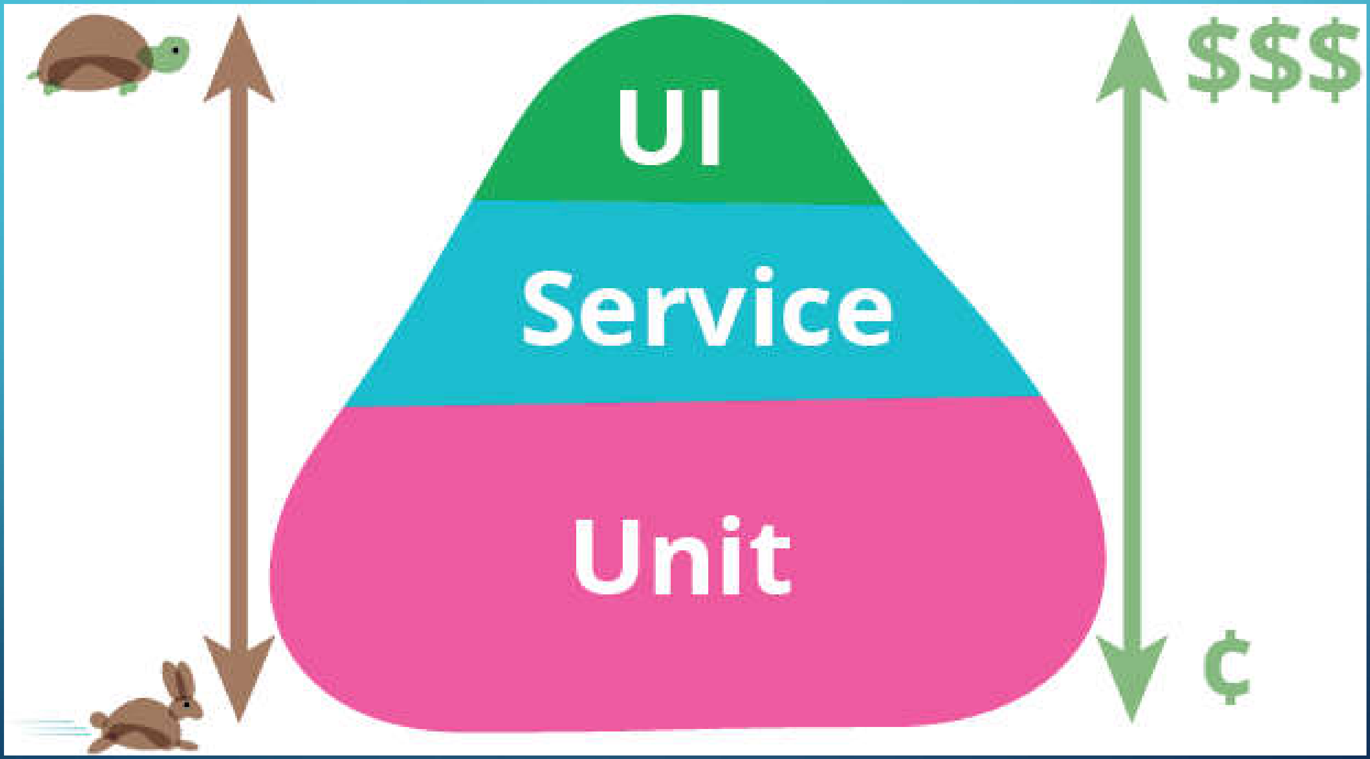
- Uncovering implementation problems (mostly in TDD)
- Documenting code
- Detecting possible design problems in class
- Acting as regression test

A GOOD UNIT TEST IS:

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

These are the **FIRST** principles

TESTING PYRAMID



FRAMEWORKS

- 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

CODE

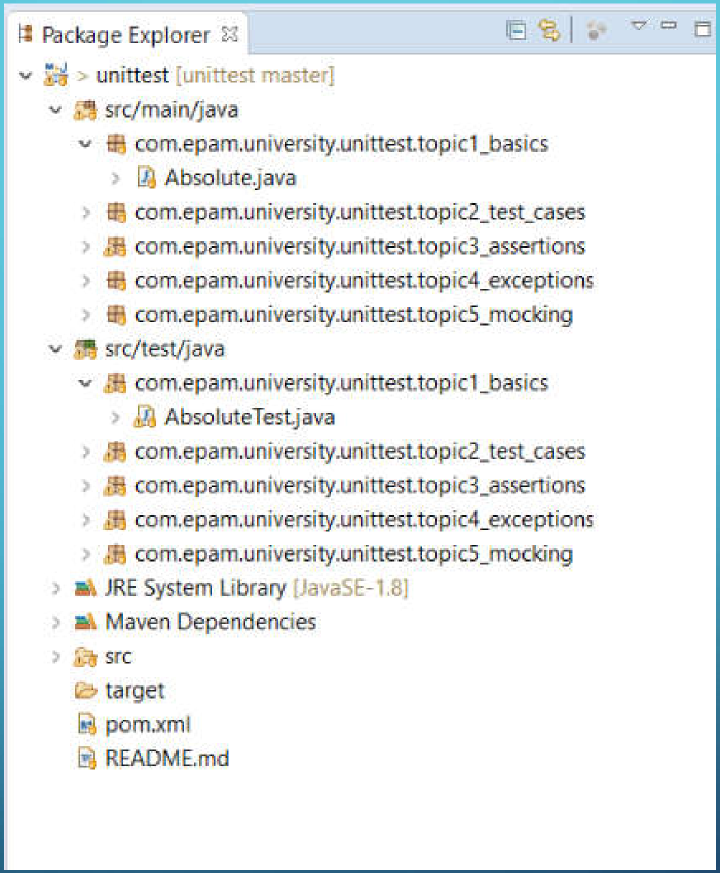
- git clone https://github.com/Zolikon/unittest.git
- Import in IntelliJ: File -> New -> Project from Existing Source
- Set location
- Maven project
- Java 1.8 if possible

CREATING A SIMPLE UNIT TEST

- 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
- One 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

CREATING A SIMPLE UNIT TEST

- Test method should be public with no return type (void) and cannot have any parameters
- Test method should be annotated with @Test (import org.junit.Test;)



```
Package Explorer
v unittest [unittest master]
  v src/main/java
    v com.epam.university.unittest.topic1_basics
      > Absolute.java
    > com.epam.university.unittest.topic2_test_cases
    > com.epam.university.unittest.topic3_assertions
    > com.epam.university.unittest.topic4_exceptions
    > com.epam.university.unittest.topic5_mocking
  v src/test/java
    v com.epam.university.unittest.topic1_basics
      > AbsoluteTest.java
    > com.epam.university.unittest.topic2_test_cases
    > com.epam.university.unittest.topic3_assertions
    > com.epam.university.unittest.topic4_exceptions
    > com.epam.university.unittest.topic5_mocking
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > src
    target
    pom.xml
    README.md
```
]

AAA PATTERN

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

RUNNING 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

- It comes with practice
- But you should test:
  - Happy path
  - Edge cases
- You can use test coverage, but that can be misleading (coverage is more for the managers than for the developers ☺)
- Let’s see the Calculator class

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, but most likely you won’t have to
- 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
- StringUtils class

TEST SETUP

- 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
- For unit test you only need @Before, the others are not needed

EXCEPTION

- Can happen to the best of code
- Many times it is intentional
- We have to test for them

EXCEPTION

Options:

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

- Option 1:
  - @Test(expected = IllegalArgumentException.class)
  - + framework supported, - cannot check for exception details

EXCEPTION

Options:

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

- Option 2: What is wwrong with this?

```
try{
    undertest.call(-1);
} catch (Exception exc){
    //assertions here
}
```

EXCEPTION

- Solution

```
try{
        undertest.call(-1);
        fail("exception should have been thrown");
} catch (Exception exc){
        //assertions here
}
```

- There are more advanced solutions, but you won't need them in this course

MOCKING

- Unit test is for testing a single method in a class
- That means external dependencies have to be mocked
- We can use a fake object instead
- How to do this manually?

MOCKING

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

MOCKING

- 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

- 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());***

MOCKING

Mocks from framework can also be 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*