UNIT TESTING

With JUnit in Eclipse

CONTENT

- What does unit testing mean?
- Why is it so important?
- How does a good Unit Test look like?
- JUnit
- Mockito
- Example in Eclipse
- Exercise

WHAT DOES UNIT TESTING MEAN?

- Unit tests are typically automated tests written and run by software developers
- Not every automated test is called unit test!
- We write unit tests to prove that the behavior of an implemented functionality works like intended
- Unit tests sould focus on one single action of our program

WHAT DOES UNIT TESTING MEAN?

- While a unit test does only test one component or unit of our code, we have no connection to any database or other network in this moment
- Unit testing requires a module based structure of code, in other words you need to have small testable components
- Sometimes it is usefull to write unit tests before implementing the logic

WHY IS UNIT TESTING IMPORTANT?

- Unit testing involves breaking your program into pieces, and subjecting each piece to a series of tests.
- The structure you need to write good tests, increases readability
- Overall, a large number of errors or bugs are found before customers can experience them
- To detect mistakes while you work on something does save a lot of time, because a that moment you know this area of code the best

WHY IS UNIT TESTING IMPORTANT?

- Unit Tests save the behavior which is defined in a project. So if anyone touches the logic of tested code the unit test will fail if the behavior changed.
- A good software developer will not delete or change a test without thinking, if his changes are working correctly
- Testing classes in isolation is super fast compared to other testing methods

WHAT ARE GOOD UNIT TESTS ABOUT

A good unit test consits of:

- An informative name
- Three separated blocks (arrange, act, assert)
- A test case which simulates the normal use of a component when the software is running (Every time we use Mockito we should think about it, if it is needed)
- Specific assertions to be sure, that the result is correct

WHAT ARE GOOD UNIT TESTS ABOUT

- Because a good unit test is testing one specific behavior, the naming should be very easy
- If a test breaks other developers should easily understand the problem
- Using for example asserThat() for better error messages

JUNIT

- Framework for automated testing of java classes and methods (unit tests)
- Can also be used for other types of tests
- Integrated in Eclipse and other development environments
- The result of a test can only be success(green) or error(red)

JUNIT ANNOTATIONS

Annotation JUnit5	Description
@Test	designates a test method
@ParameterizedTest	denotes that a method is a parameterized test
@BeforeEach	Designates a method, which runs before every testmethod is executed
@AfterEach	Designates a method, which is executed after every single test method
@BeforeAll	denotes that a method runs one time at the beginning, before any of the test methods is executed
@AfterAll	Denotes that a method is executed after all test methods are finished
@Disabled	Used to disable a test class or a test method

JUNIT ASSERTIONS

Statement	Description
assertTrue(boolean condition)	Checks if the boolean condition is true
assertFalse(boolean condition)	Checks if the boolean condition is false
assertEquals(expected, actual)	Checks if both values ar eual (Not working with Arrays, because only the reference is compared)
assertThat(value, matcher statement)	Checks if the value fits the matcher

JUNIT HAMCREST MATCHER

Using assertThat has ceveral advantages:

- More readable and typeable assertions
- It is possible to write own matcher for your type of result
- Matcher statements can be combined
- Readable failure message

MOCKITO

- Mockito is a Java library which helps to create simple unit tests with good performance
- To get a completely isolated test environment you need to shut down all interfaces/connections to the outside
- Mockito can simulate methods, classes and objects
- For mocked objects you can define actions which should happen when a method of this mock is called

EXERCISE

Load the test project from: http://softwareengineering.freeforums.net/thread/729/unit-testing-junit

Got to the test class EnemyTest and complete the two test methods called:

- ifDamageToEnemyHigherThanHP_EnemyShouldBeDead()
- whenEnemyMoves_ThePositionShouldChange()