Service Testing w/ Mockito

Open source testing framework for Java. The framework allows the creation of test double objects in automated unit tests for the purpose of TDD or BDD.

Outline

1. JUnit Annotations & Assertions Rereview
2. Testing Set Up
3. Test Methods
4. Open Book Quiz

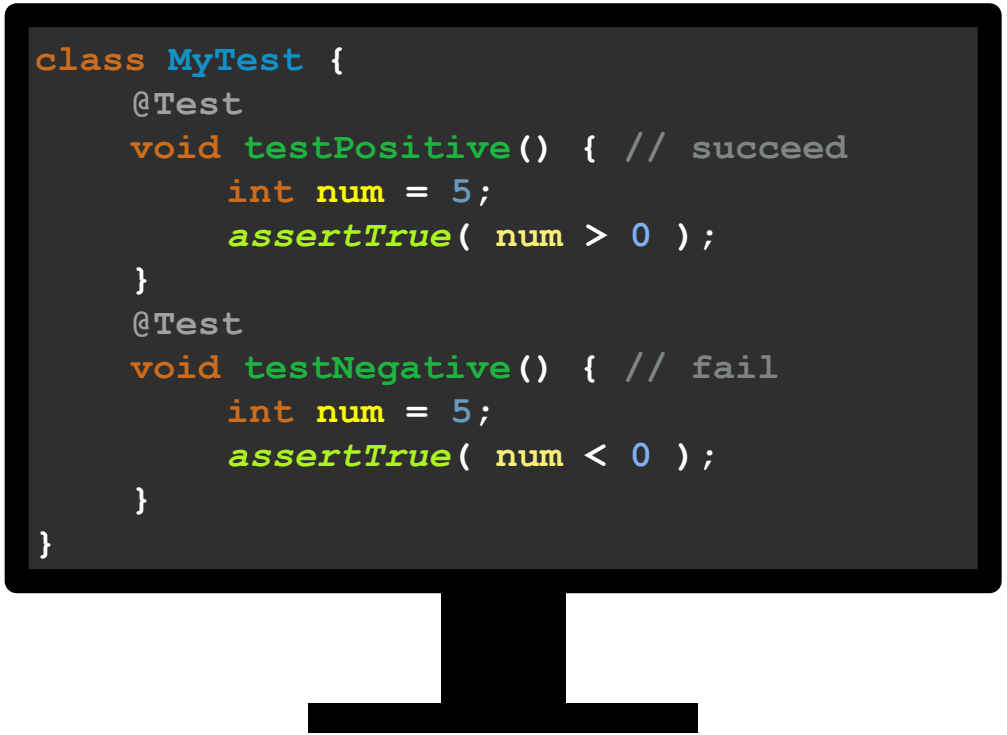
JUnit Annotations & Assertions Rereview

JUnit

- **JUnit** is a Unit Testing Framework supported in Java
- Can follow the flow of TDD in JUnit
  - Developers write tests before writing code
  - Easy way to run every test
- ***As code is changed, every test should be rerun*** in case something breaks with new updates
- Each test case is a method, these methods are executed one by one
- Other types of methods can be written to do setup and cleanup of resources after running tests

Test File

- JUnit test files ***don't contain a main method***
- Have ***multiple test methods***, each called by JUnit when file is run
- Test cases will fail or succeed based on if condition checked
- Conditions are checked with assertion methods from `Assert` class



Assertion Class Methods

The methods from the Assertion class can be used to test conditions in a test case. When the test file is run, if the assertion is true, will pass, if not, will fail.

assertTrue()	Succeeds if condition passed is true.
assertFalse()	Succeeds if condition passed is false.
assertSame()	Succeeds if two objects compared are the same object.
assertNotSame()	Succeeds if two objects compared are NOT the same object.
assertNull()	Succeeds if object checked is null.
assertNotNul()	Succeeds if object checked is not null.

Examples on how to use this assertions: https://www.guru99.com/junit-assert.html

Assertion Class Methods

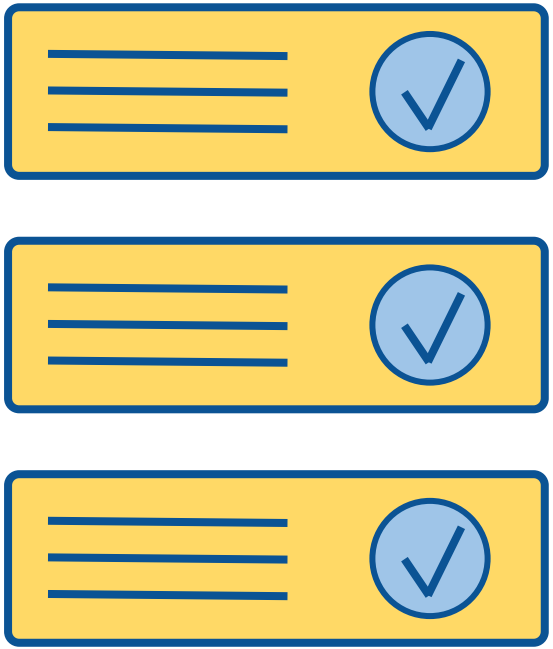
The methods from the Assertion class can be used to test conditions in a test case. When the test file is run, if the assertion is true, will pass, if not, will fail.

assertEquals()	Succeeds if two values compared are the equal. If comparing two objects, will use the equals() method to check for equality.
assertNotEquals()	Succeeds if two values compared are NOT equal. Still uses the equals() method.
assertThrows()	Succeeds if the code throws the specified exception.
fail()	Always fails if run.

Examples on how to use this assertions: https://www.guru99.com/junit-assert.html

Before & After Annotations

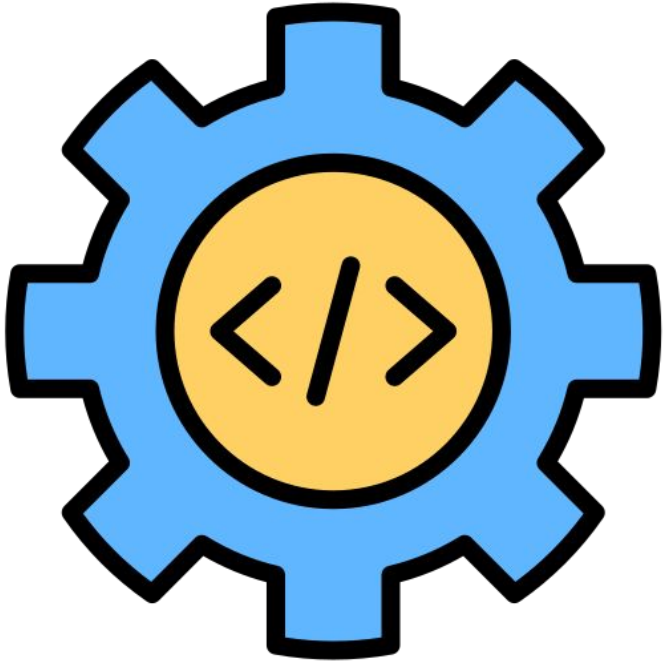
- When running tests, setup and cleanup may be necessary when running multiple tests
- In order to set up resources, set up `@BeforeAll`
- Any cleanup can be handled with `@AfterAll`
- Then if values need to be reset, setup required before each test, `@BeforeEach`
- Can also use `@AfterEach` if need code to run after each test



JUnit Annotations

Table Header	Table Header
`@Test`	Method will be executed as a test case. When test file is run, each method labeled with this annotation will fail or succeed.
`@BeforeAll`	Method will be run once before all the test cases are run. Useful to do any setup needed to run all your tests.
`@BeforeEach`	Method will be run before each test case.
`@AfterAll`	Method will be run once after all the test cases are run. May be used to do any cleanup or disconnect from any resources.
`@AfterEach`	Method will be run after each test case.

Testing Set Up



Testing Service Layer

- Tests for the **controller layer** focus on the ***HTTP response/request logic***
  - Request accepts the right arguments, status code is correct, body formatted properly, etc.
- Testing the **service layer** is like testing a regular class with JUnit
  - Test a singular method and ***expect a certain value to be returned***
- However, **Mockito** still needs to be used in order to mock the repository layer

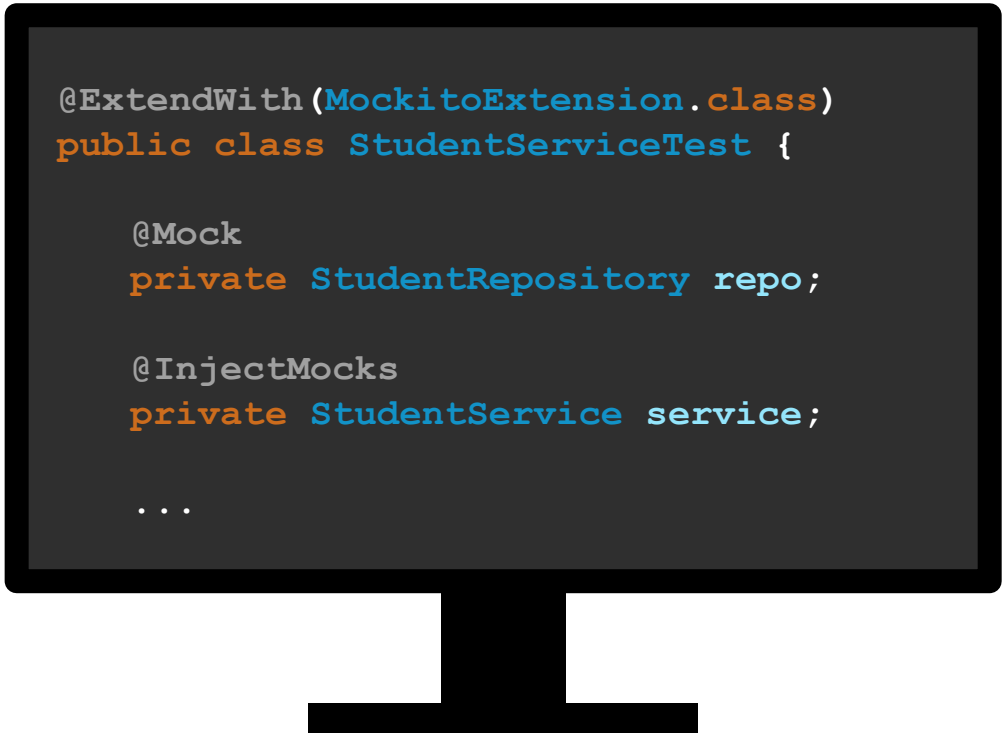
```java
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class StudentServiceTest {
    ...
}
```

- ➔ Start by marking the test class with `@ExtendWith(MockitoExtension.class)`
- ➔ This will ensure that Mockito can **create and manage the mocks we will setup** within this class
- ➔ We will now be able to use `@Mock` and `@InjectMocks`

Mock Object Setup

- Once class is setup, we use `@Mock` to mark the repository so we can mock its methods
- The `@InjectMocks` annotation on our service to make sure we create the “mocked” repository and can dictate its behavior



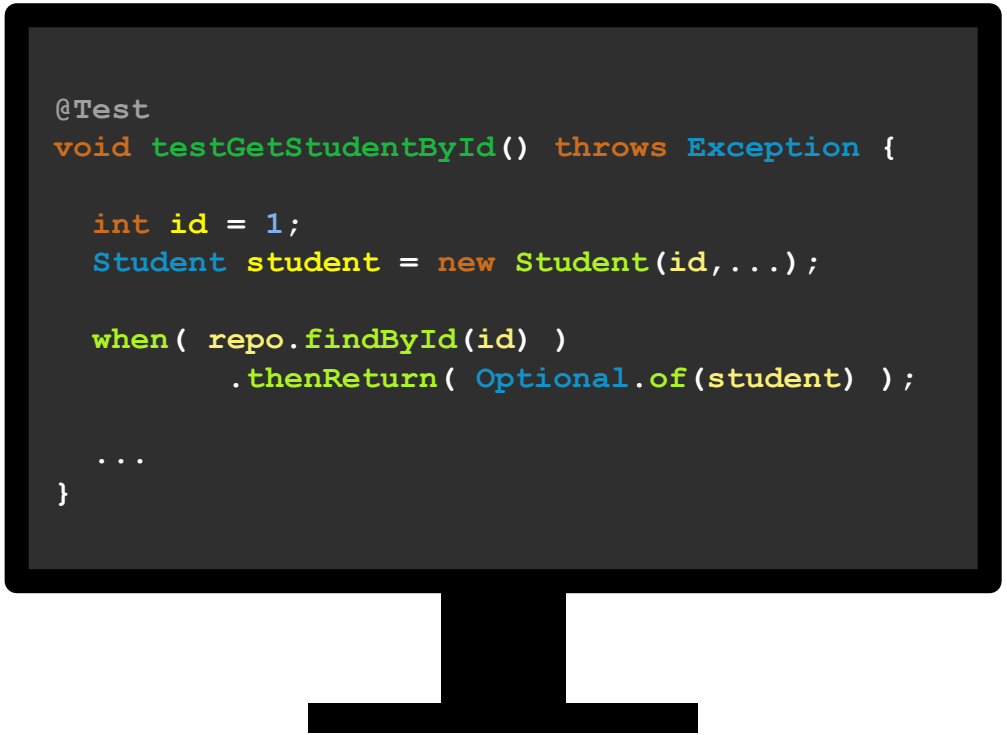
Test Methods

Arrange, Act, Assert

- To test each method in your service layer, follow these steps:
  - **Arrange:** setup all the resources you'll need for your test
  - **Act:** perform the actual test
  - **Assert:** check if your test succeeded/failed
- During the ***arrange*** step, you may need to **use Mockito to mock methods from the repository**
- Use the assertion methods from JUnit to do the ***assert*** step

Arranging Resources

- When testing a service method, look to see what ***information needs to be provided and returned*** when calling the repository methods
- You can still use the `when()` method to mock the repository and return your own values



Act & Assert

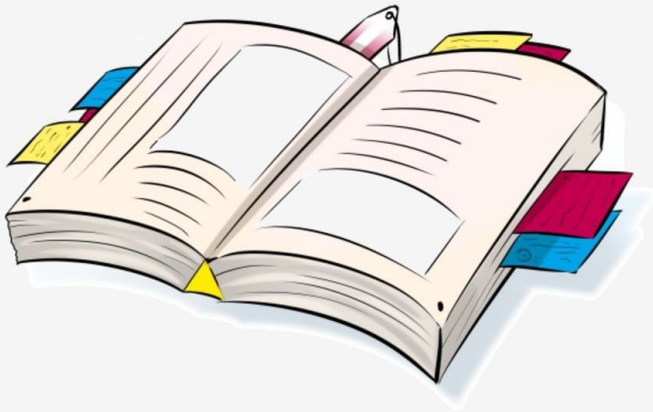
- ➔ Call the service method you want to test and get the result of that call
- ➔ Then use one of the assert methods from the JUnit library to check that the result matches what you expect

Example Code

- ➔ Please review the following project: **SpringStudentValidationExceptionHandling**
- ➔ Make sure to to review the `StudentServiceTest.java` class within ***src/test/java***

Open Book Quiz on Service Testing

- ➔ **Check the README for the quiz link**
- ➔ This is an **open note**, multiple choice quiz
- ➔ Have it completed by the **start of class tomorrow at 10AM EST**
- ➔ If there are ***any questions, ask your instructor during this time or during office hours***, as they may not able available after hours



Cognixia™