**Автоматизированное Юнит и Интеграционное тестирование Java приложений с использованием Junit 5 И Mockito**

Зачем?

**Тестирование: ручное или автоматизированное?**  
  
Сначала надо понять различия между ручными и автоматизированными тестами. Ручное тестирование проводится непосредственно человеком, который нажимает на кнопочки в приложении или взаимодействует с программным обеспечением или API с необходимым инструментарием. Это достаточно затратно, так как это требует от тестировщика установки среды разработки и выполнения тестов вручную. Имеет место вероятность ошибки за счет человеческого фактора, например опечатки или пропуска шагов в тестовом сценарии.  
  
Автоматизированные тесты, с другой стороны, производятся машиной, которая запускает тестовый сценарий, который был написан заранее. Такие тесты могут сильно варьироваться в зависимости от сложности, начиная от проверки одного единственного метода в классе до отработки последовательности сложных действий в UI, чтобы убедиться в правильности работы. Такой способ считается более надежным, однако его работоспособность все еще зависит от того насколько скрипт для тестирования был хорошо написан.  
  
Автоматизированные тесты – это ключевой компонент непрерывной интеграции ([Continuous Integration](https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration)) и непрерывной доставки ([continuous delivery](http://https/www.atlassian.com/continuous-delivery/pipeline)), а также хороший способ масштабировать ваш QA процесс во время добавления нового функционала для вашего приложения.

Во время ручного тестирования мы подготавливаем окружение (зависимости), запускаем приложение, вводим тестовые данные, проверяем работоспособность, вводим другие тестовые данные и так пока не проверим все возможные варианты входов и выходов, разумеется, если мы хотим, чтобы наше приложение работало. Все это занимает очень много времени и чем больше становиться приложение, тем сложнее выполнять этот процесс. В результате, программист практически перестает заниматься проверкой функциональности, особенно старой, что может привести к неожиданному поведению программы, а возможно даже к критическим для бизнеса ошибкам и потерям денег и времени. Похожий сценарий может произойти с автоматизированными тестами плохого качества, но об этом позже. (график с сравнения ручного с автоматизированным)

Определение юнит-теста

Юнит-тестом называется автоматизированный тест, который:

* проверяет правильность работы небольшого фрагмента кода (также называемого юнитом);
* делает это быстро
* и поддерживая изоляцию от другого кода

Неоднозначность может вызывать первый и третий пункт. Вопрос изоляции и вопрос того, что является юнитом - это основное различие между лондонской и классической школой юнит-тестирования.

**Лондонская школа**

Подход лондонской школы подразумевает изоляцию тестируемого кода, от его зависимостей (коллабораторов). То есть все зависимости класса нужно заменить на тестовые заглушки. Тестовая заглушка – объект, который имеет такой же интерфейс, как и заменяемый, но ведет себя так, как нам нужно для тестирования. Разновидности и создание тестовых заглушек рассмотрено позже. Под юнитом в лондонской школе подразумевается один класс или методов.

**Классический подход**

В классическом подходе изолируются друг от друга не классы, а тесты. То есть последовательность выполнения не влияет на результат. Тестируемые классы не должны обращаться к общему состоянию, через которое они могу влиять друг на друга.

**Список использованных источников:**

1) jUnit5 About: [Электронный ресурс]. URL: https://junit.org/junit5/ .  
2) jUnit5 JavaDocs: [Электронный ресурс]. URL:  
https://junit.org/junit5/docs/current/api/ .

3) jUnit5 исходный код [Электронный ресурс]. URL:  
https://github.com/junit-team/junit5 .

4) Mockito framework site [Электронный ресурс]. URL: <https://site.mockito.org/>.

5) Mockito исходный код [Электронный ресурс].URL: <https://github.com/mockito/mockito>.

6) Mockito JavaDocs [Электронный ресурс]. URL: <https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html>.

7) Статья: “Mockito и как его готовить” [Электронный ресурс]. URL: <https://habr.com/ru/post/444982/>.

8) Mockito Tutorial Baeldung [Электронный ресурс]. URL: https://www.baeldung.com/mockito-series

9) A Guide to Junit5 [Электронный ресурс]. URL: <https://www.baeldung.com/junit-5>.

10) Parallel Test Execution for Junit5 [Электронный ресурс]. URL: <https://www.baeldung.com/junit-5-parallel-tests>.

11) Эккель Б. Философия Java, 4-e изд. – Спб.: Питер, 2021. – 1168c.

12) Бек К. Экстремальное программирование: разработка через тестирование. – Спб.: Питер, 2022. – 224с.

13) Хориков В. Принципы юнит-тестирования. – Спб.: Питер, 2022. – 320с.

14) Mellor A. Test-Driven Development with Java. – Birmingham: Packt, 2023. – 325с.

15) Freeman S., Pryce N. Growing Object-Oriented Software, Guided by Tests. – Boston: Addison-Wesley, 2010. – 385с.

16) Мартин Р. Чистый код: создание, анализ и рефакторинг. – Спб.: Питер, 2022. – 464с

https://habr.com/ru/companies/otus/articles/443418/