**Junit для профи**

Я считаю, что разработка ПО — нечто большее, чем работа. Я вижу себя ремесленником, который каждый день пытается стать лучше. Самый «простой» путь для этого — найти несколько хороших инструментов и ответить на следующие вопросы:

* Когда я должен использовать инструмент X?
* Как я должен использовать инструмент X?

Автоматизированное тестирование — очень важная часть разработки ПО, но в блогах программистов немного постов про используемые инструменты. Эта же статья позволит вам заглянуть в мой «ящик с инструментами». Я расскажу о 12-ти библиотеках и фреймворках, которые я использую для написания unit- и интеграционных тестов, а также предоставлю ссылки на страницы, которые помогут вам понять, как их использовать.

Заглянем в мой ящик с инструментами

Перед тем, как вы сможете использовать описанные далее инструменты, вы должны настроить сборку, которая автоматически запускает интеграционные и unit-тесты. У меня есть 2 заметки по этой теме:

* [Интеграционное тестирование с Maven](http://www.petrikainulainen.net/programming/maven/integration-testing-with-maven/) описывает, как мы можем настроить Maven-сборку с интеграционными и unit-тестами в разных директориях.
* [Начало работы с Gradle: интеграционное тестирование с плагином TestSets](http://www.petrikainulainen.net/programming/gradle/getting-started-with-gradle-integration-testing-with-the-testsets-plugin/) описывает то же самое для Gradle.

Теперь вы готовы взглянуть на мои инструменты поближе. Я разделил их на категории, чтобы вам было проще ориентироваться.

Итак, вот 12 инструментов, которые я использую при интеграционном и unit-тестировании.

Запуск тестов

[JUnit](http://junit.org/) — это фреймворк, который я использую как для unit-, так и для интеграционных тестов. Мне он очень нравится, т.к. он самый популярный, поэтому имеет множество расширений. Также, если у вас возникнут проблемы — найти решение будет несложно.

[NestedRunner](https://github.com/NitorCreations/CoreComponents/tree/master/junit-runners) — расширение для JUnit, которое позволяет запускать тестовые методы из вложенных классов. Мне нравится NestedRunner по ряду причин:

* можно заменить длинные имена методов на иерархию классов в стиле BDD;
* можно избавиться от дублирующего кода путём перемещения его в установочные методы в нужных вложенных классах;
* можно объявить константы во вложенных классах и связать их с тестами, которым эти константы нужны.

[junit-davaprovider](https://github.com/TNG/junit-dataprovider) — расширение для JUnit, позволяющее писать параметризованные тесты с использованием [TestNG в качестве провайдера данных](http://testng.org/doc/documentation-main.html" \l "parameters-dataproviders" \t "_blank). Это большое улучшение по сравнению с [обычным способом написания параметризованных тестов](https://github.com/junit-team/junit/wiki/Parameterized-tests), который, прямо скажем, не очень.

Дополнительно:

* [Unit-тестирование с JUnit](http://www.vogella.com/tutorials/JUnit/article.html) описывает, как писать тесты с JUnit 4.X.
* [Написание «чистых» тестов: красота в маленьком размере](http://www.petrikainulainen.net/programming/testing/writing-clean-tests-small-is-beautiful/) рассказывает о решении проблем, связанных с написанием «чистых» тестов.
* [Начало работы с junit-dataprovider](https://github.com/TNG/junit-dataprovider/wiki/Getting-started).
* [Возможности junit-dataprovider](https://github.com/TNG/junit-dataprovider/wiki/Features).
* [~~junit-dataprovider — альтернатива параметризованным классам JUnit~~](http://java.dzone.com/articles/alternative-junit) описывает, как перейти от стандартных параметризованных тестов к тестам, которые используют junit-dataprovider.

Макеты, заглушки, подмены

[Mockito](http://mockito.org/) — самый популярный фреймворк с поддержкой макетирования для unit-тестов. Мне он нравится из-за простого API, [множества полезных возможностей](https://code.google.com/p/mockito/wiki/FeaturesAndMotivations) и [превосходной документации](https://javadoc.io/page/org.mockito/mockito-core/latest/org/mockito/Mockito.html).

[Greenmail](https://github.com/greenmail-mail-test/greenmail) — сервер электронной почты, который поддерживает SMTP, POP3 и IMAP с поддержкой SSL-соединения. Он мне нравится из-за простоты использования. Когда я искал отладочный сервер электронной почты, пересмотрев несколько альтернатив, остановился на Greenmail, т.к. он работал именно так, как мне требовалось.

[MockFtpServer](http://mockftpserver.sourceforge.net/) — библиотека, которая предоставляет две разные реализации FTP-сервера («заглушка» и «обманка»), которые можно использовать для [тестирования различных сценариев](http://mockftpserver.sourceforge.net/fakeftpserver-versus-stubftpserver.html). Если вам нужно протестировать код, взаимодействующий с FTP-сервером, наш выбор — MockFtpServer.

Дополнительно:

* [Справочник по Mockito](https://javadoc.io/page/org.mockito/mockito-core/latest/org/mockito/Mockito.html) — отличный пример справочной документации, которая действительно помогает.
* [Интеграционное тестирование IMAP, SMTP и POP3 с помощью Greenmail](http://www.hascode.com/2012/07/integration-testing-imap-smtp-and-pop3-with-greenmail/).
* [Начало работы с FakeFtpServer](http://mockftpserver.sourceforge.net/fakeftpserver-getting-started.html).
* [Начало работы с StubFtpServer](http://mockftpserver.sourceforge.net/stubftpserver-getting-started.html).
* [Что выбрать: FakeFtpServer или StubFtpServer?](http://mockftpserver.sourceforge.net/fakeftpserver-versus-stubftpserver.html) Поможет решить, какая реализация FTP-сервера необходима для проведения того или иного теста.

Утверждения

[Hamcrest](http://hamcrest.org/) предоставляет инструменты для написания утверждений (assertions) для unit- и интеграционнаых тестов. Я его использую вместе со [Spring MVC Test Framework](http://www.petrikainulainen.net/spring-mvc-test-tutorial/" \t "_blank).

[AssertJ](https://joel-costigliola.github.io/assertj/) предоставляет гибкий API для написания утверждений с полезными сообщениями об ошибках, улучшает читаемость тестового кода, позволяет превратить тесты в исполняемые спецификации, которые придерживаются нужного предметно-ориентированного языка.

Дополнительно:

* [Учебник по Hamcrest](https://code.google.com/p/hamcrest/wiki/Tutorial).
* [Использование Hamcrest в тестировании](http://www.vogella.com/tutorials/Hamcrest/article.html) рассказывает, как использовать Hamcrest для написания тестов, а также как расширить его возможности с помощью пользовательских модулей.
* [Обзор AssertJ Core](https://joel-costigliola.github.io/assertj/assertj-core.html).
* [Превращаем утверждения в предметно-ориентированный язык](http://www.petrikainulainen.net/programming/unit-testing/turning-assertions-into-a-domain-specific-language/) рассказывает, как создавать пользовательские утверждения в AssertJ.
* [Пишем чистые тесты: замена утверждений предметно-ориентированным языком](http://www.petrikainulainen.net/programming/testing/writing-clean-tests-replace-assertions-with-a-domain-specific-language/). Рассказывает, почему мы должны заменить стандартные утверждения JUnit собственными, которые используют верный предметно-ориентированный язык.

Тестирование кода доступа к данным

[H2](http://www.h2database.com/html/main.html) — быстрая БД, полезна для написания интеграционных тестов, которые запускаются на локальной машине разработчика.

[DbUnit](http://dbunit.sourceforge.net/) — расширение для JUnit, которое может быть использовано для инициализации БД в известное состояние перед выполнением каждого интеграционного теста и заполнения БД нужными данными. У DbUnit есть свои недостатки, но это очень полезный инструмент, позволяющий разделить тестовые данные и тестовый код.

Дополнительно:

* [DbUnit Core Components](http://dbunit.sourceforge.net/components.html) описывает ключевые компоненты DbUnit, знание которых необходимо для написания тестов с использованием DbUnit.
* [Лучшие практики DbUnit](http://dbunit.sourceforge.net/bestpractices.html) предоставляет пять правил написания лучших тестов для кода доступа к данным.
* [Написание тестов для кода доступа к данным](http://www.petrikainulainen.net/writing-tests-for-data-access-code/http:/www.petrikainulainen.net/writing-tests-for-data-access-code/) поможет вам писать детерминированные тесты доступа к данным, которые проверяют нужную вещь, легкочитаемые и поддерживаемые.

Тестирование Spring приложений

[Spring Test](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/testing.html) — швейцарский нож для написания автоматизированных тестов. Он предоставляет первоклассную поддержку написания unit- и интеграционных тестов для приложений, использующих Spring.

[Spring Test DbUnit](https://springtestdbunit.github.io/spring-test-dbunit/) — интегрирует DbUnit во фреймфорк String Test. Если вам нужно написать тесты доступа к данным для приложения, использующего Spring и реляционную БД, то Spring Test DbUnit вам в помощь.

Дополнительно:

* [Справочник по Spring. Раздел 11: тестирование](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/testing.html).
* [Руководсто по Spring MVC Test](http://www.petrikainulainen.net/spring-mvc-test-tutorial/) рассказывает о написании unit- и интеграционных тестов для контроллеров Spring MVC.
* [Руководсто по Spring Data JPA: интеграционное тестирование](http://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-integration-testing/).
* [Spring на передовой: использование Null-значений в наборах данных DbUnit](http://www.petrikainulainen.net/programming/spring-framework/spring-from-the-trenches-using-null-values-in-dbunit-datasets/).
* [Spring на передовой: сбрасываем поля с автоинкрементом перед каждым тестовым методом](http://www.petrikainulainen.net/programming/spring-framework/spring-from-the-trenches-resetting-auto-increment-columns-before-each-test-method/).