

Автоматизация тестирования Web UI на Java

BDD и Selenide

[Java 11]



На этом уроке

1. Изучим подходы к тестированию и разработке.
2. Узнаем, что такое BDD.
3. Познакомимся с Selenide.

Оглавление

[Введение](#)

[Что такое BDD?](#)

[Реализация BDD](#)

[Написание сценария](#)

[Реализация шагов сценария](#)

[Selenide](#)

[Метод \\$](#)

[Проверки в Selenide](#)

[Опции для настройки Selenide](#)

[Работа с ожиданиями](#)

[Используемые источники](#)

Введение

Этот урок посвящается двум темам: обёртке над Selenium WD под названием Selenide и подходу к тестированию, который называется Behavior Driven Development. Начнём с последнего.

Что такое BDD?

BDD — behaviour-driven development — это разработка, основанная на описании поведения. BDD предполагает описание тестировщиком или аналитиком пользовательских сценариев на естественном языке — если можно так выразиться, на языке бизнеса.

Тексты сценариев записываются в конкретной форме.

1. Имея (прим. given — данное) какой-то контекст.
2. Когда (прим. when) происходит событие.
3. Тогда (прим. then) проверить результат.

Пример:

```
Scenario: All done
  Given I am out shopping
  And I have eggs
  And I have milk
  And I have butter
  When I check my list
  Then I don't need anything
```

То же самое на русском:

```
Сценарий: Успешная аутентификация
  Допустим, пользователь вставляет в банкомат банковскую карту
  И банкомат выдаёт сообщение о необходимости ввода PIN-кода
  Если пользователь вводит корректный PIN-код
  То банкомат отображает меню и количество доступных денег на счету
```

Такой язык написания называется [Gherkin](#). Почти каждая строка сценария начинается с одного из его ключевых слов. Все тестовые сценарии хранятся в файлах, которые имеют расширение *.feature. В feature-файле может быть один или несколько сценариев, их удобно группировать по выполняемой функциональности или по странице, на которой происходят основные действия пользователя.

Чтобы писать на русском языке или любом другом, кроме английского (установлен по умолчанию), надо в начале файла со сценариями указать такую строчку:

```
# language: ru
```

Главное достоинство BDD-подхода — feature-файлы не зависят от языка, на котором мы программируем. [Есть фреймворки](#) для большинства известных языков программирования, у Java их — несколько. Самый популярный в последние годы — [cucumber](#), он разрабатывается компанией SmartBear.

Цель BDD-подхода для тестирования — убрать границу между аналитиками, ручными тестировщиками и автоматизаторами. Теперь кейсы хранятся в коде, и каждый шаг реализуется в соответствующих сценариях классов.

Реализация BDD

Написание сценария

Возьмём сценарий создания контактного лица из предыдущих ПЗ.

Создание контактного лица в организации с минимально заполненной информацией

Предусловия:

1. В системе есть хотя бы одна организация: список организаций можно посмотреть в «Контрагенты» → «Организации».

№	Шаг	Ожидаемый результат
1	Авторизоваться на сайте CRM, используя следующие данные: URL: https://crm.geekbrains.space/ Логин/пароль: Applanatest/Student2020!	Пользователь успешно авторизовался, видит страницу «Панель инструментов»
2	Перейти в «Контрагенты» → «Контактные лица»	Пользователь находится на странице «Контактные лица», видит таблицу контактных лиц, есть кнопка «Создать контактное лицо»
3	Нажать на кнопку «Создать контактное лицо»	Открыта страница создания контактного лица
4	Заполнить обязательные поля: 1. Фамилия. 2. Имя. 3. Организация. 4. Должность.	Поля заполнены
5	Нажать на кнопку «Сохранить и закрыть»	Страница создания контактного лица закрыта, пользователь видит страницу «Все контактные лица» и всплывающее

		уведомление «Контактное лицо сохранено»
--	--	---

Создадим файл `Contacts.feature` в `src/test/resources/<название пакета>` и перепишем его в Gherkin-нотации с небольшими изменениями. Не будем проверять наличие организации, перенесём авторизацию в предусловия:

```
Feature: I as user can create, update and delete contacts on ContactPage
Background:
  Given I am authorized
Scenario: Create a contact with a little info
  Given I am on ContactsPage
  When I click 'Create new contact' button
  And I fill in Organization field
  And I fill in LastName field
  And I fill in FirstName field
  And I fill in Position field
  And I click 'Submit and close' button
  Then I see pop-up with success message
  And I see a table of contacts
```

Мы можем добавить сюда примеры того, как будем заполнять поля. Делается это для параметризации (аналогично `@ParametrizedTest` в JUnit 5), например:

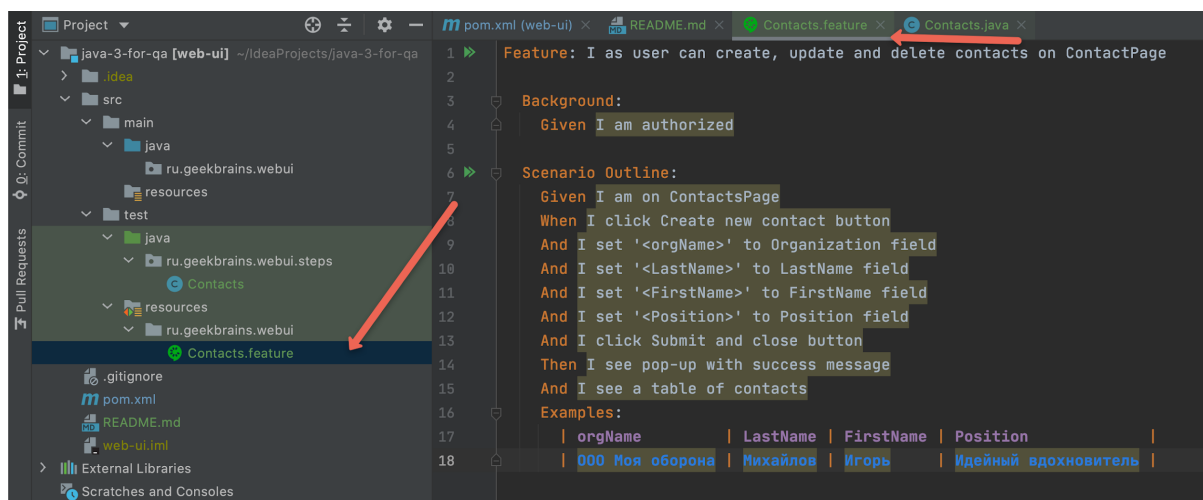
```
Background:
  Given I am authorized

Scenario Outline:
  Given I am on ContactsPage
  When I click Create new contact button
  And I set '<orgName>' to Organization field
  And I set '<LastName>' to LastName field
  And I set '<FirstName>' to FirstName field
  And I set '<Position>' to Position field
  And I click Submit and close button
  Then I see pop-up with success message
  And I see a table of contacts
Examples:
  | Organization | LastName | FirstName | Position |
  | 000 Моя оборона | Михайлов | Игорь | Идеальный вдохновитель |
```

Каждая строка нашей таблицы `Examples` — это отдельный тест с конкретными значениями. Чтобы параметры из таблицы вставлялись в тест, надо поместить их в угловые скобки. Если тип нашего

параметра — строка, то желательно добавить вокруг угловых скобок одинарные кавычки, двойные кавычки параметризуют конкретный шаг значением в кавычках.

В итоге наш проект и feature-файл выглядит следующим образом:

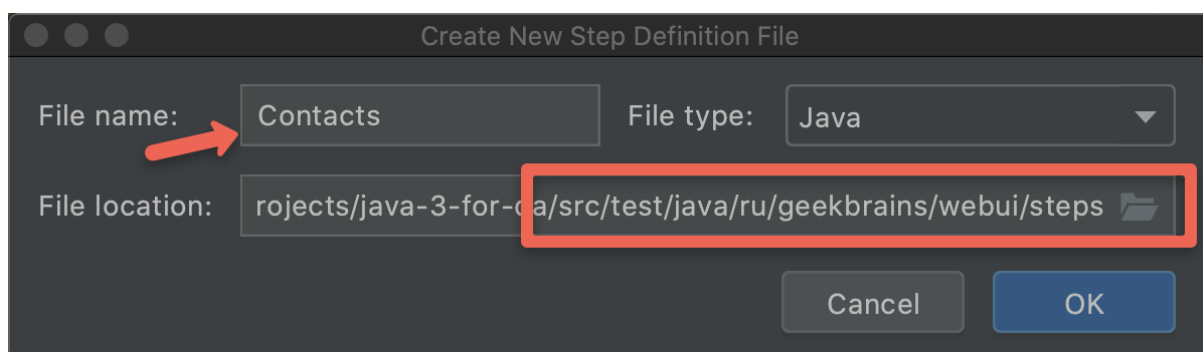


В feature-файле есть слово Feature — оно не фигурирует в сценарии, и для него не надо писать код. Однако это слово используется для словесного описания того, что делает та или иная тестируемая функциональность.

Реализация шагов сценария

Жёлтым в IDEA подсвечиваются те шаги, для которых ещё нет имплементации. Создадим её.

Нажимаем alt+Enter на любом жёлтом поле и выбираем Create all step definitions. В появившемся диалоговом окне выбираем местоположение шагов, например, src/test/java/<название основного пакета>/steps/Contacts.java.



После нажатия кнопки OK в указанном пакете появится файл Contacts с шаблонами реализации шагов:

```

public class Contacts {
    @Given("I am authorized")
    public void iAmAuthorized() {
    }

    @Given("I am on ContactsPage")
    public void iAmOnContactsPage() {
    }

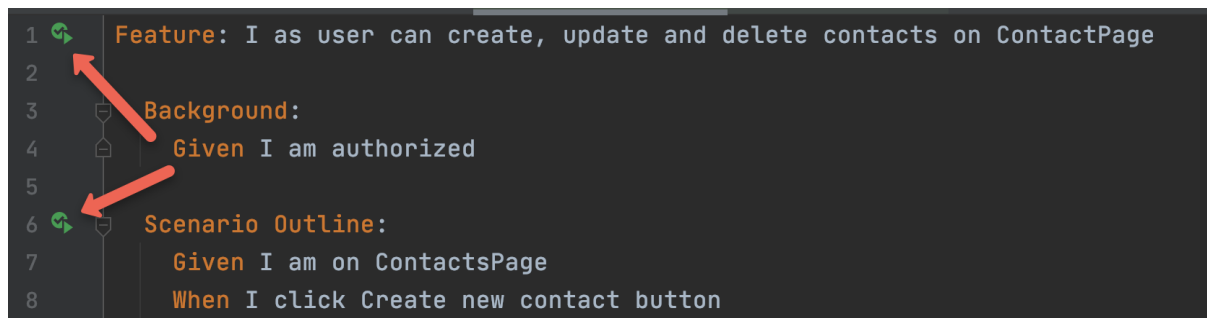
    @When("I click Create new contact button")
    public void iClickCreateNewContactButton() {
    }

    @And("I set {string} to Organization field")
    public void iSetOrgNameToOrganizationField() {
    }
}

```

Напишем имплементацию к каждому шагу. Здесь действуют все паттерны проектирования, которые мы использовали на прошлых уроках — Page Objects всё те же. В тех шагах, где требуются параметры, их надо вписать аргументами в функцию, например, метод `public void iSetOrgNameToOrganizationField(String orgName)` должен принимать аргумент `String` — название организации.

Сценарий запускается в IDE прямо из feature-файла:



Но лучше создать раннер, который будет запускать все сценарии во всех feature-файлах:

```

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

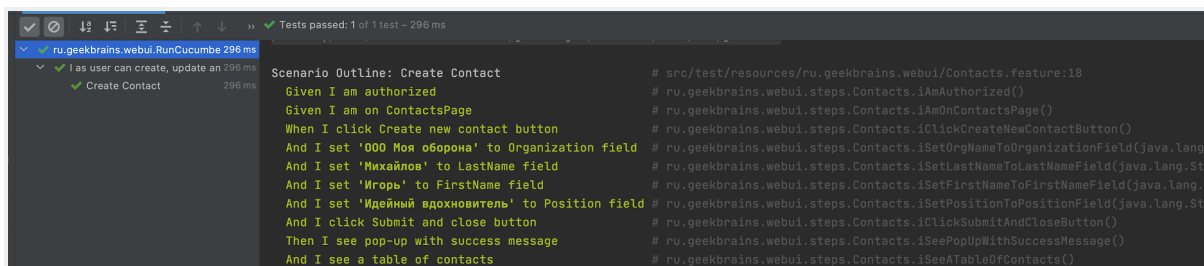
@RunWith(Cucumber.class)
@CucumberOptions(features = {"src/test/resources"},
    plugin = {"pretty"},
    publish = true)
public class RunCucumberTest {

```

```
}
```

Аннотация `CucumberOptions` позволяет настраивать раннер;

- указать, где располагаются feature-файлы (`features={}`);
- указать публикацию cucumber-отчёта (`publish=true`);
- отобразить различные плагины — например, `pretty` позволяет получать в консоли красочные результаты:

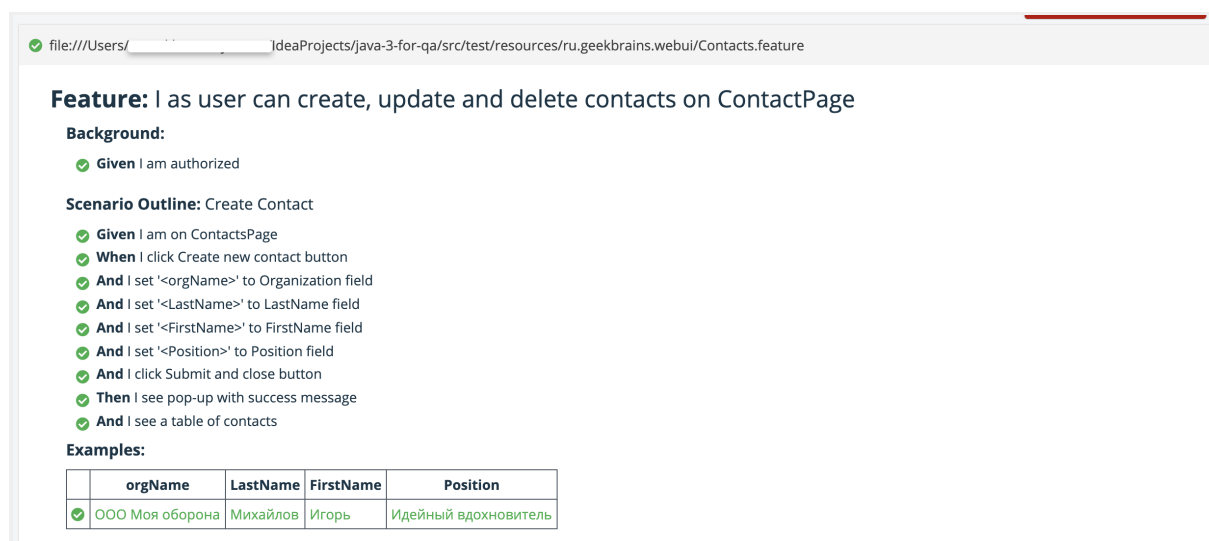


```
Scenario Outline: Create Contact
Given I am authorized
When I click Create new contact button
And I set '000 Моя оборона' to Organization field
And I set 'Михайлов' to LastName field
And I set 'Игорь' to FirstName field
And I set 'Идейный вдохновитель' to Position field
And I click Submit and close button
Then I see pop-up with success message
And I see a table of contacts
```

Используется также плагин `"html:target/cucumber-reports/report.html"`, который сохраняет локальный отчёт в формате cucumber. Отчёты также можно получить в форматах XML и JSON, вызов сразу всех отчётов будет выглядеть следующим образом:

```
plugin = { "pretty", "json:target/cucumber-reports/Cucumber.json",
"junit:target/cucumber-reports/Cucumber.xml",
"html:target/cucumber-reports/Cucumber.html" },
```

Cucumber-отчёт выглядит следующим образом:



Feature: I as user can create, update and delete contacts on ContactPage

Background:

- ✓ Given I am authorized

Scenario Outline: Create Contact

- ✓ Given I am on ContactsPage
- ✓ When I click Create new contact button
- ✓ And I set '<orgName>' to Organization field
- ✓ And I set '<LastName>' to LastName field
- ✓ And I set '<FirstName>' to FirstName field
- ✓ And I set '<Position>' to Position field
- ✓ And I click Submit and close button
- ✓ Then I see pop-up with success message
- ✓ And I see a table of contacts

Examples:

	orgName	LastName	FirstName	Position
✓	000 Моя оборона	Михайлов	Игорь	Идейный вдохновитель

Разумеется, Allure-отчёты настраиваются и для BDD-сценариев, подробнее — [здесь](#).

Selenide

Теперь воспользуемся Selenide для реализации наших шагов.

Selenide — это отечественный фреймворк, запущенный Андреем Солнцевым. Этот фреймворк получил популярность во всём мире. В основе Selenide лежит Selenium WebDriver, и если функциональности фреймворка не хватает, всегда можно воспользоваться WD. Официальная документация лежит [здесь](#). И ещё одна полезная [ссылка](#) с примерами реализованных проектов на Selenide.

Согласно официальной документации, селенид значительно упрощает жизнь — нам больше не надо заботиться о правильной версии драйвера и лишнем коде шагов. Достаточно сделать три шага:

1. открыть(страницу).
2. \$(элемент).совершитьДействие().
3. \$(элемент).проверитьУсловие().

```
open("/login");
$("#submit").click();
$(".message").shouldHave(text("Привет"));
```

Метод \$

Значок \$(By) позволяет нам получить обертку SelenideElement, который открывает больше действий, чем WebElement. Если надо вернуться к WebElement, например, для использования Actions, то сделать это очень просто:

```
WebElement element = selenideElement.getWrappedElement()
```

Наш метод авторизации будет выглядеть примерно так:

```
public class LoginPage {
    private SelenideElement usernameInput = $(By.name("_username"));
    private SelenideElement passwordInput = $(By.name("_password"));
    private SelenideElement submitButton = $(By.id("_submit"));
    private SelenideElement loginHeader = $(By.tagName("h2"));

    public HomePage authorize(){
        open(BASE_URL+LOGIN_PATH);
        usernameInput.setValue(STUDENT_LOGIN);
        passwordInput.setValue(STUDENT_PASSWORD);
        submitButton.click();
        loginHeader.should(disappear);
    }
}
```

```
        return page (HomePage.class) ;
    }
}
```

Проверки в Selenide

В нашем случае проверки не имеют ключевого слова `assert` в угоду лучшей читаемости:

```
selenideElement.should(disappear) ;
```

Кроме `disappear`, есть и другие условия. Полный список — [здесь](#).

Выше говорилось, что для использования Actions надо «развернуть» обернутый элемент. На примере открытия подменю вызов Actions-методов будет выглядеть в Selenide следующим образом:

```
public class HomePage {

    private SelenideElement contractorsMenu =
$ (By.xpath ("//a/span[text()='Контрагенты']"));
    private SelenideElement contactsSubMenu =
$ (By.xpath ("//span[contains(., 'Контактные лица')]"));

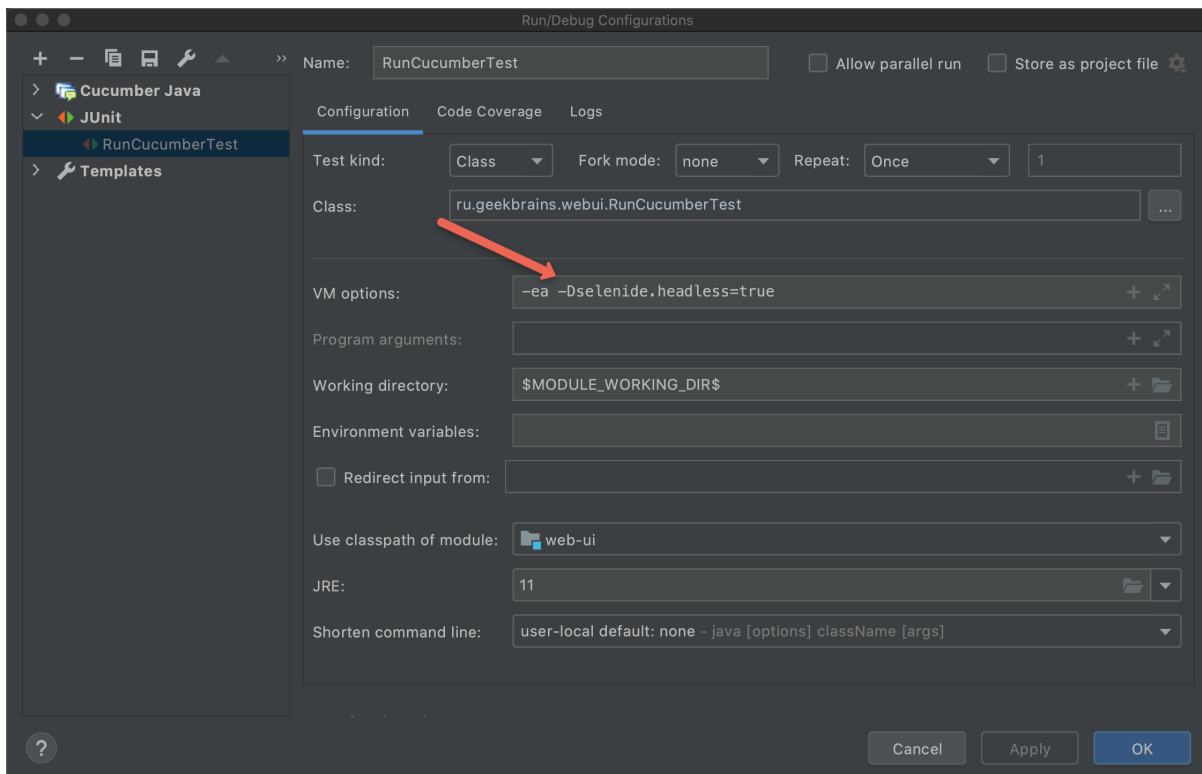
    public ContractorsPage goToContractorsPage () {
        Selenide.actions ()
            .moveToElement (contractorsMenu.getWrappedElement ())
            .build()
            .perform ();
        contactsSubMenu.click ();
        return page (ContractorsPage.class) ;
    }
}
```

Опции для настройки Selenide

Selenide также имеет много опций для настройки. Они есть в классе `Configuration`.

Например, можно установить флаг `Configuration.headless=true`, чтобы тесты прогонялись в `headless`-режиме (подробнее — [здесь](#)).

Можно передать конфигурационные параметры, как `system properties`, чтобы использовать в задачах CI (continuous integration): `Dselenide.baseUrl=http://staging-server.com/start`.



Работа с ожиданиями

Ассерты, которые мы обсуждали выше, например, `should()`, `shouldBe()` и так далее, — это новая версия явных ожиданий, встречавшиеся нам в Selenium WD. Теперь такие конструкции:

```
element = (new WebDriverWait(driver, <timeOutForElement>))
.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(<cssSelector>)));
```

заменяются на такие:

```
element = $(<cssSelector>).shouldBe(visible);
```

Используя ожидания в Selenide, можно дождаться загрузки страницы, например, выполнив двойную проверку:

Сначала ожидаем, что элемент со страницы 1 исчезнет:

```
$(element1).should(disappear);
```

Затем, что элемент 2 появится на новой странице:

```
$(cssSelector).shouldBe(visible);
```

Больше информации по ожиданиям — [здесь](#).

Используемые источники

1. Официальная документация [Selenide](#).
2. Официальная документация [JUnit 5](#).
3. Официальная документация проекта [Cucumber](#).
4. [Josdem's](#) blog.