# Лекция 2. Знакомство с Selenium WebDriver

## TestNG: основные понятия

TestNG – фреймворк для написания автотестов на языке Java. В TestNG есть много удобных вещей, которых нет в JUnit. “NG” означает “Next Generation”.

Зачем нам нужен TestNG?

1. TestNG может генерировать отчеты, основанные на результатах запуска Selenium тестов.

* WebDriver не имеет внутренний механизм для генерации отчетов.
* TestNG может генерировать отчеты в удобном для чтения формате.

1. TestNG упрощает код тестов

* Последовательность действий регулируется аннотациями с «говорящими» названиями, такими как @BeforeMethod, @Test, @AfterTest и т.п.

Каждый тест - это отдельный метод внутри класса. По правилам TestNg он должен быть public void. Жестких правил по именованию тестовых методов нет.

### Структура TestNG файла

TestNG файл является основным местом настройки всех тестов, групп, сьютов. Он представляет из себя xml файл, в котором описано, что и как включать в тесты, а так же расписана иерархия Ваших тестов.

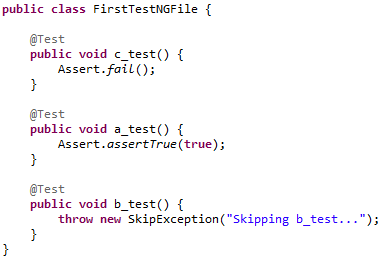
Рассмотрим, что же значит каждый из тэгов.

* **suite.** С помощью данного тэга мы объединяем ряд тестов в сьюты, чтобы можно было разделить все наши тесты, например, по функциональности.
* **test.** Внутри suite может быть один или несколько разделов-тэгов test. Каждый из них - это не тест, который мы привыкли ассоциировать с тестовым методом. Это нечто более глобальное, можно назвать это фичей, то есть, например: Тест Authorisation - тест, включающий в себя все тесты (методы), тестирующие авторизацию где-то на сервере.
* **groups**. Внутри данного тэга мы можем перечислить группы, которые нужно запустить, или, соответственно, исключить при запуске теста.
* **classes.** Тут мы можем перечислить в виде тэгов **class** классы с тестами, которые будут запущены.
* **methods**. Внутри этого тэга можно перечислить точечные методы для запуска. Если тэг methods отсутствует - будут запущены все методы класса, аннотированные @Test.

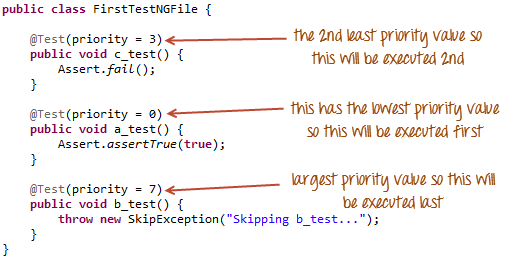
### Аннотации TestNG

Аннотация @Test используется для того, чтобы указать, что текущий метод является тест кейсом.

В одном классе может быть несколько тестовых методов. По умолчанию, тесты запускаются в алфавитном порядке.



Для изменения порядка тестов, используется параметр priority. TestNG начнет запуск с теста с минимальным значением параметра priority. Значения параметра priority не обязательно должны быть последовательны.



Какие еще бывают аннотации в TestNG:

* Аннотации @BeforeSuite, @AfterSuite обозначают методы, которые исполняются единожды до/после исполнения всех тестов. Здесь удобно располагать какие-либо тяжелые настройки общие для всех тестов, например, здесь можно создать пул соединений с базой данных.
* Аннотации @BeforeTest, @AfterTest обозначают методы, которые исполняются единожды до/после исполнения теста (тот, который включает в себя тестовые классы, не путать с тестовыми методами).
* Аннотации @BeforeClass, @AfterClass обозначают методы, которые исполняются единожды до/после исполнения всех тестов в классе, идентичны предыдущим, но применимы к тест-классам. Аннотации @BeforeMethod, @AfterMethod обозначают методы, которые исполняются каждый раз до/после исполнения тестового метода.
* Аннотации @BeforeGroups, @AfterGroups обозначает методы, которые исполняются до/после первого/последнего теста принадлежащего к заданным группам.

У всех этих аннотаций есть следующие параметры:

* enabled — можно временно отключить, установив значение в false
* groups — обозначает, для каких групп будет исполнен
* timeOut — время, после которого метод «свалится» и потянет за собой все зависимые от него тесты
* description — название, используемое в отчете
* dependsOnMethods — методы, от которых зависит, сначала будут выполнены они, а затем данный метод
* dependsOnGroups — группы, от которых зависит
* alwaysRun — если установить в true, будет вызываться всегда независимо от того, к каким группам принадлежит, не применим к @BeforeGroups, @AfterGroups

### Проверка результата теста

С помощью класса Assert библиотеки TestNG мы проверяем, можно ли считать тест пройденным.

**Assert** – это специальная конструкция, позволяющая проверять предположения о значениях произвольных данных в произвольном месте программы.

Тест считается успешно пройденным, только если в процессе не было выброшено никаких исключений.

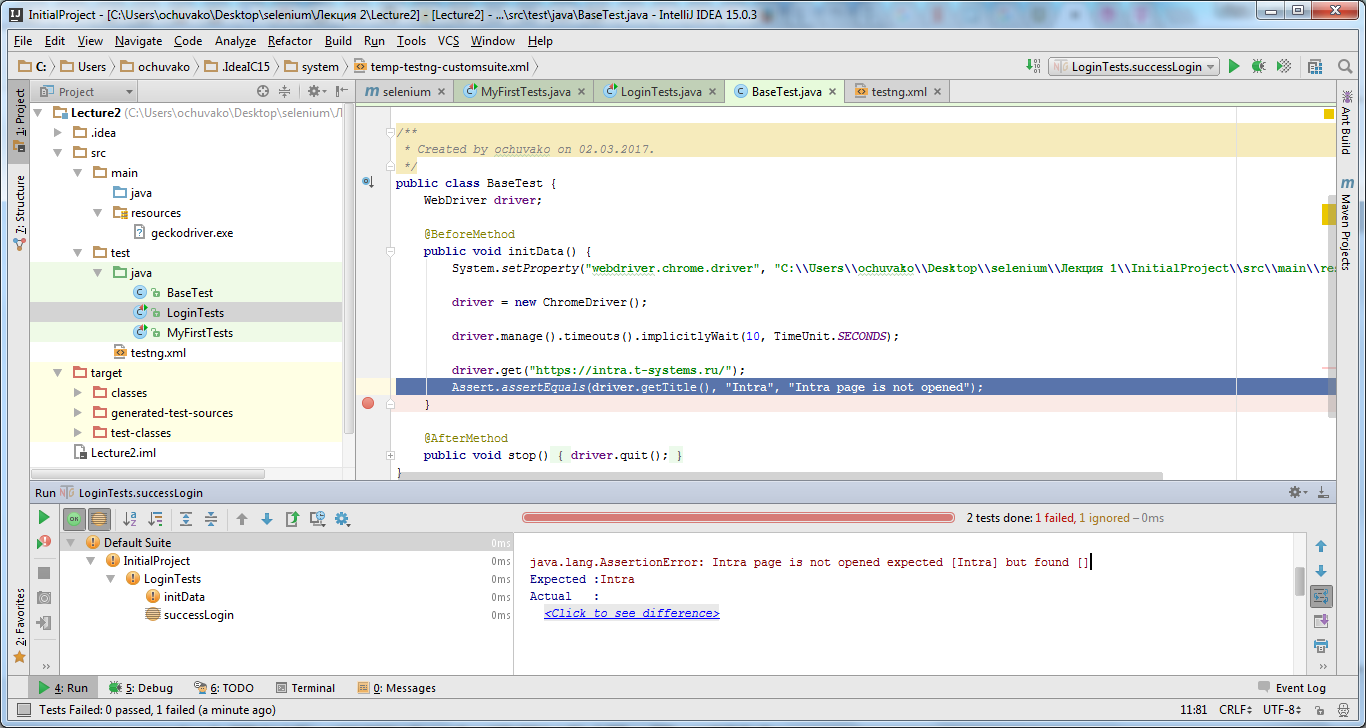
Наиболее часто используемые виды Assert вставок:

* Assert.assertTrue()
* Assert.assertFalse()
* Assert.assertEquals()
* Assert.assertNotEquals()
* Assert.assertNull()
* Assert.assertNotNull()

В каждый из методов проверки в качестве последнего параметра можно передать текст сообщения, которое отобразится в случае, если тест упадет, например,

Assert.*assertEquals*(**driver**.getTitle(), **"Intra"**, **"Intra page is not opened"**);

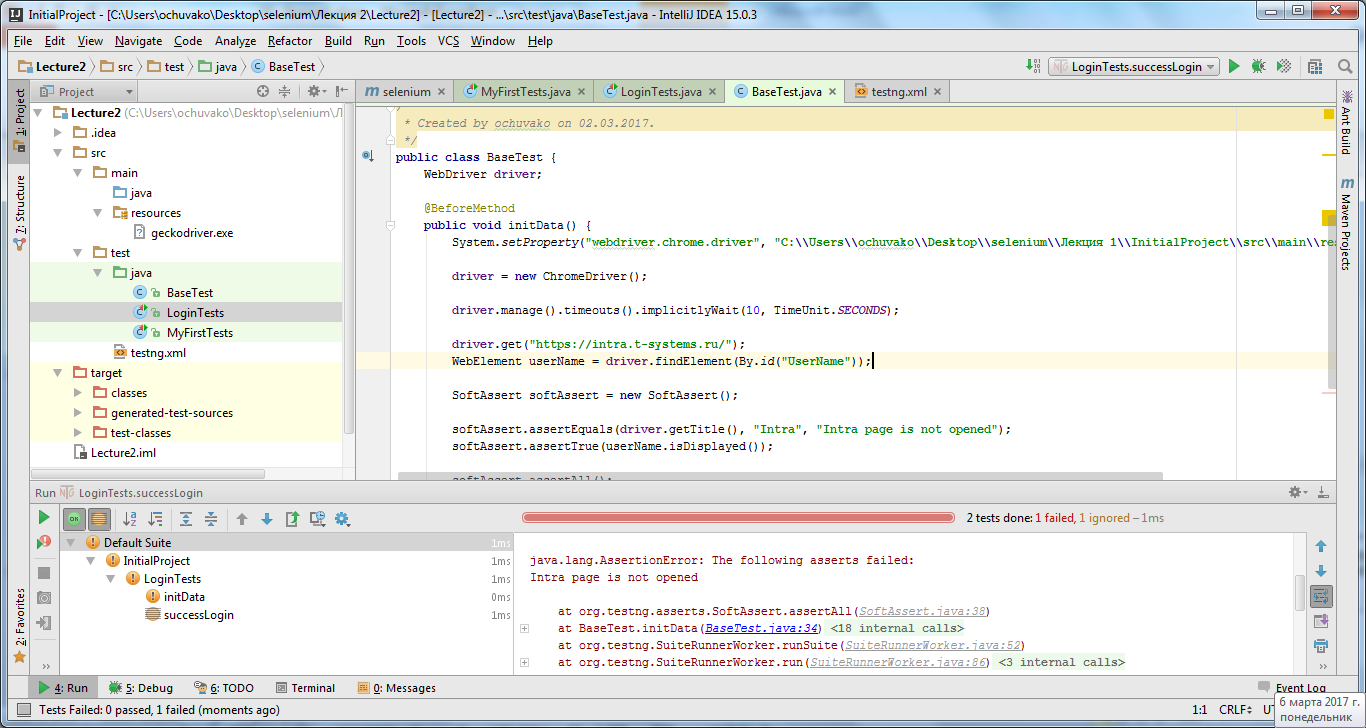
В случае ошибки, в консоли мы увидим Assertion error:



Существует более мягкая реализация ассерта, которая так и называется – SoftAssert. Например:

SoftAssert softAssert = **new** SoftAssert();  
  
softAssert.assertEquals(**driver**.getTitle(), **"Intra"**, **"Intra page is not opened"**);  
softAssert.assertTrue(userName.isDisplayed());  
  
softAssert.assertAll();

Когда assertion падает, тест не останавливается, и только в консоль записывается сообщение об ошибке. Метод assertAll() вызовет исключение, если хотя бы один из assertion упал. В случае ошибке мы увидим сообщение:



## Поиск элементов на странице

Для поиска элементов на странице реализовано 2 метода:

* WebElement findElement(By by)  — находит первый, удовлетворяющий условиям поиска, элемент на странице.

WebElement element = **driver**.findElement(byLocator);

* List<WebElement> findElements(By by) — находит все элементы в текущем контексте, удовлетворяющие условиям поиска.

List<WebElement> elements = **driver**.findElements(byLocator);

В качестве параметра эти методы принимают локатор – объект типа By, который задает критерий поиска.

Что, если ничего не нашлось?

* findElement - Если элемент не найден сразу, то метод будет вызываться повторно пока не истечет время ожидания. Если же ничего найти не удалось, будет выдано исключение *NoSuchElementException*.
* findElements - При выполнении неявного ожидания, метод вернет список сразу же, как только будет найден хоть один элемент, или вернет пустой список, если время ожидания истекло.

Поэтому не используйте findElement для проверки отсутствия элементов, вместо этого можно использовать метод findElements и проверку на нулевое количество полученных элементов.

## Типы локаторов

Для поиска элементов в Selenium используются локаторы.

**Локатор** – это строка, уникально идентифицирующая элемент страницы.

WebDriver предоставляет несколько способов использования локаторов для поиска элементов:

* **By.id** – в качестве локатора используется атрибут id элемента страницы;
* **By.name** – в качестве локатора используется атрибут name элемента страницы;
* **By.className** – поиск по классу (атрибут class) элемента;
* **By.tagName** – поиск по имени HTML тега;
* **By.xpath** – используется для поиска элемента по XPath выражению;
* **By.cssSelector** – данный тип локаторов основан на описаниях таблиц стилей (CSS);
* **By.linkText** – поиск ссылки с указанным текстом. Текст ссылки должен быть точным совпадением;
* **By.partionalLinkText** – поиск ссылки по части с указанным текстом.

Примеры использования.

<div id="first">

<a class="red\_link">

<div class="label">Первая ссылка</div>

</a>

<a>Вторая ссылка</a>

</div>

* WebElement element = **driver**.findElement(By.*id*(**"first"**));
* **driver**.findElement(By.*tagName*(**"a"**));
* **driver**.findElement(By.*cssSelector*(**"#first"**));
* **driver**.findElement(By.*linkText*(**"Вторая ссылка"**));

## Отладка локаторов

Хороший локатор всегда уникален и находит только один нужный нам элемент. Для отладки локаторов в старых версиях тMozilla Firefox можно использовать Firebug + Firepath.

* **Firebug**

Firebug — отладчик web-приложений, используется как отдельное дополнение для браузера Mozilla Firefox, являющееся консолью, отладчиком, и DOM-инспектором JavaScript, DHTML, CSS.

Среди всех возможностей Firebug полезными для нас будут:

* + Удобный просмотр HTML-кода страницы. Функция Inspect позволяет точно определить местонахождение тега того или иного элемента, просмотреть все «привязанные» к нему свойства и стили.
  + Редактирование HTML и CSS прямо в браузере. Можно изменять атрибуты тегов и значения свойств для того, чтобы пронаблюдать изменения.

Весь функционал Firebug можно изучить тут: <http://firebug.ru/>

* **FirePath**.

FirePath является расширением для Firebug, которое добавляет инструменты для редактирования, проверки и создания выражения XPath, CSS селекторов и селекторов JQuery.

После установки плагина, открываем Firebug и переходим на вкладку FirePath. Ниже увидим строку для ввода локатора. Слева от нее можно выбрать вид локатора который мы проверяем - CSS или XPath. Далее заходим на тестовую страничку и пробуем написать свой селектор. В левой верхней части экрана Firebug - есть кнопка для исследования элемента. Выглядит как стрелка курсора мыши кликающая на прямоугольник.  
Жмем на нее и наводим на нужный элемент на странице. Внизу экрана увидим как Firepath нашел нужную строку кода страницы, на которой описан интересующий нас элемент. В строку для ввода локатора он автоматически сгенерировал селектор, по которому мы можем обратиться к данному элементу. Слева внизу мы видим "1 matching node", что говорит, сколько элементов имеют такой локатор.

Преимущества Firepath:

* + генерируемый xpath элементов строится оптимальным образом, используя id элементов;
  + поддержка работы с различными namespaces;
  + отображение результатов в Firebug, в DOM-модели документа;
  + подсвечивание результатов непосредственно на странице.

В Mozilla Firefox Quantum Firebug и Firepath не поддерживаются. Надо использовать стандартную консоль для отладки локаторов.

В Google Chrome можно использовать стандартную консоль для отладки локаторов.

Для этого нажимаем F12 и переходим на вкладку Console.

Для проверки локаторов можно использовать команды:

* document.getElementById(“”);
* document.getElementsByClassName(“”);
* document.getElementsByTagName(“”);
* document.getElementsByName(“”);
* $x(“”) – поиск элементов по xpath локатору
* $(“”) – поиск первого подходящего элемента по CSS локатору
* $$(“”) – поиск списка элементов по CSS локатору

## Неявные ожидания (Implicit Waits)

Неявное ожидание указывает WebDriver опрашивать DOM в течение конкретного периода времени при поиске элементов, в случае если они не появились сразу. По умолчанию это значение равно 0.

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***);

* findElement ждёт, пока элемент появится
* findElements ждёт, пока **хотя бы один** элемент появится

Неявные ожидания обычно настраиваются сразу после создания экземпляра WebDriver и действуют в течение всей жизни этого экземпляра, хотя переопределить их можно в любой момент.

По истечении заданного в ожидании времени WebDriver бросит **ElementNotFoundException**.

## Действия click и sendKeys

Итак, вы нашли элемент страницы, что дальше?

Есть две базовые операции пользователя – нажатие левой клавиши мыши и ввод какого-либо текста с клавиатуры. Им соответствуют, методы click и sendKeys.

Клик можно выполнить по любому видимому элементу. Он выполняется в центр элемента по-умолчанию.

element.click();

При выполнении метода sendKeys – текст дописывается в конец поля.

input.sendKeys(**"some text"**);

Если sendKeys выполняется для файлового поля ввода, и в качестве параметра передается путь к файлу, то sendKeys прицепляет этот файл. В некоторых браузерах это происходит без открытия диалогового окна.

input.sendKeys(file.getAbsolutePath());

Также sendKeys может отправлять различные сочетания клавиш:

input.sendKeys(Keys.***ARROW\_DOWN***);

## Получение свойств элементов

Для получения текста элемента используется метод **getText():**

String text = webElement.getText();

Особенности:

* Возвращается только видимый текст (невидимые элементы имеют пустой текст)
* Выполняется нормализация – удаление пробелов, т.е. текст вернется таким, как отображается в браузере, несмотря на верстку.

Для получения свойств элементов используется метод **getAttribute().**

String attribute = webElement.getAttribute(**"attr"**);