GeekBrains

Автоматизация тестирования Web UI на Java

Page Object

[Java 11]

На этом уроке

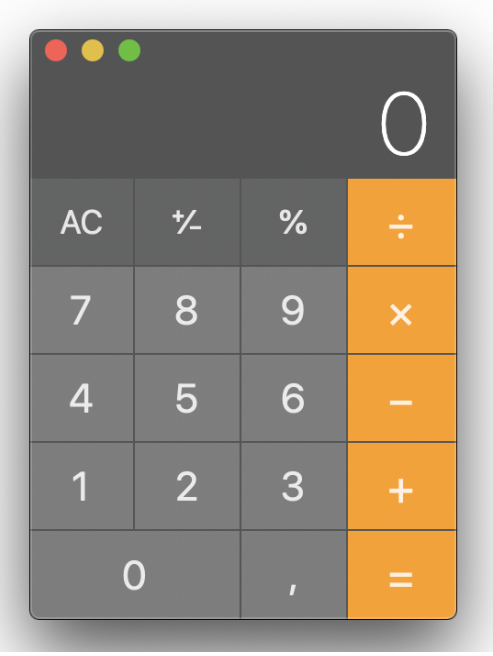
1. Узнаем, что такое паттерн проектирования.
2. Рассмотрим принципы разработки ПО (DRY, KISS).
3. Разберём, как это работает в написании автотестов.
4. Выясним, почему Page Object — самый распространённый паттерн.

Оглавление

Рефакторинг тестов

До сегодняшнего занятия мы писали тесты неоптимальным способом. Настало время исправить этот момент.

Итак, в качестве иллюстрирующего примера представим, что у нас есть веб-страница с функциональностью калькулятора, такая же как в стандартном мобильном или десктопном приложении.



Тест, проверяющий сложение двух целых чисел, выглядел бы примерно так (локаторы — для примера):

```
@Test
public void sumTwoDigitsTest() {
    driver.findElement(By.id("digit_2")).click();
    driver.findElement(By.id("button_plus")).click();
    driver.findElement(By.id("digit_9")).click();
    driver.findElement(By.id("button_equality")).click();
    String result = driver.findElement(By.id("result_textview")).getText();
    Assertions.assertEquals("11", result);
}
```

Шаги теста:

1. Найти кнопку «2» и кликнуть по ней.
2. Найти кнопку «+» и кликнуть по ней.

3. Найти кнопку «9» и кликнуть по ней.
4. Найти кнопку «=» и кликнуть по ней.
5. Найти текстовое поле результата, взять строковое значение текста.
6. Сравнить полученное значение с ожидаемым «11».

Действия несложные. Но чтобы покрыть все сценарии и функции калькулятора, придётся написать порядочное количество тестов — десятки, а то и сотни.

Представим ситуацию, что, выкатывая новую версию страницы, разработчики изменили локаторы элементов. Причины на это могут быть самыми разными. Переписывать придётся все локаторы во всех тестах. Приятного мало. Чтобы грамотно спроектировать тесты, потребуется паттерн Page Object.

Page Object

Суть паттерна проста — следуя принципам объектно-ориентированного программирования, мы представляем страницы или экраны, в случае с мобильными приложениями, нашего веб-приложения как отдельные классы. Класс экрана включает:

- поля — то, что экран знает;
- методы — то, что экран умеет.

Таким образом, в случае незначительных изменений в UI, не влияющих на общую логику прохождения теста, изменять тестовый код не придётся. Достаточно обновить локаторы в классе страницы.

Напишем класс экрана калькулятора.

Локаторы элементов — это поля класса. Но так как конкретно калькулятор имеет весьма ограниченный набор цифр, обозначим все кнопки во вложенном классе-перечислении. В качестве методов — атомарные шаги, на которые можно разложить тест-кейс.

```
public class CalculatorPage {

   private WebDriver driver;

   private final String button_1_id = "button_1";
   private final String button_2_id = "button_2";
   private final String button_3_id = "button_3";
   //...
   private final String plusButtonId = "button_plus";
   private final String equalsButtonId = "button_equality";
   //...
   private final String resultTextViewId = "result_textview";
```

```
    public CalculatorPage(WebDriver driver) {
        this.driver = driver;
    }

    public void clickSum() {
        driver.findElement(By.id(plusButtonId)).click();
    }

    public void clickEquals() {
        driver.findElement(By.id(equalsButtonId)).click();
    }

    public String getResult() {
        return driver.findElement(By.id(resultTextViewId)).getText();
    }

    public void clickDigit1() {
        driver.findElement(By.id(button_1_id)).click();
    }

    public void clickDigit2() {
        driver.findElement(By.id(button_2_id)).click();
    }

    // Остальные методы...
}
```

Для взаимодействия с API веб-драйвера классу требуется доступ к экземпляру класса Webdriver. По этой причине драйвер передаётся в конструкторе.

Код теста выглядит следующим образом:

```
@Test
public void sumTwoDigitsWithPageObjectTest() {
    CalculatorPage calculatorPage = new CalculatorPage(driver);
    calculatorPage.clickDigit2();
    calculatorPage.clickSum();
    calculatorPage.clickDigit9();
    calculatorPage.clickEquals();
    String result = calculatorPage.getResult();
    Assertions.assertEquals("11", result);
}
```

Улучшения заметны невооружённым взглядом: тестовый код практически избавился от специфики фреймворка и демонстрирует преимущественно бизнес-логику.

Что ещё можно вынести в код класса CalculatorPage?

Текучий интерфейс

В разработке программного обеспечения есть такое понятие, как текучий интерфейс (fluent interface). Суть текучести заключается в упрощении множественного вызова одного и того же объекта путём возвращения контекста вызова следующему звену.

В коде теста объект calculatorPage вызвался несколько раз. Это говорит о том, что надо улучшить наш код, сделав его fluent.

Для этого изменим типы возвращаемых значений в методах класса страницы с void на CalculatorPage, не забывая завершать тела этих функций возвращением текущего объекта.

Пример:

```
public CalculatorPage clickDigit9() {
    driver.findElement(By.id(button_1_id)).click();
    return this;
}
```

Теперь код теста будет выглядеть так:

```
@Test
public void sumTwoDigitsWithPageObjectTest() {
    String result = new CalculatorPage(driver)
        .clickDigit2()
        .clickSum()
        .clickDigit9()
        .clickEquals()
        .getResult();
    Assertions.assertEquals("11", result);
}
```

Логика стала чище: получение результата — процесс выполнение некоторой цепочки действий, основанной на бизнес-логике.

PageFactory

Пакет org.openqa.selenium.support содержит класс PageFactory, предоставляющий аннотации для удобного создания Page Objects.

Удобство заключается в объявлении всех веб-элементов страницы, именно как WebElement, а не через селектор, полями класса. Инициализация происходит «лениво» (lazy initialization), т. е. прямо в момент обращения к элементу. Это даёт возможность не беспокоиться об ElementNotFoundException, если на момент инициализации класса страницы какие-то элементы не находятся на экране.

Рассмотрим применение PageFactory на примере класса страницы CalculationScreen.

```
public class CalculatorPage {
    private WebDriver driver;

    @FindBy(id = "button_1")
    private WebElement button_1;

    @FindBy(id = "button_2")
    private WebElement button_2;

    @FindBy(id = "button_3")
    private WebElement button_3;
    //...
    @FindBy(id = "button_plus")
    private WebElement plusButton;

    @FindBy(id = "button_equality")
    private WebElement equalsButton;
    //...
    @FindBy(id = "result_textview")
    private WebElement resultTextView;

    public CalculatorPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }
    public CalculatorPage clickSum() {
        plusButton.click();
        return this;
    }
    public CalculatorPage clickEquals() {
        equalsButton.click();
        return this;
    }
    public String getResult() {
        return resultTextView.getText();
    }
    public CalculatorPage clickDigit9() {
        button_1.click();
        return this;
    }
    public CalculatorPage clickDigit2() {
        button_2.click();
        return this;
    }
    // Остальные методы...
}
```

Начнём с конца. В итоговых методах, где происходит взаимодействие с элементами страницы,

больше нет строчки driver.findElement(By.id(some_id)) , так как PageFactory берёт это на себя. Мы работаем с интерфейсом WebElement.

Далее — конструктор. Вызов статического метода PageFactory.initElements(driver, this) обеспечивает инициализацию аннотированных выше элементов.

И, наконец, элементы. Достаточно объявить поле с типом WebElement и добавить аннотацию @FindBy, добавив стратегию поиска. Нам доступны id, name, className, css, tagName, linkText, xpath.

Если потребуется список элементов, как при driver.findElements, достаточно объявить поле как List<WebElement>.

Практическое задание

1. Проведите рефакторинг в соответствии с PageObject.

Дополнительная информация

1. Статья Page Object Model and Page Factory in Selenium.

Используемые источники

1. Yujun Liang, Alex Collins. Selenium WebDriver: From Foundations to Framework.
2. Unmesh Gundecha. Selenium Testing Tools Cookbook.
3. Официальная документация.