нетология

BDD

(BEHAVIOUR DRIVEN DEVELOPMENT)

АРТЕМ РОМАНОВ

АРТЕМ РОМАНОВ

Инженер по обеспечению качества

romanov-artyom

@Artem_Telegram

Романов Артём

ПЛАН ЗАНЯТИЯ

1. Задача
2. Page Object
3. BDD
4. Akita
5. Итоги

ЗАДАЧА

ЗАДАЧА

Наша задача — протестировать:

- операцию перевода денег с карты одного клиента на карту другого клиента через Интернет Банк
- операцию перевода денег с одной карты на другую одного и того же клиента через Интернет Банк

ДУБЛИРОВАНИЕ КОДА

На прошлой лекции мы рассмотрели ключевые вопросы и увидели то, что в двух тестах будут повторяться одни и те же шаги: авторизация и ввод кода из SMS.

Самый плохой вариант решения — это `Ctrl + C`, `Ctrl + V`.

КЛЮЧЕВОЕ

Обратите внимание, что не нужно "притягивать проблему за уши" — если у вас всего 1-3 небольших теста (как у нас было до этого), то особо больших проблем у вас нет и надстраивать новые уровни абстракции и подходы не нужно, даже если так "принято" в индустрии.

Потому что один из ключевых аспектов нашей работы — это время.

Решая проблемы, которых пока не существует (а, возможно, и не появится), вы как раз это время тратите.

PAGE OBJECTS

PAGE OBJECT

`Page Object` — сокрытие внутреннего устройства определённой страницы или её виджетов за логическим интерфейсом взаимодействия.

Что получаем:

- логика теста и логика поиска элементов разделены
- чёткий интерфейс того, что можно делать с конкретной страницей или виджетом
- возможность повторного использования кода

СТРАНИЦА VS WIDGET

В современном мире веб-приложений термин "страница" достаточно условный.

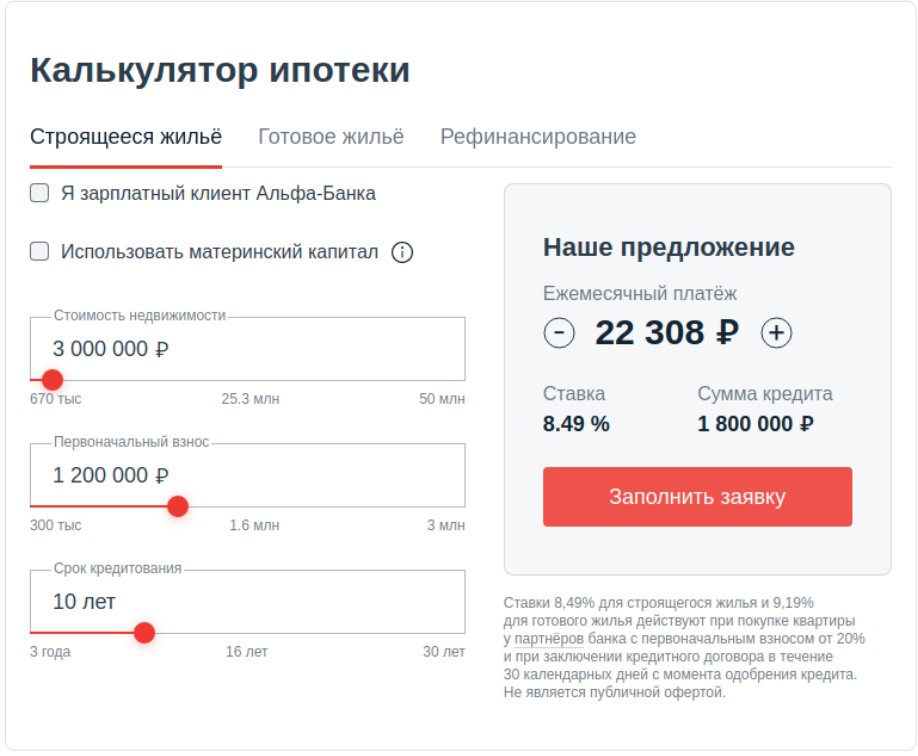
Сам термин `Page Object` не требует, чтобы вы описывали всю страницу целиком.

Но `Page Object` исторически привязалось именно к "странице" как к большому объекту.

СТРАНИЦА VS WIDGET

Но в то же время есть достаточно сложные виджеты, встречающиеся на разных страницах.

Чтобы не вступать в войны "определений" мы будем называть их не `Page Object`, а `Page Element`:



PAGE OBJECT

```
class LoginPage {
  public void login(AuthInfo info) {
    // здесь инкапсулирована вся логика работы со страницей логина
  }
}
```

```
// в тестах, в которых необходим логин:
loginPage.login(info);

// вместо того, чтобы каждый раз вызывать:
$('[data-id=login]').setValue(info.getLogin());
$('[data-id=password]').setValue(info.getPassword());
$('[data-action=login]').click();
// эта логика будет зашита в самом методе login
```

Попробуем реализовать это на живом проекте.

MASTER PASSWORD

Разработчики передали нам специальную сборку сервиса Интернет Банка для тестирования.

Но как вы знаете, для совершения входа в Интернет Банк нужно ввести пароль, присланный по SMS/PUSH.

Разработчики предложили следующее решение: сделать **Master Password**, который можно использовать вместо реально отправленного, и он будет подходить для всех подобных сценариев (они сделали специально в коде проверку на это, а в продакшн сборке они это уберут).

Вопрос к аудитории: как вы думаете, насколько хорошо это решение?

MASTER PASSWORD

В большинстве случаев это очень плохое решение, т.к.:

- есть очень серьёзный риск, что код с Master Password так и пойдёт в продакшн (и такое много раз было)
- вы тестируете "специальную" сборку, а не то, что пойдёт в продакшн (так тестируете ли вы что-то?)

Но ничего не поделаешь, разработчики не вняли нашим предостережениям и сказали: *"нормально, мы всё контролируем"*.

ТЕСТОВЫЕ ДАННЫЕ

Кроме того, нам передали фиксированный набор данных (в той же самой "специальной" тестовой сборке), которые мы можем использовать для тестирования.

ДОПУЩЕНИЯ

Несмотря на то, что мы вам говорим "не притягивать проблему за уши", мы сразу будем рассматривать финальный вариант кода, чтобы уложиться в масштабы лекции.

ВАРИАНТ 1:

```
public class LoginPageV1 {
  public VerificationPage validLogin(DataHelper.AuthInfo info) {
    $("[data-test-id=login] input").setValue(info.getLogin());
    $("[data-test-id=password] input").setValue(info.getPassword());
    $("[data-test-id=action-login]").click();
    return new VerificationPage();
  }
}
```

ВАРИАНТ 2:

```
public class LoginPageV2 {
  private SelenideElement loginField = $("[data-test-id=login] input");
  private SelenideElement passwordField = $("[data-test-id=password] input");
  private SelenideElement loginButton = $("[data-test-id=action-login]");

  public VerificationPage validLogin(DataHelper.AuthInfo info) {
    loginField.setValue(info.getLogin());
    passwordField.setValue(info.getPassword());
    loginButton.click();
    return new VerificationPage();
  }
}
```

ВАРИАНТ 3:

```
1  public class LoginPageV3 {
2    @FindBy(css = "[data-test-id=login] input")
3    private SelenideElement loginField;
4    @FindBy(css = "[data-test-id=password] input")
5    private SelenideElement passwordField;
6    @FindBy(css = "[data-test-id=action-login]")
7    private SelenideElement loginButton;
8
9    public VerificationPage validLogin(DataHelper.AuthInfo info) {
10     loginField.setValue(info.getLogin());
11     passwordField.setValue(info.getPassword());
12     loginButton.click();
13     return page(VerificationPage.class);
14   }
15 }
```

РАЗБИРАЕМСЯ

Для всех вариантов мы создаём класс, **инкапсулирующий в себе всю логику внутреннего устройства**, и, ключевое, **доменные методы для работы со страницей**.

Доменные методы — это методы предметной области. Т.е. когда мы (как нормальный человек) работаем с Интернет Банком, мы говорим "я залогинился".

Мы не говорим "я заполнил поле логин своим логином и пароль своим паролем, а потом я нажал на кнопке логин" - это уже детали реализации.

РАЗБИРАЕМСЯ

Q: Почему варианта три?

A: На самом деле, их гораздо больше, но ключевых подхода всего два:

1. Мы не храним информацию об элементах страницы и скрываем полностью эту логику в методах
2. Мы выносим в поля **ключевые** элементы страницы, чтобы затем работать с ними

У каждого подхода есть свои плюсы и минусы, кому-то нравится первый, кому-то второй. Но многие используют именно второй подход, хотя авторы Selenide ратуют за первый.

Для вас же ключевое — уметь использовать оба подхода.

`FindBy`

Аннотация `FindBy` используется для поиска элементов на странице и привязке их к полям.

Вместо неё вы можете использовать обычные `$` и `$$`.

Самое главное: не нужно в поля Page Object выносить вообще все элементы страницы!

`PageFactory`

Q: Что такое `page(VerificationPage.class)`?

A: Это реализация паттерна `PageFactory` — наличие специального объекта или метода, занимающегося созданием и инциализацией ваших объектов из классов.

Вообще говоря, в Selenide вы можете обойтись обычным `new VefiricationPage`, но вы иногда можете встретить такой код — мы его оставили для того, чтобы вы понимали, что происходит.

Ключевое: аннотация `FindBy` будет работать только если ваш `PageObject` был создан через `PageFactory`! В противном случае вы получите всеми любимый NPE (`NullPointerException`), т.к. никто не будет заниматься аннотациями, указанными в вашем классе.

`FindBy` И `PageFactory`

Q: Так зачем они нужны, если мы можем обойтись без них?

A: Некоторые фреймворки их используют (например, Akita, которую мы будем рассматривать чуть позже) в собственных целях, поэтому вы обязаны о них знать.

`loginValid`

Q: Зачем мы выделяем `loginValid`, разве этому методу не всё равно?

A: Затем, что в случае успеха мы будем возвращать новую страницу, на которой нужно ввести код подтверждения, а в случае неуспеха — виджет ошибки, т.е. перехода на другую страницу не будет. Поэтому хорошая практика - разделять эти моменты.

```java
public class DataHelper {
  private DataHelper() {}

  @Value
  public static class AuthInfo {
    private String login;
    private String password;
  }

  public static AuthInfo getAuthInfo() {
    return new AuthInfo("vasya", "qwerty123");
  }

  public static AuthInfo getOtherAuthInfo(AuthInfo original) {
    return new AuthInfo("petya", "123qwerty");
  }

  @Value
  public static class VerificationCode {
    private String code;
  }

  public static VerificationCode getVerificationCodeFor(AuthInfo authInfo) {
    return new VerificationCode("12345");
  }
}
```

Как мы уже говорили на прошлой лекции, вынесение данных в отдельный класс генератор, позволит нам затем дописать сюда необходимую логику: использовать Faker или обращаться к СУБД, другим сервисам и т.д.

```
@Test
void shouldTransferMoneyBetweenOwnCardsV1() {
  open("http://localhost:9999");
  val loginPage = new LoginPageV1();
  // можно заменить на val loginPage = open("http://localhost:9999", LoginPageV1.class);
  val authInfo = DataHelper.getAuthInfo();
  val verificationPage = loginPage.validLogin(authInfo);
  val verificationCode = DataHelper.getVerificationCodeFor(authInfo);
  verificationPage.validVerify(verificationCode);
  ...
}
@Test
void shouldTransferMoneyBetweenOwnCardsV2() {
  open("http://localhost:9999");
  val loginPage = new LoginPageV2();
  // можно заменить на val loginPage = open("http://localhost:9999", LoginPageV2.class);
  val authInfo = DataHelper.getAuthInfo();
  val verificationPage = loginPage.validLogin(authInfo);
  val verificationCode = DataHelper.getVerificationCodeFor(authInfo);
  verificationPage.validVerify(verificationCode);
  ...
}
@Test
void shouldTransferMoneyBetweenOwnCardsV3() {
  val loginPage = open("http://localhost:9999", LoginPageV3.class);
  // но здесь обратное не сработает — FindBy только с PageFactory
  val authInfo = DataHelper.getAuthInfo();
  val verificationPage = loginPage.validLogin(authInfo);
  val verificationCode = DataHelper.getVerificationCodeFor(authInfo);
  verificationPage.validVerify(verificationCode);
  ...
}
```

`val` и `@Value`

`val` конструкция Lombok'а, которая позволяет вам не писать тип для переменной.

`@Value` возможность с помощью аннотации создавать те самые Value Objects, которые мы обсуждали на прошлой лекции.

ВОПРОСЫ

Q: Но как проверить, что действие завершилось? Например, если тот же самый логин может быть долгим?

A: Есть несколько подходов:

1. Вы можете вставить это ожидание в сам метод логина
2. Вы можете вставить это ожидание в сам тест, предоставив из страницы `LoginPage` удобный метод для этого

В большинстве случаев именно первый способ позволит вам действительно инкапсулировать логику.

ВОПРОСЫ

Q: Хорошо, а как мы узнаем, что страница загрузилась, если она, например, загружается медленно?

A: Один из популярных подходов — вставить проверки необходимых вам элементов в конструктор `PageObject`:

```
public class VerificationPage {
  private SelenideElement codeField = $("[data-test-id=code] input");
  private SelenideElement verifyButton = $("[data-test-id=action-verify]");

  public VerificationPage() {
    codeField.shouldBe(visible);
  }

  public DashboardPage validVerify(DataHelper.VerificationCode verificationCode) {
    codeField.setValue(verificationCode.getCode());
    verifyButton.click();
    return new DashboardPage();
  }
}
```

В остальном, никаких новшеств здесь нет, вы просто инкапсулируете логику в объекты. Ключевое здесь - практика.

ВАЖНО

Нужно ещё раз подчеркнуть, что в сообществе нет однозначного мнения из серии "правильно только вот так".

Каждый подход, из описанных нами, обладает преимуществами и недостатками — и только практика на конкретных проектах и в конкретных командах покажет, какие из них будут применимы в конкретном случае.

При этом не забывайте, что то, что будет хорошо работать в одном проекте, совершенно не обязательно будет работать в другом.

BDD

BDD

BDD (Behavior Driven Development) — набор техник, позволяющих фокусировать на получаемом результате.

В рамках BDD спецификации (описание того, что должно быть сделано) предоставляется в виде сценариев на английском (или русском) языке с определённой структурой.

Ключевым аспектом BDD является то, что эти спецификации могут быть написаны не программистами и автоматизаторами, а аналитиками, представителями бизнеса и другими людьми.

Цель BDD: получение формального инструмента, позволяющего верифицировать поведение разработанного ПО по спецификации (подход Specification By Example).

BDD: ПРИМЕР

Например, это может выглядеть вот так:

```
Функционал: Логин в Альфа-клик
  Сценарий: Успешный логин в Альфа-клик скролл внутри
    Дано совершен переход на страницу "Страница входа" по ссылке "http://alfabank.ru"
    Когда в поле "Логин" введено значение "login"
    Когда в поле "Пароль" введено значение "password"
    И выполнено нажатие на кнопку "Войти"
    Тогда страница "Альфа-клик" загрузилась
```

Как вы видите, язык не совсем "настоящий", но разница между тем, какие усилия нужно приложить для того, чтобы написать этот тест на русском, и написать его же на Java, — колоссальна.

Примечание*: "живой" пример BDD-сценария на русском языке.

BDD

BDD — это набор техник, позволяющий вам упростить написание тестов людям, не обладающим навыками программирования (в том числе ручным тестировщикам).

Ключевое: если в вашей команде никто кроме вас (автоматизаторов) и программистов (умеющих программировать) эти тесты писать не собирается, то и BDD вам не нужен — это будет лишний слой абстракции, на который вы потратите время.

GIVEN-WHEN-THEN

Общие идеи BDD формировались на основании принципов TDD и Domain Driven Design. Вы можете почитать оригинал статьи, в которой описано зарождение BDD (среди переводов есть и перевод на русский).

Мы описываем приёмочные критерии в формате, пригодном как для написания, так и для автоматической обработки.

Каждый сценарий разбивается на три ключевых секции:

1. **given** (дано) — состояние "внешнего мира" до начала вашего сценария (включая предусловия)
2. **when** (когда) — описание непосредственно самого поведения
3. **then** (тогда) — изменения, которые мы ожидаем увидеть, в результате выполнения поведения в шаге **when**

BDD: ПРИМЕР

Пример из блога Martin Fowler:

```
Feature: User trades stocks
  Scenario: User requests a sell before close of trading
    Given I have 100 shares of MSFT stock
      And I have 150 shares of APPL stock
      And the time is before close of trading

    When I ask to sell 20 shares of MSFT stock

    Then I should have 80 shares of MSFT stock
      And I should have 150 shares of APPL stock
      And a sell order for 20 shares of MSFT stock should have been executed
```

BDD

Q: Но каким способом система знает, что такое "переход на страницу", "загрузилась", "sell 20 shares"? Это уже где-то прописано?

A: Конечно, как раз этим (прописыванием того, что это всё значит) и будете заниматься вы, как автоматизатор, чтобы другие могли писать авто-тесты, не прибегая к написанию кода.

И **это** — называется шаги (`Steps`). Т.е. для того, чтобы каждый такой шаг что-то значил, нужно написать для него реализацию на Java и связать их друг с другом.

Для этого не хотелось бы изобретать велосипед, поэтому посмотрим на то, что уже есть.

РЕАЛИЗАЦИИ

Самыми популярными реализациями (поддерживающими Java) являются:

- Cucumber
- JBehave

Мы рекомендуем вам самостоятельно ознакомиться с документацией на эти инструменты, в том числе, в плане настройки (кроме того, туда ещё нужно интегрировать Selenide и прочее).

Примеры с настройкой проектов будут опубликованы в репозитории с кодом.

AKITA

AKITA

Akita — BDD библиотека шагов для тестирования на основе Cucumber и Selenide.

Чтобы это сделать, придётся немного потрудится — подключить плагин для запуска тестов и саму Akita.

```
buildscript {
  repositories {
    maven { url "https://dl.bintray.com/alfa-laboratory/maven-releases/" }
  }
  dependencies {
    classpath 'ru.alfalab.gradle:cucumber-parallel-test-gradle-plugin:0.3.2'
  }
}
plugins {
  id 'java'
  id 'io.freefair.lombok' version '4.1.3'
}
apply plugin: 'ru.alfalab.cucumber-parallel-test'
...
```

BUILDSCRIPT, APPLY PLUGIN

Q: Почему так сложно? Мы же плагин Lombok'а подключаем в одну строчку?

A: Проблема в том, что так можно подключать только Core и Community плагины, опубликованные на портале https://plugins.gradle.org. Плагин от Альфа-Банка там не опубликован.

Поэтому его приходится подключать по-старинке: `buildscript` + `apply plugin`.

В будущем, возможно, либо плагин опубликуют на портале, либо в `plugins` добавят возможность указывать доп.репозитории.

```
...
generateRunner.glue = ["ru.alfabank.steps", "ru.netology.web.step"]

group 'ru.netology'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

compileJava.options.encoding = "UTF-8"
compileTestJava.options.encoding = "UTF-8"

repositories {
  jcenter()
  mavenCentral()
}
dependencies {
  testImplementation 'ru.alfabank.tests:akita:4.1.2'
  testImplementation 'com.codeborne:selenide:5.3.1'
}
test {
  systemProperty 'selenide.headless', System.getProperty('selenide.headless')
}
```

GENERATERUNNER.GLUE

Этот атрибут содержит список пакетов, в которых нужно искать определение тех самых шагов.

В рамках Akita предоставляется достаточно большой набор уже реализованных шагов — как для взаимодействия через веб-браузер, так и для работы с REST API:

```
public class ApiSteps extends BaseMethods {
  private AkitaScenario akitaScenario = AkitaScenario.getInstance();
  /**
    * Посылается http запрос по заданному урлу без параметров и BODY.
    * Результат сохраняется в заданную переменную
    * URL можно задать как напрямую в шаге, так и указав в application.properties
    */
  @И("^выполнен (GET|POST|PUT|DELETE) запрос на URL \"([^\"]*)\". ...$")
  @And("^(GET|POST|PUT|DELETE) request to URL \"([^\"]*)\" has been executed ...$")
  public void sendHttpRequestWithoutParams(
    String method,
    String address,
    String variableName
  ) throws Exception {
    Response response = sendRequest(method, address, new ArrayList<>());
    getBodyAndSaveToVariable(variableName, response);
  }
```

ПРИМЕРЫ

В примерах от Альфа-Банка указаны следующие:

```
#language:ru

Функционал: Логин в Альфа-клик
  Сценарий: Успешный логин в Альфа-клик скролл внутри
    Дано совершен переход на страницу "Страница входа" по ссылке "http://alfabank.ru"
    Когда в поле "Логин" введено значение "login"
    Когда в поле "Пароль" введено значение "password"
    И выполнено нажатие на кнопку "Войти"
    Тогда страница "Альфа-клик" загрузилась
```

Другие примеры можно посмотреть в демо-репозитории.

ПРИМЕРЫ

```
#language:ru

Функциональность: Вход
  Структура сценария: Вход в личный кабинет
    Пусть совершен переход на страницу "Страница входа" по ссылке "ibankLoginPage"
    Когда в поле "Логин" введено значение "<login>"
    И в поле "Пароль" введено значение "<password>"
    И выполнено нажатие на кнопку "Продолжить"
    Тогда страница "Подтверждение входа" загрузилась
    Когда в поле "Код" введено значение "<code>"
    И выполнено нажатие на кнопку "Продолжить"
    Тогда страница "Дашбоард" загрузилась

    Примеры:
      | login | password  | code  |
      | vasya | qwerty123 | 99999 |
```

Шаги кликабельны в IDEA (если у вас установлены плагины Cucumber) — поэтому кликнув (`Ctrl + Click`) на конкретном шаге, вы попадёте в код, реализующий этот шаг.

АКІТА

- #language:ru — указание Cucumber, что сценарий будет на русском (иначе сломается)
- функциональность — что тестируем
- структура сценария — шаблон сценария (снизу данные для подстановки)
- пусть (дано) — given
- когда — when
- тогда — then

AKITA

Q: Но откуда он понимает, что за страницы как называются и как определяет кнопки?

A: Всё просто, вы наследуетесь от класса `AkitaPage` в своих `PageObject`'ах и помечаете нужные элементы аннотацией `@Name`:

```
@Name("Страница входа")
public class LoginPage extends AkitaPage {
  @Name("Логин")
  @FindBy(css = "[data-test-id=login] input")
  public SelenideElement loginField;
  @Name("Пароль")
  @FindBy(css = "[data-test-id=password] input")
  public SelenideElement passwordField;
  @Name("Продолжить")
  @FindBy(css = "[data-test-id=action-login]")
  public SelenideElement loginButton;

  public VerificationPage validLogin(DataHelper.AuthInfo info) {
    loginField.setValue(info.getLogin());
    passwordField.setValue(info.getPassword());
    loginButton.click();
    return page(VerificationPage.class);
  }
}
```

AKITA

Пока не очень-то удобно, правда? Хотелось бы более высокого уровня (хотя если нужны низкоуровневые проверки, то можно и так).

Но основная сила в кастомных шагах, которые мы можем написать сами:

```
#language:ru

Функциональность: Вход
  Сценарий: Вход в личный кабинет (укороченный)
  Пусть пользователь залогинен с именем "vasya" и паролем "qwerty123"
  Тогда "true" is true
```

Естественно, вместо `"true" is true` мы в дальнейшем подставим условия, но нас интересует реализация именно первой строки.

```
public class TemplateSteps {
  // главный класс, отвечающий за сопровождение шагов
  private final AkitaScenario scenario = AkitaScenario.getInstance();

  // говорим, что обрабатываем часть "Пусть"
  // через регулярные выражения вытаскиваем значения между скобками
  @Пусть("^пользователь залогинен с именем \"([^\"]*)\" и паролем \"([^\"]*)\"$")
  public void loginWithNameAndPassword(String login, String password) {
    // из .properties файла читаем свойство loginUrl
    val loginUrl = loadProperty("loginUrl");
    open(loginUrl);

    // устанавливаем текущую страницу
    scenario.setCurrentPage(page(LoginPage.class));
    val loginPage = (LoginPage) scenario.getCurrentPage().appeared();
    val authInfo = new DataHelper.AuthInfo(login, password);
    scenario.setCurrentPage(loginPage.validLogin(authInfo));

    val verificationPage = (VerificationPage) scenario.getCurrentPage().appeared();
    val verificationCode = DataHelper.getVerificationCodeFor(authInfo);
    scenario.setCurrentPage(verificationPage.validVerify(verificationCode));

    scenario.getCurrentPage().appeared();
  }
}
```

AKITA

Таким же образом, мы можем определить свои шаги для других ключевых слов, вроде `@Тогда`, `@Когда` и т.д.

Документация на Akita достаточно скудная, поэтому большую часть информации вы можете почерпнуть либо из примеров в коде самой библиотеки, либо подписавшись на оф.канал в Telegram: akitaQA.

AKITA

Ключевое: выстраивая подобный фреймворк вокруг своих тестов обязательно учитывайте стоимость его сопровождения и поддержки.

ИТОГИ

ИТОГИ

Сегодня мы рассмотрели pattern PageObject и способы реализации его в Selenide.

Кратко рассмотрели подход BDD и познакомились с библиотекой Akita.

ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.

Важно: начиная с сегодняшнего дня вам нужно сдавать все ДЗ только с использованием PageObject. ДЗ, которые реализованы без использования PageObject будут сразу отправляться на доработку.

Задавайте вопросы и напишите отзыв о лекции!

АРТЕМ РОМАНОВ

romanov-artyom

@Artem_Telegram

Романов Артём