

Лекция 10

Page Object Model (POM)

*Дополнение: уже после подготовки лекции наткнулась на **интересную и полезную статью**, которая в лекции не освещается, поэтому очень **рекомендую к самостоятельному прочтению***

Ссылка на источник, но
он по подписке

Going Deeper into the Page Object Model

Twelve design considerations when implementing page objects



Blake Norrish · [Follow](#)

20 min read · Jul 14, 2022

Ссылка на перевод
статьи

Большой гайд по Page Object Model

testengineer.ru • 26 июля, 2022

Мы имеем тестовый класс, где прописаны шаги нашего теста:

```
public class NoPOMTest99GuruLogin {  
  
    /**  
     * This test case will login in http://demo.guru99.com/V4/  
     * Verify login page title as guru99 bank  
     * Login to application  
     * Verify the home page using Dashboard message  
     */  
    @Test(priority=0)  
    public void test_Home_Page_Appear_Correct(){  
        WebDriver driver = new FirefoxDriver();  
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
        driver.get("http://demo.guru99.com/V4/");  
        //Find user name and fill user name  
        driver.findElement(By.name("uid")).sendKeys("demo");  
        //find password and fill it  
        driver.findElement(By.name("password")).sendKeys("password");  
        //click login button  
        driver.findElement(By.name("btnLogin")).click();  
        String homeText = driver.findElement(By.xpath("//table//tr[@class='heading3']")).getText();  
        //verify login success  
        Assert.assertTrue(homeText.toLowerCase().contains("guru99 bank"));  
    }  
}
```

1 Find user name and fill it

2 Find password and fill it

3 Find Login button and click it

4 Find home page text and get it

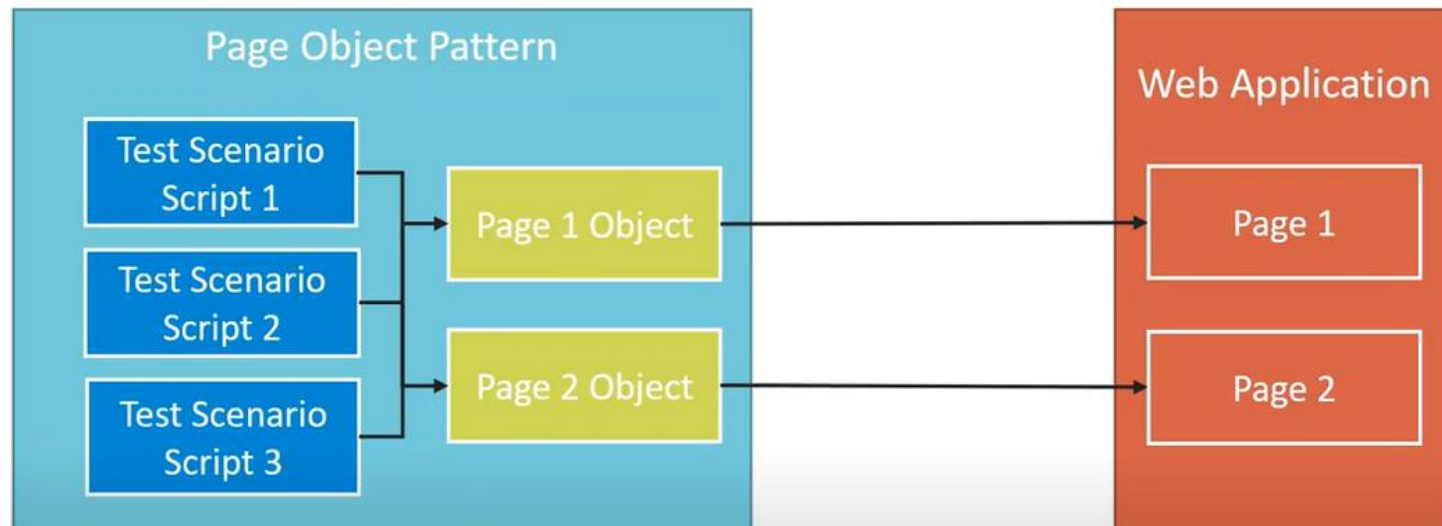
5 Verify home page has text 'Guru99 Bank'

Но такая запись сильно зависима от изменений UI и к тому же в случае больших тестовых сценариев рискует стать трудночитаемой.

Page Object Model (POM) — это шаблон проектирования в Selenium, который создает репозиторий объектов для хранения всех веб-элементов.

Основные принципы

1. Разделение тестового сценария и контента страницы. Создаем отдельные:
 - классы которые инкапсулируют тестовый сценарий;
 - классы которые инкапсулируют внутреннее устройство самих страниц и методы работы с элементами этих страниц.
2. Все взаимодействия со страницей должны находится в одном месте.



Идеи Page Object следуют из принципов написания кода

DSL



Domain Specific Language

DSL- язык специфичный для домена. Имена тестов, именованние классов страниц и методов должно содержать специфику предметной области. Например, CustomerPage – AddToCard

DRY



Don't Repeat Yourself

DRY – в рамках PageObject это значит что мы должны разделить все приложение на уникальные страницы с неповторяющимся функционалом

KISS



Keep It Simple Stupid

KISS – упрощай до невозможности. Лучше реализовать в коде несколько маленьких классов, чем один большой

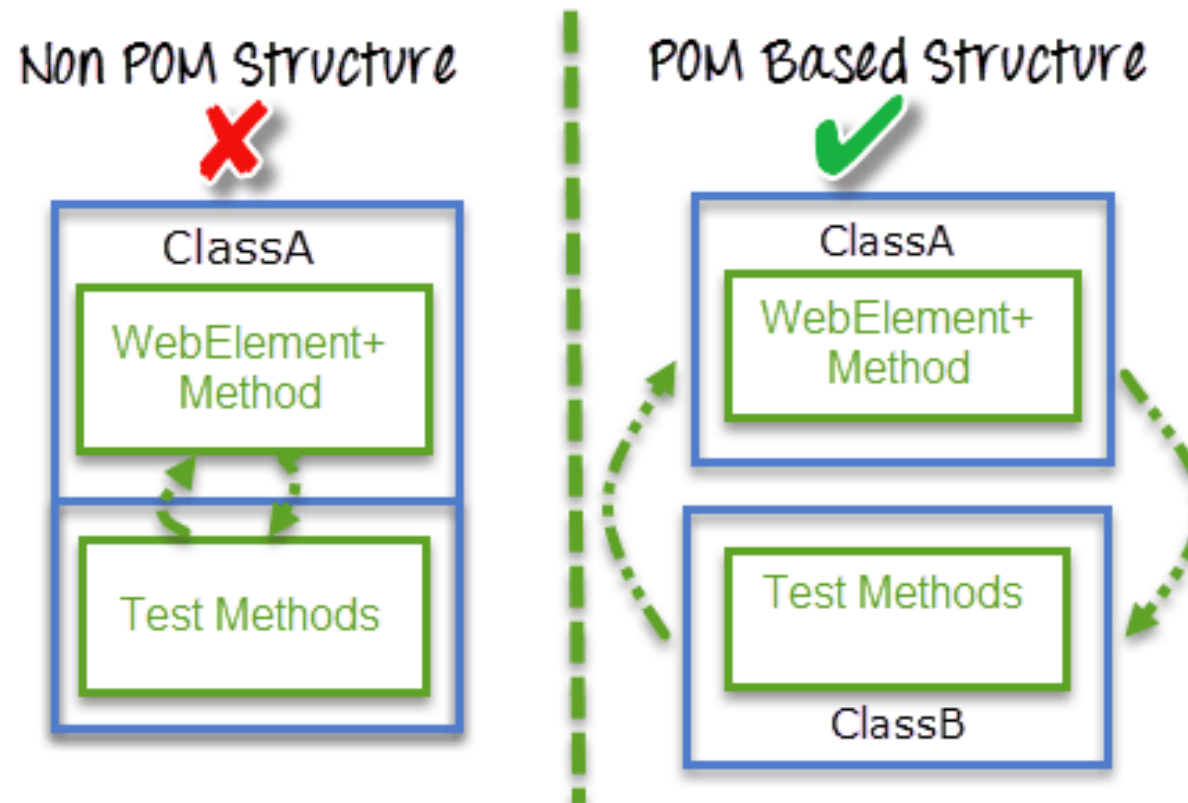
YAGNI



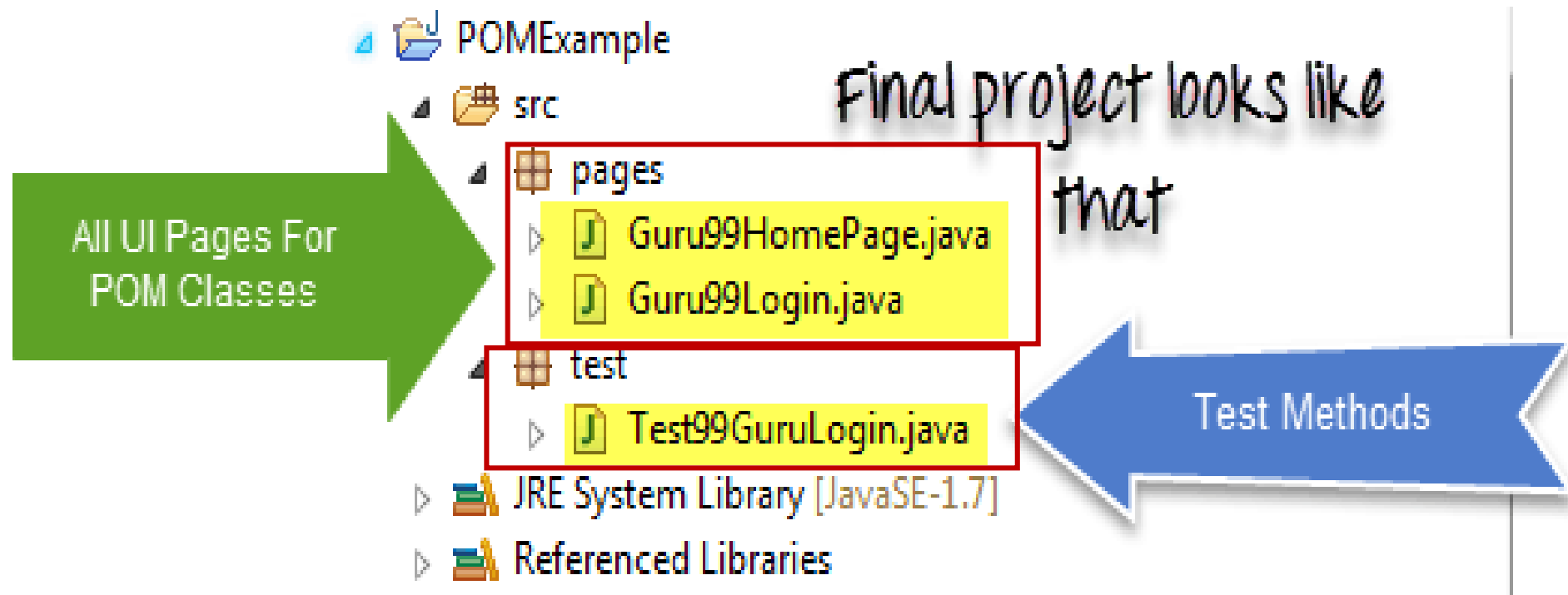
You Aren't Gonna Need It

YAGNI – не стоит реализовывать сразу весь функционал который нам может пригодиться в будущем, т.е. не стоит предугадывать сразу все изменения в системе

Лучшим подходом к обслуживанию скриптов является создание отдельного файла класса, который будет находить веб-элементы, заполнять их или проверять. Этот класс можно повторно использовать во всех сценариях, использующих этот элемент. В будущем, если в веб-элементе произойдут изменения, нужно будет внести изменения всего в 1 файл класса, а не в несколько тестовых скриптов.



Согласно этой модели, для каждой веб-страницы участвующей в стадии автоматического тестирования должен существовать соответствующий класс страницы. Этот класс Page будет идентифицировать WebElements этой веб-страницы, а также содержать методы Page, которые выполняют операции с этими WebElements.



Класс страницы в Page Object Model

инкапсулирует элементы страницы и методы работы с этими элементами.

```
public class Guru99Login {  
    WebDriver driver;  
    By user99GuruName = By.name("uid");  
    By password99Guru = By.name("password");  
    By titleText = By.className("barone");  
    By login = By.name("btnLogin");  
  
    public Guru99Login(WebDriver driver){  
        this.driver = driver;  
    }  
    //Set user name in textbox  
    public void setUserName(String strUserName){  
        driver.findElement(user99GuruName).sendKeys(strUserName);  
    }  
}
```

1 Page class in object repository

Find Web Element

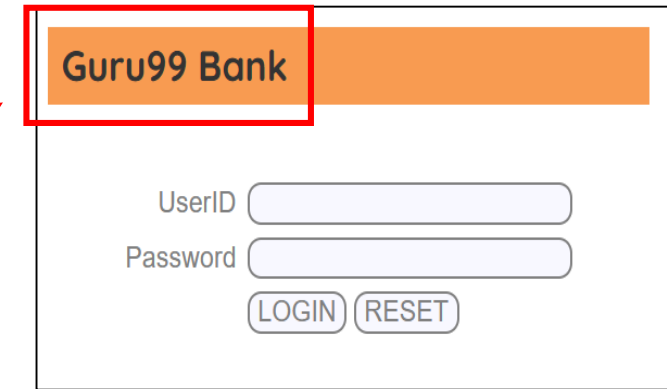
Performing operation on Web element

2

3

Тестовый пример: Перейти на демонстрационный сайт Guru99

1. Перейдите на страницу входа в guru99:
demo.guru99.com/V4
2. Чтобы убедиться, что вы находитесь на правильной домашней странице, проверьте наличие текста «Guru99bank».
3. Войдите в приложение, используя учетные данные.
4. Убедитесь, что на главной странице есть следующий текст

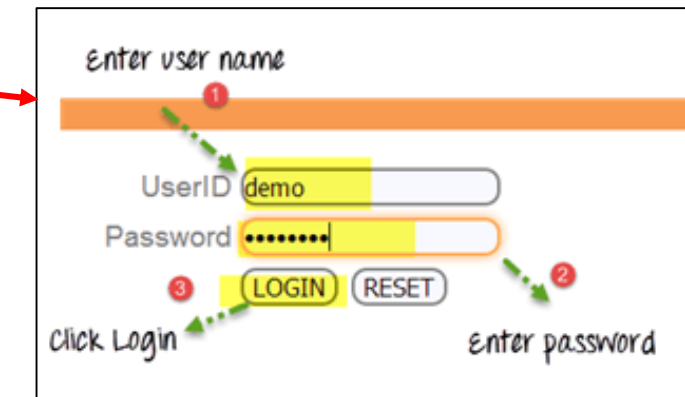


Guru99 Bank

UserID

Password

LOGIN RESET



Enter user name

1

UserID demo

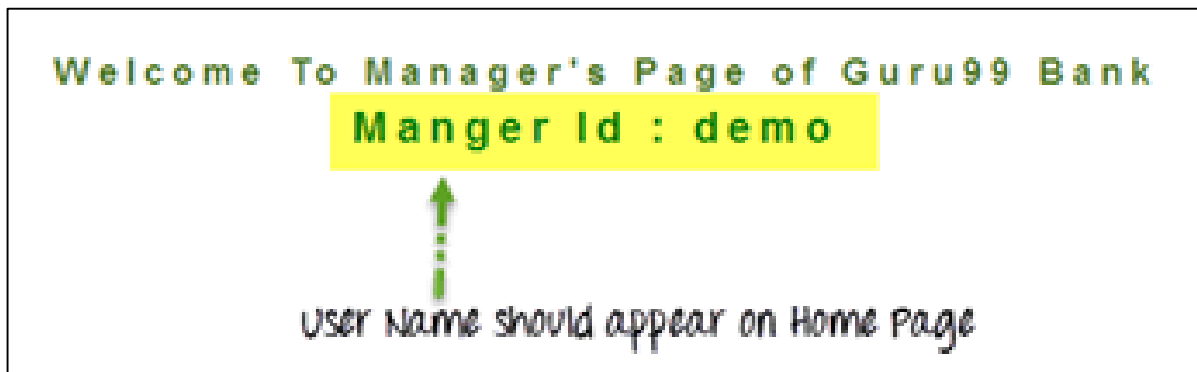
Password

2

3 LOGIN RESET

click Login

enter password



Welcome To Manager's Page of Guru99 Bank

Manger Id : demo

User Name should appear on Home Page

Для этого примера структура POM будет состоять из 2ух классов страниц и 1 тестового класса

class Guru99Login

```
public class Guru99Login {
    WebDriver driver;
    By user99GuruName = By.name("uid");
    By password99Guru = By.name("password");
    By titleText = By.className("barone");
    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){
        this.driver = driver;
    }
    //Set user name in textbox
    public void setUsername(String strUserName){
        driver.findElement(user99GuruName).sendKeys(strUserName);
    }
    //Set password in password textbox
    public void setPassword(String strPassword){
        driver.findElement(password99Guru).sendKeys(strPassword);
    }
    //Click on login button
    public void clickLogin(){
        driver.findElement(login).click();
    }
    //Get the title of Login Page
    public String getLoginTitle(){
        return driver.findElement(titleText).getText();
    }
}
```

```
/* This POM method will be exposed in test case to login in the a
 * @param strUserName
 * @param strPasword
 * @return
 */
```

```
public void loginToGuru99(String strUserName,String strPasword){
    //Fill user name
    this.setUsername(strUserName);
    //Fill password
    this.setPassword(strPasword);
    //Click Login button
    this.clickLogin();
}
```

class Guru99HomePage

```
1 package pages;
2 import org.openqa.selenium.By;
3 import org.openqa.selenium.WebDriver;
4
5 public class Guru99HomePage {
6     WebDriver driver;
7     By homePageUserName = By.xpath("//table//tr[@class='heading3']");
8
9     public Guru99HomePage(WebDriver driver){
10         this.driver = driver;
11     }
12
13     //Get the User name from Home Page
14     public String getHomePageDashboardUserName(){
15         return driver.findElement(homePageUserName).getText();
16     }
17
18 }
```

```
public class Test99GuruLogin {
    String driverPath = "C:\\\\geckodriver.exe";
    WebDriver driver;
    Guru99Login objLogin;
    Guru99HomePage objHomePage;
    @BeforeTest

    public void setup(){
        System.setProperty("webdriver.gecko.driver", driverPath);
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://demo.guru99.com/V4/");
    }

    @Test(priority=0)
    public void test_Home_Page_Appear_Correct(){
        //Create Login Page object
        objLogin = new Guru99Login(driver);
        //Verify login page title
        String loginpageTitle = objLogin.getLoginTitle();
        Assert.assertTrue(loginpageTitle.toLowerCase().contains("guru99 bank"));
        //login to application
        objLogin.loginToGuru99("mgr123", "mgr!23");
        // go the next page
        objHomePage = new Guru99HomePage(driver);
        //Verify home page
        Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));
    }
}
```

class Test99GuruLogin

Преимущества модели Page Object

- ✓ **Возможность повторного использования** – методы объекта страницы в разных классах POM могут быть повторно использованы в разных тестовых скриптах. Таким образом, общий размер кода значительно сократится.
- ✓ **Проще поддерживать тесты** — из-за разделения классов код становится чище, и на поддержку тестового кода тратится меньше усилий.
- ✓ **Минимальное изменение кода из-за изменений пользовательского интерфейса** — изменения могут потребоваться только в локаторах. Влияние на реализацию тестовых сценариев минимально или отсутствует.
- ✓ Репозиторий объектов не зависит от тестовых случаев, поэтому **можно использовать один и тот же репозиторий объектов для разных целей с разными инструментами**. Например, можно интегрировать объектную модель страницы в Selenium TestNG/JUnit для функционального тестирования и в то же время с JBehave/Cucumber для приемочных испытаний.

Page Factory

Page Factory в Selenium — это оптимизированный способ создания репозитория объектов в концепции платформы объектной модели страницы.

Она используется для инициализации объектов Page или для создания экземпляра самого объекта Page. Он также используется для инициализации элементов класса Page без использования «FindElement/s».

С помощью класса PageFactory в Selenium, мы используем аннотации @FindBy чтобы найти веб-элемент. Мы используем метод initElements для инициализации веб-элементов.

@FindBy могу принять имя тега, частичный текст ссылки, имя, текст ссылки, идентификатор, CSS, имя класса, xpath как атрибуты.


WebElements are identify by
@FindBy Annotation

Static initElements method of
PageFactory class for
initializing WebElement



```
@FindBy(xpath="//table//tr[@class='heading3']")  
WebElement homePageUserName;
```

```
public Guru99HomePage(WebDriver driver){  
    this.driver = driver;  
    //This initElements method will create all WebElements  
    PageFactory.initElements(driver, this);  
}
```



```

11 public class Guru99Login {
12     /**
13      * All WebElements are identified by @FindBy annotation
14      */

```

```

15
16     WebDriver driver;
17     @FindBy(name="uid")
18     WebElement user99GuruName;
19     @FindBy(name="password")
20     WebElement password99Guru;
21     @FindBy(className="barone")
22     WebElement titleText;
23     @FindBy(name="btnLogin")
24     WebElement login;

```

```

25
26     public Guru99Login(WebDriver driver){
27         this.driver = driver;
28         //This initElements method will create all WebElements
29         PageFactory.initElements(driver, this);
30     }

```

```

40 //Click on login button

```

```

41 public void clickLogin(){
42     login.click();
43 }

```

```

44
45 //Get the title of Login Page

```

```

46 public String getLoginTitle(){
47     return titleText.getText();
48 }

```

Стало

Было

```

public class Guru99Login {
    WebDriver driver;
    By user99GuruName = By.name("uid");
    By password99Guru = By.name("password");
    By titleText = By.className("barone");
    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){
        this.driver = driver;
    }

    //Click on login button
    public void clickLogin(){
        driver.findElement(login).click();
    }

    //Get the title of Login Page
    public String getLoginTitle(){
        return driver.findElement(titleText).getText();
    }
}

```

Пример реализации POM на Python

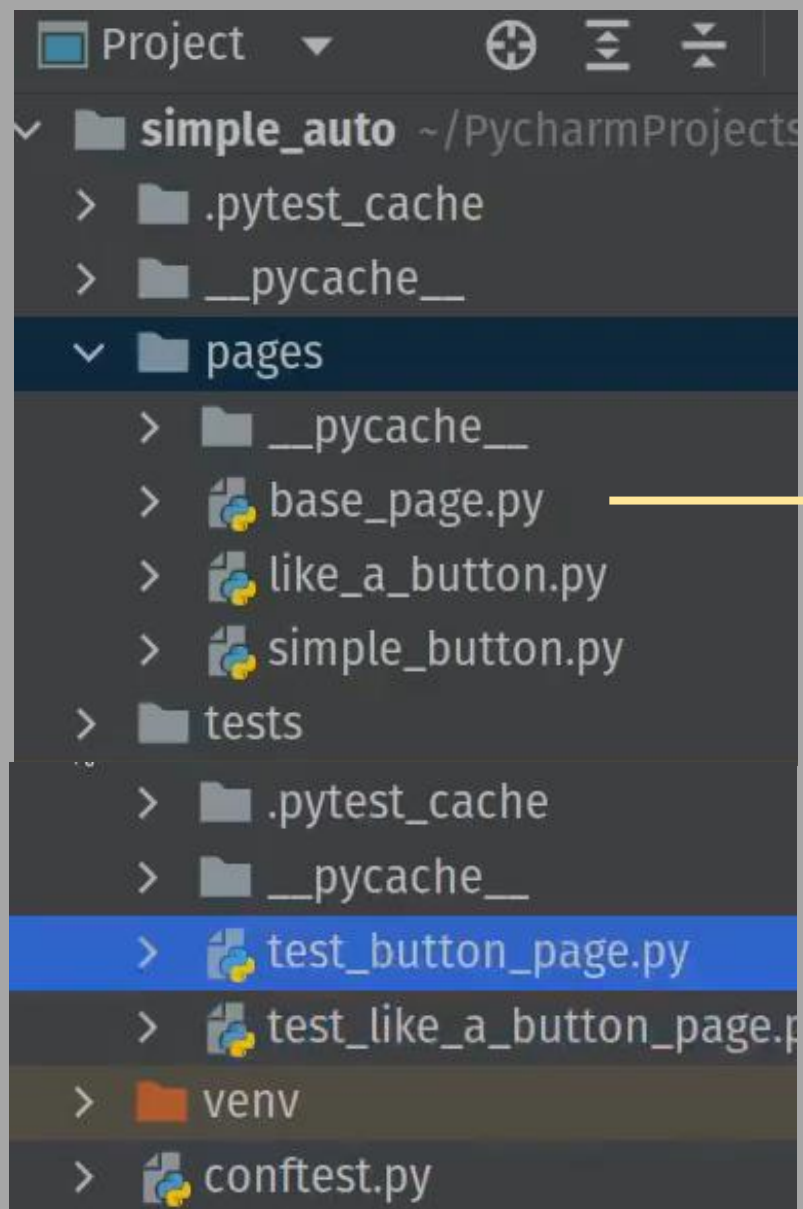
Задача:

Необходимо протестировать кнопки на двух страницах.

Реализация:

Классы для каждой из страниц и 2 тестовых класса для элементов каждой страницы.

Структура проекта



```
test.py × base_page.py × simple_button.py × like_a_button.py
class BasePage:
    def __init__(self, browser):
        self.browser = browser

    def find(self, args):
        return self.browser.find_element(*args)
```

От класса **BasePage** будем наследовать все страницы.
Вынесем в него всю повторяющуюся логику.

```
button_selector = (By.ID, 'submit-id-submit')
result_selector = (By.ID, 'result-text')
```

селекторы

```
class SimpleButtonPage(BasePage):
    def __init__(self, browser):
        super().__init__(browser)

    def open(self):
        self.browser.get('https://www.qa-practice.com/elements/button/simple')


    def button(self):
        return self.find(button_selector)

    def button_is_displayed(self):
        return self.button().is_displayed()

    def click_button(self):
        self.button().click()

    def result(self):
        return self.find(result_selector)

    def result_text(self):
        return self.result().text
```

 simple_button.py

Класс страницы

nfctest.py ×  base_page.py ×  simple_button.py ×  like_a_button.py  test_button_page.py ×

```
from selenium.webdriver.common.by import By
from pages.simple_button import SimpleButtonPage

def test_button1_exist(browser):
    simple_page = SimpleButtonPage(browser)
    simple_page.open()
    assert simple_page.button_is_displayed()

def test_button1_clicked(browser):
    simple_page = SimpleButtonPage(browser)
    simple_page.open()
    simple_page.click_button()
    assert 'Submitted' == simple_page.result_text
```

Фреймворки для Page Object

- [Splinter](#) — довольно мощная надстройка над Selenium WebDriver, имеет множество дополнительных методов. Для работы совместно с PyTest существует специальный плагин [pytest-splinter](#), включающий собственные фикстуры.
- [Selene](#) — реализация для Python идей популярного Java-фреймворка Selenide.
- [PyPOM](#) — библиотека для реализации Page Object Model от разработчиков из проекта Mozilla. Поддерживает работу со Splinter <https://github.com/mozilla/PyPOM>
- [Webium](#) — легковесная реализация Page Object Model от разработчиков и тестировщиков Wargaming.net

Еще один пример реализации POM на Python

Здесь локаторы вынесены в отдельные файлы, что бы не перегружать классы страниц

локаторы

объекты страниц

тесты

```

POM_ProjNew C:\Users\User\PycharmPro
├── .venv library root
├── data
├── generator
├── locators
│   ├── alerts_frame_windows_locators.py
│   ├── elements_page_locators.py
│   └── form_page_locators.py
├── pages
│   ├── alerts_frame_windows_page.py
│   ├── base_page.py
│   ├── elements_page.py
│   └── form_page.py
├── tests
│   ├── alerts_frame_windows_test.py
│   ├── elements_test.py
│   ├── form_test.py
│   └── conftest.py
├── requirements.txt
├── External Libraries
└── Scratches and Consoles

```


class FormPageLocators

```
6  ✓ class FormPageLocators: 2 usages
7      FIRST_NAME = (By.CSS_SELECTOR, '#firstName')
8      LAST_NAME = (By.CSS_SELECTOR, '#lastName')
9      EMAIL = (By.CSS_SELECTOR, '#userEmail')
10     GENDER = (By.CSS_SELECTOR, f"div[class*='custom-control'] label[for='gender-radio-{{random.randint(1, 2)}}']")
11     MOBILE = (By.CSS_SELECTOR, "input[id='userNumber']")
12     DATE_OF_BIRTH = (By.CSS_SELECTOR, 'id="dateOfBirthInput"')
13     SUBJECT = (By.CSS_SELECTOR, 'input[id="subjectsInput"]')
14     HOBBIES = (By.CSS_SELECTOR, f'div[class*="custom-control"] label[for="hobbies-checkbox-{{random.randint(1, 2)}}"]')
15     FILE_INPUT = (By.CSS_SELECTOR, "input[id='uploadPicture']")
16     CURRENT_ADDRESS = (By.CSS_SELECTOR, '#currentAddress')
17     SELECT_STATE = (By.CSS_SELECTOR, 'div[id="state"]')
18     STATE_INPUT = (By.CSS_SELECTOR, 'input[id="react-select-3-input"]')
19     SELECT_CITY = (By.CSS_SELECTOR, 'div[id="city"]')
20     CITY_INPUT = (By.CSS_SELECTOR, 'input[id="react-select-4-input"]')
21     SUBMIT = (By.CSS_SELECTOR, '#submit')
22
23     # table result
24     RESULT_TABLE = (By.XPATH, '//div[@class="table-responsive"]//td[2]')
```

class BasePage

```
class BasePage:
    def __init__(self, driver, url):
        self.driver = driver
        self.url = url

    def open(self):
        self.driver.get(self.url)

    def element_is_visible(self, locator, timeout=5):
        self.go_to_element(self.element_is_present(locator))
        return wait(self.driver, timeout).until(EC.visibility_of_element_located(locator))
```

```
    def element_is_clickable(self, locator, timeout=5):
        return wait(self.driver, timeout).until(EC.element_to_be_clickable(locator))
```

```
    def go_to_element(self, element): 1 usage
        self.driver.execute_script("arguments[0].scrollIntoView();", element)
```

```
    def action_double_click(self, element):
        action = ActionChains(self.driver)
        action.double_click(element)
        action.perform()
```

+

```
def action_right_click()
```

```
def action_drag_and_drop_by_offset()
```

```
def action_move_to_element()
```

.....

class FormPage

```
11 class FormPage(BasePage): 2 usages
12     locators = FormPageLocators()
13
14     @allure.step('fill in all fields') 1 usage
15     def fill_form_fields(self):
16         person = next(generated_person())
17         file_name, path = generated_file()
18         self.remove_footer()
19         self.element_is_visible(self.locators.FIRST_NAME).send_keys(person.firstname)
20         self.element_is_visible(self.locators.LAST_NAME).send_keys(person.lastname)
21         self.element_is_visible(self.locators.EMAIL).send_keys(person.email)
22         ...
34         self.element_is_visible(self.locators.SUBMIT).click()
35         return person
36
37     @allure.step('get form result') 1 usage
38     def form_result(self):
39         result_list = self.elements_are_visible(self.locators.RESULT_TABLE)
40         data = []
41         for item in result_list:
42             self.go_to_element(item)
43             data.append(item.text)
44         return data
```


class TestForm

```
from pages.form_page import FormPage

|

@allure.suite('Forms')
class TestForm:

    @allure.feature('FormPage')
    class TestFormPage:

        @allure.title('Check form')
        def test_form(self, driver):
            form_page = FormPage(driver, url: 'https://demoqa.com/automation-practice-form')
            form_page.open()
            p = form_page.fill_form_fields()
            result = form_page.form_result()
            assert [p.firstname + ' ' + p.lastname, p.email] == [result[0], result[1]], 'the form
```

Р А З Н О Е из QA

Экскурс в «святая святых» ОК: как мы пишем и ревьюим код автотестов

14.08.2024 00:00

<https://software-testing.ru/library/testing/testing-automation/4246-ok>

Материал подготовлен по мотивам доклада руководителя команды автоматизации тестирования ОК Эмили Куцаревой и младшего инженера по автоматизации тестирования соцсети Евгения Буровникова на ИТ-конференции «Стачка».

ОК — одна из самых популярных социальных сетей в рунете, которая представлена на всех возможных платформах (web, mobile web, API, android, iOS).

Наш продукт высоконагружен, имеет сложный бэкенд и сотни сервисов.

Например, у него «под капотом»: **50 тысяч Docker-контейнеров, 1 эксабайт данных и обработка данных в 7 дата-центрах.**

Так, сейчас у нас более 10 тысяч автотестов:

- web — около 3150;
- API — около 2500 (+1500 автосгенерированных);
- Android — около 1400;
- mobile web — около 1200;
- iOS — около 950.