Selenium WebDriver
Тестовый фреймворк

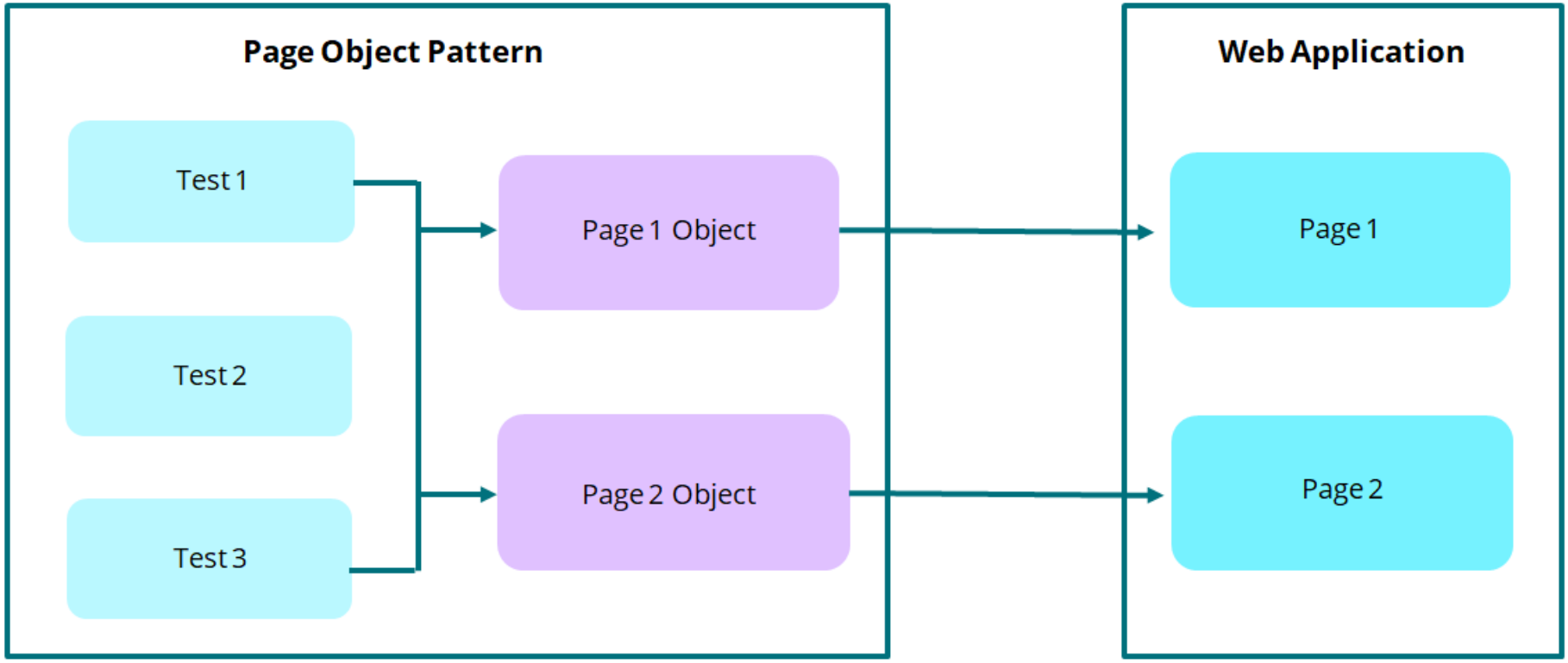
Page Object

- Если у вас много тестов для одной страницы, то у вас часто будет дублироваться следующий код:
  - Получение нужных элементов на странице (элементов форм, кнопок и т.д.)
  - Выполнение некоторых одинаковых действий с этими элементами
- Чтобы избавиться от дублирования этой логики используется шаблон проектирования **Page Object**
- https://kreisfahrer.gitbooks.io/selenium-webdriver/content/page_object_pattern_arhitektura_testovogo_proekta/ispolzovanie_patterna_page_object.html
- http://internetka.in.ua/selenium-page-object/

Page Object

- Смысл **Page Object** – для каждой страницы вы создаете отдельный класс, который упрощает работу с этой страницей
- Там вы делаете удобные методы для выполнения частых действий
- Внутри этого класса легко обращаться к элементам страницы
- **Плюсы**:
  - Отделение логики самого теста от логики работы со страницей
  - Логика работы со страницей находится в одном месте
  - Избавление от дублирования кода

Page Object


Page Object Pattern
Test 1
Test 2
Test 3
Page 1 Object
Page 2 Object
Web Application
Page 1
Page 2

Принципы

- **DSL (Domain Specific Language)** – используем термины предметной области в названиях методов
- **DRY (Don’t Repeat Yourself)** – методы каждой страницы должна быть уникальными, не повторяющимися
- **KISS (Keep It Simple Stupid)** – лучше несколько небольших page object’ов, чем все в одном классе (атомарные объекты). Меньше сложных вспомогательных классов
- **YAGNI (You Aren’t Gonna Need It)** – лучше добавлять методы (и элементы) по мере необходимости, не нужно сразу реализовывать все возможные методы (и локаторы для всех возможных элементов)

FindBy и Page Factory

- Поля для хранения элементов в page object:

```
@FindBy(name="q")
private WebElement searchField;

@FindBy(tagName="a")
private List<WebElement> list;
```

Если мы не пропишем Page Factory в конструкторе, то аннотации FindBy не обработаются и мы получим Null Pointer Exception

- В конструкторе page object (в AbstractPage):

```
PageFactory.initElements(this.driver, this);
```

Page Factory

- **Смысл:** инициализирует элементы страницы только при обращении к ним, не заранее
- В методе **initElements** драйвер начинает искать элементы только в момент обращения к полю класса.
- Дальнейшее выполнение «под капотом» будет аналогично **driver.findElement**, как мы прописывали раньше. Только теперь за нас это делаем Page Factory

Page Object

```
public class LoginPage {
    private WebDriver webDriver;
    private WebDriverWait wait;

    @FindBy(id = “username”)
    private WebElement userName;

    public LoginPage(WebDriver driver) {
        webDriver = driver;
        wait = new WebDriverWait(webDriver);

        PageFactory.initElements(webDriver, this);
    }
}
```

Поле userName само заполнится по локатору

Логика заполнения полей по локатору работает из-за этой строки

Page Object и Page Factory

Page Object Model	Page Factory
Ищем элементы с помощью By и WebElement	Ищем элементы с помощью аннотации FindBy Можно комбинировать с By и WebElement
Инициализация «не ленивая»	Инициализация «ленивая» - только при обращении к элементу (экономит ресурсы)
Дизайн паттерн	Реализация дизайн паттерна Page Object

Page Object

- ```
  public class LoginPage {
      public void login(String login) {
          userName.sendKeys(login);
          userName.sendKeys(Keys.RETURN);
      }
  }
  ```
- А в классе с тестом просто создаем **Page Object**, и работаем с ним
  ```

Facade

- И часто кроме **Page Object** используют шаблон **Facade** (**фасад**)
- В чем смысл – часто при тестировании сценарии нам надо что-то сделать на разных страницах
- Поэтому нам надо несколько **Page Object**’ов
- Facade позволяет вам легко получать **Page Object**’ы всех нужных страниц

Facade

```
public class MantisSite {
    private WebDriver webDriver;

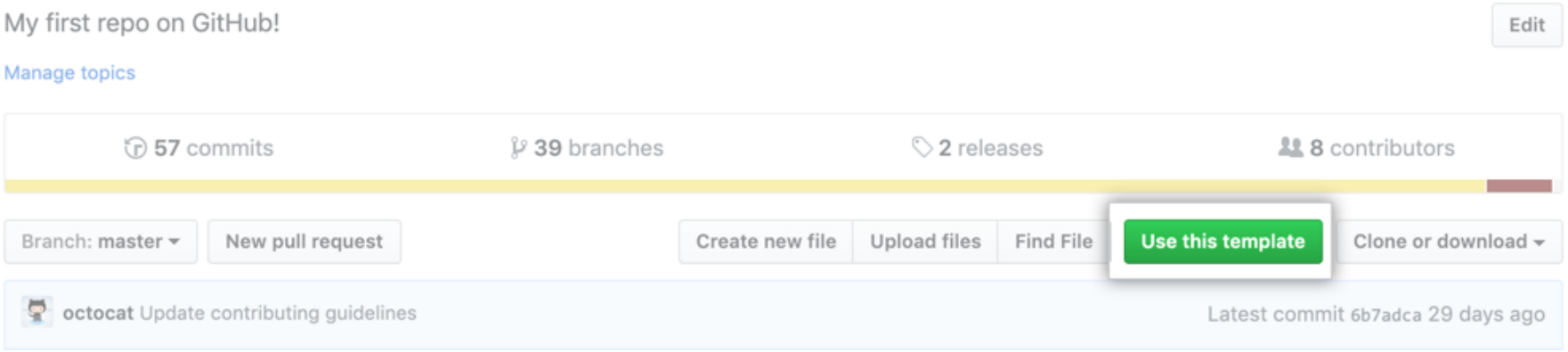
    public BooksSite(WebDriver driver) {
        webDriver = driver;
    }

    public LoginPage loginPage() {
        return new LoginPage(webDriver);
    }

    // и т.д. для каждой страницы
}
```

Создание репозитория из template’а

- Адрес репозитория: https://github.com/elena-balakina/6-seleniumWebDriverMavenPageObject
- Документация как создать репозиторий из образца: https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template



Доработка фреймворка

- **Доработать фреймворк:**
- Добавить страницу ViewIssuesPage
- Добавить метод для подсчета issues, отображаемых на странице
- Добавить необходимые данные для новой страницы в MantisSite
- Для списка элементов на странице используется следующий формат поиска селекторов:

```
@FindBy(селектор)
private List<WebElement> name;
```

Практика

- **Написать тест:**
- Сделать проверку того, что на странице ViewIssuesPage отображается 50 issues

Задание на дом «Framework»

- Написать тест на создание и удаление нового issue (в одном тесте, чтобы не удалять чужие issue)
- Доработать фреймворк:
  - добавить новую страницу,
  - прописать ее в MantisSite,
  - создать необходимые для теста методы,
  - проверки прописывать явно в самом тесте,
  - реализовать несколько проверок в одном тесте с помощью soft assertions.