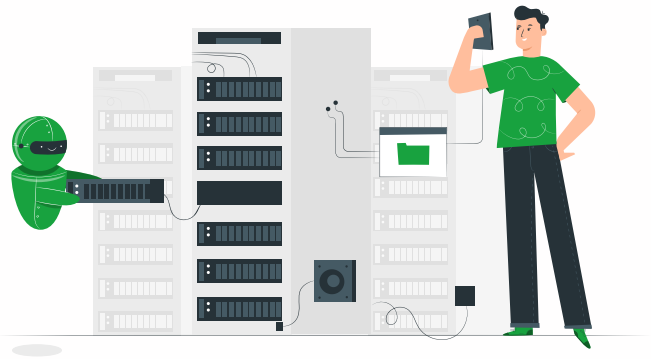
26.1. Что такое паттерны программирования и что такое PageObject



Шаблоны проектирования, они же паттерны программирования, происходят от английского «*design pattern*». Слово «*pattern*» имеет значения «шаблон», «образец». То есть мы говорим о некоем образце, принятом в индустрии, следование которому является хорошей практикой и увеличивает шансы на успех.

Воспринимать эти паттерны стоит как экспертные рекомендации. Вы, конечно, можете разрабатывать программы, не обращая на них внимания, но в этом случае вас ждёт «хождение по граблям» и непонимание коллег. Отступление от них, скорее, прерогатива опытных разработчиков, которые испробовали разные подходы и паттерны, знают их плюсы и минусы, и хорошо понимают, когда и где стоит придумывать свой подход.

Таких шаблонов достаточно много. Некоторые созданы применительно к конкретной библиотеке, а некоторые настолько высокоуровневые, что не зависят от конкретного языка программирования. Например, в своей практике вы, наверное, столкнётесь с такой концепцией, как MVC (модель-представление-контроллер). Она применяется в разработке как сайтов, так и десктопных, мобильных приложений на любых языках программирования.

Давайте обозначим плюсы и минусы применения паттернов программирования.

Плюсы	Минусы
1. Снижение сложности разработки. 2. Общепринятость решения и термина, взаимопонимание с другими разработчиками. 3. Снижение рисков за счет «обкатанности» подхода. 4. Более лёгкая интеграция с другими модулями и решениями, следующими данному паттерну. 5. Более лёгкая преемственность при передаче проекта новому разработчику или расширении проекта.	1. Иногда приводят к излишнему усложнению программы. 2. Универсальность внедряемого решения снижает адаптированность под особенности конкретного проекта, его эффективность. 3. Иногда разработчики применяют шаблоны, не задумываясь, нужен ли он в данном случае.

Существует также понятие «антипаттерн». Это часто встречающийся подход в разработке, который является плохой практикой. Конечно, можно поспорить, что считать плохим или хорошим паттерном. Примеры некоторых антипаттернов, по версии компании *DeepSource*, можно посмотреть тут.

Задание 26.1.1

1/1 point (graded)
Попробуйте догадаться, к паттерну или антипаттерну принадлежит данная конструкция:

```
try:
    do_something()
except:
    pass
```

- Паттерн
- Антипатерн ✔

Ответ
Верно:
Считается, что «замалчивание» исключений не дает вовремя исправить логические ошибки в программе, что рано или поздно приведет к неприятностям.

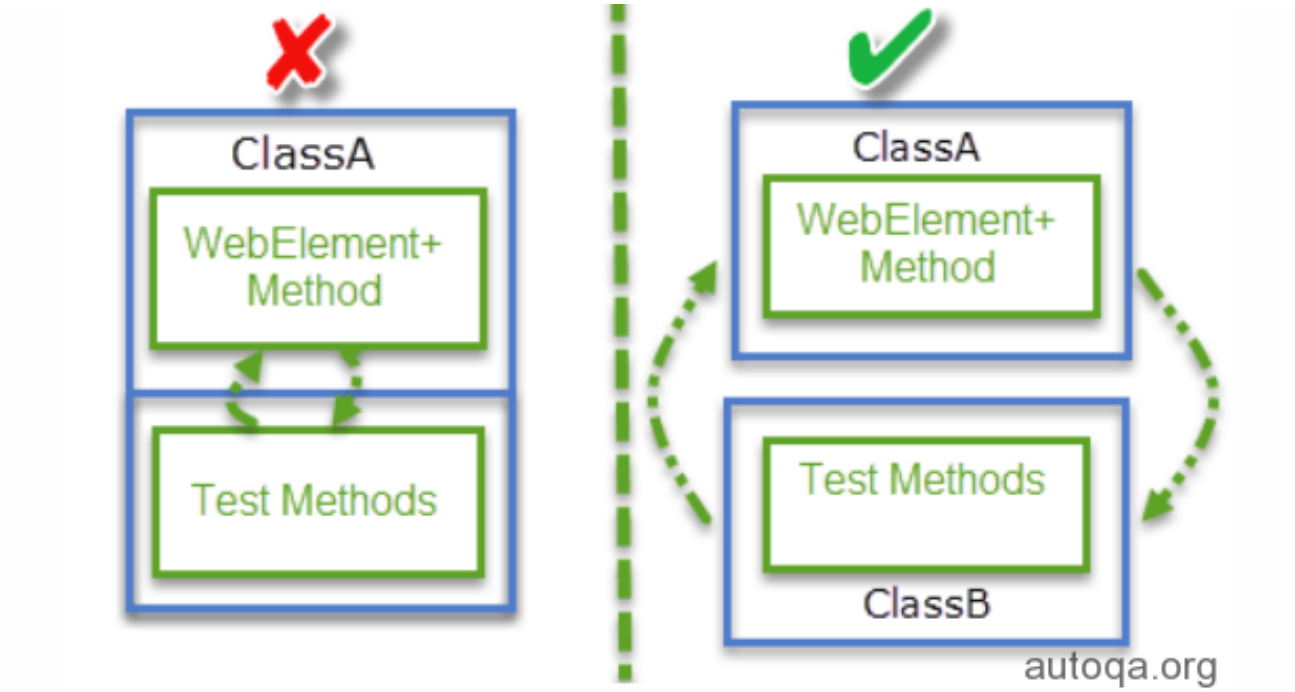
Отправить

✔ Correct (1/1 point)

Паттерн PageObject

При разработке *UI*-тестов возникают большие проблемы при, казалось бы, незначительных изменениях интерфейса. Изменит ли верстальщик вложенность блоков, переименует ли класс стиля или поменяет тег с `div` на `span` — ваши тесты начнут выдавать исключения о невозможности найти элемент. Нужно зайти в каждый тест, связанный с этим веб-элементом. А когда дизайнер решит вернуть всё назад, вам захочется его убить ;)

Минимизировать количество телодвижений поможет вынесение за скобки общего функционала, связанного с особенностями дизайна. Тогда менять код придётся лишь в одном месте.



PageObject (иногда *Page Object Model*, *POM*) — это паттерн программирования, основанный на идее создания отдельных программных объектов, в которых будут скрыты все методы по работе с конкретной страницей. Мы разделяем логику теста и специфику работы с *HTML*-элементами. Таким образом, тесты работают с интерфейсом через объекты *PageObject*, абстрагируясь от конкретной реализации в *HTML*-коде.

На самом деле, так можно поступать не только с цельными веб-страницами, но и отдельными элементами, такими как меню, форма регистрации, поле комментариев, строка пагинации (строка, где указаны номера страниц для листания по ним) и так далее.

Главный плюс использования *PageObject* в том, что при изменении элемента дизайнером вам нужно будет обновить код лишь в одном месте. Ведь тестов, работающих с одной и той же страницей, может быть десятки.

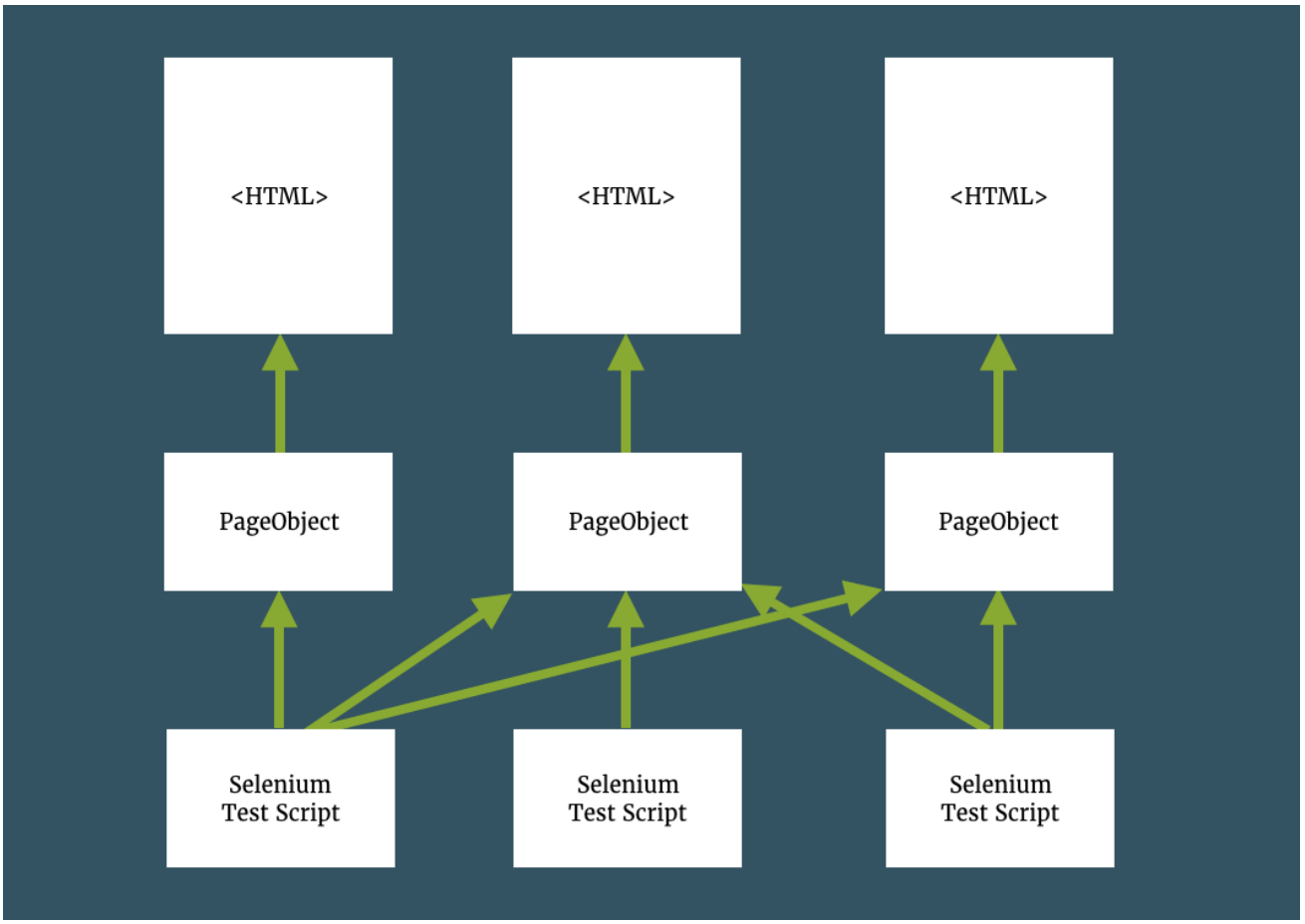


Схема взаимодействия тестовых скриптов, объектов страниц и визуального представления страниц
Источник: svittal-pensieve.blogspot.com

Для реализации паттерна *PageObject* нужен базовый класс страницы, содержащий общие методы. Далее, на его основе мы создаём классы для конкретных страниц, добавляя специфичные для них локаторы и методы. И уже затем используем объекты этих классов в наших тестах. Не стоит пытаться воссоздать все возможные действия над страницей. Необходимы те, которые нужны для тестирования.

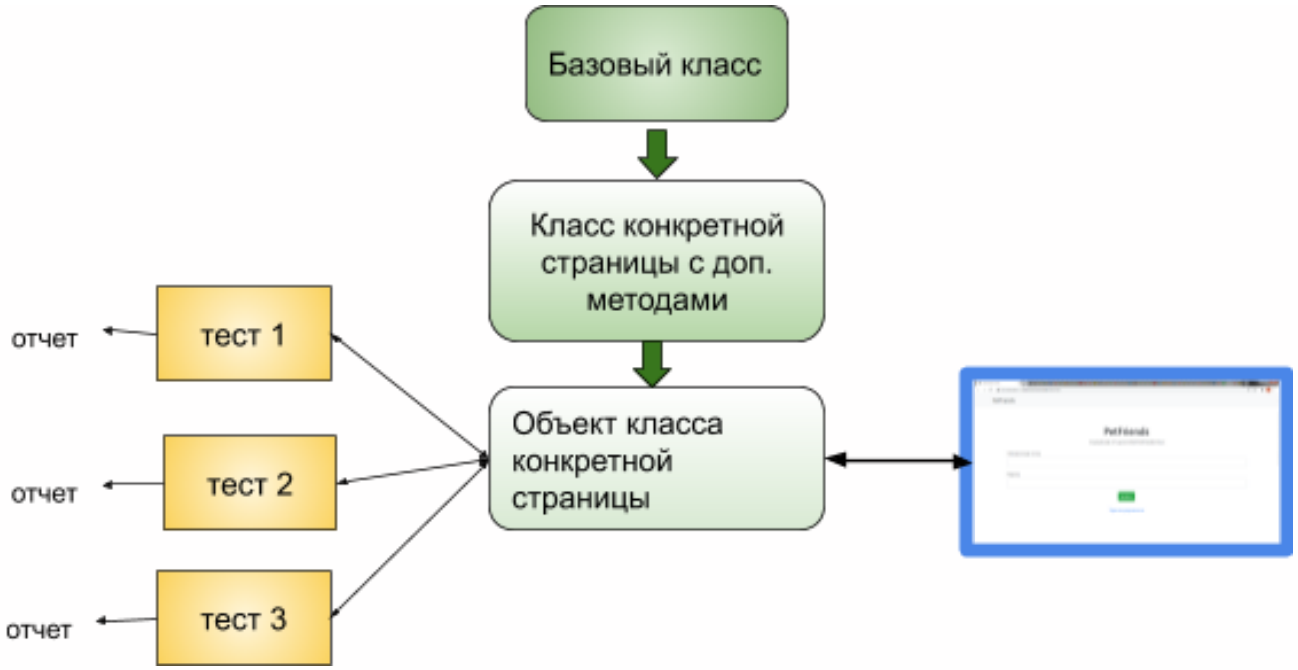


Схема реализации *PageObject* с базовым классом

Точно по такой же логике происходит создание объектов веб-элементов. Для них создаётся базовый класс, который может содержать полезные методы для большинства элементов. Затем создаются классы для конкретных элементов.

Методы объектов *PageObject* могут возвращать другие объекты *PageObject*, тем самым имитируя работу веб-приложения. Так при заполнении формы на странице авторизации мы переходим к объекту страницы личного кабинета.

При этом необходимо не забывать обо всех возможных путях развития сценария. Например, при вводе неверного пароля мы переходим на страницу ошибки, или же можем перейти на страницу восстановления пароля.

Локаторы тоже претерпят изменения. Так как при изменении *HTML*-элементов на странице мы хотим вносить изменения лишь в одном месте кода, нужно поместить в переменную локатор, по

которому ищется элемент. Но ещё удобнее оперировать переменными, в которых сразу содержится информация, и как искать (например, искать по *id*), и что искать (my_id).

Получается, что логически сама строка локатора неотделима от типа поиска, ведь искать по *id* мы можем лишь подходящими для этого методами (find_element_by_id или find_element(by='id', 'text')). Тут нам пригодится метод find_element, в который удобно подставлять нашу переменную со списком параметров!

Допустим, у нас на странице есть такое текстовое поле:

```
<input name=search_string id="search_input">
```

Тогда в нашем коде мы реализуем следующий объединённый локатор:

```
SEARCH_FIELD = (By.ID, "search_input")
```

Как мы убедимся в следующих уроках, ещё одним преимуществом паттерна *PageObject* является лаконичность и человеко-ориентированность кода тестов:

```
page = AuthPage()
page.login = "email@mail.ru"
page.password = "secret"
page.login_button.click()
```

Синтаксис команд зависит от выбранной библиотеки, но суть остается одинаковой. Написание тестов становится более лёгким и быстрым, что особенно важно при большом их количестве.

Задание 26.1.2

1/1 point (graded)

Выберите верные утверждения:

- [ ] *PageObject* — это *Python*-библиотека.
- [x] *PageObject* упрощает чтение и поддержку кода.
- [ ] Без *PageObject* нельзя писать автотесты.
- [x] В *PageObject* разделяется логика тестов и работа с реализацией веб-элементов на странице.
- [ ] *PageObject* не использует ООП.

✔

Отправить

✔ Correct (1/1 point)

Задание 26.1.3

1/1 point (graded)

Выберите верное утверждение:

- В тестах мы используем базовый класс страниц.
- В тестах мы непосредственно обращаемся к методам класса страницы.
- В тестах мы создаем объект класса страницы и работаем с ним. ✔

Отправить

✔ Correct (1/1 point)