PageObjects

Jump to bottom

Simon Stewart edited this page on 12 Mar 2015 · 1 revision

Page Objects

Within your web app's UI there are areas that your tests interact with. A Page Object simply models these as objects within the test code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place.

Implementation Notes

PageObjects can be thought of as facing in two directions simultaneously. Facing towards the developer of a test, they represent the **services** offered by a particular page. Facing away from the developer, they should be the only thing that has a deep knowledge of the structure of the HTML of a page (or part of a page) It's simplest to think of the methods on a Page Object as offering the "services" that a page offers rather than exposing the details and mechanics of the page. As an example, think of the inbox of any web-based email system. Amongst the services that it offers are typically the ability to compose a new email, to choose to read a single email, and to list the subject lines of the emails in the inbox. How these are implemented shouldn't matter to the test.

Because we're encouraging the developer of a test to try and think about the services that they're interacting with rather than the implementation, PageObjects should seldom expose the underlying WebDriver instance. To facilitate this, methods on the PageObject should return other PageObjects. This means that we can effectively model the user's journey through our application. It also means that should the way that pages relate to one another change (like when the login page asks the user to change their password the first time they log into a service, when it previously didn't do that) simply changing the appropriate method's signature will cause the tests to fail to compile. Put another way, we can tell which tests would fail without needing to run them when we change the relationship between pages and reflect this in the PageObjects.

One consequence of this approach is that it may be necessary to model (for example) both a successful and unsuccessful login, or a click could have a different result depending on the state of the app. When this happens, it is common to have multiple methods on the PageObject:

```
public class LoginPage {
    public HomePage loginAs(String username, String password) {
        // ... clever magic happens here
    }

    public LoginPage loginAsExpectingError(String username, String password) {
        //  ... failed login here, maybe because one or both of the username and
password are wrong
    }

    public String getErrorMessage() {
        // So we can verify that the correct error is shown
    }
}
```

The code presented above shows an important point: the tests, not the PageObjects, should be responsible for making assertions about the state of a page. For example:

```
public void testMessagesAreReadOrUnread() {
    Inbox inbox = new Inbox(driver);
    inbox.assertMessageWithSubjectIsUnread("I like cheese");
    inbox.assertMessageWithSubjectIsNotUnread("I'm not fond of tofu");
}
```

could be re-written as:

```
public void testMessagesAreReadOrUnread() {
    Inbox inbox = new Inbox(driver);
    assertTrue(inbox.isMessageWithSubjectIsUnread("I like cheese"));
    assertFalse(inbox.isMessageWithSubjectIsUnread("I'm not fond of tofu"));
}
```

Of course, as with every guideline there are exceptions, and one that is commonly seen with PageObjects is to check that the WebDriver is on the correct page when we instantiate the PageObject. This is done in the example below.

Finally, a PageObject need not represent an entire page. It may represent a section that appears many times within a site or page, such as site navigation. The essential principle is that there is only one place in your test suite with knowledge of the structure of the HTML of a particular (part of a) page.

Summary

- The public methods represent the services that the page offers
- Try not to expose the internals of the page
- Generally don't make assertions
- Methods return other PageObjects
- Need not represent an entire page
- Different results for the same action are modelled as different methods

Example

```
public class LoginPage {
    private final WebDriver driver;

    public LoginPage(WebDriver driver) {
        this.driver = driver;

        // Check that we're on the right page.
        if (!"Login".equals(driver.getTitle())) {
            // Alternatively, we could navigate to the login page, perhaps logging out
first
            throw new IllegalStateException("This is not the login page");
        }
    }

    // The login page contains several HTML elements that will be represented as
WebElements.
    // The locators for these elements should only be defined once.
        By usernameLocator = By.id("username");
        By passwordLocator = By.id("passwd");
        By loginButtonLocator = By.id("login");

    // The login page allows the user to type their username into the username field
    public LoginPage typeUsername(String username) {
        // This is the only place that "knows" how to enter a username
        driver.findElement(usernameLocator).sendKeys(username);

        // Return the current page object as this action doesn't navigate to a page
represented by another PageObject
        return this;
    }

    // The login page allows the user to type their password into the password field
    public LoginPage typePassword(String password) {
        // This is the only place that "knows" how to enter a password
        driver.findElement(passwordLocator).sendKeys(password);

        // Return the current page object as this action doesn't navigate to a page
represented by another PageObject
        return this;
    }

    // The login page allows the user to submit the login form
    public HomePage submitLogin() {
        // This is the only place that submits the login form and expects the
destination to be the home page.
        // A seperate method should be created for the instance of clicking login
whilst expecting a login failure.
        driver.findElement(loginButtonLocator).submit();

        // Return a new page object representing the destination. Should the login page
ever
        // go somewhere else (for example, a legal disclaimer) then changing the method
signature
        // for this method will mean that all tests that rely on this behaviour won't
compile.
        return new HomePage(driver);
    }

    // The login page allows the user to submit the login form knowing that an invalid
username and / or password were entered
    public LoginPage submitLoginExpectingFailure() {
        // This is the only place that submits the login form and expects the
destination to be the login page due to login failure.
        driver.findElement(loginButtonLocator).submit();

        // Return a new page object representing the destination. Should the user ever
be navigated to the home page after submiting a login with credentials
        // expected to fail login, the script will fail when it attempts to instantiate
the LoginPage PageObject.
        return new LoginPage(driver);
    }

    // Conceptually, the login page offers the user the service of being able to "log
into"
    // the application using a user name and password.
    public HomePage loginAs(String username, String password) {
        // The PageObject methods that enter username, password & submit login have
already defined and should not be repeated here.
        typeUsername(username);
        typePassword(password);
        return submitLogin();
    }
```

```
  }
```

Support in WebDriver

There is a PageFactory in the support package that provides support for this pattern, and helps to remove some boiler-plate code from your Page Objects at the same time.

Clone this wiki locally

```
https://github.com/SeleniumHQ/selenium.wiki.git
```