**[PageObject](http://martinfowler.com/bliki/PageObject.html)**

[*Martin Fowler*](http://martinfowler.com/)

*10 September 2013*

When you write tests against a web page, you need to refer to elements within that web page in order to click links and determine what's displayed. However, if you write tests that manipulate the HTML elements directly your tests will be brittle to changes in the UI. A page object wraps an HTML page, or fragment, with an application-specific API, allowing you to manipulate page elements without digging around in the HTML.

The basic rule of thumb for a page object is that it should allow a software client to do anything and see anything that a human can. It should also provide an interface that's easy to program to and hides the underlying widgetry in the window. So to access a text field you should have accessor methods that take and return a string, check boxes should use booleans, and buttons should be represented by action oriented method names. The page object should encapsulate the mechanics required to find and manipulate the data in the gui control itself. A good rule of thumb is to imagine changing the concrete control - in which case the page object interface shouldn't change.

Despite the term "page" object, these objects shouldn't usually be built for each page, but rather for the significant elements on a page [[1]](http://martinfowler.com/bliki/PageObject.html#footnote-panel-object). So a page showing multiple albums would have an album list page object containing several album page objects. There would probably also be a header page object and a footer page object. That said, some of the hierarchy of a complex UI is only there in order to structure the UI - such composite structures shouldn't be revealed by the page objects. The rule of thumb is to model the structure in the page that makes sense to the user of the application.

Similarly if you navigate to another page, the initial page object should return another page object for the new page [[2]](http://martinfowler.com/bliki/PageObject.html#footnote-return-type). In general page object operations should return fundamental types (strings, dates) or other page objects.

There are differences of opinion on whether page objects should include assertions themselves, or just provide data for test scripts to do the assertions. Advocates of including assertions in page objects say that this helps avoid duplication of assertions in test scripts, makes it easier to provide better error messages, and supports a more [TellDontAsk](http://martinfowler.com/bliki/TellDontAsk.html) style API. Advocates of assertion-free page objects say that including assertions mixes the responsibilities of providing access to page data with assertion logic, and leads to a bloated page object.

I favor having no assertions in page objects. I think you can avoid duplication by providing assertion libraries for common assertions - which can also make it easier to provide good diagnostics. [[3]](http://martinfowler.com/bliki/PageObject.html#footnote-assertions)

Page objects are commonly used for testing, but should not make assertions themselves. Their responsibility is to provide access to the state of the underlying page. It's up to test clients to carry out the assertion logic.

I've described this pattern in terms of HTML, but the same pattern applies equally well to any UI technology. I've seen this pattern used effectively to hide the details of a Java swing UI and I've no doubt it's been widely used with just about every other UI framework out there too.

Concurrency issues are another topic that a page object can encapsulate. This may involve hiding the asynchrony in async operations that don't appear to the user as async. It may also involve encapsulating threading issues in UI frameworks where you have to worry about allocating behavior between UI and worker threads.

Page objects are most commonly used in testing, but can also be used to provide a scripting interface on top of an application. Usually it's best to put a scripting interface underneath the UI, that's usually less complicated and faster. However with an application that's put too much behavior into the UI then using page objects may make the best of a bad job. (But look to move that logic if you can, it will be better both for scripting and the long term health of the UI.)

It's common to write tests using some form of [DomainSpecificLanguage](http://martinfowler.com/bliki/DomainSpecificLanguage.html), such as Cucumber or an internal DSL. If you do this it's best to layer the testing DSL over the page objects so that you have a parser that translates DSL statements into calls on the page object.

*If you have WebDriver APIs in your test methods, You're Doing It Wrong. --*[*Simon Stewart*](http://blog.rocketpoweredjetpants.com/)*.*

Patterns that aim to move logic out of UI elements (such as [Presentation Model](http://martinfowler.com/eaaDev/PresentationModel.html), [Supervising Controller](http://martinfowler.com/eaaDev/SupervisingPresenter.html), and [Passive View](http://martinfowler.com/eaaDev/PassiveScreen.html)) make it less useful to test through the UI and thus reduce the need for page objects.

Page objects are a classic example of encapsulation - they hide the details of the UI structure and widgetry from other components (the tests). It's a good design principle to look for situations like this as you develop - ask yourself "how can I hide some details from the rest of the software?" As with any encapsulation this yields two benefits. I've already stressed that by confining logic that manipulates the UI to a single place you can modify it there without affecting other components in the system. A consequential benefit is that it makes the client (test) code easier to understand because the logic there is about the intention of the test and not cluttered by UI details.

**Further Reading**

I first described this pattern under the name [Window Driver](http://martinfowler.com/eaaDev/WindowDriver.html). However since then the term "page object" was popularized by the Selenium web testing framework and that's become the generally used name.

[Selenium's wiki](https://github.com/SeleniumHQ/selenium/wiki/PageObjects) strongly encourages using page objects and provides advice on how they should be used. It also favors assertion-free page objects.

A team measured the times to update two versions of a suite of selenium tests after a software upgrade. They found the version with page object took a bit longer for the first test case, but much faster for the rest. For more details see [Leotta et al, "Improving test suites maintainability with the page object pattern", ICSTW 2013](http://scholar.google.com/citations?view_op=view_citation&hl=en&user=n010PrAAAAAJ&citation_for_view=n010PrAAAAAJ:roLk4NBRz8UC)

**Acknowledgements**

Perryn Fowler, Pete Hodgson, and Simon Stewart gave particularly useful comments on drafts of this post - although as usual I owe much to various denizens of ThoughtWorks's internal software development list for their suggestions and corrections.

**Notes**

**1:**There's an argument here that the name "page object" is misleading because it makes you think you should have just one page object per page. Something like "panel object" would be better - but the term "page object" is what's become accepted. Another illustration of why naming is one of the [TwoHardThings](http://martinfowler.com/bliki/TwoHardThings.html).

**2:**Having page objects be responsible for creating other page objects in response to things like navigation is common advice. However some practitioners prefer that page objects return some generic browser context, and the tests control which page objects to build on top of that context based on the flow of the test (particularly conditional flows). Their preference is based on the fact that the test script knows what pages are expected next and this knowledge doesn't need to be duplicated in the page objects themselves. They increase their preference when using statically typed languages which usually reveal page navigations in type signatures.

**3:**One form of assertions is fine even for people like me who generally favor a no-assertion style. These assertions are those that check the invariants of a page or the application at this point, rather than specific things that a test is probing.