6. Page Objects

This chapter is a tutorial introduction to the Page Objects design pattern. A page object represents an area where the test interacts within the web application user interface.

Benefits of using page object pattern:

- Easy to read test cases
- Creating reusable code that can share across multiple test cases
- Reducing the amount of duplicated code
- If the user interface changes, the fix needs changes in only one place

6.1. Test case

Here is a test case that searches for a word on the *python.org* website and ensures some results. The following section will introduce the *page* module where the page objects will be defined.

```
import unittest
from selenium import webdriver
import page

class PythonOrgSearch(unittest.TestCase):
    """A sample test class to show how page object works"""

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://www.python.org")

    def test_search_in_python_org(self):
        """Tests python.org search feature. Searches for the word "pycon" then
        verified that some results show up.  Note that it does not look for
        any particular text in search results page. This test verifies that
        the results were not empty."""

        #Load the main page. In this case the home page of Python.org.
        main_page = page.MainPage(self.driver)
        #Checks if the word "Python" is in title
        assert main_page.is_title_matches(), "python.org title doesn't match."
        #Sets the text of search textbox to "pycon"
        main_page.search_text_element = "pycon"
        main_page.click_go_button()
        search_results_page = page.SearchResultsPage(self.driver)
        #Verifies that the results page is not empty
        assert search_results_page.is_results_found(), "No results found."

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

6.2. Page object classes

The page object pattern intends to create an object for each part of a web page. This technique helps build a separation between the test code and the actual code that interacts with the web page.

The `page.py` will look like this:

```
from element import BasePageElement
from locators import MainPageLocators

class SearchTextElement(BasePageElement):
    """This class gets the search text from the specified locator"""

    #The locator for search box where search string is entered
    locator = 'q'


class BasePage(object):
    """Base class to initialize the base page that will be called from all
    pages"""

    def __init__(self, driver):
        self.driver = driver


class MainPage(BasePage):
    """Home page action methods come here. I.e. Python.org"""

    #Declares a variable that will contain the retrieved text
    search_text_element = SearchTextElement()

    def is_title_matches(self):
        """Verifies that the hardcoded text "Python" appears in page title"""

        return "Python" in self.driver.title

    def click_go_button(self):
        """Triggers the search"""

        element = self.driver.find_element(*MainPageLocators.GO_BUTTON)
        element.click()


class SearchResultsPage(BasePage):
    """Search results page action methods come here"""

    def is_results_found(self):
        # Probably should search for this text in the specific page
        # element, but as for now it works fine
        return "No results found." not in self.driver.page_source
```

6.3. Page elements

The `element.py` will look like this:

```
from selenium.webdriver.support.ui import WebDriverWait


class BasePageElement(object):
    """Base page class that is initialized on every page object class."""

    def __set__(self, obj, value):
        """Sets the text to the value supplied"""

        driver = obj.driver
        WebDriverWait(driver, 100).until(
            lambda driver: driver.find_element_by_name(self.locator))
        driver.find_element_by_name(self.locator).clear()
        driver.find_element_by_name(self.locator).send_keys(value)
```

```
    def __get__(self, obj, owner):
        """Gets the text of the specified object"""

        driver = obj.driver
        WebDriverWait(driver, 100).until(
            lambda driver: driver.find_element_by_name(self.locator))
        element = driver.find_element_by_name(self.locator)
        return element.get_attribute("value")
```

6.4. Locators

One of the practices is to separate the locator strings from the place where they are getting used. In this example, locators of the same page belong to the same class.

The `locators.py` will look like this:

```
from selenium.webdriver.common.by import By

class MainPageLocators(object):
    """A class for main page locators. All main page locators should come here"""

    GO_BUTTON = (By.ID, 'submit')

class SearchResultsPageLocators(object):
    """A class for search results locators. All search results locators should
    come here"""

    pass
```