Page Object Model(POM)

Deepak Varma <deepak.varma@citiustech.com>

Tue 10/22/2019 4:44 PM

To: Deepak Varma <deepak.varma@citiustech.com>

https://www.geeksforgeeks.org/page-object-model-pom/

```
package test_cases;

import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeSuite;

public class TestBase {

    public static WebDriver driver = null;

    @BeforeSuite
    public void initialize() throws IOException {

        System.setProperty("webdriver.chrome.driver",
        System.getProperty("user.dir") +
            "\\src\\test\\java\\drivers\\chromedriver.exe");

        driver = new ChromeDriver();

        // To maximize browser
        driver.manage().window().maximize();

        // Implicit wait
        driver.manage().timeouts().implicitlyWait(
             10, TimeUnit.SECONDS);

        // To open Gmail site
        driver.get("[https:]https:// www.gmail.com");
    }

    @AfterSuite
    // Test cleanup
    public void TeardownTest() {
        TestBase.driver.quit();
    }
}
```

```
public class GmailLoginTest extends TestBase {

    @Test
    public void init() throws Exception {

        // driver.get("[https:]https:// www.gmail.com");
        GmailLoginPage loginpage =
           PageFactory.initElements(driver,
           GmailLoginPage.class);

        loginpage.setEmail("abc@gmail.com");
        loginpage.clickOnNextButton();
        loginpage.setPassword("23456@qwe");
        loginpage.clickOnNextButton();
    }
}
```

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;

public class GmailLoginPage {

    WebDriver driver;

    public GmailLoginPage(WebDriver driver) {
        this.driver = driver;
    }

    // Using FindBy for locating elements
    @FindBy(how = How.XPATH, using =
            "// input[@type='email']]")
    WebElement emailTextBox;

    @FindBy(xpath = "// input[@type='password']")

    WebElement passwordTextBox;
    @FindBy(how = How.XPATH, using =
            "// div[@role = 'button' and @id  =
            'identifierNext']")
    WebElement nextButton;


    // Defining all the user actions (Methods)
```

```
    //that can be performed in the Facebook home page

    // This method is to set Email in the email text box
    public void setEmail(String strEmail) {
            emailTextBox.sendKeys(strEmail);
        }
        // This method is to set Password in the password text box
    public void setPassword(String strPassword) {
            passwordTextBox.sendKeys(strPassword);
        }
        // This method is to click on Next Button
    public void clickOnNextButton() {
        nextButton.click();
    }
}
```

https://www.toptal.com/selenium/test-automation-in-selenium-using-page-object-model-and-page-factory

Advantages of POM model:

Maintainability, Re-usability, Readability

we can easily integrate page classes with other frameworks /tools like JUnit/NUnit/PhpUnit as well as with TestNG/Cucumber/etc.

we will use Page Factory to initialize web elements that are defined in web page classes or Page Objects.

Web page classes or Page Objects containing web elements need to be initialized using Page Factory before the web element variables can be used. This can be done simply through the use of *initElements* function on PageFactory:

```
LoginPage page = new LoginPage(driver);
PageFactory.initElements(driver, page);
```

Or, even simpler:

```
LoginPage page = PageFactory.intElements(driver,LoginPage.class)
```

Or, inside the web page class constructor:

```
public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
}
```

Page Factory will initialize every *WebElement* variable with a reference to a corresponding element on the actual web page based on configured “locators”.

//if initElements is called in page class constructor itself we canwrite test script as below:

```
  @Test
   public void applyAsDeveloper() {
       //Create object of HomePage Class
       HomePage home = new HomePage(driver);
       home.clickOnDeveloperApplyButton();}
```

```
@FindBy(how=How.CSS,using="div[class='yt-lockup-tile yt-lockup-video']")
private List<WebElement> videoElements;
```

```
@FindBy(id="username")
private WebElement user_name;
```

```
   user_name.sendKeys(text);
```

is equivalent to:

```
      driver.findElement(By.id("user_name")).sendKeys(text);
```

-->WebElements are evaluated lazily. That is, if you never use a WebElement field in a PageObject, there will never be a call to "findElement" for it.

Advantages of Page Object Model:
-According to Page Object Model, we should keep our tests and element locators separately, this will keep code clean and easy to understand and maintain.
-The Page Object approach makes test automation framework programmer friendly, more durable and comprehensive.

-Another important advantage is our Page Object Repository is Independent of Automation Tests. Keeping separate repository for page objects helps us to use this repository for different purposes with different frameworks like, we are able to integrate this repository with other tools like JUnit/NUnit/PhpUnit as well as with TestNG/Cucumber/etc.

-Test cases become short and optimized as we are able to reuse page object methods in the POM classes.
-Any change in UI can easily be implemented, updated and maintained into the Page Objects and Classes.

java.lang.Class

Class <ABC> obj = ABC.class;
Class c = int.class; or Class<Integer> c = int.class;

this statement says: return an instance of Class of type ABC.

Java provides a class with name **Class** in java.lang package. Instances of the class Class represent classes and interfaces in a running Java application.