Page Object Model and Page Factory in Selenium

Page Object Model and Page Factory are tools in [Selenium](https://www.browserstack.com/selenium) that are popularly used in test automation. This tutorial will demonstrate how to make use of Page Object Model and Page Factory in automation projects in order to maintain test cases easily.

What is Page Object Model in Selenium?

Page Object Model, also known as POM is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance.

In Page Object Model, consider each web page of an application as a class file. Each class file will contain only corresponding web page elements. Using these elements, testers can perform operations on the website under test.

**Advantages of Page Object Model**

* **Helps with easy maintenance**: POM is useful when there is a change in a UI element or there is a change in an action. An example would be if a drop down menu is changed to a radio button.

In this case, POM helps to identify the page or screen to be modified. As every screen will have different java files, this identification is necessary to make the required changes in the right files. This makes test cases easy to maintain and reduces errors.

* **Helps with reusing code**: As already discussed, all screens are independent. By using POM, one can use the test code for any one screen, and reuse it in another test case. There is no need to rewrite code, thus saving time and effort.
* **Readability and Reliability of scripts**: When all screens have independent java files, one can easily identify actions that will be performed on a particular screen by navigating through the java file. If a change must be made to a certain section of code, it can be efficiently done without affecting other files.

Implementing POM in Selenium Project

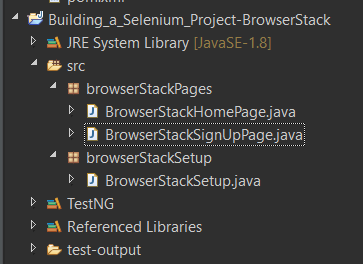
As already discussed, each java class will contain a corresponding page file. This tutorial will create 2-page files.

* BrowserStackHomePage
* BrowserStackSignUpPage

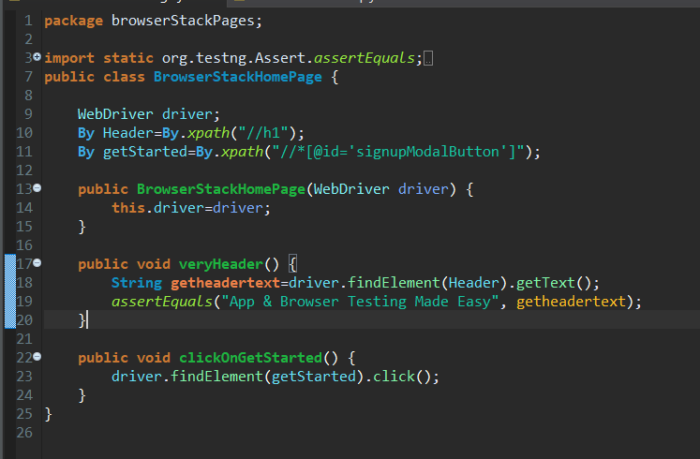
Each of these files will contain UI elements or Objects which are present on these screens. It will also contain the operations to be performed on these elements.

Also Read: [How to Build and Execute Selenium Projects](https://www.browserstack.com/guide/build-and-execute-selenium-projects)

Sample Project Structure for POM



**BrowserStackHomePage Java File**

  
**Explanation of Code**

* **Code Line-10 to 11:** Identifying elements present on BrowserStack Home Page such as header and Get Started button
* **Code Line-17 to 24:** Performing actions on identified objects on BrowserStack Home Page

**Code Snippet:**

package browserStackPages;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class BrowserStackHomePage {

WebDriver driver;

By Header=By.xpath("//h1");

By getStarted=By.xpath("//\*[@id='signupModalButton']");

public BrowserStackHomePage(WebDriver driver) {

this.driver=driver;

}

public void veryHeader() {

String getheadertext=driver.findElement(Header).getText();

assertEquals("App & Browser Testing Made Easy", getheadertext);

}

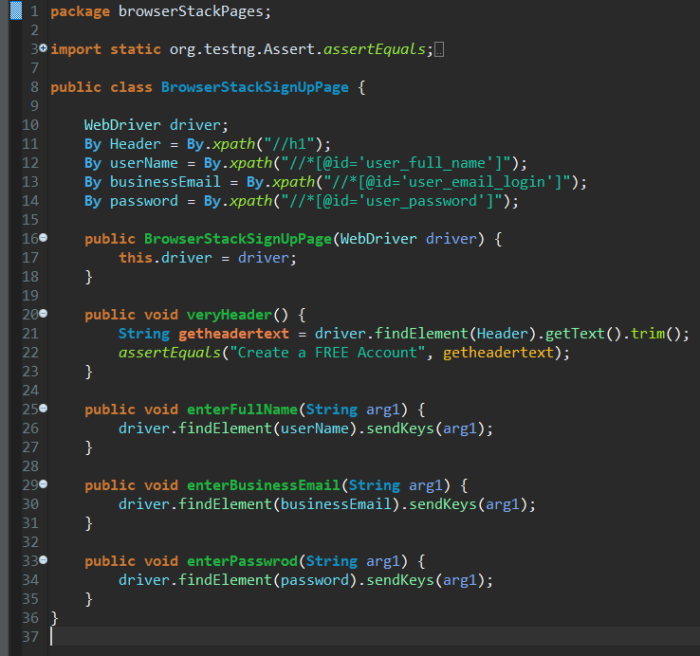
public void clickOnGetStarted() {

driver.findElement(getStarted).click();

}

}

**BrowserStackSignUpPage Java File**



**Explanation of Code**

* **Code Line-10 to 14:** Identifying elements present on BrowserStack SignUp Page such as header and Get Started button
* **Code Line-20 to 35:** Performing actions on identified objects on BrowserStack SignUp Page

Code Snippet:

package browserStackPages;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class BrowserStackSignUpPage {

WebDriver driver;

By Header = By.xpath("//h1");

By userName = By.xpath("//\*[@id='user\_full\_name']");

By businessEmail = By.xpath("//\*[@id='user\_email\_login']");

By password = By.xpath("//\*[@id='user\_password']");

public BrowserStackSignUpPage(WebDriver driver) {

this.driver = driver;

}

public void veryHeader() {

String getheadertext = driver.findElement(Header).getText().trim();

assertEquals("Create a FREE Account", getheadertext);

}

public void enterFullName(String arg1) {

driver.findElement(userName).sendKeys(arg1);

}

public void enterBusinessEmail(String arg1) {

driver.findElement(businessEmail).sendKeys(arg1);

}

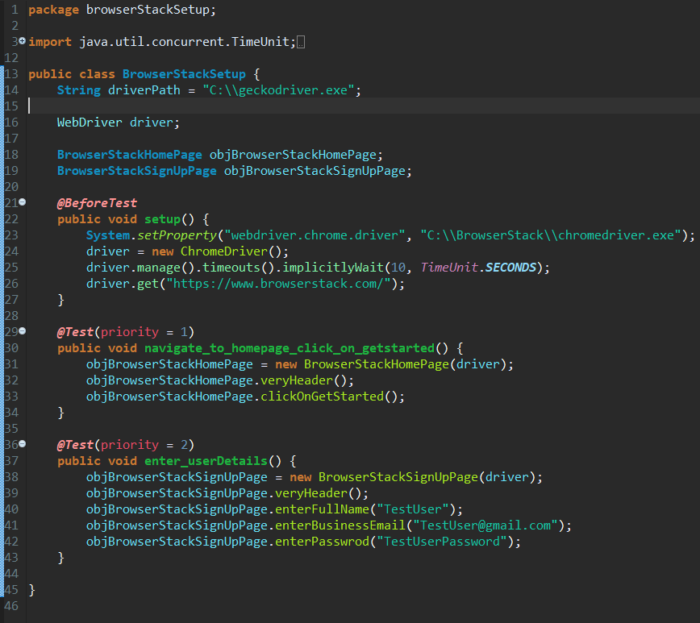
public void enterPasswrod(String arg1) {

driver.findElement(password).sendKeys(arg1);

}

}

**BrowserStackSetup Java File**



**Explanation of Code**

* **Code Line-21 to 27:** Setting up of browser and website to execute test scripts
* **Code Line-29 to 43:** Initializing driver object to BrowserStackHomePage & BrowserStackSignUpPage and performing actions on those pages

**Code Snippet:**

package browserStackSetup;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import browserStackPages.BrowserStackHomePage;

import browserStackPages.BrowserStackSignUpPage;

public class BrowserStackSetup {

String driverPath = "C:\\geckodriver.exe";

WebDriver driver;

BrowserStackHomePage objBrowserStackHomePage;

BrowserStackSignUpPage objBrowserStackSignUpPage;

@BeforeTest

public void setup() {

System.setProperty("webdriver.chrome.driver", "C:\\BrowserStack\\chromedriver.exe");

driver = new ChromeDriver();

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

driver.get("https://www.browserstack.com/");

}

@Test(priority = 1)

public void navigate\_to\_homepage\_click\_on\_getstarted() {

objBrowserStackHomePage = new BrowserStackHomePage(driver);

objBrowserStackHomePage.veryHeader();

objBrowserStackHomePage.clickOnGetStarted();

}

@Test(priority = 2)

public void enter\_userDetails() {

objBrowserStackSignUpPage = new BrowserStackSignUpPage(driver);

objBrowserStackSignUpPage.veryHeader();

objBrowserStackSignUpPage.enterFullName("TestUser");

objBrowserStackSignUpPage.enterBusinessEmail("TestUser@gmail.com");

objBrowserStackSignUpPage.enterPasswrod("TestUserPassword");

}

}

[Run Selenium Tests for Free](https://www.browserstack.com/users/sign_up?ref=guide-page-object-model-in-selenium-mid&product=automate)

What is Page Factory in Selenium?

Page Factory is a class provided by[Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial) to support Page Object Design patterns. In Page Factory, testers use @FindBy annotation. The initElements method is used to initialize web elements.

* **@FindBy**: An annotation used in Page Factory to locate and declare web elements using different locators. Below is an example of declaring an element using @FindBy

@FindBy(id="elementId") WebElement element;

Similarly, one can use @FindBy with different location strategies to find the web elements and perform actions on them. Below are [locators](https://www.browserstack.com/guide/locators-in-selenium) that can be used:

* + - ClassName
    - Css
    - Name
    - Xpath
    - TagName
    - LinkText
    - PartialLinkText
* **initElements()**: **initElements** is a static method in Page Factory class. Using the initElements method, one can initialize all the web elements located by @FindBy annotation.
* **lazy initialization:** **AjaxElementLocatorFactory** is a lazy load concept in Page Factory. This is used to identify web elements only when they are used in any operation or activity. The timeout of a web element can be assigned to the object class with the help of the AjaxElementLocatorFactory.

Implementing Page Factory in Selenium Project

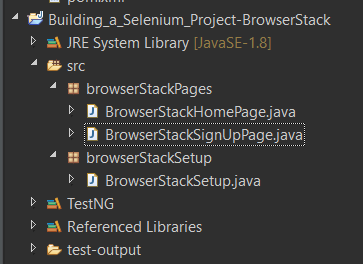
This will try to use the same project used for the POM Model. It will reuse the 2-page files and implement Page Factory.

* BrowserStackHomePage
* BrowserStackSignUpPage

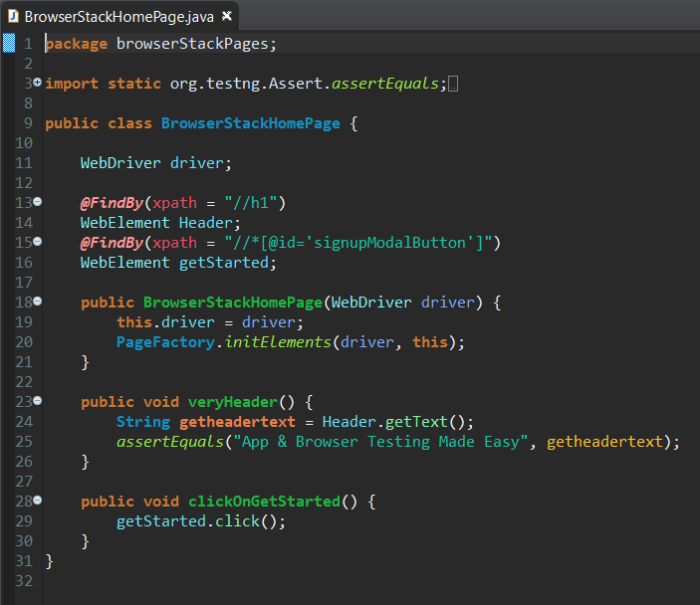
As discussed earlier, each of these files will only contain UI elements or Objects which are present on these screens along with the operations to be performed on these elements.

Sample Project Structure for Page Factory

The project structure will not be changing as the same project is being used. As already mentioned, Page Factory supports Page Object Model design pattern.



**BrowserStackHomePage Java File**

  
**Explanation of Code**  
**Code Line-13 to 17:** Identifying elements present on BrowserStack Home Page such as header and Get Started button using Page Factory @FindBy annotation

**Code Line-23 to 30:** Performing actions on identified objects on BrowserStack Home Page

**Code Snippet:**

package browserStackPages;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class BrowserStackHomePage {

WebDriver driver;

@FindBy(xpath = "//h1")

WebElement Header;

@FindBy(xpath = "//\*[@id='signupModalButton']")

WebElement getStarted;

public BrowserStackHomePage(WebDriver driver) {

this.driver = driver;

PageFactory.initElements(driver, this);

}

public void veryHeader() {

String getheadertext = Header.getText();

assertEquals("App & Browser Testing Made Easy", getheadertext);

}

public void clickOnGetStarted() {

getStarted.click();

}

}

**BrowserStackSignUpPage Java File**



**Explanation of Code**

**Code Line-14 to 24:** Identifying elements present on BrowserStack SignUp Page such as header and Get Started button using Page Factory @FindBy annotation.  
**Code Line-26 to 46:** Performing actions on identified objects on BrowserStack SignUp Page

**Code Snippet:**

package browserStackPages;

import static org.testng.Assert.assertEquals;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class BrowserStackSignUpPage {

WebDriver driver;

@FindBy(xpath = "//h1")

WebElement Header;

@FindBy(xpath = "//\*[@id='user\_full\_name']")

WebElement userName;

@FindBy(xpath = "//\*[@id='user\_email\_login']")

WebElement businessEmail;

@FindBy(xpath = "//\*[@id='user\_password']")

WebElement password;

public BrowserStackSignUpPage(WebDriver driver) {

this.driver = driver;

PageFactory.initElements(driver, this);

}

public void veryHeader() {

String getheadertext = Header.getText().trim();

assertEquals("Create a FREE Account", getheadertext);

}

public void enterFullName(String arg1) {

userName.sendKeys(arg1);

}

public void enterBusinessEmail(String arg1) {

businessEmail.sendKeys(arg1);

}

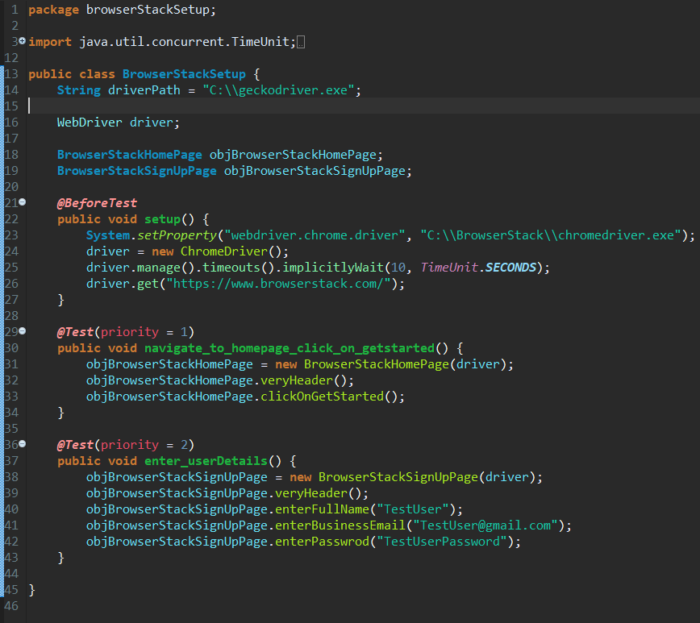
public void enterPasswrod(String arg1) {

password.sendKeys(arg1);

}

}

**BrowserStackSetup Java File**

  
**Explanation of Code**

* **Code Line-21 to 27:** Setting up of browser and website to execute our scripts
* **Code Line-29 to 43:** Initializing driver objects to BrowserStackHomePage & BrowserStackSignUpPage and performing actions on those pages

**Code Snippet:**

package browserStackSetup;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import browserStackPages.BrowserStackHomePage;

import browserStackPages.BrowserStackSignUpPage;

public class BrowserStackSetup {

String driverPath = "C:\\geckodriver.exe";

WebDriver driver;

BrowserStackHomePage objBrowserStackHomePage;

BrowserStackSignUpPage objBrowserStackSignUpPage;

@BeforeTest

public void setup() {

System.setProperty("webdriver.chrome.driver", "C:\\BrowserStack\\chromedriver.exe");

driver = new ChromeDriver();

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

driver.get("https://www.browserstack.com/");

}

@Test(priority = 1)

public void navigate\_to\_homepage\_click\_on\_getstarted() {

objBrowserStackHomePage = new BrowserStackHomePage(driver);

objBrowserStackHomePage.veryHeader();

objBrowserStackHomePage.clickOnGetStarted();

}

@Test(priority = 2)

public void enter\_userDetails() {

objBrowserStackSignUpPage = new BrowserStackSignUpPage(driver);

objBrowserStackSignUpPage.veryHeader();

objBrowserStackSignUpPage.enterFullName("TestUser");

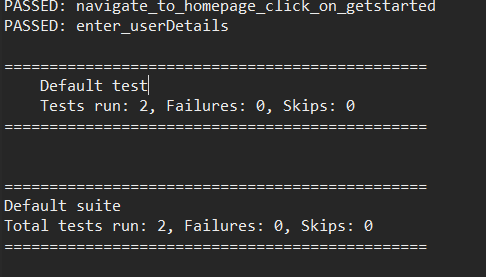
objBrowserStackSignUpPage.enterBusinessEmail("TestUser@gmail.com");

objBrowserStackSignUpPage.enterPasswrod("TestUserPassword");

}

}

**Test Result:**



[Try Selenium Testing on Real Devices for Free](https://www.browserstack.com/users/sign_up?ref=guide-page-object-model-in-selenium-bottom&product=automate)

Difference Between Page Object Model and Page Factory in Selenium

|  |  |
| --- | --- |
| **Page Object Model** | **Page Factory** |
| Finding web elements using **By** | Finding web elements using **@FindBy** |
| POM does not provide lazy initialization | Page Factory does provide lazy initialization |
| Page Object Model is a design pattern | PageFactory is a class which provides implementation of Page Object Model design pattern |
| In POM, one needs to initialize every page object individually | In PageFactory, all page objects are initialized by using the initElements() method |

Run the code in order to test the workings of Page Object Model and Page Factory. Since these are important Selenium functions, testers need to be able to use them with ease and accuracy for [Selenium automation](https://www.browserstack.com/selenium-automation). This will help them streamline [automation testing](https://www.browserstack.com/guide/automation-testing-tutorial) efforts and yield results quicker.