Verify login

Enter url

Driver.get()

Enter username

Driver.findelement.sendkeys

Enter password

Click on login

Verify login successful

**PAGE OBJECT MODEL and PAGE FACTORY**

Page Object Model, also known as POM, is a [design pattern in Selenium](https://www.browserstack.com/guide/design-patterns-in-automation-framework) that creates an object repository for storing all web elements. It helps reduce 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.

Page Factory is **a class provided by Selenium WebDriver 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.

Selenium Page Factory Pattern is like an extension to [Page Object Model](http://www.seleniumeasy.com/selenium-tutorials/page-object-model-framework-introduction), but Page Factory is much enhanced model. To start with, we just need to import package ‘org.openqa.selenium.support.PageFactory’

"Factory class can be used to make using Page Objects simpler and easier".

We use [Page Factory pattern](https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/PageFactory.html) to initialize web elements which are defined in Page Objects.

We should initialize page objects using initElements() method from PageFactory Class as below, Once we call initElements() method, all elements will get initialized.

PageFactory.initElements() static method takes the driver instance of the given class and the class type, and returns a Page Object with its fields fully initialized.

Home homePage = **new** HomePage(driver);

PageFactory.initElements(driver, homePage);

Or,

*// To initialize elements.*

HomePage homePage = PageFactory.initElements(driver, HomePage.**class**);

Or, **as a constructor for page class as below:**

**public** **HompePage**(WebDriver driver) {

**this**.driver = driver;

PageFactory.initElements(driver, **this**);

}

Constructor is invoked when object of the page is created in MainClass.java

**Steps To Create A POM Without Page Factory Model**

**#1)** Create a Java Class for every page in the application.

**#2)** In each Class, declare all the Web Elements as variable.

**#3)** Implement corresponding methods acting on the variables.

**The design pattern can be structured using 2 layers/packages:**

* **Page Layer** will hold the pages of the application as individual Java Class. Each Class will have WebElements declared as variables and the actions that are performed as methods.
* **Test Layer** will hold the test cases of the application and its verification part.

**Let’s take an example of a simple scenario:**

1. Open the URL of an application.
2. Type the Email Address and password data.
3. Click on the Login button.
4. Verify successful login message on the Search Page.

**Page Layer**

**Here we have 2 pages,**

* **HomePage**: The page opens when the URL is entered and this is where we enter the data for login.
* **SearchPage**: Page that gets displayed after a successful login.

In the Page layer, each page in the Web Application is declared as a separate Java Class and its locators and actions are mentioned there.

**Steps To Create POM With Real-Time Examples**

**#1) Create a Java Class for each page:**

In this **example**, we will access 2 web pages, “Home” and “Search” pages. Hence, we will create 2 Java classes in the Page Layer (or in a package say, com.automation.pages).

Package Name : com.automation.pages

HomePage.java

SearchPage.java

**#2) Create WebElements as variables:**

**We would be interacting with:**

* Email, Password, Login button field on the HomePage.
* A successful message in the SearchPage.

So we will create WebElements as variables using ‘By’ Class.

**For Example:** If Email has xpath as *//div[contains(@id, ‘EmailId’)],* then its variable declaration is

//Locator for EmailId field

*By EmailAddress = By.xpath(//div[contains(@id, ‘EmailId’)])*

**#3) Create methods for actions performed on WebElements:**

**Below actions are performed on WebElements:**

* Type action on EmailAddress field.
* Type action on the Password field.
* Click action on the Login button.

**For Example,** User-defined methods are created for each action on the WebElement as,

|  |
| --- |
| **public** **void** typeEmailId(String Id){  driver.findElement(EmailAddress).sendKeys(Id)  } |

Here, the Id is passed as a parameter in the method, as the input will be sent by the user from the main test case.

**Note**: A constructor has to be created for each class in the Page Layer to get the driver instance from the Main class in the Test Layer.

We do not initiate the driver here, rather its instance is received from the Main Class when the object of the Page Layer class is created.

**Two Java Classes are created for each page as shown below:**

**HomePage.java**

|  |
| --- |
| //package    com.automation.pages;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;    **public** **class** HomePage {        WebDriver driver;        // Locator for Email Address      By EmailAddress = By.xpath("//div[contains(@id,'Emaild')]");        // Locator for Password field      By Password= By.xpath("//div[contains(@id,'Password')]");        // Locator for SignIn Button      By SignInButton= By.xpath("//div[contains(@id,'SignInButton')]");          // Method to type EmailId  **public** **void** typeEmailId(String Id){          driver.findElement(EmailAddress).sendKeys(Id)      }        // Method to type Password  **public** **void** typePassword(String PasswordValue){          driver.findElement(Password).sendKeys(PasswordValue)      }        // Method to click SignIn Button  **public** **void** clickSignIn(){          driver.findElement(SignInButton).click()      }        // Constructor      // Gets called when object of this page is created in MainClass.java  **public** HomePage(WebDriver driver)      {          // "this" keyword is used here to distinguish global and local variable "driver"          //gets driver as parameter from MainClass.java and assigns to the driver instance in this class    **this**.driver=driver;    } |

**SearchPage.Java**

|  |
| --- |
| //package  com.automation.pages;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;    **public** **class** SearchPage{        WebDriver driver;        // Locator for Success Message      By SuccessMessage= By.xpath("//div[contains(@id,'Message')]");        // Method that return True or False depending on whether the message is displayed    **public** Boolean MessageDisplayed(){          Boolean status = driver.findElement(SuccessMessage).isDisplayed();  **return** status;      }          // Constructor      // This constructor is invoked when object of this page is created in MainClass.java    **public** SearchPage(WebDriver driver)      {          // "this" keyword is used here to distinguish global and local variable "driver"          //gets driver as parameter from MainClass.java and assigns to the driver instance in this class    **this**.driver=driver; |

### Test Layer

Test Cases are implemented in this class.

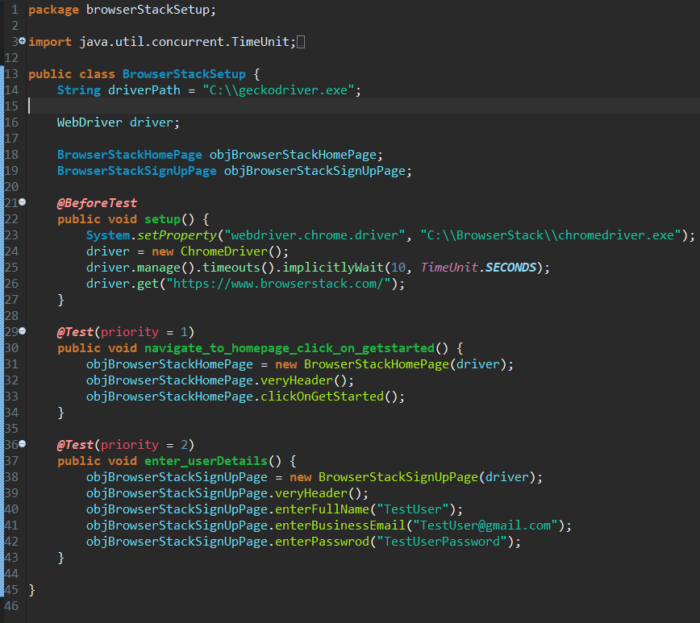
We create a separate package say, com.automation.test and then create a Java Class here (MainClass.java).

**Steps To Create Test Cases:**

1. Initialize the driver and open the application.
2. Create an object of the PageLayer Class(for each webpage) and pass the driver instance as a parameter.
3. Using the object created, make a call to the methods in the PageLayer Class(for each webpage) to perform actions/verification.
4. Repeat step 3 until all the actions are performed and then close the driver.

|  |
| --- |
| //package  com.automation.test;    **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.chrome.ChromeDriver;    **public** **class** MainClass {    **public** **static** **void** main(String[] args) {        System.setProperty("webdriver.chrome.driver","./exefiles/chromedriver.exe");      WebDriver driver= **new** ChromeDriver();      driver.manage().window().maximize();    driver.get("URL mentioned here");    // Creating object of HomePage and driver instance is passed as parameter to constructor of Homepage.Java          HomePage homePage= **new** HomePage(driver);    // Type EmailAddress      homePage.typeEmailId("abc@ymail.com");  // EmailId value is passed as paramter which in turn will be assigned to the method in HomePage.Java    // Type Password Value      homePage.typePassword("password123");  // Password value is passed as paramter which in turn will be assigned to the method in HomePage.Java    // Click on SignIn Button      homePage.clickSignIn();    // Creating an object of LoginPage and driver instance is passed as parameter to constructor of SearchPage.Java      SearchPage searchPage= **new** SearchPage(driver);    //Verify that Success Message is displayed      Assert.assertTrue(searchPage.MessageDisplayed());    //Quit browser  driver.quit();    }  } **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.  page factory in selenium  page factory in selenium - java file |





Difference between Page Object Model & 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 that implements the 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 |

References

<https://www.softwaretestinghelp.com/page-object-model-in-selenium-without-page-factory/>

<https://www.browserstack.com/guide/page-object-model-in-selenium#:~:text=Page%20Object%20Model%2C%20also%20known,application%20as%20a%20class%20file>.

<https://www.softwaretestingmaterial.com/page-object-model/>

Show examples

https://www.lambdatest.com/blog/page-object-model-tutorial-selenium-csharp/