# Page Object Model (POM) & Page Factory in Selenium: Ultimate Guide

Before we learn about Page Object Model, lets understand -

### Why POM ?

Starting a UI Automation in Selenium WebDriver is NOT a tough task. You just need to find elements, perform operations on it .

Consider this simple script to login into a website

```
public class NoPOMTest99GuruLogin {

    /**
     * This test case will login in http://demo.guru99.com/V4/
     * Verify login page title as guru99 bank
     * Login to application
     * Verify the home page using Dashboard message
     */
    @Test(priority=0)
    public void test_Home_Page_Appear_Correct(){
        WebDriver driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://demo.guru99.com/V4/");
        //Find user name and fill user name
        driver.findElement(By.name("uid")).sendKeys("demo");
        //find password and fill it
        driver.findElement(By.name("password")).sendKeys("password");
        //click login button
        driver.findElement(By.name("btnLogin")).click();
        String homeText = driver.findElement(By.xpath("//table//tr[@class='heading3']")).getText();
        //verify login success
        Assert.assertTrue(homeText.toLowerCase().contains("guru99 bank"));
    }
}
```

1 Find user name and fill it
2 Find password and fill it
3 Find Login button and click it
4 Find home page text and get it
5 Verify home page has text 'Guru99 Bank'

## Selenium Tu

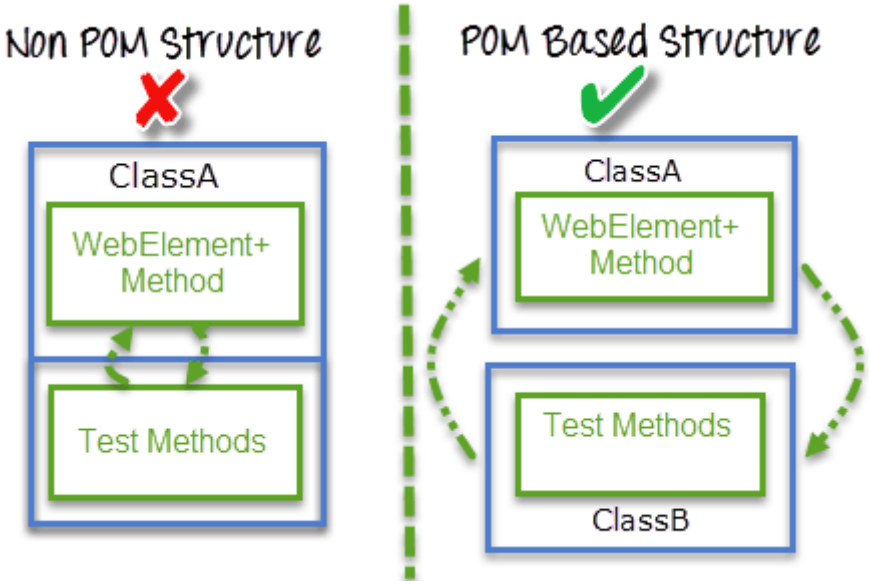
As you can observe, all we are doing is finding elements and filling values for those elements.

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

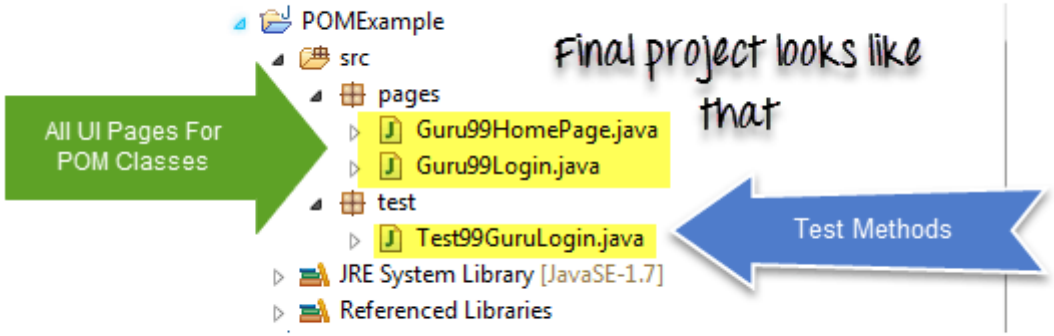
A better approach to script maintenance is to create a separate class file which would find web elements , fill them or verify them. This class can be reused in all the scripts using that element. In future if there is change in the web element , we need to make change in just 1 class file and not 10 different scripts.

This approach is called **Page Object Model(POM)**. It helps make code **more readable, maintainable**, and **reusable.**




## What is POM?

- **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
- Under this model, for each web page in the application there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Name of these methods should be given as per the task they are performing i.e., if a loader is waiting for payment gateway to be appear, POM method name can be waitForPaymentScreenDisplay().




## Advantages of POM

1. Page Object Patten says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. Second benefit is the **object repository is independent of testcases**, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods** get **more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

## How to implement POM ?

Simple POM:

It's the basic structure of Page object model (POM) where all Web Elements of the **AUT** and the method that operate on these Web Elements are maintained inside a class file. Task like **verification** should be **separate** as part of Test methods.

```
public class Guru99Login {

    WebDriver driver;
    By user99GuruName = By.name("uid");
    By password99Guru = By.name("password");
    By titleText =By.className("barone");
    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){
        this.driver = driver;
    }
    //Set user name in textbox
    public void setUserName(String strUserName){
        driver.findElement(user99GuruName).sendKeys(strUserName);
    }
```

Page class in object repository

Find Web Element

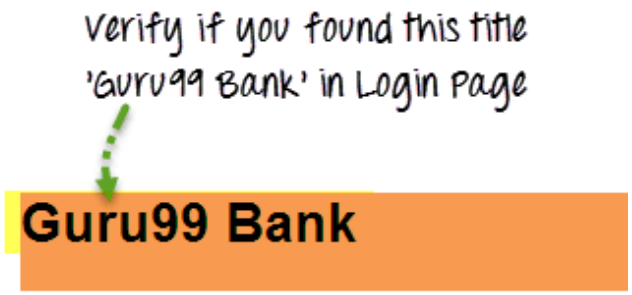
Performing operation on Web element

### Complete Example

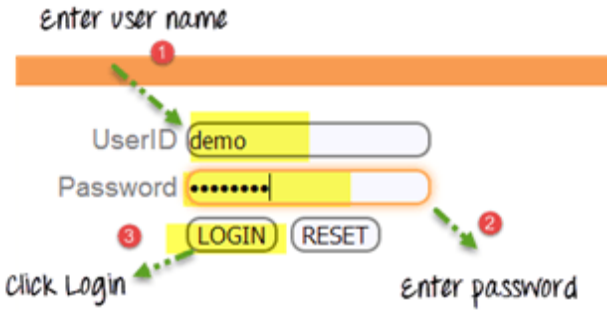
**TestCase:** Go to Guru99 Demo Site .

Step 1) Go to Guru99 Demo Site


Step 2) In home page check text **"Guru99 Bank"** is present




Step 3) Login into application




Step 4) Verify that the Home page contains text as "Manger Id : demo"


Here are we are dealing with 2 pages

1. Login Page
2. Home Page (shown once you login)

Accordingly we create 2 POM classes

## Guru99 Login page POM

```
package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class Guru99Login {

    WebDriver driver;

    By user99GuruName = By.name("uid");

    By password99Guru = By.name("password");

    By titleText =By.className("barone");

    By login = By.name("btnLogin");



    public Guru99Login(WebDriver driver){

        this.driver = driver;

    }

    //Set user name in textbox

    public void setUserName(String strUserName){

        driver.findElement(user99GuruName).sendKeys(strUserName);;

    }
```

```
//Set password in password textbox

public void setPassword(String strPassword){

     driver.findElement(password99Guru).sendKeys(strPassword);

}



//Click on login button

public void clickLogin(){

        driver.findElement(login).click();

}



//Get the title of Login Page

public String getLoginTitle(){

 return    driver.findElement(titleText).getText();

}

/**

 * This POM method will be exposed in test case to login in the application
```

```
     * @param strUserName

     * @param strPasword

     * @return

     */

    public void loginToGuru99(String strUserName,String strPasword){

        //Fill user name

        this.setUserName(strUserName);

        //Fill password

        this.setPassword(strPasword);

        //Click Login button

        this.clickLogin();



    }

}
```

## Guru99 Home Page POM

```
package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;
```

```
public class Guru99HomePage {

    WebDriver driver;

    By homePageUserName = By.xpath("//table//tr[@class='heading3']");



    public Guru99HomePage(WebDriver driver){

        this.driver = driver;

    }



    //Get the User name from Home Page

        public String getHomePageDashboardUserName(){

         return    driver.findElement(homePageUserName).getText();

        }

}
```

## Guru99 Simple POM Test case

```
package test;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import pages.Guru99HomePage;

import pages.Guru99Login;

public class Test99GuruLogin {

    WebDriver driver;

    Guru99Login objLogin;

    Guru99HomePage objHomePage;



    @BeforeTest

    public void setup(){

        driver = new FirefoxDriver();

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

        driver.get("http://demo.guru99.com/V4/");

    }

    /**
```

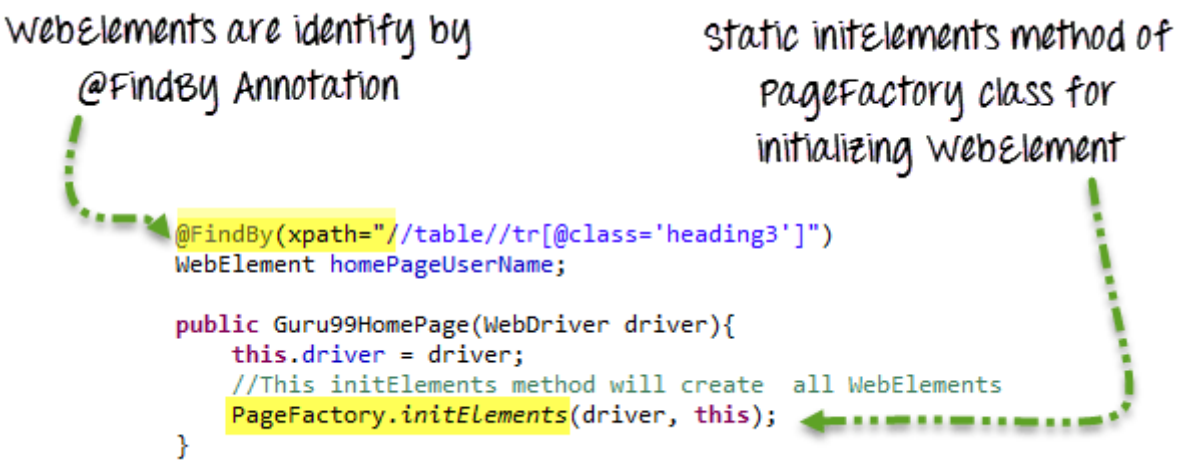
```
 * This test case will login in http://demo.guru99.com/V4/

 * Verify login page title as guru99 bank

 * Login to application

 * Verify the home page using Dashboard message

 */

@Test(priority=0)

public void test_Home_Page_Appear_Correct(){

    //Create Login Page object

objLogin = new Guru99Login(driver);

//Verify login page title

String loginPageTitle = objLogin.getLoginTitle();

Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));

//login to application

objLogin.loginToGuru99("mgr123", "mgr!23");

// go the next page

objHomePage = new Guru99HomePage(driver);

//Verify home page
```

```
    Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

    }
```

### What is Page Factory?

Page Factory is an inbuilt page object model concept for Selenium WebDriver but it is very optimized.

Here as well we follow the concept of separation of Page Object repository and Test methods. Additionally with the help of PageFactory class we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements




**@FindBy** can accept **tagName, partialLinkText, name, linkText, id, css, className, xpath** as attributes.

Let's look at the same example as above using Page Factory

### Guru99 Login page with Page Factory

```
package PageFactory;

import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class Guru99Login {

    /**

     * All WebElements are identified by @FindBy annotation

     */

    WebDriver driver;

    @FindBy(name="uid")

    WebElement user99GuruName;



    @FindBy(name="password")

    WebElement password99Guru;



    @FindBy(className="barone")

    WebElement titleText;
```

```
@FindBy(name="btnLogin")

WebElement login;



public Guru99Login(WebDriver driver){

    this.driver = driver;

    //This initElements method will create all WebElements

    PageFactory.initElements(driver, this);

}

//Set user name in textbox

public void setUserName(String strUserName){

    user99GuruName.sendKeys(strUserName);



}



//Set password in password textbox

public void setPassword(String strPassword){

password99Guru.sendKeys(strPassword);

}
```

```
//Click on login button

public void clickLogin(){

        login.click();

}



//Get the title of Login Page

public String getLoginTitle(){

 return    titleText.getText();

}

/**

 * This POM method will be exposed in test case to login in the application

 * @param strUserName

 * @param strPasword

 * @return

 */

public void loginToGuru99(String strUserName,String strPasword){
```

```
        //Fill user name

        this.setUserName(strUserName);

        //Fill password

        this.setPassword(strPasword);

        //Click Login button

        this.clickLogin();



    }

}
```

## Guru99 Home Page with Page Factory

```
package PageFactory;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class Guru99HomePage {

    WebDriver driver;

    @FindBy(xpath="//table//tr[@class='heading3']")
```

```
    WebElement homePageUserName;



    public Guru99HomePage(WebDriver driver){

        this.driver = driver;

        //This initElements method will create all WebElements

        PageFactory.initElements(driver, this);

    }



    //Get the User name from Home Page

        public String getHomePageDashboardUserName(){

         return    homePageUserName.getText();

        }

}
```

## Guru99 TestCase with Page Factory concept

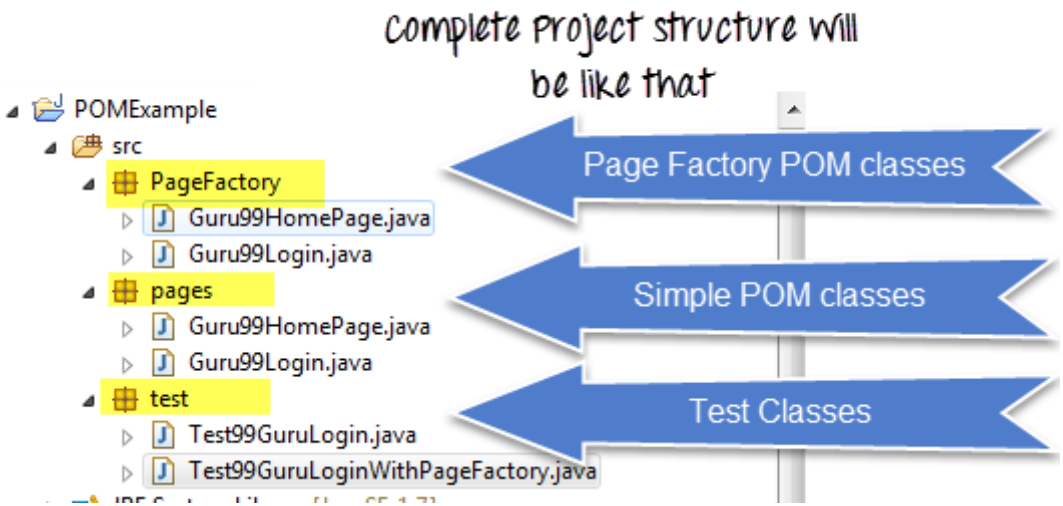
```
package test;

import java.util.concurrent.TimeUnit;
```

```
import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import PageFactory.Guru99HomePage;

import PageFactory.Guru99Login;

public class Test99GuruLoginWithPageFactory {

    WebDriver driver;

    Guru99Login objLogin;

    Guru99HomePage objHomePage;



    @BeforeTest

    public void setup(){

        driver = new FirefoxDriver();

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

        driver.get("http://demo.guru99.com/V4/");
```

```
}

/**

 * This test go to http://demo.guru99.com/V4/

 * Verify login page title as guru99 bank

 * Login to application

 * Verify the home page using Dashboard message

 */

@Test(priority=0)

public void test_Home_Page_Appear_Correct(){

    //Create Login Page object

objLogin = new Guru99Login(driver);

//Verify login page title

String loginPageTitle = objLogin.getLoginTitle();

Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));

//login to application

objLogin.loginToGuru99("mgr123", "mgr!23");

// go the next page

objHomePage = new Guru99HomePage(driver);
```

```
    //Verify home page

    Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

    }



}
```

Complete Project Structure will look like the diagram:

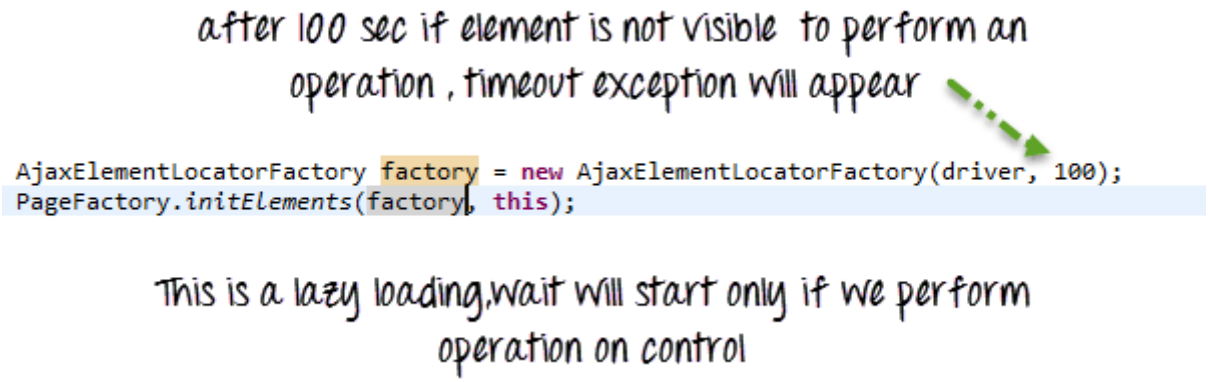



## AjaxElementLocatorFactory

One of the key advantage of using Page Factory pattern is AjaxElementLocatorFactory Class.

It is working on lazy loading concept, i.e. a timeout for a WebElement will be assigned to the Object page class with the help of AjaxElementLocatorFactory .

Here, when an operation is performed on an element the wait for its visibility starts from that moment only. If the element is not found in the given time interval, test case execution will throw 'NoSuchElementException' exception.




## Summary

1. Page Object Model is an Object repository design pattern in Selenium WebDriver.
2. POM creates our testing code maintainable, reusable.
3. Page Factory is an optimized way to create object repository in POM concept.
4. AjaxElementLocatorFactory is a lazy load concept in Page Factory pattern to identify WebElements only when they are used in any operation.

Download the Selenium Project Files for the Demo in this Tutorial

Prev Next