# Page Object Model

**Page Object Model (POM)** is a design pattern, popularly used in test automation that creates Object Repository for web UI elements. The advantage of the model is that it reduces code duplication and improves test maintenance and increase readability.

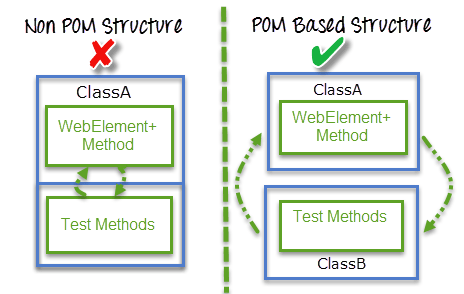
Under this model, for each web page in the application, there should be a corresponding Page Class. This Page class will identify the WebElements of that web page and also contains Page methods which perform operations on those WebElements.

**Page Object Model:**

1. Create Object Repository for WebUI Elements
2. Create objects for pages through which we perform operation in TestNG file

**Advantages:**

1. Improves Code Maintenance – Maintainability
2. Increase Code Readability - Readability
3. Reduces Code Duplication – Code Reusability



1. Page Object Design Pattern says operations and flows in the UI should be separated from test. This concept makes our code cleaner and easy to understand.

**Test Scenarios: Flipkart**

**Scenario 1:** PlaceOrder

**Scenario 2:** AddtoCart

**Scenario 3:** AddDeliveryAddress

**Scenario Navigation:**

Login -> Search a Product -> Select a Product -> Click Add to Cart button -> Place Order -> Make a Payment -> Logout

Login

launchPage

SearchProduct

Test Scenario 1 (PlaceOrder)

TestNG file

SelectProduct

AddToCart

PlaceOrder

MakePayment

Logout

**Website: Flipkart**

**Test Scenario 1: PlaceOrder - testNG class file**

Each of below is under @Test method

1. @BeforeClass() 🡪 launch browser
2. loginTest() – 3 locators
3. searchProductTest() ->
4. selectProductTest()
5. addToCartTest()
6. confirmOrderTest()
7. makePaymentTest()
8. logoutTest()
9. @AfterClass()

**Pages in the Scenario:**

1. **LoginPage**
2. **SearchProductPage**
3. **SearchResultPage**
4. **ProductDetailPage**
5. **PlaceOrderPage**
6. **MakePaymentPage**
7. **LogoutPage**

**Scenario: PlaceOrder**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **TestNG @Test Method** | **Page class** | **Object Repository + Methods at Page file** |
|  | **BeforeClass()** |  |  |
| **1** | **loginTest()** | **LoginPage** | **validateLogin()** |
| **2** | **seachProductTest()** | **SearchProductPage** | **validateSearchProduct()** |
| **3** | **selectProductTest()** | **SearchResultPage** | **validateSelectProduct()** |
| **4** | **addToCartTest()** | **ProductDetailPage** | **validateAddToCart()** |
| **5** | **placeOrderTest()** | **PlaceOrderPage** | **validatePlaceOrder()** |
| **6** | **makePayment()** | **MakePaymentPage** | **ValidateMakePayment()** |
| **7** | **logoutTest()** | **LogoutPage** | **ValidateLogout()** |
|  | **AfterClass()** |  |  |

**Scenario 2: AddProductToCart**

Navigation Flow / Traversal Flow: (Below are @Test methods or Test Cases)

1. loginTest()
2. searchProductTest()
3. selectProductTest()
4. addtToCartTest()
5. checkProductsInCartpageTest()
6. logoutTest()

**Pages:**

1. LoginPage
2. SearchProductPage
3. SearchResultPage
4. AddtoCartPage
5. CartDetailsPage
6. LogoutPage

**Scenario 3: AddDeliveryAddress**

Navigation Flow / Traversal Flow:

1. loginTest()
2. clickMyaccount()
3. addDeliveryAddressTest()
4. logoutTest()

**Pages:**

1. LoginPage
2. HomePage
3. AddressPage
4. LogoutPage

# Project Structure in POM

**src/main/java**

com.flipkart.pages 🡪 package

* LoginPage
* SearchProductPage

com.flipkart.base

* TestBase (contains reusability methods such as launchBrower(), navigateToURL(), captureScreenshot(), wait())

com.flipkart.utils

* ReadExcelData.java

**src/main/resources**

config

config.properties

**src/test/java**

com.flikart.testscenairos

1. PlaceOrder.java
2. AddTiCart.java

**src/test/resources**

* TestData
* Drivers
* Screenshots

# PageFactory

**Page Factory in Selenium** is an inbuilt Page Object Model framework concept for Selenium WebDriver but it is very optimized. It is used for initialization of Page objects or to instantiate the Page object itself. It is also used to initialize Page class elements without using "FindElement/s."



<https://www.browserstack.com/guide/page-object-model-in-selenium>

@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();

}

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);

}

}

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");

}

}

# 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 |

Facebook New Account Creation:

CreateNewAccount.java

driver.findElements(By.name(“firstname1”)).sendKeys(“Senthil”) – 10 places

driver.findElements(By.name(“lastname”)).sendKeys(“Nata”)

driver.findElement(By.id(“uname”)).sendKeys(“Senthil”); - User Name text field

driver.findElement(sendKeys(“adfsd”); - Password text field

driver.findElement(By.className(“abcded”)).click(); - Login button

By userNameLocator = By.id(“uname1”);

By pwdLocator = By.id(“pwd”)

1000 places - driver.findElement(userNameLocator).sendKeys(“senthil”);

1000 places - driver.findElement(pwdLocator).sendKeys(“adsdfdf”);

By firstNameLocator = By.name(“firstname1”);

By passwordLocator = By.xpath(“//\*[@id="password1\_step\_input"]”);

driver.findElements(firstNameLocator).sendKeys(“Senthil”) – 10 places

driver.fineElements(passwordLocator).sendKeys(“abcd”); - 4 places

Test:

PlaceOrder.java(testNG file)

@Test

Public void launchHomepage(){

LoginPage loginPage = new LoginPage();

loginPage.validateLogin();

}

LoginPage.java Loginpage() – 3 - locators for login – LaunchPage.java

SearchProductPage Searchproduct() – 1 -locator for search text box

Productdetail() – 1 -locator for click product

MakePayment() – 1 locator

Orderconfirmm() – 10 - many locators for address fields

Logout() – locator for logout button

Reference for POM Code

<https://www.perfecto.io/blog/page-object-model-in-selenium>

1. TestNG file
2. Page files
3. Excel Files
4. Property files
5. Drivers
6. Test data
7. Screenshots

**Steps to develope POM Based code:**

1. Create TextNG file for scenario1
2. Execute @BeforeClass
   1. TestNG class will extends TestBase class which has WebDriver, methods for implementations of launching the browser, navigating to URL, taking screenshot, reading from Property files.
3. Develop code for validateLoginTest() @Test method in TestNG file.
   1. Create necessary Page file and Object Repository and methods to perform the operation
   2. After the operation, perform validation and send the validation to back to @Test method
   3. Assert in the @Test method
4. Repeat the same step 3 and 4 for each @Test method