Page Object Model (POM) is a design pattern in test automation that promotes the separation of test logic from the user interface (UI) elements. It helps in maintaining the test scripts by creating **Page Objects**—separate classes that represent web pages or components of the application under test.

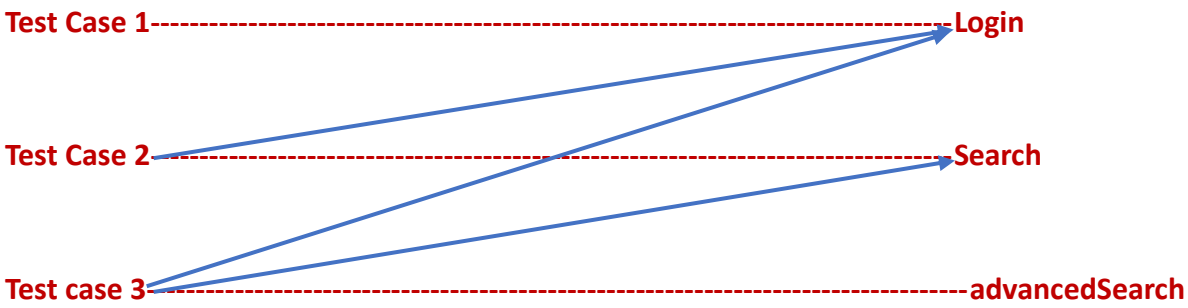
Key Points of POM:

1. **Test Case Separation**: Each web page or section of the application has its own class, which contains methods representing actions on that page, such as logging in, searching, etc.
2. **Locator Management**: Web elements (locators) for interacting with the page are stored in these separate classes, keeping the test cases clean and easier to maintain.
3. **Reusability**: Test cases can reuse methods from these page objects, making the code more modular and reducing redundancy.
4. **Maintainability**: When the UI changes (e.g., if a locator changes), only the page object needs to be updated, not every test case that uses that page.

Without Page Object Model (POM):

When we do not follow POM, test cases are written directly with locators and actions. Here's an example of how test cases can become repetitive:

- **Test Case 1**: Login
- **Test Case 2**: Login --> Search
- **Test Case 3**: Login --> Search --> Advanced Search



In each test case, you are directly writing the steps to locate and interact with the **Login page** elements (like username, password, and login button).

Problem 1: Duplication of Elements/Locators

- Each test case (Test Case 1, Test Case 2, Test Case 3) needs to locate and perform the login operation separately. This results in **duplicate code** in each of the test cases.

- Example: In all three test cases, the login page element locators are repeated.

For example:

- **Test Case 1** locates the login button:
  locator: #loginBtn
- **Test Case 2** repeats the same locator for login.
- **Test Case 3** does the same thing again.

Why this is a problem:

- **When the UI changes** (like if the login button's ID changes), you'll need to go and update the locator in **each test case**.
  - If you miss updating one test case, it will fail due to the outdated locator.

For example:

- The login button's ID changes to **#newLoginBtn**.
- Now, you need to manually go into **Test Case 1**, **Test Case 2**, and **Test Case 3** to update the locator.
- **If you forget to update** the locator in one of the test cases, it will cause that test to fail.

Problem 2: Upgradation

If the login page changes (e.g., the login button's ID or class is updated), you must update every test case that uses that locator. This creates a **high maintenance burden**. The code duplication means that **every time there's a change**, you have to check each test case to ensure the locators are updated correctly.

Example of Test Case Without POM:

Test case with locator repetition:

```
// Test Case 1: Login
driver.findElement(By.xpath("//input[@placeholder='Username']")).sendKeys("username");
driver.findElement(By.xpath("//input[@placeholder='Password']")).sendKeys("password");
driver.findElement(By.xpath("//button[@type='submit']")).click();
```

// **Test Case 2: Login → Search**

```
driver.findElement(By.xpath("//input[@placeholder='Username']")).sendKeys("username");
driver.findElement(By.xpath("//input[@placeholder='Password']")).sendKeys("password");
driver.findElement(By.xpath("//button[@type='submit']")).click();
driver.findElement(By.xpath("//input[@placeholder='Search']")).sendKeys("item");
driver.findElement(By.xpath("//button[@type='submit']")).click();
```

// **Test Case 3: Login → Search → Advanced Search**

```
driver.findElement(By.xpath("//input[@placeholder='Username']")).sendKeys("username");
driver.findElement(By.xpath("//input[@placeholder='Password']")).sendKeys("password");
driver.findElement(By.xpath("//button[@type='submit']")).click();
driver.findElement(By.xpath("//input[@placeholder='Search']")).sendKeys("item");
driver.findElement(By.xpath("//button[@type='submit']")).click();
driver.findElement(By.xpath("//button[@type='advancedSearch']")).click();
```

How POM Solves These Problems

1. Test Case Separation:

- Each page has its own class, and test cases call methods from these page classes.
- The login actions, for example, are encapsulated in the LoginPage class.

2. Locator Management:

- The locators for UI elements are stored within the page class (e.g., LoginPage), not within the test cases.
- Test cases simply call the methods defined in the page object to perform actions**.**

3. Reusability:

- Once the login logic is defined in the LoginPage class, it can be reused across all test cases that require login.
- This avoids the need to repeat the same code in each test case.

4. Maintainability:

- If a UI element changes (e.g., the login button's locator changes), we only need to update it in the LoginPage class.
- All test cases will automatically use the updated locator.

Example of Using POM to Avoid Duplication

When implementing POM, there are two ways to initialize and manage web elements in your page classes:

1. POM Without PageFactory

In this approach, you manually initialize the web elements using By locators. This method gives you more control over the element initialization, but it involves more code and requires you to explicitly initialize each element.

Login Page Class Without PageFactory:

```
package pages;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
public class LoginPage {
   private WebDriver driver;
   // Locators for username, password, and login button
   private By txt_username = By.xpath("//input[@placeholder='Username']");
   private By txt_password = By.xpath("//input[@placeholder='Password']");
   private By btn_login = By.xpath("//button[@type='submit']");
   // Constructor to initialize WebDriver
   public LoginPage(WebDriver driver) {
      this.driver = driver;
   }
   // Methods to interact with the login page
   public void setUsername(String username) {
      driver.findElement(txt_username).sendKeys(username);
   }
   public void setPassword(String password) {
      driver.findElement(txt_password).sendKeys(password);
   }
   public void clickLoginButton() {
      driver.findElement(btn_login).click();
   }
}
```

Explanation:

- The LoginPage class defines locators using By (for username, password, and login button).
- The setUsername, setPassword, and clickLoginButton methods interact with the elements by finding them using findElement.
- WebDriver is passed into the constructor to initialize the page class.

2. **POM With PageFactory**

The PageFactory class in Selenium simplifies the process of initializing web elements using annotations. It reduces boilerplate code and automatically initializes elements when the page object is created, making the code cleaner and more maintainable.

Login Page Class With PageFactory:

```
package pages;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
public class LoginPage {
   private WebDriver driver;
   // Locators for username, password, and login button using PageFactory annotations
   @FindBy(xpath = "//input[@placeholder='Username']")
   private WebElement txt_username;
   @FindBy(xpath = "//input[@placeholder='Password']")
   private WebElement txt_password
   @FindBy(xpath = "//button[@type='submit']")
   private WebElement btn_login;
   // Constructor to initialize WebDriver and PageFactory elements
   public LoginPage(WebDriver driver) {
      this.driver = driver;
      PageFactory.initElements(driver, this); // Initialize elements using PageFactory
   }
   public void setUsername(String username) {//// Methods to interact with the login page
      txt_username.sendKeys(username);
   }
   public void setPassword(String password) {
      txt_password.sendKeys(password);
   }
   public void clickLoginButton() {
      btn_login.click();
}}
```

Explanation:

- The LoginPage class uses the @FindBy annotation to define the locators for elements.
- The PageFactory.initElements(driver, this) method is used to initialize the page elements when the page object is created.
- WebDriver is passed into the constructor to initialize the page class.

Login Test Case:

```
package tests;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import pages.LoginPage;

public class LoginTest {

    public static void main(String[] args) {

        WebDriver driver = new ChromeDriver();

        driver.get("https://yourwebsite.com/login");

        // Create an instance of the LoginPage class

        LoginPage loginPage = new LoginPage(driver);

        // Use methods from LoginPage to perform login

        loginPage.setUsername("username");

        loginPage.setPassword("password");

        loginPage.clickLoginButton();

        // Assert that the login was successful

        Assert.assertEquals(driver.getTitle(), "Home Page");

        driver.quit();

    }

}
```

Importance of PageFactory in Selenium

PageFactory in Selenium helps in reducing repetitive code by automatically initializing elements when the page object is created. You don't need to explicitly use findElement in every method, leading to cleaner and more readable code. It also supports lazy initialization, meaning elements are only located when needed, enhancing performance. Additionally, it improves maintainability by reducing repetitive code and making the structure more organized, especially when UI elements change.

POM Without PageFactory:

- You manually initialize elements using By locators.
- Gives you more control but requires more code to manage.

POM With PageFactory:

- Uses the @FindBy annotation and PageFactory.initElements() method to initialize web elements.
- Reduces code, improves readability, and simplifies maintenance.

Both approaches help in applying the Page Object Model design pattern effectively in Selenium, but PageFactory provides a more streamlined and automated approach for element initialization, reducing the amount of code needed to interact with web elements.

Another real time example of using POM

POM without pagefactory

```
package pages;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.FluentWait;
public class loginpage {
   private WebDriver driver;
   private FluentWait<WebDriver> wait;
   // Constructor
   public loginpage(WebDriver driver, FluentWait<WebDriver> wait) {
      this.driver = driver;
      this.wait = wait;
   }
   // Locators
   private By txt_username_loc = By.xpath("//input[@placeholder='Username']");
   private By txt_password_loc = By.xpath("//input[@placeholder='Password']");
   private By btn_login_loc = By.xpath("//button[@type='submit']");
   // Methods
   public void setUsername(String username) {
wait.until(ExpectedConditions.visibilityOfElementLocated(txt_username_loc)).sendKeys(user
name);
   }
   public void setPassword(String password) {
wait.until(ExpectedConditions.visibilityOfElementLocated(txt_password_loc)).sendKeys(pass
word);
   }
   public void clickLoginButton() {
      wait.until(ExpectedConditions.elementToBeClickable(btn_login_loc)).click();
   }
}
```

POM with pagefactory

```
package pages;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.FluentWait;
import static org.openqa.selenium.support.ui.ExpectedConditions.elementToBeClickable;
public class Loginpage {
   private WebDriver driver;
```

```
    private FluentWait<WebDriver> wait;
    // Constructor
    public Loginpage(WebDriver driver, FluentWait<WebDriver> wait) {
        this.driver = driver;
        this.wait = wait;
        PageFactory.initElements(driver, this); // Initialize @FindBy annotations
    }
    // Locators
    @FindBy(xpath = "//input[@placeholder='Username']")
    private WebElement username_field;
    @FindBy(xpath = "//input[@placeholder='Password']")
    private WebElement password_field;
    @FindBy(xpath = "//button[@type='submit']")
    private WebElement login_btn;
    // Action Methods
    public void setUsername(String username) {
        wait.until(ExpectedConditions.visibilityOf(username_field));
        username_field.clear(); // Clears the field before entering text
        username_field.sendKeys(username);
    }
    public void setPassword(String password) {
        wait.until(ExpectedConditions.visibilityOf(password_field)); // Fixed import
        password_field.clear(); // Clears the field before entering text
        password_field.sendKeys(password);
    }
    public void clickloginbutton() {
        wait.until(ExpectedConditions.elementToBeClickable(login_btn)).click();
    }

}
```

Test

```
package pages;
import java.time.Duration;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.FluentWait;
import org.testng.Assert;
public class LoginTest {
    public static void main(String[] args) {
        WebDriver driver=null ;
        try {
```

```
        // Step 1: Setup WebDriver
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        // Step 2: Configure FluentWait
        FluentWait<WebDriver> wait = new FluentWait<>(driver)
                .withTimeout(Duration.ofSeconds(20))
                .pollingEvery(Duration.ofMillis(1000))
                .ignoring(Exception.class);
        // Step 3: Navigate to the login page
        driver.get("https://opensourcedemo.orangehrmlive.com/web/index.php/auth/login");
        // Step 4: Create an object of the LoginPage class
        Loginpage lp = new Loginpage(driver, wait);
        // Step 5: Perform login with correct credentials
        lp.setUsername("Admin");  // Correct username
        lp.setPassword("admin123"); // Correct password
        lp.clickloginbutton(); // Click the login button
        // Step 6: Check if the login was successful by verifying the page title
        Assert.assertEquals(driver.getTitle(), "OrangeHRM");
        System.out.println("Login Test Passed: Successfully logged in!");
    } catch (Exception e) {
        System.out.println("Login Test Failed: " + e.getMessage());
    } finally {
        // Step 7: Close the browser after the test
        if (driver != null) {
            driver.quit();
        }
    }
}}
```