**Page Object Model (POM)**

**Introduction**

One design pattern frequently found in test automation frameworks is the Page Object Model (POM). For web pages, it facilitates the creation of object-oriented classes that abstract interactions with page elements. This kind of test architecture improves the code's modularity, maintainability, and reuse.

**POM: What is it?**

According to the Page Object Model, every application's web page is represented by a different class, and the page's elements are defined as variables inside of that class. Methods are the ways in which these items are interacted with, as via clicking buttons or typing text. By ensuring that the tests are not closely linked to the user interface, this method makes it easier to maintain the tests as the program changes.

**Benefits of POM**

* Division of Interests
  + Because POM isolates the test code from the page-specific code, only the Page Object classes need to be updated when the UI changes. These modifications have no effect on the test scripts.
* Reusability:
* Code duplication can be greatly reduced by using the same Page Object classes in numerous test scripts. The testing framework is more effective because of this reusability.
* Maintainability:
  + The test code is easier to comprehend and manage when using POM. Updates to the UI may be handled centrally because page elements and actions are contained within their own classes.
* Readability
  + Test script readability is improved with POM. Page Object-based test scripts are simpler and more detailed since they concentrate more on the subject of testing than on the methodology.
* Scalability
  + POM facilitates the test automation framework's scalability. It is possible to add additional Page Object classes to the program without impacting the current tests as it expands.
* Reduction of code duplication:
  + POM reduces code duplication throughout the test suite by encapsulating the interactions with web elements within Page Object classes.

**Implementation**

package POM;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

import org.openqa.selenium.support.ui.Select;

import java.util.List;

public class Test1 {

WebDriver driver;

JavascriptExecutor js;

Test1(WebDriver driver) {

this.driver = driver;

this.js = (JavascriptExecutor) driver;

PageFactory.initElements(driver, this);

}

@FindBy(id = "firstName")

WebElement firstName;

@FindBy(id = "lastName")

WebElement lastName;

@FindBy(id = "userEmail")

WebElement userEmail;

@FindBy(xpath = "//label[text()='Male']")

WebElement genderMale;

@FindBy(id = "userNumber")

WebElement userNumber;

@FindBy(id = "dateOfBirthInput")

WebElement dateOfBirthInput;

@FindBy(className = "react-datepicker\_\_month-select")

WebElement monthDropdown;

@FindBy(className = "react-datepicker\_\_year-select")

WebElement yearDropdown;

@FindBy(xpath = "//div[contains(@class, 'react-datepicker\_\_day') and text()='24']")

WebElement day;

@FindBy(id = "subjectsInput")

WebElement subjectsInput;

@FindBy(xpath = "//label[text()='Music']")

WebElement musicCheckbox;

@FindBy(id = "uploadPicture")

WebElement uploadPicture;

@FindBy(id = "currentAddress")

WebElement currentAddress;

@FindBy(id = "react-select-3-input")

WebElement state;

@FindBy(id = "react-select-4-input")

WebElement city;

@FindBy(id = "submit")

WebElement submitButton;

// Methods to interact with the form fields

public void setFirstName(String fname) {

firstName.sendKeys(fname);

}

public void setLastName(String lname) {

lastName.sendKeys(lname);

}

public void setEmail(String email) {

userEmail.sendKeys(email);

}

public void selectGender() {

js.executeScript("arguments[0].click();", genderMale);

}

public void setUserNumber(String number) {

userNumber.sendKeys(number);

}

public void setDateOfBirth(String month, String year, String dayText) {

js.executeScript("arguments[0].click();", dateOfBirthInput);

new Select(monthDropdown).selectByVisibleText(month);

new Select(yearDropdown).selectByVisibleText(year);

js.executeScript("arguments[0].click();", driver.findElement(

By.xpath("//div[contains(@class, 'react-datepicker\_\_day') and text()='" + dayText + "']")));

}

public void setSubjects(String subject) {

subjectsInput.sendKeys(subject);

subjectsInput.sendKeys(Keys.ENTER);

}

public void selectHobbies() {

js.executeScript("arguments[0].click();", musicCheckbox);

}

public void uploadPicture(String filePath) {

uploadPicture.sendKeys(filePath);

}

public void setAddress(String address) {

currentAddress.sendKeys(address);

}

public void selectState(String stateName) {

state.sendKeys(stateName);

state.sendKeys(Keys.ENTER);

}

public void selectCity(String cityName) {

city.sendKeys(cityName);

city.sendKeys(Keys.ENTER);

}

public void submitForm() {

js.executeScript("arguments[0].click();", submitButton);

}

public static void main(String[] args) {

WebDriver driver = new FirefoxDriver();

System.setProperty("webdriver.gecko.driver",

"C:\\Users\\admin\\Downloads\\geckodriver-v0.34.0-win64\\geckodriver.exe");

try {

driver.get("https://demoqa.com/automation-practice-form");

driver.manage().window().maximize();

Test1 formPage = new Test1(driver);

formPage.setFirstName("N");

formPage.setLastName("Mahant");

formPage.setEmail("mahant@gmail.com");

formPage.selectGender();

formPage.setUserNumber("7710279071");

formPage.setDateOfBirth("March", "2003", "24");

formPage.setSubjects("Computer Science");

formPage.selectHobbies();

formPage.uploadPicture("C:\\Users\\admin\\Downloads\\8th Sem.jpg");

formPage.setAddress("La Casa PG, Gurugram, Haryana");

formPage.selectState("Haryana");

formPage.selectCity("Panipat");

formPage.submitForm();

Thread.sleep(3000);

} catch (Exception e) {

e.printStackTrace();

} finally {

driver.quit();

}

}

}

**Conclusion**

Testing can be automated in an organized and effective manner with Java's Page Object Model. Because of its fundamental advantages—better code readability, maintainability, and reusability—it is a recommended option for extensive test automation projects. Using POM will help you create a test framework that is more scalable and organized, which will result in tests that are more dependable and maintainable.