**Selenium ile** **Page Object Model**

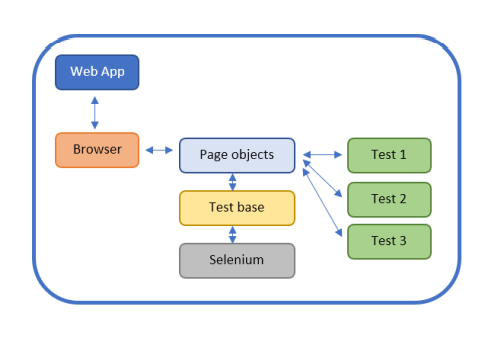
Seleniumdaki POM web uygulamalarında kullanıcı arayüzündeki elemanlar için nesne repository oluşturacak tasarımdır. Her bir web sayfası için sayfa classı olmalı ve burada o sayfaya ait tüm elamanlar tanımlanmalıdır. Page object modelin uygulanması test objeleriyle test scriptlerini ayırmayı sağlar. Bu da birçok kolaylık ve avantaj sağlar. POM sayesinde kod tekrarı yapmayız ve kod maintenance çok daha kolay olur. Page object model ile her sayfayı, bir page class’ı ile ilişkilendiririz. Bu page classında da o sayfaya ait olan web elementler ve yine o sayfaya ait metodlar yer alır. Böylece, bu sayfada herhangi bir elementin locator’ında değişiklik olduğu zaman, bunu ortak bir yerden yönetebiliriz. Bu sayfada işlem yapan her testimizde element locator’unu tek tek değiştirmek yerine, tek bir class’ta yapmış olduğumuz değişiklik ile maintenance çok daha kolay hale gelir.

POM altında, uygulamadaki her web sayfası için karşılık gelen bir page class oluşturulması gerekmektedir. Bu Page class'ı, o Web sayfasının WebElement'lerini tanımlayacak ve ayrıca bu WebElements üzerinde işlemler gerçekleştiren Web Methodlarını da içerecektir. Bu yöntemlerin adı, gerçekleştirdikleri göreve göre verilmesi en sağlıklı ve okunabilir kod için önemlidir, yani işlem yaptığınız method bir username'i içeriyorsa method adında username içeren bir isim vermek önemlidir. Page Object Model ve Page Factory konuları çoğu zaman birbirlerine karıştırılabiliyor, bu doğrultuda ikisi arasındaki farklara değinmek gerekirse:

|  |  |
| --- | --- |
| Page Object Model | Page Factory |
| POM, komut dosyalarını ve sayfa nesnelerinin kolayca ayırt edilebileceği şekilde sayfalara göre organize etmesini söyleyen bir test tasarım modelidir. | Page Factory ise, Selenium'da yerleşik olan Page Object Modelini uygulamanın bir yoludur. |
| POM'da, Page Factory kullanmadan 'By' kullanarak locator tanımlanabilir. | Page Factory kullanırken sayfa nesnelerini tanımlamak için FindBy ek açıklamasını kullanılır. |
| Düz sayfa nesne modelinde, her sayfa nesnesini ayrı ayrı başlatmanız gerekir, aksi takdirde NullPointerException ile karşılaşırsınız. | Ancak PageFactory'de tüm sayfa nesneleri initElements () yöntemi kullanılarak başlatılabilir. |

➢ Page object model ile, sayfaya özgü elementleri veya methodlari page class içinde tutar ve bunları gerçek test classlarından uzak tutarız.

➢ POM ile ihtiyacımız olan class üyelerini sadece bir kez create edip birden çok kez kullanabiliriz.



➢ Framework un verimliliğini artırmak için core Java ve Selenium konseptini kullanarak temel olarak page classları ve test classları oluşturmamız gerekmektedir.

➢Herhangi bir senaryoya başlamadan önce kullanılacak sayfaların belirlenmesi sağlık olur. Örneğin bir başarılı bir giriş yapma senaryosu yazılacaksa, login isimli bir page class tanımlaması gerçekleştirilmesi sağlıklı olur. Login işlemi başarılı olduktan sonra buradan hangi sayfaya yönlendirilecekse bunlar için page class açılmalıdır.

* Page Object Model temelde 3 package içerir:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

➢ Tests: Sadece testleri execute etmek için gerekli adımları yazacağımız class’lar içerir. Burada hiçbir data girişi yapmamıza gerek yoktur.

➢ Pages: Test yapacağımız sayfalardaki Web Elementlerini locate etmek ve temel methodları olusturmak için kullanılır.

➢ Utilities: Driver, TestBase ve ConfigurationReader ve ReusableMethods class’larını içerir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Şimdi POM kullanmadan bir örnek yapmamız gerekirse:

package tests.deneme;  
  
import org.openqa.selenium.By;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.interactions.Actions;  
import org.testng.Assert;  
import org.testng.annotations.Test;  
import utilities.TestBaseBeforeMethodAfterMethod;  
  
public class bluerentalCars extends TestBaseBeforeMethodAfterMethod {  
  
 @Test  
 public void bluerentalCarsTest() {  
   
 // blurentalcars anasayfasına gidin  
 driver.get("https://www.bluerentalcars.com/");

//login butonuna tıklayın  
 driver.findElement(By.*xpath*("//\*[@class='btn btn-primary btn-sm']")).click();  
   
   
 //Email adres kutusunu locate edip actions kullanarak aşağıdaki bilgileri yazarak giriş yapın  
 //email adres = customer@bluerentalcars.com  
 //password =12345  
 //Giris yapın (Enter ile)

WebElement email = driver.findElement(By.*xpath*("//\*[@name='email']"));  
 Actions actions = new Actions(driver);  
 actions.click(email).sendKeys("customer@bluerentalcars.com")  
 .sendKeys(Keys.*TAB*).sendKeys("12345")  
 .sendKeys(Keys.*ENTER*).perform();  
  
 //Değerler girildiğinde sayfaya doğru gidildiğini test edin  
 Assert.*assertTrue*(driver.findElement(By.*xpath*("//\*[@id='dropdown-basic-button']")).isDisplayed());  
 }  
}

* Yukarıda da gördüğünüz üzere POM kullanmadan yapmış olduğumuz testlerde her ögeyi sayfaya giderek ayrı ayrı locate etmemiz gerekmektedir. Yani gördüğünüz gibi, page object model olmadan testlerimizi yönetmek istediğimiz zaman çok daha fazla iş gücü harcıyoruz. Bu durum bir defa o sayfayla ilgili test yapıldığı zaman sorun olmamakla birlikte aynı sayfaya birden fazla test yapılması gerektiği durumda can sıkıcı olabiliyor. İşte tam olarak burada karşımıza Page Factory çıkıyor.

**Page Object Model PageFactory**

PageFactory, page object dizaynı için bir önemli classtır. Page objelerini instantiate(ilk değer atama) için page classlarında PageFactory kullanıyoruz. Bir page class’ı oluşturduğumuzda ilk isimiz bir constructor oluşturup, pageFactory class’ından initElements( ) method’unu kullanmak olmalıdır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

PageFactory.initElements(driver,this);

this => page instance

driver => bizim göndereceğimiz driver’dır

Aslında PageFactory class’ına, elementlere ilk değeri atayan initElements( ) metodunu kullanmak için ihtiyacımız vardır. Page class’larında webElementleri locate etmek için şimdiye kadar kullandığımız driver.findElement( ) method’unu kullanmayacağız. Burada @FindBy notasyonunu test class’larında kullanacağımız Web Elementlerini Page sayfasında locate etmek için kullanırız. Bunun için kullanacağımız Web Elementini public olarak oluşturmalı, sonra da @FindBy notasyonu ile locate etmeliyiz. Bu işlemi yaptıktan sonra hangi test methodumuzda bu web elemente ihtiyaç duyarsak page class’ından üreteceğimiz obje üzerinden rahatlıkla kullanabiliriz.

* Bu doğrultuda bir önceki “blurentalcars” testini POM ile page classı oluşturarak yapmadan önce Driver classı içerisinde oluşturmamız gereken kısımları göstermek isterim:

package tests.utilities;

import io.github.bonigarcia.wdm.WebDriverManager;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import java.time.Duration;

public class Driver {

public static WebDriver getDriver() {

if (driver == null) { //Defalarca webdriver çalıştırmamak amacıyla yapıldı

WebDriverManager.chromedriver().setup();

driver = new ChromeDriver();

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

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));

}

return driver;

}

public static void closeDriver(){

if (driver!=null) { //diver'a değer atanmamışsa

driver.close();

driver = null; //Kapandıktan sonraki açmaları garanti altına almak için driver'i tekrar null yaptık

}

}

}

Testlerimizi çalıştırdığımızda her seferinde yeni driver oluşturduğu için her test methodu için yeni bir pencere(driver) açıyor. Eğer dirver'e bir değer atanmamışsa yani driver==null; ise bir kere driver'i çalıştırır diyerek bir kere if() içini çalıştıracak. Ve driver artık bir kere çalıştığı için ve değer atandığı için null olmayacak ve direkt return edecek ve diğer testlerimiz aynı pencere(driver) üzerinde çalışacak. Bu if kontrolü ilk çağrılan amazon için null değildir, dolayısı ile if bloğu çalışır, diğer sayfalar için yeni chrome sayfası açmaz, daha önce açılanı kullanır.

POM'de Driver icin TestBase class'ına extends etmek yerine Driver class'ından static method'lar kullanarak driver oluşturup, ilgili ayarların yapılması ve en sonda driver'ın kapatılması tercih edilmiştir. Bu sebeple artık TestBase kullanmadan bir Driver class’ı oluşturup kodumuzu ayağa kaldıracak kodları burada tanımladık ki bundan sonraki yapacağımız testlerde her defasında başka bir class’a extends yapmadan sadece class ismiyle kodları hareket ettirebilelim.

* Driver class yapısını gösterdikten sonra “blurentalcars” artık POM ile yapabiliriz:

package tests.deneme;  
  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.interactions.Actions;  
import org.testng.annotations.Test;  
import org.testng.asserts.SoftAssert;  
import pages.BlueRentalCarsPage;  
import utilities.Driver;  
  
public class POM\_bluerentalCars {  
  
 @Test  
 public void bluerentalCars() {  
  
 BlueRentalCarsPage blueRentalCarsPage = new BlueRentalCarsPage();  
  
 //blurentalcars anasayfasına gidin  
  
 Driver.*getDriver*().get("https://www.bluerentalcars.com/");  
  
 //login butonuna tıklayın  
 blueRentalCarsPage.birinciLoginButton.click();  
  
 //Email adres kutusunu locate edip actions kullanarak aşağıdaki bilgileri yazarak giriş yapın  
 //email adres = customer@bluerentalcars.com  
 //password =12345  
 //Giris yapın (Enter ile)  
 Actions actions = new Actions(Driver.*getDriver*());  
 actions.click(blueRentalCarsPage.email)  
 .sendKeys("customer@bluerentalcars.com")  
 .sendKeys(Keys.*TAB*).sendKeys("12345")  
 .sendKeys(Keys.*ENTER*).perform();  
  
 //Değerler girildiğinde sayfaya doğru gidildiğini test edin  
  
 System.*out*.println(blueRentalCarsPage.girisDogrula.getText());  
  
 SoftAssert softAssert = new SoftAssert();  
 softAssert.assertTrue(blueRentalCarsPage.girisDogrula.isDisplayed());  
 softAssert.assertAll();  
  
 Driver.*closeDriver*();  
   
 }  
}

**Page Object Model Properties File**

Bazen kod içerisine sürekli müdahale ederek değişiklik yapmaktansa tanımlı dosyada ilgili alanın değerini değiştirmek daha kolay olabilir. Örnek olarak; kod içerisinde veritabanı login veya web servisin endpoint bilgileri gibi önemli bilgiler yer alır. Bu bilgiler test ortamınızda ve canlı ortamda farklılık gösterebilir. Paketi test ortamından canlı ortamına aktarırken kod içerisine müdahale etmektense sadece config dosyasında tanımlı bilgiyi değiştirmeniz yeterli olacaktır. İşte böyle bir durumda karşımıza configuration.properties çıkar. configuration.properties Test datalarını tuttuğumuz .properties uzantılı bir dosyadır. Bu dosya, projeyi temiz ve dinamik hale getirir bu sayede test datalarını dinamik hale getirebiliriz.

Örneğin: driver.get(<https://www.bluerentalcars.com/>) yazmak yerine configuration dosyamıza blueRentalCarsURL’i tanımlayıp test class’ında sadece driver.get(blueRentalCarsURL) kullanırız.

Temel olarak key(anahtar) ve value(değer) çiftlerini kullanırız ve ihtiyaç duyduğumuzda key kullanarak value çağırırız(url,credentials,browser,environments,...)

○ url=https://www.fhctrip.com/

○ browser=chrome

○ username=manager

○ password=Man1ager2!

○ name=Ali vb.

configuration.properties file Olusturmak icin project’imize sağ click yapılmalıdır.

New =>File => isim : configuration.properties

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

➢ Dosya oluştururken bizim için önemli olan ismi değil, uzantısıdır ve bu uzantı (.properties) şeklinde olmalıdır.

➢ Bu uzantı sayesinde Properties Class’ını kullanabilir, oluşturacağımız obje yardımıyla configuration.properties dosyasindaki key-value ikililerine ulaşabiliriz.

➢ properties file test dataları saklar. Bu dosyayı kullanmanın amacı, kodu sabit(hard coded) değil, dinamik yapmaktır. Bu dosya sayesinde tüm kullanıcılar verilere kolayca ulaşabilir ve update edebilir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Page Object Model ConfigurationReader Class**

ConfigurationReader class test method’larımız ile Configuration.properties arasında iletişimi sağlar. Bu class’da test class’larından kolayca ulaşmak için static method bulunur. Method static olduğundan method içerisinden çağıracağımız variable da static olmalıdır. Kullanacağımız static variable’a ilk değeri atamak için(instantiate) de static block kullanırız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu bilgiler doğrultusunda “blurentalcars” testini ConfigurationReader class ve configuration.properties ile yapmak istediğimizde karşımıza çıkan adımlar şöyle olur:

1. Öncelikle configuration.properties içi tanımlanıp hangi değerler kullanılacağı belirtilmelidir.

* configuration.properties içi:

browser=chrome  
  
!--blueRentalCar URL  
blueRentalCarsURL = https://www.bluerentalcars.com/  
  
!--blueRentalCar Pozitif test datalari  
blueRentalCarsEmail =customer@bluerentalcars.com  
blueRentalCarsPassword = 12345  
blueRentalCarsKullanici = John Walker

1. Daha sonra ise class’ımızı oluşturup ConfigReader class’ından almış olduğumuz fis dosya yolu sayesinde testimizi başarıla tamamlamış oluruz.

* bluerentalCarsPozitifTest class’ı:

package tests.deneme;  
  
import org.openqa.selenium.Keys;  
import org.testng.annotations.Test;  
import pages.BlueRentalCarsPage;  
import utilities.ConfigReader;  
import utilities.Driver;  
import utilities.ReusableMethods;  
  
public class bluerentalCarsPozitifTest {  
  
 @Test  
 public void bluerentalCarsPozitifTest() {  
  
 BlueRentalCarsPage blueRentalCarsPage = new BlueRentalCarsPage();  
  
 //blurentalcars anasayfasına gidin  
  
 String blueRentalCarsURL = ConfigReader.*getProperty*("blueRentalCarsURL");  
 Driver.*getDriver*().get(blueRentalCarsURL);  
  
 //login butonuna tıklayın  
 blueRentalCarsPage.birinciLoginButton.click();  
  
 //email adres = customer@bluerentalcars.com  
 //password =12345  
 //Giris yapın (Enter ile)  
 ReusableMethods.*getActions*().click(blueRentalCarsPage.email)  
 .sendKeys(ConfigReader.*getProperty*("blueRentalCarsEmail"))  
 .sendKeys(Keys.*TAB*).sendKeys(ConfigReader.*getProperty*("blueRentalCarsPassword"))  
 .sendKeys(Keys.*ENTER*).perform();  
  
 //Değerler girildiğinde sayfaya doğru gidildiğini test edin  
  
 System.*out*.println(blueRentalCarsPage.girisDogrula.getText());  
  
 ReusableMethods.*getSoftAssert*().assertTrue(blueRentalCarsPage.girisDogrula.isDisplayed());  
  
 Driver.*closeDriver*();  
   
 }  
}

**Tüm bu bilgilerden sonra (POM) Page Object Model’i özetlemek gerekirse:**

1. **Page Object Model:** Testlerimizi daha kolay ve düzenli olarak hazırlamamız ve çalıştırmamız için oluşturulmuş bir modeldir. Framework için üretilmiş benzer modeller olmakla birlikte en güncel olan ve çok kullanılan model olduğu için POM selenium testlerinde çoğunlukla kullanılır.

Temelde projemizde 3 package bulunur

1. Pages: locate işlemlerini ve reusable login method’u gibi sıradan method’ları bu kısma koyarız.

2. Utilities: projemiz için gerekli ekstra kaynaklar burada bulunur (TestBase, Driver, ReusableMethods, ConfifReader)

3. Tests: bütün testlerimizin bulunduğu package’dir.

**2- POM dosya yapısı:**

**-Pages:** Test yapacağımız web page’ler için Pages package’ın altında bir class oluşturuyoruz. Bu class’larda mutlaka yapmamız gereken şey driver’i oluşturduğumuz Driver class’ından alıp PageFactory.initElements ile ilk değer ataması yapmaktır. Sonrasında web sayfamızda kullanacağımız WebElementlerin tamamını public olarak olusturmak ve @FindBy notasyonu ile locate etmektir. Eğer istersek login gibi bazı adımları yapacak metotları da bu class’da oluşturabiliriz. Test class’ımızdan Page sayfasındaki variable ve method’lara obje oluşturup erişim sağlarız.

**- configuration.properties :** Bu dosyayı testlerimizde kullanacağımız url, test dataları gibi kullanıcıdan aldığımız dataları dinamik yapmak için kullanırız. Tüm testlerimizi bu sayfadan alacağımız datalara göre dizayn ederiz. Böylece bu dosyada yapacağımız bir değer değişikliği ile tüm testCase’lerindeki test datalarını güncelleyebiliriz. Bu sayfayı basit bir text dosyası gibi dizayn ederiz her test datasını key=value seklinde key, value ile oluştururuz.

**- ConfigReader:** Bu class test class’ımız ile configuration.properties dosyası arasında tercümanlık yapar. İçinde .properties uzantılı dosyaları okumak için gerekli bir static blok oluştururuz. Ayrıca Test class’larımızdan çağırmak için getPropert() methodunu oluştururuz. Bu method test class’ından gönderdiğimiz key değerini static blok yardımı ile configuration.properties’de bulup karsısındaki value’yu bize döndürür.

**-Driver:** Test class’ımızda ve page class’ında kullanacağımız driver’i oluşturduğumuz class’tır. Utilities Package’i altında oluştururuz. Driver class’ını Singleton yapabiliriz. Driver’i static olarak oluşturur ve oluşturduğumuz getDriver() method içinde driver’ımıza değer ataması yaparız. İs hayatında karşılaşacağımız farklı browser’lar (chrome,firefox,safari vb..) değer atama işlemi yapmadan önce kullanıcının tercihini alırız. Kullanıcı tercihini almak için configuration.properties dosyasında browser=chrome gibi bir key, value ikilisi oluşturur buradaki tercihe göre driver’a değer atamak için de switch case kullanırız. Ayrıca her driver çağırdığımızda yeni driver oluşturmaması için önce if ile driver’ın ataması yapılmış mı kontrol ederiz, atama yapılmışsa aynı driver ile devam eder, atama yapılmamışsa yeni bir driver oluşturur ve değer atayıp test sayfasına döneriz. Bu Class’ta ayrıca window.manage ayarlarını da yapar, en sonda da closeDriver method ile driver’ı kapatma işlemine de yardımcı oluruz.

**Kaynak:**

* <http://inancozkul.com.tr/Project/Selenium/Selenium-ile-Page-Object-Model-kullanimi/36>
* <https://medium.com/sahibinden-technology/selenium-java-page-object-model-page-factory-kullan%C4%B1m%C4%B1-418d66414f17>
* <https://techproeducation.com/> (Ahmet Bulutluöz)

*Zeliha Öznük*