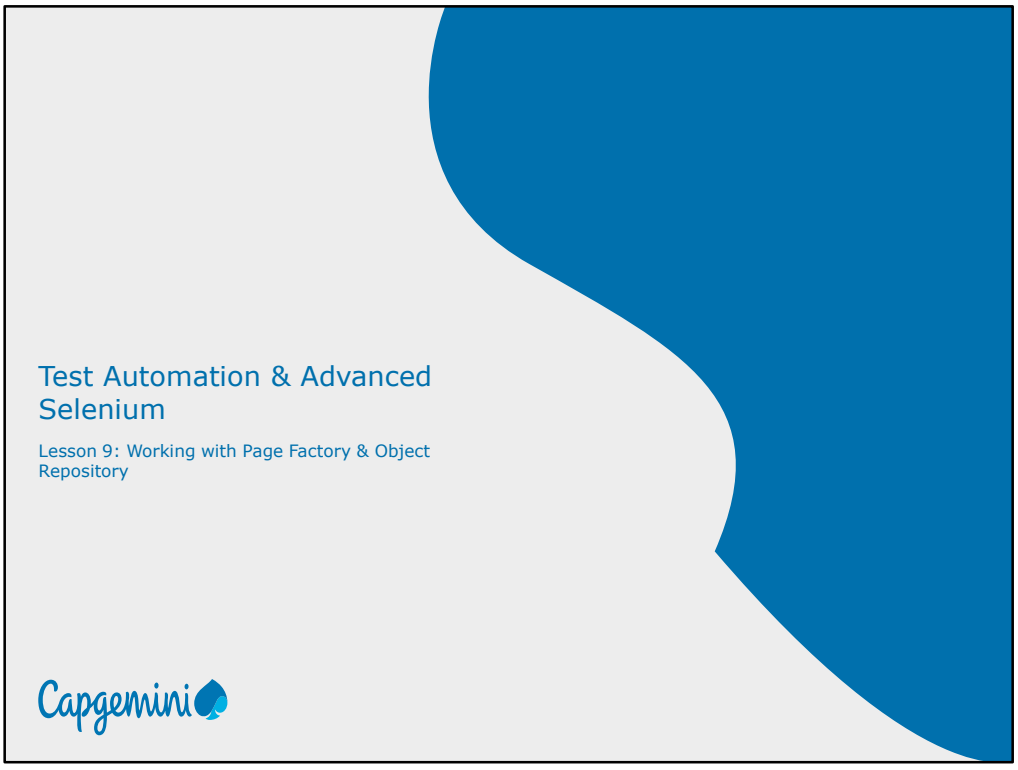
Test Automation & Advanced Selenium
Lesson 9: Working with Page Factory & Object Repository
Capgemini

Lesson Objectives

- Introduction to Page Factory Design Pattern
- Advantages of Page Factory Design Pattern
- Implementing Page Factory Design Pattern
- Difference between Page Object Model (POM) and Page Factory

9.1: Getting Started with Page Factory

Introduction to Page Factory Design Pattern

- Page Factory pattern is an optimized version of the Page Object Model (POM) Design Pattern
- It is an extension to the Page Object design pattern
- Page Factory is an inbuilt page object model concept for Selenium WebDriver, but it is very optimized
- Page Factory can be used in any kind of framework such as Data Driven, Modular or Keyword Driven
- Page Factory gives more focus on how the code is being structured to get the best benefit out of it
- By integrating POM and Page Factory with the Test Case Model, you receive more focus on how the code is being structured to get the most benefits
- Like any other design pattern in software development, Page Factory pattern solves the problems faced by automation developers

9.1: Getting Started with Page Factory

Introduction to Page Factory Design Pattern (Cont.)

- PageFactory is a class provided by Selenium WebDriver to support the Page Object design pattern
- It makes handling "Page Objects" easier and optimized by providing the annotations and methods

9.1: Getting Started with Page Factory

Advantages of Page Factory Design Pattern

- Separation of Concerns
  - Page Factory Model (Extended POM) provides clear separation of HTML Attributes, Methods to access UI Elements, Test Logic and Assert statements. This further provides us with a clear pathway to even separate Test data and Configuration from the test logic using TestNG, Maven/Ant tools. This approach enables us to build a robust test automation framework that can be sustainable with easy maintenance.
- Avoids code Duplication
  - Page Factory Model facilitates us to define all the UI Elements and the methods to access the UI Elements inside a single Web Page class and access them by declaring the object of the Web Page class inside the test method. Let's assume there are 10 different test methods that refers to the UI Elements on Flipkart home page, with Page Factory we just have to initialize the page object of the Flipkart home page. This significantly reduces the code duplication.

9.1: Getting Started with Page Factory

Advantages of Page Factory Design Pattern (Cont.)

- Easy Maintenance
  - Business requirements frequently change for web applications as compare to any other application. In most of the cases, these changes in requirements brings in the changes in the UI of the web application. This situation in project leads to your existing regression tests breaking and we need to fix them without having to do significant changes. Using Page Factory Model (Extended POM) we can easily fix the test scripts where the test logic/workflow remains the same but the UI Element locators have changed. Before Page Factory Model we had to go into every individual test method to change to the locators.

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern

- Initialization of Page Objects Using Page Factory
  - Use of initElements() : We should initialize page objects using initElements() method from PageFactory Class. once we call initElements() method, all elements will get initialized. PageFactory.initElements() static method takes the driver instance of the given class and the class type, and returns a Page Object with its fields fully initialized.
  - There are 3 ways of initializing this. We should preferably use a constructor which takes a WebDriver instance as its only argument. An exception will be thrown if the class cannot be instantiated.

Option 1:

```
ShopOnlineHomePage page = new
ShopOnlineHomePage(driver);
PageFactory.initElements(driver, page);
```

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern (Cont.)

Option 2:

```
ShopOnlineHomePage
page=PageFactory.intElements(driver,ShopOnlineH
omePage.class)
```

Option 3:

```
/*
*This is done inside the webpage
ShopOnlineHomePage class constructor.
*/
public ShopOnlineHomePage(WebDriver driver)
{
      this.driver = driver;
      PageFactory.initElements(driver, this);
}
```

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern (Cont.)

- Use Annotations of PageFactory Class
  - In Page Factory, annotations are used to give descriptive names for WebElements to improve code readability
- @FindBy Annotation
  - Example: The below given two annotations are pointing to the same UI Element.

```
Example 1:

@FindBy(id = "username")
 WebElement txt_UserName;
```

```
Example 2:

@FindBy(how = How.ID, using = "username")
 WebElement txt_UserName;
```

@FindBy Annotation

As the name suggest, it helps to find the elements in the page using By strategy. @FindBy can accept TagName, PartialLinkText, Name, LinkText, Id, Css, ClassName, XPath as attributes. An alternative mechanism for locating the element or a list of elements. This allows users to quickly and easily create PageObjects.

Page Factory will initialize every WebElement variable with a reference to a corresponding element on the actual web page based on “locators” defined. This is done by using @FindBy annotations.

By default, PageFactory will search for UI Elements on the page with a matching id attribute, If that fails, then it will search by the name attribute. But as we need more control over identifying elements in the HTML page and mapping them to our Page Object fields. One way to do this is to use the @FindBy annotation.

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern (Cont.)

- To work with class name, we will define as below:

```
Example 3:

@FindBy(className=".input.username")
private WebElement userName;
```

- When we have multiple elements we can initialize them using PageFactory as below:

```
Example 4:

@FindBy(tagName = "mylist")
private List<WebElement> links;
```

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern (Cont.)

- @CacheLookup: Every time when a method is called on a WebElement, the driver will first find it on the current page and then simulate the action on the WebElement. There are cases where we will be working with a basic page, and we know that we will find the element on the page every time we look for it, In such cases we can use annotation `@CacheLookup` which is another annotation in page factory.

```
Example 1:

@FindBy(name="username")
@CacheLookup
private WebElement userName;
```

@CacheLookup:

We will mark annotation @CacheLookup to WebElements to indicate that it never changes that is, that the same instance in the DOM will always be used.

@CacheLookup annotation can be used to instruct the InitElements() method to cache the element once its located and so that it will not be searched over and over again. This is useful when the elements that are always going to be there.

But whenever we use @CacheLookup annotation, we will be losing one of the page factory benefit as it will find the element once and then keep a reference to it, hence, we are more likely to see StaleElementExceptions. On the contrary, if this annotation is not used and every time there is a reference to a WebElement, it will go and find it again so you shouldn't see StaleElementExceptions.

9.2: How to Implement Page Factory Design Pattern

Implementing Page Factory Design Pattern (Cont.)

- AjaxElementLocatorFactory: AjaxElementLocatorFactory is a lazy load concept in Page Factory pattern to identify WebElements only when they are used in any operation i.e. a timeOut for a WebElement can be assigned to the Object page class with the help of AjaxElementLocatorFactory.

Example 1:

```
public int TimeoutValue = 30;
public SearchResultsPage(Webdriver driver)
{
    PageFactory.initElements(new
    AjaxElementLocatorFactory(driver,
    TimeoutValue), this);
}
```

The above code will wait for maximum of 30 seconds until the elements specified by annotations is loaded. If the element is not found in the given time interval, it will throw NoSuchElementException exception.

Demo – How to implement Page Factory Design Pattern

- Implementing Page Factory Design Pattern

9.3: Difference between Page Object Model (POM) and Page Factory

Comparison Between Page Object Model (POM) and Page Factory

- A Page Object Model (POM) is a test design pattern which works on the principle of organizing the page objects in such a manner that script and page objects can be separated easily
- A Page Factory is one way of implementing PageObject Model which is inbuilt in selenium
- Page Object Model is a design approach whereas the PageFactory class from the package 'org.openqa.selenium' is used to provide additional support for the Page Object pattern
- In plain POM, you define locators using 'By' while in Page Factory, you use FindBy annotation to define page objects
- Plain Page Object Model (POM) is not optimal as it does not provide lazy initialization while Page Factory provides lazy initialization
- Plain Page Object Model (POM) will not help in StaleElementReferecneException while Page Factory takes care of this exception by relocating web element every time whenever it is used

9.3: Difference between Page Object Model (POM) and Page Factory

Comparison Between Page Object Model (POM) and Page Factory (Cont.)

- In plain Page Object Model (POM), you need to initialize every page object individually otherwise you will encounter NullPointerException while in PageFactory all page objects are initialized (Lazily) by using initElements() method

9.4: Working with External Object Repository in Selenium WebDriver

Introduction to External Object Repository

- An object repository is a common storage location for all objects
- With reference to Selenium WebDriver, objects would typically be the locators used to uniquely identify web elements
- The important advantage of implementing Object Repository is the separation of objects from test cases
- You are required to make changes only in the object repository when any of the WebElement's locator value is changes rather than making changes in all the test cases in which the locator has been used
- Maintaining an object repository increases the modularity of framework implementation

Introduction to External Object Repository:

One of the main burdens of automated GUI test script maintainability is the amount of maintenance needed when object properties change within the application under test. A very common way of minimizing the time it takes to update your automated test scripts is the use of a central object repository. An Object Repository is a map between UI element and its locator. Which can also be written as an Object Map between UI element and the way to find it. In Selenium WebDriver's context it means a mapping between WebElement and the corresponding locator type and value.
Unlike QTP/UFT, Selenium does not offer the default implementation for object repository. However, implementing and using a basic object repository is straightforward. A basic object repository can be implemented as a collection of key-value pairs, with the key being a logical name identifying the object and the value containing unique objects properties used to identify the object on a screen.

Demo – Implementing Object Repository

- Object Repository Using Properties file
- Object Repository Using XML file

Demo – Test Case Parameterization

- Test case parameterization using Excel Apache POI
- Test case parameterization using CSV
- Test case parameterization using XML & Data Provider in TestNG

Summary

- In this lesson you have learnt:
  - Introduction to Page Factory Design Pattern
  - Advantages of implementing Page Factory Design Pattern
  - How to implement Page Factory Design Pattern?
  - Difference between Page Object Model (POM) and Page Factory
  - Introduction to External Object Repository
  - How to implement External Object Repository
  - Advantages of implementing External Object Repository

Review Question

Question 1:

- _____ is an optimized version of the Page Object Model Design pattern.

Question 2:

Page Factory is an inbuilt page object model concept for Selenium WebDriver

- True
- False

Question 3:

Page Factory can be used in which of the below framework?

- Data Driven Framework
- Modular Framework
- Keyword Driven Framework
- All of the above

Review Question

Question 4:
Which annotations we use in Page Factory?

- initelement
- FindBy
- CacheLookup
- All of the above

Question 5:
_____ is a lazy load concept in Page Factory pattern.