## *Data Encapsulation*

- Encapsulation is defined as the process in which we wrap the data into a single unit.
- Encapsulation binds the data and code. It basically creates a shield and code cannot be accessed outside the shield or by any code outside the shield.

- The **variables in Java** or the method of the class are hidden from any other class and cannot be accessed outside the class.
- This process is called Data-hiding.
- Java Encapsulation can be achieved by declaring the **data** as private while the methods as public so that the variables can be accessed.

Here, data refers to variable and for any given variable we perform 3 steps-

- Declaration
- Initialization
- Utilization

Example Program:

```
package java_encapsulation;

public class A {

    // Declaration
    int i;

    // Initialization
    public A(int j) {
        i = j;
    }

    // Utilization
    public void print() {
        System.out.println("Value of I is: " + i);
    }
}
```

```
package java_encapsulation;

public class B {
    public static void main(String[] args) {
        A a = new A(5);
        a.print();
    }
}
```

Example:

```
package java_encapsulation;

class StudentOne {
    // Declaration
    private int stID;
    private double stMarks;

    // Initialization
    public StudentOne(int id, double marks) {
                this.stID = id;
                this.stMarks = marks;
            }

    // Utilization
    public void print() {
        System.out.println("Student ID: " + stID);
        System.out.println("Student ID: " + stMarks);
    }
}

public class Java_Bean_Class {
    public static void main(String[] args) {
        System.out.println("Program starts...");
        StudentOne st = new StudentOne(1, 65.89);
        st.print();
        System.out.println("Program ends...");

    }
}
```

## Encapsulation in Selenium

```
package java_encapsulation;

import org.openqa.selenium.By;

class LoginPage {
    //Declaration
    private WebElement unTextBox;

    //Initialization
    public LoginPage(WebDriver driver) {
        unTextBox = driver.findElement(By.id("username"));
    }

    //Utilization
    public void setUserName() {
        unTextBox.sendKeys("admin");
    }
}

public class MainMethod {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.actitime.com");
        LoginPage l = new LoginPage(driver);
        l.setUserName();

        driver.close();
    }
}
```



## Example program for login page

```
import org.openqa.selenium.By;

class LoginPage {
    // Declaration
    private WebElement unTextBox;
    private WebElement pwTextBox;
    private WebElement loginButton;

    // Initialization
    public LoginPage(WebDriver driver) {
        unTextBox = driver.findElement(By.id("username"));
        pwTextBox = driver.findElement(By.name("pwd"));
        loginButton = driver.findElement(By.id("loginButton"));
    }

    // Utilization
    public void setUserName(String username) {
        unTextBox.sendKeys(username);

    }

    public void setPassword(String password) {
        pwTextBox.sendKeys(password);
    }

    public void clickLogin() {
        loginButton.click();
    }
}

public class MainMethod {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.actitime.com");
        LoginPage l = new LoginPage(driver);
        l.setUserName("admin");
        l.setPassword("manager");
        l.clickLogin();

        driver.close();
    }
}
```

## Stale Element Reference Exception

### What is StaleElementReferenceException?

- Stale means old, decayed, and no longer fresh. Stale Element means an old element or no longer available element.

When do we get this exception?

- After finding the element and before performing the action on that element, if the page is refreshed then we get the stale element reference exception.

  **Example:** Assume there is an element that is found on a web page referenced as a WebElement in WebDriver. If the DOM changes then the WebElement goes stale. If we try to interact with an element which is staled then the **StaleElementReferenceException** is thrown.

### Program to raise the stale element reference exception?

```
package java_encapsulation;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class StaleElementException {
    public static void main(String[] args) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.actitime.com");
        driver.manage().window().maximize();

        WebElement userName = driver.findElement(By.id("username"));

        userName.sendKeys("admin");

        driver.navigate().refresh();
        Thread.sleep(5000);

        userName.sendKeys("admin");

        driver.close();
    }
}
```

Note: If you run the above program, you will get the below exception

```
INFO: Detected dialect: OSS
Exception in thread "main" org.openqa.selenium.StaleElementReferenceException: stale element reference: element is not attached to the page
  (Session info: chrome=69.0.3497.100)
```

Another example program to reproduce the stale element reference exception?

```
package java_encapsulation;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class StaleElementException {
    public static void main(String[] args) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.actitime.com");
        driver.manage().window().maximize();

        WebElement userName = driver.findElement(By.id("username"));
        WebElement password = driver.findElement(By.name("pwd"));
        WebElement loginButton = driver.findElement(By.id("loginButton"));

        //Invalid login
        userName.sendKeys("admin");
        password.sendKeys("hjdsfgsdf");
        loginButton.click();

        Thread.sleep(5000);

        //Valid login
        userName.sendKeys("admin");
        password.sendKeys("manager");
        loginButton.click();

        driver.close();
    }
}
```

## Common Causes

A stale element reference exception is thrown in one of two cases, the first being more common than the second:

- The element has been deleted entirely.
- The element is no longer attached to the DOM.

## The Element has been deleted

- The most frequent cause of this is that page that the element was part of has been refreshed, or
- The user has navigated away to another page.

**Note:** We face this stale element reference exception when the element we are interacting is destroyed and then recreated again. When this happens the reference of the element in the DOM becomes stale. Hence we are not able to get the reference to the element.

## The Element is not attached to the DOM

A common technique used for simulating a tabbed UI in a web app is to prepare DIVs for each tab, but only attach one at a time, storing the rest in variables. In this case, it's entirely possible

that your code might have a reference to an element that is no longer attached to the DOM (that is, that has an ancestor which is "document.documentElement").

**How to solve this issue! Or how do you handle stale element reference exception**

**Solution 1:** Page object model - (POM).

---

## *Page Object Model*

---

**What is POM?**

- Page Object model is an object design pattern in Java.
- Where web pages are represented as classes, and the various elements on the page are defined as variables on the class. All possible user interactions can then be implemented as methods on the class.

**What are the uses of page object model?**

- Using POM we can develop and test web pages.
- It is used to handle the stale element reference exception.
- Code reusability.
- Code maintainability.
- Object Repository.
- Readability.

**Why POM?**

Starting an UI Automation in Selenium WebDriver is NOT a tough task. You just need to find elements, perform operations on it.

Consider this simple script to login into a website,

```
import org.openqa.selenium.By;

/**
 * @author PriyaPramod
 * This test case will login in http://demo.actitime.com
 * Login to application
 * Verify the home page by getting the title of the Home Page
 */
public class TestClassWithOutPOM {
    public static void main(String[] args) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("http://demo.actitime.com");
        driver.manage().window().maximize();

        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.id("loginButton")).click();

        String homePageTitle = driver.getTitle();
        if(homePageTitle.contains("actiTIME - Enter Time-Track")) {
            System.out.println("PASS: Home page is displayed");
        }else {
            System.out.println("FAIL: Home page is not displayed");
        }
        driver.close();
    }
}
```

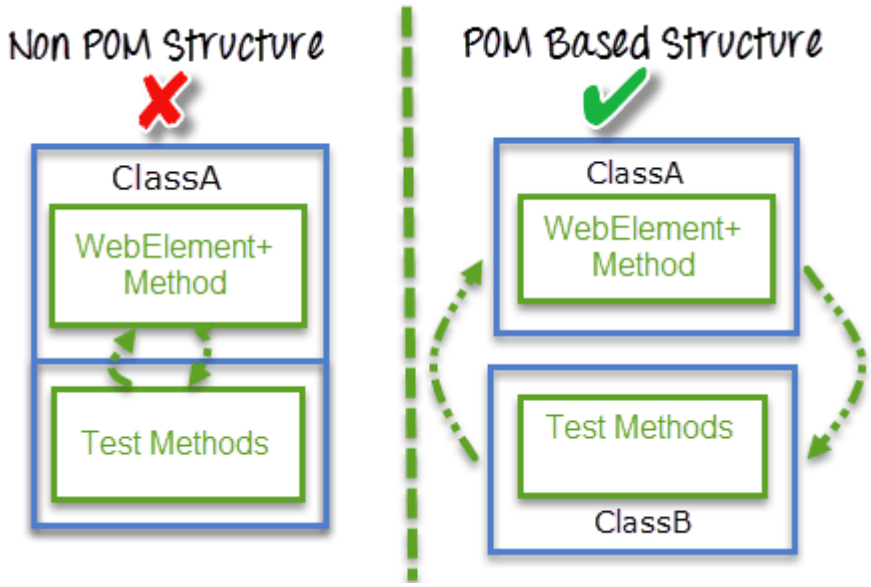
As you can observe, all we are doing is finding elements and filling values for those elements.

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them. This class can be reused in all the scripts using that element. In future, if there is a change in the web element, we need to make the change in just 1 class file and not 10 different scripts.

- This approach is called **Page Object Model (POM)**.
- It helps make the code **more readable, maintainable**, and **reusable.**




**Note:**

- Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too many duplicated code is used. Duplicated code could be caused by duplicated functionality and this will result in duplicated usage of locators. The disadvantage of duplicated code is that the project is less maintainable. If some locator will change, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. Beside of that it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests

with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts.

- Page Object Model Framework has now a days become very popular test automation framework in the industry and many companies are using it because of its easy test maintenance and reduces the duplication of code.

- The main advantage of Page Object Model is that if the UI changes for any page, it don't require us to change any tests, we just need to change only the code within the page objects (Only at one place). Many other tools which are using selenium, are following the page object model.

# Advantages of POM

- **Code reusability** – We could achieve code reusability by writing the code once and use it in different tests.
- **Code maintainability** – There is a clean separation between test code and page specific code such as locators and layout which becomes very easy to maintain code. Code changes only on Page Object Classes when a UI change occurs. It enhances test maintenance and reduces code duplication.
- **Object Repository –** Each page will be defined as a java class. All the fields in the page will be defined in an interface as members. The class will then implement the interface.
- **Readability –** Improves readability due to clean separation between test code and page specific code
- **Low maintenance**: Any User Interface changes can swiftly be implemented into the interface as well as class.
- **Programmer Friendly**: Robust and more readable. The Object-oriented approach makes the framework programmer friendly.
- **Low Redundancy**: Helps reduce duplication of code. If the architecture is correctly and sufficiently defined, the POM gets more done in less code.
- **Efficient & Scalable**: Faster than other keyword-driven/data-driven approaches where Excel sheets are to be read/written.

**Disadvantages**

1. **High Setup Time & Effort**: Initial effort investment in development of Automation Framework is high. This is the biggest weight of POM in case of web applications with hundreds/thousands of pages. It is highly suggested that if this model is decided to be implemented, then it should be done parallel to development of the application. Refer V-Model for Software Development Life Cycle.
2. **Skilled labour**: Testers not technically sound or aware of programming best practices are a nightmare in this case. Perhaps this is the biggest mistake to make, employing unskilled labour in hopes of training them during implementation.
3. **Specific**: Not a generic model. Automation Framework developed using POM approach is specific to the application. Unlike keyword-driven/data-driven frameworks, it is not a generic framework.

Irrespective of the disadvantages, POM is perhaps the most efficient and highly recommended approach towards any web application. As the framework matures it is perhaps easier to modify it into a hybrid framework from a POM approach than from other keyword/data driven approaches.

**How to implement POM?**

**Rules to Implement POM classes:**

1. Develop a separate class for every web page in the application.
    - For example: If we have a 10 web pages in an application, then we need to develop 10 classes.
2. Elements should be identified by using the FindBy or FindBys annotation.
3. All the web elements should be declared with private access specifier.
4. Develop public methods to perform the operation using the private web elements.
5. All the page classes should initialize using "PageFactory.initElements" method before calling the methods from POM classes.

Three ways we can implement the POM.

1. Simple POM
2. POM with page factory class
3. Advance POM

**Simple POM**

- It's the basic structure of Page object model (POM) where all Web Elements of the Application under Test (**AUT)** and the method that operate on these Web Elements are maintained inside a class file.
- A task like **verification** should be **separate** as part of Test methods.

```
public class LoginPage {

    WebDriver driver;
    public LoginPage(WebDriver driver)
    {
        this.driver = driver;
    }

    private By userName = By.id("username");
    private By password = By.name("pwd");
    private By loginButton = By.id("loginButton");

    public void enterUserName(String username)
    {
        driver.findElement(userName).sendKeys(username);
    }

    public void enterPassword(String pass)
    {
        driver.findElement(password).sendKeys(pass);
    }

    public void clickOnLoginButton()
    {
        driver.findElement(loginButton).click();
    }

}
```

**POM with page factory class:**

What is page factory?

- Page factory is a class which implements the POM concept.
- Page Factory is an inbuilt Page Object Model concept for Selenium WebDriver but it is much optimized.
- In order to support Page Object model, we use Page Factory. Page Factory in Selenium is an extension to Page Object and can be used in various ways. In this case we will use Page Factory to initialize web elements that are defined in web page classes or Page Objects.
- In POM class we declare the element using ***FindBy Annotation*** and we write it as ***@FindBy***. It should be imported from the following package.
- **Import org.openqa.selenium.support.FindBy;**

- In page object model using the page factory we use 3 annotation methods for declaring the web elements.

**@FindBy**

- The @FindBy annotation is used to locate one or more web elements using a single criterion.

**For identifying the single web element.**

@FindBy (Locator Type=”Locator Value”) Private WebElement ElementName;

Or

@FindBy (Locator Type=”Locator Value”)

Private WebElement ElementName;

Or

@FindBy (how = How.Locator, using = “Value”)

Private WebElement ElementName;

**For identifying the multiple element:**

@FindBy (Locator Type=”Locator Value”) Private List<WebElement> ElementName;

Or

@FindBy (Locator Type=”Locator Value”)

Private List<WebElement> ElementName;

Or

@FindBy (how = How.Locator, using = “Value”)

Private List<WebElement> ElementName;

**@FindBys**

- @FindBys annotation is used in case elements need to match all of the given criteria.

Syntax:

```
@FindBys ({@FindBy (how = How.NAME, using = "username"),
        @FindBy (how = How.NAME, using = "password"})
Private List<WebElement> ElementName;
```

**@FindAll**

- The *@FindAll* annotation is used in case elements need to match **at least one of the given criteria**

Syntax:

```
@FindAll ({@FindBy (how = How.NAME, using = "username"),
        @FindBy (how = How.NAME, using = "password"})
Private List<WebElement> ElementName;
```

**Note:**

- Web page classes or Page Objects containing web elements need to be initialized using **Page Factory class** before the web element variables can be used. \
- This can be done simply through the use of ***initElements*** function on Page Factory: all elements will get initialized.
- InitElements is a static method of page factory class. Which takes two arguments, driver instance and the class type and returns a page object with its fields fully initialized.

What happens if we don't use initElements statement?

- We get "NullPointerException".

Following are the different ways of using the PageFactory.InitElements method.

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

Or, even simpler:

```
LoginPage page = PageFactory.intElements(driver,LoginPage.class)
```

Or, inside the web page class constructor:

```
public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
}
```

-

**POM Code using Page Factory**

Sample POM class for login page:

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class LoginPage {

    @FindBy(id="username")
    private WebElement usernameTB;

    @FindBy(name="pwd")
    private WebElement passwordTB;

    @FindBy(id="loginButton")
    private WebElement loginButton;

    public void setUsername(String username) {
        usernameTB.sendKeys(username);
    }

    public void setPassword(String password) {
        passwordTB.sendKeys(password);
    }

    public void clickLoginButton() {
        loginButton.click();
    }
}
```

**Login test class**

```
package testScripts;

import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.PageFactory;

public class TestLogin {
    public static void main(String[] args) throws InterruptedException, IOException
    {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        LoginPage login = new LoginPage();
        PageFactory.initElements(driver, login);

        login.setUsername("admin");
        login.setPassword("manager");
        login.clickLoginButton();

        driver.close();
    }
}
```

Note: In the above login page class is called as POM class, whereas TestLogin is called as test class.

Example program to handle **StaleElementReferenceException** using POM class?

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class LoginPage {

    @FindBy(id="username")
    private WebElement usernameTB;

    @FindBy(name="pwd")
    private WebElement passwordTB;

    @FindBy(id="loginButton")
    private WebElement loginButton;

    public void setUsername(String username) {
        usernameTB.sendKeys(username);
    }

    public void setPassword(String password) {
        passwordTB.sendKeys(password);
    }

    public void clickLoginButton() {
        loginButton.click();
    }
}
```

```
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.PageFactory;

public class TestLogin {
    public static void main(String[] args) throws InterruptedException, IOException
    {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        LoginPage login = new LoginPage();
        PageFactory.initElements(driver, login);

        login.setUsername("admin");
        login.setPassword("managersdhfagk");
        login.clickLoginButton();
```

```
        String loginPageTitle = driver.getTitle();
        if(loginPageTitle.contains("actiTIME - Login")) {
            System.out.println("PASS: Home page is displayed");
        }else {
            System.out.println("FAIL: Home page is not displayed");
        }

        login.setUsername("admin");
        login.setPassword("manager");
        login.clickLoginButton();

        String homePageTitle = driver.getTitle();
        if(homePageTitle.contains("actiTIME - Enter Time-Track")) {
            System.out.println("PASS: Home page is displayed");
        }else {
            System.out.println("FAIL: Home page is not displayed");
        }

        driver.close();
    }
}
```

Developing the POM class to initialize in page classes?

POM class for login page.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {

    WebDriver driver;

    public LoginPage(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }

    @FindBy(how = How.ID, using = "username")
    private WebElement usernameTB;

    @FindBy(how = How.NAME, using = "pwd")
    private WebElement passwordTB;

    @FindBy(how = How.ID, using = "loginButton")
    private WebElement loginButton;

    public void login(String username, String password) {
        usernameTB.sendKeys(username);
        passwordTB.sendKeys(password);
        loginButton.click();
    }
}
```

**Test Class:**

```
package testScripts;

import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class TestLogin {
    public static void main(String[] args) throws InterruptedException, IOException
    {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        LoginPage login = new LoginPage(driver);
        login.login("admin", "manager");

        driver.close();
    }
}
```

**What's the difference between Page Object model and Page Factory?**

- Page Object model is a design pattern, as previously outlined in this section. Page Factory expands on Page Object model functionality by introducing more advanced features. It allows users to initialize specific elements within the Page Object model, using annotations.

**What are the types of classes we create while using the page object model?**

- We create two types of classes.
    - POM class
    - Test Class

**Is it mandatory to write "initElements" method inside the POM class?**

- No, however it is mandatory to initialize the elements before calling any method of POM class.

**How many POM class and Test class we need to develop?**

- Number of **POM class** depends on the number of **pages** in application.

- Number of **Test class** depends on the number of **Test Cases**.
- **Example**: If the application has 20 Pages with 100 test cases then we will develop 20 POM class and 100 test classes.

**Automate Login test cases using POM design pattern.**

**Login Test Case:**

| **Step no** | **Description** | **Input** | **Expected Result** | **Actual Result** | **Status** | **Comment** |
|---|---|---|---|---|---|---|
| **1** | Open the browser & enter the URL | https://demo.actitime.com | Login page should be displayed | Login Page is displayed | Pass | |
| **2** | Enter valid username and valid password and click on login | UN: admin<br>PW: manager | Home page should be displayed | Home page is displayed | Pass | |

**Automation script without using the POM class.**

```
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class LoginTestWithOutPOM {
    public static void main(String[] args) throws InterruptedException, IOException
    {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        // There are 3 ways to verify whether the page is displayed or not
        // First Check point: Title
        String eResult = "actiTime - Login";
        String aResult = driver.getTitle();
        if(aResult.equals(eResult)) {
            System.out.println("PASS: Login page is displayed");
        }else {
            System.out.println("FAIL: Login page is not displayed");
        }

        // Second Check Point: URL
        String eURL = "Login";
        String aURL = driver.getCurrentUrl();
        if (aURL.equals(eURL)) {
            System.out.println("PASS: Login page is displayed");
        } else {
            System.out.println("FAIL: Login page is not displayed");
        }

        // Third Check Point: Element
        WebElement usernameTB = driver.findElement(By.id("username"));
        if (usernameTB.isDisplayed()) {
            System.out.println("PASS: Login page is displayed");
        } else {
            System.out.println("FAIL: Login page is not displayed");
        }

        // Automation Script for the second test step
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.id("loginButton")).click();


        // Automation Script for the second test step
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.id("loginButton")).click();

        String eResult2 = "actiTime - Enter Time-Track";
        WebDriverWait wait = new WebDriverWait(driver, 6);
        try {
            wait.until(ExpectedConditions.titleIs(eResult2));
            System.out.println("PASS: Home page is displayed");
        } catch (Exception e) {
            System.out.println("FAIL: Home page is not displayed");
        }

        driver.close();
    }
}
```

**Optimized Automation Script: for the above test case:**

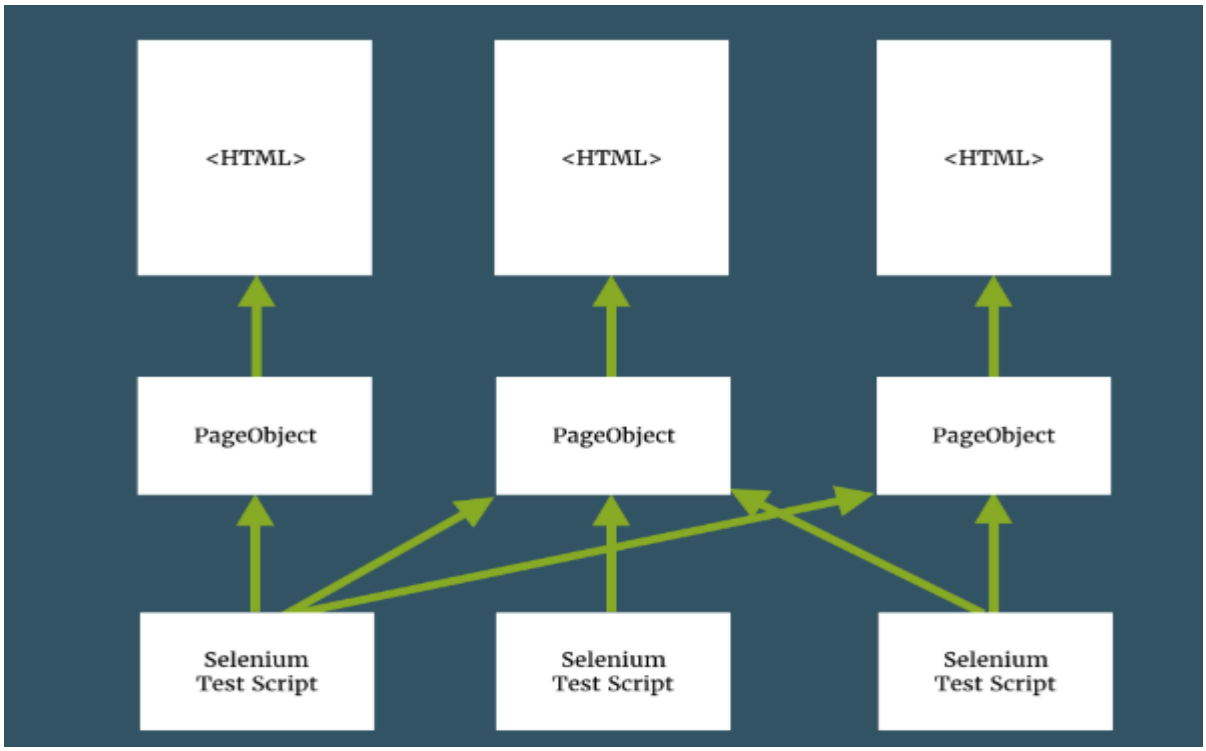
```
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class TestScriptWithOutPOM {

    static
    {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
    }

    public static void verifyPageDisplayed(WebDriver driver, String eResult)
    {
        WebDriverWait wait = new WebDriverWait(driver, 6);
        try {
            wait.until(ExpectedConditions.titleIs(eResult));
            System.out.println("PASS: Home page is displayed");
        } catch (Exception e) {
            System.out.println("FAIL: Home page is not displayed");
        }
    }


    public static void main(String[] args) throws InterruptedException, IOException {
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        verifyPageDisplayed(driver, "actiTime - Login");

        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.name("pwd")).sendKeys("manager");
        driver.findElement(By.id("loginButton")).click();

        verifyPageDisplayed(driver, "actiTime - Enter Time-Track");

        driver.close();
    }
}
```

So some of the typical problems of writing the Selenium test are:

- Test cases are difficult to read
- Changes in the UI breaks multiple tests often in several places
- Duplication of selectors both inside and across tests - no reuse

**Solution: POM**

- You create an object that represents the UI you want to test, which could be a whole page or a significant part of it. The responsibility of this object is to wrap HTML elements and encapsulate interactions with the UI, meaning that this is where all calls to WebDriver will go. This is where most WebElements are. And this is the only place you need to modify when the UI changes.
- This figure illustrates the pattern:




**Automation Scripts for Login Test Using POM design pattern:**

Before developing the POM classes, first execute the test case manually once. Identify the common behaviours and functionalities which is shared by all the page classes and develop it in base page. And extend the base page from all the page classes.

**NOTE: In Base page, we need to develop only the functionaries which is shared by all the page classes.**

**For Example:** In the above Login Test, in both the test steps, we are verifying the title of the page. Instead of repeating the same code in both the page classes. Develop the

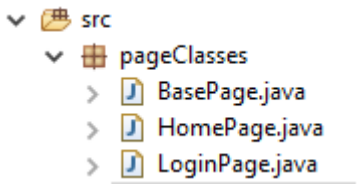
method which verifies the title in base page, and extend the base page. Now you can reuse the same code. Which helps in reuse.

**Create a Package, name it as PageClasses or POMclasses and develop following Three Classes:**

**BasePage ->** to develop reusable code which is shared by all the page classes.

**LoginPage ->** identify all the required web elements of login page and store and develop the methods.

**HomePage ->** identify all the required web elements of home page and store and develop the methods.




**BasePage Code:**

```
package pageClasses;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class BasePage {

    public void verifyPageDisplayed(WebDriver driver, String eResult)
    {
        WebDriverWait wait = new WebDriverWait(driver, 6);
        try {
            wait.until(ExpectedConditions.titleIs(eResult));
            System.out.println("PASS: Home page is displayed");
        } catch (Exception e) {
            System.out.println("FAIL: Home page is not displayed");
        }
    }

}
```

**LoginPage:**

```
package pageClasses;

import org.openqa.selenium.WebDriver;

public class LoginPage extends BasePage {

    public LoginPage(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }

    @FindBy(how = How.ID, using = "username")
    private WebElement usernameTB;

    @FindBy(how = How.NAME, using = "pwd")
    private WebElement passwordTB;

    @FindBy(how = How.ID, using = "loginButton")
    private WebElement loginButton;

    public void setUserName(String username) {
        usernameTB.sendKeys(username);
    }

    public void setPassword(String password) {
        passwordTB.sendKeys(password);
    }

    public void clickLoginButton() {
        loginButton.click();
    }
}
```

**HomePage:**

```
package pageClasses;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class HomePage extends BasePage {
    public HomePage(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }

    @FindBy(id="logoutLink")
    private WebElement logoutLink;

    public void logOut() {
        logoutLink.click();
    }

}
```

**Create a separate package called Test Script:**

- Under the Test scripts create Test classes based on the number of test cases.

  Example: If we have 10 TC, create 10 Test Scripts.
- In Test script classes, create the object of page classes and call the methods according to the test cases.

LoginTestScript:

```
package testScripts;

import java.util.concurrent.TimeUnit;

public class TestLogin {
    static {
        String Chrome_Key = "webdriver.chrome.driver";
        String Chrome_Value = "D:/Softwares/Drivers/chromedriver.exe";
        System.setProperty(Chrome_Key, Chrome_Value);
    }

    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(50, TimeUnit.SECONDS);
        driver.manage().window().maximize();
        driver.get("http://demo.actitime.com");

        LoginPage login = new LoginPage(driver);
        login.verifyPageDisplayed(driver, "actiTime - Login");
        login.setUserName("admin");
        login.setPassword("manager");
        login.clickLoginButton();

        HomePage home = new HomePage(driver);
        home.verifyPageDisplayed(driver, "actiTime - Enter Time-Track");
        home.logOut();

        driver.close();
    }
}
```

## Page Load timeout

We can **set the amount of time to wait for a page load to complete** before throwing an error.

Example:

```
driver.manage().timeouts().pageLoadTimeout(20, TimeUnit.SECONDS);
```

Once added in the script, the WebDriver instance waits for 20 seconds for every page to get loaded before throwing an exception. If the page is not loaded in 20 seconds of time, then it throws TimedOutException at run time.

**Best practices to develop the POM classes:**

There are a few best practices in using page objects, that you should make an effort to follow.

- A page object should not have any assertions
- A page object should represent meaningful elements of a page and not necessarily a complete page
- When you navigate you should return the page object for the next page.

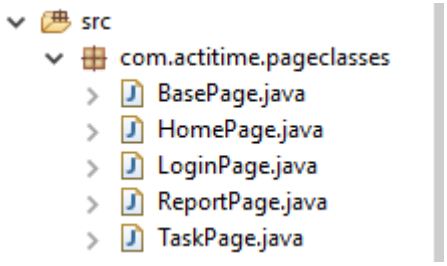
**Advance POM:**

**Create the Same Structure:**

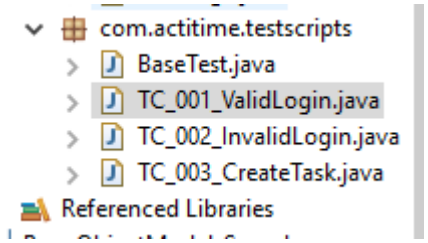
**BasePage:** For storing all the common elements which is shared by all the pages, common functionalities which is again used by all the page classes.

Example: In ActiTime application, navigate to any page. There are few elements like logout link, settings button, help button ETC. These elements are not going to change. For such kind elements which are common among every pages. Identify them and store it in base page. So that every web page can access since base page is extended by all the page classes.

Create Page classes based on number of pages are available in the Application under test.




Create the TEST SCRIPTS package, and develop test scripts based on the test case count.



BasePage:

```
package com.actitime.pageclasses;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.TimeoutException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;
import org.openqa.selenium.support.PageFactory;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class BasePage {

    WebDriver driver;
    public BasePage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(this.driver, this);
        this.driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
    }

    @FindBy(how = How.ID, using = "logoutLink")
    private WebElement logoutLink;

    public void logout() {
        logoutLink.click();
    }

    public Boolean verifyPageDisplayed(String title) {
        Boolean isTitle = false;
        try {
            WebDriverWait wait = new WebDriverWait(driver, 10);
            isTitle = wait.until(ExpectedConditions.titleContains(title));

        } catch (TimeoutException e) {
            System.out.println(e);
        }
        return isTitle;
    }
```

LoginPage:

```
package com.actitime.pageclasses;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;

public class LoginPage extends BasePage {

    public LoginPage(WebDriver driver) {
        super(driver);
    }

    @FindBy(how = How.ID, using = "username")
    private WebElement usernameTB;

    @FindBy(how = How.NAME, using = "pwd")
    private WebElement passwordTB;

    @FindBy(how = How.ID, using = "loginButton")
    private WebElement loginButton;

    @FindBy(how = How.XPATH, using = "//span[@class='errormsg' and not(@id='errorSpan')]")
    private WebElement errorMessage;

    public HomePage navigateToHomePage(String username, String password) {
        usernameTB.sendKeys(username);
        passwordTB.sendKeys(password);
        loginButton.click();

        return new HomePage(this.driver);
    }

     }

     public String getErrorMessge() {
         return errorMessage.getText();
     }

     public void enterUsername(String username) {
         usernameTB.sendKeys(username);
     }

     public void enterPassword(String password) {
         passwordTB.sendKeys(password);
     }

     public void clickLoginButton() {
         loginButton.click();
     }
}
```

HomePage:

```
package com.actitime.pageclasses;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;

public class HomePage extends BasePage {

    public HomePage(WebDriver driver) {
        super(driver);
    }

    @FindBy(how = How.XPATH, using = "//div[text()='TASKS']")
    private WebElement taskTab;

    @FindBy(how = How.XPATH, using = "//div[text()='REPORTS']")
    private WebElement reportTab;

    public TaskPage navigateToTaskPage() {
        taskTab.click();
        return new TaskPage(this.driver);
    }

    public ReportPage navigateToReportPage() {
        reportTab.click();
        return new ReportPage(this.driver);
    }
}
```

TaskPage

```
package com.actitime.pageclasses;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class TaskPage extends BasePage {

    public TaskPage(WebDriver driver) {
        super(driver);
    }

    @FindBy(xpath = "//div[text()='Add New Task']")
    private WebElement addNewTaskButton;

    @FindBy(xpath = "//div[text()='Create new tasks']")
    private WebElement createNewTaskButton;

    public void clickOnCreateNewTask() {
        addNewTaskButton.click();
        createNewTaskButton.click();
    }
}
```

Since, there are few steps are common for all the test cases, like launching the browser and closing the browser. Developing the BaseTest class to keep all the repetitive steps and calling the functions from BaseTest in all the Test classes.

Base_Test:

```
package com.actitime.testscripts;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BaseTest {

    public static WebDriver driver;
    public static void launchBrowser()
    {
        System.setProperty("webdriver.chrome.driver", "D:/Softwares/Drivers/chromedriver.exe");
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.get("https://demo.actitime.com/");
    }

    public static void closeBrowser()
    {
        driver.close();
    }
}
```

Test Script_1: Valid Login

```
package com.actitime.testscripts;

import com.actitime.pageclasses.HomePage;
import com.actitime.pageclasses.LoginPage;

public class TC_001_ValidLogin {
    public static void main(String[] args) {
        BaseTest.launchBrowser();
        LoginPage login = new LoginPage(BaseTest.driver);
        HomePage home = login.navigateToHomePage("admin", "manager");

        Boolean isTrue = home.verifyPageDisplayed("Enter Time-Track");
        if(isTrue) {
            System.out.println("PASS: Home page is displayed");
        }else {
            System.out.println("FAIL: Home page is not displayed");
        }
        login.logout();
        BaseTest.closeBrowser();
    }
}
```

Test Script_2: Invalid Login

```
package com.actitime.testscripts;

import com.actitime.pageclasses.LoginPage;

public class TC_002_InvalidLogin {
    public static void main(String[] args) {
        BaseTest.launchBrowser();
        LoginPage login = new LoginPage(BaseTest.driver);
        login.enterUsername("adminss");
        login.enterPassword("nasdhgasdf");
        login.clickLoginButton();

        String errorMessage = login.getErrorMessge();

        if (errorMessage.contains("Username or Password is invalid. Please try again.")) {
            System.out.println("PASS: Error message is displaying");
        } else {
            System.out.println("FAIL: Error message is not displaying");
        }

        BaseTest.closeBrowser();
    }
}
```

Test Script_3: Create Task

```
package com.actitime.testscripts;

import com.actitime.pageclasses.HomePage;
import com.actitime.pageclasses.LoginPage;
import com.actitime.pageclasses.TaskPage;

public class TC_003_CreateTask {
    public static void main(String[] args) {
        BaseTest.launchBrowser();
        LoginPage login = new LoginPage(BaseTest.driver);
        HomePage home = login.navigateToHomePage("admin", "manager");
        TaskPage task = home.navigateToTaskPage();
        task.clickOnCreateNewTask();

        /*
         *
         * Write a code to create the task
         *
         */

        task.logout();
        BaseTest.closeBrowser();
    }
}
```