4. Page Object Model (POM)

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.

1. Create a '*New Package*' file and name it as 'pageObjects'

2. Create a '*New Class*' file and refer the name. In our case it is Register Page

- Create abstract class to extend some methods

```
package pageObjects;

import org.openqa.selenium.WebDriver;

public abstract class BaseClass {
   public static WebDriver driver;
   public static boolean bResult;

   public  BaseClass(WebDriver driver){
      BaseClass.driver = driver;
      BaseClass.bResult = true;
   }

}
```

```
package pageObjects;

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

/**
 * Created by nguyenthihongthuy on 10/08/17.
 */
public class RegisterPage extends BaseClass{

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


    @FindBy(how= How.NAME, using="name")
    public static WebElement username;

    @FindBy(how=How.NAME, using="phone")
    public static WebElement phone;

    @FindBy(how=How.NAME, using="email")
    public static WebElement email;
}
```

3. Create a '*New Package*' file and name it as 'modules'

- Create RegisterAction.java

Noted: In action files, you are not only do multi actions inside but also do the single action

```
package modules

import helpers.Log;
import org.openqa.selenium.WebDriver;
import pageObjects.RegisterPage;

import java.util.HashMap;

/**
 * Created by nguyenthihongthuy on 10/08/17.

             egisterAction {


 //multi action
    public static void Execute(WebDriver driver,HashMap<String,String>
map) throws Exception{

        Log.info("Type username");
        RegisterPage.username.sendKeys(map.get("username"));
        Thread.sleep(2000);
        Log.info("Type phone");
        RegisterPage.phone.sendKeys(map.get("phone"));
        Thread.sleep(2000);
    }

   //single action
   public static void TypeUserName(WebDriver
driver,HashMap<String,String> map) throws Exception{
        Log.info("Type username");
        RegisterPage.username.sendKeys(map.get("username"));
   }

}
```

4. Create a '*New Package*' file and name it as 'resources'

- Create register.feature, you can reference 3. First Cucumber Scripts (BDD Style)

5. Create a '*New Package*' file and name it as 'stepsdefinition'

With RunCukesTest.java, SetUp.java you can reference 3. First Cucumber Scripts (BDD Style)

- Create RegisterDefinition.java

```
package stepsdefinition;

import cucumber.api.java.en.And;
```

```
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import modules.RegisterAction;
import org.openqa.selenium.support.PageFactory;
import pageObjects.RegisterPage;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * Created by nguyenthihongthuy on 10/08/17.
 */
public class RegisterDefinition {


    @Given("^user navigates to (.+)$")
    public void user_navigates_to(String website) throws Throwable {
        SetUp setup = new SetUp();
        setup.setupDriver();
        SetUp.driver.get(website);
        SetUp.datamap = new ArrayList<HashMap<String, String>>();

    }

    @When("^user register in using Name as (.+), Phone as (.+), Email as
(.+), Country as (.+), City as (.+), Username as (.+), Password as
(.+)$")
    public void
user_register_in_using_name_as_phone_as_email_as_country_as_city_as_user
name_as_password_as(String name, String phone, String email, String
country, String city, String username, String password) throws Throwable
{
        HashMap<String, String> sampledData = new HashMap<String,
String>();
        sampledData.put("username", name);
        sampledData.put("phone", phone);
        SetUp.datamap.add(sampledData);
        PageFactory.initElements(SetUp.driver, RegisterPage.class);
        RegisterAction.Execute(SetUp.driver, SetUp.datamap.get(0));


    }

    @Then("^user submit information$")
    public void user_submit_information() throws Throwable {


    }
```

```
    @And("^Welcome page should be displayed$")
    public void welcome_page_should_be_displayed() throws Throwable {

    }

}
```

Noted: Before working on the Page, you should init all elements insides the Page . After that you can work on it as codes belows:

```
PageFactory.initElements(SetUp.driver, RegisterPage.class);
    RegisterAction.Execute(SetUp.driver, SetUp.datamap.get(0));
```

5. Structure