***POM(Page Object Model)

-->POM is a java design pattern preferred by google to develop object repository

OR

-->it is a java design pattern used by Automation Test Eng. to design Test Script.

Ques: Why POM?

-->It is a well organized structure design pattern, Where we can maintain all the webELement in page wise,

due to POM design pattern maintains and modification of element is easy & faster.

**Advantages of POM:

1: Well organized strucutre

2: Handle StaleElement exception

3: Maintains & modification of element is easy

***POM Class Rules:

1: No. of webpages that need to be handled should be handled by creating POM class for each page

2: No. of POM class= No. of webpages in application

3: In every POM class to handle Webpage element, data memebers need to be created

4: No. of data members in POM class = No. of element in webpage

5: Every data member in POM class should follow encapsulation.

  1: Declare globally with access specifier Private

  2: Initilaize within constructor with access specifier public

  3: Utilize data members using methods

6: POM class will not contain main method

7: To run POM class, we use test class which contain main method

Examp: UN<input type="text" id="username">

1: Normal Code

driver.findElement(By.id(username));

2: Encapsulation

```
WebElement un;
un= driver.findElement(By.id(username));
un.sendkeys("abc");
```

3: POM class

```
Class LoginPage
{
private WebElement un;
public LoginPage(WebDriver driver)
{
un= driver.findElement(By.id("username"));
}
public void setUsername()
{
un.sendkeys("abc");
}
}

Class LoginPageTest
{
main()
{
WebDriver driver = new FirefoxDriver();
driver.get("");
LoginPage rv = new LoginPage(driver);
rv.setUsername();
}
}
```

Script:

```
package pom;

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

public class ActiLogin
{
	//var--->declaration
	private WebElement username;
	private WebElement password;
	private WebElement checkbox;
	private WebElement loginbutton;

	//var------>Initialized
	public ActiLogin(WebDriver driver)
	{
		username=driver.findElement(By.id("username"));
		password=driver.findElement(By.name("pwd"));
		checkbox=driver.findElement(By.id("keepLoggedInCheckBox"));
		loginbutton=driver.findElement(By.id("loginButton"));
	}

	//var---->Utilization
	public void enterUsername(String name)
	{
		username.sendKeys(name);
	}
   public void enterPassword(String pwd)
```

```
    {
              password.sendKeys(pwd);
    }
    public void selectCheckbox()
    {
              checkbox.click();
    }
    public void clickOnLoginButton()
    {
              loginbutton.click();
    }
}

package pom;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class ActiLoginTest {

          public static void main(String[] args)
          {
                    WebDriver driver = new FirefoxDriver();
                    driver.get("http://localhost/login.do");
                    ActiLogin act = new ActiLogin(driver);
                    act.enterUsername("admin");
                    act.enterPassword("manager");
                    act.selectCheckbox();
                    act.clickOnLoginButton();


          }
```

}

AssignQues: WATS to login facebook App using POM

Ques: WATS to display error msg as an output

Script:

```
package pom;

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

public class ActiLoginInvalid
{
	private WebElement login;
	private WebElement errormsg;

	public ActiLoginInvalid(WebDriver driver)
	{
		login=driver.findElement(By.id("loginButton"));
		errormsg=driver.findElement(By.xpath("//span[@class='errormsg']"));
	}
	public void invalidLogin()
	{
		login.click();
	}
	public void error()
	{
		if(errormsg.isDisplayed())
		{
```

```
                              String str = errormsg.getText();
                              System.out.println(str);
                    }
          }



}
package pom;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class ActiInvalidtTest {

          public static void main(String[] args)

          {
                    WebDriver driver = new FirefoxDriver();
                    driver.get("http://localhost/login.do");
                    ActiLoginInvalid  rv = new ActiLoginInvalid(driver);
                    rv.invalidLogin();
                    rv.error();
                    driver.close();

          }

}
```

Output: StaleElementReferenceException

-->Whenever webdriver try to identify an element, element was avaliable in GUI but at the time of performing action on elements, element was not recognized due to page got refresheds or elements may become old or element

not attached to page in such case we get StaleElementReferenceException

----

**PageFactory

--> It is a class which contain static method initElements() which is used to initialize POM class data members

-->PageFactory.initElements(WebDriver arg, Object arg.);

--> If intialization in POM class constructor, then object arg. should be this keyword which will indicates

 initilaizing elements of same page

-->initElements() method will initialize data memebers by identifying elements in webpage using @FindBy

 @FindBy(Locator type="Locator value")

-->While executing constructor initElements() method will convert @Findby annotation to findelement().

Script:

```
package pom;


import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;


public class ActiInvalid_PF

{

	@FindBy(id="loginButton")

	private WebElement login;

	@FindBy(xpath="//span[@class='errormsg']")

	private WebElement errormsg;
```

```
        public ActiInvalid_PF(WebDriver driver)
        {
                PageFactory.initElements(driver, this);
        }
        public void invalidLogin()
        {
                login.click();
        }
        public void error()
        {
                if(errormsg.isDisplayed())
                {
                        String str = errormsg.getText();
                        System.out.println(str);
                }
        }
}

package pom;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class ActiInvalidTestClassPF {

        public static void main(String[] args)
        {
                WebDriver driver = new FirefoxDriver();
                driver.get("http://localhost/login.do");
                ActiInvalid_PF  rv=  new ActiInvalid_PF(driver);
                rv.invalidLogin();
```

```
		rv.error();
		driver.close();

	}

}
```

AssignQues: WATS to login https://www.saucedemo.com/ using POM Concept

AssignQues: WATS to display all the links Text as an output in Google webpage using POM

Script:

```
package pom;

import org.openqa.selenium.WebElement;
import java.util.List;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class Alllinks
{
	@FindBy(xpath="//a")
	private List<WebElement> links;

	public Alllinks(WebDriver driver)
	{
		PageFactory.initElements(driver,this);
	}
	public void getLinkText()
	{
		int s=links.size();
		for(int i=0;i<s;i++)
```

```
                    {
                              WebElement rv = links.get(i);
                              System.out.println(rv.getText());
                    }
          }



}
package pom;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class AllLInksTest {

          public static void main(String[] args)
          {
                    WebDriver driver = new FirefoxDriver();
                    driver.get("https://www.google.com/");
                    Alllinks rv = new Alllinks(driver);
                    rv.getLinkText();
                    driver.close();




          }

}
```

***Difficulties in executing Test Classes

1: Executing each test script manually is a time consuming process.

2: Generating test execution report will be difficult

3: Tester will not get sufficent time to automate new test cases

4: Executing failed test script will be difficult

**To overcome above difficulties we use TestNG(NG=Next Generation)

TestNG**

-->It is a java unit framework designed to write test classes

Advantages:

1: We can execute script using suite file

2: Automatic test execution report preparation

3: We can execute any failed scripts

4: supoorts cross browser testing.

Procedure to configure testNG with Eclipse

1: Open Eclipse-->select Eclipse market place in help option

-->Enter TestNG in search field and search

-->Click on install button for testNG

-->follow installation steps and click on finish, Restart Eclipse

----

1: To execute test class, test methods in TestNG we use annotation like @Test

2: After executing test class, Refresh the project

3: "Test Output" folder will be created which contains emailiable report

4: To print test class output in report we use Reporter class static method log

Examp: Reporter.log("");

Script_1:

package testNG;

```
import org.testng.Reporter;
import org.testng.annotations.Test;

public class Script_1
{
	@Test
	public void test()
	{
		//System.out.println("hello");---display output in console
		//Reporter.log("hello");  ---display output in report but not in console
		Reporter.log("helloo",true);
	}

}
```
