Featured Page Object Model in Selenium | Feature Framework

Table of content

Page Object Model | Feature Framework in Selenium

In the last tutorial, we have seen how to create a basic POM framework, but in organization level framework should serve all the needs.

In this tutorial, we are going to learn how to implement page Object Model with full features using selenium webdriver

A good Framework should support :

Reusable methods

Logging

Low maintenance

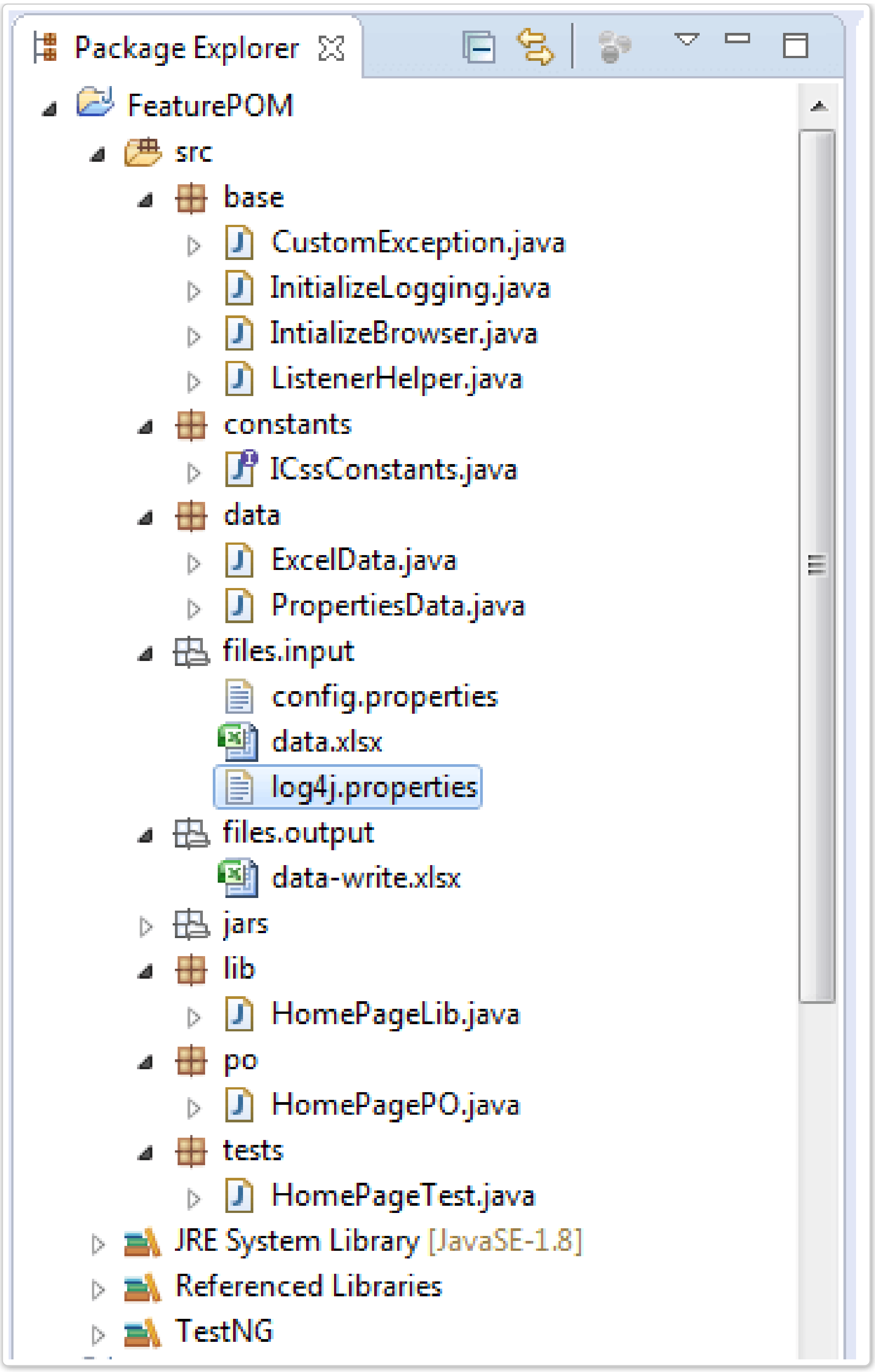
Reporting

Test case mapping with manual test cases

Data Changes without compiling

Handling non Supported application
We will learn how to implement all these features in the page object model

Page Object Model Folder Structure



```
Package Explorer
FeaturePOM
  src
    base
      CustomException.java
      InitializeLogging.java
      IntializeBrowser.java
      ListenerHelper.java
    constants
      ICssConstants.java
    data
      ExcelData.java
      PropertiesData.java
    files.input
      config.properties
      data.xlsx
      log4j.properties
    files.output
      data-write.xlsx
    jars
    lib
      HomePageLib.java
    po
      HomePagePO.java
    tests
      HomePageTest.java
  JRE System Library [JavaSE-1.8]
  Referenced Libraries
  TestNG
```

Page object model contains several packages, let's learn about each package and its purpose in the framework.

base : base package will contain all the initialization files like opening browser, initializing logging

constants : constants package will have all the constant files like CSS properties, Attribute properties.

data : data package will have all the data handling java files like handling excel, handling properties java files.

files : files folder will have all the file s used in our framework which includes input and output files like excel sheets, properties fils

jars : jars folder will contain all the jar files related to our framework

lib : lib package will have all the re-usable libraries po : po package will have all the page object java classes

test : The test package will have all the test cases in the framework.

Logging with Log4j

Logging is essential for the framework; without logs, the user may not be able to find the execution point.

We will be using log4j for logging purposes; you can find the complete tutorial about logging on **Log4j with Selenium Webdriver**.

Steps for Logging :

1. Create InitializeLogging class under base package; All remaining classes are going to extend this class

2. Create a static block, as we are not going to change anything in the log4j properties during run time, we can hard code the values.

Static Block : static block will be executed once the class is loaded to stack; static block executes even before the main method.

3. Create a variable called log for logging all the logs from the framework, which should be static and public so that we can access across test cases.

4. Configure the properties file using *configure* method from the PropertyConfigurator class

log4j properties file

```
# Root logger denotes where to write and priority level to write
log4j.rootLogger=INFO, file

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:UsersuserPicturesfirstLogOutput.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10

# layout of the log output pattern
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%t] %-5p %c %x - %m%n
```

Complete program for logging with log4j:

```
package base;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class InitializeLogging {
   public static Logger log = Logger.getLogger("Regression");
   // static block
   static{
      //PropertiesConfigurator is used to configure logger from properties file

      PropertyConfigurator.configure("C:UsersuserPictureslog4j.properties");

      //log the message to file
      log.trace("Logging Initialized");
   }
}
```

Listeners in Selenium

Handle Excel data

It is important for any framework to work with data changes without re-compilation, Apache POI helps us to read data from the excel sheet in selenium webdriver.

Below are the steps to read Excel:

1. Create java class ExcelData under *data* package, ExcelData should extend InitializeLogging class

2. To support multi-threading, we have to write all the methods as non-static methods.

3. Let's write a method called readExcel, which accepts the Excel file name, Excel Sheet name, row number, column number.

4. Read the data by creating an object to Workbook class, call getStingCellvalue() method to get the value of the cell

5. We can log the data that we retrieved from the excel.

6. Return the cell value

Steps to writing Excel in data :

1. In ExcelData class write one more non-static method called writeExcel, writeExcel method should accept Excel file name, Excel Sheet name, row number, column number, and data to write

2. We can set the cell value using setCellValue() method present in Apache POI

3. Log the data that we about to write to the excel.

Complete Excel reading and writing program :

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

public class ExcelData {
   public String readData(String inputFile, String sheetname, int rowNumber, int columnNumber) throws Exception {
      // create file input stream object for the excel sheet
      FileInputStream fis = new FileInputStream(inputFile);
      // create object for work book
      Workbook wb = WorkbookFactory.create(fis);
      //create object for sheet present in excel using Workbook object 'wb'
      Sheet sheet = wb.getSheet(sheetname);
      //create object for row present in sheet using Sheet object 'sheet'
      Row row = sheet.getRow(rowNumber);
      //create object for cell present in row using Row object 'row'
      Cell cell = row.getCell(columnNumber);
      // return the value present in the excel sheet cell
      return cell.getStringCellValue();
   }

   public void writeData(String outputFile, String sheetName, int rowNumber, int columnNumber, String dataToWrite) throws Exception
      // create file input stream object for the excel sheet
      FileInputStream fis = new FileInputStream(outputFile);
      // create object for work book
      Workbook wb = WorkbookFactory.create(fis);
```

```
        //create object for sheet present in excel using Workbook object 'wb'
        Sheet sheet = wb.getSheet(sheetName);
        //create object for row present in sheet using Sheet object 'sheet'
        Row row = sheet.getRow(rowNumber);
        //create object for cell present in row using Row object 'row'
        Cell cell = row.createCell(columnNumber);
        cell.setCellValue(dataToWrite);
        FileOutputStream fos = new FileOutputStream(outputFile);
        wb.write(fos);
    }
}
```

FindElement with Selenium

Handle Properties file

When we do not have much data or when we have data which almost never going to change, we can use data in Properties file such kind of situations — details like username, password, application URL.

Steps to Read properties file :

1. Create a Java class called PropertiesData under data package

2. Create a non-static method called readProperty, which should accept the following parameters Properties file name, Property name. readProperty method should return a string value.

3. Create an object for the Properties class

4. Load FileInputStream object into properties file using load method present in Properties class object

5. Get the values using 'get("key")' method or getProperty("key") method by passing key as the patrameter

Steps to Write properties file :

1. Write a non-static method called writeProperty, which should accept the following parameters Properties file name, Property name, property value.

2. Create a Properties class object to access the property file.

3. Set the values using setProperty("key", "Value") method by passing key and values as the parameters

Complete Properties file reading and writing program :

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Properties;

public class PropertiesData {
    public String readProperty(String fileName, String property) throws Exception {
        // create file input stream object for the properties file
        FileInputStream fis = new FileInputStream(fileName);
        // create Properties class object to access properties file
        Properties prop = new Properties();
        // load the properties file
        prop.load(fis);
        // return the property value
        return prop.getProperty(property);
    }

    public void writeProperty(String outputFile, String property, String propertyValue) throws Exception{
        // create file output stream object for the properties file
        FileOutputStream fis = new FileOutputStream(outputFile);
        // create Properties class object to access properties file
        Properties prop = new Properties();
        // load the properties file
        // set the properties
        prop.setProperty(property, propertyValue);
        // store the file into local system
        prop.store(fis, null);
    }
}
```

Implementing Class Constructor locator in Selenium

Initialization of driver

We have to initialize our browser, and we have to store it in the driver variable. We should initialize the browser at the topmost class in the page object model so that we can use the driver across all other scripts. This class should not have any scripts.

We should also make the browser invocation on run time, i.e., we should choose our browser on run time by getting the data either from an excel sheet or from the Properties file.

Steps to Initialize browser :

1. Create a class called IntializeBrowser which should extend InitializeLogging for logging purpose

2. Initialize the 'browser' variable with the object of the EventFiringWebDriver to listen to all the events occurring in the framework. This browser only will be used across all the test cases for accessing all the elements from the webpage.

3. Read properties file to know which browser to invoke

4. We have to write if else condition block to decide which browser to open based on the value retrieved from the properties file.

5. Log the browser name using log4j

6. We have to modularize the class for easy debugging purpose

Complete program for IntializeBrowser

```
package base;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;
import org.openqa.selenium.support.events.EventFiringWebDriver;

import data.PropertiesData;

public class IntializeBrowser extends InitializeLogging {
    // this browser variable will be used through the framework
    public static EventFiringWebDriver browser;

    public static void main(String[] args) throws Exception {
        // initialize driver
        WebDriver driver = initializeDriver();
        // Create object for Listener implementation
        ListenerHelper listen = new ListenerHelper();
        // create object event listener object
        browser = new EventFiringWebDriver(driver);
        // register even listener object
```

```
		browser.register(listen);

	}

	private static WebDriver initializeDriver() throws Exception
	{
		WebDriver driver = null;
		PropertiesData pd = new PropertiesData();
		String propertiesFilePath = "C:Usersuserworkspace1FeaturePOMsrcfilesinputconfig.properties";
		String browserToChoose = pd.readProperty(propertiesFilePath, "browser");
		log.info("Trying to browser : "+browserToChoose);
		if(browserToChoose.equalsIgnoreCase("firefox") || browserToChoose.equalsIgnoreCase("ff")){
			System.setProperty("webdriver.gecko.driver", "C:/~/geckodriver.exe");
			driver = new FirefoxDriver();

		}else if(browserToChoose.equalsIgnoreCase("chrome") || browserToChoose.equalsIgnoreCase("gc")){
			System.setProperty("webdriver.chrome.driver", "C:/~/chromedriver.exe");
			driver = new ChromeDriver();
		}else if(browserToChoose.equalsIgnoreCase("internet explorer") || browserToChoose.equalsIgnoreCase("ie")){
			System.setProperty("webdriver.ie.driver", "C:/~/IEdriverserver.exe");
			driver = new InternetExplorerDriver();
		}
		log.info(browserToChoose+" browser opened successfully");
		return driver;
	}
}
```

Constants in Page Object Model

We will be using some constants in our application framework, instead of writing those constants on test or lib files, we can write all of them under constants package.

For example, we may be using CSS properties like 'font, color, background-color, padding-top, margin-left, font-size..'

Steps to create constants :

1. Create a java Interface called ICssContants under constants package.

2. Write all the properties as a public static String variable.

3. We are writing the string snot the CSS values

4. In this way, we can avoid the creation of string during run time, and also can avoid duplicate strings.

ICssConstants file:

```
package constants;

public interface ICssConstants {
	public static String FONT_SIZE = "font-size";
	public static String COLOR = "color";
	public static String FONT_WEIGHT = "font-weight";
	public static String PADDING_TOP = "padding-top";
}
```

Close and Quit in Selenium

Page Objects

We are going to use google.com as project URL, so we have to create PageObject Classes based on the

1. Google Home Page,

2. Google Search Result Page

We are going to use non-static methods as page objects and which returns WebElement or WebElements

Steps to create PageObjects :

1. Create a class called HomePagePO which inherits IntializeBrowser, so that we can access browser variable directly

2. Create non-static methods which return webelement or web elements
3. Create a header for teach method you create in the framework
4. write log statements based on our need

Complete HomePagePO file :

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import base.IntializeBrowser;

public class HomePagePO extends IntializeBrowser {
   /**
    * @author user
    * @return WebElement
    */
   public static WebElement searchBar(){
      return browser.findElement(By.name("q"));
   }

   /**
    * @author user
    * @return WebElement
    */
   public static WebElement searchButton(){
      return browser.findElement(By.xpath("//input[contains(@value, 'Search')]"));
   }
   /**
    * @author user
    * @return WebElement
    */
   public static WebElement googleBanner(){
      return browser.findElement(By.id("hplogo"));
   }
}
```

Library Files

Sometimes we may have a repeating test step across the script; in such cases, we have to create those steps as re-usable methods. Re-usable methods can be used across multiple scripts in the framework

Steps to create a Library :

1. Library Name should be Same as PageObject Class name except for the PO at the end; library file names will have Lib in the end like HomePageLib, SearchResultPageLib...

2. Create a class called HomePageLib under lib package,

3. Import the static member of InitializeLogging class

4. We have to create an object for the PO class that we are going to use in this Lib file. Eg:HomePagePO hp = new HomePagePO();

5. Write all the methods as non-static methods, and return the values if required.

6. We can add a log statement based on our need

Complete HomePageLib

```
import static base.InitializeLogging.*;
import po.HomePagePO;

public class HomePageLib {
   HomePagePO hp = new HomePagePO();

   public void searchGoogle(String searchTerm){
      hp.searchBar().sendKeys(searchTerm);
      log.info("Searching google for the term : "+searchTerm);
      hp.searchButton().click();
      log.info("Google search button clicked");
   }
}
```

Note: For example purpose, I have created only one method

Handle Multiple Windows in Selenium

Unit Testing Framework | TestNG

When we want to create test cases, we should have a unit testing framework integrated with our application. With selenium java, we can use either TestNG or JUnit. For this framework, we will be utilizing the TestNG unit testing framework.

Steps to integrate TestNG :

1. Right-click on the project and click Properties Option
2. Click On the Add Library
3. Choose TestNG from available libraries
4. Click Next and Finish

Note: Please visit for **TestNG with Selenium Webdriver** tutorials section for more Details

Test Cases With TestNG in Selenium Webdriver

We have created a test case class to create test scripts, which are nothing but automation scripts for our manual test cases. We are using the TestNG tool for mapping manual and automation test cases. Every @Test denotes that it is a test case, and the result will be generated based on this test.

Steps to create test case Class :

1. Create a class called HomePageTest, name of the class should be in accordance with PO and Lib classes

2. Import the static member of the InitializeBrowser class so that we can access browser variable.

3. Create a non-static method to write the test steps and annotate the method with @Test annotation

4. based on our need we can write @Before and @After annotations

5. Log details if required, details like what test case, test case name, test case number, what step is running

The complete file of HomePageTest

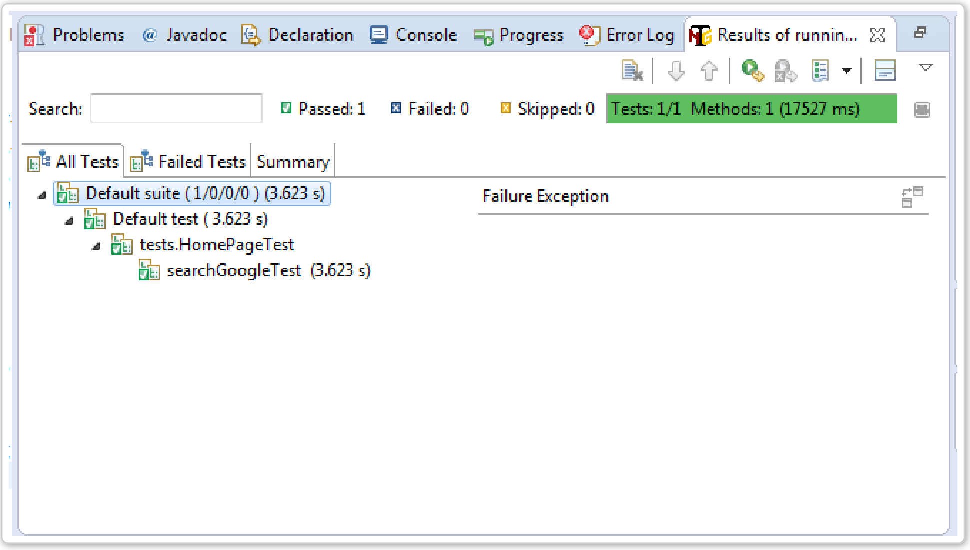
```
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
import static base.IntializeBrowser.*;

import lib.HomePageLib;

public class HomePageTest {

   HomePageLib hpLib = new HomePageLib();
   @BeforeClass
   public void setUp(){
      browser.get("https://google.com");
   }
   @Test
   public void searchGoogleTest(){
      hpLib.searchGoogle("first test with POM");
   }
   @AfterClass
   public void tearDown(){
      browser.quit();
   }
}
```

Locators in Selenium

Result and Logging

Result :

1. On the eclipse console, we can check the result under the TestNG section.

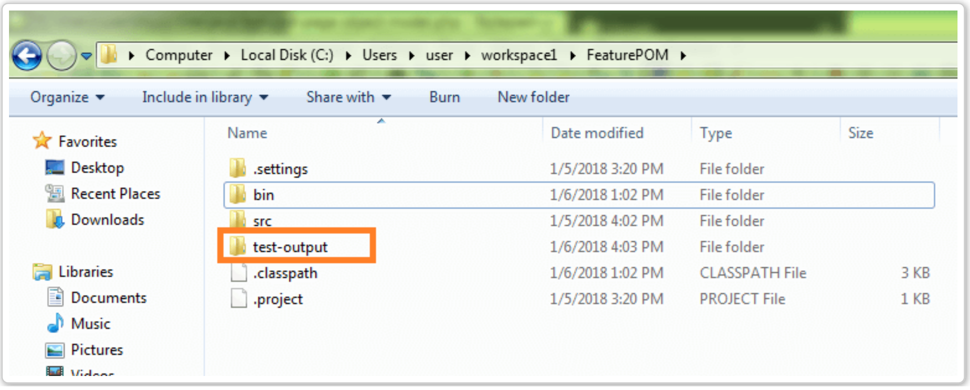


Search: Passed: 1 Failed: 0 Skipped: 0 Tests: 1/1 Methods: 1 (17527 ms)

```
Default suite ( 1/0/0/0 ) (3.623 s)
  Default test ( 3.623 s)
    tests.HomePageTest
      searchGoogleTest  (3.623 s)
```

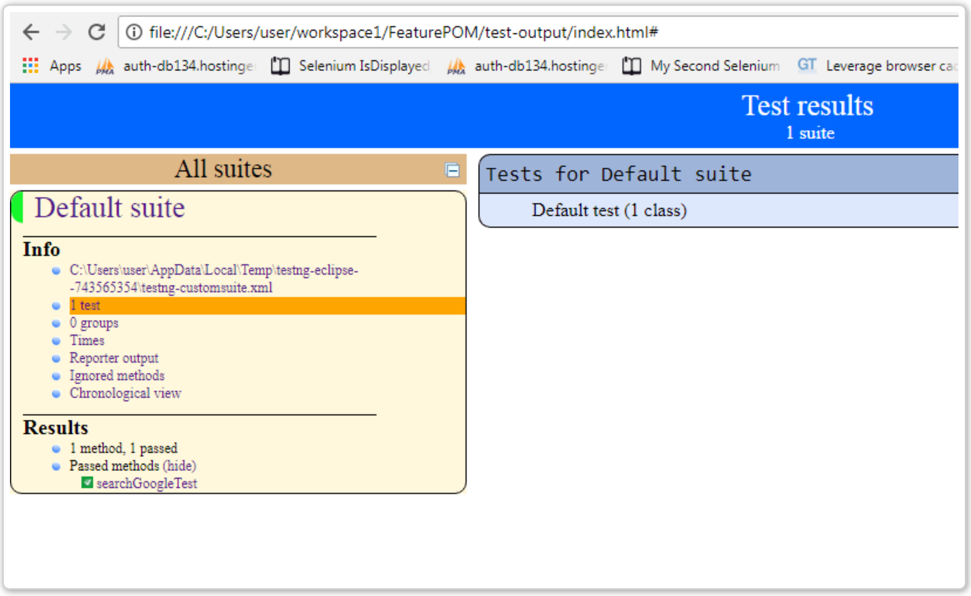
2. We can also Check the HTML report from the TestNG by navigating to the test-output folder under the project and open index.html

test-output folder

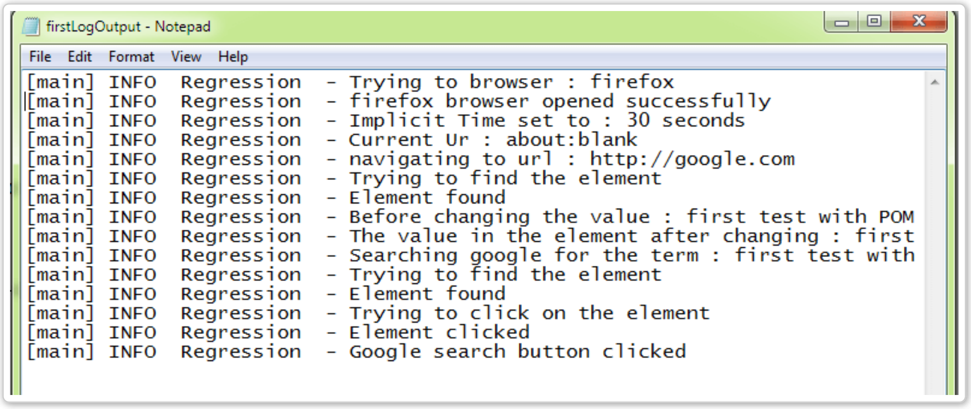


Name	Date modified	Type	Size
.settings	1/5/2018 3:20 PM	File folder	
bin	1/6/2018 1:02 PM	File folder	
src	1/6/2018 4:02 PM	File folder	
test-output	1/6/2018 4:03 PM	File folder	
.classpath	1/6/2018 1:02 PM	CLASSPATH File	3 KB
.project	1/5/2018 3:20 PM	PROJECT File	1 KB

Index.html



Logging :



```
[main] INFO  Regression  - Trying to browser : firefox
[main] INFO  Regression  - firefox browser opened successfully
[main] INFO  Regression  - Implicit Time set to : 30 seconds
[main] INFO  Regression  - Current Ur : about:blank
[main] INFO  Regression  - navigating to url : http://google.com
[main] INFO  Regression  - Trying to find the element
[main] INFO  Regression  - Element found
[main] INFO  Regression  - Before changing the value : first test with POM
[main] INFO  Regression  - The value in the element after changing : first
[main] INFO  Regression  - Searching google for the term : first test with
[main] INFO  Regression  - Trying to find the element
[main] INFO  Regression  - Element found
[main] INFO  Regression  - Trying to click on the element
[main] INFO  Regression  - Element clicked
[main] INFO  Regression  - Google search button clicked
```

Learn OOPS in Java with selenium

Recommended Readings

- SonarLint / SonarQube With Selenium
- GIT | BitBucket | SourceTree with Selenium
- Maven with Selenium
- Jenkins Integration with Selenium
- AutoIT in Selenium | Keyboard & Mouse
- CSV Files in Selenium
- Apache POI to read write excel in Selenium Webdriver
- Properties file with Java in Selenium | prop.getProperty()