**Why Framework**

A framework defines the organization’s way of doing things – a ‘Single Standard’. Following this standard would result in the project team achieving:

## Script-less representation of Automated tests:

The testing framework should offer point-and-click interface for accessing and interacting with the application components under test—as opposed to presenting line after line of scripting. Testers should be able to visualize each step of the business scenario, view and edit test cases intuitively. This will shorten the learning curve for testers and help QA teams meet deadlines.

## Data Driven tests:

[](http://toolsqa.wpengine.com/wp-content/uploads/2014/02/Excel.png)A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort. Test Automation Frameworks must have capability to integrate with spreadsheets and provide powerful calculation features.

## Standard Scripting and Team Consistency:

Scripting standard should be maintained across the framework library creation, which includes business components, system communications, data check points, loggers, reporters etc. Project team should follow the defined scripting standards. Published standards across the project team pre-empt the effort involved in duplicate coding, which prevent individuals from following their own coding standards.

## Encapsulation from Complexities:

Test engineers are encapsulated from the complexities and critical aspects of the code. Engineers are exposed only to the implemented libraries and tests are executed by just invoking the libraries.

## Implement and Maximize Re-Usability:

## Key Driven tests:

Objects.properties

# Log4j Introduction

**Why logging is important in any application?**

Logging is very important to any application. It helps us collect information about how the application is running and also helps us debug if any failure occurs.

**What is Log4j?**

Log4j is a brilliant logging API available both on Java and .net framework. Advantages are:  
*– Log4j allows you to have a very good logging infrastructure without putting in any efforts.*  
*– Log4j gives you the ability to categorize logs at different levels (Trace, Debug, Info, Warn, Error and Fatal).*  
*– Log4j gives you the ability to direct logs to different outputs. For e.g. to a file, Console or a Database.*  
*– Log4j gives you the ability to define the format of output logs.*  
*– Log4j gives you the ability to write Asynchronous logs which helps to increase the performance of the application.*  
*– Loggers in Log4j follow a class hierarchy which may come handy to your applications.*

If you are not able to understand any of these points than don’t worry. Things will get clearer as we approach to the end of Log4j Tutorial series.

**Log4j**

Log4j consists of four main components

***– LogManager***  
***– Loggers***  
***– Appenders***  
***– Layouts***  
With these comes some additional components which will be covered in the individual headings in following tutorials

**LogManager**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package Log4jSample;    import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;    public class SampleEntry {    //mainLogger is a logger object that we got from LogManager. All loggers are          //using this method only. We can consider LogManager as a factory to create          //Logger objects  static Logger mainLogger = LogManager.getLogger(SampleEntry.class.getName());    public static void main(String[] args) {  // TODO Auto-generated method stub    BasicConfigurator.configure();  mainLogger.info("This is just a logger");    }  } |

**Logger**

This is a class which helps you log information at different logging levels. In the above sample code you can see that we have created a logger named mainLogger using the LogManager static class. Now we can use it to write logs. As you can see we have mainLogger.info(“Comments that you want to log”) statement which logs the string.

**Appenders**

Appenders are objects which help Logger objects write logs to different outputs. Appenders can specify a file, console or a data base as the output location.

In this code sample you will see that we have used a console appender to print logs like we would do using System.out or System.err.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package Log4jSample;    import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.ConsoleAppender;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;    public class SampleEntry {    //All the loggers that can be used  static Logger mainLogger = LogManager.getLogger(SampleEntry.class.getName());    public static void main(String[] args) {  // TODO Auto-generated method stub    BasicConfigurator.configure();    //Create a console appender and attach it to our mainLogger  ConsoleAppender appender = new ConsoleAppender();  mainLogger.addAppender(appender);  mainLogger.info("This is just a logger");  }  } |

**Layouts**

Layout class help us define how the log information should appear in the outputs. Here is a sample code which uses *PatternLayout* Class to change the formatting of logs:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package Log4jSample;    import java.util.Enumeration;    import org.apache.log4j.Appender;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.ConsoleAppender;  import org.apache.log4j.Layout;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  import org.apache.log4j.PatternLayout;    public class SampleEntry {    //All the loggers that can be used  static Logger mainLogger = LogManager.getLogger(SampleEntry.class.getName());    public static void main(String[] args) {  // TODO Auto-generated method stub    BasicConfigurator.configure();  ConsoleAppender appender = new ConsoleAppender();  appender.activateOptions();  PatternLayout layoutHelper = new PatternLayout();  layoutHelper.setConversionPattern("%-5p [%t]: %m%n");  layoutHelper.activateOptions();          //mainLogger.getAppender("ConsoleAppender").setLayout(layoutHelper);  appender.setLayout(layoutHelper);  mainLogger.addAppender(appender);  //Create a console appender and attach it to our mainLogger  mainLogger.info("Pattern 1 is displayed like this");  layoutHelper.setConversionPattern("%C %m%n");  mainLogger.info("Pattern 2 is displayed like this");    }    } |

* Hybrid Automation Framework
  + Automation Framework
  + Page Object Model
  + Modular Driven Framework
  + Functions Parameters
  + Constant Variables
  + Data Driven Framework
  + Log4j Logging
  + TestNG Reporting
  + User Defined Function
  + Object Repository
  + Exception Handling

# 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.

**Note:**We will follow the same example which we have used in ***First Test Case***. Let’s assume it our base test case and implement the Page Object Model (POM) in it.

## How to do it…

1. Create a ‘**New Package‘** file and name it as ‘**pageObjects’**, by right click on the Project and select **New** > **Package**. We will be creating different packages for Page Objects, Utilities, Test Data, Test Cases and Modular actions. It is always recommended to use this structure, as it is easy to understand, easy to use and easy to maintain.

2. Create a **‘New Class‘** file and refer the name to the actual page from the test object, by right click on the above created Package and select **New** > **Class**. In our case it is **Home** **Page** and **LogIn** **Page**.

3. Now create a **Static Method** for each **Element** (Object) in the Home Page. Each method will have an **Argument** (driver) and a **Return** value (element).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package pageObjects;        import org.openqa.selenium.By;        import org.openqa.selenium.WebDriver;        import org.openqa.selenium.WebElement;    public class Home\_Page {        private static WebElement element = null;    public static WebElement lnk\_MyAccount(WebDriver driver){        element = driver.findElement(By.id("account"));        return element;        }    public static WebElement lnk\_LogOut(WebDriver driver){        element = driver.findElement(By.id("account\_logout"));    return element;        }    } |

**Driver** is being passed as an **Argument** so that Selenium is able to locate the element on the browser (driver).

**Element** is returned, so that an **Action** can be performed on it.

**Method** is declared as **Public Static**, so that it can be called in any other method without **instantiate** the class.

Follow the same rule for creating **LogIn Page** class.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package pageObjects;    import org.openqa.selenium.\*;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.WebElement;    public class LogIn\_Page {            private static WebElement element = null;        public static WebElement txtbx\_UserName(WebDriver driver){             element = driver.findElement(By.id("log"));             return element;             }         public static WebElement txtbx\_Password(WebDriver driver){             element = driver.findElement(By.id("pwd"));             return element;             }         public static WebElement btn\_LogIn(WebDriver driver){             element = driver.findElement(By.id("login"));             return element;             }    } |

4) Create a ‘***New Class***‘ and name it as **POM\_TC** by right click on the ‘**automationFramework**‘ Package and select **New** > **Class**. We will be creating all our test cases under this package.

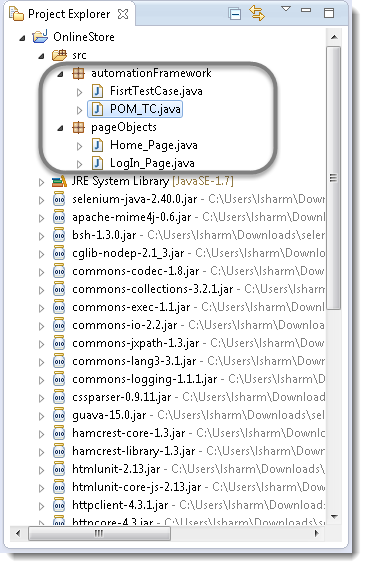
Now convert your old ***First Test Case*** in to the new Page Object Model test case.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | package automationFramework;         import java.util.concurrent.TimeUnit;         import org.openqa.selenium.WebDriver;         import org.openqa.selenium.firefox.FirefoxDriver;         // Import package pageObject.\*         import pageObjects.Home\_Page;         import pageObjects.LogIn\_Page;    public class PageObjectModel {         private static WebDriver driver = null;       public static void main(String[] args) {         driver = new FirefoxDriver();         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);         driver.get("http://www.store.demoqa.com");         // Use page Object library now         Home\_Page.lnk\_MyAccount(driver).click();         LogIn\_Page.txtbx\_UserName(driver).sendKeys("testuser\_1");         LogIn\_Page.txtbx\_Password(driver).sendKeys("Test@123");         LogIn\_Page.btn\_LogIn(driver).click();         System.out.println(" Login Successfully, now it is the time to Log Off buddy.")        Home\_Page.lnk\_LogOut(driver).click();         driver.quit();         }    } |

You will notice that once you type Home\_Page in your test script and the moment you press dot, all the methods in the Home Page will display. We can expose methods in order to reduce duplicated code. We are able to call these method multiple times. This will ensure a better maintainable test code, because we only have to make adjustments and improvements in one particular place.

Your Project explorer window will look like this now.



# Modular Driven Framework

In most of the web application we have few set of actions which are always executed in the series of actions. Rather than writing those actions again and again in our test, we can club those actions in to a method and then calling that method in our test script. Modularity avoids duplicacy of code. In future if there is any change in the series of action, all you have to do is to make changes in your main modular method script. No test case will be impacted with the change.

## How to do it..

Look for repeated functionality in your application for example the ‘login’ functionality. We can simple wrap this functionality in a method and we can give it a sensible name.

1) Create a ‘**New Package**‘ file and name it as ‘**appModule’**, by right click on the Project and select **New** > **Package**. We will be creating different packages for Page Objects, Utilities, Test Data, Test Cases and Modular actions. It is always recommended to use this structure, as it is easy to understand, easy to use and easy to maintain.

2) Create ‘**New Class**‘ and name it as **SignIn\_Action** by right click on package ‘**appModule**‘ and select **New** > **Class.** It will add new class ‘SignIn\_Action’ under package ‘appModule’.

3) Now create a **Public** **Static Void** Method and name it as **Execute** and club the following steps in to it:

* Click on the My Account link.
* Enter Username
* Enter Password
* Click on the Submit button

This method will not have any **Argument** (driver) and **Return** value as it is a **void** method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package framework.appModule;           import org.openqa.selenium.WebDriver;           import framework.pageObject.Home\_Page;           import framework.pageObject.LogIn\_Page;    public class SignIn\_Action{           public static void Execute(WebDriver driver){       Home\_Page.lnk\_SignIn(driver).click();       LogIn\_Page.txtbx\_UserName(driver).sendKeys("testuser\_1");       LogIn\_Page.txtbx\_Password(driver).sendKeys("Test@123");       LogIn\_Page.btn\_LogIn(driver).click();       }    } |

**Note:** Please visit **First Test Case** & **Page Object Model** first, in case you are finding it hard to understand the above used SignIn\_Action class.

4) Create a **New Class** and name it as **Module\_TC** by right click on the ‘**automationFramework**‘ Package and select **New** > **Class**. We will be creating all our test cases under this package.

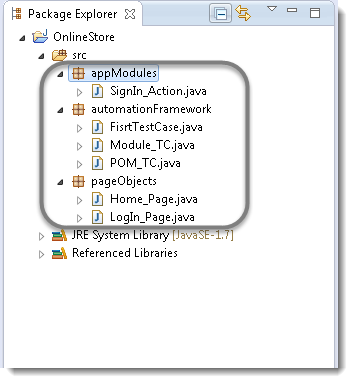
Now convert your old **POM\_TC** in to the new Module based test case.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package automationFramework;            import java.util.concurrent.TimeUnit;            import org.openqa.selenium.WebDriver;            import org.openqa.selenium.firefox.FirefoxDriver;            // Import package appModule.\*            import appModules.SignIn\_Action;            import pageObjects.Home\_Page;       public class Module\_TC {            private static WebDriver driver = null;         public static void main(String[] args) {            driver = new FirefoxDriver();            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);            driver.get("http://www.store.demoqa.com");            // Use your Module SignIn now            SignIn\_Action.Execute(driver);            System.out.println("Login Successfully, now it is the time to Log Off buddy.");            Home\_Page.lnk\_LogOut(driver).click();            driver.quit();         }    } |

You will notice that your call to SignIn\_Action will automatically execute all the steps mentioned under it.

Your Project explorer window will look like this now.



# Function Parameters

It is always a good practice to pass parameters when calling the method, rather than providing parameters inside the method. We can pass parameters through methods, just as in normal programming code. The code below will show us how we can login with parameterized username and password.

## How to do it…

1) First let’s have a look over our previous example of **SignIn\_Action** class**.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package framework.appModule;             import org.openqa.selenium.WebDriver;             import framework.pageObject.Home\_Page;             import framework.pageObject.LogIn\_Page;      public class SignIn\_Action{          public static void Execute(WebDriver driver){             Home\_Page.lnk\_SignIn(driver).click();             LogIn\_Page.txtbx\_UserName(driver).sendKeys("testuser\_1");             LogIn\_Page.txtbx\_Password(driver).sendKeys("Test@123");             LogIn\_Page.btn\_LogIn(driver).click();          }    } |

**Note:** Please visit **First Test Case, Page Object Model**& **Modularity** first, in case you are finding it hard to understand the above used SignIn\_Action class.

2) Modify the above Execute method of class SignIn\_Action to accept string Arguments (Username & Password).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package appModules;            import org.openqa.selenium.WebDriver;            import pageObjects.Home\_Page;            import pageObjects.LogIn\_Page;      public class SignIn\_Action {         // Pass Arguments (Username and Password) as string         public static void Execute(WebDriver driver,String sUsername, String sPassword){            Home\_Page.lnk\_MyAccount(driver).click();            // Enter sUsername variable in place of hardcoded value            LogIn\_Page.txtbx\_UserName(driver).sendKeys(sUsername);            // Enter sPassword variable in place of hardcoded value            LogIn\_Page.txtbx\_Password(driver).sendKeys(sPassword);            LogIn\_Page.btn\_LogIn(driver).click();            }    } |

3) Create a **New Class** and name it as **Param\_TC** by right click on the ‘**automationFramework**‘ Package and select **New** > **Class**. We will be creating all our test cases under this package.

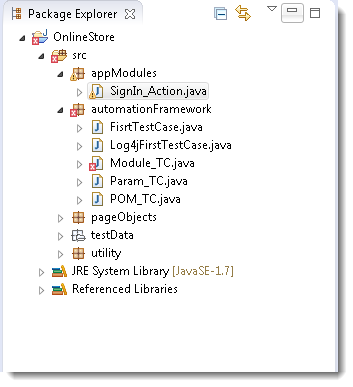
Now convert your old **Module\_TC**in to the new passing parameters based test case.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package automationFramework;            import java.util.concurrent.TimeUnit;            import org.openqa.selenium.WebDriver;            import org.openqa.selenium.firefox.FirefoxDriver;            import appModules.SignIn\_Action;            import pageObjects.Home\_Page;      public class Param\_TC {            private static WebDriver driver = null;         public static void main(String[] args) {            driver = new FirefoxDriver();            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);            driver.get("http://www.store.demoqa.com");            // Pass the Arguments (Username and Password) to this method            SignIn\_Action.Execute(driver, "testuser\_1","Test@123");            System.out.println("Login Successfully, now it is the time to Log Off buddy.");            Home\_Page.lnk\_LogOut(driver).click();            driver.quit();            }    } |

You see how easy it is to pass arguments to your methods. It increase you code readability.

Now your Project explorer window will look like this:



**Note:** But it is still not the best idea to hardcode your input anywhere in your test script, as it will still impact the bulk of your test scripts if there is any change in user data. I used this example just to give you an idea to how we use arguments in method.

# Constant Variables

Test data can be of two types, fixed or variable. If it is fixed, we can easily hard code the test data in to our test scripts. But sometimes the fixed test data is also used in so many scripts and if it gets changed then it is a huge task to update all the effected test scripts for example the URL of your test application. It remains same but once you shifted to other environment, you need to change it in all of your test scripts. We can easily place the URL in Text file or Excel file outside our test scripts but Java gives us special feature of creating Constants variables which works exactly the same as Environment and Global variable in QTP.

## How to do it…

1) Create a ‘**New Package**‘ file and name it as “**utility**“, by right click on the Project and select **New** > **Package**.

2) Create a ‘**New Class**‘ file, by right click on the above created Package and select **New** > **Class** and name it as **Constant**.

3) Assign keywords in this class to your fixed data for e.g. Url, Username and Password.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | package utility;       public class Constant {           public static final String URL = "http://www.store.demoqa.com";           public static final String Username = "testuser\_1";           public static final String Password = "Test@123";       } |

**Constants Variables** are declared as **public static**, so that they can be called in any other methods without **instantiate** the class.

**Constant Variables** are declared a **final**, so that they cannot be changed during the execution.

4) SignIn\_Class will remain same.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package appModules;            import org.openqa.selenium.WebDriver;            import pageObjects.Home\_Page;            import pageObjects.LogIn\_Page;      public class SignIn\_Action {         public static void Execute(WebDriver driver,String sUsername, String sPassword){            Home\_Page.lnk\_MyAccount(driver).click();            LogIn\_Page.txtbx\_UserName(driver).sendKeys(sUsername);            LogIn\_Page.txtbx\_Password(driver).sendKeys(sPassword);            LogIn\_Page.btn\_LogIn(driver).click();         }    } |

**Note:** Please visit **First Test Case, Page Object Model,** **Modular Driven** & **Function Parameters** first, in case you are finding it hard to understand the above used SignIn\_Action class.

5) Create a **New Class** and name it as **Global\_Var\_TC** by right click on the ‘**automationFramework**‘ Package and select **New** > **Class**. We will be creating all our test cases under this package.

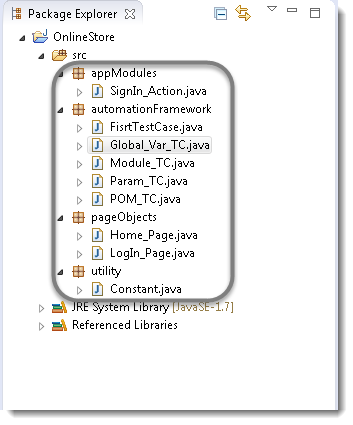
Now pass your **Constant Variables**(URL, Username and Password) as arguments to your **Execute** method of **SignIn\_Action** class in your **Global\_Var\_TC** test case.



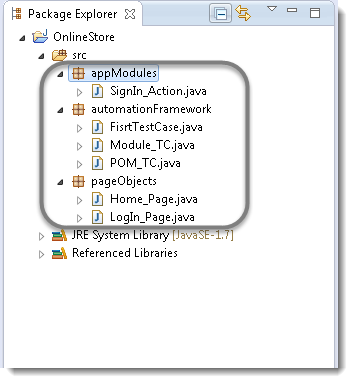
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | package automationFramework;            import java.util.concurrent.TimeUnit;            import org.openqa.selenium.WebDriver;            import org.openqa.selenium.firefox.FirefoxDriver;            import pageObjects.Home\_Page;            import appModules.SignIn\_Action;            // Import package utility.\*            import utility.Constant;      public class Global\_Var\_TC {            private static WebDriver driver = null;         public static void main(String[] args) {            driver = new FirefoxDriver();            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);            // Launch the Online Store Website using Constant Variable            driver.get(Constant.URL);            // Pass Constant Variables as arguments to Execute method            SignIn\_Action.Execute(driver,Constant.Username,Constant.Password);            System.out.println("Login Successfully, now it is the time to Log Off buddy.");            Home\_Page.lnk\_LogOut(driver).click();            driver.quit();         }    } |

You will notice that once you type ‘Constant’ and the moment you press dot, all the variables in the Constant class will display. We can expose variables in order to reduce duplicated code. We are able to call these Constant variables multiple times. This will ensure a better maintainable test code, because we only have to make adjustments and improvements in one particular place.

Your Project explorer window will look like this now.



When the test data is not fixed or if the same test script can be executed with the large amount of test data, we use external files for test data. In next chapter of Data Driven Technique – Apache POI you will see how to set large amount of variable test data for the test scripts.



Data Driven Framework with Apache POI – Excel

Most commercial automated software tools on the market support some sort of data driven testing, which allows you to automatically run a test case multiple times with different input and validation values. As Selenium Webdriver is more an automated testing framework than a ready-to-use tool, you will have to put in some effort to support data driven testing in your automated tests. I usually prefer to use Microsoft Excel as the format for storing my parameters. An additional advantage of using Excel is that you can easily outsource the test data administration to someone other than yourself, someone who might have better knowledge of the test cases that need to be run and the parameters required to execute them.

Reading data from the Excel

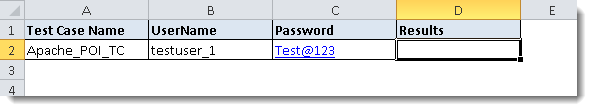
We need a way to open this Excel sheet and read data from it within our Selenium test script. For this purpose, I use the Apache POI library, which allows you to read, create and edit Microsoft Office-documents using Java. The classes and methods we are going to use to read data from Excel sheet are located in the org.apache.poi.hssf.usermodel package.

How to do it…

1) Download JAR files of Apache POI  and Add Jars to your project library. You can download it from here. That’s all about configuration of Apache POI with eclipse. Now you are ready to write your test.

2) Create a ‘New Package‘ file and name it as ‘testData’, by right click on the Project and select New > Package. Place all of your test data in this folder (package) whether it is a sql file, excel file or anything.

3) Place a Excel file in the above created package location and save it as TestData.xlsx. Fill the data in the excel like below image:



4) Add two constant variables (testData package path & Excel file name) in the Constant class.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package utility;       public class Constant {          public static final String URL = "http://www.store.demoqa.com";          public static final String Username = "testuser\_1";          public static final String Password = "Test@123";          public static final String Path\_TestData = "D://ToolsQA//OnlineStore//src//testData//"          public static final String File\_TestData = "TestData.xlsx"       } |

5) Create a ‘New Class‘ file, by right click on the ‘utility‘ Package and select New > Class and name it as ‘ExcelUtils‘. First we will write basic read/write methods.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | package utility;             import java.io.FileInputStream;                import java.io.FileOutputStream;                import org.apache.poi.xssf.usermodel.XSSFCell;             import org.apache.poi.xssf.usermodel.XSSFRow;             import org.apache.poi.xssf.usermodel.XSSFSheet;             import org.apache.poi.xssf.usermodel.XSSFWorkbook;        public class ExcelUtils {             private static XSSFSheet ExcelWSheet;             private static XSSFWorkbook ExcelWBook;             private static XSSFCell Cell;             private static XSSFRow Row;         //This method is to set the File path and to open the Excel file, Pass Excel Path and Sheetname as Arguments to this method         public static void setExcelFile(String Path,String SheetName) throws Exception {           try {               // Open the Excel file    FileInputStream ExcelFile = new FileInputStream(Path);    // Access the required test data sheet    ExcelWBook = new XSSFWorkbook(ExcelFile);    ExcelWSheet = ExcelWBook.getSheet(SheetName);    } catch (Exception e){    throw (e);    }    }         //This method is to read the test data from the Excel cell, in this we are passing parameters as Row num and Col num             public static String getCellData(int RowNum, int ColNum) throws Exception{           try{               Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);               String CellData = Cell.getStringCellValue();               return CellData;               }catch (Exception e){    return"";               }        }         //This method is to write in the Excel cell, Row num and Col num are the parameters         public static void setCellData(String Result,  int RowNum, int ColNum) throws Exception {           try{               Row  = ExcelWSheet.getRow(RowNum);    Cell = Row.getCell(ColNum, Row.RETURN\_BLANK\_AS\_NULL);    if (Cell == null) {    Cell = Row.createCell(ColNum);    Cell.setCellValue(Result);    } else {    Cell.setCellValue(Result);    }              // Constant variables Test Data path and Test Data file name               FileOutputStream fileOut = new FileOutputStream(Constant.Path\_TestData + Constant.File\_TestData);               ExcelWBook.write(fileOut);               fileOut.flush();    fileOut.close();    }catch(Exception e){    throw (e);    }    }    } |

6) Once we are done with writing Excel functions we can go ahead and modify the SignIn\_Action module to accept the test data from excel file.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package appModules;            import org.openqa.selenium.WebDriver;            import pageObjects.Home\_Page;            import pageObjects.LogIn\_Page;            import utility.ExcelUtils;        // Now this method does not need any arguments        public class SignIn\_Action {    public static void Execute(WebDriver driver) throws Exception{    //This is to get the values from Excel sheet, passing parameters (Row num &amp; Col num)to getCellData method    String sUserName = ExcelUtils.getCellData(1, 1);    String sPassword = ExcelUtils.getCellData(1, 2);    Home\_Page.lnk\_MyAccount(driver).click();    LogIn\_Page.txtbx\_UserName(driver).sendKeys(sUserName);    LogIn\_Page.txtbx\_Password(driver).sendKeys(sPassword);    LogIn\_Page.btn\_LogIn(driver).click();            }    } |

Note: In the later chapters we will see how to parameterise the row column as well, as we also have to avoid hard coded values in the scripts. This is just to give you an idea to use Excel and we will move forward step by step towards proper framework.

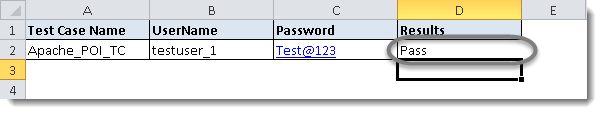
7) Create a ‘New Class‘ and name it as Apache\_POI\_TC by right click on the ‘automationFramework‘ Package and select New > Class. In this we will read the values from the Excel sheet to use them as the test data and write the test result in the Excel.



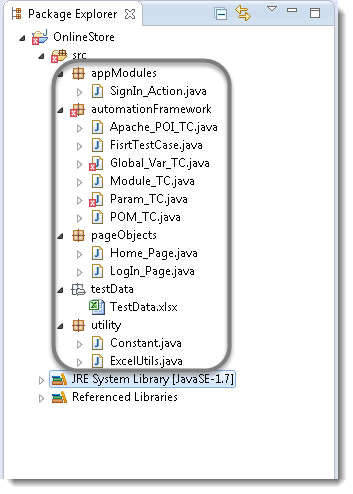
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | package automationFramework;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.firefox.FirefoxDriver;    import pageObjects.\*;    import utility.Constant;    // Import Package utility.\*    import utility.ExcelUtils;    import appModules.SignIn\_Action;    public class Apache\_POI\_TC {    private static WebDriver driver = null;    public static void main(String[] args) throws Exception {            //This is to open the Excel file. Excel path, file name and the sheet name are parameters to this method            ExcelUtils.setExcelFile(Constant.Path\_TestData + Constant.File\_TestData,"Sheet1");            driver = new FirefoxDriver();            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);            driver.get(Constant.URL);            SignIn\_Action.Execute(driver);            System.out.println("Login Successfully, now it is the time to Log Off buddy.");            Home\_Page.lnk\_LogOut(driver).click();            driver.quit();            //This is to send the PASS value to the Excel sheet in the result column.            ExcelUtils.setCellData("Pass", 1, 3);    }    } |

Give it a run, see how beautify your script will execute the code.

8) Once it finished open the Excel file and check for the result.



Your Project explorer window will look like this now.



Log4J Logging

During the running of test case user wants some information to be logged in the console. Information could be any detail depends upon the purpose. Keeping this in mind that we are using Selenium for testing, we need the information which helps the User to understand the test steps or any failure during the test case execution. With the help of Log4j it is possible to enable loggings during the Selenium test case execution for e.g. let’s say you have encountered a failure in automation test script and it has to be reported in the system. The set of information that you have required to report a bug is :

A complete test steps to replicate the scenario

Issue, Description of the failure or reason for the failed test case

Time stamp for the developers to investigate the issue in detail

Log4j helps us to acheive the above objectives in Selenium Webdriver. When logging is wisely used, it can prove to be an essential tool.

Logging inside the Methods

Logging inside the testcase is very tedious task and sooner or later you will find it boring and annoying. Plus everybody has their own way of writing log messages and messages can be less informative and confusing. So why not make it universal. Writing logs message inside the methods is much helpful way, with that we can avoid lots of confusions, save lot of time and maintain consistency.

How to do it…

1) Download JAR files of Log4j and Add Jars to your project library. You can download it from here. That’s all about configuration of Apache POI with eclipse. Now you are ready to write your test.

2) Create a new XML file – log4j.xml and place it under the Project root folder.

3) Paste the following code in the log4j.xml file.

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | <?xml version="1.0" encoding="UTF-8"?>    <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">    <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">    <appender name="fileAppender" class="org.apache.log4j.FileAppender">    <param name="Threshold" value="INFO" />    <param name="File" value="logfile.log"/>    <layout class="org.apache.log4j.PatternLayout">    <param name="ConversionPattern" value="%d %-5p [%c{1}] %m %n" />    </layout>    </appender>    <root>    <level value="INFO"/>    <appender-ref ref="fileAppender"/>    </root>    </log4j:configuration> |

Note: After pasting the code make sure that the code is exactly same, as copying from HTML may change some symbols (“) to (?).

Let’s take an example of our previous Apache\_POI\_TC test case and put log messages in every method and module it interacts with.

4) To achieve that we need to create a static Log class, so that we can access its log method in any of our project class. Log class will look like this:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | package utility;    import org.apache.log4j.Logger;    public class Log {    // Initialize Log4j logs    private static Logger Log = Logger.getLogger(Log.class.getName());//    // This is to print log for the beginning of the test case, as we usually run so many test cases as a test suite    public static void startTestCase(String sTestCaseName){    Log.info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    Log.info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    Log.info("$$$$$$$$$$$$$$$$$$$$$                 "+sTestCaseName+ "       $$$$$$$$$$$$$$$$$$$$$$$$$");    Log.info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    Log.info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");    }    //This is to print log for the ending of the test case    public static void endTestCase(String sTestCaseName){    Log.info("XXXXXXXXXXXXXXXXXXXXXXX             "+"-E---N---D-"+"             XXXXXXXXXXXXXXXXXXXXXX");    Log.info("X");    Log.info("X");    Log.info("X");    Log.info("X");    }    // Need to create these methods, so that they can be called    public static void info(String message) {    Log.info(message);    }    public static void warn(String message) {        Log.warn(message);    }    public static void error(String message) {        Log.error(message);    }    public static void fatal(String message) {        Log.fatal(message);    }    public static void debug(String message) {        Log.debug(message);    }    } |

5) Insert log messages in Home\_Page class of pageObject package.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package pageObjects;    import org.openqa.selenium.By;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.WebElement;                    import framework.utility.Log;    public class Home\_Page {    private static WebElement element = null;    public static WebElement lnk\_MyAccount(WebDriver driver){    element = driver.findElement(By.id("account"));    Log.info("My Account link element found");    return element;    }    public static WebElement lnk\_LogOut(WebDriver driver){    element = driver.findElement(By.id("account\_logout"));    Log.info("Log Out link element found");    return element;    }    } |

6) Insert log messages in LogIn\_Page class of pageObject package.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package pageObjects;    import org.openqa.selenium.\*;    import utility.Log;    public class LogIn\_Page {    static WebElement element = null;            public static WebElement txtbx\_UserName(WebDriver driver){                element = driver.findElement(By.id("log"));    Log.info("Username text box found");                return element;                }            public static WebElement txtbx\_Password(WebDriver driver){                element = driver.findElement(By.id("pwd"));    Log.info("Password text box found");                return element;                }            public static WebElement btn\_LogIn(WebDriver driver){                element = driver.findElement(By.id("login"));    Log.info("Submit button found");                return element;                }            } |

7) Insert log messages in SignIn\_Action class of appModule package.



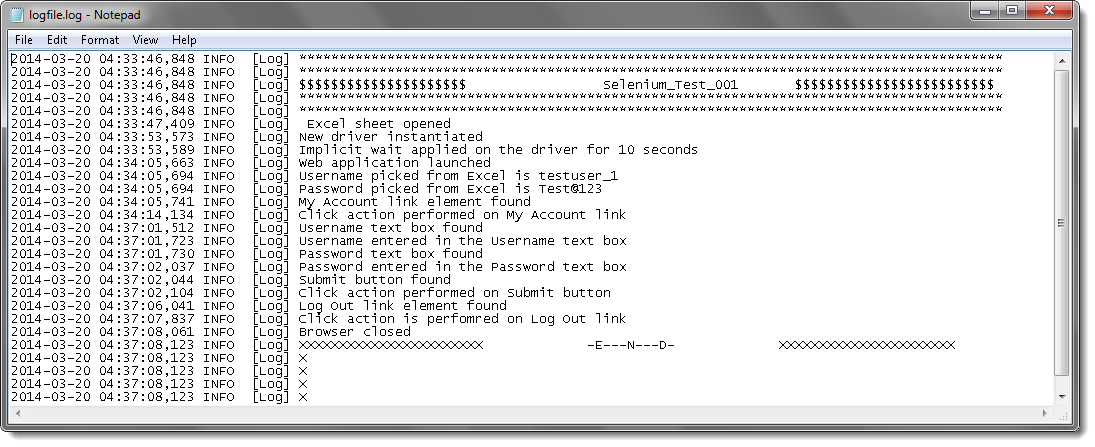
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | package appModules;            import org.openqa.selenium.WebDriver;            import pageObjects.Home\_Page;            import pageObjects.LogIn\_Page;            import utility.ExcelUtils;            import utility.Log;        public class SignIn\_Action {    public static void Execute(WebDriver driver) throws Exception{    String sUserName = ExcelUtils.getCellData(1, 1);    Log.info("Username picked from Excel is "+ sUserName );    String sPassword = ExcelUtils.getCellData(1, 2);    Log.info("Password picked from Excel is "+ sPassword );    Home\_Page.lnk\_MyAccount(driver).click();    Log.info("Click action performed on My Account link");    LogIn\_Page.txtbx\_UserName(driver).sendKeys(sUserName);    Log.info("Username entered in the Username text box");    LogIn\_Page.txtbx\_Password(driver).sendKeys(sPassword);    Log.info("Password entered in the Password text box");    LogIn\_Page.btn\_LogIn(driver).click();    Log.info("Click action performed on Submit button");            }    } |

8) Now it is the time to insert log message in to your test script but before that create a ‘New Class‘ and name it as Log4j\_Logging\_TC by right click on the ‘automationFramework‘ Package and select New > Class.

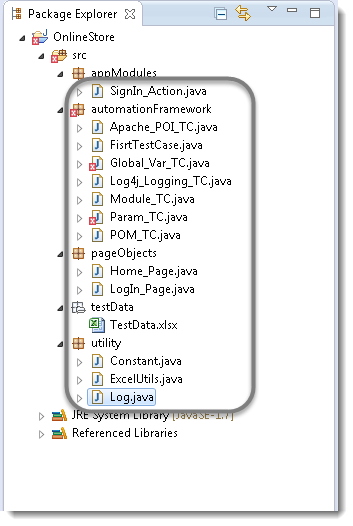


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | package automationFramework;    // Import Package Log4j.\*    import org.apache.log4j.xml.DOMConfigurator;    import java.util.concurrent.TimeUnit;    import org.openqa.selenium.\*;    import pageObjects.\*;    import utility.\*;    import appModules.\*;    public class Log4j\_Logging\_TC {    private static WebDriver driver = null;            public static void main(String[] args) throws Exception {    // Provide Log4j configuration settings    DOMConfigurator.configure("log4j.xml");    Log.startTestCase("Selenium\_Test\_001");    ExcelUtils.setExcelFile(Constant.Path\_TestData + Constant.File\_TestData,"Sheet1");    Log.info(" Excel sheet opened");    driver = new FirefoxDriver();    Log.info("New driver instantiated");    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);    Log.info("Implicit wait applied on the driver for 10 seconds");    driver.get(Constant.URL);    Log.info("Web application launched");    SignIn\_Action.Execute(driver);    System.out.println("Login Successfully, now it is the time to Log Off buddy.");    Home\_Page.lnk\_LogOut(driver).click();    Log.info("Click action is perfomred on Log Out link");    driver.quit();    Log.info("Browser closed");    ExcelUtils.setCellData("Pass", 1, 3);    Log.endTestCase("Selenium\_Test\_001");    }    } |

Once your test is complete, go to your project root folder and open the log file. Your log file will look like this:



Your Project explorer window will look like this now.



# TestNG Reporting

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. In simple words TestNG is a tool that help us to organize the tests and help us to produce the test reports. TestNG framework can be used for automation testing with Selenium (web application automation testing tool).

## TestNG Advantages

* – Multiple built in Annotations which are easier to use and understand
* – Test method can be dependent to other method
* – Test cases can be Grouped and can be execute separately by groups
* – Parallel testing is possible
* – TestNG has built in HTML report and XML report generation facility. It has also built in  logging facility

## How to use it…

1) First step is to **Install TestNG**. It is easy to install TestNG, as it comes as a plugin for Eclipse IDE.

2) Create a ‘***New Class***‘ by right click on the ‘**automationFramework**‘ package then select **TestNG** > **Create a TestNG Class** and name it as **TestNG\_Framework** .

3) Let’s take an example of our previous **Log4j\_Logging\_TC** test case and divide the test case in to three parts .

**@BeforeMethod** : Configure Log4j xml, Open Excel sheet, Launch Firefox and direct it to the Base URL

**@Test** : Execute SIgnIn action and Log out

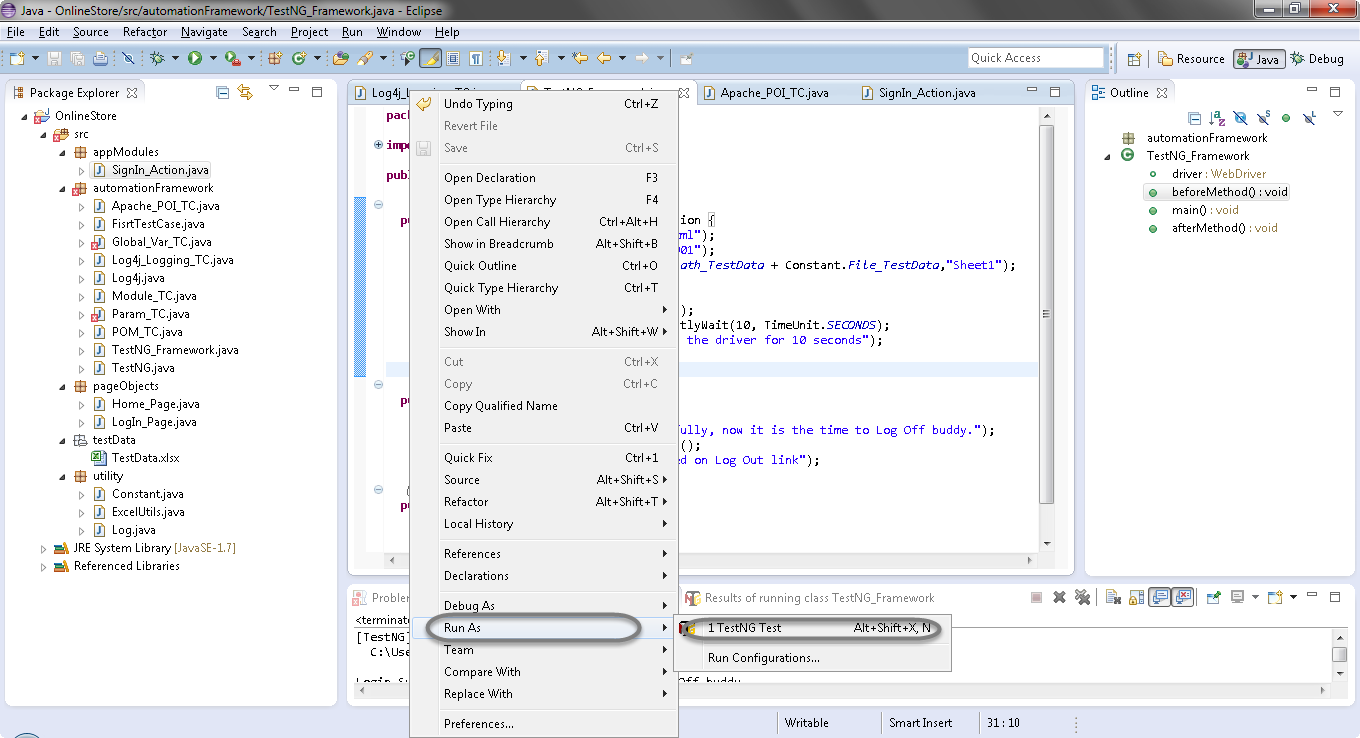
**@AfterMethod** : Close Firefox browser



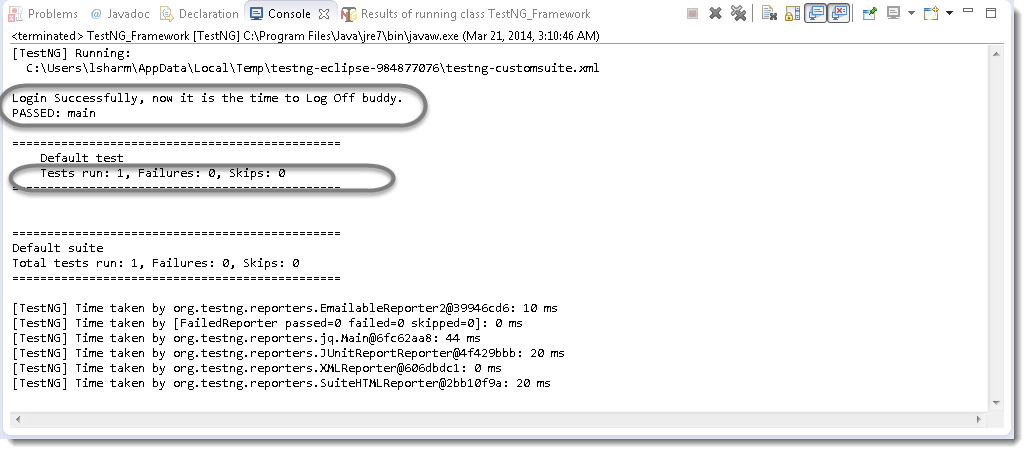
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | package automationFramework;    import java.util.concurrent.TimeUnit;    import org.apache.log4j.xml.DOMConfigurator;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.testng.annotations.Test;    import org.testng.annotations.BeforeMethod;    import org.testng.annotations.AfterMethod;    import pageObjects.Home\_Page;    import appModules.SignIn\_Action;    import utility.Constant;    import utility.ExcelUtils;    import utility.Log;    public class TestNG\_Framework {    public WebDriver driver;    @BeforeMethod      public void beforeMethod() throws Exception {       DOMConfigurator.configure("log4j.xml");    Log.startTestCase("Selenium\_Test\_001");    ExcelUtils.setExcelFile(Constant.Path\_TestData + Constant.File\_TestData,"Sheet1");    Log.info(" Excel sheet opened");    driver = new FirefoxDriver();    Log.info("New driver instantiated");    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);    Log.info("Implicit wait applied on the driver for 10 seconds");    driver.get(Constant.URL);            }    @Test      public void main() throws Exception {        SignIn\_Action.Execute(driver);    System.out.println("Login Successfully, now it is the time to Log Off buddy.");    Home\_Page.lnk\_LogOut(driver).click();    Log.info("Click action is perfomred on Log Out link");            }       @AfterMethod      public void afterMethod() {        driver.quit();            }    } |

**Note:** Method **@test** has been renamed as **main()**. By default is **f()**.

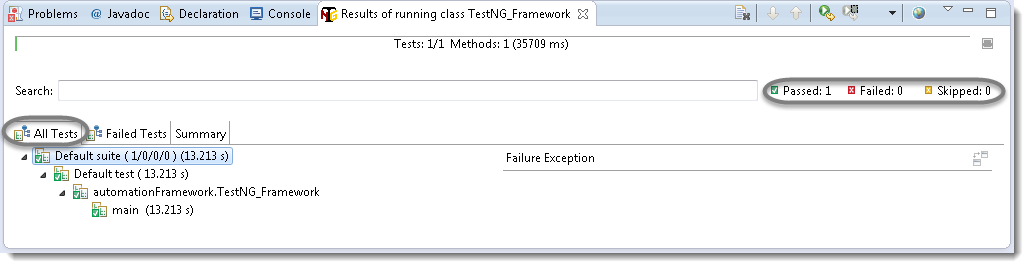
3) Run the test by right click on the test case script and select **Run As** > **TestNG Test**.



4) Give it few minutes to complete the execution, once it is finished the results will look like this in the **Console** window.

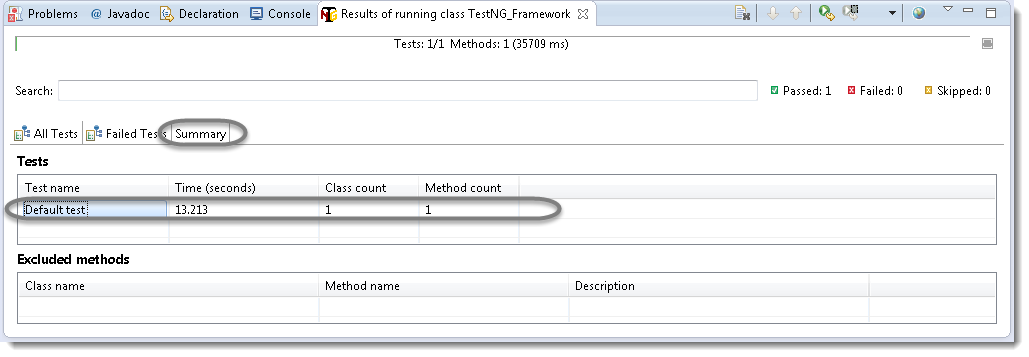


5) Click on the **Results of TestNG** tab. It will display the total passed, failed and skipped test with time taken during the execution.

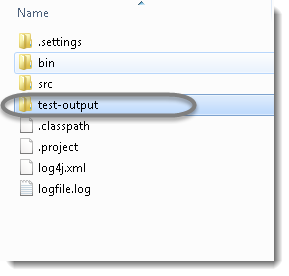


It displayed ‘passed : 1’. This means test is successful and  Passed.

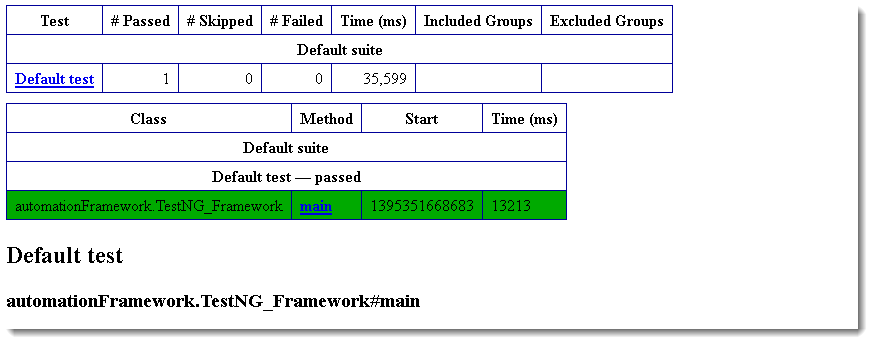
There are 3 sub tabs. “All Tests”, “Failed Tests” and “Summary”. Just click “All Tests” to see what is there.



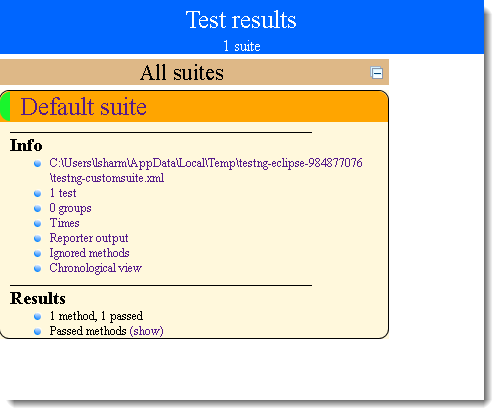
6) TestNG also produce HTML reports. To access those reports go to your **Project** folder and open **test-output** folder.



7) Open ‘**emailable-report.html**‘, as this is a html report open it with browser.



8) TestNG also produce ‘**index.html**‘ report and it resides in the same **test-output** folder. This reports gives the link to all the different component of the TestNG reports like **Groups** & **Reporter Output**. On clicking these will display detailed descriptions of execution. In the advance chapter of TestNG we will go though each of the TestNG topics.



User Defined Functions

User defined functions (UDF) are the functions which are created by users as per their own requirement. In order to use a UDF, it is necessary to call the appropriate package file at the beginning of the program. The header file informs the program of the name, type, and number and type of arguments, of all of the functions contained.

Java



|  |  |
| --- | --- |
| 1 | import<packageName> |

An UDF function is accessed by simply writing the function name, followed by a list of arguments, which represent the information being passed to the function. The arguments must be enclosed in parentheses, and separated by commas: they can be constants, variables, or more complex expressions.

Automation Best Practices: Avoid logics when writing automation test scripts. Every logic is to be maintained in the function libraries and only be called with their name in the test scripts. Every arthematic calculation, date calculation, string manipulation etc. should be avoided in the Test scripts rather put them in to the functions and use them.

Benefits of User Defined Functions

– Can be used in a number of places without rewriting the code.

– Code can be made less complex and easier to write.

– Parameters can be passed to the function.

– Simpler to invoke.

For example:  It is a three steps process to open a URL. First Instantiate a New driver, second Apply an Implicit wait on the driver and third Navigate to URL.  Browser can be any browser; it can be Mozilla, IE or any. It makes sense to create a function for opening a browser which will accept an argument (Browser Type) and it will open that particular browser. This ‘Browser Type’ argument will be driven from the Test Data sheet. To achieve this few more functions are required.

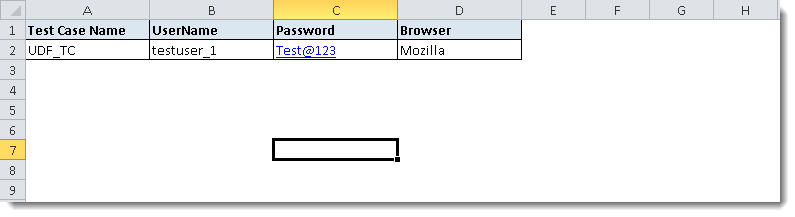
Function One: openBrowser(int iTestCaseRow), it will return a WebDriver

Function Two: getTestCaseName(String sTestCase), it will return refined Test case name

Function Three: getRowContains(String sTestCaseName, int colNum), it will return the row number of the Test case name from the test data sheet.

How to use it…

1) Create a new column (Browser) in the Test Data sheet.



Make some entries in the Constant class for the column numbers:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package utility;    public class Constant {        public static final String URL = "http://www.store.demoqa.com";    public static final String Path\_TestData = "D://ToolsQA//OnlineStore//src//testData//";    public static final String File\_TestData = "TestData.xlsx";    //Test Data Sheet Columns    public static final int Col\_TestCaseName = 0;    public static final int Col\_UserName =1 ;    public static final int Col\_Password = 2;    public static final int Col\_Browser = 3;    } |

2) Create a ‘New Class‘ by right click on the ‘utility‘ package then select New > Class and name it as Utils. Now create a Static Method for Initiate Browser in the ‘Utils’ class. This method will have an Argument (TestCase Row) and a Return value (WebDriver).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | public class Utils {            public static WebDriver driver = null;        public static WebDriver openBrowser(int iTestCaseRow) throws Exception{            String sBrowserName;            try{            sBrowserName = ExcelUtils.getCellData(iTestCaseRow, Constant.Col\_Browser);            if(sBrowserName.equals("Mozilla")){                driver = new FirefoxDriver();                Log.info("New driver instantiated");                driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);                Log.info("Implicit wait applied on the driver for 10 seconds");                driver.get(Constant.URL);                Log.info("Web application launched successfully");                }            }catch (Exception e){                Log.error("Class Utils | Method OpenBrowser | Exception desc : "+e.getMessage());            }            return driver;        }    } |

3) To get the Test Case row from the Test data sheet, it is required to get Test Case name, so that it can be searched in the Test Data sheet. Write a function in ‘Utils‘ class to get the Test Case name. Test case name can be easily get by using “this.toString()“. This function will return the package name and the class name for e.g. ‘automationFramework.UDF\_TC@2550036c’. Another function is required to refine the long test case name in to UDF\_TC.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public static String getTestCaseName(String sTestCase)throws Exception{            String value = sTestCase;            try{                int posi = value.indexOf("@");                value = value.substring(0, posi);                posi = value.lastIndexOf(".");                value = value.substring(posi + 1);                return value;                    }catch (Exception e){                Log.error("Class Utils | Method getTestCaseName | Exception desc : "+e.getMessage());                throw (e);            } |

4) Once Test Case name is captured, it can be used as an Argument for a function which will return the Test case row from the Excel sheet.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static int getRowContains(String sTestCaseName, int colNum) throws Exception{  int i;      try {      int rowCount = ExcelWSheet.getLastRowNum();          for ( i=0 ; i<rowCount; i++){          if  (ExcelUtils.getCellData(i,colNum).equalsIgnoreCase(sTestCaseName)){              break;             }          }         return i;      }catch (Exception e){      Log.error("Class ExcelUtil | Method getRowContains | Exception desc : " + e.getMessage());         throw(e);      }      } |

5) Create a ‘New Class‘ by right click on the ‘automationFramework‘ package then select TestNG > Create a TestNG Class and name it as UDF\_TC.

Note: Take previous executed test case ‘TestNG\_Framework‘ and modify its Before Method only. The new test script will look like this:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | package automationFramework;    import org.apache.log4j.xml.DOMConfigurator;    import org.openqa.selenium.WebDriver;    import org.testng.annotations.\*;    import pageObjects.\*;    import utility.\*;    import appModules.\*;    public class UDF\_TC {        public WebDriver driver;        private String sTestCaseName;        private int iTestCaseRow;        @BeforeMethod      public void beforeMethod() throws Exception {             DOMConfigurator.configure("log4j.xml");             sTestCaseName = this.toString();             sTestCaseName = Utils.getTestCaseName(this.toString());            Log.startTestCase(sTestCaseName);            ExcelUtils.setExcelFile(Constant.Path\_TestData + Constant.File\_TestData,"Sheet1");            iTestCaseRow = ExcelUtils.getRowContains(sTestCaseName,Constant.Col\_TestCaseName);            driver = Utils.openBrowser(iTestCaseRow);            }        @Test      public void main() throws Exception {            SignIn\_Action.Execute(driver);            System.out.println("Login Successfully, now it is the time to Log Off buddy.");            Home\_Page.lnk\_LogOut(driver).click();            Log.info("Click action is performed on Log Out link");            }       @AfterMethod      public void afterMethod() {            driver.quit();            }    } |

Isn’t it easy to call functions rather than writing them again and again and increase code complexity.

# Selenium WebDriver Object Repository

Many of us are migrated from **QTP** to **Selenium** and trust me when it comes to **Object Repository**, QTP is far better than Selenium WebDriver, who in this world do not want the flexibility of **Intellisence** of QTP. Isn’t great that you just press the dot after typing the browser name and it gives you all the pages under that parent browser. Then you again press dot after the page name and it gives you all the elements of that page.

But still Selenium is a Freeware tool and we should be **very thankful** to the team behind it for their efforts.

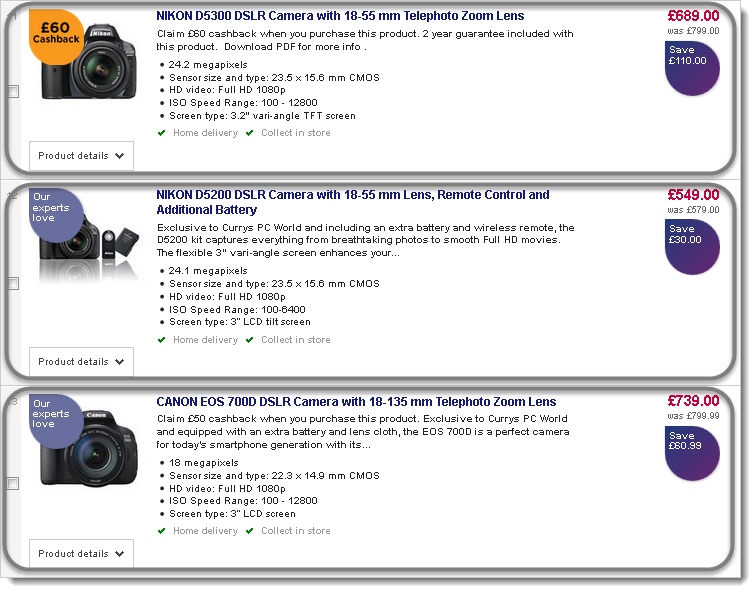
In this tutorial I will explain that how one can have better Object Repository in Selenium like we have in QTP and how one can take the advantage of Intellisence in Selenium.

Before moving on to this I would suggest you to go through the **Page Object Model** first.

## ****Object Repository****

Think of a scenario where you have a Webpage and it has multiple sections, multiple frames and hundreds of WebElements. Obviously you do not want that once you type the page name and it will give you all the elements available on the webpage. If there are few elements, it is fine to have the same structure but if there are many then it is advisable to divide your page in to different sections for e.g. header, footer, left navigation, center content and right navigation. Then categories each WebElement under their parent element.

Think of another scenario where you have a Product listing page and it has many products available on that page. Each product has its product price, sales price, info, title, image, add to cart button and rating. What would be the better approach to it, what would you do when you have multiple same elements on the page like multiple add to cart button, sale price etc. Would you like to name them like price\_1, price\_2 and price\_3?



What if I give you an approach which gives you the same behavior we have in QTP. Once you type the page name, it will give you different parent elements only and on selecting parent element, it will give you linked child elements only.

**How to do it…**

1)  Create a ‘***New Class***‘ by right click on the ‘**pageObjects**‘ package then select **New** > **Class**and name it as **ProductListing\_Page**.



|  |  |
| --- | --- |
| 1  2  3 | public class ProductListing\_Page {    } |

2) Create another public static class inside the above class ‘**pageObjects**‘ and name it as **Product\_1**.



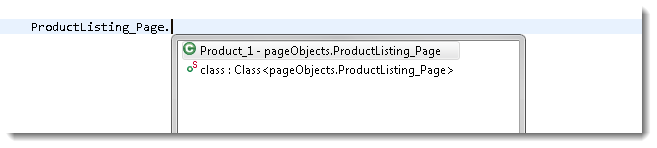
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class ProductListing\_Page {        public static class Product\_1{         }    } |

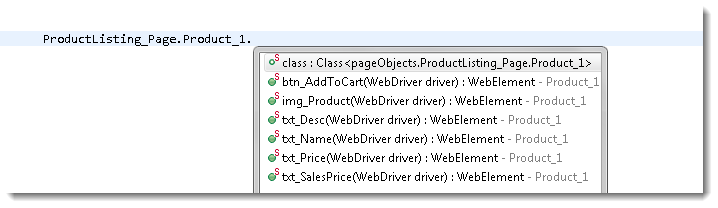
3) Now create different **Static Methods** for each child element of Product \_1.  These methods will have an**Argument** (WebDriver) and a **Return** value (WebElement).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | package pageObjects;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.WebElement;    public class ProductListing\_Page {    public static class Product\_1{    public static WebElement txt\_Price(WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    public static WebElement txt\_SalesPrice(WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    public static WebElement img\_Product(WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    public static WebElement txt\_Name(WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    public static WebElement txt\_Desc(WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    public static WebElement btn\_AddToCart( WebDriver driver){    WebElement element = null;    // Write Code to find element here    return element;    }    }    } |

You are done with creating Intellisence structure for your Selenium Object Repository. Now let’s taste the benefits of it.

4) Just type ProductListing\_Page in your test script and press dot. It will display all the products you have specified in your class. 

5) Select Product\_1 and again press dot, it will now display all the child elements associated with the parent Product\_1.

## What is an Exception

An **Exception**is an event, which occurs during the execution of a program, that disrupts the normal flow of the program’s instructions or in simple words, any issue which makes your test case stop in between the execution is an Exception.

## Difference between Error and Exception

An **Error** “indicates serious problems that a reasonable application should not try to catch.”

An **Exception** “indicates conditions that a reasonable application might want to catch.”

Whenever an error occurs while executing a statement, creates an exception object and then the normal flow of the program halts and it tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as **method hierarchy**, **line number** where the exception occurred, **type of exception** etc. When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called “**throwing the exception”**.

## What is Exception Handling

**Exception handling** refers to the anticipation, detection, and resolution of programming application, and communications errors. It is the block of code that processes the exception object and helps us by giving us a chance to act on it.

## Why Exception Handling is Important

1) Think of a situation where you have got an exception and you want to print some custom message in your  logs, so that it can be understandable by the whole team.

2) There can be some situations where you want to just eat up the exception and want your test to carry on with rest of the execution.

3) In case you want to perform some series of steps on occurring of an certain exception for e.g. if you got an exception because a product is out of stock, that product is no longer displayed on the page and you want you to go with another product available on the page.

4) In case you want to handle some certain kind of exception in Selenium like ElementNotSelectableException, ElementNotVisibleException, NoSuchElementException etc. exceptions.

## Different Exceptions in Selenium

There is a complete list of Exceptions mentioned on the Selenium Doc which you may or may not encounter in course of your  testing.

**Most common Exceptions:**

1) NoSuchElementException : FindBy method can’t find the element.

2) StaleElementReferenceException : This tells that element is no longer appearing on the DOM page.

3) TimeoutException: This tells that the execution is failed because the command did not complete in enough time.

4) ElementNotVisibleException: Thrown to indicate that although an element is present on the DOM, it is not visible, and so is not able to be interacted with

5) ElementNotSelectableException: Thrown to indicate that may be the element is disabled, and so is not able to select.

You can access the complete list of exception here, but if you are a beginner that I would suggest you to move forward as of now. It can be  over dose.

## How to Handle Exception

**Try/Catch:**A method catches an exception using a combination of the try and catch keywords. **Try** is the start of the block and **Catch** is at the end of try block to handle the exceptions. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as **protected code**, and the syntax for using try/catch looks like the following:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | try    {    // Some code    }catch(Exception e){    // Code for Handling the exception    } |

**Multiple Catch blocks:**A try block can be followed by **multiple catch** blocks. Like I said earlier, that there are multiple exceptions and you can expect more than one type of exception on a single code block and if you like to handle each type of exception separately with a separate block of code. The syntax for multiple catch blocks looks like the following:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | try    {       //Some code    }catch(ExceptionType1 e1){       //Code for Handling the Exception 1    }catch(ExceptionType2 e2){       //Code for Handling the Exception 2    } |

There is no restriction on the number of the catch blocks, you can use more than two. You may be wondering that how does it work. It is pretty simple, if an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the exception thrown matches the ExceptionType1, it gets caught there and it executes the code which is under the same exception block. If not, the exception passes down to the second catch statement and goes on like this.

**Note:**In case the exception does not match with any exception type and falls through all catches, the current method stops execution and exception is thrown. That is why it is advisable to include default exception as well in the end, so in case if the exception falls through, it can be handled by the default one.

**Throw:**Sometimes we want to generate exception explicitly in our code, for example in Selenium Automation Framework most of the time we print self-written logs, once we catch an exception and then we need to throw that exception back to the system so that the test case can be terminated. **Throw** keyword is used to throw exception to the runtime to handle it.

**Throws:**When we are throwing any exception in a method and not handling it, then we need to use **throws**keyword in **method signature** to let caller program know the exceptions that might be thrown by the method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // Method Signatur\    public static void anyFunction() throws Exception{        try{    // write your code here    }catch (Exception e){    // Do whatever you wish to do here    // Now throw the exception back to the system            throw(e);            }        } |

**Multiple Exceptions:**We can provide multiple exceptions in the throws clause.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static void anyFunction() throws ExceptionType1, ExceptionType2{    try {    //Some code    }catch(ExceptionType1 e1){    //Code for Handling the Exception 1    }catch(ExceptionType2 e2){    //Code for Handling the Exception 2    } |

**Finally:** The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | try    {       //Protected code    }catch(ExceptionType1 e1)    {       //Catch block    }catch(ExceptionType2 e2)    {       //Catch block    }catch(ExceptionType3 e3)    {       //Catch block    }finally    {       //The finally block always executes.    } |

# Exception Handling in Selenium

Your Selenium test should be able to fail, but not because of exceptions that are thrown. If your test is failing from exceptions then quite likely you have no exception handling. By doing this, you don’t have the opportunity to cleanup the WebDriver object at the end of the test.

The tests should be failing under your terms only for example, you should never be getting exceptions like NullPointerException but if you are getting such as ElementNotFoundException, then also it is good idea to catch the exception, stop the further execution and end your test in a Logical way.

**Example 1:** I do not use any Page Object Factory but I use my own Page Object Pattern and I always print error logs and take screenshot on any exception I encounter. Please look at the code below:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public static WebElement btn\_ReportCategory(WebDriver driver) throws Exception{            try{                WebElement element = driver.findElement(By.linkText("+ Report Categories"));            }catch (Exception e){    // Printing logs for my report                Log.error("Report Category button element is not found.");    // Taking screenshot for defect reporting    Utils.captureScreenShot();    // After doing my work, now i want to stop my test case                throw(e);            }    // This will return the Element in case of no Exception            return element;        } |

**Example 2:**TimeoutException using Selenium WebDriver.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | try{        myTestDriver.findElement(By.xpath("//\*[@id='register']")).click();    }catch (TimeoutException toe) {    wait.until( ExpectedConditions.elementToBeClickable(By.xpath("//\*[@id='register']")));    myTestDriver.findElement(By.xpath("//\*[@id='register']")).click();    }catch (Exception e) {    Log.error("Register element is not found.");    throw(e);        }    } |

**Example 3:** Let’s assume that in Selenium WebDriver you want to verify the presence of any element on the page. You would not be able to get this with element locator because if the element is present, your locator will work and you will easily be able to print that the element is present but in case your element is not present on the page, your locator will fail and simply throw the exception. This case would be easily handled with the self-written function.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static boolean verifyObjectPresent(WebDriver driver) {        try {         driver.findElement(By.linkText("+ Report Categories"));         return true;        } catch (Exception e) {         return false;        }    } |