**Using TestNG listeners to close browser when Tests Failed:**

We will be using ITestListener. Create a class Listeners implementing ITestListener under executionEngine package. Add all the methods that need to be implemented.

**Listeners class:**

**package** executionEngine;

**import** org.testng.ITestContext;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**public** **class** Listeners **implements** ITestListener{

@Override

**public** **void** onTestStart(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestSuccess(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestFailure(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestSkipped(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestFailedButWithinSuccessPercentage(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onStart(ITestContext context) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onFinish(ITestContext context) {

// **TODO** Auto-generated method stub

}

}

Change the TestNG xml to point our tests to this listener. We do this using listeners tags.

TestNG xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite"*>

<listeners>

<listener class-name=*"executionEngine.Listeners"*/>

</listeners>

<test thread-count=*"5"* name=*"Test"*>

<classes>

<class name=*"executionEngine.DriverScript"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

We have added the listeners tag in the xml.

Now let’s implement onTestFailure() method to close the browser when a test is failed. We need to have the driver to close the browser so that we can call the driver.quit() method to close it. But how to pass the driver. For this we are creating a separate class which contains a map.

**driverHolding.java under executionEngine package:**

**package** executionEngine;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.openqa.selenium.WebDriver;

**public** **class** driverHolding {

**public** **static** Map<String, WebDriver> *mapdriver* = **new** HashMap<String, WebDriver>();

}

This class has a map which holds a string as a key and driver we have created. But this string should be unique for each test case. Because we will be running these test cases parallelly so the string should be unique. One such unique string is the test case like TC01, TC02 etc. We are passing the test case as a data provider parameter to prepareKeywords() method. But the driver we are creating in executeTC() method. So The methods until exturetC are modified to pass this test case to executeTC() method.

The methods that are modified are:

*getDependencies*(keywords, testcase)

*executeTC*(originalkeywords,actionclass,keywordvsac,objectrepository,keywordvsor, testcase);

Now in executeTC() method after creating the driver, we will add the test case and driver to the map we have created earlier. We will be adding it like testcase+”driver”, so each test will have it’s own key and value.

**public** **static** **void** executeTC(List<String> originalkeywords, Set<String> actionclass, Map<String, String> keywordvsac, Set<String> objectrepository, Map<String, String> keywordvsor, String testcase) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException, InstantiationException, IOException {

System.*setProperty*("webdriver.gecko.driver", "C:\\BrowserDrivers\\geckodriver.exe");

WebDriver driver = **new** FirefoxDriver();

**executionEngine.driverHolding.*mapdriver*.put(testcase+"driver", driver);**

Properties gldata = **new** Properties();

InputStream input = **new** FileInputStream("src/executionEngine/config.properties");

gldata.load(input);

**for**(String str : originalkeywords) {

String keyword = str;

System.***out***.println(keyword);

String[] specialkeywords = gldata.getProperty("SPECIAL\_KEYWORDS").split("\\,");

**boolean** skstatus = **false**;

**for**(**int** i=0; i<specialkeywords.length; i++) {

**if**(keyword.indexOf(specialkeywords[i]) != -1) {

skstatus = **true**;

**break**;

}

}

String[] checkkeywords = gldata.getProperty("CHECK\_KEYWORDS").split("\\,");

**boolean** ckstatus = **false**;

**for**(**int** i=0; i<checkkeywords.length; i++) {

**if**(keyword.indexOf(checkkeywords[i]) != -1) {

ckstatus = **true**;

**break**;

}

}

String[] bookingvehicles = gldata.getProperty("BOOKING\_VEHICLES").split("\\,");

**boolean** bkstatus = **false**;

**for**(**int** i=0; i<bookingvehicles.length; i++) {

**if**(keyword.indexOf(bookingvehicles[i]) != -1) {

bkstatus = **true**;

**break**;

}

}

**if**(skstatus) {

**if**(keyword.indexOf("(") != -1){

String[] parts = keyword.split("\\(");

String[] dataelements = (parts[1].split("\\)"))[0].split("\\,");

*specialfunction*(parts[0],dataelements,driver);

}

**else** {

String parts = keyword;

String[] dataelements = **null**;

*specialfunction*(parts,dataelements,driver);

}

}

**else** **if**(ckstatus) {

**if**(keyword.indexOf("(") != -1){

String[] parts = keyword.split("\\(");

String checkkeyword = parts[0];

**int** expectedvalue = Integer.*parseInt*((parts[1].split("\\)")[0]));

//System.out.println("check keyword: "+checkkeyword);

//System.out.println("expected value: "+expectedvalue);

*specialcheckfunction*(checkkeyword,expectedvalue,driver);

}

}

**else** **if**(bkstatus) {

**if**(keyword.indexOf("(") != -1){

String[] parts = keyword.split("\\(");

String bookingkeyword = parts[0];

**int** vehno = Integer.*parseInt*((parts[1].split("\\)")[0]));

//System.out.println("check keyword: "+checkkeyword);

//System.out.println("expected value: "+expectedvalue);

*bookfunction*(bookingkeyword,vehno,driver);

}

}

**else** {

String actioncl = keywordvsac.get(keyword);

String objectcl = keywordvsor.get(keyword);

System.***out***.println(actioncl);

System.***out***.println(objectcl);

Class<?> cls = Class.*forName*("actions."+actioncl);

Class<?> orc = Class.*forName*("objectrepository."+objectcl);

Method[] methodcall = cls.getDeclaredMethods();

**for**(Method m : methodcall) {

**if**(keyword.equalsIgnoreCase(m.getName()) && m.getParameterCount() == 0)

{

Method mc = cls.getDeclaredMethod(keyword);

Constructor<?> constructor = cls.getConstructor(WebDriver.**class**);

mc.invoke(constructor.newInstance(driver));

}

**else** **if**(keyword.equalsIgnoreCase(m.getName()) && m.getParameterCount() == 1)

{

Method morc = orc.getDeclaredMethod(keyword);

Constructor<?> orconstructor = orc.getConstructor(WebDriver.**class**);

WebElement we = (WebElement) morc.invoke(orconstructor.newInstance(driver));

Method mc = cls.getDeclaredMethod(keyword,WebElement.**class**);

Constructor<?> constructor = cls.getConstructor(WebDriver.**class**);

mc.invoke(constructor.newInstance(driver),we);

}

}

}

}

}

Now let’s change the onTestFailure() method of listener. Here, we get the testcase first by calling the getParameters() method.

Then we get the driver from the map by using testcase+”driver”. Then we quit the driver. We are using quit method here as we want to close all the instances. Because sometimes we get the alerts like “the subscriber already exists” and if we use driver.close(), only the alert will be closed and not the browser. So we are using quit method. Once the browser is closed, since the test case is completed, we remove the driver object from the map.

@Override

**public** **void** onTestFailure(ITestResult result) {

String testcase = (String)(result.getParameters())[0];

WebDriver driver = driverHolding.*mapdriver*.get(testcase+"driver");

driver.quit();

driverHolding.*mapdriver*.remove(testcase+"driver");

System.***out***.println(testcase + " failed.");

}

Since I want to remove the driver from the map when the test case is passed also, we have to implement the onTestSuccess() method as well.

@Override

**public** **void** onTestSuccess(ITestResult result) {

String testcase = (String)(result.getParameters())[0];

driverHolding.*mapdriver*.remove(testcase+"driver");

System.***out***.println(testcase + " passed.");

}

The following is the full program for Listeners.

**package** executionEngine;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.remote.RemoteWebDriver;

**import** org.openqa.selenium.remote.SessionId;

**import** org.testng.ITestContext;

**import** org.testng.ITestListener;

**import** org.testng.ITestResult;

**public** **class** Listeners **implements** ITestListener{

@Override

**public** **void** onTestStart(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestSuccess(ITestResult result) {

String testcase = (String)(result.getParameters())[0];

driverHolding.*mapdriver*.remove(testcase+"driver");

System.***out***.println(testcase + " passed.");

}

@Override

**public** **void** onTestFailure(ITestResult result) {

String testcase = (String)(result.getParameters())[0];

WebDriver driver = driverHolding.*mapdriver*.get(testcase+"driver");

driver.quit();

driverHolding.*mapdriver*.remove(testcase+"driver");

System.***out***.println(testcase + " failed.");

}

@Override

**public** **void** onTestSkipped(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onTestFailedButWithinSuccessPercentage(ITestResult result) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onStart(ITestContext context) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onFinish(ITestContext context) {

// **TODO** Auto-generated method stub

}

}