**Advance Report [TestNg]**

Add this code to the TestNg index file after parameter tag

<listeners>

<listener class-name=*"com.utility.TestListener"* />

<listener class-name=*"com.utility.CustomReporterTestFailureDetails"* />

<listener class-name=*"com.utility.CustomReporterTestpassDetails"* />

<listener class-name=*"com.utility.CustomReporterTestskipDetails"* />

<listener class-name=*"com.utility.CustomReporterBuildSummary"* />

</listeners>

**These are the classes to be used to generate the advanced report**

1. Class name : CustomReporter

**package** com.utility;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Scanner;

**import** java.util.Set;

**import** org.testng.IInvokedMethod;

**import** org.testng.IResultMap;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestClass;

**import** org.testng.ITestContext;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.collections.Lists;

**import** org.testng.internal.Utils;

**import** org.testng.log4testng.Logger;

**import** org.testng.xml.XmlSuite;

**import** com.init.SeleniumInit;

**public** **class** CustomReporter **extends** CustomReporterListener {

**private** **static** **final** Logger ***L*** = Logger

.*getLogger*(CustomReporterListener.**class**);

// ~ Instance fields ------------------------------------------------------

**private** PrintWriter m\_out;

**private** **int** m\_row;

**private** Integer m\_testIndex;

**private** **int** m\_methodIndex;

**private** Scanner scanner;

**int** passCount=0;

**private** **static** HashMap<String, String> *map* = **new** HashMap<String, String>();

**int** namecount = 0;

**int** qty\_tests = 0;

**int** passed = 0;

**int** skipped = 0;

**int** failedcount = 0;

**int** total\_a = 0;

**int** qty\_pass= 0;

// ~ Methods --------------------------------------------------------------

/\*

\* public static void maperrors() {

\*

\* System.out.println(

\* "In Error mapping..............\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3465768487488............"

\* ); map.put(

\* "org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure. "

\* , "Server or Node is not running.");

\* map.put("Timed out after 20 seconds waiting for visibility of Proxy element"

\* ,

\* "Particular Element is not found on page after 20 second waiting can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("Unable to locate the element",

\* "Particular Element is not located on page can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("no such element",

\* "element could not be found. you may want to check : 1.Check your xpath in automation code. 2.Element may not present on the screen."

\* ); map.put("Unable to bind to locking port 7054 within 45000 ms",

\* "Port is already locked by other browser and may not be free.");

\* map.put("Unexpected error launching Internet Explorer",

\* "Unable to launch IE.");

\* map.put("Unable to find element on closed window",

\* "Browser window may closed unexpectedly.");

\* map.put("Error communicating with the remote browser.",

\* "remote browser may have died.");

\* map.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

\* "xpath of the particular element getting changed OR Page is not loaded completely."

\* ); map.put(

\* "Error forwarding the new session Error forwarding the request Connect to"

\* , "Connection may be refused by the node/server."); }

\*/

**public** CustomReporter() {

**super**();

*map*.put("Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.",

"Server or Node is not running.");

*map*.put("Timed out after 35 seconds waiting for visibility of Proxy element",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

*map*.put("Unable to locate",

"Particular Element is not located on page. Either Page is not loaded completely OR Element is Not found on page");

*map*.put("no such element",

"element could not be found. Check Possibility: 1.Update automation code. 2.Element may not present on the screen. 3.Possibility for \"BUG\"");

*map*.put("Unable to bind to locking port 7054 within 45000 ms",

"Port is already locked by other browser and may not be free.Please restart selenium node and server");

*map*.put("Unexpected error launching Internet Explorer",

"Unable to launch IE.");

*map*.put("Unable to find element on closed window",

"Browser window may closed unexpectedly. This will fix automatically on next run");

*map*.put("Error communicating with the remote browser.",

"remote browser may have died. Please restart selenium node and server");

*map*.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

"xpath of the particular element getting changed OR Page is not loaded completely.");

*map*.put("Error forwarding the new session Error forwarding the request Connect to",

"Connection may be refused by the node/server. Please restart selenium node and server");

*map*.put("element not visible",

"Element is not found on page : 1.Update automation code. 2.Element may not present on the screen.");

*map*.put("Timed out after 35 seconds waiting for visibility of [[AppiumDriver:",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

}

/\*\* Creates summary of the run \*/

@Override

**public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites,

String outdir) { //1

**try** {

m\_out = createWriter(outdir); //2

} **catch** (IOException e) {

***L***.error("output file", e);

**return**;

}

startHtml(m\_out); //3

generateSuiteSummaryReport(suites);

TotalTime(suites); //4

generateMethodSummaryReport(suites); //5

// generateMethodDetailReport(suites);

//endHtml(m\_out); //previous

m\_out.flush();

m\_out.close();

}

String Time;

**public** String TotalTime(List<ISuite> suites) { //4.1

**long** time\_start = Long.***MAX\_VALUE***;

**long** time\_end = Long.***MIN\_VALUE***;

ITestContext overview = **null**;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> itests = suite.getResults();

**for** (ISuiteResult r : itests.values()) {

overview = r.getTestContext();

time\_start = Math.*min*(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.*max*(overview.getEndDate().getTime(), time\_end);

}

}

// m\_out.println("</tr><td class=\"numi\"><center>"+((time\_end -

// time\_start) / 1000.) / 60.+"</center></td> </tr>");

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

Time = String.*valueOf*(formatter

.format(((time\_end - time\_start) / 1000.) / 60.));

**return** Time;

}

**protected** PrintWriter createWriter(String outdir) **throws** IOException { //2.1

// java.util.Date now = new Date();

**new** File(outdir).mkdirs();

**return** **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(**new** File(

outdir, "emailable-report-new" + ".html"))));

}

/\*\*

\* Creates a table showing the highlights of each test method with links to

\* the method details

\*/

**protected** **void** generateMethodSummaryReport(List<ISuite> suites) { //5.1

m\_methodIndex = 0;

startResultSummaryTable("methodOverview"); //5.2

**int** testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() > 1) {

titleRow(suite.getName(), 5);

}

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

String testName = testContext.getName();

m\_testIndex = testIndex;

//resultSummary\_passed(suite, testContext.getPassedTests()); //5.3

System.***out***.println("Passed---");

resultSummary\_passed(suite, testContext.getPassedTests(), testName,

"passed", "");

System.***out***.println("Failed---");

resultSummary(suite, testContext.getFailedConfigurations(), //5.4

testName, "failed", " (configuration methods)");

resultSummary(suite, testContext.getFailedTests(), testName,

"failed", "");

System.***out***.println("Skipped---");

resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"skipped", "");

/\*

\* resultSummary(suite, testContext.getSkippedConfigurations(),

\* testName, "skipped", " (configuration methods)");

\* resultSummary(suite, testContext.getSkippedTests(), testName,

\* "skipped", ""); resultSummary(suite,

\* testContext.getPassedTests(), testName, "passed", "");

\*/

testIndex++;

}

}

endHtml(m\_out);

testCaseNo();

m\_out.println("</table>");

}

/\*\* Creates a section showing known results for each method \*/

**protected** **void** generateMethodDetailReport(List<ISuite> suites) {

m\_methodIndex = 0;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

**if** (r.values().size() > 0) {

m\_out.println("<h1>" + testContext.getName() + "</h1>");

}

resultDetail(testContext.getFailedConfigurations());

resultDetail(testContext.getFailedTests());

/\*

\* resultDetail(testContext.getSkippedConfigurations());

\* resultDetail(testContext.getSkippedTests());

\* resultDetail(testContext.getPassedTests());

\*/

}

}

}

**public** **void** testCaseNo() {

// m\_out.println("<td bgcolor='DeepSkyBlue' colspan='4' align='left' height='30px'><h3 style='margin-top:0px;margin-bottom:0px;'> Total Test Cases : "

// + (qty\_tests) + "<br/> Failed Test Cases : "

// + (failedcount/2) + "<br/> Passed Test cases : "

// + passed + "<br/> Skipped Test cases : "

// + skipped + "<br/></h3></td>");

m\_out.println(

"<table width='350px' height='30px' border='1' align='left'><tbody><tr colspan='2'><td bgcolor='#0088cc' colspan='2'><h3><center><font color='white'>Build Summary</font></center></h3></td></tr><tr><td><b>"

+ "Passed Test cases</b> </td> <td> <center><b>"

+ passed + "</b></center></td></tr><tr><td><b> Failed Test Cases </b></td><td> <center><b>" + (failedcount-skipped)

+ "</b></center></td></tr> <tr><td><b>Skipped Test cases</b> </td><td><center><b> " + skipped

+ "</b></center> </td></tr><tr bgcolor='skyblue'><td> <b>Total Test Cases </b> </td><td> <center><b>" + qty\_tests

+ "</b></center></td></tr></tbody></table>");

}

/\*private void resultSummary\_passed(ISuite suite, IResultMap tests) { //5.3.1

System.out.println("passed count : " + passed);

//passCount= passed;

for (ITestNGMethod method : getMethodSet(tests, suite)) {

passed++;

}

}\*/

**private** **void** resultSummary\_total(ISuite suite, IResultMap tests) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

total\_a++;

}

}

// private void resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

// String style, String details) { //5.4.1

//

// if (tests.getAllResults().size() > 0) {

//

// StringBuffer buff = new StringBuffer();

// String lastClassName = "";

// int mq = 0;

// int cq = 0;

// for (ITestNGMethod method : getMethodSet(tests, suite)) {

// ++passed;

// }

// }

// }

**private** **void** resultSummary\_skipped(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

skipped++;

}

}

}

ArrayList<String> PassedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

**if**(!checkpassedTestCases(testname))

{System.***out***.println("Name and Count:"+testname+"=="+passed);

PassedTestCases.add(testname);

++passed;

}

}

}

}

ArrayList<String> PassedTestName = **new** ArrayList<String>();

**public** **boolean** checkpassedTestCases(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\*

\* **@param** tests

\*/

ArrayList<String> testArray = **new** ArrayList<String>();

**int** retry = 0;

**private** **void** resultSummary(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCases(testname) && !isPassed(testname))

{

testArray.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

{

String id = (m\_testIndex == **null** ? **null** : "t"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><td width='35%'><b>" + testname + "</b></td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

failedcount++; //perivious

}

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

getShortException(tests);

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

**public** **boolean** checkTestCases(String testName)

{

**return** testArray.contains(testName);

}

**public** **boolean** isPassed(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\* Starts and defines columns result summary table \*/

**private** **void** startResultSummaryTable(String style) { //5.2.1

tableStart(style, "summary");

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm a z,MM/dd/yyyy");

m\_out.println("<tr><td bgcolor='white' colspan='4'> <table border='0' width='100%' bgcolor='#e6f7ff'><tr>"

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.kiwiqa.com/wp-content/themes/twentythirteen/images/logo.png'/></center>"

+ "</td><td ><center><font color='#008bcc'><b><h1>Failed Test Cases Analysis</h1></b></font></center></td> "

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.genixventures.com/wp-content/uploads/2015/05/genix\_logo\_03.png'/></center></td> "

+ "</tr></table> </td></tr>");

//m\_out.println("<tr><td colspan='4'>To view Full Report : <a href=\"http://localhost:8080/job/Videogram/HTML\_Report/\">http://localhost:8080/job/Videogram\_Chrome/HTML\_Report</a></td></tr>");

m\_out.println("<tr><td colspan='4'>Overall test suite completion : <b>"

+ Time + " minutes</b><br/> Date and Time of Run: <b>"

+ sdf.format(date) + "</b><br/> Browser : <b>"+SeleniumInit.*browserName*+"<t></t>"

+ SeleniumInit.*browserVersion* + "</b><br/>OS: <b>"

+ System.*getProperty*("os.name") + "</b></td></tr>");

m\_out.println("<tr bgcolor='SkyBlue'><th>Test Cases</th>"

+ "<th>Failure Reason</th><th>Failure Error</th><th>Total Time<br/>(sec.)</th>");

m\_row = 0;

}

**private** String qualifiedName(ITestNGMethod method) {

StringBuilder addon = **new** StringBuilder();

String[] groups = method.getGroups();

**int** length = groups.length;

**if** (length > 0 && !"basic".equalsIgnoreCase(groups[0])) {

addon.append("(");

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

**if** (i > 0) {

addon.append(", ");

}

addon.append(groups[i]);

}

addon.append(")");

}

**return** "<b>" + method.getMethodName() + "</b> " + addon;

}

**private** **void** resultDetail(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

ITestNGMethod method = result.getMethod();

m\_methodIndex++;

String cname = method.getTestClass().getName();

m\_out.println("<h2 id=\"m" + m\_methodIndex + "\">" + cname + ":"

+ method.getMethodName() + "</h2>");

Set<ITestResult> resultSet = tests.getResults(method);

generateForResult(result, method, resultSet.size());

m\_out.println("<p class=\"totop\"><a href=\"#summary\">back to summary</a></p>");

}

}

/\*\*

\* Write the first line of the stack trace

\*

\* **@param** tests

\*/

**private** **void** getShortException(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

**boolean** hasThrowable = exception != **null**;

**if** (hasThrowable) {

String str = Utils.*stackTrace*(exception, **true**)[0];

scanner = **new** Scanner(str);

String firstLine = scanner.nextLine();

m\_out.println("<td width='25%'>");

**for** (Entry<String, String> e : *map*.entrySet()) {

**if** (firstLine.contains(e.getKey())) {

// m\_out.print(map.get(str));

// m\_out.print("contains <br/>");

m\_out.print(e.getValue() + "<br/>");

} **else** {

// m\_out.print("Not contains <br/>");

// m\_out.print(str+"<br/>");

}

}

m\_out.println("</td>");

/\*

\* if(map.containsKey(str)) { m\_out.print(map.get(str)); }else{

\* m\_out.print("Not contains"); m\_out.print(str); }

\*/

m\_out.println("<td width='35%'>");

**boolean** wantsMinimalOutput = result.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

m\_out.print("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

// Getting first line of the stack trace

m\_out.println(firstLine);

m\_out.println("</td>");

}

}

}

/\*\*

\* Write all parameters

\*

\* **@param** tests

\*/

**private** **void** getParameters(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Object[] parameters = result.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

**for** (Object p : parameters) {

m\_out.println(Utils.*escapeHtml*(Utils.*toString*(p)) + " | ");

}

}

}

}

**private** **void** generateForResult(ITestResult ans, ITestNGMethod method,

**int** resultSetSize) {

Object[] parameters = ans.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

tableStart("result", **null**);

m\_out.print("<tr class=\"param\">");

**for** (**int** x = 1; x <= parameters.length; x++) {

m\_out.print("<th>Param." + x + "</th>");

}

m\_out.println("</tr>");

m\_out.print("<tr class=\"param stripe\">");

**for** (Object p : parameters) {

m\_out.println("<td>" + Utils.*escapeHtml*(Utils.*toString*(p))

+ "</td>");

}

m\_out.println("</tr>");

}

List<String> msgs = Reporter.*getOutput*(ans);

**boolean** hasReporterOutput = msgs.size() > 0;

Throwable exception = ans.getThrowable();

**boolean** hasThrowable = exception != **null**;

**if** (hasReporterOutput || hasThrowable) {

**if** (hasParameters) {

m\_out.print("<tr><td");

**if** (parameters.length > 1) {

m\_out.print(" colspan=\"" + parameters.length + "\"");

}

m\_out.println(">");

} **else** {

m\_out.println("<div>");

}

**if** (hasReporterOutput) {

**if** (hasThrowable) {

m\_out.println("<h3>Test Messages</h3>");

}

**for** (String line : msgs) {

m\_out.println(line + "<br/>");

}

}

**if** (hasThrowable) {

**boolean** wantsMinimalOutput = ans.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

m\_out.println("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

generateExceptionReport(exception, method);

}

**if** (hasParameters) {

m\_out.println("</td></tr>");

} **else** {

m\_out.println("</div>");

}

}

**if** (hasParameters) {

m\_out.println("</table>");

}

}

**protected** **void** generateExceptionReport(Throwable exception,

ITestNGMethod method) {

m\_out.print("<div class=\"stacktrace\">");

m\_out.print(Utils.*stackTrace*(exception, **true**)[0]);

m\_out.println("</div>");

}

/\*\*

\* Since the methods will be sorted chronologically, we want to return the

\* ITestNGMethod from the invoked methods.

\*/

**private** Collection<ITestNGMethod> getMethodSet(IResultMap tests,

ISuite suite) {

List<IInvokedMethod> r = Lists.*newArrayList*();

List<IInvokedMethod> invokedMethods = suite.getAllInvokedMethods();

**for** (IInvokedMethod im : invokedMethods) {

**if** (tests.getAllMethods().contains(im.getTestMethod())) {

r.add(im);

}

}

Arrays.*sort*(r.toArray(**new** IInvokedMethod[r.size()]), **new** TestSorter());

List<ITestNGMethod> result = Lists.*newArrayList*();

// Add all the invoked methods

**for** (IInvokedMethod m : r) {

result.add(m.getTestMethod());

}

// Add all the methods that weren't invoked (e.g. skipped) that we

// haven't added yet

**for** (ITestNGMethod m : tests.getAllMethods()) {

**if** (!result.contains(m)) {

result.add(m);

}

}

**return** result;

}

@SuppressWarnings("unused")

**public** **void** generateSuiteSummaryReport(List<ISuite> suites) {

/\*tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Browser");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time(hh:mm:ss)");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups");

m\_out.println("</tr>");\*/

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

**int** qty\_pass\_m = 0;

**int** qty\_pass\_s = 0;

**int** qty\_skip = 0;

**long** time\_start = Long.***MAX\_VALUE***;

**int** qty\_fail = 0;

**long** time\_end = Long.***MIN\_VALUE***;

m\_testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() >= 1) {

//titleRow(suite.getName(), 10);

}

Map<String, ISuiteResult> tests = suite.getResults();

**for** (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

//startSummaryRow(overview.getName());

**int** q = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass\_m += q;

System.***err***.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa----->"+qty\_tests);

//summaryCell(q, Integer.MAX\_VALUE);

/\*q = getMethodSet(overview.getSkippedTests(), suite).size();

qty\_skip += q;

summaryCell(q, 0);

q = getMethodSet(overview.getFailedTests(), suite).size();

qty\_fail += q;\*/

// summaryCell(q, 0);

// Write OS and Browser

/\* summaryCell(suite.getParameter("browserType"), true);

m\_out.println("</td>");

SimpleDateFormat summaryFormat = new SimpleDateFormat("hh:mm:ss");

summaryCell(summaryFormat.format(overview.getStartDate()),true);

m\_out.println("</td>");

summaryCell(summaryFormat.format(overview.getEndDate()),true);

m\_out.println("</td>");

time\_start = Math.min(overview.getStartDate().getTime(), time\_start);

time\_end = Math.max(overview.getEndDate().getTime(), time\_end);

//summaryCell(timeConversion((overview.getEndDate().getTime() - overview.getStartDate().getTime()) / 1000), true);

summaryCell(overview.getIncludedGroups());

summaryCell(overview.getExcludedGroups());

m\_out.println("</tr>");

m\_testIndex++; TheMo\*/

}

}

/\*if (qty\_tests > 1) {

m\_out.println("<tr class='total'><td>Total</td>");

summaryCell(qty\_pass\_m, Integer.MAX\_VALUE);

summaryCell(qty\_skip, 0);

summaryCell(qty\_fail, 0);

summaryCell(" ", true);

summaryCell(" ", true);

summaryCell(" ", true);

//summaryCell(timeConversion(((time\_end - time\_start) / 1000)), true);

m\_out.println("<td colspan='3'>&nbsp;</td></tr>");

}

m\_out.println("</table>"); TheMo \*/

}

/\* @SuppressWarnings("unused")

public void generateSuiteSummaryReport(List<ISuite> suites) {

tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("Scenarios<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Error messages");

tableColumnStart("Parameters");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups"); TheMo

m\_out.println("</tr>"); Themo

NumberFormat formatter = new DecimalFormat("#,##0.0");

int qty\_tests = 0;

int qty\_pass\_m = 0;

int qty\_pass\_s = 0;

int qty\_skip = 0;

int qty\_fail = 0;

long time\_start = Long.MAX\_VALUE;

long time\_end = Long.MIN\_VALUE;

m\_testIndex = 1;

for (ISuite suite : suites) {

if (suites.size() > 1) {

titleRow(suite.getName(), 8);

}

Map<String, ISuiteResult> tests = suite.getResults();

for (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

IResultMap irm = overview.getFailedTests();

//startSummaryRow(overview.getName());

\* int q = getMethodSet(overview.getPassedTests(),

\* suite).size(); qty\_pass\_m += q; summaryCell(q,

\* Integer.MAX\_VALUE); q = overview.getPassedTests().size();

\* qty\_pass\_s += q; summaryCell(q, Integer.MAX\_VALUE); q =

\* getMethodSet(overview.getSkippedTests(), suite).size();

\* qty\_skip += q; summaryCell(q, 0);

int q = getMethodSet(overview.getFailedTests(), suite).size();

qty\_fail += q;

//ummaryCell(q, 0);

int q1 = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass += q1;

// summaryCell(q1, 0); TheMo

// NEW

// Insert error found

// m\_out.print("<td class=\"numi" + (true ? "" : "\_attn") + "\">"); TheMo

getShortException(overview.getFailedTests());

getShortException(overview.getSkippedTests()); TheMo

// m\_out.println("</td>"); TheMo

// NEW

// Add parameters for each test case (failed or passed)

// m\_out.print("<td class=\"numi" + (true ? "" : "\_attn") + "\">"); TheMo

// Write OS and Browser

// m\_out.println(suite.getParameter("os").substring(0, 3) +

// " | "

// + suite.getParameter("browser").substring(0, 3) + " | ");

//getParameters(overview.getFailedTests()); TheMo

// getParameters(overview.getPassedTests());

// getParameters(overview.getSkippedTests());

// m\_out.println("</td>"); TheMo

Date dt = new Date();

// NEW

summaryCell(dt.toString(), true);

m\_out.println("</td>");

summaryCell(dt.toString(), true);

m\_out.println("</td>"); TheMo

time\_start = Math.min(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.max(overview.getEndDate().getTime(), time\_end);

summaryCell(

formatter.format((overview.getEndDate().getTime() - overview

.getStartDate().getTime()) / 1000.)

+ " seconds", true);

summaryCell(overview.getIncludedGroups());

summaryCell(overview.getExcludedGroups());

m\_out.println("</tr>");

m\_testIndex++; TheMo

}

}

if (qty\_tests > 1) {

m\_out.println("<tr class=\"total\"><td>Total</td>");

summaryCell(qty\_pass\_m, Integer.MAX\_VALUE);

summaryCell(qty\_pass\_s, Integer.MAX\_VALUE);

summaryCell(qty\_skip, 0);

summaryCell(qty\_fail, 0);

summaryCell(" ", true);

summaryCell(" ", true);

summaryCell(" ", true);

summaryCell(" ", true);

summaryCell(

formatter.format(((time\_end - time\_start) / 1000.) / 60.)

+ " minutes", true);

m\_out.println("<td colspan=\"3\">&nbsp;</td></tr>"); TheMo

}

m\_out.println("</table>"); TheMo

}\*/

**private** **void** summaryCell(String[] val) {

StringBuffer b = **new** StringBuffer();

**for** (String v : val) {

b.append(v + " ");

}

summaryCell(b.toString(), **true**);

}

**private** **void** summaryCell(String v, **boolean** isgood) {

m\_out.print("<td class=\"numi" + (isgood ? "" : "\_attn") + "\">" + v

+ "</td>");

}

**private** **void** startSummaryRow(String label) {

m\_row += 1;

m\_out.print("<tr"

+ (m\_row % 2 == 0 ? " class=\"stripe\"" : "")

+ "><td style=\"text-align:left;padding-right:2em\"><a href=\"#t"

+ m\_testIndex + "\">" + label + "</a>" + "</td>");

}

**private** **void** summaryCell(**int** v, **int** maxexpected) {

summaryCell(String.*valueOf*(v), v <= maxexpected);

}

**private** **void** tableStart(String cssclass, String id) {

m\_out.println("<table width='80%' border=\"5\" cellspacing=\"0\" cellpadding=\"0\""

+ (cssclass != **null** ? " class=\"" + cssclass + "\"" : " ")

+ (id != **null** ? " id=\"" + id + "\"" : "") + ">");

m\_row = 0;

}

**private** **void** tableColumnStart(String label) {

m\_out.print("<th>" + label + "</th>");

}

**private** **void** titleRow(String label, **int** cq) {

titleRow(label, cq, **null**);

}

**private** **void** titleRow(String label, **int** cq, String id) {

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><th bgcolor='#cce6ff' colspan=\"" + cq + "\"><font color='black' style='text-shadow:2px 2px white;'>" + label + "<font></th></tr>");

m\_row = 0;

}

/\*\* Starts HTML stream \*/

**protected** **void** startHtml(PrintWriter out) { //3.1

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<title> Automation build Summary - TestNG Report</title>");

out.println("<style type=\"text/css\">");

out.println("table {margin-bottom:1px;border-collapse:collapse;empty-cells:show}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println(".result th {vertical-align:bottom}");

out.println(".param th {padding-left:1em;padding-right:1em}");

out.println(".param td {padding-left:.5em;padding-right:2em}");

out.println(".stripe td,.stripe th {background-color: #E6EBF9}");

out.println(".numi,.numi\_attn {text-align:right}");

out.println(".total td {font-weight:bold}");

out.println(".passedodd td {background-color: #0A0}");

out.println(".passedeven td {background-color: #3F3}");

out.println(".skippedodd td {background-color: #CCC}");

out.println(".skippedodd td {background-color: #DDD}");

out.println(".failedodd td,.numi\_attn {background-color: #F9C1C1}");

out.println(".failedeven td,.stripe .numi\_attn {background-color: #F9C1C1}");

out.println(".stacktrace {white-space:pre;font-family:monospace}");

out.println(".totop {font-size:85%;text-align:center;border-bottom:2px solid #000}");

out.println("</style>");

out.println("</head>");

out.println("<body>");

}

/\*\* Finishes HTML stream \*/

**protected** **void** endHtml(PrintWriter out) {

//out.println("<center> Report customized by KiwiQA </center><br/><br/>");

out.println("<tr bgcolor='SkyBlue'><td align='right' colspan='4'><center><b><i>Report customized by KiwiQA </i><b><center></center></b></b></center></td></tr>");

out.println("</body></html>");

}

// ~ Inner Classes --------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**private** **class** TestSorter **implements** Comparator<IInvokedMethod> {

// ~ Methods

// -------------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**public** **int** compare(IInvokedMethod o1, IInvokedMethod o2) {

// System.out.println("Comparing " + o1.getMethodName() + " " +

// o1.getDate()

// + " and " + o2.getMethodName() + " " + o2.getDate());

**return** (**int**) (o1.getDate() - o2.getDate());

// int r = ((T) o1).getTestClass().getName().compareTo(((T)

// o2).getTestClass().getName());

// if (r == 0) {

// r = ((T) o1).getMethodName().compareTo(((T) o2).getMethodName());

// }

// return r;

}

}

}

1. **Class name :** CustomReporterBuildSummary

**package** com.utility;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Scanner;

**import** java.util.Set;

**import** org.testng.IInvokedMethod;

**import** org.testng.IResultMap;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestClass;

**import** org.testng.ITestContext;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.collections.Lists;

**import** org.testng.internal.Utils;

**import** org.testng.log4testng.Logger;

**import** org.testng.xml.XmlSuite;

**public** **class** CustomReporterBuildSummary **extends** CustomReporterListener {

**private** **static** **final** Logger ***L*** = Logger

.*getLogger*(CustomReporterListener.**class**);

// ~ Instance fields ------------------------------------------------------

**private** PrintWriter m\_out;

**private** **int** m\_row;

**private** Integer m\_testIndex;

**private** **int** m\_methodIndex;

**public** **static** **int** *num*=0;

**public** **static** String *id1*="";

**public** **static** **int** *g*=0;

**private** Scanner scanner;

**int** passCount=0;

**private** **static** HashMap<String, String> *map* = **new** HashMap<String, String>();

**int** namecount = 0;

**int** qty\_tests = 0;

**int** passed = 0;

**int** skipped = 0;

**int** failedcount = 0;

**int** total\_a = 0;

**int** qty\_pass= 0;

// ~ Methods --------------------------------------------------------------

/\*

\* public static void maperrors() {

\*

\* System.out.println(

\* "In Error mapping..............\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3465768487488............"

\* ); map.put(

\* "org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure. "

\* , "Server or Node is not running.");

\* map.put("Timed out after 20 seconds waiting for visibility of Proxy element"

\* ,

\* "Particular Element is not found on page after 20 second waiting can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("Unable to locate the element",

\* "Particular Element is not located on page can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("no such element",

\* "element could not be found. you may want to check : 1.Check your xpath in automation code. 2.Element may not present on the screen."

\* ); map.put("Unable to bind to locking port 7054 within 45000 ms",

\* "Port is already locked by other browser and may not be free.");

\* map.put("Unexpected error launching Internet Explorer",

\* "Unable to launch IE.");

\* map.put("Unable to find element on closed window",

\* "Browser window may closed unexpectedly.");

\* map.put("Error communicating with the remote browser.",

\* "remote browser may have died.");

\* map.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

\* "xpath of the particular element getting changed OR Page is not loaded completely."

\* ); map.put(

\* "Error forwarding the new session Error forwarding the request Connect to"

\* , "Connection may be refused by the node/server."); }

\*/

**public** CustomReporterBuildSummary() {

**super**();

*map*.put("Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.",

"Server or Node is not running.");

*map*.put("Timed out after 35 seconds waiting for visibility of Proxy element",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

*map*.put("Unable to locate",

"Particular Element is not located on page. Either Page is not loaded completely OR Element is Not found on page");

*map*.put("no such element",

"element could not be found. Check Possibility: 1.Update automation code. 2.Element may not present on the screen. 3.Possibility for \"BUG\"");

*map*.put("Unable to bind to locking port 7054 within 45000 ms",

"Port is already locked by other browser and may not be free.Please restart selenium node and server");

*map*.put("Unexpected error launching Internet Explorer",

"Unable to launch IE.");

*map*.put("Unable to find element on closed window",

"Browser window may closed unexpectedly. This will fix automatically on next run");

*map*.put("Error communicating with the remote browser.",

"remote browser may have died. Please restart selenium node and server");

*map*.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

"xpath of the particular element getting changed OR Page is not loaded completely.");

*map*.put("Error forwarding the new session Error forwarding the request Connect to",

"Connection may be refused by the node/server. Please restart selenium node and server");

*map*.put("element not visible",

"Element is not found on page : 1.Update automation code. 2.Element may not present on the screen.");

*map*.put("Timed out after 35 seconds waiting for visibility of [[AppiumDriver:",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

}

/\*\* Creates summary of the run \*/

@Override

**public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites,

String outdir) { //1

**try** {

m\_out = createWriter(outdir); //2

} **catch** (IOException e) {

***L***.error("output file", e);

**return**;

}

startHtml(m\_out); //3

generateSuiteSummaryReport(suites);

TotalTime(suites); //4

generateMethodSummaryReport(suites); //5

// generateMethodDetailReport(suites);

//endHtml(m\_out); //previous

m\_out.flush();

m\_out.close();

}

String Time;

**public** String TotalTime(List<ISuite> suites) { //4.1

**long** time\_start = Long.***MAX\_VALUE***;

**long** time\_end = Long.***MIN\_VALUE***;

ITestContext overview = **null**;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> itests = suite.getResults();

**for** (ISuiteResult r : itests.values()) {

overview = r.getTestContext();

time\_start = Math.*min*(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.*max*(overview.getEndDate().getTime(), time\_end);

}

}

// m\_out.println("</tr><td class=\"numi\"><center>"+((time\_end -

// time\_start) / 1000.) / 60.+"</center></td> </tr>");

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

Time = String.*valueOf*(formatter

.format(((time\_end - time\_start) / 1000.) / 60.));

**return** Time;

}

**protected** PrintWriter createWriter(String outdir) **throws** IOException { //2.1

// java.util.Date now = new Date();

**new** File(outdir).mkdirs();

**return** **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(**new** File(

outdir, "CustomReporterBuildSummary" + ".html"))));

}

/\*\*

\* Creates a table showing the highlights of each test method with links to

\* the method details

\*/

**protected** **void** generateMethodSummaryReport(List<ISuite> suites) { //5.1

m\_methodIndex = 0;

startResultSummaryTable("methodOverview"); //5.2

**int** testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() > 1) {

//titleRow(suite.getName(), 5);

}

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

String testName = testContext.getName();

m\_testIndex = testIndex;

//resultSummary\_passed(suite, testContext.getPassedTests()); //5.3

System.***out***.println("Passed---");

resultSummary\_passed(suite, testContext.getPassedTests(), testName,

"passed", "");

/\*resultSummarypassed(suite, testContext.getPassedTests(), //5.4

testName, "passed", " (configuration methods)");\*/

System.***out***.println("Failed---");

/\*resultSummary(suite, testContext.getFailedConfigurations(), //5.4

testName, "failed", " (configuration methods)");\*/

resultSummary(suite, testContext.getFailedTests(), testName,

"failed", "");

System.***out***.println("Skipped---");

resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"skipped", "");

/\*resultSummarypassed(suite, testContext.getSkippedTests(), //5.4

testName, "passed", " (configuration methods)");\*/

/\*

\* resultSummary(suite, testContext.getSkippedConfigurations(),

\* testName, "skipped", " (configuration methods)");

\* resultSummary(suite, testContext.getSkippedTests(), testName,

\* "skipped", ""); resultSummary(suite,

\* testContext.getPassedTests(), testName, "passed", "");

\*/

testIndex++;

}

}

endHtml(m\_out);

testCaseNo();

m\_out.println("</table>");

}

/\*\* Creates a section showing known results for each method \*/

**protected** **void** generateMethodDetailReport(List<ISuite> suites) {

m\_methodIndex = 0;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

**if** (r.values().size() > 0) {

m\_out.println("<h1>" + testContext.getName() + "</h1>");

}

resultDetail(testContext.getFailedConfigurations());

resultDetail(testContext.getFailedTests());

/\*

\* resultDetail(testContext.getSkippedConfigurations());

\* resultDetail(testContext.getSkippedTests());

\* resultDetail(testContext.getPassedTests());

\*/

}

}

}

**public** **void** testCaseNo() {

// m\_out.println("<td bgcolor='DeepSkyBlue' colspan='4' align='left' height='30px'><h3 style='margin-top:0px;margin-bottom:0px;'> Total Test Cases : "

// + (qty\_tests) + "<br/> Failed Test Cases : "

// + (failedcount/2) + "<br/> Passed Test cases : "

// + passed + "<br/> Skipped Test cases : "

// + skipped + "<br/></h3></td>");

m\_out.println("<center>");

m\_out.println(

"<table width='450px' height='200px' border='1' align='left'><tbody><tr colspan='2'><td bgcolor='#0088cc' colspan='2'><h1><center><font color='white'>Build Summary</font></center></h1></td></tr><tr bgcolor='#b1edc0'><td><b>"

+ "<h2><a href='CustomReporterTestPassedDetails.html' target='iframepass' >Passed Test cases</a></b> </td> <td> <center><b>"

+ passed + "</b></center></td></tr><tr><td><b><a href='CustomReporterTestFailureDetails.html' target='iframepass'>Failed Test Cases</a></b></td><td> <center><b>" + (failedcount)

+ "</b></center></td></tr> <tr bgcolor='#fcf77a'><td><b><a href='CustomReporterTestSkippedDetails.html' target='iframepass' >Skipped Test cases</b> </td><td><center><b> " + skipped

+ "</b></center> </td></tr><tr bgcolor='skyblue'><td> <b>Total Test Cases </b> </td><td> <center><b>" + qty\_tests

+ "</b></center></td></tr></tbody></table>");

m\_out.println("<div><iframe name='iframepass' src='CustomReporterTestPassedDetails.html' align='top' height='1000px' width='71%' hspace='10' vspace='10'></div>");

m\_out.println("</center>");

}

/\*private void resultSummary\_passed(ISuite suite, IResultMap tests) { //5.3.1

System.out.println("passed count : " + passed);

//passCount= passed;

for (ITestNGMethod method : getMethodSet(tests, suite)) {

passed++;

}

}\*/

**private** **void** resultSummary\_total(ISuite suite, IResultMap tests) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

total\_a++;

}

}

// private void resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

// String style, String details) { //5.4.1

//

// if (tests.getAllResults().size() > 0) {

//

// StringBuffer buff = new StringBuffer();

// String lastClassName = "";

// int mq = 0;

// int cq = 0;

// for (ITestNGMethod method : getMethodSet(tests, suite)) {

// ++passed;

// }

// }

// }

ArrayList<String> skippedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_skipped(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

skippedTestCases.add(testname);

skipped++;

}

}

}

ArrayList<String> PassedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

**if**(!checkpassedTestCases(testname))

{System.***out***.println("Name and Count:"+testname+"=="+passed);

PassedTestCases.add(testname);

++passed;

}

}

}

}

ArrayList<String> PassedTestName = **new** ArrayList<String>();

**public** **boolean** checkpassedTestCases(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\*

\* **@param** tests

\*/

ArrayList<String> testArray = **new** ArrayList<String>();

**int** retry = 0;

**private** **void** resultSummary(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCases(testname) && !isPassed(testname))

{

testArray.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

{

String id = (m\_testIndex == **null** ? **null** : "test"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

*num*=m\_testIndex;

//m\_out.print("<tr");

**if** (id != **null**) {

//m\_out.print(" id=\"" + id + "\"");

*id1*=id;

}

//m\_out.println("><td width='25%' style=\"font-size:14px; font-family:Times New Roman;\">" + testname + "</td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

failedcount++; //perivious

}

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

//getShortException(tests,num,id1);

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

/\*m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");\*/

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

ArrayList<String> testArraypassed = **new** ArrayList<String>();

**private** **void** resultSummarypassed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCasespassed(testname) && (isPassed(testname) || isskipped(testname)))

{

testArraypassed.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

String id = (m\_testIndex == **null** ? **null** : "test"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

*num*=m\_testIndex;

m\_out.print("<tr");

**if** (id != **null**) {

//m\_out.print(" id=\"" + id + "\"");

*id1*=id;

}

m\_out.println("><td width='25%' style=\"font-size:14px; font-family:Times New Roman;background-color:##aae2c4\">" + testname + "</td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

//failedcount++; //perivious

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

**if**(isskipped(testname))

{

getPass(tests,*num*,*id1*,"#fcf77a");

}**else**{

getPass(tests,*num*,*id1*,"#b1edc0");

}

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

//m\_out.println("<table><tr><td></td></tr></table>");

//m\_out.println("<td width='5%' class=\"numi\"><center> </center></td>");

m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

**public** **boolean** checkTestCasespassed(String testName)

{

**return** testArraypassed.contains(testName);

}

**public** **boolean** checkTestCases(String testName)

{

**return** testArray.contains(testName);

}

**public** **boolean** isPassed(String testName)

{

**return** PassedTestCases.contains(testName);

}

**public** **boolean** isskipped(String testName)

{

**return** skippedTestCases.contains(testName);

}

/\*\* Starts and defines columns result summary table \*/

**private** **void** startResultSummaryTable(String style) { //5.2.1

tableStart(style, "summary");

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm a z,MM/dd/yyyy");

m\_out.println(

"<tr><td bgcolor='white' colspan='4'> <table border='0' width='100%' bgcolor='#e6f7ff'><tr>"

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='https://www.kiwiqa.com/wp-content/uploads/2017/06/KiwiQA\_option2.png'/></center>"

+ "</td><td ><center><font color='#008bcc'><b><h1>OWLS Automation Test Cases Analysis</h1></b></font></center></td> "

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.genixventures.com/wp-content/uploads/2015/05/genix\_logo\_03.png'/></center></td> "

+ "</tr></table> </td></tr>");

//m\_out.println("<tr><td colspan='4'>To view Full Report : <a href=\"http://localhost:8080/job/Videogram/HTML\_Report/\">http://localhost:8080/job/Videogram\_Chrome/HTML\_Report</a></td></tr>");

m\_out.println("<tr><td colspan='4'>Overall test suite completion : <b>"

+ Time + " minutes</b><br/> Date and Time of Run: <b>"

+ sdf.format(date) + "</b><br/> Browser : <b> <t></t> </b><br/>OS: <b>"

+ System.*getProperty*("os.name") + "</b></td></tr>");

/\* m\_out.println("<tr bgcolor='SkyBlue'><th>Test Cases</th><th>Steps</th>"

+ "<th>Failure Reason</th><th>Total Time<br/>(sec.)</th>");\*/

m\_row = 0;

}

**private** String qualifiedName(ITestNGMethod method) {

StringBuilder addon = **new** StringBuilder();

String[] groups = method.getGroups();

**int** length = groups.length;

**if** (length > 0 && !"basic".equalsIgnoreCase(groups[0])) {

addon.append("(");

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

**if** (i > 0) {

addon.append(", ");

}

addon.append(groups[i]);

}

addon.append(")");

}

**return** "<b>" + method.getMethodName() + "</b> " + addon;

}

**private** **void** resultDetail(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

ITestNGMethod method = result.getMethod();

m\_methodIndex++;

String cname = method.getTestClass().getName();

m\_out.println("<h2 id=\"m" + m\_methodIndex + "\">" + cname + ":"

+ method.getMethodName() + "</h2>");

Set<ITestResult> resultSet = tests.getResults(method);

generateForResult(result, method, resultSet.size());

m\_out.println("<p class=\"totop\"><a href=\"#summary\">back to summary</a></p>");

}

}

**private** **void** getPass(IResultMap tests, **int** num, String id,String color) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

//String str = Utils.stackTrace(exception, true)[0];

// scanner = new Scanner(str);

//String firstLine = scanner.nextLine();

m\_out.println("<td width='50%'");

**for** (String line : msgs) {

**if**(*g*==0)

{

/\*m\_out.println(" style=\"background-color:"+color+"\"><b id=\""+id+"\">");

m\_out.println("<a href=\"#hide"+num+"\" data-toggle=\"tooltip\" title=\"Click here to see list of steps\" class=\"hide\" id=\"hide"+num+"\">+</a>");

m\_out.println("<a href=\"#show"+num+"\" class=\"show\" id=\"show"+num+"\">-</a></b> &nbsp;&nbsp;Click here to see list of steps</br></br>");

m\_out.println("<table id=\"t"+num+"\" style=\"display: none;\"><tr><td>");\*/

}

**else**

{

m\_out.println(line+"");

}

*g*++;

**if**(msgs.size()==*g*)

m\_out.println("</td></tr></table></td>");

m\_out.println("<td></td>");

}

**if**(*g*==0||*g*==1){

//m\_out.println("style=\"background-color:#fcf77a\"><b><font color='Blue'>Skipped</font></b></br>");

//m\_out.println(firstLine);

}

m\_out.println("</td>");

*g*=0;

**boolean** hasThrowable = exception != **null**;

**if** (hasThrowable) {

//m\_out.println("<td width='15%'>");

**for** (Entry<String, String> e : *map*.entrySet()) {

/\*if (firstLine.contains(e.getKey())) {

// m\_out.print(map.get(str));

// m\_out.print("contains <br/>");

m\_out.print(e.getValue() + "<br/>");

} else {

// m\_out.print("Not contains <br/>");

// m\_out.print(str+"<br/>");

}\*/

}

//m\_out.println("</td>");

/\*

\* if(map.containsKey(str)) { m\_out.print(map.get(str)); }else{

\* m\_out.print("Not contains"); m\_out.print(str); }

\*/

/\* m\_out.println("<td width='15%'>");

boolean wantsMinimalOutput = result.getStatus() == ITestResult.SUCCESS;

if (hasReporterOutput) {

m\_out.print("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

// Getting first line of the stack trace

m\_out.println(firstLine);

m\_out.println("</td>");\*/

}

}

}

/\*\*

\* Write the first line of the stack trace

\*

\* **@param** tests

\*/

**private** **void** getShortException(IResultMap tests, **int** num, String id) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

//String str = Utils.stackTrace(exception, true)[0];

// scanner = new Scanner(str);

//String firstLine = scanner.nextLine();

m\_out.println("<td width='50%'");

**for** (String line : msgs) {

**if**(*g*==0)

{

m\_out.println(" style=\"background-color:#fae7e6\"><b id=\""+id+"\">");

m\_out.println("<a href=\"#hide"+num+"\" data-toggle=\"tooltip\" title=\"Click here to see list of steps\" class=\"hide\" id=\"hide"+num+"\">+</a>");

m\_out.println("<a href=\"#show"+num+"\" class=\"show\" id=\"show"+num+"\">-</a></b> &nbsp;&nbsp;Click here to see list of steps</br></br>");

m\_out.println("<table id=\"t"+num+"\" style=\"display: none;\"><tr><td>");

}

**else**{

m\_out.println(line);

}

*g*++;

**if**(msgs.size()==*g*)

m\_out.println("</td></tr></table></td>");

}

**if**(*g*==0||*g*==1){

m\_out.println("style=\"background-color:#fcf77a\"><b><font color='Blue'>Skipped</font></b></br>");

//m\_out.println(firstLine);

}

m\_out.println("</td>");

*g*=0;

**boolean** hasThrowable = exception != **null**;

//if (hasThrowable || !hasThrowable)

//{

m\_out.println("<td width='15%'>");

**for** (Entry<String, String> e : *map*.entrySet()) {

}

m\_out.println("</td>");

//}

}

}

/\*\*

\* Write all parameters

\*

\* **@param** tests

\*/

**private** **void** getParameters(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Object[] parameters = result.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

**for** (Object p : parameters) {

m\_out.println(Utils.*escapeHtml*(Utils.*toString*(p)) + " | ");

}

}

}

}

**private** **void** generateForResult(ITestResult ans, ITestNGMethod method,

**int** resultSetSize) {

Object[] parameters = ans.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

//tableStart("result", null);

//m\_out.print("<tr class=\"param\">");

**for** (**int** x = 1; x <= parameters.length; x++) {

//m\_out.print("<th>Param." + x + "</th>");

}

/\*m\_out.println("</tr>");

m\_out.print("<tr class=\"param stripe\">");\*/

**for** (Object p : parameters) {

/\*m\_out.println("<td>" + Utils.escapeHtml(Utils.toString(p))

+ "</td>");\*/

}

//m\_out.println("</tr>");

}

List<String> msgs = Reporter.*getOutput*(ans);

**boolean** hasReporterOutput = msgs.size() > 0;

Throwable exception = ans.getThrowable();

**boolean** hasThrowable = exception != **null**;

**if** (hasReporterOutput || hasThrowable) {

**if** (hasParameters) {

//m\_out.print("<tr><td");

**if** (parameters.length > 1) {

// m\_out.print(" colspan=\"" + parameters.length + "\"");

}

//m\_out.println(">");

} **else** {

// m\_out.println("<div>");

}

**if** (hasReporterOutput) {

**if** (hasThrowable) {

//m\_out.println("<h3>Test Messages</h3>");

}

**for** (String line : msgs) {

//m\_out.println(line + "<br/>");

}

}

**if** (hasThrowable) {

**boolean** wantsMinimalOutput = ans.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

/\*m\_out.println("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");\*/

}

//generateExceptionReport(exception, method);

}

**if** (hasParameters) {

//m\_out.println("</td></tr>");

} **else** {

//m\_out.println("</div>");

}

}

**if** (hasParameters) {

//m\_out.println("</table>");

}

}

**protected** **void** generateExceptionReport(Throwable exception,

ITestNGMethod method) {

m\_out.print("<div class=\"stacktrace\">");

m\_out.print(Utils.*stackTrace*(exception, **true**)[0]);

m\_out.println("</div>");

}

/\*\*

\* Since the methods will be sorted chronologically, we want to return the

\* ITestNGMethod from the invoked methods.

\*/

**private** Collection<ITestNGMethod> getMethodSet(IResultMap tests,

ISuite suite) {

List<IInvokedMethod> r = Lists.*newArrayList*();

List<IInvokedMethod> invokedMethods = suite.getAllInvokedMethods();

**for** (IInvokedMethod im : invokedMethods) {

**if** (tests.getAllMethods().contains(im.getTestMethod())) {

r.add(im);

}

}

Arrays.*sort*(r.toArray(**new** IInvokedMethod[r.size()]), **new** TestSorter());

List<ITestNGMethod> result = Lists.*newArrayList*();

// Add all the invoked methods

**for** (IInvokedMethod m : r) {

result.add(m.getTestMethod());

}

// Add all the methods that weren't invoked (e.g. skipped) that we

// haven't added yet

**for** (ITestNGMethod m : tests.getAllMethods()) {

**if** (!result.contains(m)) {

result.add(m);

}

}

**return** result;

}

@SuppressWarnings("unused")

**public** **void** generateSuiteSummaryReport(List<ISuite> suites) {

/\*tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Browser");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time(hh:mm:ss)");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups");

m\_out.println("</tr>");\*/

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

**int** qty\_pass\_m = 0;

**int** qty\_pass\_s = 0;

**int** qty\_skip = 0;

**long** time\_start = Long.***MAX\_VALUE***;

**int** qty\_fail = 0;

**long** time\_end = Long.***MIN\_VALUE***;

m\_testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() >= 1) {

//titleRow(suite.getName(), 10);

}

Map<String, ISuiteResult> tests = suite.getResults();

**for** (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

//startSummaryRow(overview.getName());

**int** q = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass\_m += q;

System.***err***.println("aa----->"+qty\_tests);

}

}

}

**private** **void** summaryCell(String[] val) {

StringBuffer b = **new** StringBuffer();

**for** (String v : val) {

b.append(v + " ");

}

summaryCell(b.toString(), **true**);

}

**private** **void** summaryCell(String v, **boolean** isgood) {

m\_out.print("<td class=\"numi" + (isgood ? "" : "\_attn") + "\">" + v

+ "</td>");

}

**private** **void** startSummaryRow(String label) {

m\_row += 1;

m\_out.print("<tr"

+ (m\_row % 2 == 0 ? " class=\"stripe\"" : "")

+ "><td style=\"text-align:left;padding-right:2em\"><a href=\"#t"

+ m\_testIndex + "\">" + label + "</a>" + "</td>");

}

**private** **void** summaryCell(**int** v, **int** maxexpected) {

summaryCell(String.*valueOf*(v), v <= maxexpected);

}

**private** **void** tableStart(String cssclass, String id) {

m\_out.println("<table width='100%' border=\"5\" cellspacing=\"0\" cellpadding=\"0\""

+ (cssclass != **null** ? " class=\"" + cssclass + "\"" : " ")

+ (id != **null** ? " id=\"" + id + "\"" : "") + ">");

m\_row = 0;

}

**private** **void** tableColumnStart(String label) {

m\_out.print("<th>" + label + "</th>");

}

**private** **void** titleRow(String label, **int** cq) {

titleRow(label, cq, **null**);

}

**private** **void** titleRow(String label, **int** cq, String id) {

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><th bgcolor='#cce6ff' colspan=\"" + cq + "\"><font color='black' style='text-shadow:2px 2px white;'>" + label + "<font></th></tr>");

m\_row = 0;

}

/\*\* Starts HTML stream \*/

**protected** **void** startHtml(PrintWriter out) { //3.1

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<title> Automation build Summary - TestNG Report</title>");

out.println("<style type=\"text/css\">");

out.println("table {margin-bottom:1px;border-collapse:collapse;empty-cells:show}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println(".result th {vertical-align:bottom}");

out.println(".param th {padding-left:1em;padding-right:1em}");

out.println(".param td {padding-left:.5em;padding-right:2em}");

out.println(".stripe td,.stripe th {background-color: #E6EBF9}");

out.println(".numi,.numi\_attn {text-align:right}");

out.println(".total td {font-weight:bold}");

out.println(".passedodd td {background-color: #0A0}");

out.println(".passedeven td {background-color: #3F3}");

out.println(".skippedodd td {background-color: #CCC}");

out.println(".skippedodd td {background-color: #DDD}");

out.println(".failedodd td,.numi\_attn {background-color: #F9C1C1}");

out.println(".failedeven td,.stripe .numi\_attn {background-color: #F9C1C1}");

out.println(".stacktrace {white-space:pre;}");

out.println(".totop {font-size:85%;text-align:center;border-bottom:2px solid #000}");

out.println("html \* {");

out.println(" font-family: \"Open Sans\",sans-serif; font-size:11px;}");

out.println("h1 { font-size:25px; }");

out.println("th {font-size:14px; }");

/\*\*\*Collapse expands\*\*\*\*/

out.println(".list { display:none;");

out.println("height:auto;");

out.println(" margin:0;");

out.println("float: left; }");

out.println(".show {");

out.println("display: none; }");

out.println(".hide:target + .show {");

out.println("display: inline; }");

out.println(".hide:target {");

out.println("display: none; }");

out.println(".hide:target ~ .list {");

out.println("display:inline; }");

/\*style the (+) and (-) \*/

out.println(".hide, .show {");

out.println("width: 16px;");

out.println("height: 16px;");

out.println("border-radius: 30px;");

out.println("font-size: 15px;");

out.println("color: #000;");

out.println("text-shadow: 0 1px 0 #666;");

out.println("text-align: center;");

out.println("text-decoration: none;");

out.println("box-shadow: 1px 1px 2px #000;");

out.println("background: #91DDFE;");

out.println("opacity: .95;");

out.println("margin-right: 0;");

out.println("float: left;");

out.println("margin-bottom: 25px; }");

out.println(".hide:hover, .show:hover {");

out.println("color: #eee;");

out.println("text-shadow: 0 0 1px #666;");

out.println("text-decoration: none;");

out.println("box-shadow: 0 0 4px #222 inset;");

out.println("opacity: 1;");

out.println("margin-bottom: 25px; }");

out.println(".list tr{");

out.println("height:auto;");

out.println("margin:0; }");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

out.println("</style>");

out.println("<script src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js\"></script>");

out.println("<script>");

out.println("$(document).ready(function(){");

/\*for (int i=1;i<180;i++)

{

out.println(" $(\"#test"+i+"\").click(function(){");

out.println("$(\"#t"+i+"\").toggle(100);");

out.println("});");

}\*/

/\*for (int i = 0; i < CustomReporterTestFailureDetails.failedtoggles.size(); i++) {

out.println(" $(\"#test"+CustomReporterTestFailureDetails.failedtoggles.get(i)+"\").click(function(){");

out.println("$(\"#t"+CustomReporterTestFailureDetails.failedtoggles.get(i)+"\").toggle(100);");

out.println("});");

}

for (int i = 0; i < CustomReporterTestpassDetails.passedtoggles.size(); i++) {

out.println(" $(\"#test"+CustomReporterTestpassDetails.passedtoggles.get(i)+"\").click(function(){");

out.println("$(\"#t"+CustomReporterTestpassDetails.passedtoggles.get(i)+"\").toggle(100);");

out.println("});");

}

for (int i = 0; i < CustomReporterTestskipDetails.skiptoggles.size(); i++) {

out.println(" $(\"#test"+CustomReporterTestskipDetails.skiptoggles.get(i)+"\").click(function(){");

out.println("$(\"#t"+CustomReporterTestskipDetails.skiptoggles.get(i)+"\").toggle(100);");

out.println("});");

}\*/

out.println("});");

out.println("</script>");

out.println("</head>");

out.println("<body>");

}

/\*\* Finishes HTML stream \*/

**protected** **void** endHtml(PrintWriter out) {

//out.println("<center> Report customized by KiwiQA </center><br/><br/>");

out.println("<tr bgcolor='SkyBlue'><td align='right' colspan='4'><center><b><i>Report customized by KiwiQA </i><b><center></center></b></b></center></td></tr>");

out.println("</body></html>");

}

// ~ Inner Classes --------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**private** **class** TestSorter **implements** Comparator<IInvokedMethod> {

// ~ Methods

// -------------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

@Override

**public** **int** compare(IInvokedMethod o1, IInvokedMethod o2) {

// System.out.println("Comparing " + o1.getMethodName() + " " +

// o1.getDate()

// + " and " + o2.getMethodName() + " " + o2.getDate());

**return** (**int**) (o1.getDate() - o2.getDate());

// int r = ((T) o1).getTestClass().getName().compareTo(((T)

// o2).getTestClass().getName());

// if (r == 0) {

// r = ((T) o1).getMethodName().compareTo(((T) o2).getMethodName());

// }

// return r;

}

}

}

1. Class name : CustomReporterListener
2. **package** com.utility;
3. **import** java.util.List;
4. **import** org.testng.IReporter;
5. **import** org.testng.ISuite;
6. **import** org.testng.xml.XmlSuite;
7. **public** **class** CustomReporterListener **implements** IReporter
8. {
9. @Override
10. **public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites, String outdir)
11. {
13. }
14. }

4.) class name: CustomReporterTestFailureDetails

**package** com.utility;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Scanner;

**import** java.util.Set;

**import** org.testng.IInvokedMethod;

**import** org.testng.IResultMap;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestClass;

**import** org.testng.ITestContext;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.collections.Lists;

**import** org.testng.internal.Utils;

**import** org.testng.log4testng.Logger;

**import** org.testng.xml.XmlSuite;

**public** **class** CustomReporterTestFailureDetails **extends** CustomReporterListener {

**private** **static** **final** Logger ***L*** = Logger

.*getLogger*(CustomReporterListener.**class**);

// ~ Instance fields ------------------------------------------------------

**private** PrintWriter m\_out;

**private** **int** m\_row;

**private** Integer m\_testIndex;

**private** **int** m\_methodIndex;

**public** **static** **int** *num*=0;

**public** **static** String *id1*="";

**public** **static** **int** *g*=0;

**private** Scanner scanner;

**int** passCount=0;

**private** **static** HashMap<String, String> *map* = **new** HashMap<String, String>();

**int** namecount = 0;

**int** qty\_tests = 0;

**int** passed = 0;

**int** skipped = 0;

**int** failedcount = 0;

**int** total\_a = 0;

**int** qty\_pass= 0;

**public** **static** ArrayList<String> *failedtoggles* = **new** ArrayList<String>();

// ~ Methods --------------------------------------------------------------

/\*

\* public static void maperrors() {

\*

\* System.out.println(

\* "In Error mapping..............\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3465768487488............"

\* ); map.put(

\* "org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure. "

\* , "Server or Node is not running.");

\* map.put("Timed out after 20 seconds waiting for visibility of Proxy element"

\* ,

\* "Particular Element is not found on page after 20 second waiting can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("Unable to locate the element",

\* "Particular Element is not located on page can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("no such element",

\* "element could not be found. you may want to check : 1.Check your xpath in automation code. 2.Element may not present on the screen."

\* ); map.put("Unable to bind to locking port 7054 within 45000 ms",

\* "Port is already locked by other browser and may not be free.");

\* map.put("Unexpected error launching Internet Explorer",

\* "Unable to launch IE.");

\* map.put("Unable to find element on closed window",

\* "Browser window may closed unexpectedly.");

\* map.put("Error communicating with the remote browser.",

\* "remote browser may have died.");

\* map.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

\* "xpath of the particular element getting changed OR Page is not loaded completely."

\* ); map.put(

\* "Error forwarding the new session Error forwarding the request Connect to"

\* , "Connection may be refused by the node/server."); }

\*/

**public** **void** CustomReporterTestFailureDetails() {

//super();

*map*.put("Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.",

"Server or Node is not running.");

*map*.put("Timed out after 35 seconds waiting for visibility of Proxy element",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

*map*.put("Unable to locate",

"Particular Element is not located on page. Either Page is not loaded completely OR Element is Not found on page");

*map*.put("no such element",

"element could not be found. Check Possibility: 1.Update automation code. 2.Element may not present on the screen. 3.Possibility for \"BUG\"");

*map*.put("Unable to bind to locking port 7054 within 45000 ms",

"Port is already locked by other browser and may not be free.Please restart selenium node and server");

*map*.put("Unexpected error launching Internet Explorer",

"Unable to launch IE.");

*map*.put("Unable to find element on closed window",

"Browser window may closed unexpectedly. This will fix automatically on next run");

*map*.put("Error communicating with the remote browser.",

"remote browser may have died. Please restart selenium node and server");

*map*.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

"xpath of the particular element getting changed OR Page is not loaded completely.");

*map*.put("Error forwarding the new session Error forwarding the request Connect to",

"Connection may be refused by the node/server. Please restart selenium node and server");

*map*.put("element not visible",

"Element is not found on page : 1.Update automation code. 2.Element may not present on the screen.");

*map*.put("Timed out after 35 seconds waiting for visibility of [[AppiumDriver:",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

}

/\*\* Creates summary of the run \*/

@Override

**public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites,

String outdir) { //1

**try** {

m\_out = createWriter(outdir); //2

} **catch** (IOException e) {

***L***.error("output file", e);

**return**;

}

startHtml(m\_out); //3

generateSuiteSummaryReport(suites);

TotalTime(suites); //4

generateMethodSummaryReport(suites); //5

// generateMethodDetailReport(suites);

//endHtml(m\_out); //previous

m\_out.flush();

m\_out.close();

}

String Time;

**public** String TotalTime(List<ISuite> suites) { //4.1

**long** time\_start = Long.***MAX\_VALUE***;

**long** time\_end = Long.***MIN\_VALUE***;

ITestContext overview = **null**;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> itests = suite.getResults();

**for** (ISuiteResult r : itests.values()) {

overview = r.getTestContext();

time\_start = Math.*min*(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.*max*(overview.getEndDate().getTime(), time\_end);

}

}

// m\_out.println("</tr><td class=\"numi\"><center>"+((time\_end -

// time\_start) / 1000.) / 60.+"</center></td> </tr>");

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

Time = String.*valueOf*(formatter

.format(((time\_end - time\_start) / 1000.) / 60.));

**return** Time;

}

**protected** PrintWriter createWriter(String outdir) **throws** IOException { //2.1

// java.util.Date now = new Date();

**new** File(outdir).mkdirs();

**return** **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(**new** File(

outdir, "CustomReporterTestFailureDetails" + ".html"))));

}

/\*\*

\* Creates a table showing the highlights of each test method with links to

\* the method details

\*/

**protected** **void** generateMethodSummaryReport(List<ISuite> suites) { //5.1

m\_methodIndex = 0;

startResultSummaryTable("methodOverview"); //5.2

**int** testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() > 1) {

titleRow(suite.getName(), 5);

}

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

String testName = testContext.getName();

m\_testIndex = testIndex;

//resultSummary\_passed(suite, testContext.getPassedTests()); //5.3

System.***out***.println("Passed---");

resultSummary\_passed(suite, testContext.getPassedTests(), testName,

"passed", "");

System.***out***.println("Failed---");

/\*resultSummary(suite, testContext.getFailedConfigurations(), //5.4

testName, "failed", " (configuration methods)");\*/

resultSummary(suite, testContext.getFailedTests(), testName,

"failed", "");

System.***out***.println("Skipped---");

resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"skipped", "");

/\*

\* resultSummary(suite, testContext.getSkippedConfigurations(),

\* testName, "skipped", " (configuration methods)");

\* resultSummary(suite, testContext.getSkippedTests(), testName,

\* "skipped", ""); resultSummary(suite,

\* testContext.getPassedTests(), testName, "passed", "");

\*/

testIndex++;

}

}

endHtml(m\_out);

//testCaseNo();

m\_out.println("</table>");

}

/\*\* Creates a section showing known results for each method \*/

**protected** **void** generateMethodDetailReport(List<ISuite> suites) {

m\_methodIndex = 0;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

**if** (r.values().size() > 0) {

m\_out.println("<h1>" + testContext.getName() + "</h1>");

}

resultDetail(testContext.getFailedConfigurations());

resultDetail(testContext.getFailedTests());

/\*

\* resultDetail(testContext.getSkippedConfigurations());

\* resultDetail(testContext.getSkippedTests());

\* resultDetail(testContext.getPassedTests());

\*/

}

}

}

**public** **void** testCaseNo() {

// m\_out.println("<td bgcolor='DeepSkyBlue' colspan='4' align='left' height='30px'><h3 style='margin-top:0px;margin-bottom:0px;'> Total Test Cases : "

// + (qty\_tests) + "<br/> Failed Test Cases : "

// + (failedcount/2) + "<br/> Passed Test cases : "

// + passed + "<br/> Skipped Test cases : "

// + skipped + "<br/></h3></td>");

m\_out.println(

"<table width='350px' height='30px' border='1' align='left'><tbody><tr colspan='2'><td bgcolor='#0088cc' colspan='2'><h3><center><font color='white'>Build Summary</font></center></h3></td></tr><tr><td><b>"

+ "Passed Test cases</b> </td> <td> <center><b>"

+ passed + "</b></center></td></tr><tr><td><b> Failed Test Cases </b></td><td> <center><b>" + (failedcount)

+ "</b></center></td></tr> <tr><td><b>Skipped Test cases</b> </td><td><center><b> " + skipped

+ "</b></center> </td></tr><tr bgcolor='skyblue'><td> <b>Total Test Cases </b> </td><td> <center><b>" + qty\_tests

+ "</b></center></td></tr></tbody></table>");

}

/\*private void resultSummary\_passed(ISuite suite, IResultMap tests) { //5.3.1

System.out.println("passed count : " + passed);

//passCount= passed;

for (ITestNGMethod method : getMethodSet(tests, suite)) {

passed++;

}

}\*/

**private** **void** resultSummary\_total(ISuite suite, IResultMap tests) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

total\_a++;

}

}

// private void resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

// String style, String details) { //5.4.1

//

// if (tests.getAllResults().size() > 0) {

//

// StringBuffer buff = new StringBuffer();

// String lastClassName = "";

// int mq = 0;

// int cq = 0;

// for (ITestNGMethod method : getMethodSet(tests, suite)) {

// ++passed;

// }

// }

// }

**private** **void** resultSummary\_skipped(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

skipped++;

}

}

}

ArrayList<String> PassedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

**if**(!checkpassedTestCases(testname))

{System.***out***.println("Name and Count:"+testname+"=="+passed);

PassedTestCases.add(testname);

++passed;

}

}

}

}

ArrayList<String> PassedTestName = **new** ArrayList<String>();

**public** **boolean** checkpassedTestCases(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\*

\* **@param** tests

\*/

ArrayList<String> testArray = **new** ArrayList<String>();

**int** retry = 0;

**private** **void** resultSummary(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCases(testname) && !isPassed(testname))

{

testArray.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

{

String id = (m\_testIndex == **null** ? **null** : "testfailtoggle"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

*num*=m\_testIndex;

m\_out.print("<tr");

**if** (id != **null**) {

//m\_out.print(" id=\"" + id + "\"");

*id1*=id;

}

m\_out.println("><td width='25%' style=\"font-size:14px; font-family:Times New Roman;\">" + testname + "</td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

failedcount++; //perivious

}

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

getShortException(tests,*num*,*id1*);

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

**public** **boolean** checkTestCases(String testName)

{

**return** testArray.contains(testName);

}

**public** **boolean** isPassed(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\* Starts and defines columns result summary table \*/

**private** **void** startResultSummaryTable(String style) { //5.2.1

tableStart(style, "summary");

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm a z,MM/dd/yyyy");

/\*m\_out.println("<tr><td bgcolor='white' colspan='4'> <table border='0' width='100%' bgcolor='#e6f7ff'><tr>"

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.kiwiqa.com/wp-content/themes/twentythirteen/images/logo.png'/></center>"

+ "</td><td ><center><font color='#008bcc'><b><h1>Failed Test Cases Analysis</h1></b></font></center></td> "

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.genixventures.com/wp-content/uploads/2015/05/genix\_logo\_03.png'/></center></td> "

+ "</tr></table> </td></tr>");\*/

//m\_out.println("<tr><td colspan='4'>To view Full Report : <a href=\"http://localhost:8080/job/Videogram/HTML\_Report/\">http://localhost:8080/job/Videogram\_Chrome/HTML\_Report</a></td></tr>");

/\*m\_out.println("<tr><td colspan='4'>Overall test suite completion : <b>"

+ Time + " minutes</b><br/> Date and Time of Run: <b>"

+ sdf.format(date) + "</b><br/> Browser : <b>"+SeleniumInit.browsernm+"<t></t>"

+ SeleniumInit.browserVersion + "</b><br/>OS: <b>"

+ System.getProperty("os.name") + "</b></td></tr>");\*/

m\_out.println("<tr bgcolor='SkyBlue'><th>Test Cases</th><th>Steps</th>"

+ "<th>Failure Reason</th><th>Total Time<br/>(sec.)</th>");

m\_row = 0;

}

**private** String qualifiedName(ITestNGMethod method) {

StringBuilder addon = **new** StringBuilder();

String[] groups = method.getGroups();

**int** length = groups.length;

**if** (length > 0 && !"basic".equalsIgnoreCase(groups[0])) {

addon.append("(");

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

**if** (i > 0) {

addon.append(", ");

}

addon.append(groups[i]);

}

addon.append(")");

}

**return** "<b>" + method.getMethodName() + "</b> " + addon;

}

**private** **void** resultDetail(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

ITestNGMethod method = result.getMethod();

m\_methodIndex++;

String cname = method.getTestClass().getName();

m\_out.println("<h2 id=\"m" + m\_methodIndex + "\">" + cname + ":"

+ method.getMethodName() + "</h2>");

Set<ITestResult> resultSet = tests.getResults(method);

generateForResult(result, method, resultSet.size());

m\_out.println("<p class=\"totop\"><a href=\"#summary\">back to summary</a></p>");

}

}

/\*\*

\* Write the first line of the stack trace

\*

\* **@param** tests

\*/

**private** **void** getShortException(IResultMap tests, **int** num, String id) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

String str = Utils.*stackTrace*(exception, **true**)[0];

scanner = **new** Scanner(str);

String firstLine = scanner.nextLine();

m\_out.println("<td width='50%'");

String toggle = "failtoggle"+num;

*failedtoggles*.add(toggle);

**for** (String line : msgs) {

**if**(*g*==0)

{

m\_out.println(" style=\"background-color:#fae7e6\"><b id=\""+id+"\">");

m\_out.println("<a href=\"#hide"+toggle+"\" data-toggle=\"tooltip\" title=\"Click here to see list of steps\" class=\"hide\" id=\"hide"+toggle+"\">+</a>");

m\_out.println("<a href=\"#show"+toggle+"\" class=\"show\" id=\"show"+toggle+"\">-</a></b> &nbsp;&nbsp;Click here to see list of steps</br></br>");

m\_out.println("<table id=\"t"+toggle+"\" style=\"display: none;\"><tr><td>");

}

**else**

{

m\_out.println(line+"");

}

*g*++;

**if**(msgs.size()==*g*)

m\_out.println("</td></tr></table></td>");

}

**if**(*g*==0||*g*==1){

m\_out.println("style=\"background-color:#fcf77a\"><b><font color='Blue'>Skipped</font></b></br>");

//m\_out.println(firstLine);

}

m\_out.println("</td>");

*g*=0;

**boolean** hasThrowable = exception != **null**;

**if** (hasThrowable) {

m\_out.println("<td width='15%'>");

**for** (Entry<String, String> e : *map*.entrySet()) {

**if** (firstLine.contains(e.getKey())) {

// m\_out.print(map.get(str));

// m\_out.print("contains <br/>");

m\_out.print(e.getValue() + "<br/>");

} **else** {

// m\_out.print("Not contains <br/>");

// m\_out.print(str+"<br/>");

}

}

m\_out.println("</td>");

/\*

\* if(map.containsKey(str)) { m\_out.print(map.get(str)); }else{

\* m\_out.print("Not contains"); m\_out.print(str); }

\*/

/\* m\_out.println("<td width='15%'>");

boolean wantsMinimalOutput = result.getStatus() == ITestResult.SUCCESS;

if (hasReporterOutput) {

m\_out.print("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

// Getting first line of the stack trace

m\_out.println(firstLine);

m\_out.println("</td>");\*/

}

**break**;

}

}

/\*\*

\* Write all parameters

\*

\* **@param** tests

\*/

**private** **void** getParameters(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Object[] parameters = result.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

**for** (Object p : parameters) {

m\_out.println(Utils.*escapeHtml*(Utils.*toString*(p)) + " | ");

}

}

}

}

**private** **void** generateForResult(ITestResult ans, ITestNGMethod method,

**int** resultSetSize) {

Object[] parameters = ans.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

//tableStart("result", null);

//m\_out.print("<tr class=\"param\">");

**for** (**int** x = 1; x <= parameters.length; x++) {

//m\_out.print("<th>Param." + x + "</th>");

}

/\*m\_out.println("</tr>");

m\_out.print("<tr class=\"param stripe\">");\*/

**for** (Object p : parameters) {

/\*m\_out.println("<td>" + Utils.escapeHtml(Utils.toString(p))

+ "</td>");\*/

}

//m\_out.println("</tr>");

}

List<String> msgs = Reporter.*getOutput*(ans);

**boolean** hasReporterOutput = msgs.size() > 0;

Throwable exception = ans.getThrowable();

**boolean** hasThrowable = exception != **null**;

**if** (hasReporterOutput || hasThrowable) {

**if** (hasParameters) {

//m\_out.print("<tr><td");

**if** (parameters.length > 1) {

// m\_out.print(" colspan=\"" + parameters.length + "\"");

}

//m\_out.println(">");

} **else** {

// m\_out.println("<div>");

}

**if** (hasReporterOutput) {

**if** (hasThrowable) {

//m\_out.println("<h3>Test Messages</h3>");

}

**for** (String line : msgs) {

//m\_out.println(line + "<br/>");

}

}

**if** (hasThrowable) {

**boolean** wantsMinimalOutput = ans.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

/\*m\_out.println("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");\*/

}

//generateExceptionReport(exception, method);

}

**if** (hasParameters) {

//m\_out.println("</td></tr>");

} **else** {

//m\_out.println("</div>");

}

}

**if** (hasParameters) {

//m\_out.println("</table>");

}

}

**protected** **void** generateExceptionReport(Throwable exception,

ITestNGMethod method) {

m\_out.print("<div class=\"stacktrace\">");

m\_out.print(Utils.*stackTrace*(exception, **true**)[0]);

m\_out.println("</div>");

}

/\*\*

\* Since the methods will be sorted chronologically, we want to return the

\* ITestNGMethod from the invoked methods.

\*/

**private** Collection<ITestNGMethod> getMethodSet(IResultMap tests,

ISuite suite) {

List<IInvokedMethod> r = Lists.*newArrayList*();

List<IInvokedMethod> invokedMethods = suite.getAllInvokedMethods();

**for** (IInvokedMethod im : invokedMethods) {

**if** (tests.getAllMethods().contains(im.getTestMethod())) {

r.add(im);

}

}

Arrays.*sort*(r.toArray(**new** IInvokedMethod[r.size()]), **new** TestSorter());

List<ITestNGMethod> result = Lists.*newArrayList*();

// Add all the invoked methods

**for** (IInvokedMethod m : r) {

result.add(m.getTestMethod());

}

// Add all the methods that weren't invoked (e.g. skipped) that we

// haven't added yet

**for** (ITestNGMethod m : tests.getAllMethods()) {

**if** (!result.contains(m)) {

result.add(m);

}

}

**return** result;

}

@SuppressWarnings("unused")

**public** **void** generateSuiteSummaryReport(List<ISuite> suites) {

/\*tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Browser");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time(hh:mm:ss)");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups");

m\_out.println("</tr>");\*/

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

**int** qty\_pass\_m = 0;

**int** qty\_pass\_s = 0;

**int** qty\_skip = 0;

**long** time\_start = Long.***MAX\_VALUE***;

**int** qty\_fail = 0;

**long** time\_end = Long.***MIN\_VALUE***;

m\_testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() >= 1) {

//titleRow(suite.getName(), 10);

}

Map<String, ISuiteResult> tests = suite.getResults();

**for** (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

//startSummaryRow(overview.getName());

**int** q = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass\_m += q;

System.***err***.println("aa----->"+qty\_tests);

}

}

}

**private** **void** summaryCell(String[] val) {

StringBuffer b = **new** StringBuffer();

**for** (String v : val) {

b.append(v + " ");

}

summaryCell(b.toString(), **true**);

}

**private** **void** summaryCell(String v, **boolean** isgood) {

m\_out.print("<td class=\"numi" + (isgood ? "" : "\_attn") + "\">" + v

+ "</td>");

}

**private** **void** startSummaryRow(String label) {

m\_row += 1;

m\_out.print("<tr"

+ (m\_row % 2 == 0 ? " class=\"stripe\"" : "")

+ "><td style=\"text-align:left;padding-right:2em\"><a href=\"#t"

+ m\_testIndex + "\">" + label + "</a>" + "</td>");

}

**private** **void** summaryCell(**int** v, **int** maxexpected) {

summaryCell(String.*valueOf*(v), v <= maxexpected);

}

**private** **void** tableStart(String cssclass, String id) {

m\_out.println("<table width='100%' border=\"5\" cellspacing=\"0\" cellpadding=\"0\""

+ (cssclass != **null** ? " class=\"" + cssclass + "\"" : " ")

+ (id != **null** ? " id=\"" + id + "\"" : "") + ">");

m\_row = 0;

}

**private** **void** tableColumnStart(String label) {

m\_out.print("<th>" + label + "</th>");

}

**private** **void** titleRow(String label, **int** cq) {

titleRow(label, cq, **null**);

}

**private** **void** titleRow(String label, **int** cq, String id) {

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><th bgcolor='#cce6ff' colspan=\"" + cq + "\"><font color='black' style='text-shadow:2px 2px white;'>" + label + "<font></th></tr>");

m\_row = 0;

}

/\*\* Starts HTML stream \*/

**protected** **void** startHtml(PrintWriter out) { //3.1

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<title> Automation build Summary - TestNG Report</title>");

out.println("<style type=\"text/css\">");

out.println("table {margin-bottom:1px;border-collapse:collapse;empty-cells:show}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println(".result th {vertical-align:bottom}");

out.println(".param th {padding-left:1em;padding-right:1em}");

out.println(".param td {padding-left:.5em;padding-right:2em}");

out.println(".stripe td,.stripe th {background-color: #E6EBF9}");

out.println(".numi,.numi\_attn {text-align:right}");

out.println(".total td {font-weight:bold}");

out.println(".passedodd td {background-color: #0A0}");

out.println(".passedeven td {background-color: #3F3}");

out.println(".skippedodd td {background-color: #CCC}");

out.println(".skippedodd td {background-color: #DDD}");

out.println(".failedodd td,.numi\_attn {background-color: #F9C1C1}");

out.println(".failedeven td,.stripe .numi\_attn {background-color: #F9C1C1}");

out.println(".stacktrace {white-space:pre;}");

out.println(".totop {font-size:85%;text-align:center;border-bottom:2px solid #000}");

out.println("html \* {");

out.println(" font-family: \"Open Sans\",sans-serif; font-size:11px;}");

out.println("h1 { font-size:25px; }");

out.println("th {font-size:14px; }");

/\*\*\*Collapse expands\*\*\*\*/

out.println(".list { display:none;");

out.println("height:auto;");

out.println(" margin:0;");

out.println("float: left; }");

out.println(".show {");

out.println("display: none; }");

out.println(".hide:target + .show {");

out.println("display: inline; }");

out.println(".hide:target {");

out.println("display: none; }");

out.println(".hide:target ~ .list {");

out.println("display:inline; }");

/\*style the (+) and (-) \*/

out.println(".hide, .show {");

out.println("width: 16px;");

out.println("height: 16px;");

out.println("border-radius: 30px;");

out.println("font-size: 15px;");

out.println("color: #000;");

out.println("text-shadow: 0 1px 0 #666;");

out.println("text-align: center;");

out.println("text-decoration: none;");

out.println("box-shadow: 1px 1px 2px #000;");

out.println("background: #91DDFE;");

out.println("opacity: .95;");

out.println("margin-right: 0;");

out.println("float: left;");

out.println("margin-bottom: 25px; }");

out.println(".hide:hover, .show:hover {");

out.println("color: #eee;");

out.println("text-shadow: 0 0 1px #666;");

out.println("text-decoration: none;");

out.println("box-shadow: 0 0 4px #222 inset;");

out.println("opacity: 1;");

out.println("margin-bottom: 25px; }");

out.println(".list tr{");

out.println("height:auto;");

out.println("margin:0; }");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

out.println("</style>");

out.println("<script src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js\"></script>");

out.println("<script>");

out.println("$(document).ready(function(){");

**for** (**int** i = 0; i < 400; i++) {

out.println(" $(\"#testfailtoggle"+i+"\").click(function(){");

out.println("$(\"#tfailtoggle"+i+"\").toggle(100);");

out.println("});");

}

/\*for (int i=1;i<180;i++)

{

out.println(" $(\"#test"+i+"\").click(function(){");

out.println("$(\"#t"+i+"\").toggle(100);");

out.println("});");

}\*/

out.println("});");

out.println("</script>");

out.println("</head>");

out.println("<body>");

}

/\*\* Finishes HTML stream \*/

**protected** **void** endHtml(PrintWriter out) {

//out.println("<center> Report customized by KiwiQA </center><br/><br/>");

out.println("<tr bgcolor='SkyBlue'><td align='right' colspan='4'><center><b><i>Report customized by KiwiQA </i><b><center></center></b></b></center></td></tr>");

out.println("</body></html>");

}

// ~ Inner Classes --------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**private** **class** TestSorter **implements** Comparator<IInvokedMethod> {

// ~ Methods

// -------------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

@Override

**public** **int** compare(IInvokedMethod o1, IInvokedMethod o2) {

// System.out.println("Comparing " + o1.getMethodName() + " " +

// o1.getDate()

// + " and " + o2.getMethodName() + " " + o2.getDate());

**return** (**int**) (o1.getDate() - o2.getDate());

// int r = ((T) o1).getTestClass().getName().compareTo(((T)

// o2).getTestClass().getName());

// if (r == 0) {

// r = ((T) o1).getMethodName().compareTo(((T) o2).getMethodName());

// }

// return r;

}

}

}

**5.) class name:** CustomReporterTestpassDetails

**package** com.utility;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Scanner;

**import** java.util.Set;

**import** org.testng.IInvokedMethod;

**import** org.testng.IResultMap;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestClass;

**import** org.testng.ITestContext;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.collections.Lists;

**import** org.testng.internal.Utils;

**import** org.testng.log4testng.Logger;

**import** org.testng.xml.XmlSuite;

**public** **class** CustomReporterTestpassDetails **extends** CustomReporterListener {

**private** **static** **final** Logger ***L*** = Logger

.*getLogger*(CustomReporterListener.**class**);

// ~ Instance fields ------------------------------------------------------

**private** PrintWriter m\_out;

**private** **int** m\_row;

**private** Integer m\_testIndex;

**private** **int** m\_methodIndex;

**public** **static** **int** *num*=0;

**public** **static** String *id1*="";

**public** **static** **int** *g*=0;

**private** Scanner scanner;

**int** passCount=0;

**private** **static** HashMap<String, String> *map* = **new** HashMap<String, String>();

**int** namecount = 0;

**int** qty\_tests = 0;

**int** passed = 0;

**int** skipped = 0;

**int** failedcount = 0;

**int** total\_a = 0;

**int** qty\_pass= 0;

**public** **static** ArrayList<String> *passedtoggles* = **new** ArrayList<String>();

// ~ Methods --------------------------------------------------------------

/\*

\* public static void maperrors() {

\*

\* System.out.println(

\* "In Error mapping..............\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3465768487488............"

\* ); map.put(

\* "org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure. "

\* , "Server or Node is not running.");

\* map.put("Timed out after 20 seconds waiting for visibility of Proxy element"

\* ,

\* "Particular Element is not found on page after 20 second waiting can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("Unable to locate the element",

\* "Particular Element is not located on page can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("no such element",

\* "element could not be found. you may want to check : 1.Check your xpath in automation code. 2.Element may not present on the screen."

\* ); map.put("Unable to bind to locking port 7054 within 45000 ms",

\* "Port is already locked by other browser and may not be free.");

\* map.put("Unexpected error launching Internet Explorer",

\* "Unable to launch IE.");

\* map.put("Unable to find element on closed window",

\* "Browser window may closed unexpectedly.");

\* map.put("Error communicating with the remote browser.",

\* "remote browser may have died.");

\* map.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

\* "xpath of the particular element getting changed OR Page is not loaded completely."

\* ); map.put(

\* "Error forwarding the new session Error forwarding the request Connect to"

\* , "Connection may be refused by the node/server."); }

\*/

**public** **void** CustomReporterTestFailureDetails() {

//super();

*map*.put("Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.",

"Server or Node is not running.");

*map*.put("Timed out after 35 seconds waiting for visibility of Proxy element",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

*map*.put("Unable to locate",

"Particular Element is not located on page. Either Page is not loaded completely OR Element is Not found on page");

*map*.put("no such element",

"element could not be found. Check Possibility: 1.Update automation code. 2.Element may not present on the screen. 3.Possibility for \"BUG\"");

*map*.put("Unable to bind to locking port 7054 within 45000 ms",

"Port is already locked by other browser and may not be free.Please restart selenium node and server");

*map*.put("Unexpected error launching Internet Explorer",

"Unable to launch IE.");

*map*.put("Unable to find element on closed window",

"Browser window may closed unexpectedly. This will fix automatically on next run");

*map*.put("Error communicating with the remote browser.",

"remote browser may have died. Please restart selenium node and server");

*map*.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

"xpath of the particular element getting changed OR Page is not loaded completely.");

*map*.put("Error forwarding the new session Error forwarding the request Connect to",

"Connection may be refused by the node/server. Please restart selenium node and server");

*map*.put("element not visible",

"Element is not found on page : 1.Update automation code. 2.Element may not present on the screen.");

*map*.put("Timed out after 35 seconds waiting for visibility of [[AppiumDriver:",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

}

/\*\* Creates summary of the run \*/

@Override

**public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites,

String outdir) { //1

**try** {

m\_out = createWriter(outdir); //2

} **catch** (IOException e) {

***L***.error("output file", e);

**return**;

}

startHtml(m\_out); //3

generateSuiteSummaryReport(suites);

TotalTime(suites); //4

generateMethodSummaryReport(suites); //5

// generateMethodDetailReport(suites);

//endHtml(m\_out); //previous

m\_out.flush();

m\_out.close();

}

String Time;

**public** String TotalTime(List<ISuite> suites) { //4.1

**long** time\_start = Long.***MAX\_VALUE***;

**long** time\_end = Long.***MIN\_VALUE***;

ITestContext overview = **null**;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> itests = suite.getResults();

**for** (ISuiteResult r : itests.values()) {

overview = r.getTestContext();

time\_start = Math.*min*(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.*max*(overview.getEndDate().getTime(), time\_end);

}

}

// m\_out.println("</tr><td class=\"numi\"><center>"+((time\_end -

// time\_start) / 1000.) / 60.+"</center></td> </tr>");

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

Time = String.*valueOf*(formatter

.format(((time\_end - time\_start) / 1000.) / 60.));

**return** Time;

}

**protected** PrintWriter createWriter(String outdir) **throws** IOException { //2.1

// java.util.Date now = new Date();

**new** File(outdir).mkdirs();

**return** **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(**new** File(

outdir, "CustomReporterTestPassedDetails" + ".html"))));

}

/\*\*

\* Creates a table showing the highlights of each test method with links to

\* the method details

\*/

**protected** **void** generateMethodSummaryReport(List<ISuite> suites) { //5.1

m\_methodIndex = 0;

startResultSummaryTable("methodOverview"); //5.2

**int** testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() > 1) {

titleRow(suite.getName(), 5);

}

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

String testName = testContext.getName();

m\_testIndex = testIndex;

//resultSummary\_passed(suite, testContext.getPassedTests()); //5.3

System.***out***.println("Passed---");

resultSummary\_passed(suite, testContext.getPassedTests(), testName,

"passed", "");

System.***out***.println("Failed---");

resultSummary(suite, testContext.getFailedConfigurations(), //5.4

testName, "failed", " (configuration methods)");

resultSummary(suite, testContext.getPassedTests(), testName,

"failed", "");

System.***out***.println("Skipped---");

resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"skipped", "");

/\*

\* resultSummary(suite, testContext.getSkippedConfigurations(),

\* testName, "skipped", " (configuration methods)");

\* resultSummary(suite, testContext.getSkippedTests(), testName,

\* "skipped", ""); resultSummary(suite,

\* testContext.getPassedTests(), testName, "passed", "");

\*/

testIndex++;

}

}

endHtml(m\_out);

//testCaseNo();

m\_out.println("</table>");

}

/\*\* Creates a section showing known results for each method \*/

**protected** **void** generateMethodDetailReport(List<ISuite> suites) {

m\_methodIndex = 0;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

**if** (r.values().size() > 0) {

m\_out.println("<h1>" + testContext.getName() + "</h1>");

}

resultDetail(testContext.getFailedConfigurations());

resultDetail(testContext.getFailedTests());

/\*

\* resultDetail(testContext.getSkippedConfigurations());

\* resultDetail(testContext.getSkippedTests());

\* resultDetail(testContext.getPassedTests());

\*/

}

}

}

**public** **void** testCaseNo() {

// m\_out.println("<td bgcolor='DeepSkyBlue' colspan='4' align='left' height='30px'><h3 style='margin-top:0px;margin-bottom:0px;'> Total Test Cases : "

// + (qty\_tests) + "<br/> Failed Test Cases : "

// + (failedcount/2) + "<br/> Passed Test cases : "

// + passed + "<br/> Skipped Test cases : "

// + skipped + "<br/></h3></td>");

m\_out.println(

"<table width='350px' height='30px' border='1' align='left'><tbody><tr colspan='2'><td bgcolor='#0088cc' colspan='2'><h3><center><font color='white'>Build Summary</font></center></h3></td></tr><tr><td><b>"

+ "Passed Test cases</b> </td> <td> <center><b>"

+ passed + "</b></center></td></tr><tr><td><b> Failed Test Cases </b></td><td> <center><b>" + (failedcount)

+ "</b></center></td></tr> <tr><td><b>Skipped Test cases</b> </td><td><center><b> " + skipped

+ "</b></center> </td></tr><tr bgcolor='skyblue'><td> <b>Total Test Cases </b> </td><td> <center><b>" + qty\_tests

+ "</b></center></td></tr></tbody></table>");

}

/\*private void resultSummary\_passed(ISuite suite, IResultMap tests) { //5.3.1

System.out.println("passed count : " + passed);

//passCount= passed;

for (ITestNGMethod method : getMethodSet(tests, suite)) {

passed++;

}

}\*/

**private** **void** resultSummary\_total(ISuite suite, IResultMap tests) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

total\_a++;

}

}

// private void resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

// String style, String details) { //5.4.1

//

// if (tests.getAllResults().size() > 0) {

//

// StringBuffer buff = new StringBuffer();

// String lastClassName = "";

// int mq = 0;

// int cq = 0;

// for (ITestNGMethod method : getMethodSet(tests, suite)) {

// ++passed;

// }

// }

// }

**private** **void** resultSummary\_skipped(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

skipped++;

}

}

}

ArrayList<String> PassedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

**if**(!checkpassedTestCases(testname))

{System.***out***.println("Name and Count:"+testname+"=="+passed);

PassedTestCases.add(testname);

++passed;

}

}

}

}

ArrayList<String> PassedTestName = **new** ArrayList<String>();

**public** **boolean** checkpassedTestCases(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\*

\* **@param** tests

\*/

ArrayList<String> testArray = **new** ArrayList<String>();

**int** retry = 0;

**private** **void** resultSummary(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCases(testname) && isPassed(testname))

{

testArray.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

{

String id = (m\_testIndex == **null** ? **null** : "testpasstoggle"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

*num*=m\_testIndex;

m\_out.print("<tr");

**if** (id != **null**) {

//m\_out.print(" id=\"" + id + "\"");

*id1*=id;

}

m\_out.println("><td width='25%' style=\"font-size:14px; font-family:Times New Roman;\">" + testname + "</td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

failedcount++; //perivious

}

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

getShortException(tests,*num*,*id1*);

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

**public** **boolean** checkTestCases(String testName)

{

**return** testArray.contains(testName);

}

**public** **boolean** isPassed(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\* Starts and defines columns result summary table \*/

**private** **void** startResultSummaryTable(String style) { //5.2.1

tableStart(style, "summary");

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm a z,MM/dd/yyyy");

/\*m\_out.println("<tr><td bgcolor='white' colspan='4'> <table border='0' width='100%' bgcolor='#e6f7ff'><tr>"

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.kiwiqa.com/wp-content/themes/twentythirteen/images/logo.png'/></center>"

+ "</td><td ><center><font color='#008bcc'><b><h1>Passed Test Cases Analysis</h1></b></font></center></td> "

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.genixventures.com/wp-content/uploads/2015/05/genix\_logo\_03.png'/></center></td> "

+ "</tr></table> </td></tr>");\*/

//m\_out.println("<tr><td colspan='4'>To view Full Report : <a href=\"http://localhost:8080/job/Videogram/HTML\_Report/\">http://localhost:8080/job/Videogram\_Chrome/HTML\_Report</a></td></tr>");

/\*m\_out.println("<tr><td colspan='4'>Overall test suite completion : <b>"

+ Time + " minutes</b><br/> Date and Time of Run: <b>"

+ sdf.format(date) + "</b><br/> Browser : <b>"+SeleniumInit.browsernm+"<t></t>"

+ SeleniumInit.browserVersion + "</b><br/>OS: <b>"

+ System.getProperty("os.name") + "</b></td></tr>");\*/

m\_out.println("<tr bgcolor='SkyBlue'><th>Test Cases</th><th>Steps</th>"

+ /\*"<th>Failure Reason</th>"\*/"<th>Total Time<br/>(sec.)</th>");

m\_row = 0;

}

**private** String qualifiedName(ITestNGMethod method) {

StringBuilder addon = **new** StringBuilder();

String[] groups = method.getGroups();

**int** length = groups.length;

**if** (length > 0 && !"basic".equalsIgnoreCase(groups[0])) {

addon.append("(");

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

**if** (i > 0) {

addon.append(", ");

}

addon.append(groups[i]);

}

addon.append(")");

}

**return** "<b>" + method.getMethodName() + "</b> " + addon;

}

**private** **void** resultDetail(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

ITestNGMethod method = result.getMethod();

m\_methodIndex++;

String cname = method.getTestClass().getName();

m\_out.println("<h2 id=\"m" + m\_methodIndex + "\">" + cname + ":"

+ method.getMethodName() + "</h2>");

Set<ITestResult> resultSet = tests.getResults(method);

generateForResult(result, method, resultSet.size());

m\_out.println("<p class=\"totop\"><a href=\"#summary\">back to summary</a></p>");

}

}

/\*\*

\* Write the first line of the stack trace

\*

\* **@param** tests

\*/

**private** **void** getShortException(IResultMap tests, **int** num, String id) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

//Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

//String str = Utils.stackTrace(exception, true)[0];

//scanner = new Scanner(str);

//String firstLine = scanner.nextLine();

String toggle = "passtoggle"+num;

*passedtoggles*.add(toggle);

System.***out***.println("jhgudfhf = "+*passedtoggles*);

m\_out.println("<td width='50%'");

**for** (String line : msgs) {

**if**(*g*==1)

{

m\_out.println(" style=\"background-color:#afefa5\"><b id=\""+id+"\">");

m\_out.println("<a href=\"#hide"+toggle+"\" data-toggle=\"tooltip\" title=\"Click here to see list of steps\" class=\"hide\" id=\"hide"+toggle+"\">+</a>");

m\_out.println("<a href=\"#show"+toggle+"\" class=\"show\" id=\"show"+toggle+"\">-</a></b> &nbsp;&nbsp;Click here to see list of steps</br></br>");

m\_out.println("<table id=\"t"+toggle+"\" style=\"display: none;\"><tr><td>");

}

**else**

{

m\_out.println(line+"");

}

*g*++;

**if**(msgs.size()==*g*)

m\_out.println("</td></tr></table></td>");

}

**if**(*g*==0||*g*==1){

m\_out.println("style=\"background-color:#fcf77a\"><b><font color='Blue'>Skipped</font></b></br>");

//m\_out.println(firstLine);

}

m\_out.println("</td>");

*g*=0;

//boolean hasThrowable = exception != null;

/\* if (hasThrowable) {

m\_out.println("<td width='15%'>");

for (Entry<String, String> e : map.entrySet()) {

if (firstLine.contains(e.getKey())) {

// m\_out.print(map.get(str));

// m\_out.print("contains <br/>");

m\_out.print(e.getValue() + "<br/>");

} else {

// m\_out.print("Not contains <br/>");

// m\_out.print(str+"<br/>");

}

}

m\_out.println("</td>");

\* if(map.containsKey(str)) { m\_out.print(map.get(str)); }else{

\* m\_out.print("Not contains"); m\_out.print(str); }

m\_out.println("<td width='15%'>");

boolean wantsMinimalOutput = result.getStatus() == ITestResult.SUCCESS;

if (hasReporterOutput) {

m\_out.print("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

// Getting first line of the stack trace

m\_out.println(firstLine);

m\_out.println("</td>");

}\*/

}

}

/\*\*

\* Write all parameters

\*

\* **@param** tests

\*/

**private** **void** getParameters(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Object[] parameters = result.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

**for** (Object p : parameters) {

m\_out.println(Utils.*escapeHtml*(Utils.*toString*(p)) + " | ");

}

}

}

}

**private** **void** generateForResult(ITestResult ans, ITestNGMethod method,

**int** resultSetSize) {

Object[] parameters = ans.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

//tableStart("result", null);

//m\_out.print("<tr class=\"param\">");

**for** (**int** x = 1; x <= parameters.length; x++) {

//m\_out.print("<th>Param." + x + "</th>");

}

/\*m\_out.println("</tr>");

m\_out.print("<tr class=\"param stripe\">");\*/

**for** (Object p : parameters) {

/\*m\_out.println("<td>" + Utils.escapeHtml(Utils.toString(p))

+ "</td>");\*/

}

//m\_out.println("</tr>");

}

List<String> msgs = Reporter.*getOutput*(ans);

**boolean** hasReporterOutput = msgs.size() > 0;

Throwable exception = ans.getThrowable();

**boolean** hasThrowable = exception != **null**;

**if** (hasReporterOutput || hasThrowable) {

**if** (hasParameters) {

//m\_out.print("<tr><td");

**if** (parameters.length > 1) {

// m\_out.print(" colspan=\"" + parameters.length + "\"");

}

//m\_out.println(">");

} **else** {

// m\_out.println("<div>");

}

**if** (hasReporterOutput) {

**if** (hasThrowable) {

//m\_out.println("<h3>Test Messages</h3>");

}

**for** (String line : msgs) {

//m\_out.println(line + "<br/>");

}

}

**if** (hasThrowable) {

**boolean** wantsMinimalOutput = ans.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

/\*m\_out.println("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");\*/

}

//generateExceptionReport(exception, method);

}

**if** (hasParameters) {

//m\_out.println("</td></tr>");

} **else** {

//m\_out.println("</div>");

}

}

**if** (hasParameters) {

//m\_out.println("</table>");

}

}

**protected** **void** generateExceptionReport(Throwable exception,

ITestNGMethod method) {

m\_out.print("<div class=\"stacktrace\">");

m\_out.print(Utils.*stackTrace*(exception, **true**)[0]);

m\_out.println("</div>");

}

/\*\*

\* Since the methods will be sorted chronologically, we want to return the

\* ITestNGMethod from the invoked methods.

\*/

**private** Collection<ITestNGMethod> getMethodSet(IResultMap tests,

ISuite suite) {

List<IInvokedMethod> r = Lists.*newArrayList*();

List<IInvokedMethod> invokedMethods = suite.getAllInvokedMethods();

**for** (IInvokedMethod im : invokedMethods) {

**if** (tests.getAllMethods().contains(im.getTestMethod())) {

r.add(im);

}

}

Arrays.*sort*(r.toArray(**new** IInvokedMethod[r.size()]), **new** TestSorter());

List<ITestNGMethod> result = Lists.*newArrayList*();

// Add all the invoked methods

**for** (IInvokedMethod m : r) {

result.add(m.getTestMethod());

}

// Add all the methods that weren't invoked (e.g. skipped) that we

// haven't added yet

**for** (ITestNGMethod m : tests.getAllMethods()) {

**if** (!result.contains(m)) {

result.add(m);

}

}

**return** result;

}

@SuppressWarnings("unused")

**public** **void** generateSuiteSummaryReport(List<ISuite> suites) {

/\*tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Browser");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time(hh:mm:ss)");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups");

m\_out.println("</tr>");\*/

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

**int** qty\_pass\_m = 0;

**int** qty\_pass\_s = 0;

**int** qty\_skip = 0;

**long** time\_start = Long.***MAX\_VALUE***;

**int** qty\_fail = 0;

**long** time\_end = Long.***MIN\_VALUE***;

m\_testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() >= 1) {

//titleRow(suite.getName(), 10);

}

Map<String, ISuiteResult> tests = suite.getResults();

**for** (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

//startSummaryRow(overview.getName());

**int** q = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass\_m += q;

System.***err***.println("aa----->"+qty\_tests);

}

}

}

**private** **void** summaryCell(String[] val) {

StringBuffer b = **new** StringBuffer();

**for** (String v : val) {

b.append(v + " ");

}

summaryCell(b.toString(), **true**);

}

**private** **void** summaryCell(String v, **boolean** isgood) {

m\_out.print("<td class=\"numi" + (isgood ? "" : "\_attn") + "\">" + v

+ "</td>");

}

**private** **void** startSummaryRow(String label) {

m\_row += 1;

m\_out.print("<tr"

+ (m\_row % 2 == 0 ? " class=\"stripe\"" : "")

+ "><td style=\"text-align:left;padding-right:2em\"><a href=\"#t"

+ m\_testIndex + "\">" + label + "</a>" + "</td>");

}

**private** **void** summaryCell(**int** v, **int** maxexpected) {

summaryCell(String.*valueOf*(v), v <= maxexpected);

}

**private** **void** tableStart(String cssclass, String id) {

m\_out.println("<table width='100%' border=\"5\" cellspacing=\"0\" cellpadding=\"0\""

+ (cssclass != **null** ? " class=\"" + cssclass + "\"" : " ")

+ (id != **null** ? " id=\"" + id + "\"" : "") + ">");

m\_row = 0;

}

**private** **void** tableColumnStart(String label) {

m\_out.print("<th>" + label + "</th>");

}

**private** **void** titleRow(String label, **int** cq) {

titleRow(label, cq, **null**);

}

**private** **void** titleRow(String label, **int** cq, String id) {

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><th bgcolor='#cce6ff' colspan=\"" + cq + "\"><font color='black' style='text-shadow:2px 2px white;'>" + label + "<font></th></tr>");

m\_row = 0;

}

/\*\* Starts HTML stream \*/

**protected** **void** startHtml(PrintWriter out) { //3.1

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<title> Automation build Summary - TestNG Report</title>");

out.println("<style type=\"text/css\">");

out.println("table {margin-bottom:1px;border-collapse:collapse;empty-cells:show}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println(".result th {vertical-align:bottom}");

out.println(".param th {padding-left:1em;padding-right:1em}");

out.println(".param td {padding-left:.5em;padding-right:2em}");

out.println(".stripe td,.stripe th {background-color: #E6EBF9}");

out.println(".numi,.numi\_attn {text-align:right}");

out.println(".total td {font-weight:bold}");

out.println(".passedodd td {background-color: #0A0}");

out.println(".passedeven td {background-color: #3F3}");

out.println(".skippedodd td {background-color: #CCC}");

out.println(".skippedodd td {background-color: #DDD}");

out.println(".failedodd td,.numi\_attn {background-color: #F9C1C1}");

out.println(".failedeven td,.stripe .numi\_attn {background-color: #F9C1C1}");

out.println(".stacktrace {white-space:pre;}");

out.println(".totop {font-size:85%;text-align:center;border-bottom:2px solid #000}");

out.println("html \* {");

out.println(" font-family: \"Open Sans\",sans-serif; font-size:11px;}");

out.println("h1 { font-size:25px; }");

out.println("th {font-size:14px; }");

/\*\*\*Collapse expands\*\*\*\*/

out.println(".list { display:none;");

out.println("height:auto;");

out.println(" margin:0;");

out.println("float: left; }");

out.println(".show {");

out.println("display: none; }");

out.println(".hide:target + .show {");

out.println("display: inline; }");

out.println(".hide:target {");

out.println("display: none; }");

out.println(".hide:target ~ .list {");

out.println("display:inline; }");

/\*style the (+) and (-) \*/

out.println(".hide, .show {");

out.println("width: 16px;");

out.println("height: 16px;");

out.println("border-radius: 30px;");

out.println("font-size: 15px;");

out.println("color: #000;");

out.println("text-shadow: 0 1px 0 #666;");

out.println("text-align: center;");

out.println("text-decoration: none;");

out.println("box-shadow: 1px 1px 2px #000;");

out.println("background: #91DDFE;");

out.println("opacity: .95;");

out.println("margin-right: 0;");

out.println("float: left;");

out.println("margin-bottom: 25px; }");

out.println(".hide:hover, .show:hover {");

out.println("color: #eee;");

out.println("text-shadow: 0 0 1px #666;");

out.println("text-decoration: none;");

out.println("box-shadow: 0 0 4px #222 inset;");

out.println("opacity: 1;");

out.println("margin-bottom: 25px; }");

out.println(".list tr{");

out.println("height:auto;");

out.println("margin:0; }");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

out.println("</style>");

out.println("<script src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js\"></script>");

out.println("<script>");

out.println("$(document).ready(function(){");

**for** (**int** i = 0; i < 400; i++) {

out.println(" $(\"#testpasstoggle"+i+"\").click(function(){");

out.println("$(\"#tpasstoggle"+i+"\").toggle(100);");

out.println("});");

}

/\*for (int i=1;i<180;i++)

{

out.println(" $(\"#test"+i+"\").click(function(){");

out.println("$(\"#t"+i+"\").toggle(100);");

out.println("});");

}\*/

out.println("});");

out.println("</script>");

out.println("</head>");

out.println("<body>");

}

/\*\* Finishes HTML stream \*/

**protected** **void** endHtml(PrintWriter out) {

//out.println("<center> Report customized by KiwiQA </center><br/><br/>");

out.println("<tr bgcolor='SkyBlue'><td align='right' colspan='4'><center><b><i>Report customized by KiwiQA </i><b><center></center></b></b></center></td></tr>");

out.println("</body></html>");

}

// ~ Inner Classes --------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**private** **class** TestSorter **implements** Comparator<IInvokedMethod> {

// ~ Methods

// -------------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

@Override

**public** **int** compare(IInvokedMethod o1, IInvokedMethod o2) {

// System.out.println("Comparing " + o1.getMethodName() + " " +

// o1.getDate()

// + " and " + o2.getMethodName() + " " + o2.getDate());

**return** (**int**) (o1.getDate() - o2.getDate());

// int r = ((T) o1).getTestClass().getName().compareTo(((T)

// o2).getTestClass().getName());

// if (r == 0) {

// r = ((T) o1).getMethodName().compareTo(((T) o2).getMethodName());

// }

// return r;

}

}

}

**6.) class name :** CustomReporterTestskipDetails

**package** com.utility;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.text.DecimalFormat;

**import** java.text.NumberFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Scanner;

**import** java.util.Set;

**import** org.testng.IInvokedMethod;

**import** org.testng.IResultMap;

**import** org.testng.ISuite;

**import** org.testng.ISuiteResult;

**import** org.testng.ITestClass;

**import** org.testng.ITestContext;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**import** org.testng.Reporter;

**import** org.testng.collections.Lists;

**import** org.testng.internal.Utils;

**import** org.testng.log4testng.Logger;

**import** org.testng.xml.XmlSuite;

**public** **class** CustomReporterTestskipDetails **extends** CustomReporterListener {

**private** **static** **final** Logger ***L*** = Logger

.*getLogger*(CustomReporterListener.**class**);

// ~ Instance fields ------------------------------------------------------

**private** PrintWriter m\_out;

**private** **int** m\_row;

**private** Integer m\_testIndex;

**private** **int** m\_methodIndex;

**public** **static** **int** *num*=0;

**public** **static** String *id1*="";

**public** **static** **int** *g*=0;

**private** Scanner scanner;

**int** passCount=0;

**private** **static** HashMap<String, String> *map* = **new** HashMap<String, String>();

**int** namecount = 0;

**int** qty\_tests = 0;

**int** passed = 0;

**int** skipped = 0;

**int** failedcount = 0;

**int** total\_a = 0;

**int** qty\_pass= 0;

**public** **static** ArrayList<String> *skiptoggles* = **new** ArrayList<String>();

// ~ Methods --------------------------------------------------------------

/\*

\* public static void maperrors() {

\*

\* System.out.println(

\* "In Error mapping..............\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3465768487488............"

\* ); map.put(

\* "org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure. "

\* , "Server or Node is not running.");

\* map.put("Timed out after 20 seconds waiting for visibility of Proxy element"

\* ,

\* "Particular Element is not found on page after 20 second waiting can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("Unable to locate the element",

\* "Particular Element is not located on page can be due to Page is not loaded completely OR Element is Not found on page"

\* ); map.put("no such element",

\* "element could not be found. you may want to check : 1.Check your xpath in automation code. 2.Element may not present on the screen."

\* ); map.put("Unable to bind to locking port 7054 within 45000 ms",

\* "Port is already locked by other browser and may not be free.");

\* map.put("Unexpected error launching Internet Explorer",

\* "Unable to launch IE.");

\* map.put("Unable to find element on closed window",

\* "Browser window may closed unexpectedly.");

\* map.put("Error communicating with the remote browser.",

\* "remote browser may have died.");

\* map.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

\* "xpath of the particular element getting changed OR Page is not loaded completely."

\* ); map.put(

\* "Error forwarding the new session Error forwarding the request Connect to"

\* , "Connection may be refused by the node/server."); }

\*/

**public** **void** CustomReporterTestFailureDetails() {

//super();

*map*.put("Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.",

"Server or Node is not running.");

*map*.put("Timed out after 35 seconds waiting for visibility of Proxy element",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

*map*.put("Unable to locate",

"Particular Element is not located on page. Either Page is not loaded completely OR Element is Not found on page");

*map*.put("no such element",

"element could not be found. Check Possibility: 1.Update automation code. 2.Element may not present on the screen. 3.Possibility for \"BUG\"");

*map*.put("Unable to bind to locking port 7054 within 45000 ms",

"Port is already locked by other browser and may not be free.Please restart selenium node and server");

*map*.put("Unexpected error launching Internet Explorer",

"Unable to launch IE.");

*map*.put("Unable to find element on closed window",

"Browser window may closed unexpectedly. This will fix automatically on next run");

*map*.put("Error communicating with the remote browser.",

"remote browser may have died. Please restart selenium node and server");

*map*.put("Unable to locate element: {\"method\":\"xpath\",\"selector\":",

"xpath of the particular element getting changed OR Page is not loaded completely.");

*map*.put("Error forwarding the new session Error forwarding the request Connect to",

"Connection may be refused by the node/server. Please restart selenium node and server");

*map*.put("element not visible",

"Element is not found on page : 1.Update automation code. 2.Element may not present on the screen.");

*map*.put("Timed out after 35 seconds waiting for visibility of [[AppiumDriver:",

"Particular Element is not located on page. 1.Page is not loaded completely 2. Element is Not found on page 3. Possibility for \"BUG\"");

}

/\*\* Creates summary of the run \*/

@Override

**public** **void** generateReport(List<XmlSuite> xml, List<ISuite> suites,

String outdir) { //1

**try** {

m\_out = createWriter(outdir); //2

} **catch** (IOException e) {

***L***.error("output file", e);

**return**;

}

startHtml(m\_out); //3

generateSuiteSummaryReport(suites);

TotalTime(suites); //4

generateMethodSummaryReport(suites); //5

// generateMethodDetailReport(suites);

//endHtml(m\_out); //previous

m\_out.flush();

m\_out.close();

}

String Time;

**public** String TotalTime(List<ISuite> suites) { //4.1

**long** time\_start = Long.***MAX\_VALUE***;

**long** time\_end = Long.***MIN\_VALUE***;

ITestContext overview = **null**;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> itests = suite.getResults();

**for** (ISuiteResult r : itests.values()) {

overview = r.getTestContext();

time\_start = Math.*min*(overview.getStartDate().getTime(),

time\_start);

time\_end = Math.*max*(overview.getEndDate().getTime(), time\_end);

}

}

// m\_out.println("</tr><td class=\"numi\"><center>"+((time\_end -

// time\_start) / 1000.) / 60.+"</center></td> </tr>");

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

Time = String.*valueOf*(formatter

.format(((time\_end - time\_start) / 1000.) / 60.));

**return** Time;

}

**protected** PrintWriter createWriter(String outdir) **throws** IOException { //2.1

// java.util.Date now = new Date();

**new** File(outdir).mkdirs();

**return** **new** PrintWriter(**new** BufferedWriter(**new** FileWriter(**new** File(

outdir, "CustomReporterTestSkippedDetails" + ".html"))));

}

/\*\*

\* Creates a table showing the highlights of each test method with links to

\* the method details

\*/

**protected** **void** generateMethodSummaryReport(List<ISuite> suites) { //5.1

m\_methodIndex = 0;

startResultSummaryTable("methodOverview"); //5.2

**int** testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() > 1) {

titleRow(suite.getName(), 5);

}

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

String testName = testContext.getName();

m\_testIndex = testIndex;

//resultSummary\_passed(suite, testContext.getPassedTests()); //5.3

System.***out***.println("Passed---");

/\*resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"passed", "");\*/

System.***out***.println("Failed---");

/\* resultSummary(suite, testContext.getFailedConfigurations(), //5.4

testName, "failed", " (configuration methods)");\*/

resultSummary(suite, testContext.getSkippedTests(), testName,

"skipped", "");

System.***out***.println("Skipped---");

/\*resultSummary\_skipped(suite, testContext.getSkippedTests(), testName,

"skipped", "");\*/

/\*

\* resultSummary(suite, testContext.getSkippedConfigurations(),

\* testName, "skipped", " (configuration methods)");

\* resultSummary(suite, testContext.getSkippedTests(), testName,

\* "skipped", ""); resultSummary(suite,

\* testContext.getPassedTests(), testName, "passed", "");

\*/

testIndex++;

}

}

endHtml(m\_out);

//testCaseNo();

m\_out.println("</table>");

}

/\*\* Creates a section showing known results for each method \*/

**protected** **void** generateMethodDetailReport(List<ISuite> suites) {

m\_methodIndex = 0;

**for** (ISuite suite : suites) {

Map<String, ISuiteResult> r = suite.getResults();

**for** (ISuiteResult r2 : r.values()) {

ITestContext testContext = r2.getTestContext();

**if** (r.values().size() > 0) {

m\_out.println("<h1>" + testContext.getName() + "</h1>");

}

resultDetail(testContext.getFailedConfigurations());

resultDetail(testContext.getFailedTests());

/\*

\* resultDetail(testContext.getSkippedConfigurations());

\* resultDetail(testContext.getSkippedTests());

\* resultDetail(testContext.getPassedTests());

\*/

}

}

}

**public** **void** testCaseNo() {

// m\_out.println("<td bgcolor='DeepSkyBlue' colspan='4' align='left' height='30px'><h3 style='margin-top:0px;margin-bottom:0px;'> Total Test Cases : "

// + (qty\_tests) + "<br/> Failed Test Cases : "

// + (failedcount/2) + "<br/> Passed Test cases : "

// + passed + "<br/> Skipped Test cases : "

// + skipped + "<br/></h3></td>");

m\_out.println(

"<table width='350px' height='30px' border='1' align='left'><tbody><tr colspan='2'><td bgcolor='#0088cc' colspan='2'><h3><center><font color='white'>Build Summary</font></center></h3></td></tr><tr><td><b>"

+ "Passed Test cases</b> </td> <td> <center><b>"

+ passed + "</b></center></td></tr><tr><td><b> Failed Test Cases </b></td><td> <center><b>" + (failedcount)

+ "</b></center></td></tr> <tr><td><b>Skipped Test cases</b> </td><td><center><b> " + skipped

+ "</b></center> </td></tr><tr bgcolor='skyblue'><td> <b>Total Test Cases </b> </td><td> <center><b>" + qty\_tests

+ "</b></center></td></tr></tbody></table>");

}

/\*private void resultSummary\_passed(ISuite suite, IResultMap tests) { //5.3.1

System.out.println("passed count : " + passed);

//passCount= passed;

for (ITestNGMethod method : getMethodSet(tests, suite)) {

passed++;

}

}\*/

**private** **void** resultSummary\_total(ISuite suite, IResultMap tests) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

total\_a++;

}

}

// private void resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

// String style, String details) { //5.4.1

//

// if (tests.getAllResults().size() > 0) {

//

// StringBuffer buff = new StringBuffer();

// String lastClassName = "";

// int mq = 0;

// int cq = 0;

// for (ITestNGMethod method : getMethodSet(tests, suite)) {

// ++passed;

// }

// }

// }

**private** **void** resultSummary\_skipped(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

skipped++;

}

}

}

ArrayList<String> PassedTestCases = **new** ArrayList<String>();

**private** **void** resultSummary\_passed(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

**if**(!checkpassedTestCases(testname))

{System.***out***.println("Name and Count:"+testname+"=="+passed);

PassedTestCases.add(testname);

++passed;

}

}

}

}

ArrayList<String> PassedTestName = **new** ArrayList<String>();

**public** **boolean** checkpassedTestCases(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\*

\* **@param** tests

\*/

ArrayList<String> testArray = **new** ArrayList<String>();

**int** retry = 0;

**private** **void** resultSummary(ISuite suite, IResultMap tests, String testname,

String style, String details) { //5.4.1

**if** (tests.getAllResults().size() > 0) {

StringBuffer buff = **new** StringBuffer();

String lastClassName = "";

**int** mq = 0;

**int** cq = 0;

**for** (ITestNGMethod method : getMethodSet(tests, suite)) {

//failedcount++;

**if**(!checkTestCases(testname) /\*&& !isPassed(testname)\*/)

{

testArray.add(testname);

m\_row += 1;

m\_methodIndex += 1;

ITestClass testClass = method.getTestClass();

String className = testClass.getName();

// if (mq == 0)

{

String id = (m\_testIndex == **null** ? **null** : "testskiptoggle"

+ Integer.*toString*(m\_testIndex));

// titleRow(testname + " &#8212; " + style + details, 5,

// id);

//

*num*=m\_testIndex;

m\_out.print("<tr");

**if** (id != **null**) {

//m\_out.print(" id=\"" + id + "\"");

*id1*=id;

}

m\_out.println("><td width='25%' style=\"font-size:14px; font-family:Times New Roman;\">" + testname + "</td>");

m\_row = 0;

//

m\_testIndex = **null**;

namecount++;

failedcount++; //perivious

}

/\*

\* if (!className.equalsIgnoreCase(lastClassName)) { if (mq > 0)

\* { cq += 1; m\_out.print("<tr class=\"" + style + (cq % 2 == 0

\* ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff);

\*

\* } mq = 0; buff.setLength(0); lastClassName = className; }

\*/

Set<ITestResult> resultSet = tests.getResults(method);

**long** end = Long.***MIN\_VALUE***;

**long** start = Long.***MAX\_VALUE***;

**for** (ITestResult testResult : tests.getResults(method)) {

**if** (testResult.getEndMillis() > end) {

end = testResult.getEndMillis();

}

**if** (testResult.getStartMillis() < start) {

start = testResult.getStartMillis();

}

}

mq += 1;

**if** (mq > 1) {

/\*

\* buff.append("<tr class=\"" + style + (cq % 2 == 0 ? "odd"

\* : "even") + "\">");

\*/

}

**if** (mq > 0) {

cq += 1;

/\*

\* m\_out.print("<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); }

\* m\_out.println(">" + lastClassName + "</td>");

\*/

getShortException(tests,*num*,*id1*);

}

String description = method.getDescription();

String testInstanceName = resultSet

.toArray(**new** ITestResult[] {})[0].getTestName();

/\*

\* buff.append("<td class=\"numi\"><center>" + (end -

\* start)/1000 + "</center></td>" + "</tr>" );

\*/

m\_out.println("<td width='5%' class=\"numi\"><center>" + (end - start)

/ 1000 + "</center></td>" + "");

}//to check test name

}

/\*

\* if (mq > 0) { cq += 1; m\_out.print("<tr class=\"" + style + (cq %

\* 2 == 0 ? "even" : "odd") + "\">" + "<td"); if (mq > 1) {

\* m\_out.print(" rowspan=\"" + mq + "\""); } m\_out.println(">" +

\* lastClassName + "</td>" + buff); }

\*/

//

}

System.***out***.println("Test Cases No. : " + namecount);

}

**public** **boolean** checkTestCases(String testName)

{

**return** testArray.contains(testName);

}

**public** **boolean** isPassed(String testName)

{

**return** PassedTestCases.contains(testName);

}

/\*\* Starts and defines columns result summary table \*/

**private** **void** startResultSummaryTable(String style) { //5.2.1

tableStart(style, "summary");

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm a z,MM/dd/yyyy");

/\*m\_out.println("<tr><td bgcolor='white' colspan='4'> <table border='0' width='100%' bgcolor='#e6f7ff'><tr>"

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.kiwiqa.com/wp-content/themes/twentythirteen/images/logo.png'/></center>"

+ "</td><td ><center><font color='#008bcc'><b><h1>Passed Test Cases Analysis</h1></b></font></center></td> "

+ "<td width='25%' bgcolor='white'>"

+ "<center><img width='150px' src='http://www.genixventures.com/wp-content/uploads/2015/05/genix\_logo\_03.png'/></center></td> "

+ "</tr></table> </td></tr>");\*/

//m\_out.println("<tr><td colspan='4'>To view Full Report : <a href=\"http://localhost:8080/job/Videogram/HTML\_Report/\">http://localhost:8080/job/Videogram\_Chrome/HTML\_Report</a></td></tr>");

/\*m\_out.println("<tr><td colspan='4'>Overall test suite completion : <b>"

+ Time + " minutes</b><br/> Date and Time of Run: <b>"

+ sdf.format(date) + "</b><br/> Browser : <b>"+SeleniumInit.browsernm+"<t></t>"

+ SeleniumInit.browserVersion + "</b><br/>OS: <b>"

+ System.getProperty("os.name") + "</b></td></tr>");\*/

m\_out.println("<tr bgcolor='SkyBlue'><th>Test Cases</th><th>Steps</th>"

+ /\*"<th>Failure Reason</th>"\*/"<th>Total Time<br/>(sec.)</th>");

m\_row = 0;

}

**private** String qualifiedName(ITestNGMethod method) {

StringBuilder addon = **new** StringBuilder();

String[] groups = method.getGroups();

**int** length = groups.length;

**if** (length > 0 && !"basic".equalsIgnoreCase(groups[0])) {

addon.append("(");

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

**if** (i > 0) {

addon.append(", ");

}

addon.append(groups[i]);

}

addon.append(")");

}

**return** "<b>" + method.getMethodName() + "</b> " + addon;

}

**private** **void** resultDetail(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

ITestNGMethod method = result.getMethod();

m\_methodIndex++;

String cname = method.getTestClass().getName();

m\_out.println("<h2 id=\"m" + m\_methodIndex + "\">" + cname + ":"

+ method.getMethodName() + "</h2>");

Set<ITestResult> resultSet = tests.getResults(method);

generateForResult(result, method, resultSet.size());

m\_out.println("<p class=\"totop\"><a href=\"#summary\">back to summary</a></p>");

}

}

/\*\*

\* Write the first line of the stack trace

\*

\* **@param** tests

\*/

**private** **void** getShortException(IResultMap tests, **int** num, String id) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

//Throwable exception = result.getThrowable();

List<String> msgs = Reporter.*getOutput*(result);

**boolean** hasReporterOutput = msgs.size() > 0;

//String str = Utils.stackTrace(exception, true)[0];

//scanner = new Scanner(str);

//String firstLine = scanner.nextLine();

String toggle = "skiptoggle"+num;

*skiptoggles*.add(toggle);

m\_out.println("<td width='50%'");

**for** (String line : msgs) {

**if**(*g*==1)

{

m\_out.println(" style=\"background-color:#fcf77a\"><b id=\""+id+"\">");

m\_out.println("<a href=\"#hide"+toggle+"\" data-toggle=\"tooltip\" title=\"Click here to see list of steps\" class=\"hide\" id=\"hide"+toggle+"\">+</a>");

m\_out.println("<a href=\"#show"+toggle+"\" class=\"show\" id=\"show"+toggle+"\">-</a></b> &nbsp;&nbsp;Click here to see list of steps</br></br>");

m\_out.println("<table id=\"t"+toggle+"\" style=\"display: none;\"><tr><td>");

}

**else**

{

m\_out.println(line+"");

}

*g*++;

**if**(msgs.size()==*g*)

m\_out.println("</td></tr></table></td>");

}

**if**(*g*==0||*g*==1){

m\_out.println("style=\"background-color:#fcf77a\"><b><font color='Blue'>Skipped</font></b></br>");

//m\_out.println(firstLine);

}

m\_out.println("</td>");

*g*=0;

//boolean hasThrowable = exception != null;

/\* if (hasThrowable) {

m\_out.println("<td width='15%'>");

for (Entry<String, String> e : map.entrySet()) {

if (firstLine.contains(e.getKey())) {

// m\_out.print(map.get(str));

// m\_out.print("contains <br/>");

m\_out.print(e.getValue() + "<br/>");

} else {

// m\_out.print("Not contains <br/>");

// m\_out.print(str+"<br/>");

}

}

m\_out.println("</td>");

\* if(map.containsKey(str)) { m\_out.print(map.get(str)); }else{

\* m\_out.print("Not contains"); m\_out.print(str); }

m\_out.println("<td width='15%'>");

boolean wantsMinimalOutput = result.getStatus() == ITestResult.SUCCESS;

if (hasReporterOutput) {

m\_out.print("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");

}

// Getting first line of the stack trace

m\_out.println(firstLine);

m\_out.println("</td>");

}\*/

}

}

/\*\*

\* Write all parameters

\*

\* **@param** tests

\*/

**private** **void** getParameters(IResultMap tests) {

**for** (ITestResult result : tests.getAllResults()) {

m\_methodIndex++;

Object[] parameters = result.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

**for** (Object p : parameters) {

m\_out.println(Utils.*escapeHtml*(Utils.*toString*(p)) + " | ");

}

}

}

}

**private** **void** generateForResult(ITestResult ans, ITestNGMethod method,

**int** resultSetSize) {

Object[] parameters = ans.getParameters();

**boolean** hasParameters = parameters != **null** && parameters.length > 0;

**if** (hasParameters) {

//tableStart("result", null);

//m\_out.print("<tr class=\"param\">");

**for** (**int** x = 1; x <= parameters.length; x++) {

//m\_out.print("<th>Param." + x + "</th>");

}

/\*m\_out.println("</tr>");

m\_out.print("<tr class=\"param stripe\">");\*/

**for** (Object p : parameters) {

/\*m\_out.println("<td>" + Utils.escapeHtml(Utils.toString(p))

+ "</td>");\*/

}

//m\_out.println("</tr>");

}

List<String> msgs = Reporter.*getOutput*(ans);

**boolean** hasReporterOutput = msgs.size() > 0;

Throwable exception = ans.getThrowable();

**boolean** hasThrowable = exception != **null**;

**if** (hasReporterOutput || hasThrowable) {

**if** (hasParameters) {

//m\_out.print("<tr><td");

**if** (parameters.length > 1) {

// m\_out.print(" colspan=\"" + parameters.length + "\"");

}

//m\_out.println(">");

} **else** {

// m\_out.println("<div>");

}

**if** (hasReporterOutput) {

**if** (hasThrowable) {

//m\_out.println("<h3>Test Messages</h3>");

}

**for** (String line : msgs) {

//m\_out.println(line + "<br/>");

}

}

**if** (hasThrowable) {

**boolean** wantsMinimalOutput = ans.getStatus() == ITestResult.***SUCCESS***;

**if** (hasReporterOutput) {

/\*m\_out.println("<h3>"

+ (wantsMinimalOutput ? "Expected Exception"

: "Failure") + "</h3>");\*/

}

//generateExceptionReport(exception, method);

}

**if** (hasParameters) {

//m\_out.println("</td></tr>");

} **else** {

//m\_out.println("</div>");

}

}

**if** (hasParameters) {

//m\_out.println("</table>");

}

}

**protected** **void** generateExceptionReport(Throwable exception,

ITestNGMethod method) {

m\_out.print("<div class=\"stacktrace\">");

m\_out.print(Utils.*stackTrace*(exception, **true**)[0]);

m\_out.println("</div>");

}

/\*\*

\* Since the methods will be sorted chronologically, we want to return the

\* ITestNGMethod from the invoked methods.

\*/

**private** Collection<ITestNGMethod> getMethodSet(IResultMap tests,

ISuite suite) {

List<IInvokedMethod> r = Lists.*newArrayList*();

List<IInvokedMethod> invokedMethods = suite.getAllInvokedMethods();

**for** (IInvokedMethod im : invokedMethods) {

**if** (tests.getAllMethods().contains(im.getTestMethod())) {

r.add(im);

}

}

Arrays.*sort*(r.toArray(**new** IInvokedMethod[r.size()]), **new** TestSorter());

List<ITestNGMethod> result = Lists.*newArrayList*();

// Add all the invoked methods

**for** (IInvokedMethod m : r) {

result.add(m.getTestMethod());

}

// Add all the methods that weren't invoked (e.g. skipped) that we

// haven't added yet

**for** (ITestNGMethod m : tests.getAllMethods()) {

**if** (!result.contains(m)) {

result.add(m);

}

}

**return** result;

}

@SuppressWarnings("unused")

**public** **void** generateSuiteSummaryReport(List<ISuite> suites) {

/\*tableStart("testOverview", null);

m\_out.print("<tr>");

tableColumnStart("Test");

tableColumnStart("Methods<br/>Passed");

tableColumnStart("# skipped");

tableColumnStart("# failed");

tableColumnStart("Browser");

tableColumnStart("Start<br/>Time");

tableColumnStart("End<br/>Time");

tableColumnStart("Total<br/>Time(hh:mm:ss)");

tableColumnStart("Included<br/>Groups");

tableColumnStart("Excluded<br/>Groups");

m\_out.println("</tr>");\*/

NumberFormat formatter = **new** DecimalFormat("#,##0.0");

**int** qty\_pass\_m = 0;

**int** qty\_pass\_s = 0;

**int** qty\_skip = 0;

**long** time\_start = Long.***MAX\_VALUE***;

**int** qty\_fail = 0;

**long** time\_end = Long.***MIN\_VALUE***;

m\_testIndex = 1;

**for** (ISuite suite : suites) {

**if** (suites.size() >= 1) {

//titleRow(suite.getName(), 10);

}

Map<String, ISuiteResult> tests = suite.getResults();

**for** (ISuiteResult r : tests.values()) {

qty\_tests += 1;

ITestContext overview = r.getTestContext();

//startSummaryRow(overview.getName());

**int** q = getMethodSet(overview.getPassedTests(), suite).size();

qty\_pass\_m += q;

System.***err***.println("aa----->"+qty\_tests);

}

}

}

**private** **void** summaryCell(String[] val) {

StringBuffer b = **new** StringBuffer();

**for** (String v : val) {

b.append(v + " ");

}

summaryCell(b.toString(), **true**);

}

**private** **void** summaryCell(String v, **boolean** isgood) {

m\_out.print("<td class=\"numi" + (isgood ? "" : "\_attn") + "\">" + v

+ "</td>");

}

**private** **void** startSummaryRow(String label) {

m\_row += 1;

m\_out.print("<tr"

+ (m\_row % 2 == 0 ? " class=\"stripe\"" : "")

+ "><td style=\"text-align:left;padding-right:2em\"><a href=\"#t"

+ m\_testIndex + "\">" + label + "</a>" + "</td>");

}

**private** **void** summaryCell(**int** v, **int** maxexpected) {

summaryCell(String.*valueOf*(v), v <= maxexpected);

}

**private** **void** tableStart(String cssclass, String id) {

m\_out.println("<table width='100%' border=\"5\" cellspacing=\"0\" cellpadding=\"0\""

+ (cssclass != **null** ? " class=\"" + cssclass + "\"" : " ")

+ (id != **null** ? " id=\"" + id + "\"" : "") + ">");

m\_row = 0;

}

**private** **void** tableColumnStart(String label) {

m\_out.print("<th>" + label + "</th>");

}

**private** **void** titleRow(String label, **int** cq) {

titleRow(label, cq, **null**);

}

**private** **void** titleRow(String label, **int** cq, String id) {

m\_out.print("<tr");

**if** (id != **null**) {

m\_out.print(" id=\"" + id + "\"");

}

m\_out.println("><th bgcolor='#cce6ff' colspan=\"" + cq + "\"><font color='black' style='text-shadow:2px 2px white;'>" + label + "<font></th></tr>");

m\_row = 0;

}

/\*\* Starts HTML stream \*/

**protected** **void** startHtml(PrintWriter out) { //3.1

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.1//EN\" \"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<title> Automation build Summary - TestNG Report</title>");

out.println("<style type=\"text/css\">");

out.println("table {margin-bottom:1px;border-collapse:collapse;empty-cells:show}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println("td,th {solid #009;padding:.25em .5em;}");

out.println(".result th {vertical-align:bottom}");

out.println(".param th {padding-left:1em;padding-right:1em}");

out.println(".param td {padding-left:.5em;padding-right:2em}");

out.println(".stripe td,.stripe th {background-color: #E6EBF9}");

out.println(".numi,.numi\_attn {text-align:right}");

out.println(".total td {font-weight:bold}");

out.println(".passedodd td {background-color: #0A0}");

out.println(".passedeven td {background-color: #3F3}");

out.println(".skippedodd td {background-color: #CCC}");

out.println(".skippedodd td {background-color: #DDD}");

out.println(".failedodd td,.numi\_attn {background-color: #F9C1C1}");

out.println(".failedeven td,.stripe .numi\_attn {background-color: #F9C1C1}");

out.println(".stacktrace {white-space:pre;}");

out.println(".totop {font-size:85%;text-align:center;border-bottom:2px solid #000}");

out.println("html \* {");

out.println(" font-family: \"Open Sans\",sans-serif; font-size:11px;}");

out.println("h1 { font-size:25px; }");

out.println("th {font-size:14px; }");

/\*\*\*Collapse expands\*\*\*\*/

out.println(".list { display:none;");

out.println("height:auto;");

out.println(" margin:0;");

out.println("float: left; }");

out.println(".show {");

out.println("display: none; }");

out.println(".hide:target + .show {");

out.println("display: inline; }");

out.println(".hide:target {");

out.println("display: none; }");

out.println(".hide:target ~ .list {");

out.println("display:inline; }");

/\*style the (+) and (-) \*/

out.println(".hide, .show {");

out.println("width: 16px;");

out.println("height: 16px;");

out.println("border-radius: 30px;");

out.println("font-size: 15px;");

out.println("color: #000;");

out.println("text-shadow: 0 1px 0 #666;");

out.println("text-align: center;");

out.println("text-decoration: none;");

out.println("box-shadow: 1px 1px 2px #000;");

out.println("background: #91DDFE;");

out.println("opacity: .95;");

out.println("margin-right: 0;");

out.println("float: left;");

out.println("margin-bottom: 25px; }");

out.println(".hide:hover, .show:hover {");

out.println("color: #eee;");

out.println("text-shadow: 0 0 1px #666;");

out.println("text-decoration: none;");

out.println("box-shadow: 0 0 4px #222 inset;");

out.println("opacity: 1;");

out.println("margin-bottom: 25px; }");

out.println(".list tr{");

out.println("height:auto;");

out.println("margin:0; }");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

out.println("</style>");

out.println("<script src=\"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js\"></script>");

out.println("<script>");

out.println("$(document).ready(function(){");

**for** (**int** i = 0; i < 400; i++) {

out.println(" $(\"#testskiptoggle"+i+"\").click(function(){");

out.println("$(\"#tskiptoggle"+i+"\").toggle(100);");

out.println("});");

}

/\*for (int i=1;i<180;i++)

{

out.println(" $(\"#test"+i+"\").click(function(){");

out.println("$(\"#t"+i+"\").toggle(100);");

out.println("});");

}\*/

out.println("});");

out.println("</script>");

out.println("</head>");

out.println("<body>");

}

/\*\* Finishes HTML stream \*/

**protected** **void** endHtml(PrintWriter out) {

//out.println("<center> Report customized by KiwiQA </center><br/><br/>");

out.println("<tr bgcolor='SkyBlue'><td align='right' colspan='4'><center><b><i>Report customized by KiwiQA </i><b><center></center></b></b></center></td></tr>");

out.println("</body></html>");

}

// ~ Inner Classes --------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

**private** **class** TestSorter **implements** Comparator<IInvokedMethod> {

// ~ Methods

// -------------------------------------------------------------

/\*\* Arranges methods by classname and method name \*/

@Override

**public** **int** compare(IInvokedMethod o1, IInvokedMethod o2) {

// System.out.println("Comparing " + o1.getMethodName() + " " +

// o1.getDate()

// + " and " + o2.getMethodName() + " " + o2.getDate());

**return** (**int**) (o1.getDate() - o2.getDate());

// int r = ((T) o1).getTestClass().getName().compareTo(((T)

// o2).getTestClass().getName());

// if (r == 0) {

// r = ((T) o1).getMethodName().compareTo(((T) o2).getMethodName());

// }

// return r;

}

}

}

**7.) class name :** TestListener

**package** com.utility;

**import** java.util.Set;

**import** org.testng.ITestContext;

**import** org.testng.ITestListener;

**import** org.testng.ITestNGMethod;

**import** org.testng.ITestResult;

**public** **class** TestListener **implements** ITestListener {

@Override

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

Set<ITestResult> failedTests = context.getFailedTests().getAllResults();

Set<ITestResult> skippedTests = context.getSkippedTests().getAllResults();

/\*for (ITestResult temp : failedTests) {

ITestNGMethod method = temp.getMethod();

if (context.getFailedTests().getResults(method).size() > 1) {

failedTests.remove(temp);

} else {

if (context.getPassedTests().getResults(method).size() > 0) {

failedTests.remove(temp);

}

}

}\*/

**for** (ITestResult tempfailed : failedTests) {

ITestNGMethod methodfailed = tempfailed.getMethod();

**for** (ITestResult tempskip : skippedTests) {

ITestNGMethod methodskipped = tempskip.getMethod();

**if** (methodfailed.equals(methodskipped)) {

System.***err***.println(methodfailed.getMethodName()+"======="+methodskipped.getMethodName());

skippedTests.remove(methodfailed);

}

}

/\*else {

if (context.getPassedTests().getResults(method).size() > 0) {

failedTests.remove(temp);

}

}\*/

}

}

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

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

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

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

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

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

}