Testing Report

TestNG framework

Team members:
Bahi Ali.
Berlnty Kerlos.
Gehad Mohsen.
Yasmine Alaa.

Table of contents:

Contents of CD:
presentation file
Demo code folder
Report pdf file

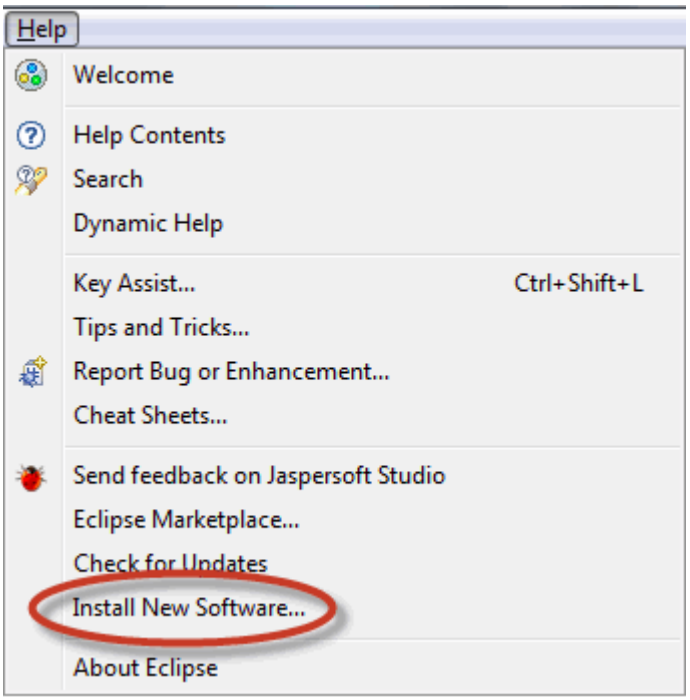
1-Inroduction

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

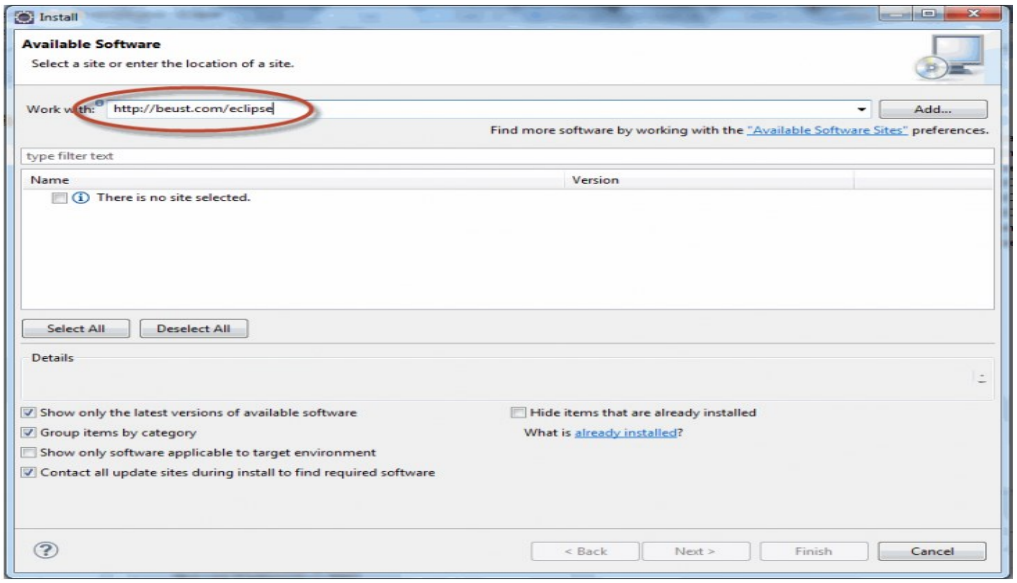
2-How to install testNg in eclipce IDE

1-Step 1:

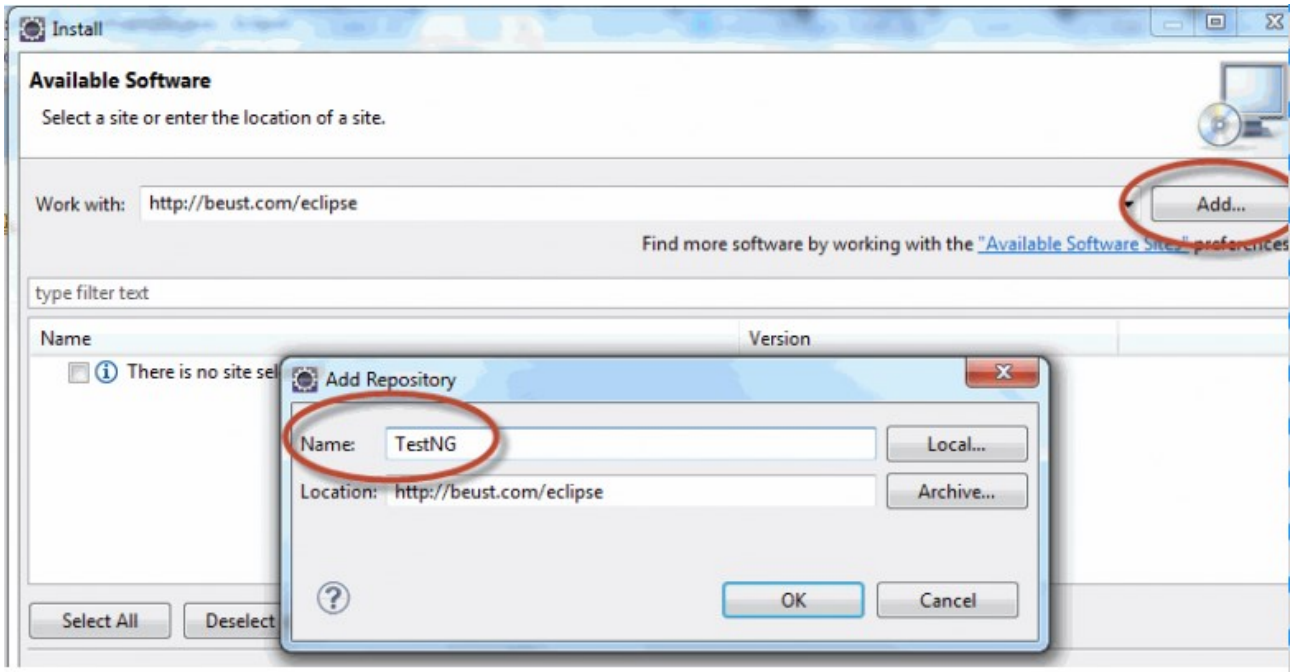
In Eclipse, on top menu bar, Under Help Menu, Click on "Install new Software" in help window



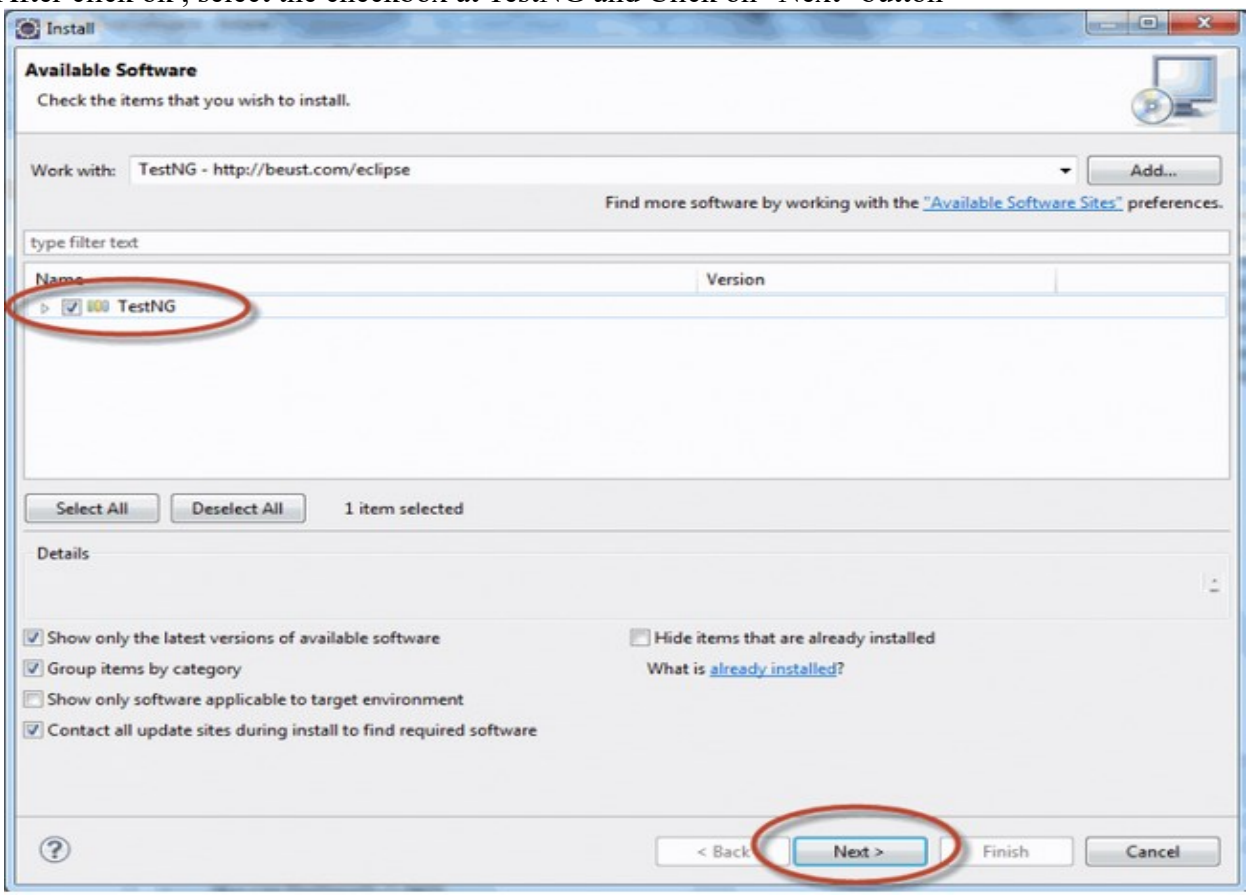
2- Enter the URL (http://beust.com/eclipse/) at Work with field and click on "Add" button



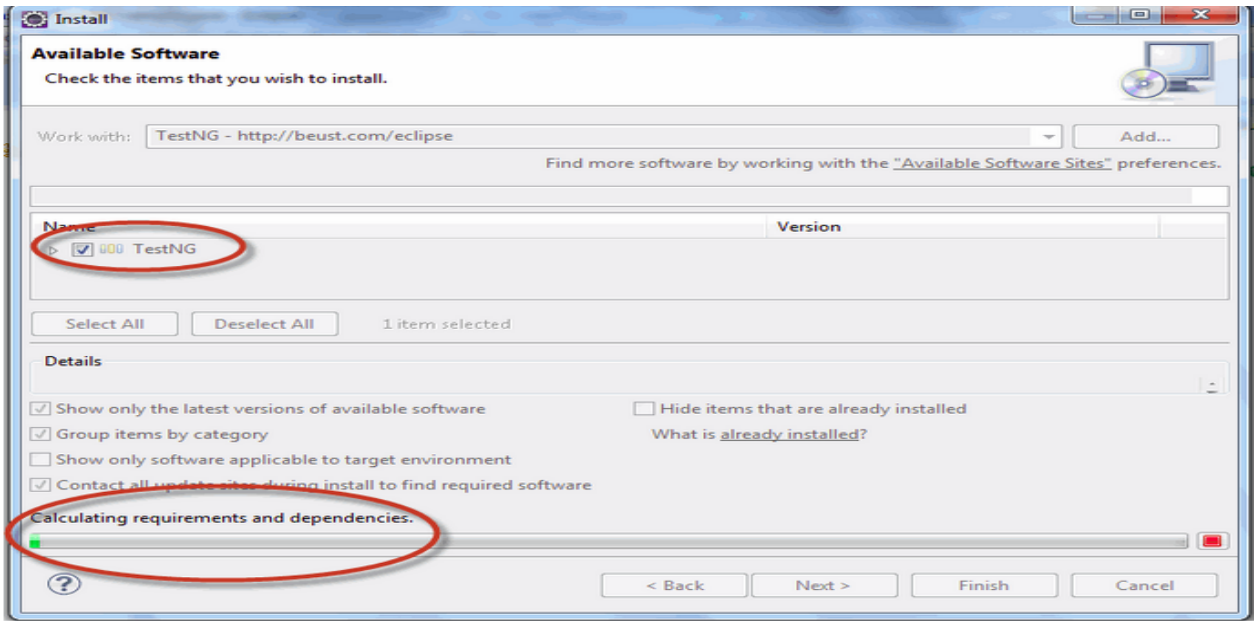
3-Once you click on "Add", it will display the screen, enter the Name as "TestNG"



4-After click ok , select the checkbox at TestNG and Click on "Next" button



5-It will check for the requirement and dependencies before starting the installation

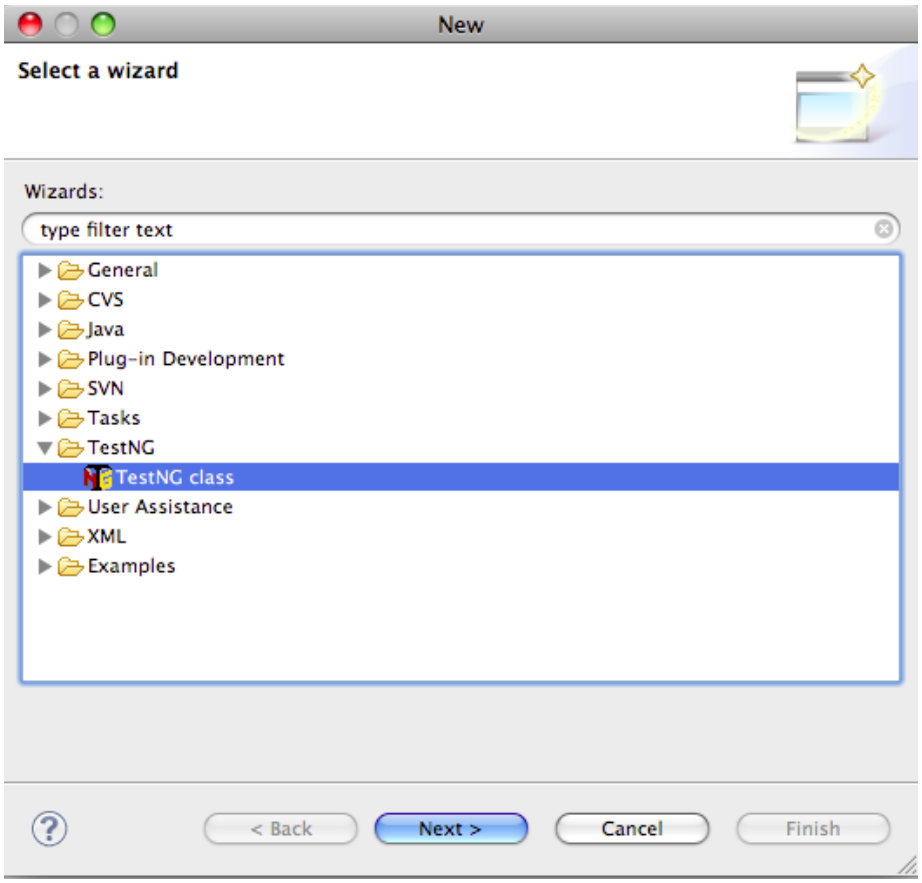


6- Once the above step is done, it will ask you to review the installation details. If your are ready or Ok to install TestNG, click on "Next" to continue

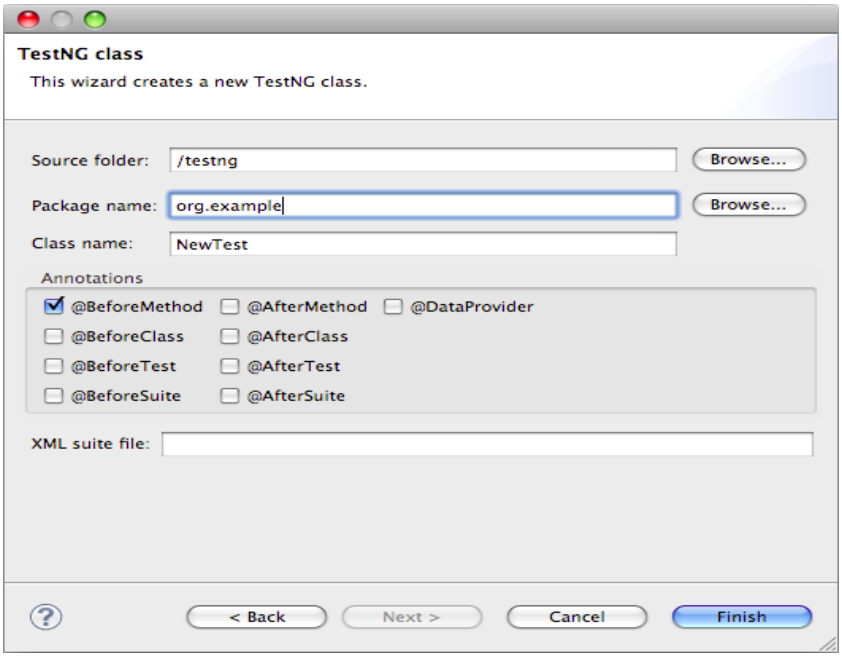
7-Accept the Terms of the license agreement and Click on "Finish" button

3-Creating a TestNG class

To create a new TestNG class, select the menu File / New / TestNG:



then write package name , class name and xml file name



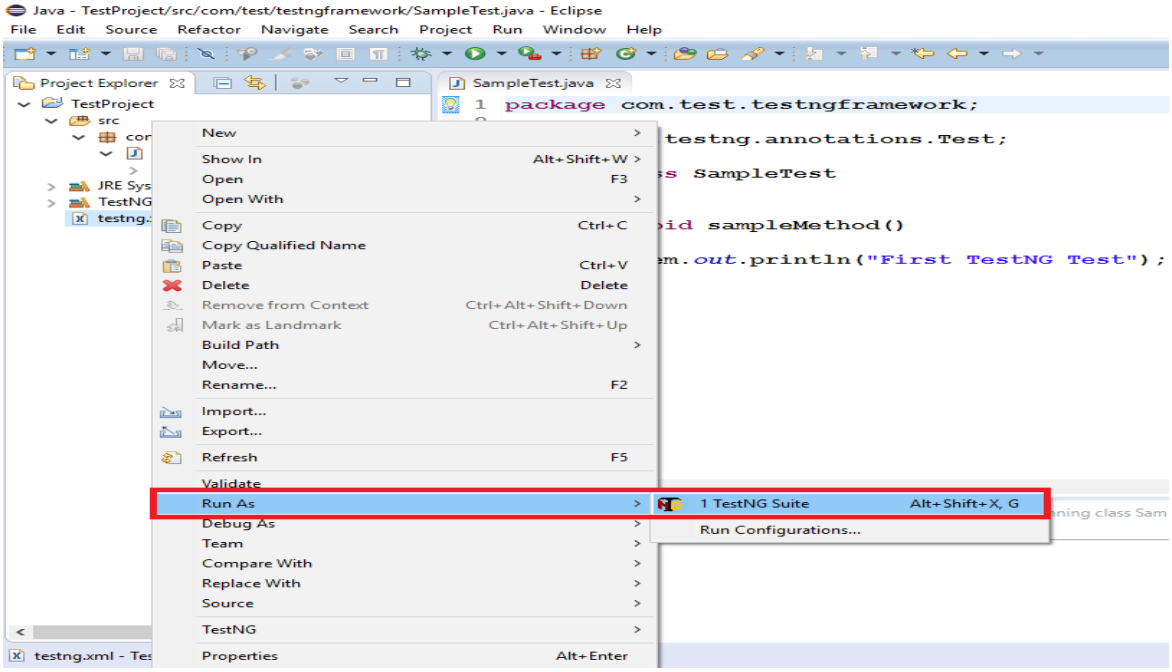
4- How to run TestNg class

first Method:

```
1  <suite name="testingSuite">
2     <test name="testingTest">
3
4        <classes>
5            <class name="testNgTut.testEmp_stat"/>
6            <class name="testNgTut.SecondTestNgScriptTest"/>
7
8        </classes>
9     </test>
10 </suite>
```

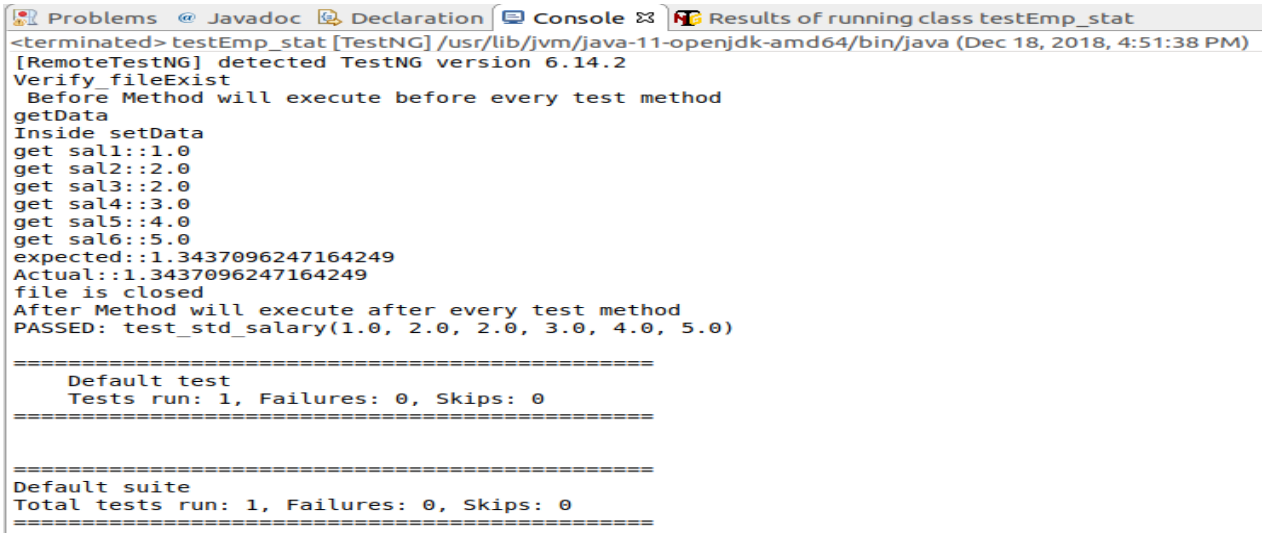
second method

by writing name of testing class in xml file and right click on xml and run as testNg suite

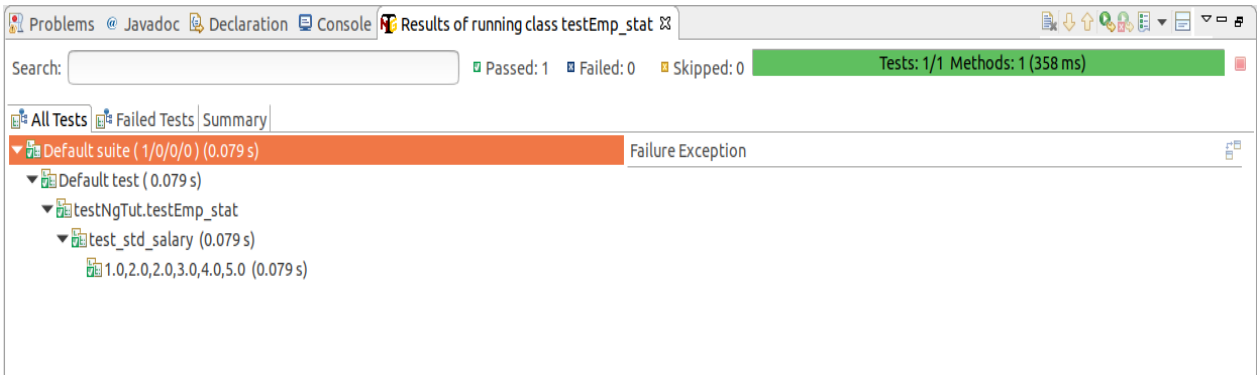


5-Where the results of the testing are viewed

1- In Console

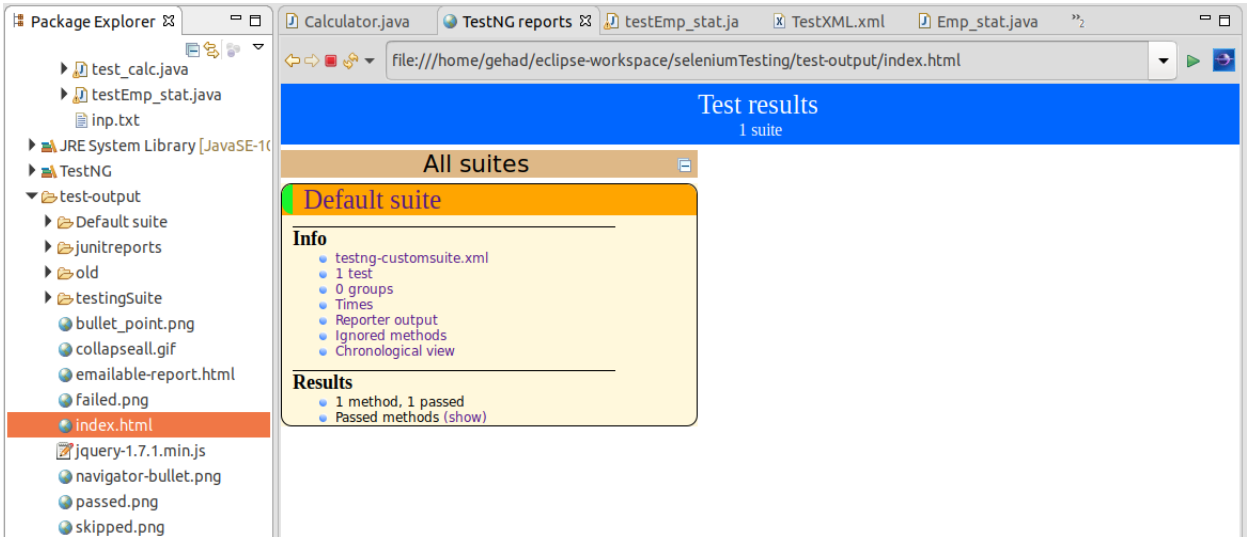


2- In results of running class



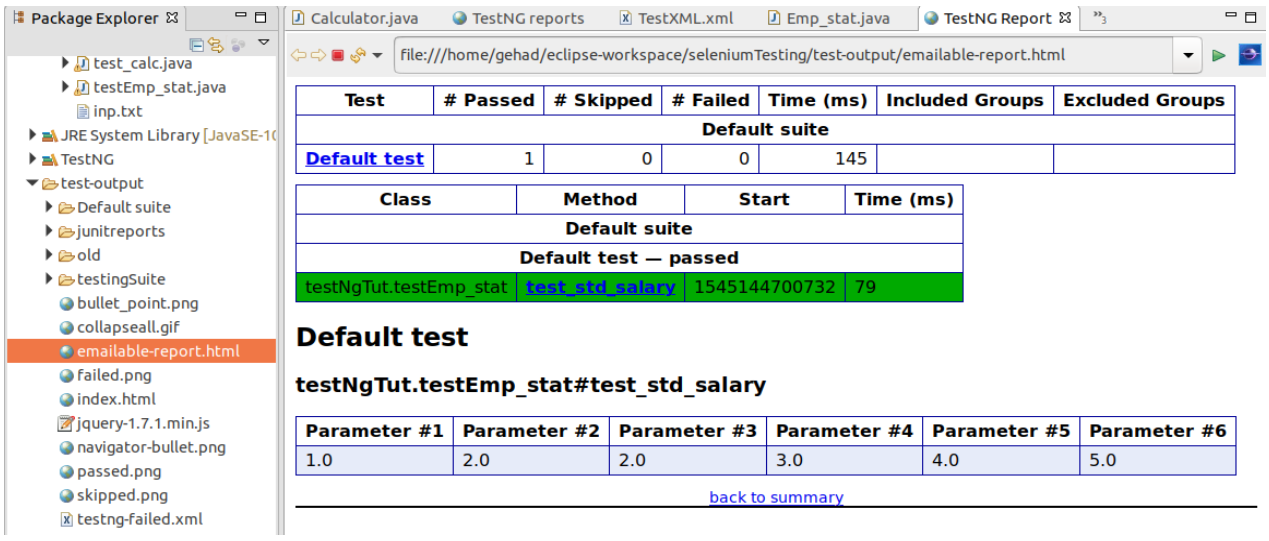
3- reports generated in test-output file

1- index.html



2- emailable-report.html

which can be emailed in the attachment



6-Functionalities in TestNg :

1-TestNG Annotations are easy to create Test cases
2-Test cases can be grouped and prioritized more easily
3-Supports Parameterization
4-Support data driven testing using Dataproviders
5-Execute multiple programs /classes using xml
6-Generate HTML reports
7-Generate emailable reports
8-parallel test execution is possible
9-supports integration with other tools and plugins (Eclipce IDE , build tools like ANT , Maven … etc)

7 – Annotations in TestNg

@BeforeSuite	The annotated method will be run before all tests in this suite have run
@AfterSuite	The annotated method will be run after all tests in this suite have run
@BeforeTest	The annotated method will be run before any test method belonging to the classes inside the <test> tag is run
@AfterTest	The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run
@BeforeGroups	The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked
@AfterGroups	The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked
@BeforeClass	The annotated method will be run before the first test method in the current class is invoked
@AfterClass	The annotated method will be run after all the test methods in the current class have been run
@BeforeMethod	The annotated method will be run before each test method
@AfterMethod	The annotated method will be run after each test method
@DataProvider	**Marks a method as supplying data for a test method. The annotated method must return an Object[][] where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation**
@Factory	**Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[]**
@Listeners	**Defines listeners on a test class**
@Parameters	**Describes how to pass parameters to a @Test method**
@Test	**Marks a class or a method as part of the test**

Example to show **the order of methods called using the below script:**

```
public class TestngAnnotation {

    // Test Case 1
    @Test
    public void testCase1() {
       System.out.println("in Test Case 1");
    }

    // Test Case 2
    @Test
    public void testCase2() {
       System.out.println("in Test Case 2");
    }

    @BeforeMethod
    public void beforeMethod() {
       System.out.println("in Before Method");
    }

    @AfterMethod
    public void afterMethod() {
       System.out.println("in After Method");
    }

    @BeforeClass
    public void beforeClass() {
       System.out.println("in Before Class");
    }

    @AfterClass
    public void afterClass() {
       System.out.println("in After Class");
    }

    @BeforeTest
    public void beforeTest() {
       System.out.println("in Before Test");
    }

    @AfterTest
    public void afterTest() {
       System.out.println("in After Test");
    }

    @BeforeSuite
    public void beforeSuite() {
       System.out.println("in Before Suite");
    }

    @AfterSuite
    public void afterSuite() {
       System.out.println("in After Suite");
    }
```

Console output:

```
[TestNG] Running:

in Before Suite
in Before Test
in Before Class
in Before Method
in Test Case 1
in After Method
in Before Method
in Test Case 2
in After Method
in After Class
in After Test
in After Suite

===============================================
Default suite
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

8-Dependencies in TestNg

To make sure a certain number of test methods have completed and succeeded before running more test methods

TestNG allows to specify dependencies either with annotations or in XML

8.1 - Dependencies with anntations

Example 1:

```
@Test
public void serverStartedOk() {}

@Test(dependsOnMethods = { "serverStartedOk" })
public void method1() {}
```

In this example, method1() is declared as depending on method serverStartedOk(), which guarantees that serverStartedOk() will always be invoked first and if it failed then method1() will be skipped

Example 2:

can also have methods that depend on entire groups:

```
@Test(groups = { "init" })
public void serverStartedOk() {}

@Test(groups = { "init" })
public void initEnvironment() {}

@Test(dependsOnGroups = { "init.*" })
public void method1() {}
```

In this example, method1() is declared as depending on any group matching the regular expression "init.*", which guarantees that the methods serverStartedOk() and initEnvironment() will always be invoked before method1()

8.2 - Dependencies in XML

can specify your group dependencies in the testng.xml file. You use the <dependencies> tag to achieve this:

```
<test name="My suite">
  <groups>
    <dependencies>
      <group name="c" depends-on="a  b" />
      <group name="z" depends-on="c" />
    </dependencies>
  </groups>
</test>
```

9-parametric testing in testNg

Sometimes the business logic requires a hugely varying number of tests. Parameterized tests allow developers to run the same test over and over again using different values

9.1 parametric testing With testng.xml

With this technique, define the simple parameters in the *testng.xml* file and then reference those parameters in the source files

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

public class ParameterizedTest1 {
   @Test
   @Parameters("myName")
   public void parameterTest(String myName) {
      System.out.println("Parameterized value is : " + myName);
   }
}
```

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">
   <test name = "test1">

      <parameter name = "myName" value="manisha"/>

      <classes>
         <class name = "ParameterizedTest1" />
      </classes>

   </test>
</suite>
```

9.2 parametric testing with Data Providers

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc.), parameters can be passed using Dataproviders.

A Data Provider is a method annotated with **@DataProvider**. This annotation has only one string attribute: its name. If the name is not supplied, the data provider's name automatically defaults to the method's name. A data provider returns an array of objects.

```
public class PrimeNumberChecker {
   public Boolean validate(final Integer primeNumber) {

      for (int i = 2; i < (primeNumber / 2); i++) {
         if (primeNumber % i == 0) {
            return false;
         }
      }
      return true;
   }
}
```

```java
import org.testng.Assert;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class ParamTestWithDataProvider1 {
   private PrimeNumberChecker primeNumberChecker;

   @BeforeMethod
   public void initialize() {
      primeNumberChecker = new PrimeNumberChecker();
   }

   @DataProvider(name = "test1")
   public static Object[][] primeNumbers() {
      return new Object[][] {{2, true}, {6, false}, {19, true}, {22, false}
   }

   // This test will run 4 times since we have 5 parameters defined
   @Test(dataProvider = "test1")
   public void testPrimeNumberChecker(Integer inputNumber, Boolean expectec
      System.out.println(inputNumber + " " + expectedResult);
      Assert.assertEquals(expectedResult, primeNumberChecker.validate(input
   }
}
```

10-Priority in Testng

You can run a single or multiple test cases in your **Testng** code. If test **priority** is not defined while, running multiple test cases, **TestNG** assigns all @Test a **priority** as zero(0) and execute them by alphabetical order of their names , otherwise execute them according to their priority (highest priority with the larger number)

```java
public class MultipleTest {

    public WebDriver driver;

  @Test(priority = 0)

  public void One() {

      System.out.println("This is the Test Case number One");

  }

  @Test(priority = 1)

  public void Two() {

      System.out.println("This is the Test Case number Two");

  }

  @Test(priority = 2)

  public void Three() {

      System.out.println("This is the Test Case number Three");

  }

  @Test(priority = 3)

  public void Four() {

      System.out.println("This is the Test Case number Four");

  }

}
```

if test case of highest priority failed then all other lower priority will run ,

while in dependsOnMethod if the test case that another test case depends on failed then this test case will be skipped and isn't run

11- Grouping test cases in TestNg

Group test is a new innovative feature in TestNG, which doesn't exist in JUnit framework. It permits to dispatch methods into proper portions and perform sophisticated groupings of test methods .

Group tests provide maximum flexibility in how to partition tests, and doesn't require to recompile anything if you want to run two different sets of tests back to back.

Groups are specified in testng.xml file using the <groups> tag. It can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.

Example :

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

public class GroupTestExample {
   String message = ".com";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test(groups = { "functest", "checkintest" })

   public void testPrintMessage() {
      System.out.println("Inside testPrintMessage()");
      message = ".com";
      Assert.assertEquals(message, messageUtil.printMessage());
   }

   @Test(groups = { "checkintest" })

   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "tutorialspoint" + ".com";
      Assert.assertEquals(message, messageUtil.salutationMessage());
   }

   @Test(groups = { "functest" })

   public void testingExitMessage() {
      System.out.println("Inside testExitMessage()");
      message = "www." + "tutorialspoint"+".com";
      Assert.assertEquals(message, messageUtil.exitMessage());
   }
}
```

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">
   <test name = "test1">

      <groups>
         <run>
            <include name = "functest" />
         </run>
      </groups>

      <classes>
         <class name = "GroupTestExample" />
      </classes>

   </test>
</suite>
```

Name	Mail	Activities
Bahi Ali	**Bahi.ali26@hotmail.com**	**Searching for suitable testing tool** **Searching for suitable code to test**
Berlnty Kerlos	**berlnty@gmail.com**	**Group testing** **Xml configuration file** **Exception test**
Gehad Mohsen	**gehadmohsen9519@gmail.com**	**Searching for suitable testing tool** **Annotations** **Class testing**
Yasmine Alaa	**Yasminelgourisy@hotmail.com**	**Dependency test** **Parametrization test** **Data providing test**