FREE eBook

LEARNING testng

#testng

Table of Contents

About

Chapter 1: Getting started with testng

Remarks

This section provides an overview of what testng is, and why a developer might want to use it.

It should also mention any large subjects within testng, and link out to the related topics. Since the Documentation for testng is new, you may need to create initial versions of those related topics.

Versions

Version	Date
1.0	2017-06-07

Examples

Installation or Setup

TestNG requires JDK 7 or higher to use.

According to http://testng.org/doc/download.html in order to install testng you need to add testng dependency to your maven pom.xml or gradle build.gradle file

Maven:

```
<repositories>
  <repository>
    <id>jcenter</id>
    <name>bintray</name>
    <url>http://jcenter.bintray.com</url>
  </repository>
</repositories>

<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.9.12</version>
  <scope>test</scope>
</dependency>
```

Gradle:

```
repositories {
    jcenter()
}

dependencies {
    testCompile 'org.testng:testng:6.9.12'
}
```

More options can be found in the official page.

Quick program using TestNG

```
package example;

import org.testng.annotations.*; // using TestNG annotations

public class Test {

 @BeforeClass
 public void setUp() {
   // code that will be invoked when this test is instantiated
 }

 @Test(groups = { "fast" })
 public void aFastTest() {
   System.out.println("Fast test");
 }

 @Test(groups = { "slow" })
 public void aSlowTest() {
    System.out.println("Slow test");
 }

}
```

The method `setUp()` will be invoked after the test class has been built and before any test method is run. In this example, we will be running the group fast, so `aFastTest()` will be invoked while `aSlowTest()` will be skipped.

TestNG Hello World Example

Writing and executing a simple `TestNG` program is mainly 3 step process.

1. Code - write business logic of your test and annotate it with TestNG annotations
2. Configure - add information of your test in `testng.xml` or in `build.xml`
3. Run TestNG - it can be invoked from command line, ANT, IDE like Eclipse, IntelliJ's IDEA)

Brief explanation of example (what needs to be tested):

We have a `RandomNumberGenerator` class which has a method `generateFourDigitPin` that generates a 4 digit PIN and returns as `int`. So here we want to test whether that random number is if of 4 digits or not. Below is the code:

Class to be tested:

```
package example.helloworld;

public class RandomNumberGenerator {
```

```
public int generateFourDigitPin(){
    return (int)(Math.random() * 10000);
}
}
```

The TestNG test class:

```
package example.helloworld;

import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class TestRandomNumberGenerator {

    RandomNumberGenerator rng = null;

    @BeforeClass
    public void deSetup(){
        rng = new RandomNumberGenerator();
    }

    @Test
    public void testGenerateFourDigitPin(){
        int randomNumber = rng.generateFourDigitPin();
        Assert.assertEquals(4, String.valueOf(randomNumber).length());
    }

    @AfterClass
    public void doCleanup(){
        //cleanup stuff goes here
    }
}
```

Ther testng.xml:

```
<suite name="Hello World">
    <test name="Random Number Generator Test">
        <classes>
            <class name="example.helloworld.TestRandomNumberGenerator" />
        </classes>
    </test>
</suite>
```

Run TestNG suite with Gradle

Sample `build.gradle` file:

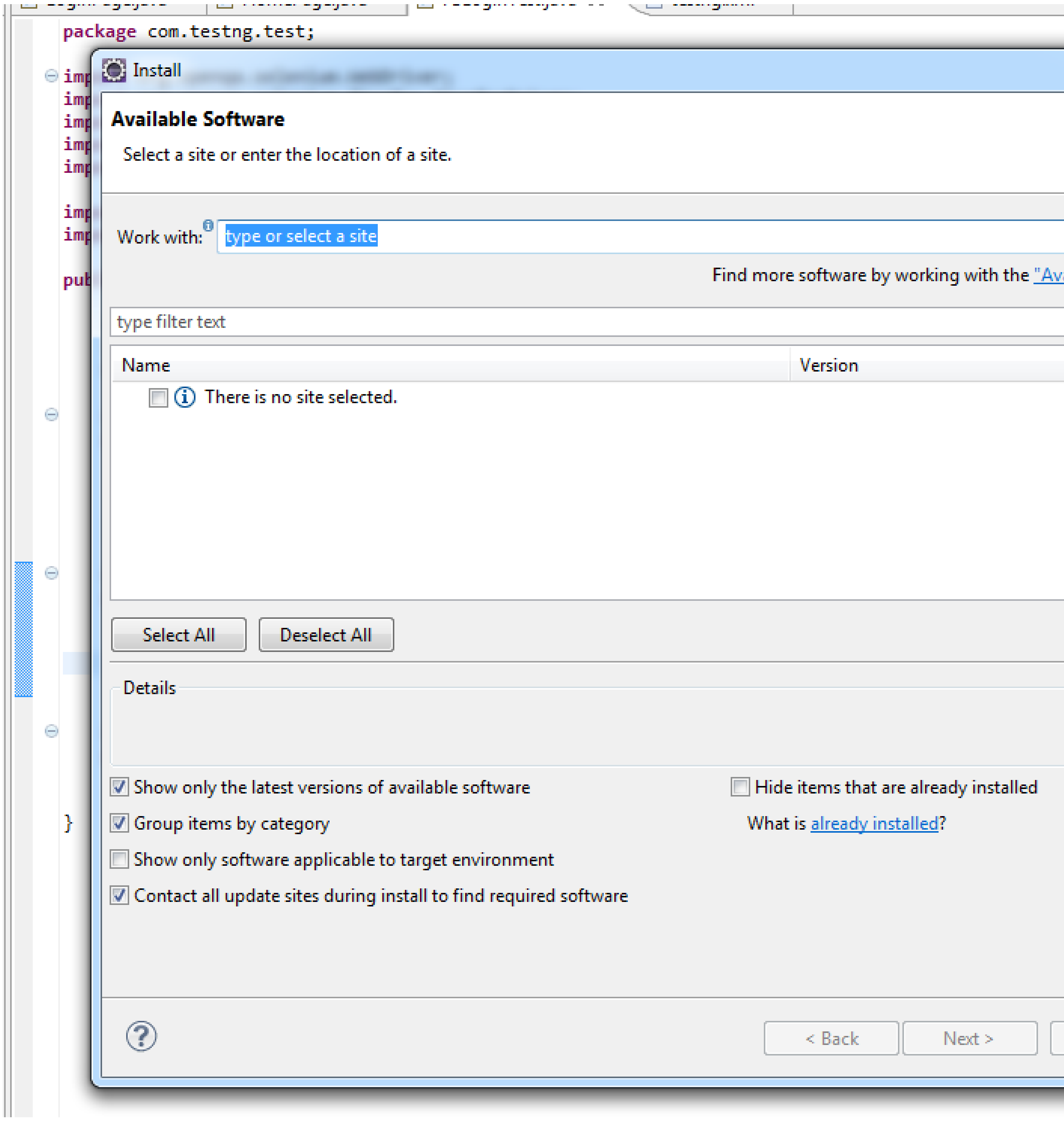
```
plugin: 'java'

repositories {
    mavenLocal()
    mavenCentral()
    jcenter()
```

```
}

dependencies {
    compile "org.testng:testng:6.9.12"
}

test {
    useTestNG() {
    suiteXmlBuilder().suite(name: 'Sample Suite') {
        test(name : 'Sample Test') {
            classes('') {
                'class'(name: 'your.sample.TestClass')
            }
        }
    }
}
```

How to configure TestNG in Eclipse & Run test using xml

How to install TestNG in eclipse

1. Open eclipse
2. Click on Help > Install New software



```
package com.testng.test;
```

Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Av

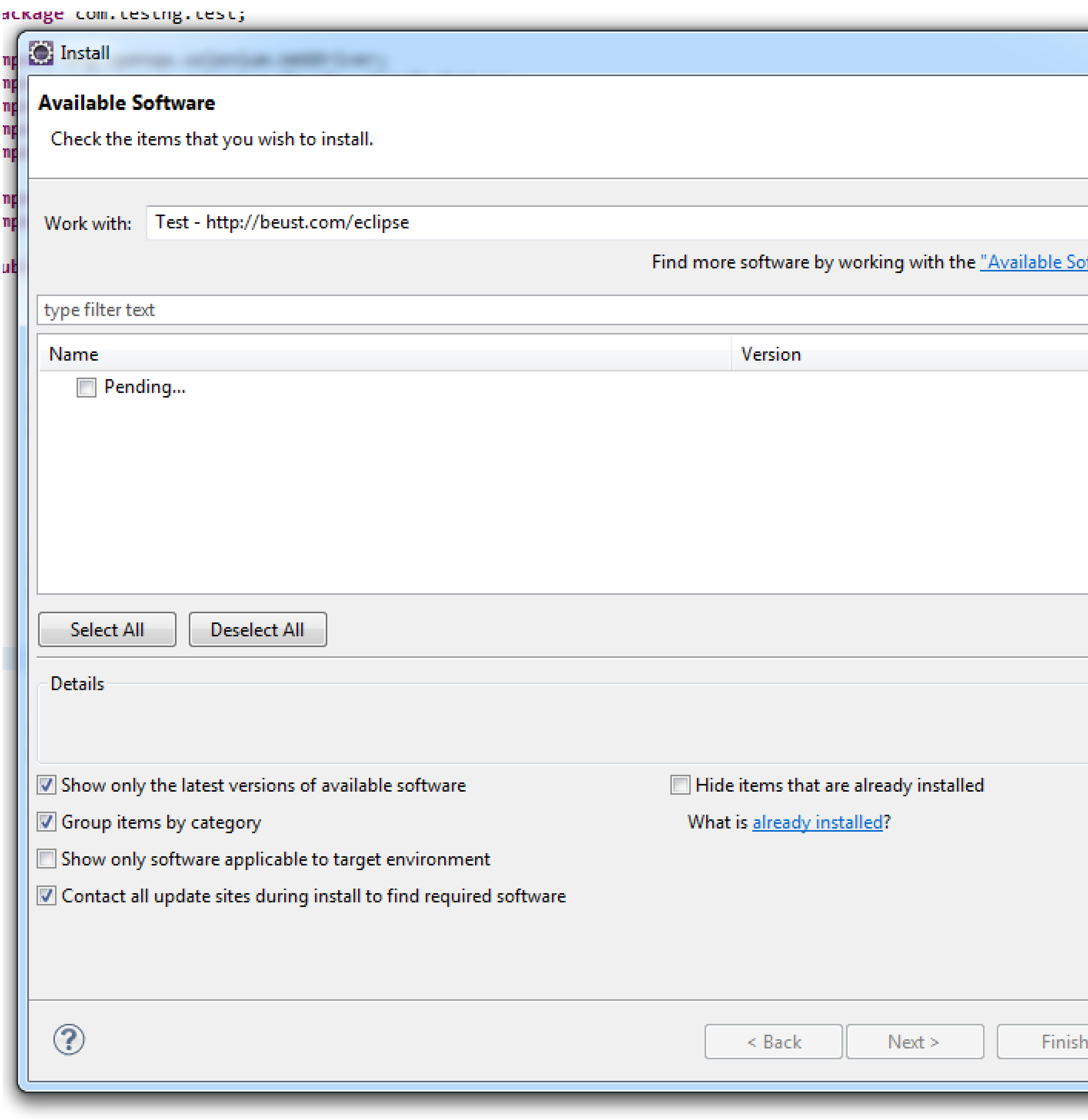
type filter text

Name	Version
☐ There is no site selected.	

- ☑ Show only the latest versions of available software
- ☑ Group items by category
- ☐ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software
- ☐ Hide items that are already installed

What is already installed?

3. Click Add
4. Provide name & URL - http://beust.com/eclipse


Install

Available Software

Check the items that you wish to install.

Work with: Test - http://beust.com/eclipse

Find more software by working with the "Available So

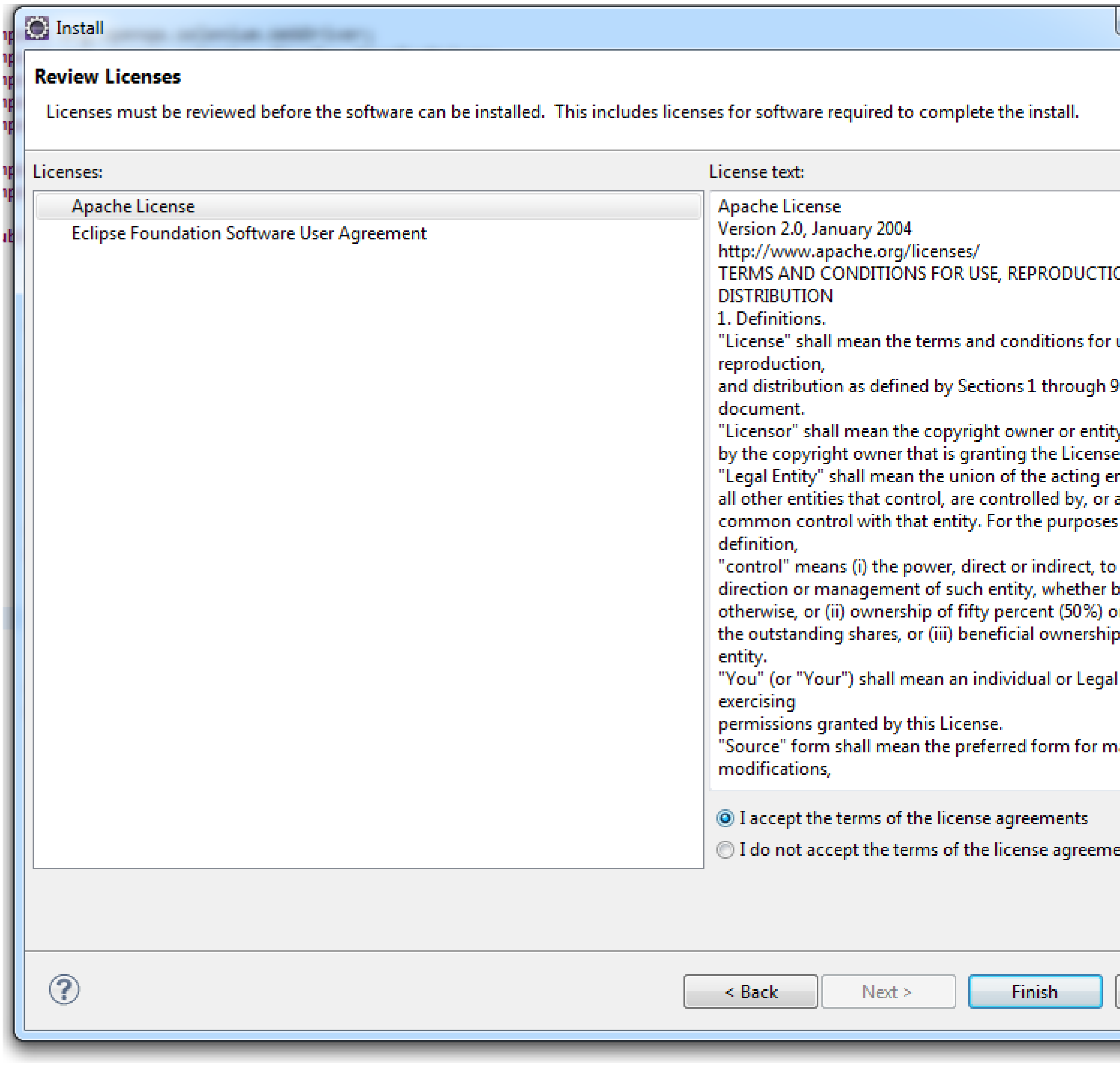
type filter text

Name	Version
☐ Pending...	

- [✓] Show only the latest versions of available software
- [✓] Group items by category
- [ ] Show only software applicable to target environment
- [✓] Contact all update sites during install to find required software
- [ ] Hide items that are already installed

What is already installed?

5. Select TestNG
6. Click Next



Review Licenses

Licenses must be reviewed before the software can be installed. This includes licenses for software required to complete the install.

Licenses:

Apache License
Eclipse Foundation Software User Agreement

License text:

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
TERMS AND CONDITIONS FOR USE, REPRODUCTIO
DISTRIBUTION
1. Definitions.
"License" shall mean the terms and conditions for u
reproduction,
and distribution as defined by Sections 1 through 9
document.
"Licensor" shall mean the copyright owner or entity
by the copyright owner that is granting the License
"Legal Entity" shall mean the union of the acting er
all other entities that control, are controlled by, or a
common control with that entity. For the purposes
definition,
"control" means (i) the power, direct or indirect, to
direction or management of such entity, whether b
otherwise, or (ii) ownership of fifty percent (50%) o
the outstanding shares, or (iii) beneficial ownership
entity.
"You" (or "Your") shall mean an individual or Legal
exercising
permissions granted by this License.
"Source" form shall mean the preferred form for ma
modifications,

[✓ I accept the terms of the license agreements]
I do not accept the terms of the license agreeme

7. Click Finish
8. It will take some time to install TestNG

Once installed then restart eclipse.

Lets create a TestNG project

1. File > New > Java Project > Provide some name and click finish

2. Create a class as TestNGClass

3. Create following class

   1.LoginPage.class

2.HomePage.class

3.FBLoginTest.class

Here goes the code:

LoginPage class

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {

    @FindBy(id = "email")
    private WebElement username;

    @FindBy(id = "pass")
    private WebElement password;

    @FindBy(xpath = ".//input[@data-testid='royal_login_button']")
    private WebElement login;

    WebDriver driver;

    public LoginPage(WebDriver driver){
        this.driver = driver;
         PageFactory.initElements(driver, this);
    }
    public void enterUserName(String name){
        username.clear();
        username.sendKeys(name);
    }

    public void enterPassword(String passwrd){
        password.clear();
        password.sendKeys(passwrd);
    }


    public HomePage clickLoginButton(){
        login.click();
        return new HomePage(driver);
    }
}
```

HomePage class.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class HomePage {

    @FindBy(id = "userNavigationLabel")
    private WebElement userDropdown;
    WebDriver driver;

    public HomePage(WebDriver driver){
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public boolean isUserLoggedIn(){
        return userDropdown.isDisplayed();
    }

}
```

FBLoginTest class

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.AfterClass;

import com.testng.pages.HomePage;
import com.testng.pages.LoginPage;

public class FBLoginTest {

    WebDriver driver;
    LoginPage loginPage;
    HomePage homePage;

    @BeforeClass
    public void openFBPage(){
        driver = new FirefoxDriver();
        driver.get("https://www.facebook.com/");
        loginPage = new LoginPage(driver);
    }

    @Test
    public void loginToFB(){
        loginPage.enterUserName("");
        loginPage.enterPassword("");
        homePage = loginPage.clickLoginButton();
        Assert.assertTrue(homePage.isUserLoggedIn());
    }

    @AfterClass
    public void closeBrowser(){
        driver.quit();
    }

}
```

Here comes the testng xml: Right click on Project create a xml file and copy paste this content.

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Suite">
  <test name="Test">
```

```
    <classes>
      <class name="com.testng.FBLoginTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

How to add selenium standalone jar:
Download the latest selenium standalone jar & Add that in the Project's Build path.

1. Right click on Project > Build path > Configure Build path > Select Libraries > Add external Jars

How to run the TestNG xml? Right click on the xml > Run as > TestNGSuite

Happy Coding :)

Read Getting started with testng online: https://riptutorial.com/testng/topic/5393/getting-started-with-testng

Chapter 2: @Test Annotation

Syntax

- @Test
- @Test(attribute1 = attributeValue, atrribute2 = attributeValue, etc)

Parameters

Parameter	Details
alwaysRun	If set to true, this test method will always be run even if it depends on a method that failed.
dataProvider	The name of the data provider for this test method.
dataProviderClass	The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class.
dependsOnGroups	The list of groups this method depends on.
dependsOnMethods	The list of methods this method depends on.
description	The description for this method.
enabled	Whether methods on this class/method are enabled.
expectedExceptions	The list of exceptions that a test method is expected to throw. If no exception or a different than one on this list is thrown, this test will be marked a failure.
groups	The list of groups this class/method belongs to.
invocationCount	The number of times this method should be invoked.
invocationTimeOut	The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified.
priority	The priority for this test method. Lower priorities will be scheduled first.
successPercentage	The percentage of success expected from this method
singleThreaded	If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with

Parameter	Details
	`parallel="methods"`. This attribute can only be used at the class level and it will be ignored if used at the method level. **Note**: this attribute used to be called sequential (now deprecated).
timeOut	The maximum number of milliseconds this test should take.
threadPoolSize	The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. **Note**: this attribute is ignored if invocationCount is not specified

Examples

Quick example on @Test annotation

`@Test` annotation can be applied to any **class** or **method**. This annotation marks a class or a method as part of the test.

1. `@Test` at method level - mark annotated method as test method
2. `@Test` at class level
    - The effect of a class level `@Test` annotation is to make all the public methods of the class to become test methods even if they are not annotated.
    - `@Test` annotation can also be repeated on a method if you want to add certain attributes.

Example of `@Test` at method level:

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

public class TestClass1 {
    public void notTestMethod() {
    }

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

Example of `@Test` at class level:

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

@Test
public class TestClass2 {
    public void testMethod1() {
    }

    @Test
    public void testMethod2() {
    }
}
```

Read @Test Annotation online: https://riptutorial.com/testng/topic/6716/-test-annotation

Chapter 3: Parametrized tests

Examples

Data providers

Data providers allow creating multiple test inputs to be run within a test. Let's consider a test which verifies that numbers are doubled correctly. To create data provider provide a static method which returns either `Object[][]` or `Iterator<Object[]>` (the latter allows for lazy computation of the test inputs) annotated with `@DataProvider` annotation, with property `name` being a unique string identifying the provider.

```
import org.testng.annotations.DataProvider;

public class DoublingDataProvider {
    public final static String DOUBLING_DATA_PROVIDER = "doublingDataProvider";

    @DataProvider(name = DOUBLING_DATA_PROVIDER)
    public static Object[][] doubling() {
        return new Object[][]{
                new Object[]{1, 2},
                new Object[]{2, 4},
                new Object[]{3, 6}
        };
    }
}
```

In the above case each `Object[]` represents a set of data for a single test case - here the number to be doubled, followed by the expected value after doubling.

To use the data provider fill the `dataProvider` property of the test with the name of the provider. If the provider method was defined outside of the test class or its base classes, you also have to specify the `dataProviderClass` property. The test method should take parameters corresponding to the elements of the test case description - here it's two ints.

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

import static org.testng.Assert.assertEquals;

public class DoublingTest {

    @Test(dataProvider = DoublingDataProvider.DOUBLING_DATA_PROVIDER, dataProviderClass =
DoublingDataProvider.class)
    public void testDoubling(int number, int expectedResult) {
        assertEquals(number * 2, expectedResult);
    }
}
```

Read Parametrized tests online: https://riptutorial.com/testng/topic/5684/parametrized-tests

Chapter 4: TestNG - Execution Procedure

Examples

Execution procedure of the TestNG test API methods

```
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 beforeMethod");
   }

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

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

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

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

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

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

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

}
```

let's create the file testng.xml in C:>WORKSPACE to execute annotations.

```
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="TestngAnnotation"/>
    </classes>
  </test>
</suite>
```

C:\WORKSPACE>javac TestngAnnotation.java

Now run the testng.xml, which will run the test case defined in the provided Test Case class.

```
in beforeSuite
in beforeTest
in beforeClass
in beforeMethod
in test case 1
in afterMethod
in beforeMethod
in test case 2
in afterMethod
in afterClass
in afterTest
in afterSuite

===============================================
Suite
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

Execution procedure is as follows:

1. First of all, **beforeSuite()** method is executed only once.
2. Lastly, the **afterSuite()** method executes only once.
3. Even the methods **beforeTest()**, **beforeClass()**, **afterClass()**, and **afterTest()** methods are executed only once.
4. **beforeMethod()** method executes for each test case but before executing the test case.
5. **afterMethod()** method executes for each test case but after executing the test case.
6. In between **beforeMethod()** and **afterMethod()**, each test case executes.

Read TestNG - Execution Procedure online: https://riptutorial.com/testng/topic/7889/testng---execution-procedure

Chapter 5: TestNG Groups

Syntax

- @Test(groups = {"group1", "group.regression" }, dependsOnGroups = {"group2", "group3"})

Examples

TestNG Groups configuration and basic example

Groups can be configured under `Suite` and/or `Test` element of `testng.xml`. All groups which are marked as included in `tesng.xml` will be considered for execution, excluded one will be ignored. If a `@Test` method has multiple groups and from those groups if any single groups is excluded in `testng.xml` that `@Test` method will not run.

Below is the typical `testng.xml` configuration at `Test` level for running groups:

```
<suite name="Suite World">
<test name="Test Name">
  <groups>
    <run>
      <include name="functest" />
      <exclude name="regtest" />
    </run>
  </groups>
  <classes>
    <class name="example.group.GroupTest"/>
  </classes>
</test>
</suite>
```

And this how it's test class will look like:

```
package example.group;

import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class GroupTest {

    @BeforeClass
    public void deSetup(){
        //do configuration stuff here
    }

    @Test(groups = { "functest", "regtest" })
    public void testMethod1() {
    }

    @Test(groups = {"functest", "regtest"} )
    public void testMethod2() {
    }

    @Test(groups = { "functest" })
    public void testMethod3() {
    }

    @AfterClass
    public void cleanUp(){
        //do resource release and cleanup stuff here
    }
}
```

On running this `GroupTest` `TestNG` class only `testMethod3()` will be executed.

Explanation:

- `<include name="functest" />` all the test methods of `functest` group are eligible for run if it not excluded by any other group.
- `<exclude name="regtest" />` no test methods of `regtest` group are eligible for run.
- `testMethod1()` and `testMethod2()` are in `regtest` group, so they will not have run.
- `testMethod3()` is in `regtest` group, so it will run.

TestNG MetaGroups - Groups of groups

TestNG allows defining groups which can include other groups. MetaGroups logically combine one or more group(s) and control the execution of the `@Test`methods belonging to those groups.

In below example there are various `@Test` methods belonging to different group(s). Few are specific to particular stack and few are regression and acceptance tests. Here MetaGroups can be created. Let's pick any two simple **MetaGroups**:

1. `allstack` - includes both `liux.jboss.oracle` and `aix.was.db2` groups and enables all test methods belonging to any one of those groups to run together.
2. `systemtest` - includes `allstack`, `regression` and `acceptance` groups and enables all test methods belonging to any one of those groups to run together.

testng.xml configuration

```
<suite name="Groups of Groups">
    <test name="MetaGroups Test">
        <groups>
            <!-- allstack group includes both liux.jboss.oracle and aix.was.db2 groups -->
            <define name="allstack">
                <include name="liux.jboss.oracle" />
                <include name="aix.was.db2" />
            </define>

            <!-- systemtest group includes all groups allstack, regression and acceptance -->
            <define name="systemtest">
                <include name="allstack" />
                <include name="regression" />
                <include name="acceptance" />
            </define>
            <run>
                <include name="systemtest" />
            </run>
        </groups>

        <classes>
            <class name="example.group.MetaGroupsTest" />
        </classes>
    </test>

</suite>
```

MetaGroupsTest class

```
package example.group;

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class MetaGroupsTest {

    @BeforeMethod
    public void beforeMethod(){
        //before method stuffs - setup
    }

    @Test(groups = { "liux.jboss.oracle", "acceptance" })
    public void testOnLinuxJbossOracleStack() {
        //your test logic goes here
    }

    @Test(groups = {"aix.was.db2", "regression"} )
    public void testOnAixWasDb2Stack() {
        //your test logic goes here
    }

    @Test(groups = "acceptance")
    public void testAcceptance() {
        //your test logic goes here
    }

    @Test(groups = "regression")
    public void testRegression(){
        //your test logic goes here
    }

    @AfterMethod
    public void afterMthod(){
        //after method stuffs - cleanup
    }
}
```

Read TestNG Groups online: https://riptutorial.com/testng/topic/5821/testng-groups

Credits

S. No	Chapters	Contributors
1	Getting started with testng	Atul Dwivedi, Community, Idos, mackowski, RocketRaccoon, Sudha Velan
2	@Test Annotation	Atul Dwivedi, Benoit
3	Parametrized tests	Benoit, mszymborski
4	TestNG - Execution Procedure	Shrikant
5	TestNG Groups	Atul Dwivedi