eBook Gratuit

APPRENEZ testng

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#testng

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: testng

It is an unofficial and free testng ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official testng.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer à tester

Remarques

Cette section fournit une vue d'ensemble de ce qu'est le test, et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans le test, et établir un lien avec les sujets connexes. La documentation de testng étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Rendez-vous amoureux
1.0	2017-06-07

Examples

Installation ou configuration

TestNG nécessite JDK 7 ou supérieur pour être utilisé.

Selon http://testng.org/doc/download.html pour installer testng, vous devez ajouter une dépendance testng à votre fichier maven pom.xml ou gradle build.gradle

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'
}
```

Plus d'options peuvent être trouvées dans la page officielle .

Programme rapide utilisant 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");
 }

}
```

La méthode `setUp()` sera appelée après la construction de la classe de test et avant toute méthode de test. Dans cet exemple, nous allons exécuter le groupe rapidement, donc `aFastTest()` sera invoqué pendant que `aSlowTest()` sera ignoré.

TestNG Bonjour World Exemple

Ecrire et exécuter un simple programme `TestNG` est principalement un processus en 3 étapes.

1. Code - écrivez la logique métier de votre test et annotez-la avec les annotations TestNG
2. Configure - ajoute des informations sur votre test dans `testng.xml` ou dans `build.xml`
3. Exécutez TestNG - il peut être appelé depuis la ligne de commande, ANT, IDE comme Eclipse, IDEA d'IntelliJ)

Brève explication de l'exemple (ce qui doit être testé) :

Nous avons une classe `RandomNumberGenerator` qui a une méthode `generateFourDigitPin` qui génère un code PIN à 4 chiffres et renvoie en tant que `int` . Donc, ici, nous voulons tester si ce nombre aléatoire est de 4 chiffres ou non. Voici le code:

Classe à tester :

```
package example.helloworld;

public class RandomNumberGenerator {

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

La classe de test TestNG :

```
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>
```

Exécuter la suite TestNG avec Gradle

Exemple de fichier `build.gradle` :

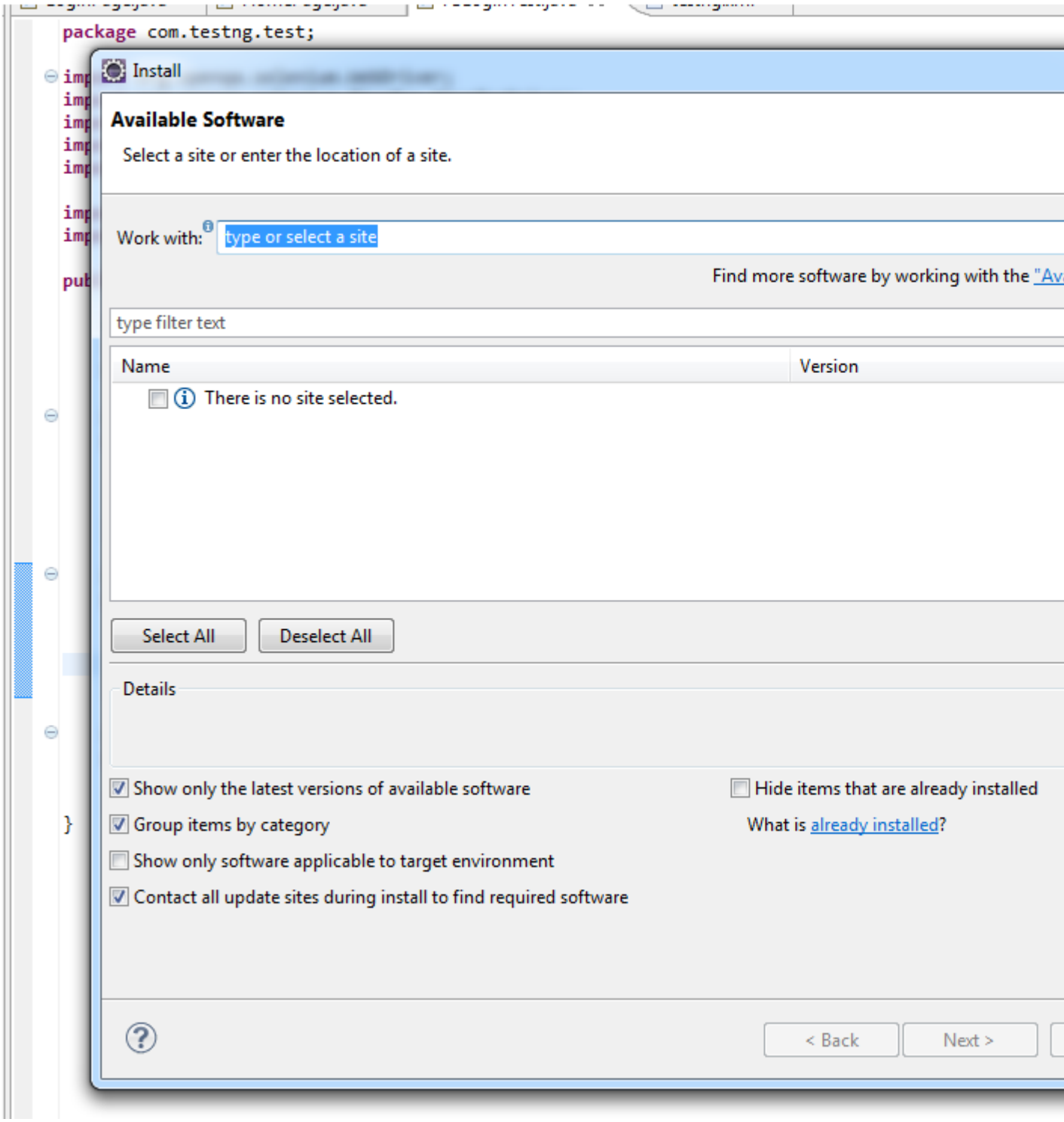
```
plugin: 'java'

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')
            }
        }
    }
}
```

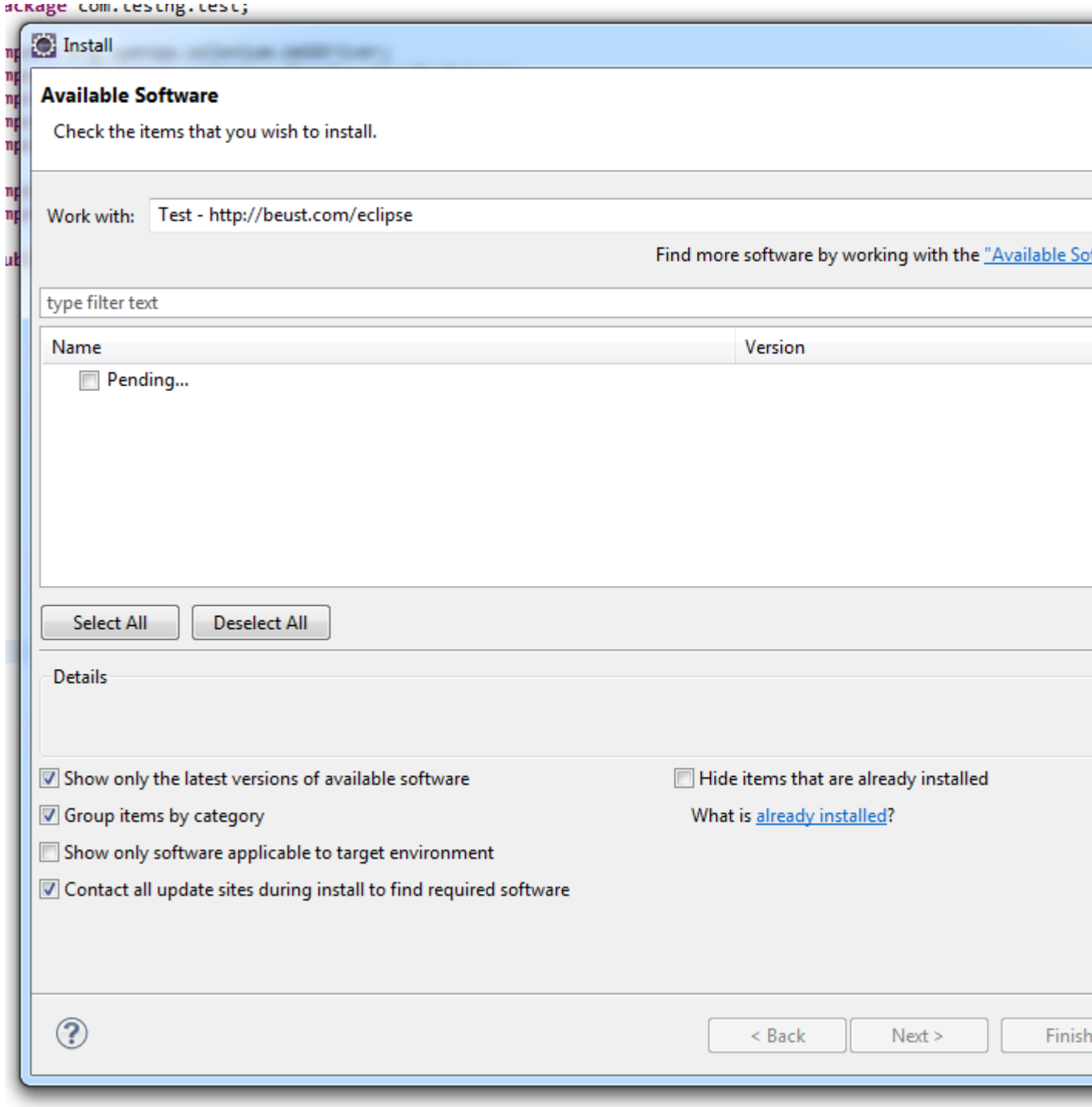
Comment configurer TestNG dans Eclipse & Run test en utilisant xml

Comment installer TestNG dans éclipse

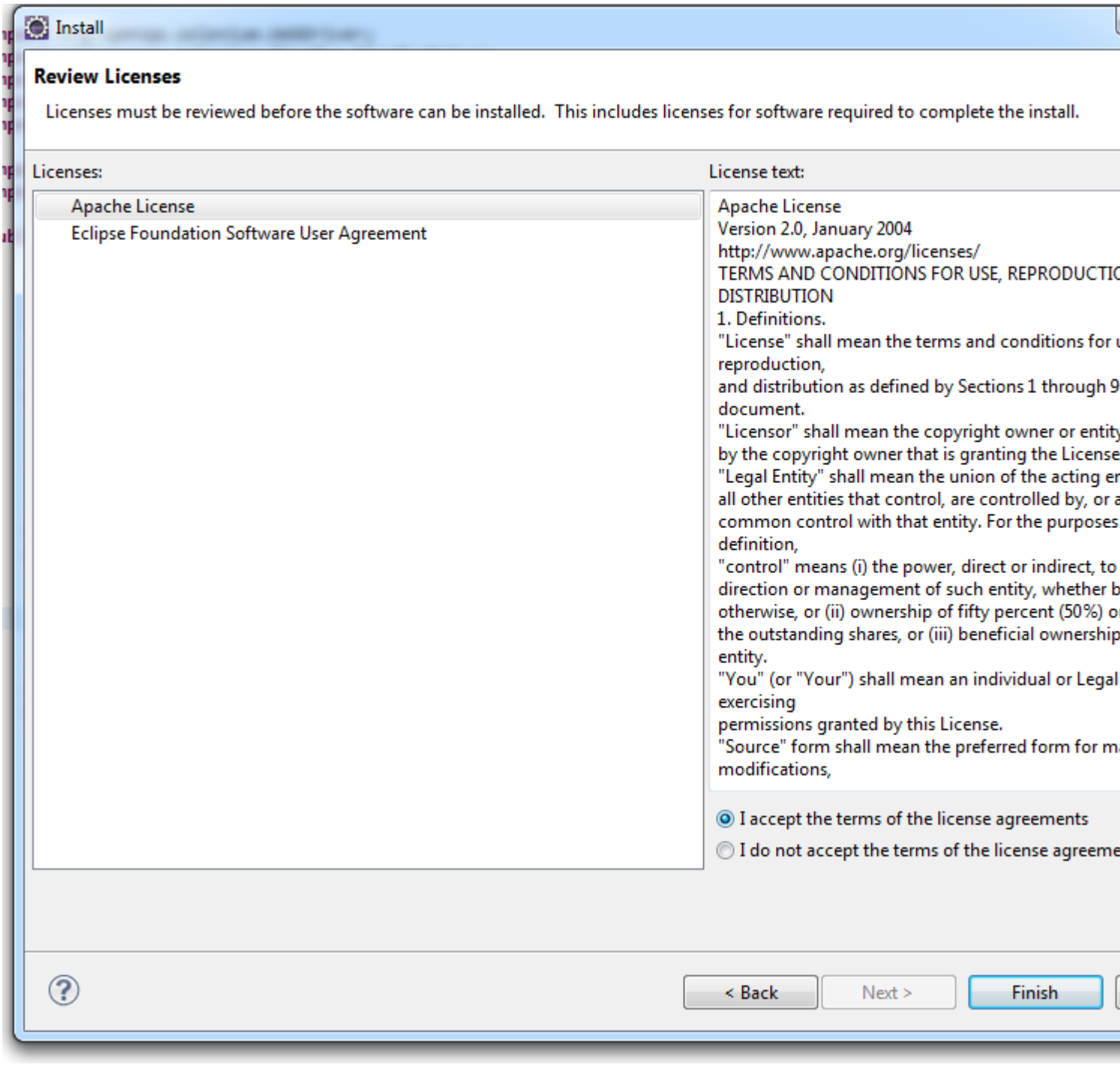
1. Éclipse ouverte
2. Cliquez sur Aide> Installer un nouveau logiciel



3. Cliquez sur Ajouter
4. Indiquez le nom et l'URL - http://beust.com/eclipse



5. Sélectionnez TestNG
6. Cliquez sur Suivant



7. Cliquez sur Terminer
8. L'installation de TestNG prendra du temps

Une fois installé, redémarrez eclipse.

Permet de créer un projet TestNG

1. Fichier> Nouveau> Projet Java> Indiquez un nom et cliquez sur Terminer

2. Créez une classe comme TestNGClass

3. Créer la classe suivante

    1.LoginPage.class

2.HomePage.class

3.FBLoginTest.class

Voici le code:

Classe LoginPage

```
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);
    }
}
```

Classe de page d'accueil

```
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();
    }

}
```

Classe FBLoginTest

```
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();
    }

}
```

Voici le xml testng: Faites un clic droit sur Project créez un fichier xml et copiez-le coller.

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

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

Comment ajouter le pot autonome de sélénium:
Téléchargez le dernier bocal autonome au sélénium et ajoutez-le au chemin de construction du projet.

1. Cliquez avec le bouton droit de la souris sur Projet> Créer un chemin> Configurer le chemin de génération> Sélectionnez les bibliothèques> Ajouter des fichiers Jars externes.

Comment exécuter le XML TestNG? Faites un clic droit sur le xml> Exécuter en tant que> TestNGSuite

Bonne codage :)

Lire Commencer à tester en ligne: https://riptutorial.com/fr/testng/topic/5393/commencer-a-tester

Chapitre 2: @Test Annotation

Syntaxe

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

Paramètres

Paramètre	Détails
Toujours courir	Si elle est définie sur true, cette méthode de test sera toujours exécutée même si elle dépend d'une méthode ayant échoué.
fournisseur de données	Le nom du fournisseur de données pour cette méthode de test.
dataProviderClass	La classe où chercher le fournisseur de données. S'il n'est pas spécifié, le fournisseur de données sera examiné dans la classe de la méthode de test actuelle ou dans l'une de ses classes de base. Si cet attribut est spécifié, la méthode du fournisseur de données doit être statique sur la classe spécifiée.
dependOnGroups	La liste des groupes dont dépend cette méthode.
dependOnMethods	La liste des méthodes dont dépend cette méthode.
la description	La description de cette méthode.
activée	Si les méthodes de cette classe / méthode sont activées.
Exceptions attendues	La liste des exceptions qu'une méthode de test est censée lancer. Si aucune exception ou autre que celle de cette liste n'est levée, ce test sera marqué comme un échec.
groupes	La liste des groupes auxquels appartient cette classe / méthode.
invocationCount	Nombre de fois où cette méthode doit être invoquée.
invocationTimeOut	Nombre maximal de millisecondes que ce test doit prendre pour le temps cumulé de tous les décomptes. Cet attribut sera ignoré si invocationCount n'est pas spécifié.
priorité	La priorité pour cette méthode de test. Les priorités moins élevées seront programmées en premier.
succèsPourcentage	Le pourcentage de réussite attendu de cette méthode

Paramètre	Détails
single threaded	Si défini sur true, toutes les méthodes de cette classe de test sont garanties pour s'exécuter dans le même thread, même si les tests sont en cours d'exécution avec `parallel="methods"` . Cet attribut ne peut être utilisé qu'au niveau de la classe et il sera ignoré s'il est utilisé au niveau de la méthode. **Remarque** : cet attribut était appelé séquentiel (maintenant obsolète).
temps libre	Le nombre maximum de millisecondes que ce test doit prendre.
threadPoolSize	La taille du pool de threads pour cette méthode. La méthode sera appelée à partir de plusieurs threads, comme spécifié par invocationCount. **Remarque** : cet attribut est ignoré si invocationCount n'est pas spécifié

Examples

Exemple rapide sur l'annotation @Test

`@Test` annotation `@Test` peut être appliquée à n'importe quelle **classe** ou **méthode** . Cette annotation marque une classe ou une méthode dans le cadre du test.

1. `@Test` au niveau de la méthode - marque méthode annotée comme méthode de test
2. `@Test` au niveau de la classe
    - `@Test` annotation `@Test` niveau de la classe a pour effet de faire de toutes les méthodes publiques de la classe des méthodes de test même si elles ne sont pas annotées.
    - `@Test` annotation `@Test` peut également être répétée sur une méthode si vous souhaitez ajouter certains attributs.

Exemple de `@Test` au niveau de la méthode :

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

public class TestClass1 {
    public void notTestMethod() {
    }

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

Exemple de `@Test` au niveau de la classe :

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

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

```
    }

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

Lire @Test Annotation en ligne: https://riptutorial.com/fr/testng/topic/6716/-test-annotation

Chapitre 3: Groupes de test

Syntaxe

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

Examples

Configuration de TestNG Groups et exemple de base

Les groupes peuvent être configurés sous `Suite` et / ou élément `Test` de `testng.xml` . Tous les groupes qui sont marqués comme inclus dans `tesng.xml` seront considérés pour l'exécution, un exclu sera ignoré. Si une méthode `@Test` a plusieurs groupes et de ces groupes si un seul groupe est exclu dans `testng.xml` , la méthode `@Test` ne sera pas exécutée.

Vous trouverez ci-dessous la configuration typique de `testng.xml` au niveau du `Test` pour les groupes en cours d'exécution:

```
<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>
```

Et ce à quoi sa classe de test ressemblera:

```
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
    }
}
```

Lors de l'exécution de cette classe `GroupTest TestNG` , seul `testMethod3()` sera exécuté.

Explication:

- `<include name="functest" />` toutes les méthodes de test du groupe `functest` sont éligibles pour une exécution si elles ne sont exclues par aucun autre groupe.
- `<exclude name="regtest" />` aucune méthode de test du groupe `regtest` peut être exécutée.
- `testMethod1()` et `testMethod2()` sont dans le groupe `regtest` , ils ne seront donc pas exécutés.
- `testMethod3()` est dans le groupe `regtest` , il sera donc exécuté.

TestNG MetaGroups - Groupes de groupes

TestNG permet de définir des groupes pouvant inclure d'autres groupes. Les MetaGroups combinent logiquement un ou plusieurs groupes et contrôlent l'exécution des méthodes `@Test` appartenant à ces groupes.

Dans l'exemple ci-dessous, il existe différentes méthodes `@Test` appartenant à différents groupes. Peu sont spécifiques à une pile particulière et peu sont des tests de régression et d'acceptation. Ici, MetaGroups peut être créé. Choisissons deux simples **MetaGroups** :

1. `allstack` - inclut à la fois les groupes `liux.jboss.oracle` et `aix.was.db2` et permet à toutes les méthodes de test appartenant à l'un de ces groupes d'être exécutées ensemble.
2. `systemtest` - inclut des `allstack` , de `regression` et d' `acceptance` et permet à toutes les méthodes de test appartenant à l'un de ces groupes d'être exécutées ensemble.

configuration **testng.xml**

```
<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>
```

Classe **MetaGroupsTest**

```
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
    }
}
```

Lire Groupes de test en ligne: https://riptutorial.com/fr/testng/topic/5821/groupes-de-test

Chapitre 4: TestNG - Procédure d'exécution

Examples

Procédure d'exécution des méthodes de test TestNG

```
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");
```

```
   }

}
```

Créons le fichier testng.xml dans C:> WORKSPACE pour exécuter les annotations.

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

C: \ WORKSPACE> javac TestngAnnotation.java

Exécutez maintenant le fichier testng.xml, qui exécutera le scénario de test défini dans la classe Test Case fournie.

```
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
===============================================
```

La procédure d'exécution est la suivante:

1. Tout d'abord, la méthode **beforeSuite ()** n'est exécutée qu'une seule fois.
2. Enfin, la méthode **afterSuite ()** ne s'exécute qu'une seule fois.
3. Même les méthodes **beforeTest ()** , **beforeClass ()** , **afterClass ()** et **afterTest ()** ne sont exécutées qu'une seule fois.
4. **La** méthode **beforeMethod ()** s'exécute pour chaque **scénario de** test, mais avant d'exécuter le **scénario de** test.
5. **La** méthode **afterMethod ()** s'exécute pour chaque **scénario de** test, mais après l'exécution du **scénario de** test.
6. Entre **beforeMethod ()** et **afterMethod ()** , chaque cas de test s'exécute.

Lire TestNG - Procédure d'exécution en ligne: https://riptutorial.com/fr/testng/topic/7889/testng---procedure-d-execution

Chapitre 5: Tests paramétrés

Examples

Fournisseurs de données

Les fournisseurs de données permettent de créer plusieurs entrées de test à exécuter dans un test. Considérons un test qui vérifie que les nombres sont correctement doublés. Pour créer un fournisseur de données, fournissez une méthode statique qui retourne `Object[][]` ou `Iterator<Object[]>` (ce dernier permet un calcul `@DataProvider` entrées de test) annoté avec l'annotation `@DataProvider` , le `name` propriété étant une chaîne unique identifiant le fournisseur.

```
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}
        };
    }
}
```

Dans le cas ci-dessus, chaque `Object[]` représente un ensemble de données pour un seul test élémentaire - ici le nombre à doubler, suivi de la valeur attendue après le doublement.

Pour utiliser le fournisseur de données, remplissez la propriété `dataProvider` du test avec le nom du fournisseur. Si la méthode provider a été définie en dehors de la classe de test ou de ses classes de base, vous devez également spécifier la propriété `dataProviderClass` . La méthode de test doit prendre les paramètres correspondant aux éléments de la description du test élémentaire - ici deux pouces.

```
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);
    }
}
```

Lire Tests paramétrés en ligne: https://riptutorial.com/fr/testng/topic/5684/tests-parametres

Crédits

S. No	Chapitres	Contributeurs
1	Commencer à tester	Atul Dwivedi, Community, Idos, mackowski, RocketRaccoon, Sudha Velan
2	@Test Annotation	Atul Dwivedi, Benoit
3	Groupes de test	Atul Dwivedi
4	TestNG - Procédure d'exécution	Shrikant
5	Tests paramétrés	Benoit, mszymborski