Бесплатная электронная книга

# учусь

# testng

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#testng

# Около

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

# глава 1: Начало работы с testng

## замечания

В этом разделе представлен обзор того, что такое testng, и почему разработчик может захотеть его использовать.

Следует также упомянуть о любых крупных предметах в рамках тестирования и ссылки на связанные темы. Поскольку Documentation for testng является новым, вам может потребоваться создать начальные версии этих связанных тем.

## Версии

| Версия | Дата |
| --- | --- |
| 1,0 | 2017-06-07 |

## Examples

### Установка или настройка

TestNG требует использования JDK 7 или выше.

Согласно http://testng.org/doc/download.html , чтобы установить testng, вам нужно добавить зависимость testng к файлу maven pom.xml или 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'
}
```

Дополнительные параметры можно найти на официальной странице .

## Быстрая программа с использованием 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");
 }

}
```

Метод `setUp()` будет вызываться после создания тестового класса и до запуска любого тестового метода. В этом примере мы будем быстро запускать группу, поэтому `aFastTest()` будет вызываться, пока `aSlowTest()` будет пропущен.

## Пример теста TestNG Hello World

Написание и выполнение простой программы `TestNG` - это в основном трехэтапный процесс.

1. Код - написать бизнес-логику вашего теста и аннотировать его с помощью аннотаций TestNG
2. Настроить - добавить информацию о своем тесте в `testng.xml` или в `testng.xml build.xml`
3. Запустите TestNG - его можно вызвать из командной строки, ANT, IDE, например Eclipse, IDEA IntelliJ)

**Краткое объяснение примера (что нужно проверить)** :

У нас есть класс `RandomNumberGenerator` который имеет метод `generateFourDigitPin` который генерирует 4-значный PIN-код и возвращает его как `int` . Итак, здесь мы хотим проверить,

имеет ли это случайное число 4 цифры или нет. Ниже приведен код:

**Класс для тестирования** :

```
package example.helloworld;

public class RandomNumberGenerator {

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

**Класс тестирования 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>
```

## Запустите комплект TestNG с Gradle

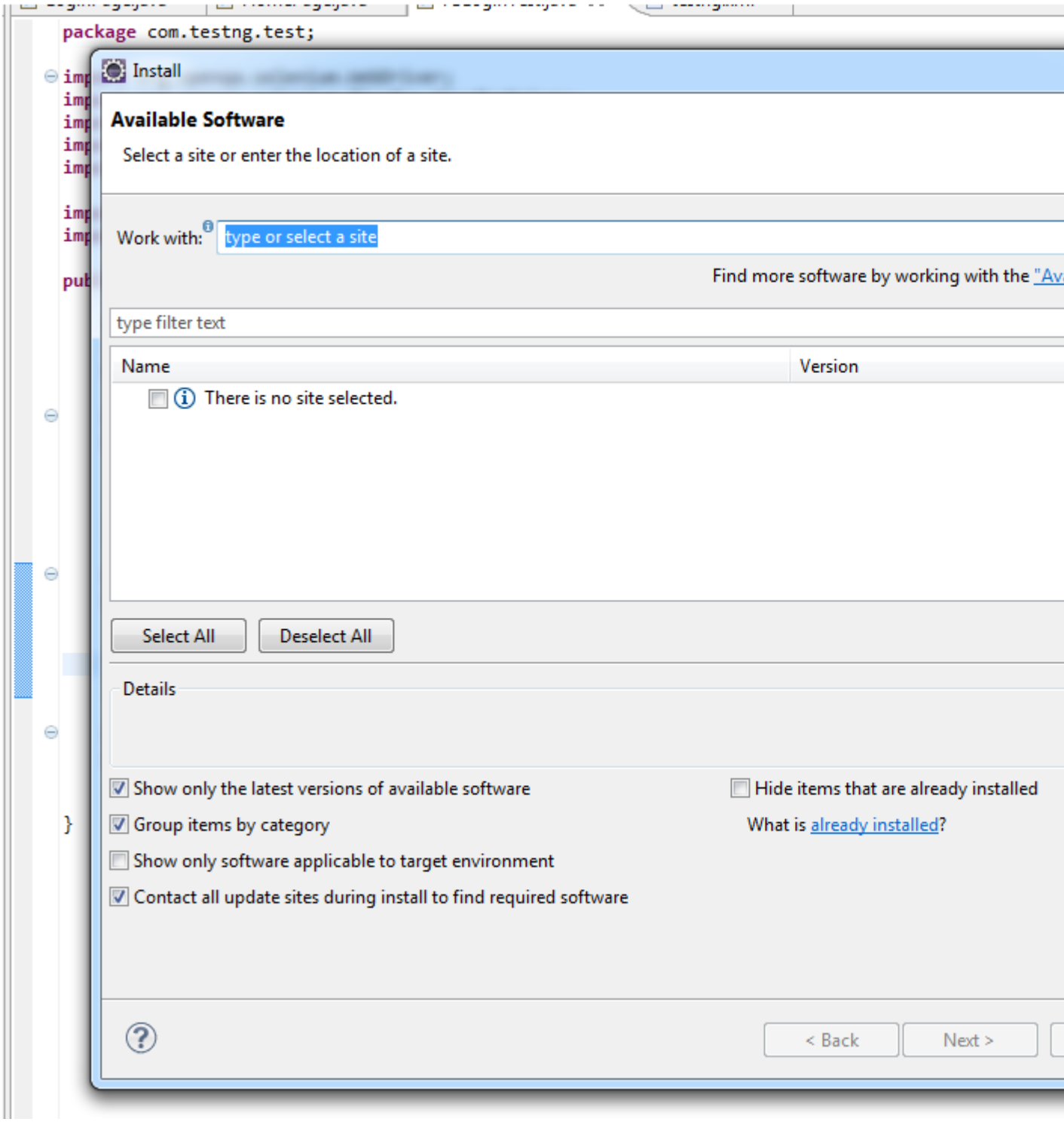
Файл образца `build.gradle` :

```
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')
            }
        }
    }
}
```

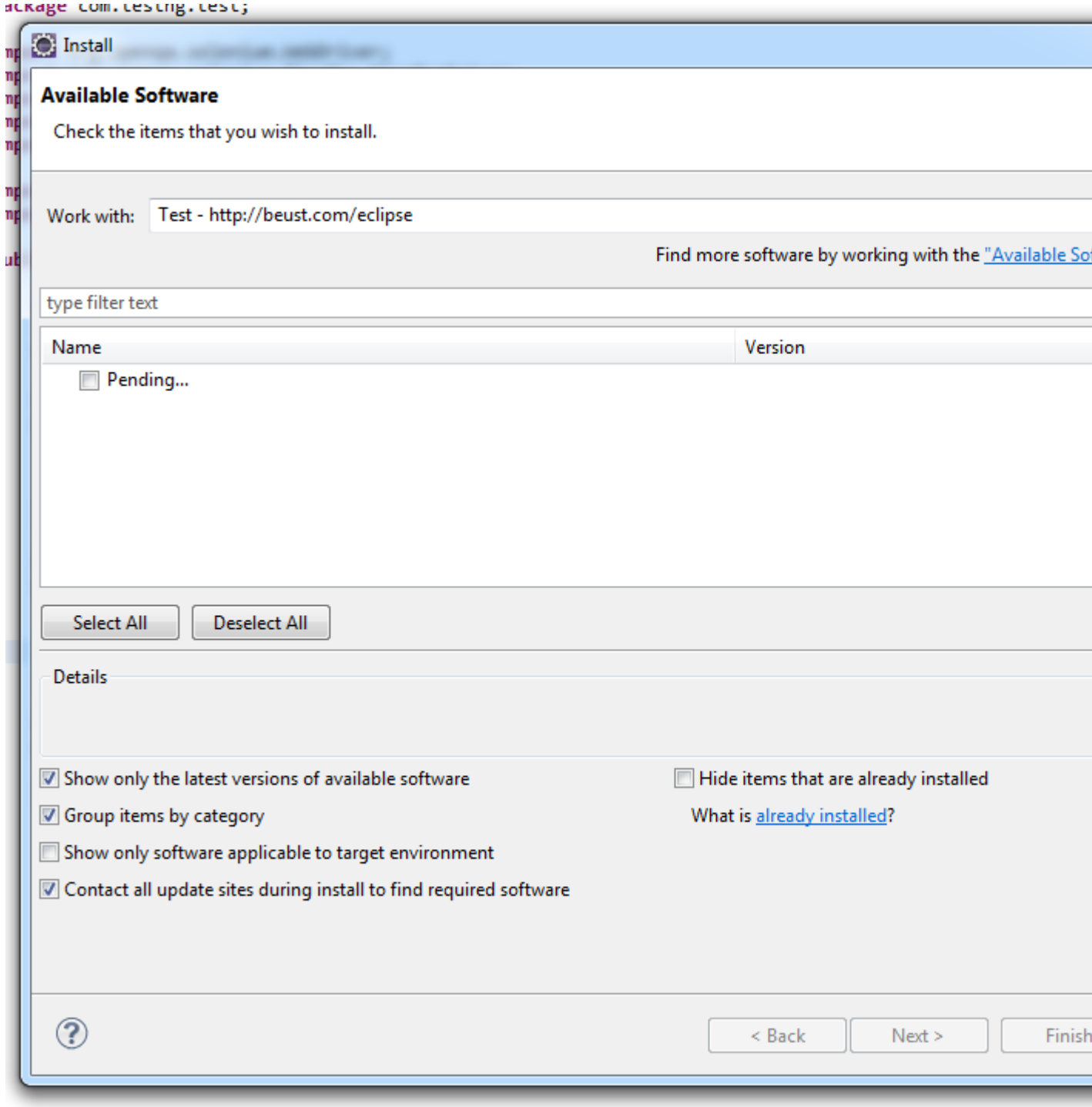
## Как настроить TestNG в тесте Eclipse & Run с помощью xml

### Как установить TestNG в eclipse

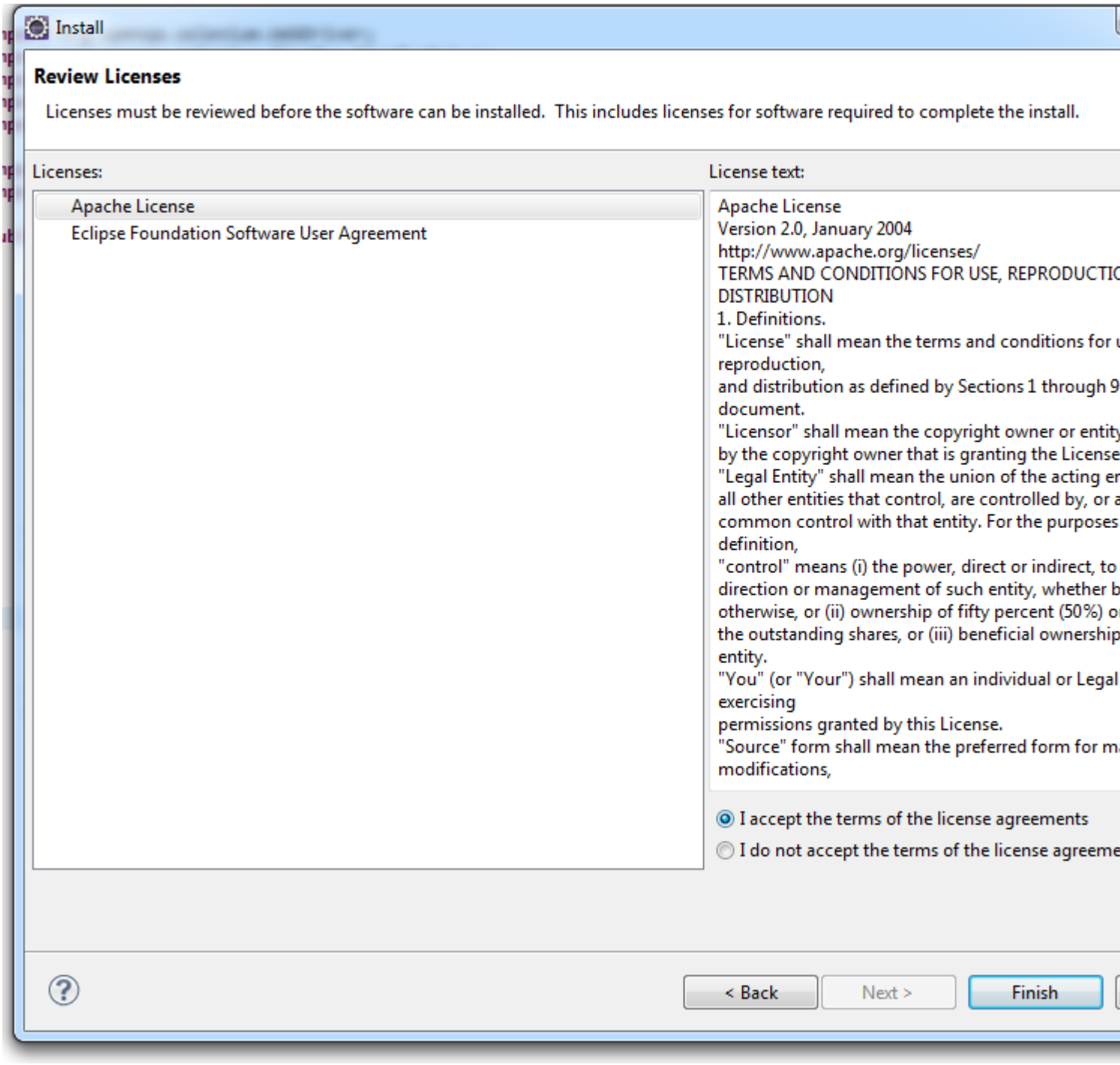
1. Открыть затмение
2. Нажмите «Справка»> «Установить новое программное обеспечение».




3. Нажмите Добавить
4. Укажите имя и URL-адрес - http://beust.com/eclipse




5. Выберите TestNG
6. Нажмите кнопку "Далее




7. Нажмите Готово
8. Для установки TestNG потребуется некоторое время

После установки перезапустите eclipse.

**Позволяет создать проект TestNG**

1. Файл> Создать> Проект Java> Укажите название и нажмите кнопку завершения

2. Создайте класс как TestNGClass

3. Создать следующий класс

1.LoginPage.class

2.HomePage.class

3.FBLoginTest.class

Вот код:

***Класс 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);
    }
}
```

***Класс HomePage*** .

```
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();
    }

}
```

### *Класс 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();
    }

}
```

Здесь идет testng xml: Щелкните правой кнопкой мыши Project, создайте xml-файл и скопируйте его.

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Suite">
  <test name="Test">
    <classes>
      <class name="com.testng.FBLoginTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

**Как добавить отдельную банку из селена:**
Загрузите последнюю отдельную банку selenium и добавьте ее в путь сборки проекта.

1. Щелкните правой кнопкой мыши Project> Путь сборки> Настроить путь сборки> Выбрать библиотеки> Добавить внешние банки

**Как запустить TestNG xml?** Щелкните правой кнопкой мыши на xml> Выполнить как> TestNGSuite

Счастливое кодирование :)

Прочитайте Начало работы с testng онлайн: https://riptutorial.com/ru/testng/topic/5393/начало-работы-с-testng

# глава 2: @Test Annotation

## Синтаксис

- @Тестовое задание
- @Test (attribute1 = attributeValue, atrribute2 = attributeValue и т. Д.)

## параметры

| параметр | подробности |
|---|---|
| alwaysRun | Если установлено значение true, этот метод тестирования всегда будет выполняться, даже если он зависит от метода, который не прошел. |
| DataProvider | Имя поставщика данных для этого метода тестирования. |
| dataProviderClass | Класс, где искать поставщика данных. Если не указано, поставщик данных будет смотреться на классе текущего метода тестирования или одного из его базовых классов. Если этот атрибут указан, метод поставщика данных должен быть статическим в указанном классе. |
| dependsOnGroups | Список групп, от которых зависит этот метод. |
| dependsOnMethods | Список методов, от которых этот метод зависит. |
| описание | Описание этого метода. |
| включен | Включены ли методы этого класса / метода. |
| expectedExceptions | Список исключений, которые, как ожидается, вызовет тестовый метод. Если исключение или другой, кроме одного в этом списке, не будут выбрасываться, этот тест будет отмечен как сбой. |
| группы | Список групп, к которым принадлежит этот класс / метод. |
| invocationCount | Сколько раз этот метод должен быть вызван. |
| invocationTimeOut | Максимальное количество миллисекунд, которое должен пройти этот тест для суммарного времени всех счетов invocation. Этот атрибут будет проигнорирован, если invocationCount не указан. |
| приоритет | Приоритет для этого метода тестирования. Сначала будут |

| параметр | подробности |
| --- | --- |
| | назначены более низкие приоритеты. |
| successPercentage | Процент ожидаемого успеха от этого метода |
| singleThreaded | Если установлено значение true, все методы в этом тестовом классе гарантированно будут выполняться в одном и том же потоке, даже если тесты в настоящее время выполняются с помощью `parallel="methods"` . Этот атрибут может использоваться только на уровне класса, и он будет игнорироваться при использовании на уровне метода. **Примечание** : этот атрибут назывался последовательным (теперь он устарел). |
| Тайм-аут | Максимальное количество миллисекунд, которое должен пройти этот тест. |
| threadPoolSize | Размер пула потоков для этого метода. Метод будет вызван из нескольких потоков, как указано invocationCount. **Примечание** : этот атрибут игнорируется, если invocationCount не указан |

# Examples

### Быстрый пример аннотации @Test

Аннотацию `@Test` можно применить к любому **классу** или **методу** . Эта аннотация обозначает класс или метод как часть теста.

1. `@Test` на уровне метода - отметьте аннотированный метод как метод тестирования
2. `@Test` на уровне класса
    - Эффект аннотации `@Test` уровня `@Test` заключается в том, чтобы все общедоступные методы класса стали методами тестирования, даже если они не были аннотированы.
    - Аннотацию `@Test` можно также повторить по методу, если вы хотите добавить определенные атрибуты.

**Пример `@Test` на уровне метода** :

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

public class TestClass1 {
    public void notTestMethod() {
    }

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

```
}
```

**Пример `@Test` на уровне класса** :

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

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

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

Прочитайте @Test Annotation онлайн: https://riptutorial.com/ru/testng/topic/6716/-test-annotation

# глава 3: TestNG - Процедура выполнения

## Examples

### Процедура выполнения методов API тестирования 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");
   }

}
```

давайте создадим файл testng.xml в C:> WORKSPACE для выполнения аннотаций.

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

C: \ WORKSPACE> javac TestngAnnotation.java

Теперь запустите testng.xml, который запустит тестовый пример, определенный в классе Test Case.

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

Процедура выполнения следующая:

1. Прежде всего, **метод beforeSuite ()** выполняется только один раз.
2. Наконец, метод **afterSuite ()** выполняется только один раз.
3. Даже методы **beforeTest ()** , **beforeClass ()** , **afterClass ()** и **afterTest ()** выполняются только один раз.
4. **Метод beforeMethod ()** выполняется для каждого тестового примера, но перед выполнением тестового примера.
5. **метод afterMethod ()** выполняется для каждого тестового примера, но после выполнения тестового примера.
6. Между **beforeMethod ()** и **afterMethod ()** выполняется каждый тестовый пример.

Прочитайте TestNG - Процедура выполнения онлайн:
https://riptutorial.com/ru/testng/topic/7889/testng---процедура-выполнения

# глава 4: Группы TestNG

## Синтаксис

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

## Examples

### Конфигурация групп TestNG и базовый пример

Группы могут быть настроены в разделе «Элемент `Suite` и / или « `Test` в `testng.xml` . Все группы, отмеченные как включенные в `tesng.xml` будут считаться `tesng.xml` , исключены, они будут проигнорированы. Если метод `@Test` имеет несколько групп и из этих групп, если в `testng.xml` исключены какие-либо отдельные группы, метод `@Test` не будет запущен.

Ниже приведена типичная конфигурация `testng.xml` на уровне `Test` для запуска групп:

```
<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>
```

И это, как это будет тестовый класс, будет выглядеть так:

```
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
    }
}
```

При управлении этой `GroupTest TestNG` класс только `testMethod3()` будет выполняться.

Объяснение:

- `<include name="functest" />` все методы `functest` группы `functest` имеют право на запуск, если они не исключены какой-либо другой группой.
- `<exclude name="regtest" />` никакие методы `regtest` группы `regtest` не могут быть запущены.
- `testMethod1()` и `testMethod2()` находятся в группе `regtest` , поэтому они не будут выполняться.
- `testMethod3()` находится в группе `regtest` , поэтому он будет работать.

## TestNG MetaGroups - Группы групп

TestNG позволяет определять группы, которые могут включать другие группы. MetaGroups логически объединяют одну или несколько групп (групп) и контролируют выполнение методов `@Test` принадлежащих к этим группам.

В приведенном ниже примере существуют различные методы `@Test` принадлежащие разным группам (группам). Немногие специфичны для конкретного стека, и немногие являются регрессионными и приемочными испытаниями. Здесь могут быть созданы MetaGroups. Давайте выберем любые две простые **MetaGroups** :

1. `allstack` - включает в себя как `liux.jboss.oracle` и `aix.was.db2` группы и позволяет всем методам тестирования, принадлежащим любой из этих групп, работать вместе.
2. `systemtest` - включает группы `allstack` , `regression` и `acceptance` и позволяет всем методам тестирования, принадлежащим любой из этих групп, работать вместе.

Конфигурация **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>
```

### Класс **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
    }
}
```

Прочитайте Группы TestNG онлайн: https://riptutorial.com/ru/testng/topic/5821/группы-testng

# глава 5: Параметризированные тесты

## Examples

### Поставщики данных

Поставщики данных позволяют создавать несколько тестовых входов в рамках теста. Рассмотрим тест, который проверяет правильность удвоения чисел. Для создания поставщика данных предоставляется статический метод, который возвращает либо `Object[][]` либо `Iterator<Object[]>` (последний допускает ленивое вычисление тестовых входов), аннотированный аннотацией `@DataProvider` , причем `name` свойства является уникальной строкой, идентифицирующей поставщик.

```
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}
        };
    }
}
```

В приведенном выше случае каждый `Object[]` представляет собой набор данных для одного тестового примера - здесь число должно быть удвоено, а затем ожидаемое значение после удвоения.

Чтобы использовать поставщика данных, заполните свойство `dataProvider` теста именем поставщика. Если метод провайдера был определен вне класса тестирования или его базовых классов, вам также необходимо указать свойство `dataProviderClass` . Метод тестирования должен принимать параметры, соответствующие элементам описания тестового примера - здесь это два 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);
    }
```

```
}
```

Прочитайте Параметризированные тесты онлайн: https://riptutorial.com/ru/testng/topic/5684/параметризированные-тесты

# кредиты

| S. No | Главы | Contributors |
|---|---|---|
| 1 | Начало работы с testng | Atul Dwivedi, Community, Idos, mackowski, RocketRaccoon, Sudha Velan |
| 2 | @Test Annotation | Atul Dwivedi, Benoit |
| 3 | TestNG - Процедура выполнения | Shrikant |
| 4 | Группы TestNG | Atul Dwivedi |
| 5 | Параметризированные тесты | Benoit, mszymborski |