

Latest release

Apache NetBeans 16

[Download \(/download/nb16/\)](#)

Написание тестов JUnit в IDE NetBeans



This tutorial needs a review. You can [edit it in GitHub](https://github.com/apache/netbeans-website/blob/master/netbeans.apache.org/src/content/kb/docs/java/junit-intro_ru.adoc) (https://github.com/apache/netbeans-website/blob/master/netbeans.apache.org/src/content/kb/docs/java/junit-intro_ru.adoc) following these [contribution guidelines](#). ([/kb/docs/contributing.html](#))

В этом учебном курсе описываются основы написания и запуска тестов JUnit в IDE NetBeans. Тестирование приложения является неотъемлемой частью цикла разработки, а написание и поддержка модульных тестов могут гарантировать корректную работу отдельных методов исходного кода. Интегрированная поддержка среды IDE для инфраструктуры модульного тестирования JUnit позволяет быстро и просто создавать тесты JUnit и наборы тестов.

В этом руководстве создаются простые модульные тесты JUnit 3 и JUnit 4 и наборы тестов для проекта библиотеки классов Java. Первая часть руководства посвящена созданию тестов в JUnit 3. Во второй части показан способ создания этих тестов в JUnit 4 с помощью аннотаций JUnit. Не обязательно завершать обе части курса, поскольку тесты в них одинаковы, но с помощью сравнения тестов в обеих версиях можно увидеть изменения, представленные в JUnit 4.

Для получения дополнительных сведений относительно JUnit см. www.junit.org (<http://www.junit.org>).

[Создание проекта](#)

[Создание проекта библиотеки классов Java](#)

[Загрузка проекта решения](#)

[Создание классов Java](#)

[Написание модульных тестов JUnit 3](#)

[Создание тестового класса для Vectors.java](#)

[Написание тестовых методов для](#)

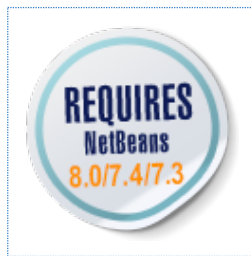


Figure 1. Содержимое этой страницы применимо к IDE NetBeans 7.2, 7.3, 7.4 и 8.0

Для работы с этим учебным курсом требуется следующее программное обеспечение и ресурсы.

Программное обеспечение или материал	Требуемая версия
IDE NetBeans (https://netbeans.org/downloads/index.html)	7.2, 7.3, 7.4, 8.0
Комплект для разработчика на языке Java (JDK) (http://www.oracle.com/technetwork/java/javase/downloads/index.html)	версия 7 или 8
Проект JUnitSampleSol	загрузить (https://netbeans.org/projects/samples/downloads/download/Samples/Java/JUnitSampleSol.zip)

Для выполнения действий данной учебной программы необходимо установить подключаемый модуль JUnit, загрузив его из Центра обновлений. Вы можете установить подключаемый модуль JUnit в диспетчере подключаемых модулей, если не сделали это в процессе установки IDE.

Создание проекта

[Vectors.java](#)

[Создание тестового класса для Utils.java](#)

[Написание тестовых методов для Utils.java](#)

[Выполнение тестов](#)

[Написание тестов JUnit 4](#)

[Создание тестового класса для Vectors.java](#)

[Написание тестовых методов для Vectors.java](#)

[Создание тестового класса для Utils.java](#)

[Написание тестовых методов для Utils.java](#)

[Выполнение тестов](#)

[Создание наборов тестов](#)

[Создание набора тестов JUnit 3](#)

[Создание набора тестов JUnit 4](#)

[Выполнение наборов тестов](#)

Для работы с этим руководством требуется сначала создать проект библиотеки класса Java с именем JUnit-Sample. После создания проекта JUnit-Sample необходимо скопировать в него два класса из демонстрационного проекта JUnitSampleSol.

[Заключение](#)[Дополнительные сведения](#)

Создание проекта библиотеки классов Java

1. В главном меню выберите "Файл" > "Новый проект".
2. Выберите библиотеку классов Java в категории Java и нажмите кнопку "Next".
3. Введите **JUnit-Sample** в качестве имени проекта и укажите местоположение проекта.
4. Снимите флажок "Использовать отдельную папку", если он установлен.

В рамках этого руководства копирование библиотек проекта в выделенную папку не целесообразно, поскольку совместное использование библиотек с другими пользователями или проектами не требуется.

Нажмите кнопку "Завершить".

При первом создании теста JUnit среда IDE требует выбрать версию, после чего добавляет узел "Тестовые библиотеки" и библиотеку JUnit.

Загрузка проекта решения

Загрузить демонстрационный проект JUnitSampleSol, который используется в данном учебном курсе, можно следующими способами.

- Загрузите архив завершенного проекта в формате zip (<https://netbeans.org/projects/samples/downloads/download/Samples/Java/JUnitSampleSol.zip>).
- Выполните проверку исходных файлов проекта на выходе из примеров NetBeans, выполнив перечисленные ниже действия.
 1. Выберите в главном меню "Группа > Subversion > Проверить".
 2. В диалоговом окне "Проверка" введите следующий URL-адрес репозитория: <https://svn.netbeans.org/svn/samples~samples-source-code> (<https://svn.netbeans.org/svn/samples~samples-source-code>). Нажмите кнопку "Далее".
 1. В папках на панели "Взятие для изменения" нажмите кнопку "Обзор", чтобы открыть диалоговое окно "Обзор папок репозитория".

2. Разверните корневой узел и выберите **samples/java/JUnitSampleSol**. Нажмите кнопку "OK".
3. Укажите локальную папку для исходных файлов. Нажмите кнопку "Завершить".

После нажатия кнопки "Готово" среда IDE инициализирует локальную папку в качестве репозитория Subversion и выполняет проверку исходных файлов проекта на выходе.

1. Щелкните команду "Открыть проект" в диалоговом окне, которое появится после завершения проверки.

Дополнительные сведения об установке Subversion см. в разделе [Настройка Subversion \(../ide/subversion.html#settingUp\)](#) в [Руководстве по Subversion в IDE NetBeans \(../ide/subversion.html\)](#).

Примечание. Если вы не установите плагин JUnit при установке IDE, то когда вы откроете проект NetBeans, будет отображен запрос на установку подключаемого модуля JUnit для разрешения ссылки на библиотеки JUnit.

Создание классов Java

В этом упражнении требуется скопировать файлы `Utils.java` и `Vectors.java` из демонстрационного проекта `JUnitSampleSol` в проект библиотеки классов, созданный ранее.

1. В окне 'Проекты' щелкните правой кнопкой мыши узел 'Исходные пакеты' проекта **JUnit-Sample** и выберите 'Создать' > 'Пакет Java' во всплывающем меню.
2. Введите **sample** в качестве имени пакета. Нажмите кнопку "Завершить".
3. Откройте проект **JUnitSampleSol** (если он еще не открыт) и разверните узел "Пакеты исходных файлов" в окне проектов.

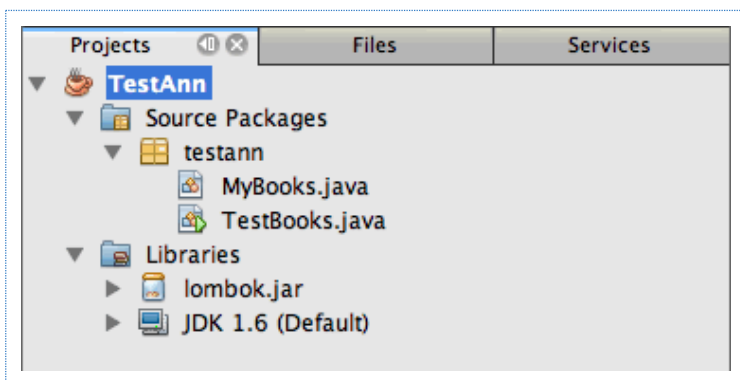


Figure 2. Проекты JUnit-Sample и JUnitSampleSol в окне 'Проекты'

1. Скопируйте файлы классов `Utils.java` и `Vectors.java` из проекта `JUnitSampleSol` в пакет исходного кода `sample` в `JUnit-Sample`.

Если открыть исходный код этих классов, можно заметить, что класс `Utils.java` содержит три метода (`computeFactorial` , `concatWords` и `normalizeWord`), а класс `Vectors.java` — два (`equal` и `scalarMultiplication`). В следующем действии будут созданы тестовые классы для каждого класса и написаны тестовые примеры для методов.

Примечание. Проект `JUnitSampleSol` можно закрыть, поскольку он более не потребуется. Проект `JUnitSampleSol` содержит все тесты, описанные в документе.

Написание модульных тестов JUnit 3

В этой части руководства рассматривается создание основных модульных тестов JUnit 3 для классов `Vectors.java` и `Utils.java` . Для создания скелетных тестовых классов, основанных на классах проекта, будет использована среда IDE. Затем созданные тестовые методы будут изменены, а также добавлены новые тестовые методы.

При первом использовании среды IDE для создания тестов для проекта будет выведен запрос на выбор версии JUnit. Выбранная версия определяется как версия JUnit по умолчанию, и все последующие тесты и наборы тестов в среде IDE будут создаваться для этой версии.

Создание тестового класса для `Vectors.java`

В этом упражнении будет создан скелет теста JUnit для `Vectors.java` . В качестве тестовой среды также выберите JUnit, а в качестве версии - JUnit 3.

Примечание. Если используется NetBeans IDE 7.1 или более ранняя версия, указывать тип тестовой среды не требуется, так как JUnit выбран по умолчанию. В NetBeans IDE 7.2 можно выбрать в качестве тестовой среды JUnit или TestNG.

1. Щелкните правой кнопкой мыши `Vectors.java` и выберите "Сервис > Создать тесты".
2. Измените имя тестового класса на **`VectorsJUnit3Test`** в диалоговом окне "Create Tests".

В результате изменения имени тестового класса появится предупреждение об изменении имени. Имя по умолчанию – это имя тестируемого класса с добавленным словом "Test". Например, для класса `MyClass.java` именем по умолчанию тестового класса будет `MyClassTest.java` . Рекомендуется сохранить имя по умолчанию, но

в рамках данного руководства имя будет изменено, так как в этом же пакете будут созданы тесты JUnit 4, а имена тестовых классов должны быть различными.

1. В списке "Среда" выберите JUnit.
2. Снимите флажки "Test Initializer" и "Test Finalizer". Нажмите кнопку "OK".

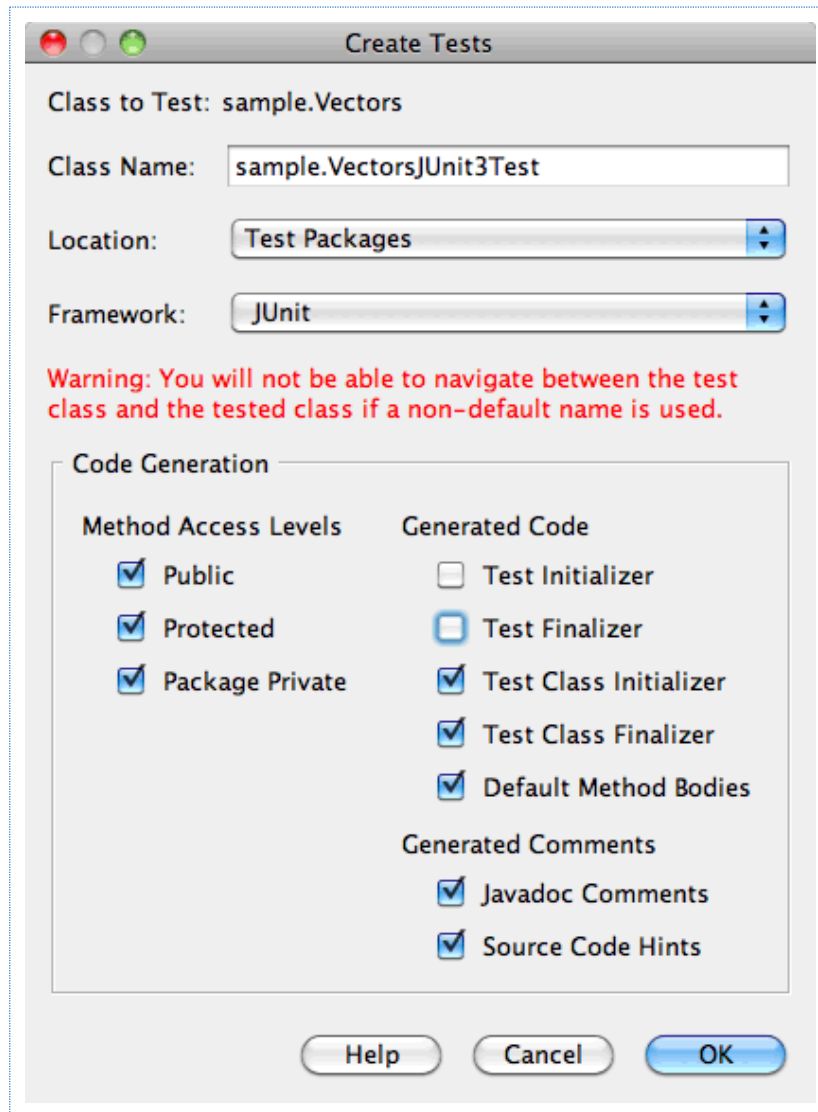


Figure 3. Диалоговое окно 'Выбрать версию JUnit'

1. В диалоговом окне "Select JUnit Version" выберите JUnit 3.x.

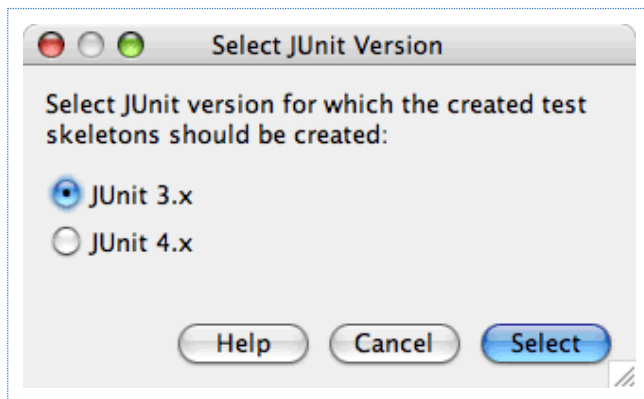


Figure 4. Диалоговое окно 'Выбрать версию JUnit'

Если выбрана версия JUnit 3.x, среда IDE добавляет в проект библиотеку JUnit 3.

При нажатии кнопки "Выбрать" среда IDE создает тестовый класс `VectorsJUnit3Test.java` в пакете `sample` под узлом "Тестовые пакеты" в окне "Проекты".

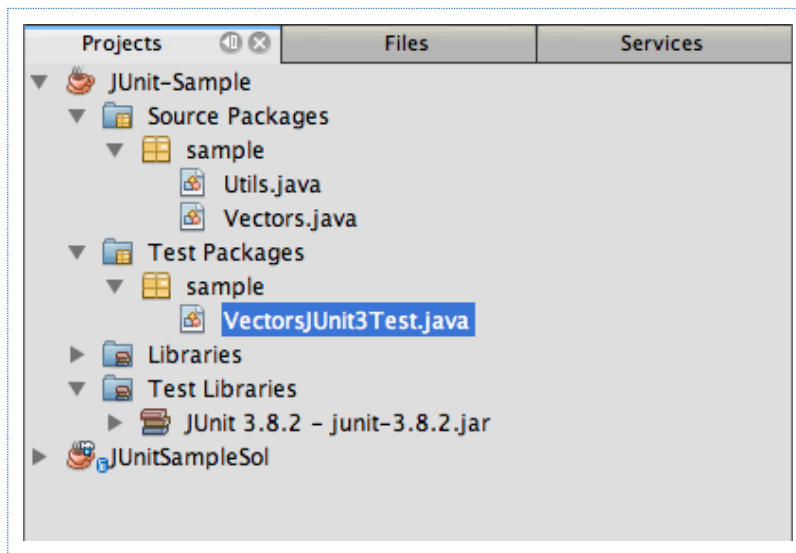


Figure 5. структура проекта JUnit-Sample в окне 'Проекты'

Для создания тестов в пакетах тестов необходимо указать каталог. Местоположение по умолчанию для каталога пакетов тестов находится на корневом уровне проекта, но в зависимости от типа проекта можно указать другое местоположение для каталога в диалоговом окне свойств проекта "Properties".

Анализ созданного тестового класса `VectorsJUnit3Test.java` в редакторе показывает, что в среде IDE был создан

следующий тестовый класс с тестовыми методами для методов `equal` и `scalarMultiplication` .

```
public class VectorsJUnit3Test extends TestCase {
    /**
     * Test of equal method, of class Vectors.
     */
    public void testEqual() {
        System.out.println("equal");
        int[] a = null;
        int[] b = null;
        boolean expectedResult = false;
        boolean result = Vectors.equal(a, b);
        assertEquals(expectedResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    /**
     * Test of scalarMultiplication method, of class Vectors.
     */
    public void testScalarMultiplication() {
        System.out.println("scalarMultiplication");
        int[] a = null;
        int[] b = null;
        int expectedResult = 0;
        int result = Vectors.scalarMultiplication(a, b);
        assertEquals(expectedResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }
}
```

Тело метода каждого созданного теста приводится исключительно в учебных целях и для рассматриваемого теста должно быть изменено. Если автоматическое создание кода не требуется, можно снять флажок "Default Method Bodies" в диалоговом окне "Create Tests".

При создании средой IDE имен для тестовых методов каждому имени метода предшествует слово `test` , так как в

JUnit 3 для определения тестов используются правила именования и отражения. Чтобы тестовые методы могли быть определены, имя каждого из них должно соответствовать синтаксису `test_<NAME>_`.

Примечание. В JUnit 4 не требуется использовать этот синтаксис для имен тестовых методов, так как для идентификации тестовых методов можно применять аннотации, а тестовый класс больше не используется для расширения `TestCase`.

Написание тестовых методов для `Vectors.java`

В этом упражнении созданные тестовые методы будут изменены для обеспечения их функционирования, а также будут изменены выходные сообщения по умолчанию. Изменять выходные сообщения для выполнения тестов не требуется, но может потребоваться их изменение для идентификации результатов, отображаемых в окне вывода "JUnit Test Results".

1. Откройте файл `VectorsJUnit3Test.java` в редакторе.
2. Измените скелет теста для `testScalarMultiplication` путем изменения значения `println` и удаления созданных переменных. После этого тестовый метод должен выглядеть следующим образом (изменения выделены полужирным шрифтом):

```
public void testScalarMultiplication() {  
    System.out.println("** VectorsJUnit3Test: testScalarMultiplication()*");  
    assertEquals(expResult, result);  
}
```

1. Затем добавьте несколько подтверждений для тестирования метода.

```
public void testScalarMultiplication() {
    System.out.println("* VectorsJUnit3Test: testScalarMultiplication()");
    *assertEquals( 0, Vectors.scalarMultiplication(new int[] { 0, 0}, new int[] { 0, 0}));
    assertEquals( 39, Vectors.scalarMultiplication(new int[] { 3, 4}, new int[] { 5, 6}));
    assertEquals(-39, Vectors.scalarMultiplication(new int[] {-3, 4}, new int[] { 5,-6}));
    assertEquals( 0, Vectors.scalarMultiplication(new int[] { 5, 9}, new int[] {-9, 5}));
    assertEquals(100, Vectors.scalarMultiplication(new int[] { 6, 8}, new int[] { 6, 8}));*
}
```

В этом тестовом методе используется метод JUnit `assertEquals`. Для использования утверждения необходимо указать входные переменные и ожидаемый результат. Для успешного прохождения теста метод теста должен выдать все ожидаемые результаты на основе переменных, введенных при выполнении тестового метода. Для охвата возможных перестановок следует добавить достаточное количество утверждений.

1. Измените скелет теста для `testEqual` путем удаления созданных тел методов и добавления следующего `println`.

```
*System.out.println("* VectorsJUnit3Test: testEqual()");*
```

Тестовый метод в результате должен выглядеть следующим образом:

```
public void testEqual() {
    System.out.println("* VectorsJUnit3Test: testEqual()");
}
```

1. Измените метод `testEqual` путем добавления следующих утверждений (выделены полужирным шрифтом).

```
public void testEqual() {
    System.out.println("* VectorsJUnit3Test: testEqual()");
    *assertTrue(Vectors.equal(new int[] {}, new int[] {}));
    assertTrue(Vectors.equal(new int[] {0}, new int[] {0}));
    assertTrue(Vectors.equal(new int[] {0, 0}, new int[] {0, 0}));
    assertTrue(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 0}));
    assertTrue(Vectors.equal(new int[] {5, 6, 7}, new int[] {5, 6, 7}));

    assertFalse(Vectors.equal(new int[] {}, new int[] {0}));
    assertFalse(Vectors.equal(new int[] {0}, new int[] {0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0, 0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0}));
    assertFalse(Vectors.equal(new int[] {0}, new int[] {}));

    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 1}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 1, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {1, 0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 1}, new int[] {0, 0, 3}));*
}
```

В этом тесте используются методы JUnit `assertTrue` и `assertFalse` для тестирования всех возможных результатов. Для успешного прохождения теста утверждения `assertTrue` должны быть истинными, а `assertFalse` – ложными.

1. Сохраните изменения.

Сравните: [Написание тестовых методов для Vectors.java \(JUnit 4\)](#)

Создание тестового класса для `Utils.java`

Теперь следует создать скелеты тестов для `Utils.java`. При создании теста в предыдущем упражнении в среде IDE запрашивалась версия JUnit. В этом случае запрос выбора версии не выводится.

1. Щелкните правой кнопкой мыши `Utils.java` и выберите "Сервис > Создать тесты".
2. В списке "Среда" выберите JUnit (если среда еще не выбрана).

3. В диалоговом окне установите флажки "Инициализатор теста" и "Финализатор теста"(если они еще не установлены).
4. Измените имя тестового класса на **UtilsJUnit3Test** в диалоговом окне "Create Tests". Нажмите кнопку "OK".

При нажатии кнопки "OK" в среде IDE создается файл теста `UtilsJUnit3Test.java` в каталоге "Test Packages > samples". Обратите внимание на то, что помимо создания скелетов тестов `testComputeFactorial` , `testConcatWords` и `testNormalizeWord` для методов в `Utils.java` в среде IDE также создаются методы инициализатора теста `setUp` и финализатора теста `tearDown` .

Написание тестовых методов для `Utils.java`

В этом упражнении будет добавлено несколько тестов, демонстрирующих общие принципы работы тестов JUnit. К методам также будет добавлен `println` , так как некоторые из методов не выводят данные по умолчанию. В результате добавления к методам `println` можно просмотреть окно результата тестирования JUnit для проверки выполнения методов и порядка их запуска.

Инициализаторы и финализаторы тестов

Методы `setUp` и `tearDown` используются для инициализации и финализации условий теста. Для тестирования `Utils.java` методы `setUp` и `tearDown` не требуются, они представлены здесь для демонстрации принципов их работы.

Метод `setUp` является методом инициализации теста и выполняется перед каждым тестом в классе теста. Для выполнения тестов метод инициализации теста не требуется, однако его следует использовать при необходимости инициализации некоторых переменных до выполнения теста.

Метод `tearDown` является методом финализатора теста и выполняется после каждого тестового примера в тестовом классе. Метод финализатора теста не требуется для выполнения тестов, однако он может использоваться для удаления всех данных, задействованных при выполнении тестов.

1. Внесите следующие изменения (выделены полужирным шрифтом) в код `println` каждого метода.

```
@Override
protected void setUp() throws Exception {
    super.setUp();
    *System.out.println("* UtilsJUnit3Test: setUp() method");*
}

@Override
protected void tearDown() throws Exception {
    super.tearDown();
    *System.out.println("* UtilsJUnit3Test: tearDown() method");*
}
```

При выполнении теста для каждого метода в окне вывода "Test Results" отображается текст `println` . Если код `println` не добавлен, окно результата выполнения методов не появится.

Тестирование с помощью простого подтверждения

Этот простой тест предназначен для тестирования метода `concatWords` . Вместо использования созданного метода теста `testConcatWords` будет добавлен новый метод теста с именем `testHelloWorld` , использующий единственное простое утверждение для проверки правильности сцепления строк методом. Для утверждения `assertEquals` в тесте используется синтаксис `assertEquals(EXPECTED_RESULT, ACTUAL_RESULT)` для проверки соответствия фактического результата ожидаемому результату. Если входные данные для метода `concatWords` – " Hello ", " , " , " world " и " ! " , то ожидаемый результат должен быть равен "Hello, world!" .

1. Удалите автоматически созданный тестовый метод `testConcatWords` из класса `UtilsJUnit3Test.java` .
2. Добавьте следующий метод для тестирования метода `concatWords` .**`public void testHelloWorld() {
 assertEquals("Hello, world!", Utils.concatWords("Hello", " , " , "world", " !"); }`**

1. Добавьте оператор `println` для вывода на экран текста о тесте в окне "JUnit Test Results".

```
public void testHelloWorld() {
    *System.out.println("* UtilsJUnit3Test: test method 1 - testHelloWorld()");*
    assertEquals("Hello, world!", Utils.concatWords("Hello", " , " , "world", " !"));
}
```

Сравните: [Тестирование с помощью простого утверждения \(JUnit 4\)](#).

Тестирование с использованием тайм-аута

Этот тест демонстрирует проверку метода на длительность выполнения. Если метод выполняется слишком долго, поток выполнения теста прерывается, а тест завершается сбоем. Можно указать предел времени для теста.

Тестовый метод вызывает метод `computeFactorial` в `Utils.java`. Можно предположить, что метод `computeFactorial` правилен, но в этом случае требуется его протестировать на выполнение вычисления за 1000 миллисекунд. Поток выполнения `computeFactorial` и поток выполнения теста запускаются одновременно. Поток выполнения теста останавливается через 1000 миллисекунд и выдает `TimeoutException`, если поток выполнения `computeFactorial` не завершается раньше. Потребуется добавить сообщение для его отображения при выдаче `TimeoutException`.

1. Удалите созданный тестовый метод `testComputeFactorial`.
2. Добавьте метод `testWithTimeout`, вычисляющий факториал случайного числа.

```
public void testWithTimeout() throws  
InterruptedException, TimeoutException { final int factorialOf = 1 + (int) (30000 * Math.random());  
System.out.println("computing " + factorialOf + "!");
```

```
Thread testThread = new Thread() {  
    public void run() {  
        System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf));  
    }  
};  
}*
```

1. Исправьте операторы импорта для импорта `java.util.concurrent.TimeoutException`.
2. Добавьте к методу следующий код (выделен полужирным шрифтом) для прерывания потока выполнения и вывода на экран сообщения в случае слишком долгого выполнения теста.

```
Thread testThread = new Thread() {  
    public void run() {  
        System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf));  
    }  
};  
  
*testThread.start();  
Thread.sleep(1000);  
testThread.interrupt();  
  
if (testThread.isInterrupted()) {  
    throw new TimeoutException("the test took too long to complete");  
}*  
}
```

Можно изменить строку `Thread.sleep` для изменения количества миллисекунд до выдачи тайм-аута.

1. Добавьте следующий код `println` (выделен полужирным шрифтом) для отображения текста о тесте в окне "JUnit Test Results".

```
public void testWithTimeout() throws InterruptedException, TimeoutException {  
    *System.out.println("* UtilsJUnit3Test: test method 2 - testWithTimeout()");*  
    final int factorialOf = 1 + (int) (30000 * Math.random());  
    System.out.println("computing " + factorialOf + '!');
```

Сравните: [Тестирование с использованием тайм-аута \(JUnit 4\)](#)

Тестирование на ожидаемое исключение

Этот тест предназначен для тестирования на ожидаемое исключение. Метод завершится сбоем, если не будет выдано ожидаемое исключение. В этом случае выполняется тестирование метода `computeFactorial` на результат `IllegalArgumentException` с отрицательной входной переменной (-5).

1. Добавьте следующий метод `testExpectedException` для вызова метода `computeFactorial` со входной

```
переменной -5.public void testExpectedException() { try { final int factorialOf = -5; System.out.println(factorialOf + " != " + Utils.computeFactorial(factorialOf)); fail("IllegalArgumentException was expected"); } catch (IllegalArgumentException ex) { } }
```

1. Добавьте следующий код `println` (выделен полужирным шрифтом) для отображения текста о тесте в окне "JUnit Test Results".

```
public void testExpectedException() {  
    *System.out.println("* UtilsJUnit3Test: test method 3 - testExpectedException()");*  
    try {
```

Сравните: [Тестирование на ожидаемое исключение \(JUnit 4\)](#)

Отключение теста

Этот тест включает способы временного отключения тестового метода. В JUnit 3 в качестве тестовых методов распознаются только методы с именем, начинающимся с `test`. В этом случае для отключения тестового метода к его имени добавляется приставка `DISABLED_`.

1. Удалите созданный тестовый метод `testNormalizeWord`.
2. Добавьте следующий тестовый метод к тестовму классу.**public void testTemporarilyDisabled() throws Exception { System.out.println(" UtilsJUnit3Test: test method 4 - checkExpectedException()"); assertEquals("Malm\u00f6", Utils.normalizeWord("Malmo\u0308")); }***

При выполнении тестового класса будет выполнен тестовый метод `testTemporarilyDisabled`.

1. Введите `DISABLED_` (выделено полужирным шрифтом) перед именем тестового метода.

```
public void *DISABLED_*testTemporarilyDisabled() throws Exception {  
    System.out.println("* UtilsJUnit3Test: test method 4 - checkExpectedException()");  
    assertEquals("Malm\u00f6", Utils.normalizeWord("Malmo\u0308"));  
    }
```


Сравните: [Отключение теста \(JUnit 4\)](#)

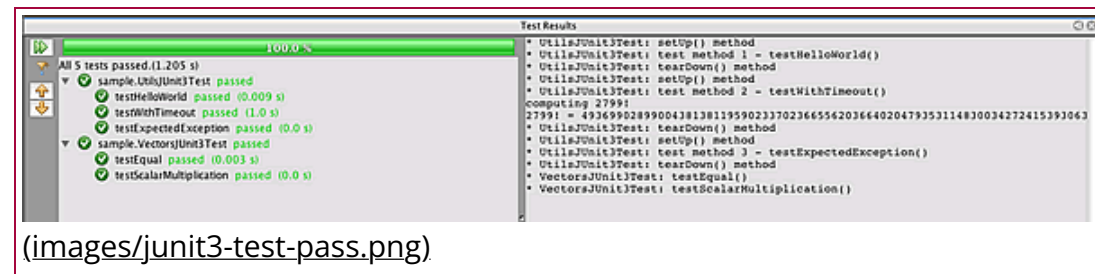
После написания всех тестов можно выполнить тест и посмотреть результат в окне "JUnit Test Results".

Выполнение тестов

При выполнении теста JUnit результаты отображаются в окне "Результаты теста JUnit" в среде IDE. Можно выполнить отдельные тестовые классы JUnit либо выбрать в главном меню "Run > Test *ИМЯ_ПРОЕКТА*" для выполнения всех тестов проекта. При выборе "Run > Test" в среде IDE выполняются все тестовые классы в папке "Test Packages". Для выполнения отдельного класса тестирования щелкните правой кнопкой мыши класс теста в узле 'Пакеты тестов' и выберите 'Выполнить файл'.

1. Выберите "Выполнить > Выбрать основной проект" в главном меню, затем выберите проект JUnit-Sample.
2. Выберите "Run > Test Project (JUnit-Sample)" из главного меню.
3. Выберите "Окно > Инструменты IDE > Результаты теста", чтобы открыть окно "Результаты теста".

При выполнении теста будет получен один из следующих результатов в окне "JUnit Test Results".



В примере на этом рисунке (для увеличения щелкните изображение) проект успешно прошел все тесты. На левой панели выводятся результаты отдельных тестовых методов, а на правой панели выводится результат теста. В окне вывода отображается порядок выполнения тестов. Добавление к каждому тестовому методу `println` обеспечивает вывод имени теста в окне вывода. Можно также отметить, что в `UtilJUnit3Test` метод `setUp` выполнялся перед каждым тестовым методом, а метод `tearDown` выполнялся после каждого метода.

В примере на этом рисунке (для увеличения щелкните изображение) тестирование проекта завершилось сбоем.

Выполнение метода `testTimeout` заняло слишком много времени, поэтому поток выполнения теста был прерван и явился причиной сбоя теста. Для вычисления факториала случайного числа потребовалось более 1000 миллисекунд (22,991).

Следующим действием после создания классов модульных тестов будет создание тестовых наборов. Описание способа запуска указанных тестов группой вместо выполнения каждого теста по отдельности приводится в разделе "Создание наборов тестов "JUnit 3".

Написание тестов JUnit 4

В этом упражнении будут созданы модульные тесты JUnit 4 для классов `Vectors.java` и `Utils.java`. Тесты JUnit 4 аналогичны тестам JUnit 3, однако при написании этих тестов применяется более простой синтаксис.

Для создания скелетов тестов на основе классов проекта будут использоваться мастера IDE. При первом использовании среды IDE для создания некоторых скелетов тестов будет выведен запрос на выбор версии JUnit.

Примечание. Если JUnit 3.x уже выбрана как версия по умолчанию для тестирования, необходимо изменить настройки по умолчанию на настройки версии JUnit 4.x. Чтобы изменить версию по умолчанию JUnit, разверните узел 'Библиотеки тестов', щелкните правой кнопкой мыши библиотеку JUnit и выберите 'Удалить'. Теперь можно использовать диалоговое окно "Добавить библиотеку", чтобы добавить библиотеку JUnit 4, или выбрать версию 4.x, если при создании нового теста требуется выбрать версию JUnit. Тесты JUnit 3 также можно будет выполнять, но для новых тестов будет использоваться JUnit 4.

Создание тестового класса для `Vectors.java`

В этом упражнении будут созданы скелеты теста JUnit для `Vectors.java`.

Примечание. Если используется NetBeans IDE 7.1 или более ранняя версия, указывать тип тестовой среды не требуется, так как JUnit выбран по умолчанию. В NetBeans IDE 7.2 можно выбрать в качестве тестовой среды JUnit или TestNG.

1. Щелкните правой кнопкой мыши `Vectors.java` и выберите "Сервис > Создать тесты".
2. В диалоговом окне "Create Tests" измените имя тестового класса на **VectorsJUnit4Test**.

В результате изменения имени тестового класса появится предупреждение об изменении имени. Имя по умолчанию – это имя тестируемого класса с добавленным словом "Test". Например, для класса `MyClass.java`

именем по умолчанию тестового класса будет `MyClassTest.java`. В отличие от JUnit 3, в JUnit 4 добавление слова "Test" к имени теста не обязательно. Рекомендуется сохранить имя по умолчанию, но так как в рамках данного руководства все тесты JUnit создаются в одном пакете, имена тестовых классов должны быть различны.

1. В списке "Среда" выберите JUnit.
2. Снимите флажки "Test Initializer" и "Test Finalizer". Нажмите кнопку "OK".

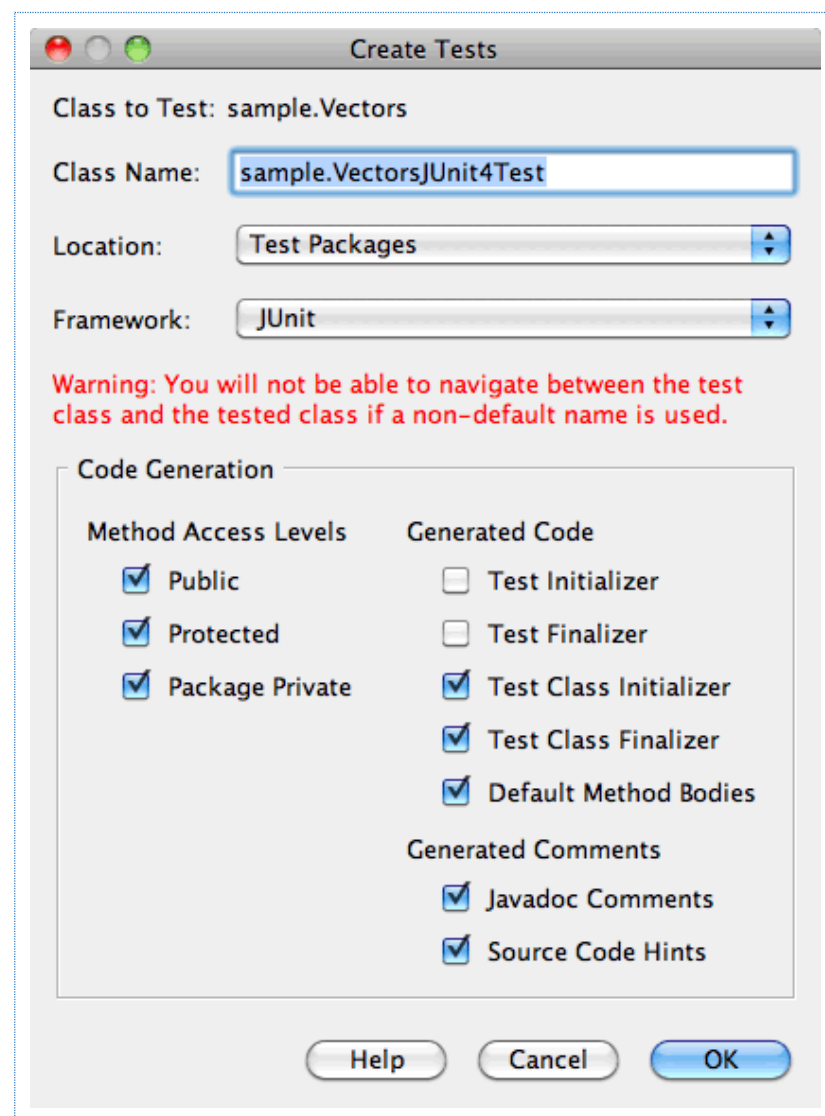


Figure 6. Диалоговое окно 'Создать тесты для JUnit 4'

1. В диалоговом окне "Select JUnit Version" выберите JUnit 4.x. Нажмите кнопку "Выбрать".

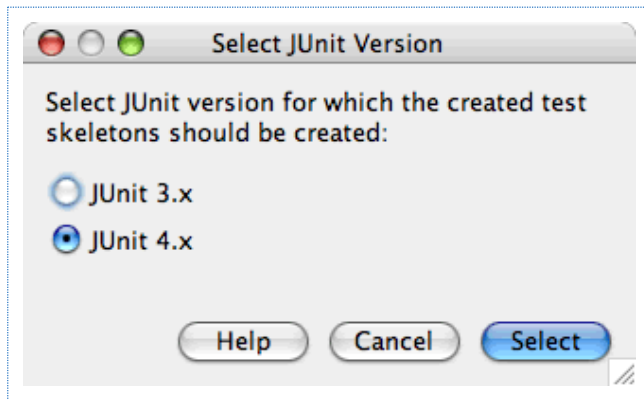


Figure 7. Диалоговое окно 'Выбрать версию JUnit'

При нажатии кнопки "ОК" среда IDE создает тестовый класс `VectorsJUnit4Test.java` в пакете `sample` под узлом "Тестовые пакеты" окна проектов.

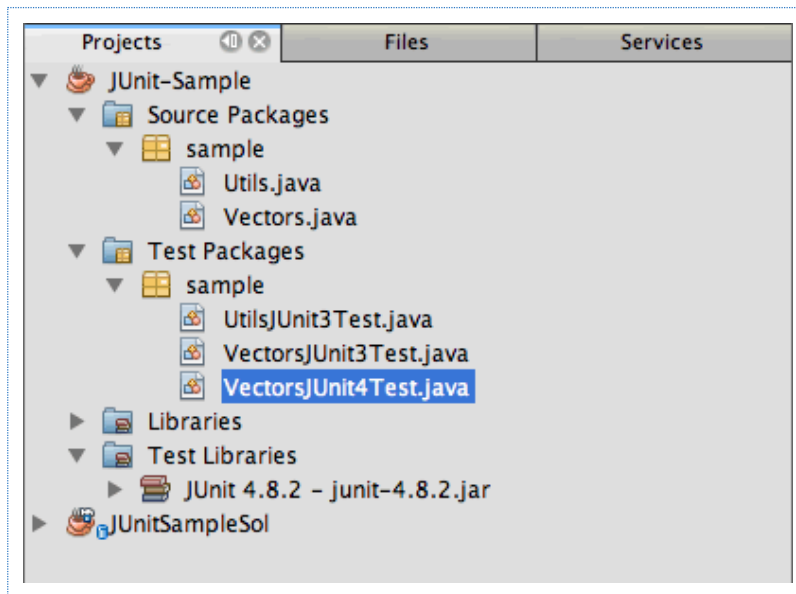


Figure 8. структура проекта JUnit-Sample с классами тестов JUnit 3 и JUnit 4

Примечание. Для создания тестов в пакетах тестов необходимо указать каталог. Местоположение по умолчанию для каталога пакетов тестов находится на корневом уровне проекта, но можно указать другое местоположение для каталога в диалоговом окне свойств проекта "Properties".

При просмотре в редакторе `VectorsJUnit3Test.java` можно отметить, что в среде IDE созданы тестовые методы `testEqual` и `testScalarMultiplication`. В `VectorsJUnit4Test.java` для каждого тестового метода используется аннотация `@Test`. В среде IDE имена для тестовых методов создаются на основе имен метода в `Vectors.java`, но к имени тестового метода не обязательно добавлять `test`. Тело по умолчанию каждого созданного тестового метода представлено исключительно в учебных целях и для фактического использования должно быть изменено.

Если автоматическое создание тел методов не требуется, можно снять флажок "Default Method Bodies" в диалоговом окне "Create Tests".

В среде IDE также создаются следующие методы инициализатора и финализатора классов теста:

```
@BeforeClass
public static void setUpClass() throws Exception {
}

@AfterClass
public static void tearDownClass() throws Exception {
}
```

В среде IDE методы инициализатора и финализатора классов создаются по умолчанию при создании класса теста `JUnit4`. Аннотации `@BeforeClass` и `@AfterClass` используются для выбора методов, которые должны быть запущены до и после выполнения тестового класса. Методы можно удалить, так как для тестирования `Vectors.java` они не нужны.

Также можно выполнить настройку методов, созданных по умолчанию при настройке свойств JUnit в окне "Options".

Примечание. Для тестов JUnit обратите внимание, что по умолчанию среда IDE добавляет статическое объявление импорта для `org.junit.Assert.*`.

Написание тестовых методов для `Vectors.java`

В этом упражнении будет изменен каждый из автоматически созданных тестовых методов для тестирования методов при помощи метода JUnit `assert` и изменения имен тестовых методов. JUnit 4 предоставляет

дополнительную гибкость при именовании тестовых методов, поскольку они определяются аннотацией `@Test` и не требуют добавления слова `test` к имени.

1. Откройте в редакторе `VectorsJUnit4Test.java` .
2. Измените тестовый метод для `testScalarMultiplication` путем изменения имени метода, значения `println` и удаления созданных переменных. После этого тестовый метод должен выглядеть следующим образом (изменения выделены полужирным шрифтом):

```
@Test
public void *ScalarMultiplicationCheck*() {
    System.out.println("** VectorsJUnit4Test: ScalarMultiplicationCheck()*");
    assertEquals(expResult, result);
}
```

Примечание. При написании тестов изменять результат вывода не требуется. В этом упражнении это выполнено для упрощения идентификации результатов тестирования в окне вывода.

1. Затем добавьте несколько подтверждений для тестирования метода.

```
@Test
public void ScalarMultiplicationCheck() {
    System.out.println("* VectorsJUnit4Test: ScalarMultiplicationCheck()");
    *assertEquals( 0, Vectors.scalarMultiplication(new int[] { 0, 0}, new int[] { 0, 0}));
    assertEquals( 39, Vectors.scalarMultiplication(new int[] { 3, 4}, new int[] { 5, 6}));
    assertEquals(-39, Vectors.scalarMultiplication(new int[] {-3, 4}, new int[] { 5,-6}));
    assertEquals( 0, Vectors.scalarMultiplication(new int[] { 5, 9}, new int[] {-9, 5}));
    assertEquals(100, Vectors.scalarMultiplication(new int[] { 6, 8}, new int[] { 6, 8}));*
}
```

В этом тестовом методе используется метод JUnit `assertEquals` . Для использования утверждения необходимо указать входные переменные и ожидаемый результат. Для успешного прохождения теста метод теста должен выдать все ожидаемые результаты на основе переменных, введенных при выполнении тестового метода. Для охвата возможных перестановок следует добавить достаточное количество утверждений.

1. Измените имя тестового метода с `testEqual` на `equalsCheck` .
2. Удалите созданное тело тестового метода `equalsCheck` .
3. Добавьте следующий метод `println` в тестовый метод `equalsCheck` .**`System.out.println(" VectorsJUnit4Test: equalsCheck()");`***

Тестовый метод в результате должен выглядеть следующим образом:

```
@Test
public void equalsCheck() {
    System.out.println("* VectorsJUnit4Test: equalsCheck()");
}
```

1. Измените метод `equalsCheck` путем добавления следующих утверждений (выделены полужирным шрифтом).

```
@Test
public void equalsCheck() {
    System.out.println("* VectorsJUnit4Test: equalsCheck()");
    *assertTrue(Vectors.equal(new int[] {}, new int[] {}));
    assertTrue(Vectors.equal(new int[] {0}, new int[] {0}));
    assertTrue(Vectors.equal(new int[] {0, 0}, new int[] {0, 0}));
    assertTrue(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 0}));
    assertTrue(Vectors.equal(new int[] {5, 6, 7}, new int[] {5, 6, 7}));

    assertFalse(Vectors.equal(new int[] {}, new int[] {0}));
    assertFalse(Vectors.equal(new int[] {0}, new int[] {0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0, 0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0}, new int[] {0}));
    assertFalse(Vectors.equal(new int[] {0}, new int[] {}));

    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 0, 1}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {0, 1, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 0}, new int[] {1, 0, 0}));
    assertFalse(Vectors.equal(new int[] {0, 0, 1}, new int[] {0, 0, 3}));*
}
```

В этом тесте используются методы JUnit `assertTrue` и `assertFalse` для тестирования всех возможных результатов. Для успешного прохождения теста утверждения `assertTrue` должны быть истинными, а `assertFalse` – ложными.

Сравните: [Написание тестовых методов для Vectors.java \(JUnit 3\)](#)

Создание тестового класса для `Utils.java`

Теперь перейдем к созданию тестовых методов JUnit для `Utils.java`. При создании тестового класса в предыдущем упражнении в среде IDE выводился запрос на выбор версии JUnit. В этот раз выбирать версию не потребуется, так как версия JUnit уже определена, и все последующие тесты JUnit будут созданы с использованием этой версии.

Примечание. Если выбрана версия JUnit 4, написание и выполнение тестов JUnit 3 также допустимо, но в среде IDE

для создания скелетов теста используется шаблон JUnit 4.

1. Щелкните правой кнопкой мыши `Utils.java` и выберите "Сервис > Создать тесты".
2. В списке "Среда" выберите JUnit (если среда еще не выбрана).
3. В диалоговом окне установите флажки "Инициализатор теста" и "Финализатор теста"(если они еще не установлены).
4. В диалоговом окне "Create Tests" измените имя тестового класса на **UtilsJUnit4Test**. Нажмите кнопку "OK".

При нажатии кнопки "OK" в среде IDE создается тестовый файл `UtilsJUnit4Test.java` в каталоге "Тестовые пакеты" > демонстрационный каталог. При этом в среде IDE будут созданы тестовые методы `testComputeFactorial`, `testConcatWords` и `testNormalizeWord` для методов в `Utils.java`. В среде IDE также создаются методы инициализатора и финализатора для теста и тестового класса.

Написание тестовых методов для `Utils.java`

В этом упражнении будет добавлено несколько тестов, демонстрирующих общие элементы теста JUnit. Также необходимо добавить в методы `println`, так как выполнение некоторых методов не приводит к отображению в окне "JUnit Test Results" информации, указывающей на выполнение или успешное прохождение теста. Добавление `println` в методы позволит отслеживать процесс выполнения методов и порядок их выполнения.

Инициализаторы и финализаторы тестов

При создании тестового класса для `Utils.java` в среде IDE создаются аннотированные методы инициализатора и финализатора. В качестве имени метода можно выбрать любое имя, так как обязательных требований в отношении имен не существует.

Примечание. Для тестирования `Utils.java` не требуются методы инициализатора и финализатора, но они рассматриваются в этом руководстве для демонстрации принципов их работы.

В JUnit 4 для обозначения следующих типов методов инициализатора и финализатора можно использовать аннотации.

- **Инициализатор класса тестов.** Аннотация `@BeforeClass` отмечает метод как метод инициализации класса теста. Метод инициализации тестового класса запускается только один раз и выполняется только перед выполнением любых других методов в тестовом классе. Например, вместо создания подключения к базе

данных в инициализаторе теста и создания нового подключения перед каждым тестовым методом можно использовать инициализатор тестового класса для открытия подключения перед выполнением тестов. Затем можно закрыть подключение в финализаторе тестового класса.

- **Финализатор класса тестов.** Аннотация `@AfterClass` помечает метод как метод финализации класса теста. Метод финализатора тестового класса выполняется только один раз и только после выполнения других методов в тестовом классе.
- **Инициализатор теста.** Аннотация `@Before` отмечает метод как метод инициализации теста. Метод инициализации теста выполняется перед каждым тестом в тестовом классе. Для выполнения тестов метод инициализации теста не требуется, однако его следует использовать при необходимости инициализации некоторых переменных до выполнения теста.
- **Финализатор теста.** Аннотация `@After` помечает метод как метод финализации теста. Метод финализатора теста выполняется после каждого теста в тестовом классе. Метод финализатора теста не требуется для выполнения тестов, но финализатор может использоваться для удаления всех данных, задействованных при выполнении тестов.

Измените следующее (выделено полужирным шрифтом) в `UtilsJUnit4Test.java` .

```
@BeforeClass
public static void setUpClass() throws Exception {
    *System.out.println("* UtilsJUnit4Test: @BeforeClass method");*
}

@AfterClass
public static void tearDownClass() throws Exception {
    *System.out.println("* UtilsJUnit4Test: @AfterClass method");*
}

@Before
public void setUp() {
    *System.out.println("* UtilsJUnit4Test: @Before method");*
}

@After
public void tearDown() {
    *System.out.println("* UtilsJUnit4Test: @After method");*
}
```

Сравните: Инициализаторы и финализаторы тестов (JUnit 3)

При выполнении тестового класса добавленный ранее текст `println` отображается в окне вывода "JUnit Test Results". Таким образом, информация, указывающая на выполнение методов инициализатора и финализатора, выводится только в том случае, если был добавлен `println`.

Тестирование с помощью простого подтверждения

Этот простой тест предназначен для тестирования метода `concatWords`. Вместо использования созданного тестового метода `testConcatWords` будет добавлен новый тестовый метод с именем `helloWorldCheck`, использующий единственное простое утверждение для проверки правильности сцепления строк методом. Для утверждения `assertEquals` в тесте используется синтаксис `assertEquals(EXPECTED_RESULT, ACTUAL_RESULT)` для проверки соответствия фактического результата ожидаемому результату. Если входные данные для метода `concatWords` – " Hello ", " , ", " world " и " ! ", то ожидаемый результат должен быть равен "Hello, world!" .

1. Удалите созданный тестовый метод `testConcatWords`.

2. Добавьте следующий метод `helloWorldCheck` для тестирования `Utils.concatWords` .**@Test public void helloWorldCheck() { assertEquals("Hello, world!", Utils.concatWords("Hello", " ", " ", "world", "!")); }**
1. Добавьте оператор `println` для вывода на экран текста о тесте в окне "JUnit Test Results".

```
@Test
public void helloWorldCheck() {
    *System.out.println("* UtilsJUnit4Test: test method 1 - helloWorldCheck()");*
    assertEquals("Hello, world!", Utils.concatWords("Hello", " ", " ", "world", "!"));
}
```

Сравните: [Тестирование с помощью простого утверждения \(JUnit 3\)](#)

Тестирование с использованием тайм-аута

Этот тест демонстрирует проверку метода на длительность выполнения. Если метод выполняется слишком долго, поток выполнения теста прерывается, а тест завершается сбоем. Можно указать предел времени для теста.

Тестовый метод вызывает метод `computeFactorial` в `Utils.java` . Можно предположить, что метод `computeFactorial` правилен, но в этом случае требуется его протестировать на выполнение вычисления за 1000 миллисекунд. Это выполняется путем прерывания потока выполнения теста через 1000 миллисекунд. При прерывании потока выполнения тестовый метод выдает `TimeoutException` .

1. Удалите созданный тестовый метод `testComputeFactorial` .
2. Добавьте метод `testWithTimeout` , вычисляющий факториал случайного числа.**@Test public void testWithTimeout() { final int factorialOf = 1 + (int) (30000 * Math.random()); System.out.println("computing " + factorialOf + "!"); System.out.println(factorialOf + "! = " + Utils.computeFactorial(factorialOf)); }**
 1. Добавьте следующий код (выделен полужирным шрифтом) для определения тайм-аута и прерывания потока выполнения в случае слишком долгого выполнения метода.

```
@Test*(timeout=1000)*
public void testWithTimeout() {
    final int factorialOf = 1 + (int) (30000 * Math.random());
```

Как видно в примере, для тайм-аута установлено значение 1000 миллисекунд.

1. Добавьте следующий код `println` (выделен полужирным шрифтом) для отображения текста о тесте в окне "JUnit Test Results".

```
@Test(timeout=1000)
public void testWithTimeout() {
    *System.out.println("* UtilsJUnit4Test: test method 2 - testWithTimeout()");*
    final int factorialOf = 1 + (int) (30000 * Math.random());
    System.out.println("computing " + factorialOf + '!');
```

Сравните: [Тестирование с использованием тайм-аута \(JUnit 3\)](#)

Тестирование на ожидаемое исключение

Этот тест предназначен для тестирования на ожидаемое исключение. Метод завершится сбоем, если не будет выдано ожидаемое исключение. В этом случае выполняется тестирование метода `computeFactorial` на результат `IllegalArgumentException` с отрицательной входной переменной (-5).

1. Добавьте следующий метод `testExpectedException` для вызова метода `computeFactorial` со входной переменной -5. **@Test public void checkExpectedException() { final int factorialOf = -5; System.out.println(factorialOf + "!" + Utils.computeFactorial(factorialOf)); }**

1. Добавьте следующее свойство (выделено полужирным шрифтом) в аннотацию `@Test` для определения необходимости выдачи `IllegalArgumentException` в результате выполнения теста.

```
@Test*(expected=IllegalArgumentException.class)*
public void checkExpectedException() {
    final int factorial0f = -5;
    System.out.println(factorial0f + "! = " + Utils.computeFactorial(factorial0f));
}
```

1. Добавьте следующий код `println` (выделен полужирным шрифтом) для отображения текста о тесте в окне "JUnit Test Results".

```
@Test (expected=IllegalArgumentException.class)
public void checkExpectedException() {
    *System.out.println("* UtilsJUnit4Test: test method 3 - checkExpectedException()");*
    final int factorial0f = -5;
    System.out.println(factorial0f + "! = " + Utils.computeFactorial(factorial0f));
}
```

Сравните: [Тестирование на ожидаемое исключение \(JUnit 3\)](#)

Отключение теста

Этот тест включает способы временного отключения тестового метода. Для отключения теста в JUnit 4 следует добавить аннотацию `@Ignore` .

1. Удалите созданный тестовый метод `testNormalizeWord` .
2. Добавьте следующий тестовый метод к тестовму классу.**`@Test public void temporarilyDisabledTest() throws Exception { System.out.println(" UtilsJUnit4Test: test method 4 - checkExpectedException()"); assertEquals("Malm\u00f6", Utils.normalizeWord("Malmo\u0308")); }`***

При выполнении тестового класса будет выполнен тестовый метод `temporarilyDisabledTest` .

1. Для отключения теста добавьте аннотацию `@Ignore` (выделена полужирным шрифтом) над `@Test` .**`@Ignore`**

```
@Test
public void temporarilyDisabledTest() throws Exception {
    System.out.println("* UtilsJUnit4Test: test method 4 - checkExpectedException()");
    assertEquals("Malm\u00f6", Utils.normalizeWord("Malmo\u0308"));
}
```

1. Исправьте операторы импорта для импорта `org.junit.Ignore` .

Сравните: [Отключение теста \(JUnit 3\)](#)

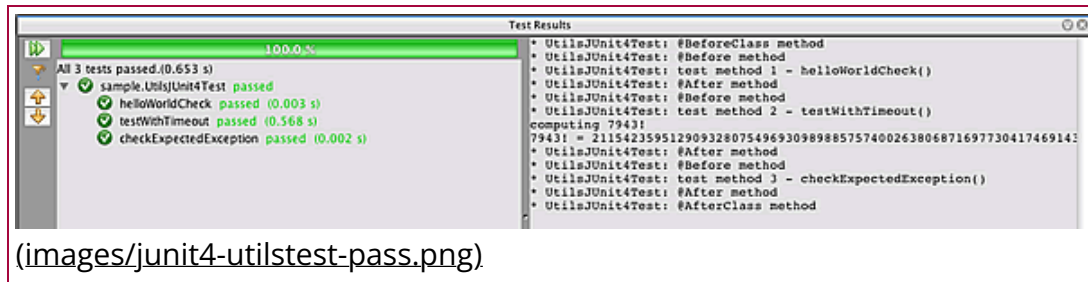
Теперь после написания всех тестов можно выполнить тест и просмотреть результат в окне "JUnit Test Results".

Выполнение тестов

В среде IDE можно выполнять тесты JUnit для всего приложения или для отдельных файлов и просматривать результаты. Самым простым способом выполнения всех модульных тестов для проекта является выбор в главном меню "Run > Test <ИМЯ_ПРОЕКТА>". При выборе этого метода в среде IDE запускаются все тестовые классы в папке с тестами. Для выполнения отдельного класса тестирования щелкните правой кнопкой мыши класс теста в узле 'Пакеты тестов' и выберите 'Выполнить файл'.

1. Щелкните правой кнопкой мыши `UtilsJUnit4Test.java` в окне "Projects".
2. Выберите тестовый файл.
3. Выберите "Окно > Инструменты IDE > Результаты теста", чтобы открыть окно "Результаты теста".

При запуске `UtilsJUnit4Test.java` в среде IDE выполняются только тесты в тестовом классе. На следующем рисунке представлен пример окна "JUnit Test Results" в случае успешного прохождения классом всех тестов.



В примере на рисунке (для увеличения щелкните изображение) в среде IDE был выполнен тест JUnit для `Utils.java`, при этом класс успешно прошел все тесты. На левой панели выводятся результаты отдельных тестовых методов, а на правой панели выводится результат теста. В окне вывода отображается порядок выполнения тестов. `println`, который был добавлен в тестовые методы, печатает имя теста в окне "Результаты тестирования" и в окне "Вывод".

В рассматриваемом примере в тесте `UtilsJUnit4Test` метод инициализатора тестового класса с аннотацией `@BeforeClass` был выполнен до выполнения всех других методов и только один раз. Метод финализатора тестового класса с аннотацией `@AfterClass` был выполнен последним, после выполнения всех остальных методов в классе. Метод инициализатора теста с аннотацией `@Before` выполнялся до выполнения каждого тестового метода.

Элементы управления в левой части окна "Результаты тестирования" позволяют легко перезапускать тесты. Чтобы переключаться между всеми результатами тестов и только сбойными тестами, можно использовать фильтр. Для того чтобы пропустить сбой и перейти к следующему, используются стрелки.

Если щелкнуть правой кнопкой мыши результат теста в окне 'Результаты теста', во всплывающем меню можно выбрать переход к источнику теста, повторное выполнение теста или отладку теста.

Следующее действие после создания классов модульного теста заключается в создании наборов тестов. Описание способа запуска указанных тестов группой вместо выполнения каждого теста по отдельности приводится в разделе [Создание наборов тестов "JUnit 4"](#).

Создание наборов тестов

При создании тестов для проекта обычно необходимо создать большое количество тестовых классов. При

выполнении тестовых классов по отдельности или запуске всех тестов проекта во многих случаях требуется выполнить определенное количество тестов или тесты в определенном порядке. Это можно осуществить путем создания одного или более набора тестов. Например, можно создать наборы тестов для тестирования определенных аспектов кода или конкретных условий.

Набор тестов, по сути, является классом, в который включен метод для вызова указанных тестов, например, определенных тестовых классов, тестовых методов в тестовых классах и других наборов тестов. Набор тестов может быть включен в тестовый класс, однако для набора тестов рекомендуется создать отдельные классы.

Наборы тестов JUnit 3 и JUnit 4 можно создать для проекта вручную или с использованием возможностей среды IDE. При использовании среды IDE для создания набора тестов по умолчанию в среде IDE генерируется код, вызывающий все тестовые классы в той же папке, где находится набор тестов. После создания набора тестов можно изменить класс для определения тестов, которые требуется выполнить в составе этого набора.

Создание набора тестов JUnit 3

При выборе JUnit 3 в качестве версии тестов в среде IDE могут быть созданы наборы тестов JUnit 3 на основе тестовых классов в папке с тестами. В JUnit 3 необходимо определить тестовые классы, которые должны быть включены в набор тестов, путем создания экземпляра `TestSuite` и использования метода `addTest` для каждого теста.

1. Щелкните правой кнопкой мыши узел проекта **JUnit-Sample** в окне проектов и выберите "Новый" > "Другие", чтобы открыть мастер создания файла.
2. В категории "Модульные тесты" выберите "Набор тестов". Нажмите кнопку "Далее".
3. Введите имя **JUnit3TestSuite** в качестве имени класса.
4. Выберите папку `sample` для создания набора тестов в типовой папке в папке с тестами.
5. Снимите флажки "Test Initializer" и "Test Finalizer". Нажмите кнопку "Завершить".

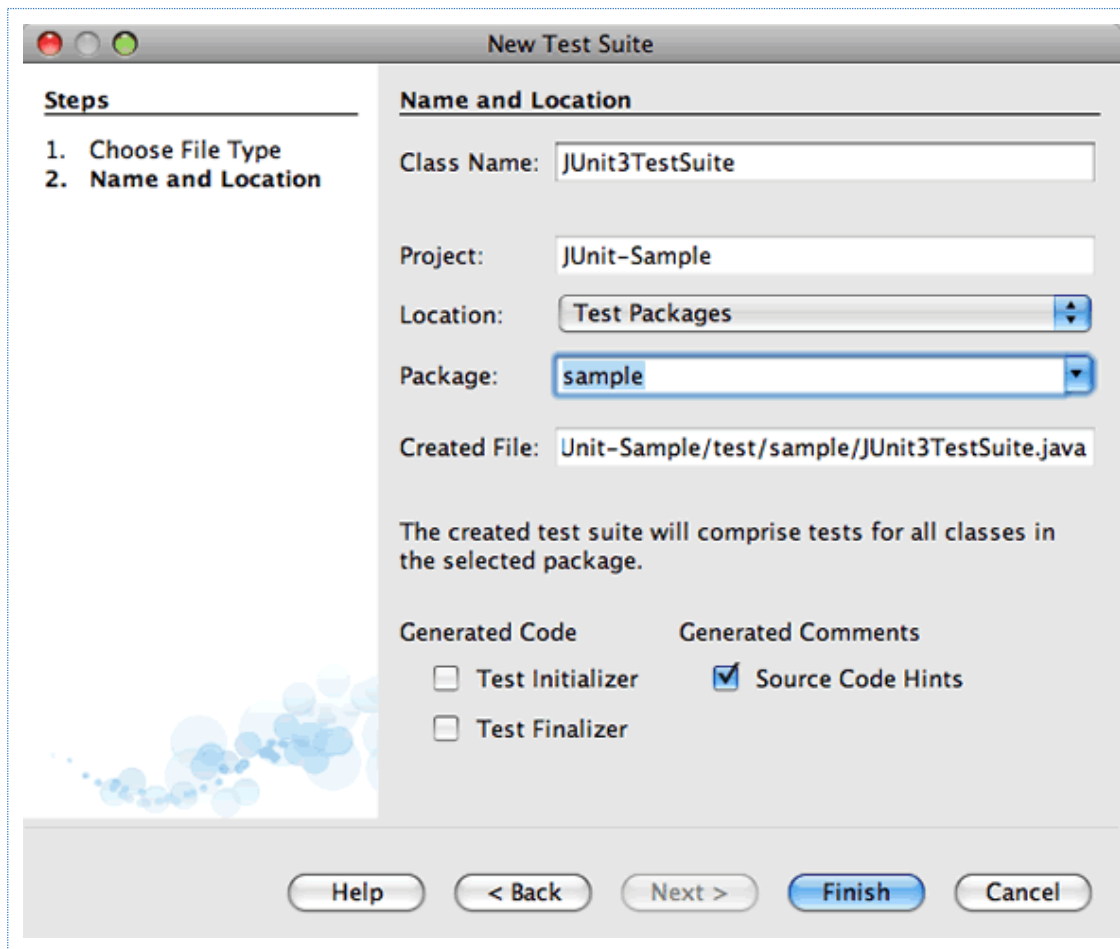


Figure 9. Мастер наборов тестов JUnit

При нажатии кнопки "Finish" в среде IDE создается класс набора тестов в папке `sample`, который затем открывается в редакторе. Тестовый набор будет содержать следующий код.

```
public class JUnit3TestSuite extends TestCase {
    public JUnit3TestSuite(String testName) {
        super(testName);
    }

    public static Test suite() {
        TestSuite suite = new TestSuite("JUnit3TestSuite");
        return suite;
    }
}
```

1. Измените метод `suite()` , чтобы добавить тестовые классы, которые будут запускать часть набора тестов.

```
public JUnit3TestSuite(String testName) {
    super(testName);
}

public static Test suite() {
    TestSuite suite = new TestSuite("JUnit3TestSuite");
    *suite.addTest(new TestSuite(sample.VectorsJUnit3Test.class));
    suite.addTest(new TestSuite(sample.UtillsJUnit3Test.class));*
    return suite;
}
```

1. Сохраните изменения.

Создание набора тестов JUnit 4

Если в качестве версии по умолчанию выбрана версия JUnit 4, в среде IDE могут быть созданы наборы тестов JUnit

4. Версия JUnit 4 совместима с предыдущими версиями, поэтому можно выполнять наборы тестов JUnit 4, содержащие тесты JUnit 4 и JUnit 3. В наборе тестов JUnit 4 указываются тестовые классы для включения их как значений аннотации `@Suite` .

Примечание. Для выполнения набора тестов JUnit 3 в составе набора тестов JUnit 4 требуется JUnit 4.4 или выше.

1. Щелкните правой кнопкой мыши узел проекта в окне "Projects" и выберите "New > Other" для открытия мастера создания файла.
2. В категории "Модульные тесты" выберите "Набор тестов". Нажмите кнопку "Далее".
3. Введите имя файла **JUnit4TestSuite**.
4. Выберите папку `sample` для создания набора тестов в типовой папке в папке с тестами.
5. Снимите флажки "Test Initializer" и "Test Finalizer". Нажмите кнопку "Завершить".

При нажатии кнопки "Finish" в среде IDE создается класс набора тестов в папке `sample`, который затем открывается в редакторе. Набор тестов содержит код, похожий на следующий:

```
@RunWith(Suite.class)
@Suite.SuiteClasses(value={UtilsJUnit4Test.class, VectorsJUnit4Test.class})
public class JUnit4TestSuite {
}
```

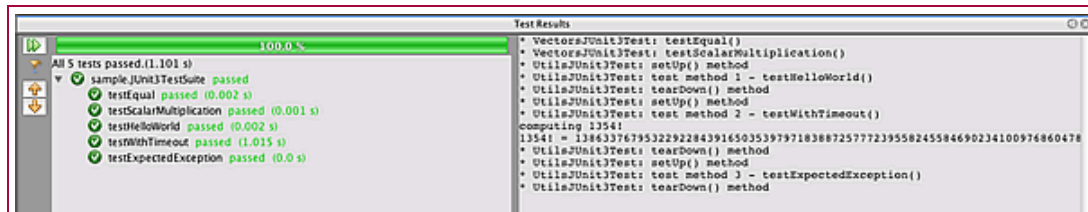
При запуске набора тестов среда IDE будет запускать тестовые классы в перечисленном порядке.

Выполнение наборов тестов

Набор тестов выполняется аналогично любому отдельному тестовому классу.

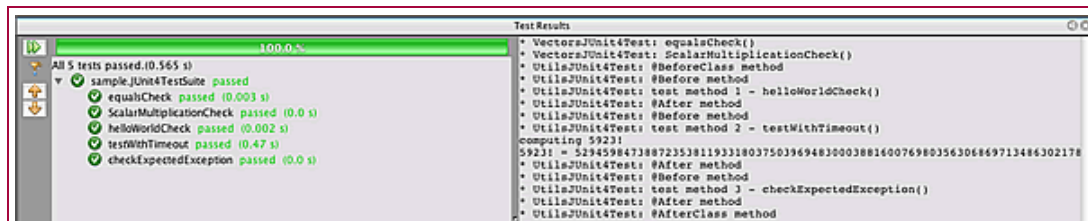
1. Разверните узел "Test Packages" в окне "Projects".
2. Щелкните правой кнопкой мыши класс набора тестов и выберите "Тестовый файл".

При выполнении набора тестов в среде IDE тесты, включенные в набор, выполняются в указанном порядке. Результаты отображаются в окне "JUnit Test Results".



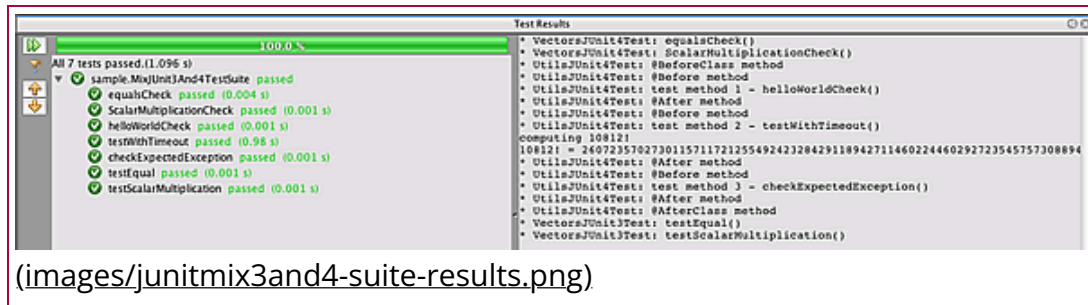
(images/junit3-suite-results.png)

В примере на этом рисунке (для увеличения щелкните изображение) в окне отображаются результаты набора тестов JUnit 3. В наборе тестов тестовые классы `UtilsJUnit3Test` и `VectorsJUnit3Test` были выполнены как один тест, и результаты выведены на экран в левой панели как результаты одного теста. Данные в правой панели представляют собой результат выполнения тестов по отдельности.



(images/junit4-suite-results.png)

В примере на этом рисунке (для увеличения щелкните изображение) в окне отображаются результаты набора тестов JUnit 4. В наборе тестов тестовые классы `UtilsJUnit4Test` и `VectorsJUnit4Test` были выполнены как один тест, и результаты выведены на экран в левой панели как результаты одного теста. Данные в правой панели представляют собой результат выполнения тестов по отдельности.



В примере на этом рисунке (для увеличения щелкните изображение) в окне отображаются результаты смешанного набора тестов. Этот набор тестов включает набор тестов JUnit 4 и один тестовый класс JUnit 3. В наборе тестов тестовые классы `UtilsJUnit3Test.java` и `JUnit4TestSuite.java` были выполнены как один тест, и результаты выведены на экран в левой панели как результаты одного теста. Данные в правой панели представляют собой результаты выполнения тестов по отдельности.

Заключение

Этот учебный курс представляет собой базовое введение в создание тестов JUnit и наборов тестов в IDE NetBeans. Среда IDE поддерживает JUnit 3 и JUnit 4, и данный документ продемонстрировал некоторые изменения в JUnit 4, разработанные для упрощения запуска и создания тестов.

Как было показано в данном руководстве, одним из главных усовершенствований JUnit 4 стала поддержка аннотаций. В JUnit 4 теперь можно использовать аннотации для следующего:

- Определять тест, используя аннотацию `@Test` вместо соглашения о присвоении имен.
- Определять методы `setUp` и `tearDown` аннотациями `@Before` и `@After`.
- Определять методы `setUp` и `tearDown`, которые присваиваются всему тестовому классу. Методы, которые аннотированы `@BeforeClass`, запускаются только один раз перед запуском всех тестовых методов класса. Методы, которые аннотированы `@AfterClass`, также запускаются только один раз, после завершения всех тестовых методов класса.
- Определять ожидаемые исключения
- Определять тесты, которые следует пропустить, с помощью аннотации `@Ignore`.
- Указывать тесту параметр "время ожидания".

Получить более подробную информацию об использовании JUnit и других изменениях в JUnit 4 можно в следующих материалах:

- [Группа JUnit в группах Yahoo](http://tech.groups.yahoo.com/group/junit/) (<http://tech.groups.yahoo.com/group/junit/>)
- www.junit.org (<http://www.junit.org>)

Тестирование кода позволяет убедиться в том, что небольшие изменения, внесенные в код, не вызовут сбой в работе приложения. Автоматизированные инструментальные средства тестирования, такие как JUnit, рационализируют процесс тестирования, а частое тестирование позволяет выявлять ошибки в коде на ранней стадии.


[Отправить отзыв по этому учебному курсу](https://netbeans.org/about/contact_form.html?to=3&subject=Feedback:%20Writing%20JUnit%20Tests%20in%20NetBeans%20IDE) (https://netbeans.org/about/contact_form.html?to=3&subject=Feedback:%20Writing%20JUnit%20Tests%20in%20NetBeans%20IDE)

Дополнительные сведения

Дополнительные сведения об использовании IDE NetBeans для разработки приложений Java см. следующие ресурсы:


- [Создание проектов Java](http://www.oracle.com/pls/topic/lookup?ctx=nb8000&id=NBDAG366) (<http://www.oracle.com/pls/topic/lookup?ctx=nb8000&id=NBDAG366>) в документе *Разработка приложений в IDE NetBeans*
- [Учебная карта по основам среды IDE и программирования на языке Java](http://trails.java-se.html) (<http://trails.java-se.html>)

 (<https://www.facebook.com/NetBeans>)

 (<https://twitter.com/netbeans>)

 (<https://github.com/apache/netbeans>)

 (<https://www.youtube.com/user/netbeansvideos>)

 (<https://tinyurl.com/netbeans-slack-signup/>)

 (<https://github.com/apache/netbeans/issues>)

 See this page in GitHub. (https://github.com/apache/netbeans-website/blob/master/netbeans.apache.org/src/content/kb/docs/java/junit-intro_ru.adoc)



(<https://www.apache.org/>)



[/events/current-event.html](https://www.apache.org/events/current-event.html))

(<https://www.apache.org>

ABOUT (/ABOUT /INDEX.HTML)

[Who's Who](https://netbeans.apache.org/community/who.html)

(<https://netbeans.apache.org/community/who.html>)

[Thanks](https://www.apache.org/foundation/thanks.html)

(<https://www.apache.org/foundation/thanks.html>)

[Sponsorship](https://www.apache.org/foundation/sponsorship.html)

(<https://www.apache.org/foundation/sponsorship.html>)

[Security](https://www.apache.org/security/)

(<https://www.apache.org/security/>)

COMMUNITY (/COMMUNITY /INDEX.HTML)

[Mailing lists](https://netbeans.apache.org/community/mailling-lists.html)

([/community/mailling-lists.html](https://netbeans.apache.org/community/mailling-lists.html))

[Becoming a committer](https://www.apache.org/community/commmitter.html)

([/community/commmitter.html](https://www.apache.org/community/commmitter.html))

[NetBeans Events](https://www.apache.org/community/events.html)

([/community/events.html](https://www.apache.org/community/events.html))

[Apache Events](https://www.apache.org/events/current-event.html)

(<https://www.apache.org/events/current-event.html>)

PARTICIPATE (/PARTICIPATE /INDEX.HTML)

[Submitting Pull](https://netbeans.apache.org/participate/submit-pr.html)

[Requests \(/participate /submit-pr.html\)](https://netbeans.apache.org/participate/submit-pr.html)

[Reporting Issues](https://netbeans.apache.org/participate/report-issue.html)

([/participate/report-issue.html](https://netbeans.apache.org/participate/report-issue.html))

[Improving the documentation](https://netbeans.apache.org/participate/documentation/)

([/participate /index.html#documentation](https://netbeans.apache.org/participate/documentation/))

GET HELP (/HELP /INDEX.HTML)

[Documentation \(/help](https://netbeans.apache.org/help/index.html#documentation)

[/index.html#documentation\)](https://netbeans.apache.org/help/index.html#documentation)

[Wiki \(/wiki/index.html\)](https://netbeans.apache.org/wiki/index.html)

[Community Support](https://netbeans.apache.org/help/index.html#support)

([/help /index.html#support](https://netbeans.apache.org/help/index.html#support))

[Commercial Support](https://netbeans.apache.org/help/commercial-support.html)

([/help/commercial-support.html](https://netbeans.apache.org/help/commercial-support.html))

DOWNLOAD (/DOWNLOAD /INDEX.HTML)

[Releases \(/download](https://netbeans.apache.org/download/index.html)

[/index.html\)](https://netbeans.apache.org/download/index.html)

[Plugins](https://plugins.netbeans.apache.org)

([https://plugins.netbeans.apache.o](https://plugins.netbeans.apache.org)

[Building from source](https://netbeans.apache.org/download/index.html#source)

([/download /index.html#source](https://netbeans.apache.org/download/index.html#source))

[Previous releases](https://netbeans.apache.org/download/index.html#previous)

([/download /index.html#previous](https://netbeans.apache.org/download/index.html#previous))

Copyright © 2017-2022 [The Apache Software Foundation](https://www.apache.org) (<https://www.apache.org>).

Licensed under the Apache [license](https://www.apache.org/licenses/) (<https://www.apache.org/licenses/>), version 2.0

Apache, Apache NetBeans, NetBeans, the Apache feather logo and the Apache NetBeans logo are trademarks of [The Apache Software Foundation](https://www.apache.org) (<https://www.apache.org>).

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The Apache NetBeans website conforms to the [Apache Software Foundation Privacy Policy](https://privacy.apache.org/policies/privacy-policy-public.html) (<https://privacy.apache.org/policies/privacy-policy-public.html>)