**Примеры Junit**

Когда делаешь первые попытки писать юнит-тесты обычно обычно сталкиваешься с проблемой начинания: вроде бы документация прочитана, цель ясна, а с чего начинать — не понятно.  
Попробуем вместе написать простой юнит-тест, для более-менее настоящего класса, в котором испытаем почти весь базовый функционал [JUnit](http://easyjava.ru/category/testing/junit/).

**Подготовка**

Создадим пустой maven проект:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >mvn archetype:generate -DgroupId=ru.easyjava.junit -DartifactId=base -Dversion=1  -DinteractiveMode=false  [INFO] Scanning for projects...  [...skipped...]  [INFO] ------------------------------------------------------------------------  [INFO] BUILD SUCCESS  [INFO] ------------------------------------------------------------------------ |

И добавим в него [JUnit](http://easyjava.ru/category/testing/junit/):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <properties>          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>          <junit.version>4.12</junit.version>      </properties>          <dependencies>          <dependency>              <groupId>junit</groupId>              <artifactId>junit</artifactId>              <version>${junit.version}</version>              <scope>test</scope>          </dependency>      </dependencies> |

Зависимость добавлена с  <scope>test</scope> , что говорит maven, что она требуется только при сборке и исполнении тестов.

**Класс для тестирования**

Поскольку я обещал почти настоящий пример, придётся написать хоть сколько-то полезный класс. Пускай это будем набор утилит для работы со строками:

* **Слияние строк из массива**
* **Разделение строки на подстроки**
* **Проверка строк на пустоту**
* **Преобразование в число и обратно**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | public final class StringUtils {        /\*\*       \* Do not construct me.       \*/      private StringUtils() { }        /\*\*       \* Объединяет массив строк в одну строку, вставляя разделители между элементами массива.       \* @параметр source это массив строк для присоединения.       \* @param del это разделитель для записей массива.       \* @ вернуть ноль, если массив равен нулю или объединить записи массива.       \*/      public static String joinArray(final String[] source, final char del) {          if (source = = null) {  return null;  }          StringBuilder result = new StringBuilder();          for (int i = 0; i < source.length - 1; i++) {              result.append(source[i]);              result.append(del);          }            result.append(source[source.length - 1]); //добавляет в строку последний элемент массива, но уже без разделителя            return result.toString();      }        /\*\*       \* Разбивает предоставленную строку на массив строк.       \* @параметр String source - массив для разделения.       \* @параметр delimiter Character – тот символ по которому будем делить.       \* @ вернуть пустой массив, если источник равен нулю, или массив подстрок, разделенных на символ разделителя.       \*/      public static String[] toArray(final String source, final char delimiter) {          if (source == null) {              return new String[]{};          }          return source.split(Character.toString(delimiter));      }        /\*\*       \* Проверяет, содержит ли строка какой-либо используемый контент (любые непустые символы).       \* @param subject String – строка для проверки.       \* @ вернуть true, если строка не имеет содержимого или содержит только пробельные символы, иначе false.       \*/      public static boolean isEmpty(final String subject) {          return subject = = null || subject.replaceAll("\\s", "").isEmpty();      }        /\*\*       \* Пытается извлечь double значение из String *(преобразование в число)*.       \* @параметр source String для обработки.       \* @возвращает извлеченное double значение или NaN если source равен null.       \*/      public static double toDouble(final String source) {          if (source = = null) {              return Double.NaN;          }          return Double.valueOf(source);      }        /\*\*       \* Конвертирует double значение в string *(обратное предыдущему преобразование)*.       \* @param source value to convert.       \* @return Textual representation of double.       \*/      public static String fromDouble(final double source) {          return String.valueOf(source);      }  } |

Не самый полезный набор утилит, но всё же

**Тесты**

Тесты в [JUnit](http://easyjava.ru/category/testing/junit/) располагаются в отдельных классах, методы которых, имеющие аннотацию  @Test, и возвращающие  void, и есть сами тесты. Имя класса может быть в принципе любое, но рекомендуется придерживаться шаблона  ИмяТестируемогоКлассаTest , так как это упрощает чтение кода. К тому же обычно средства автоматического запуска тестов, такие как плагин maven  maven-surefire-plugin предполагают, что классы с юнит-тестами оканчиваются на \*Test

|  |  |
| --- | --- |
|  | public class StringUtilsTest {      @Test      public void testToArray() {      }        @Test      public void testJoinArray() {      }        @Test      public void testIsEmpty() {      }        @Test      public void testToDouble() {      }        @Test      public void testFromDouble() {      }  } |

Maven традиционно располагает тесты в каталоге src/test , в то время как основной исходный код располагается в src/main . Разумеется это всего лишь договорённость, используемая в maven по умолчанию, и тесты и код можно располагать любым удобным образом.

Название тестовых методов так же могут быть любыми, однако для повышения читаемости кода, рекомендуется начинать их с префикса  test\*  и отражать в названии суть теста.

**Первый юнит-тест**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Test  public void testFromDouble() {      double source = 3.1415;      String expected="3.1415";        String actual = StringUtils.fromDouble(source);      assertEquals("Unexpected string value", expected, actual);  } |

Все юнит-тесты пишутся по единому шаблону: создаются входные данные, создаются эталонные данные (expected), вызывается тестируемый код и результат его работы(actual) сравнивается с эталонными данными. [JUnit](https://easyjava.ru/tag/junit/" \o "Posts tagged with junit) предоставляет несколько assert\*  функций, выполняюших сравнение.

В первом юнит-тесте строка

|  |  |
| --- | --- |
| 1 | double source = 3.1415; |

и есть входные данные, которые мы отдаём в проверяемую функцию.

Эталонные данные определены в следующей строке:

|  |  |
| --- | --- |
| 1 | String expected="3.1415"; |

Вызываем проверяемый код и сохраняем результат его работы:

|  |  |
| --- | --- |
| 1 | String actual = StringUtils.fromDouble(source); |

Наконец самая главная часть теста, проверка:

|  |  |
| --- | --- |
| 1 | assertEquals("Unexpected string value", expected, actual); |

assertEquals сравнивает эквивалентность объектов expected и actual и, в случае когда они не эквивалентны, проваливает тест и выводит сообщение "Unexpected string value". Функции assert\* можно использовать и без сообщения:

|  |  |
| --- | --- |
| 1 | assertEquals(expected, actual); |

Однако с сообщением результаты тестирования становятся гораздо приятнее при чтении.

Разработчики обычно пишут юнит-тесты только для предусмотренных разработчиком/архитектором/документацией/etc вариантов поведения функции. Для  
StringUtils.fromDouble()  документация указывает что функция должна преобразовать цисло с плавающей запятой в строку.  
Юнит-тест этой функции покрывает только описанный функционал. Цель юнит-тестирования — убедиться, что функция работает правильно, а не искать условия,  
в которых она работает неправильно.

Более того, сам юнит-тест уже является краткой и понятной документацией к функции. В четырёх строках чётко и однозначно написано, как ведёт себя функция: возвращает новый строковый объект, значение которого является переданным ей числом с плавающей запятой, записанное в десятичной системе счисления.

А самый главный бонус юнит-тестирования, это фиксация поведения кода. Вы знаете, прямо сейчас, что функция ведёт себя определённым образом. И код, который её использует, полагается на это поведение.  Когда вы захотите изменить эту функцию, юнит-тест будет вам гарантировать, что  
поведение функции осталось таким же (либо тест провалится). Следовательно остальной код не заметит изменения реализации функции, а это значит, что с этого момента вы можете спокойной менять любую часть кода: юнит-тесты не позволят вам что-нибудь сломать.

**Второй юнит-тест**

Следующий юнит-тест напишем для обратной функции преобразования строки в число с плавающей запятой:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Test      public void testToDouble() {          assertEquals(3.1415, StringUtils.toDouble("3.1415"), 0.0001);          assertEquals("Not NaN for null", Double.NaN, StringUtils.toDouble(null), 0.00001);      } |

Это оцень простая функция и поэтому тест тоже очень простой: входное значение, эталонное значение и сравнение с результатом.Обычно при написании таких простых тестов явно не заводят переменные для значений, а пишут их прямо в assert\* функции.

Однако у assertEquals в этом тесте появился дополнительный параметр! Дело в том, что сравнивать числа с плавающей запятой непосредственно друг с другом нельзя, так как они не имеют точного двоичного представления. Обычно числа сравниваются с некоторой погрешностью: можно сказать что 3.1415000000001 эквивалентно  
3.1415000000002 с погрешностью до 0.000000000001. И именно эта погрешность передаётся в третий параметр assertEquals для числе с плавающей запятой. Вторая часть теста очевидна — проверяется что для переданного null возвращается NaN.

**Третий юнит-тест**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Test      public void testIsEmpty() {          assertFalse("Non empty string claimed to be empty", StringUtils.isEmpty("TEST"));          assertTrue("Empty string not recognized", StringUtils.isEmpty(""));          assertTrue("Whitespaces not recognized",StringUtils.isEmpty(" "));      } |

Теперь наоборот — у функций assertTrue и assertFalse не хватает одного аргумента. А всё потому, что эти функции проверяют только логические значения (первая ожидает true, вторая, соответственно, false ), которые и не с чем сравнивать.  В самом же тесте проверяется документированние поведение:  строка с значением очевидно не пуста, строка без каких-либо символов в ней — пуста и, наконец, строка с невидимыми символами тоже признаётся пустой.

**Четвёртый и пятый юнит-тесты**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Test      public void testToArray() {          String[] expected = {"T", "E", "S", "T"};          String source="T:E:S:T";          assertArrayEquals("Wrong array", expected, StringUtils.toArray(source, ':'));          assertNull(StringUtils.toArray(null, ':'));      } |

В последних двух тестах нам придётся работать с массивами. Массивы нельзя проверить через assertEquals, так как для массивов assertEquals ведёт себя как assertSame:

|  |  |
| --- | --- |
| 1  2 | assertEquals(expected, StringUtils.toArray(source, ':'));  junit.framework.AssertionFailedError: expected:<[Ljava.lang.String;@61b383e9> but was:<[Ljava.lang.String;@5099681b> |

Поэтому в JUnit предусмотрена специальная функция для сравнения массивов assertArrayEquals, которая сравнивает эквивалентность каждого элемента обоих массивов друг с другом. Разумеется сравниваются между собой элементы с одинаковой позицией в массиве и массивы разной длины сразу признаются не эквивалентными. Надо отметить что обратной функции для assertArrayEquals не предусмотрено.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @Test  public void testToArray() {      String expected = "T:E:S:T";      String[] source = {"T", "E", "S", "T"};      assertArrayEquals("Wrong array", expected, StringUtils.toArray(source, ':'));      assertEquals(0,StringUtils.toArray(null, ':').length);  } |

[свернуть]

**Исполнение тестов**

Проще всего использовать для этого maven:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | >mvn test  [INFO] Scanning for projects...  [...skipped...]  [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ base ---    -------------------------------------------------------  T E S T S  -------------------------------------------------------  Running ru.easyjava.junit.StringUtilsTest  Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.066 sec    Results :    Tests run: 5, Failures: 0, Errors: 0, Skipped: 0  [INFO] BUILD SUCCESS |

Но при необходимости можно и вручную:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Z:\Dropbox\Work\MorningJava\Blog\Testing\JUnit\Base\manual>java -cp .;junit-4.12.jar;hamcrest-core-1.3.jar org.junit.runner.JUnitCore ru.easyjava.junit.StringUtilsTest  JUnit version 4.12  .....  Time: 0,02    OK (5 tests) |

Нужно всего лишь вручную указать правильный classpath, включающий в себя junit с зависимостиями и ваши классы.

Исходный код примера доступен на [github](https://github.com/EasyJavaRu/junit-base" \t "_blank)