



Разработка через тестирование XUnit

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Содержание

1

Семейство xUnit

2

Аннотации JUnit

3

Утверждения. Методы assert*

4

Метод fail

5

Работа с исключениями

6

Запуск тестов

7

Альтернативные фреймверки

- Как и в любом другом деле, в модульном тестировании не обойтись без подходящих инструментов – нет смысла «забивать гвозди микроскопом»
- Для этого существуют xUnit и Mock-фреймворки, применяемые для state-based и interaction тестирования соответственно

- Самыми яркими представителями семейства xUnit являются фреймворки JUnit (для Java) и его портированная под .NET версия – NUnit
- Синтаксис обоих фреймворков практически идентичен, поэтому рассмотрим аннотации и методы JUnit

- В ходе написания модульных тестов у нас появляются как сами тестовые классы и методы, так и вспомогательные
- Для их разделения в среде JUnit, начиная с 4-й версии, используются аннотации
- Аннотация – ключевое слово, начинающееся с символа “@”, и помещаемое перед объявлением класса и/или метода

Фикстура – разделяемые между тестами данные и бизнес-логика

Аннотации JUnit

@Before
@After
@BeforeClass
@AfterClass

- Объявляет метод фикстурой
- Данный метод будет вызван единожды, перед началом (после выполнения) тестового набора
- Используется для инициализации (очистки) тестовых данных и объектов

@Before

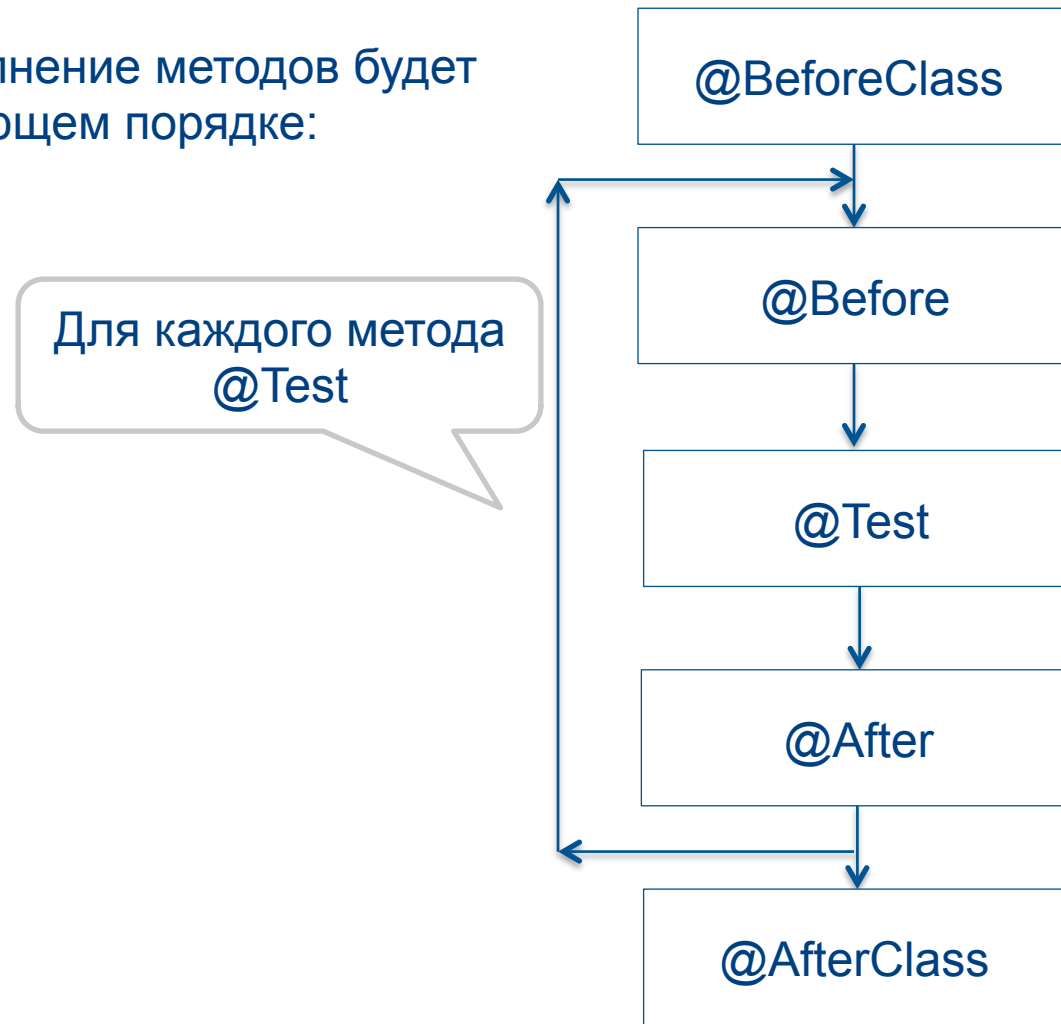
- Объявляет метод фикстурой
- Данный метод будет вызываться перед началом (после завершения) КАЖДОГО тестового метода

@Test

- Объявляет метод тестом

Аннотации JUnit

Таким образом, выполнение методов будет выполняться в следующем порядке:



Пример

```
public class MathTest {  
  
    @BeforeClass  
    public static void runsBeforeSuite() {  
        System.out.println("Я запустился перед выполнением тестов");  
    }  
  
    @Before  
    public void runsBeforeTest() {  
        System.out.println("Я запускаюсь перед каждым тестом");  
    }  
  
    @Test  
    public static void someTestMethod() {  
        assertEquals(4, 2 + 2);  
        System.out.println("Я тестовый метод 0.0");  
    }  
  
    @After  
    public void runsAfterTest() {  
        System.out.println("Я запускаюсь после каждого теста");  
    }  
}
```

Пример

Для создания тестовых наборов, используются аннотации `@RunWith` и `@SuiteClasses`, например:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses( {FirstTest.class,
                SecondTest.class,
                })
public class JUnit4Suite {}
```

Методы assert*

```
assertTrue(False)([message], condition)
```

Для проверки значений, возвращаемых тестируемыми методами, используются следующие методы JUnit:

- Проверка истинности выражения
- Опционально комментируется сообщением message

Методы assert*

```
assertEquals([message], obj1, obj2)
```

- проверка эквивалентности объектов
- перегружена для базовых классов
- опционально комментируется сообщением message

```
assert(Not)Null([message], obj)
```

- проверка на null
- опционально комментируется сообщением message

Методы assert*

Asserts

- assertEquals
- assertFalse
- assertNotNull
- assertNull
- assertNotSame
- assertSame
- assertTrue

TestCase

- run
- setUp
- tearDown

Annotations

- @Test
- @Before
- @After

- Итак, для написания модульного теста в среде JUnit нам необходимо:
 - создать класс
 - каждый тестовый случай описать в отдельном методе (с аннотацией `@Test`)
 - при необходимости написать методы инициализации / очистки (с аннотациями `@Before/After[Class]`)
- Рассмотрим небольшой пример тестирования математических операций

Пример использования JUnit



```
public class MathTest {  
    @Test  
    public void testEquals() {  
        assertEquals("Правильный результат сложения", 4, 2 + 2);  
        assertTrue(4 == 2 + 2);  
    }  
}
```


Пример использования JUnit

```
public class MathTest {  
    @Test  
    public void TestNotEquals() {  
        assertFalse(5 == 2 + 2);  
    }  
  
    @Test  
    public void DevizionByZeroTest() {  
        try {  
            int x = 1 / 0;  
            fail("Не поймано исключение при делении на 0");  
        } catch (Exception e) {  
            fail("Неожиданное исключение при делении на 0");  
        }  
    }  
}
```

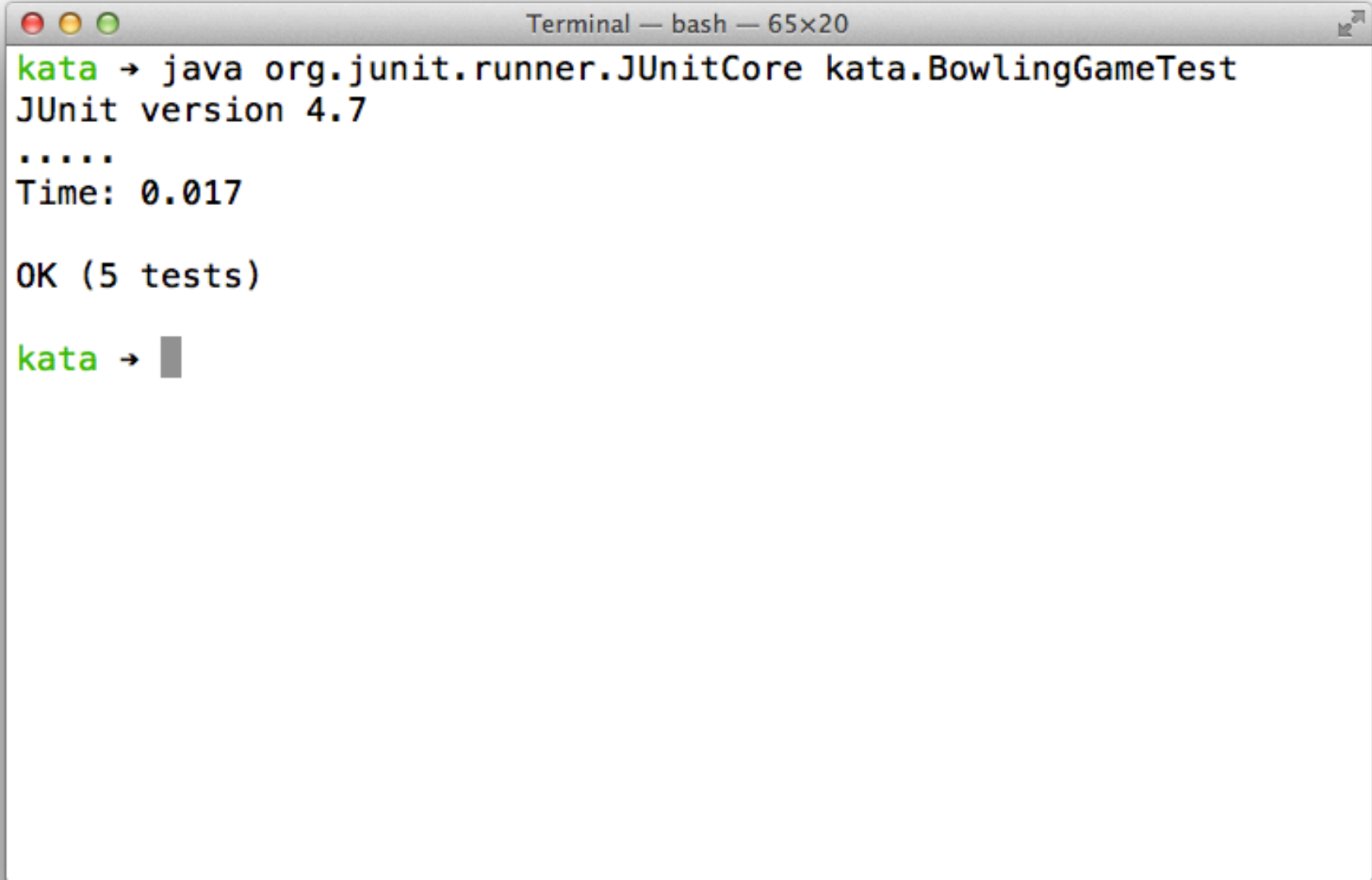
- Как мы видим, в последнем тестовом методе используется метод `fail([message]);`
- Этот метод используется для прямого сообщения фреймворку об ошибке (в данном случае – если не было вызвано ожидаемое исключение, или оно не того типа)

Запуск тестов из консоли



```
java -cp \  
C:\testing\lib\junit-4.5.jar; \  
C:\testing\bin org.junit.runner.JUnitCore \  
MathTestClass
```

Запуск тестов из консоли



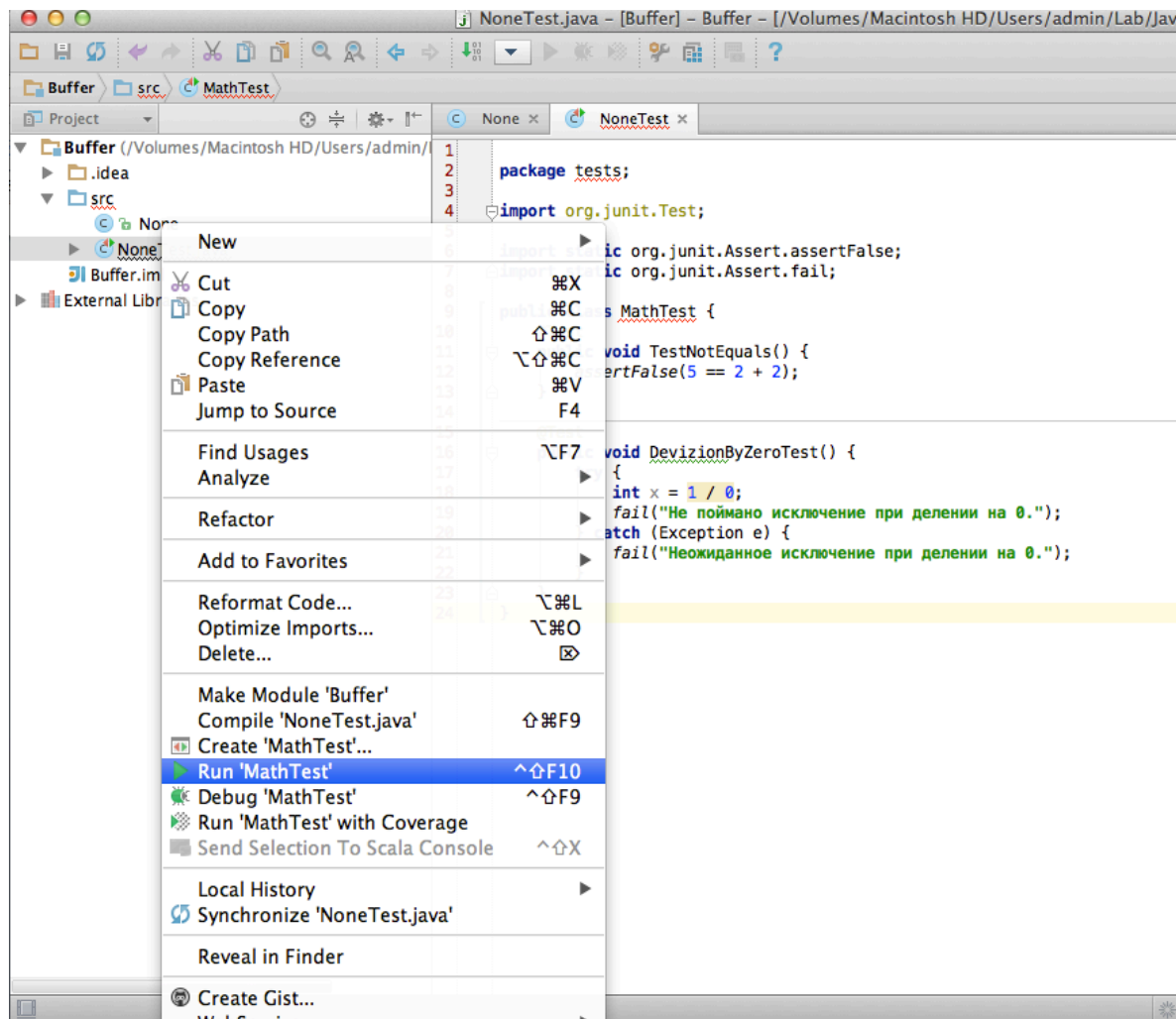
```
Terminal — bash — 65x20
kata → java org.junit.runner.JUnitCore kata.BowlingGameTest
JUnit version 4.7
.....
Time: 0.017

OK (5 tests)

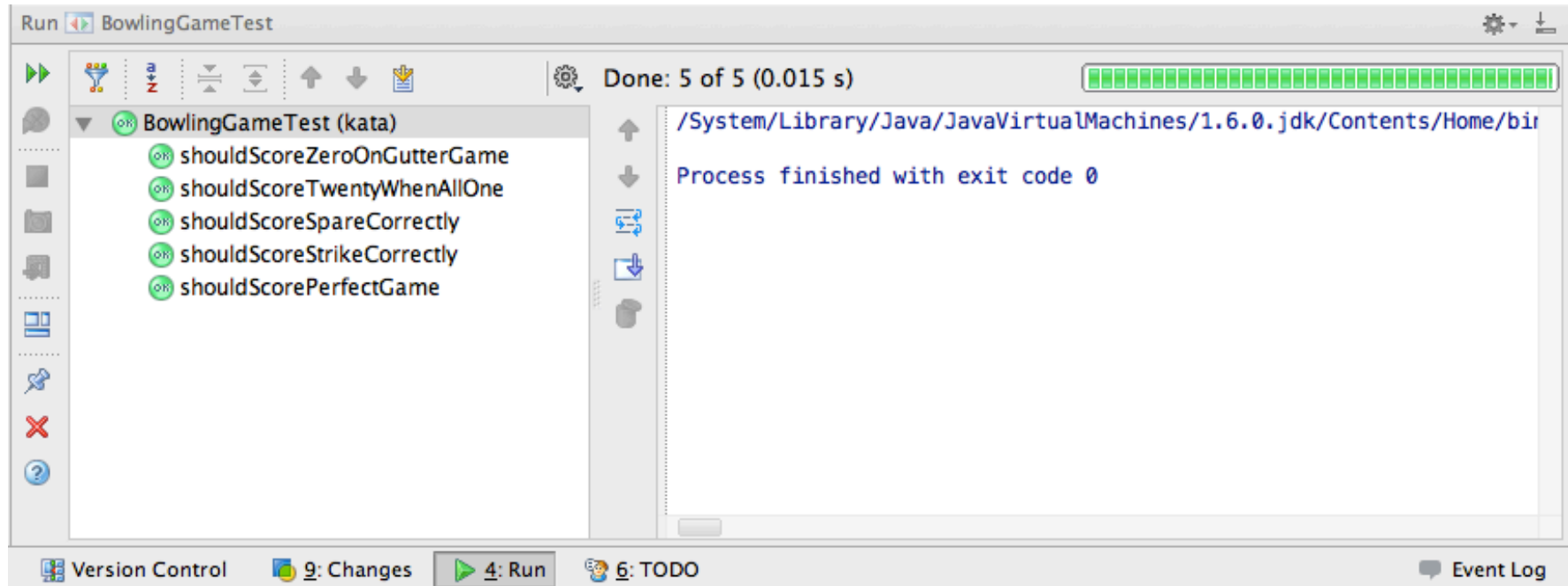
kata →
```

Запуск тестов в IDE

В среде Eclipse\IntelliJ IDEA достаточно кликнуть правой кнопкой на тестовом классе и выбрать Run As -> JUnit Test



Запуск тестов в IDE





Вопросы ?



Разработка через тестирование

IDyachenko@luxoft.com

```
git clone git://github.com/ivan-dyachenko/Trainings.git
```

```
https://github.com/ivan-dyachenko/Trainings
```