

Модульное тестирование в Java



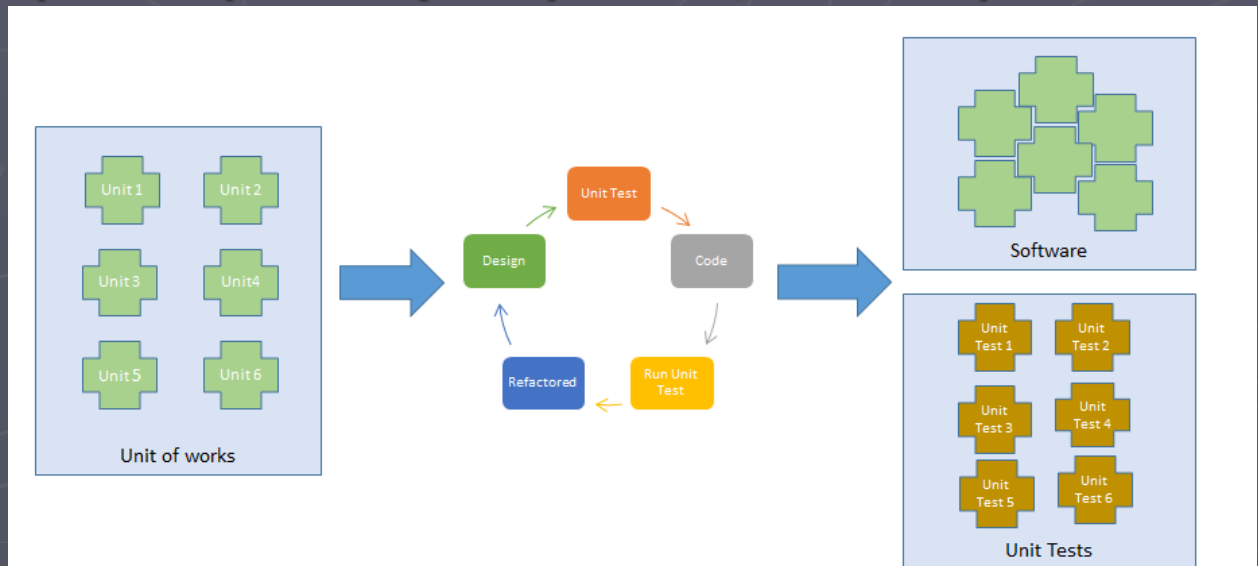
- Модульное тестирование (unit testing, юнит-тестирование) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

покрывают атомарные участки кода, что позволяет удостовериться в их работоспособности (в т.ч. после внесения изменений).



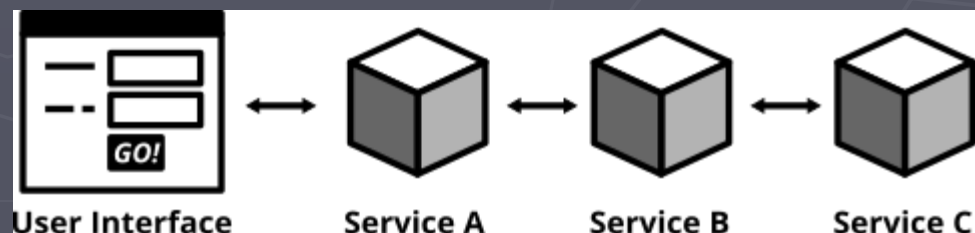
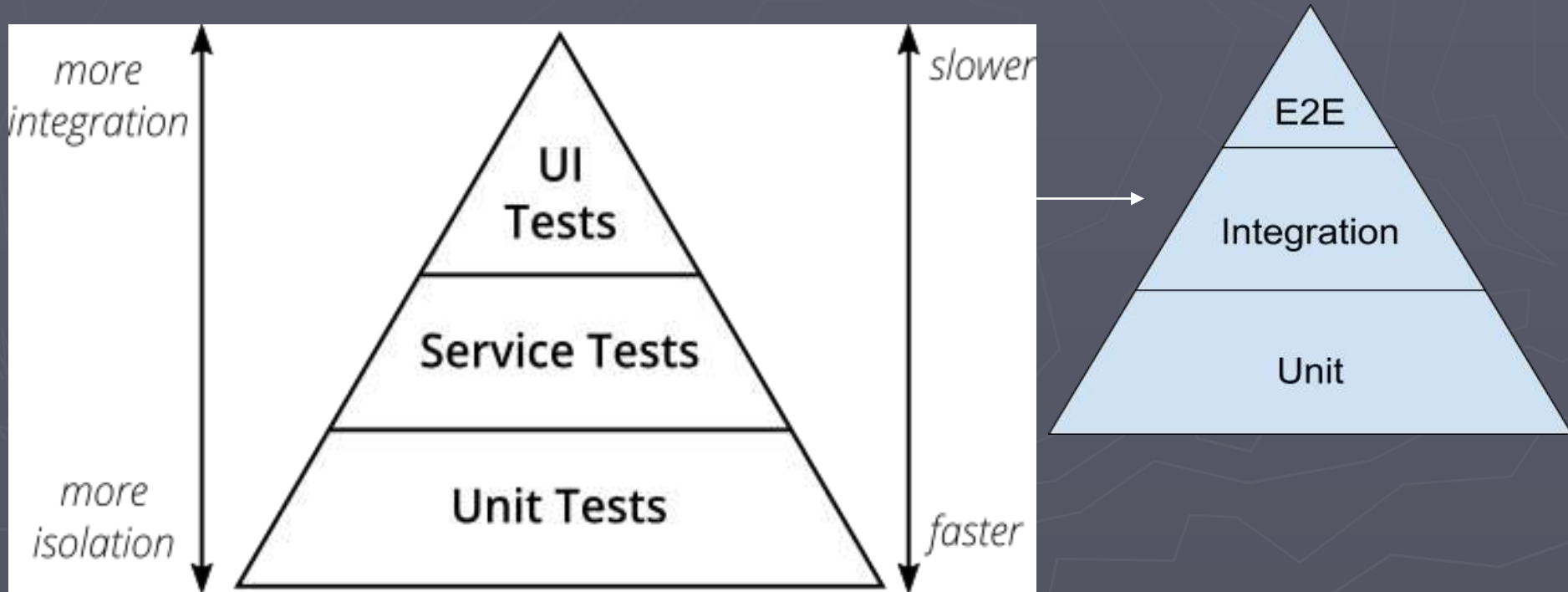
Назначение

- ▶ Улучшают качество архитектуры приложения.
- ▶ Стимулируют написание простых методов.
- ▶ Упрощают интеграцию кода.
- ▶ Помогают документированию кода.
- ▶ Минимизируют зависимости в системе.
- ▶ Отладка и рефакторинг (скорость, повторный запуск)



Виды тестирования

- ▶ Писать тесты разной детализации.
- ▶ Чем выше уровень, тем меньше тестов.



Что проверяют модульные тесты

► Тестируют

- Отдельные методы
- Взаимодействие объектов

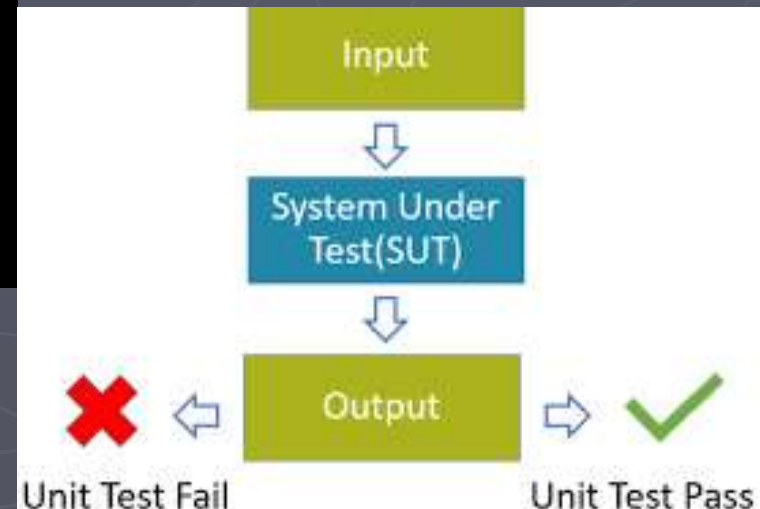
► Не тестируют

- Интеграцию компонентов
- Высокоуровневую логику приложения
- Пользовательский интерфейс

Концепция UNIT

- *Формы конечного результата:*
- возврат значения из функции
 - видимое изменение состояния или поведения системы
 - имеет место обращение к сторонней системе, над которой у теста нет контроля.

Юнит тест проверяет одну
и только одну автономную
единицу работы.



► Свойства юнит теста:

- автоматизированный и повторяемый
- реализован просто

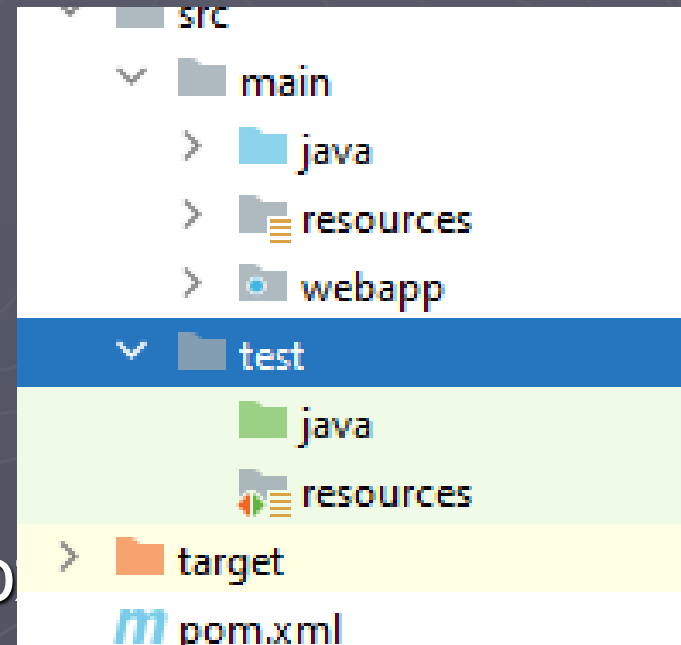
```
@Test
public void tst_mega_test()
{
    Animal cat = new Animal(1, 10);
    Animal mouse = new Animal(2, 1);
    assertEquals(1, cat.getType());
    assertEquals(10.0, cat.getWeight(), 0.01);
    assertEquals(0.0, cat.getHunger(), 0.01);
    int old_type = mouse.getType();
    assertEquals(1.0, cat.eat(mouse, mouse.getWe
    assertEquals(0.0, cat.getHunger(), 0.01);
    assertTrue(old_type == -mouse.getType());
    cat.setIntellect(10);
    assertEquals(5.5, cat.getIntellect(), 0.01);
}

@Test
public void tst_born_hunger()
{
    assertEquals(0.5, cat.getHunger(), 0.01);
}
```

- сохраняет актуальность во времени
- можно выполнить одним нажатием кнопки
- работает быстро (менее 100 мс)
- его результаты стабильны и повторяемы

► *Свойства юнит теста:*

- полностью контролирует тестируемую автономную единицу
- полностью изолирован от других тестов
- при неудачном завершении легко диагностировать ошибку
- отделены от кода



- могут писаться ДО кода прило
Test-Driven Development, разработка под
управлением тестированием и эмуляция
поведения ещё не созданного кода)

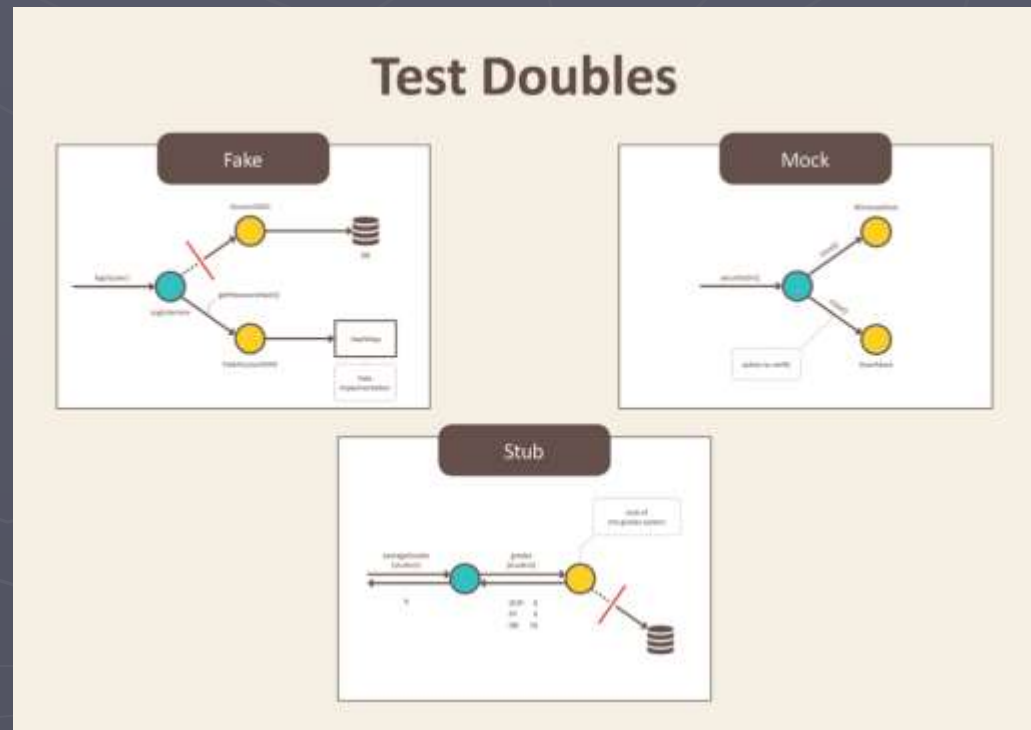
Объекты имитаторы (*fake*)

► заглушка («stub»).

- Не влияет на исход теста и предоставляют заранее заготовленные ответы на вызовы.

► подставка («mock»).

- Относительно него есть «утверждения» в тесте. Используются для взаимодействия или проверки поведения.



Метрики покрытия модульных тестов

- **Метрика покрытия** (coverage metric) – числовое выражение степени охвата тестами функций приложения.



- **Плотность покрытия** (coverage density metric) учитывает количество тестов, написанных для проверки той или иной функции.

► средств автоматизированного анализа метрик покрытия

► <http://java-source.net/open-source/code-coverage>

► CodeCover:

► <http://www.codecover.org>



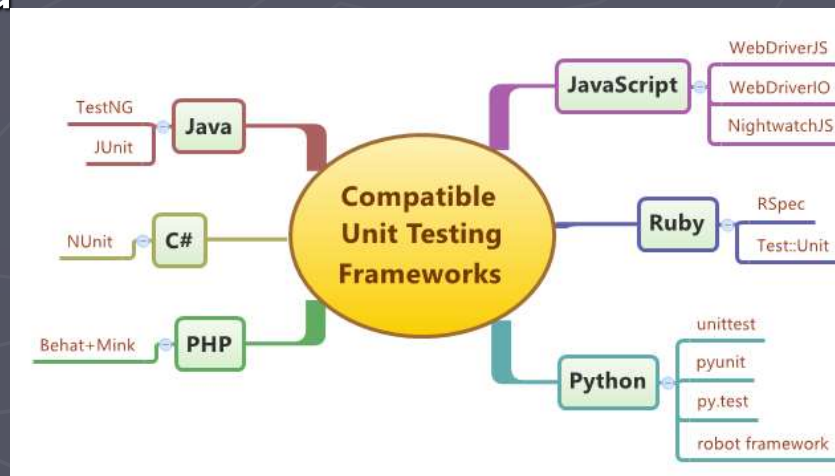
Фреймворки модульного тестирования в Java

- ▶ JUnit (3, 4, 5) <https://junit.org/junit5/>
 - набор расширений – таких как JMock, HtmlUnit и т.д.
 - Портирован на другие языки: PHP, C#, Python, Delphi, Perl, C++, JavaScript и т.д.
- ▶ TestNG <https://testng.org/doc/>
 - широкие возможностями по созданию тестов и проверок, управлению выполнением тестов
 - Не имеет аналогов вне Java

▶ Mockito

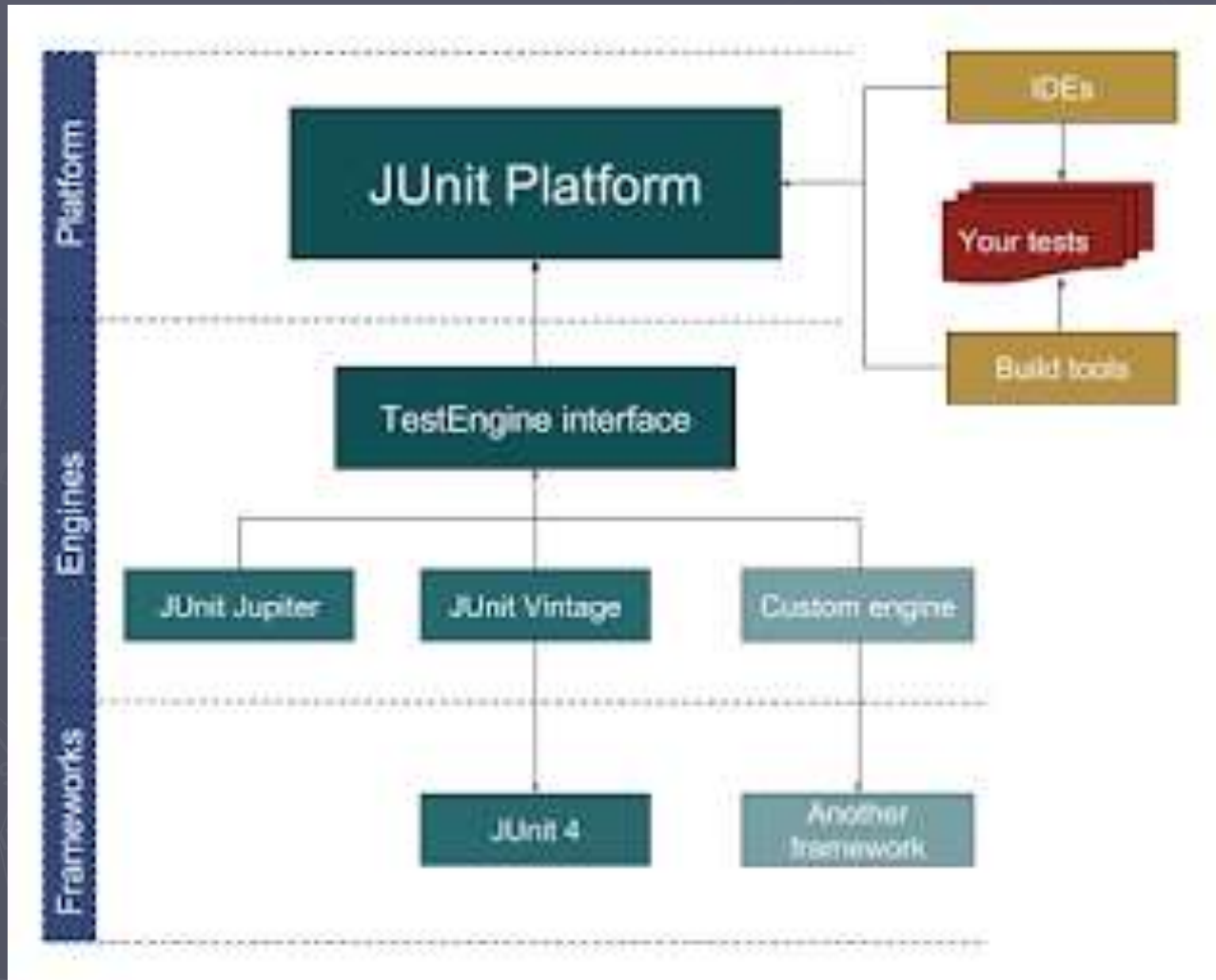
▶ EasyMock

▶ JMockit.



JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

<https://junit.org/junit5/docs/current/user-guide/>



Project Structure



Project Settings

Project

Modules

Libraries

Facets

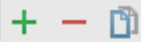
Artifacts

Platform Settings

SDKs

Global Libraries

Problems



junit:junit:5

Name: junit:junit:5

junit:junit:4.12



Classes

C:\Users\npats\.m2\repository\junit\junit\4.12\junit-4.12.jar

C:\Users\npats\.m2\repository\org\hamcrest\hamcrest-core\1.3\hamcrest-core-1.3.jar

Download Library From Maven Repository: too many results found

com.github.database-rider:rider-junit5:1.0.0

Found: 191
Showing: 191

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

☐ Download to:

C:\NATALIA\soft\PROJECTS\2018\UnitTest\jna66\lib

☐ Sources ☐ JavaDocs

OK

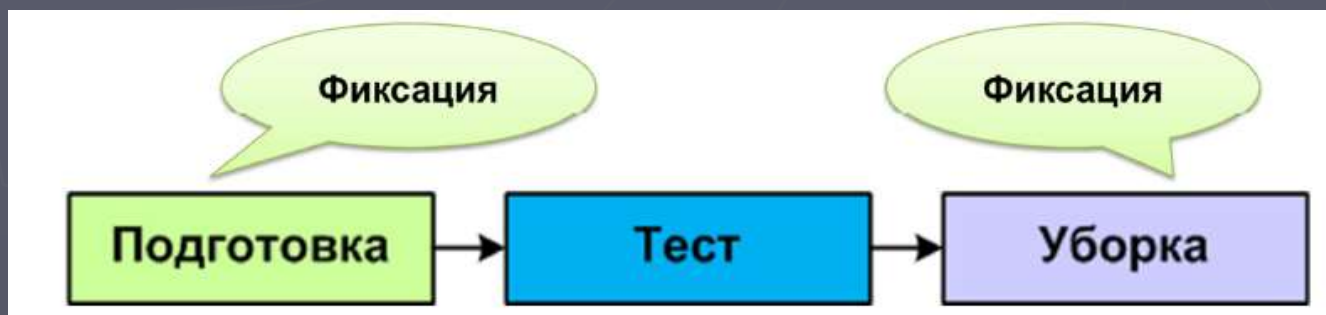
Cancel

Help

Основные аннотации в JUnit

<https://junit.org/junit5/docs/current/user-guide>

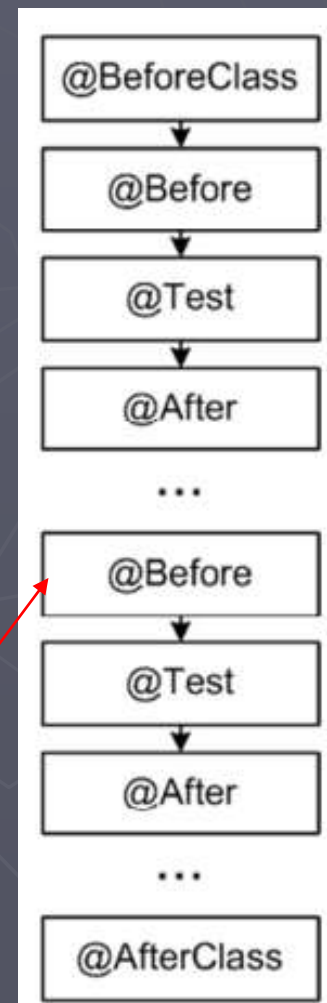
- Фиксации (fixtures) – методы, выполняющие подготовку к выполнению тестового метода и «уборку» после его выполнения.



@Before / @After – выполняются до и после КАЖДОГО тестового метода.

@BeforeClass / @AfterClass – выполняются до и после ВСЕГО НАБОРА тестовых методов.

НЕ гарантируется порядок выполнения @Before и @After.



► @Test

- Тесты (tests) – методы, непосредственно выполняющие проверку («тестовые методы»).

```
@Test  
void testExample1()  
{  
}
```

public — всё с Junit 5

► Тесты, проверяющие исключения

```
@Test(expected=ArithmeticException.class)  
void testExample2()  
{  
}
```

► Тесты с таймаутом

тестовый метод должен выполняться за некоторое время

```
@Test(timeout=2000)
public void testExample3()
{
}
```

► Игнорирование тестового метода **@Ignore()** **@Disabled**

```
@Ignore()
@Test
void testExample4()
{
}
```

тестовый метод выполнять не нужно



Параметризованные тесты

```
@RunWith(value = Parameterized.class)
class TestData{

    @Parameters

}
```

запуск будет параметризованным

Данные для параметризации

► Параметризованные тесты JUnit 5

```
@ParameterizedTest
@ValueSource(strings = { "Hello", "World" })
void testWithStringParameter(String argument) {
    assertNotNull(argument)
}
```

данные примитивных
типов: *int, long, double, String*

```
✓ testWithStringParameter(String)
  ✓ [1] Hello
  ✓ [2] World
  ○ skippedTest()
```

Parameterized Tests

```
@ParameterizedTest
```

```
    @ValueSource(strings = { "01.01.2021", "31.12.2021" })
```

```
    void testWithConverter(@JavaTimeConversionPattern("dd.MM.yyyy") LocalDate  
date) {
```

```
        assertEquals(2021, date.getYear());
```

```
    }
```

```
    // Пример с разбором CSV.
```

```
@ParameterizedTest
```

```
@CsvSource({ "foo, 1", "bar, 2", "'baz, qux', 3" })
```

```
    // или даже так: @CsvFileSource(resources = "/two-column.csv")
```

```
    void testWithCsvSource(String first, int second) {
```

```
        Assertions.assertNotNull(first);
```

```
        assertNotEquals(0, second);
```

```
    }
```

```
    // Пример с Enum.
```

```
@ParameterizedTest
```

```
@EnumSource(value = TimeUnit.class, names = { "DAYS", "HOURS" })
```

```
    void testWithEnumSourceInclude(TimeUnit timeUnit) {
```

```
        Assertions.assertTrue(EnumSet.of(TimeUnit.DAYS,
```

```
TimeUnit.HOURS).contains(timeUnit));
```

```
    }
```

► @Nested

```
@DisplayName("A stack")
public class StackTestJunit {
    Stack<Object> stack;
    @Test
    @DisplayName("is instantiated with new Stack()")
    void isInstantiatedWithNew() {
        new Stack<>();
    }
    @Nested
    @DisplayName("when new")
    class WhenNew {
        @BeforeEach
        void createNewStack() {
            stack = new Stack<>();
        }
        @Test
        @DisplayName("is empty")
        void isEmpty() {
            assertTrue(stack.isEmpty());
        }
    }
}
```

Наборов тестов

```
@RunWith(value=Suite.class)
@Suite.SuiteClasses(value={testExample1.class,
                           testExample4.class})

class _testRunner {
}
```

Указание набора

Выполнение
набора

- 1) Тесты выполняются в том порядке, в каком перечислены
- 2) В JUnit 4 НЕЛЬЗЯ включить в набор отдельные тестовые методы, как в JUnit 3.

Проверки в JUnit

- Проверки (assertions) – специальные методы, выполнение которых может закончиться успешно (проверка прошла) или не успешно (проверка не прошла).

1) Остановка теста

```
@Test  
  
void testExample1()  
{  
    fail();  
    fail("Stop!");  
}
```

проверка, которая всегда заканчивается неудачей.

Метод генерирует ошибку выполнения теста (завершает тест ошибкой) с выводом сообщения или без вывода.

используется как «заглушка» для тестов, логика которых ещё не реализована

2) Сравнение

`assertEquals()` сравнивает два значения

```
assertEquals(4.0, 2.0*2.0, 0.0001);  
assertEquals(4, 2*2);
```

Проверка считается пройденной,
если значения равны

3) ИСТИННОСТЬ И ЛОЖНОСТЬ

`assertTrue()` проверяет, является ли переданный аргумент логически равным `true`.

```
boolean some_bool_var = 3<5;  
assertTrue(some_bool_var);  
assertTrue(2*2 == 4);  
assertTrue( 2*2 == 4, "Math error!");
```

```
public void junit4Test() {  
    Assert.assertTrue(true);  
    Assertions.assertTrue(true);
```

```
// JUnit 4 Assertion  
// JUnit 5 Assertion
```

`assertFalse()` проверяет, является ли переданный аргумент логически равным `false`.

```
boolean some_bool_var = 3<5;  
    assertFalse(some_bool_var);  
    assertFalse(2 * 3 == 4);  
    assertFalse(2 * 3 == 4, "Math OK!");  
}
```

4) Отсутствие и наличие

`assertNull()` проверяет, является ли переданный аргумент логически равным `null`

```
testobj = null;  
    assertNull(testobj);  
    assertNull( testobj, "Self-created obj?");
```

`assertNotNull()` проверяет, является ли переданный аргумент логически НЕ равным `null`.

```
assertNotNull(testobj);  
assertNotNull(testobj, "Creation failed!?!");
```

5) Сравнение объектов

`assertSame()` проверяет, ссылаются ли переданные аргументы на один и тот же объект

```
assertSame(obj1, obj2);  
assertSame(obj1, obj2, "Different!");
```

`assertNotSame()` проверяет, ссылаются ли переданные аргументы на разные объекты.

```
assertNotSame(obj1, obj2);  
assertNotSame(obj1, obj2, "The same!");
```

`assertThat(T actual, Matcher<T> matcher)`

```
MatcherAssert.assertThat(dummyFruits,  
    Matchers.anyOf(Collections.singletonList(Matchers.contains(Arrays.asList(  
        Matchers.hasProperty("name", Matchers.is("Baby")),  
        Matchers.hasProperty("name", Matchers.is("Granny")),  
        Matchers.hasProperty("name", Matchers.is("Grapefruit"))  
)))));
```

6) Сравнение массивов

`assertArrayEquals()` проверяет, являются ли массивы идентичными.

```
assertArrayEquals(int_arr1, int_arr2);  
assertArrayEquals(sa1, sa2, "NO!");
```

JUnit 5

► 7) Сравнение группы, строк

```
@Test
@DisplayName("Check all assert")
void GroupTest() {
    Assertions.assertAll("all assertion",
        () -> assertThat("JUNIT", startsWith("J")),
        () -> assertThat("JUNIT", endsWith("T")));
    Assertions.assertLinesMatch(List.of("1", "2"), List.of("1", "2"));
}
```

► 8) Тестирование исключений

► assertThrows ()

```
@Test
void assertThrowsException() {
    String str = null;
    assertThrows(IllegalArgumentException.class, () -> {
        Integer.valueOf(str);
    });
}
```

```
// Alternative to assertThrows of JUnit 5 with fluent api
assertThatThrownBy(() -> { throw new Exception("I'm an exception!");
}).hasMessage("I'm an exception!");
```

```
// Or get the Throwable as an instance
Throwable thrown = catchThrowable(() -> { throw new Exception("I'm an
exception!"); });
assertThat(thrown).hasMessageContaining("I'm an exception!");    //
ThrowableAssert
```


JUnit 5

```
// вместо @BeforeClass
@BeforeAll
static void initAll() {
}

// вместо @Before
@BeforeEach
void init() {
}

@Test
void succeedingTest() {
}

// Вместо @Ignore
@Test
@Disabled("for demonstration purposes")
void skippedTest() {
    // not executed
}

// Новая аннотация для улучшения читаемости при выводе результатов тестов.
@DisplayName(" ")
void testWithDisplayNameContainingSpecialCharacters() {}

// вместо @After
@AfterEach
void tearDown() {
}

// вместо @AfterClass
@AfterAll
static void tearDownAll() {
}
```

```
}
```

JUnit 5

► Автоматический повторный запуск теста

```
@RepeatedTest (5)
@Test
@DisplayName ("is empty")
void isEmpty() {
    assertTrue (stack.isEmpty());
}
```

<https://junit.org/junit5/docs/current/user-guide>

org.junit.Assume

- ▶ **assumeNoException(Throwable t)** — тестируемый метод завершится, не вызвав исключения;
- ▶ **assumeNotNull(Object...objects)** — передаваемый аргумент(ы) не является ссылкой на null;
- ▶ **assumeThat(T actual, Matcher<T> matcher)** — условие выполнится; **assumeTrue(boolean b)** — значение передаваемого аргумента истинно.

Тестовые сценарии в JUnit

- ▶ Тестовые сценарии (тест-сьюты, test-suites) – это наборы тестов, которые следуют некоторой логике:
 - проверяем модуль;
 - выполняем важные тесты;
 - проверяем функционал (работа с БД и т.п.)

Сценарии и реализация:

- Тестовые методы внутри одного класса.
- Несколько классов, объединённых в тест-сьют

составные элементы сценария должны быть независимы друг от друга



JUnit

- ControllerTest
- ControllerTest
- ControllerTest.testWithConverter
- ControllerTest.testWithCsvSource
- ControllerTest.testWithArgumentsSource

Templates

Name: ControllerTest.testWithCsvSource

☐ Share through VCS ☐ Allow parallel run

Configuration Code Coverage Logs

Test kind: Method Fork mode: none Repeat: Once 1

Class: patsei.ControllerTest

Method: testWithCsvSource(java.lang.String,int)

VM options: -ea

Program arguments:

Working directory: \$MODULE_DIRS

Environment variables:

☐ Redirect input from:

Use classpath of module: unit

JRE: Default (11.0.2 - SDK of 'unit' module)

Shorten command line: user-local default: none - java [options] classname [args]

Before launch: Build, Activate tool window



Build

☐ Show this page ☒ Activate tool window

OK

Cancel

Apply

Mockito

- ▶ <https://static.javadoc.io/org.mockito/mockito-core/2.27.0/org/mockito/Mockito.html>
- ▶ <https://site.mockito.org/>

```
@Test
void mockIteratorTest() { //подготавливаем
    Iterator i = Mockito.mock(Iterator.class);
    Mockito.when(i.next()).thenReturn("Hello").thenReturn("World");
    //выполняем
    String result = i.next() + " " + i.next();
    //сравниваем
    Assertions.assertEquals("Hello World", result);
}
```

```
org.opentest4j.AssertionFailedError:
Expected :Hell World
Actual   :Hello World
<Click to see difference>
```

```
@RunWith(MockitoJUnitRunner.class)
```

```
public class SongServiceTest {
```

```
    @InjectMocks
```

```
    private SongService songService;
```

```
    @Mock
```

```
    private SongRepository songRepository;
```

```
    @Test
```


TestNG

- ▶ Поддержка DDT с использованием `@DataProvider`
- ▶ Поддержка нескольких копий тестового класса с использованием `@Factory`
- ▶ Возможность распределённого тестирования.
- ▶ Специальная модель управления наборами тестов (без `TestSuite`).

<http://testng.org/doc/documentation-main.html>

Основные аннотации в TestNG

► Фиксации

@BeforeSuite / @AfterSuite – запускаются до и после набора тестов.

@BeforeTest / @AfterTest – запускаются до и после каждого теста.

@BeforeGroups / @AfterGroups – запускаются до и после группы тестов.

@BeforeClass / @AfterClass – запуск перед первым и после последнего тестового метода.

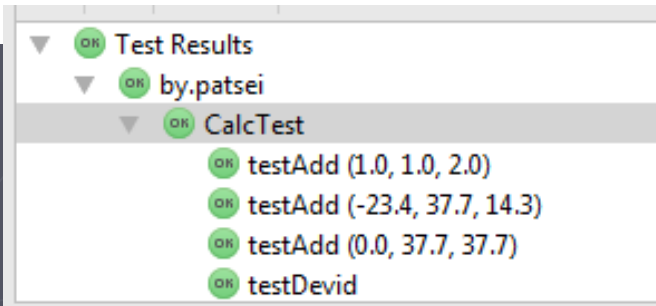
@BeforeMethod / @AfterMethod – запуск перед и после каждого тестового метода.

► Использование внешних данных

```
@DataProvider(name = "testdata")
public Object[][] createData1() {
    return new Object[][] {
        { new Double(1.0), new Double(1.0), new Double(2.0) },
        { new Double(-23.4), new Double(37.7), new Double(14.3) },
        { new Double(0), new Double(37.7), new Double(37.7) },
    };
}
```

```
@Test(dataProvider = "testdata")
public void testAdd(Double a, Double b, Double c) throws Exception {

    Assert.assertEquals(c, testobj.add(a, b), 0.001 );
    System.out.println(a + " + " + b + " = " + c);
}
```



```
1.0 + 1.0 = 2.0
тестирование начато
-23.4 + 37.7 = 14.3
тестирование начато
0.0 + 37.7 = 37.7
тестирование начато
тестирование начато
```

► Упрощённая параметризация

@Parameters

```
public class tstParametersExample {  
  
    @Parameters({ "paramInt", "paramStr" })  
    @Test  
    public void tstSomeTest(int in, String st)  
    {  
        System.out.println(in + " = " + st);  
    }  
  
}
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<suite name="Suite" parallel="false">  
    <parameter name="paramInt" value="1" />  
    <parameter name="paramStr" value="One" />  
    <test name="SampleTestOne">  
        <classes>  
            <class name="environment.tstParametersExample" />  
        </classes>  
    </test>  
</suite>
```

Источники данных в TestNG

Annotation	ITestContext	XmlTest	Method	Object[]	ITestResult
BeforeSuite	Yes	No	No	No	No
BeforeTest	Yes	Yes	No	No	No
BeforeGroups	Yes	Yes	No	No	No
BeforeClass	Yes	Yes	No	No	No
BeforeMethod	Yes	Yes	Yes	Yes	Yes
Test	Yes	No	No	No	No
DataProvider	Yes	No	Yes	No	No
AfterMethod	Yes	Yes	Yes	Yes	Yes
AfterClass	Yes	Yes	No	No	No
AfterGroups	Yes	Yes	No	No	No
AfterTest	Yes	Yes	No	No	No
AfterSuite	Yes	No	No	No	No

Основные проверки в TestNG

► Остановка теста

```
@Test
public void testExample4()
{
    Assert.fail();
    Assert.fail("Test failed!");
}
```

OR testEnvironment

! testExample4

OR testMinus

тестирование начато

junit.framework.AssertionFailedError

at junit.framework.Assert.fail([Assert.java:47](#))

at junit.framework.Assert.fail([Assert.java:53](#))

at by.patsei.CalcTest.testExample4([CalcTest.java:37](#))

junit.framework.AssertionFailedError: Test failed!

at junit.framework.Assert.fail([Assert.java:47](#))

at by.patsei.CalcTest.testExample4([CalcTest.java:38](#))

тестирование начато

► Сравнение

```
Assert.assertEquals(2.0*2.0, 4.0);  
Assert.assertEquals("Err!", 2.0 * 2.0, 4.0 );  
Assert.assertEquals(2*2, 4);
```

► ИСТИННОСТЬ И ЛОЖНОСТЬ

```
Assert.assertTrue(2 * 2 == 4);  
Assert.assertTrue("Math error!", 2 * 2 == 4 );
```

```
Assert.assertFalse(2 * 3 == 4);  
Assert.assertFalse("Math OK!", 2*3 == 4 );
```

► Отсутствие и наличие

```
Assert.assertNull(testobj) ;  
Assert.assertNull("????", testobj) ;
```

```
Assert.assertNotNull(testobj) ;  
Assert.assertNotNull("????", testobj) ;
```

► Сравнение объектов

```
Assert.assertSame(obj1, obj2) ;  
Assert.assertSame("Different!", obj1, obj2) ;
```

```
Assert.assertNotSame(obj1, obj2) ;  
Assert.assertNotSame("Different!", obj1, obj2) ;
```


► Сравнение массивов

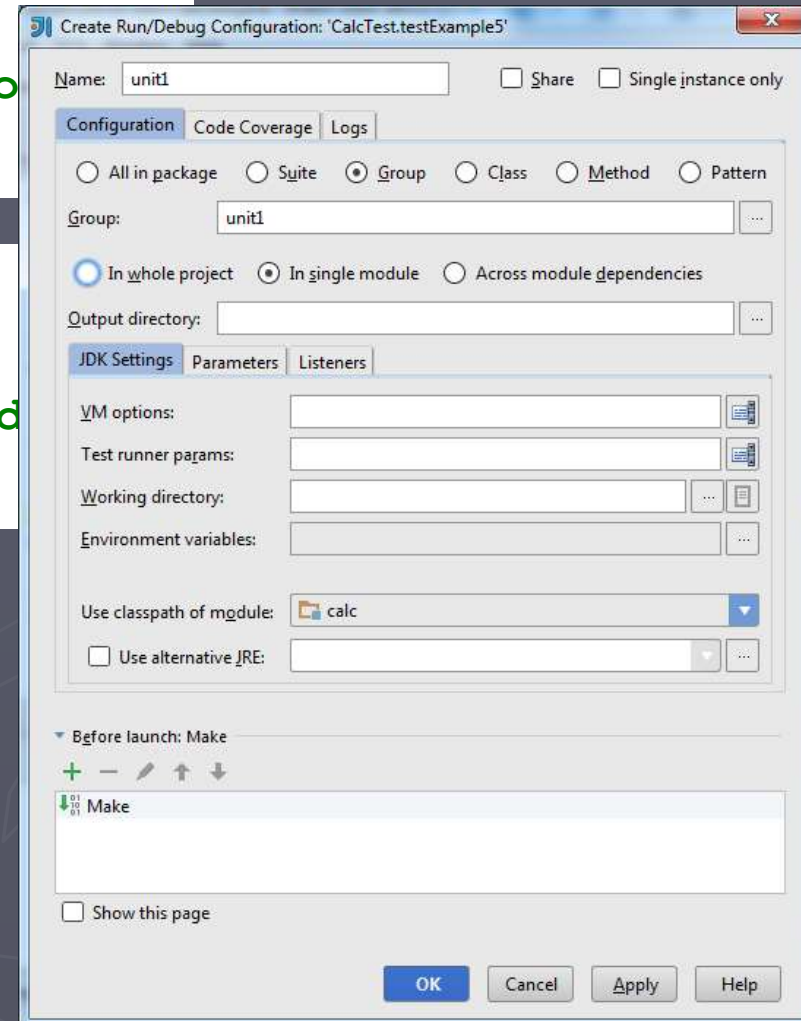
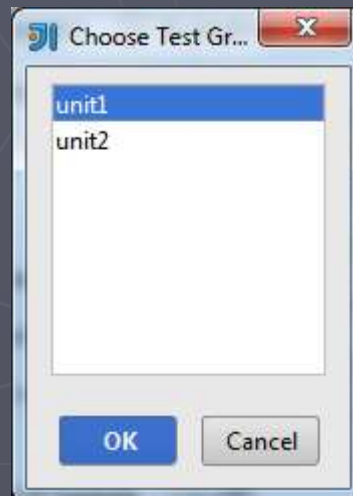
```
assertArrayEquals(int_arr1, int_arr2);
```

проверяет, содержат ли массивы одинаковый набор элементов

Групповое тестирование

```
@Test(groups={"unit1"})  
public void testExample5() {  
    System.out.println("testingMethod")  
}
```

```
@Test(groups={"unit1", "unit2"})  
public void testExample5() {  
    System.out.println("testingMethod")  
}
```





Зависимые тесты



```
@Test
public void initEnvironmentTest() {
    System.out.println("This is initEnvironmentTest");
}
```

```
@Test(dependsOnMethods={"initEnvironmentTest"})
public void testmethod() {
    System.out.println("This is testmethod");
}
```

```
@Test(dependsOnMethods={"unit1"})
public void testmethod() {
    System.out.println("This is testmethod");
}
```



▼  CalcTest

-  initEnvironmentTest
-  testmethod

МНОГОПОТОЧНОСТЬ

```
@Test( threadPoolSize = 30,  
      invocationCount = 100,  
      invocationTimeOut = 1000)
```

Максимальное
количество потоков

количество тестов

```
public void testWithFakeEncrypter() throws IOException {  
  
}
```

Общее время запуска всех
тестов, после которого
тест считается
провалившимся

Best Practice

Что тестировать?

Методы, классы, взаимодействие классов, геттеры и сеттеры, конструкторы, исключения, внешние зависимости.

1) Метод

```
Calc testobj = new Calc();
```

```
@Test
```

```
public void testCalcAdd(Double a, Double b, Double c)  
    throws Exception {
```

Ожидаемое значение

```
Assert.assertEquals(c, testobj.add(a,b), 0.001 );  
System.out.println(a + " + " + b + " = " + c);
```

```
}
```

Вызов метода

2) Класс

- ▶ Определить ситуации, когда объект изменяет своё состояние
- ▶ Проверить соответствующие сценарии
 - инициализация
 - выполнение методов
 - Т.П.

```
public class CalcTest {  
  
    Calc testobj = new Calc();  
  
    @Test  
    public void testInit() throws Exception {  
  
        Assert.assertEquals(0, testobj.getState(), 0.001);  
  
    }  
}
```

3) Взаимодействие классов

Определить ситуации, когда объекты взаимодействуют

А) Провести взаимодействие.

В) Проверить изменения в объектах.

4) Тестирование конструктора

А) Создать объект

В) Проверить те изменения, которые производит конструктор

```
@Test
public void testConstructor() {

    testobj = new Calc(2, 4);
    Assert.assertEquals(2, testobj.getOperandA(0), 0.001);
    Assert.assertEquals(4, testobj.getOperandB(0), 0.001);
}
```

5) Геттеры и сеттеры

A) Вызвать сеттер.

B) Вызвать геттер.

C) Проверить, что переданное в сеттер верно возвращено геттером.

тестировать простые get и set (которые просто устанавливают или возвращают значение) не надо →, тесты бесполезны в силу малой вероятности обнаружения ошибки

```
@Test
public void testOperandA() throws Exception {

    testobj.setOperandA(0);
    Assert.assertEquals(0, testobj.getOperandA(), 0.001);
}
```


6) Тестирование исключений

А) Вызвать исключение

В) Проверить факт его возникновения

```
@Test(expectedExceptions = Exception.class)

public void testZeroDiv() throws Exception {

    assertEquals(1.0, testobj.devid(1.0,0.0), 0.001);
}
```

Принципы написания тестов

- ▶ **AAA (Arrange, Act, Assert)**. Четкое разделение в тесте этапа подготовки (Arrange), воздействия на объект (Act) и утверждения о результате (Assert)
- ▶ **KISS (Keep It Simple)**. Тесты должны быть максимально просты и линейны (не содержать циклов и ветвлений)
- ▶ **Small**. Средний размер теста — 1-16 строк кода
- ▶ **DRY (Do not repeat yourself)**. В тестах не должно быть дублей кода
- ▶ **DAMP (Descriptive And Meaningful Phrases)** — имена тестов и переменных внутри теста должны быть содержательными

Принципы именования тестов

```
@Test
public void getProduct_oneProductInDb_productReturned() {
    Product product = product("productName").build();
    productRepository.save(product);

    Product result = productController.getProduct(product.getId());

    assertEquals("productName", result.getName());
}

@Test
public void getProduct_twoProductsInDb_correctProductReturned() {
    Product product1 = product("product1").build();
    Product product2 = product("product2").build();
    productRepository.save(product1);
    productRepository.save(product2);

    Product result = productController.getProduct(product1.getId());

    assertEquals("product1", result.getName());
}
```

- ▶ Project — [ProjectUnderTest].UnitTests
- ▶ Class — [ClassName]Tests
- ▶ Unit of work —
[UnitOfWorkName]_[ScenarioUnderTest]_[ExpectedBehavior]
(ИмяТестируемогоМетода_Сценарий_ОжидаемоеПоведение)

Given, When, Then

- ▶ Given (Input): подготовка к тестам, например создание данных или настройка
- ▶ When (Action): вызовите метод или действие, которое вы хотите проверить
- ▶ Then (Output): Выполните утверждения, чтобы проверить правильность вывода или поведения действия.

@Test

```
public void findProduct() {  
    insertIntoDatabase(new Product(100, "Smartphone"));  
  
    Product product = dao.findProduct(100);  
  
    assertThat(product.getName()).isEqualTo("Smartphone");  
}
```

Используйте префиксы actual*" and "expected*"

```
ProductDTO actualProduct = requestProduct(1);
```

```
ProductDTO expectedProduct =  
    new ProductDTO("1",  
        List.of(State.ACTIVE, State.REJECTED))
```

```
assertThat(actualProduct).isEqualTo(expectedProduct);
```

Пишите небольшие и специфичные тесты

```
@Test
public void categoryQueryParameter() throws Exception {
    List<ProductEntity> products = List.of(
        new
ProductEntity().setId("1").setName("Envelope").setCategory("Office").setDescription("An Envelope").setStockAmount(1),
        new
ProductEntity().setId("2").setName("Pen").setCategory("Office").setDescription("A Pen").setStockAmount(1),
        new
ProductEntity().setId("3").setName("Notebook").setCategory("Hardware").setDescription("A Notebook").setStockAmount(2)
    );
    for (ProductEntity product : products) {
        template.execute(createSqlInsertStatement(product));
    }

    String responseJson = client.perform(get("/products?category=Office"))
        .andExpect(status().is(200))
        .andReturn().getResponse().getContentAsString();

    assertThat(toDTOs(responseJson))
        .extracting(ProductDTO::getId)
        .containsOnly("1", "2");
}
```

@Test

```
public void categoryQueryParameter2() throws Exception {  
    insertIntoDatabase(  
        createProductWithCategory("1", "Office"),  
        createProductWithCategory("2", "Office"),  
        createProductWithCategory("3", "Hardware")  
    );  
  
    String responseJson =  
requestProductsByCategory("Office");  
  
    assertThat(toDTOs(responseJson))  
        .extracting(ProductDTO::getId)  
        .containsOnly("1", "2");  
}
```


Не злоупотребляйте переменными

@Test

```
public void variables() throws Exception {  
    String relevantCategory = "Office";  
    String id1 = "4243";  
    String id2 = "1123";  
    String id3 = "9213";  
    String irrelevantCategory = "Hardware";  
    insertIntoDatabase(  
        createProductWithCategory(id1, relevantCategory),  
        createProductWithCategory(id2, relevantCategory),  
        createProductWithCategory(id3, irrelevantCategory)  
    );  
  
    String responseJson =  
requestProductsByCategory(relevantCategory);  
  
    assertThat(toDTOs(responseJson))  
        .extracting(ProductDTO::getId)  
        .containsOnly(id1, id2);  
}
```

```
@Test
public void variables() throws Exception {
    insertIntoDatabase(
        createProductWithCategory("4243", "Office"),
        createProductWithCategory("1123", "Office"),
        createProductWithCategory("9213", "Hardware")
    );

    String responseJson = requestProductsByCategory("Office");

    assertThat(toDTOs(responseJson))
        .extracting(ProductDTO::getId)
        .containsOnly("4243", "1123");
}
```

Не расширяйте существующие тесты, чтобы
«просто протестировать еще одну мелочь»

// Do

```
public class ProductControllerTest {
```

```
    @Test
```

```
    public void multipleProductsAreReturned() {}
```

```
    @Test
```

```
    public void allProductValuesAreReturned() {}
```

```
    @Test
```

```
    public void filterByCategory() {}
```

```
    @Test
```

```
    public void filterByDateCreated() {}
```

```
}
```

Assert только то, что проверяем

```
String responseJson = requestProducts();
```

```
ProductDTO expectedDTO1 = new ProductDTO("1", "evelope", new  
Category("office"), List.of(States.ACTIVE, States.REJECTED));  
ProductDTO expectedDTO2 = new ProductDTO("2", "evelope", new  
Category("smartphone"), List.of(States.ACTIVE));  
assertThat(toDTOs(responseJson))  
    .containsOnly(expectedDTO1, expectedDTO2);
```

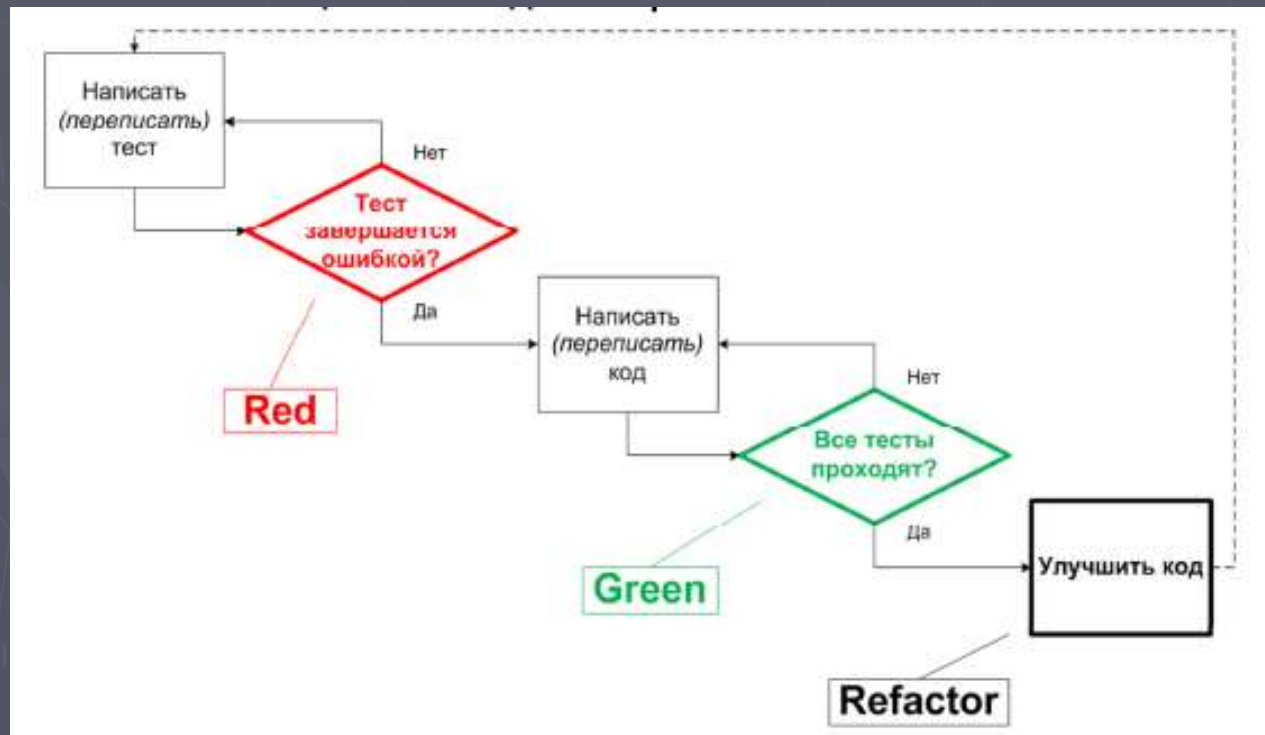


```
String responseJson =  
requestProductsByCategory("Office");  
  
assertThat(toDTOs(responseJson))  
    .extracting(ProductDTO::getId)  
    .containsOnly("1", "2");
```

```
assertThat(actualProduct.getPrice()).isEqualTo(100);
```

Test-Driven Development (TDD)

- Разработка под управлением тестированием – техника разработки ПО, опирающаяся на очень короткие повторяющиеся циклы, в которых написание тестов предшествует написанию кода.



Red-Green-Refactor

TDD: 1 – добавить тест

Перед реализацией какого бы то ни было требования, функции и т.п. сначала пишется тест или несколько тестов, которые проверят корректность впоследствии созданного кода.

TDD: 2 – убедиться, что тест заваливается

Только что написанный тест (тесты) выполняется и ДОЛЖЕН завершиться неудачей (т.к. ещё нет кода, который возвратил бы корректный результат). Если на данной стадии тест проходит без ошибок, значит, сам тест содержит ошибку.

TDD: 3 – написать код

Теперь, когда у нас есть тесты, мы можем писать код и периодически повторять запуск тестов. Не стоит концентрироваться на оптимальности кода. Код просто должен проходить тесты.

TDD: 4 – убедиться, что тесты проходят

Когда все написанные в начале этого цикла разработки, стали успешно проходить, у нас есть работающий код.

TDD: 5 – улучшить код

Теперь в код можно вносить правки с целью повышения производительности, сопровождаемости и иных показателей качества. Если тесты по-прежнему проходят, значит «мы ничего не сломали». Если же какие-то тесты стали «заваливаться», – это повод исправить тот участок кода, который перестал работать.

Тестирование под управлением данными

- ▶ Тестирование под управлением данными (Data-Driven Testing, DDT) – подход, при котором исключается «хардкодинг» данных внутри теста. Данные берутся ИЗВНЕ.

```
System.getProperty("today");
```

```
System.getenv("Path");
```



Файлы свойств

```
String path =  
  
Runner.class.getProtectionDomain().getCodeSource().getLocation().getPath();  
FileInputStream fis = new FileInputStream(path + "../_data/properties1.abc");  
  
Properties p = new Properties();  
p.load(fis);  
System.out.println("Double = " +  
    Double.parseDouble(p.getProperty("My.Double.Number")));
```

Файловые репозитории (File Repositories) – хранилища файлов (как правило, под управлением какой-то CVS-системы).



Тестирование под управлением ключевыми словами

- ▶ Тестирование под управлением ключевыми словами (Keyword-Driven Testing, KDT) – подход, при котором за пределы теста выносятся не только данные, но и логика теста.
- ▶ В итоге KTD-тесты начинают напоминать некий предельно простой высокоуровневый язык.
(Selenium IDE и его язык Selenese)

Рекомендации

- ▶ На новом приложении
- ▶ Регулярный запуск
- ▶ Качество не количество
- ▶ Время запуска
- ▶ Понятные и читаемы
- ▶ Не увлекаться TDD
- ▶ Рефакторинг