нетология

Тестирование кода и Unit-тесты

Георгий
Власов

Георгий Власов

Java/Kotlin Dev, Developer Advocate в компании "Haulmont", "

План занятия

1. Ошибки в тестировании
2. Зачем нам тестировать
3. Основные понятия Unit-тестирования
4. Уровни тестирования
5. JUnit
6. Что ещё нужно знать?
7. Итоги
8. Домашнее задание

Ошибки в тестировании

USS John F. Kennedy (CVN-79)

Стоимость
11 300 000 000$

Водоизмещение
100 000 тонн

Команда
4 660

Самолётов
80

Ущерб от ошибок в ПО

1 730 000 000 000 или
1,73 триллиона долларов в год

x 153

Ущерб от ошибок в ПО

Ракета
Ariane 5

Ущерб
8,5 миллиардов долларов

[Самые дорогие ошибки в IT >>]

*Пострадавших не было

Стоимость ошибок
(кондитерская)

4$ Себестоимость коржа

30$ Себестоимость торта
(2 коржа, крем + работа)

5$ Упаковка

15$ Доставка

Стоимость ошибок

(кондитерская)

Выпечка	Упаковка	Доставка	Употребление
			
4$	30$	35$	50$
0%	650%	16%	42%
0%	650%	785%	1150%

Зачем нам тестировать?

Всё ведь просто

Как мы тестировали на предыдущих лекциях?

Всё ведь просто

Как мы тестировали на предыдущих лекциях?

Можно ли так тестировать большие программы?

Что дают тесты

- Где ошибка
- Какая ошибка
- На каких данных проявляется
- Уверенность, что всё работает
- Часто даже улучшают архитектуру
- Примеры использования кода (всегда актуальная документация)

ЭКОНОМИМ ВРЕМЯ
ПОВЫШАЕМ КАЧЕСТВО

Что дают тесты

- Где ошибка
- Какая ошибка
- На каких данных проявляется
- Уверенность, что всё работает
- Часто даже улучшают архитектуру
- Примеры использования кода (всегда актуальная документация)

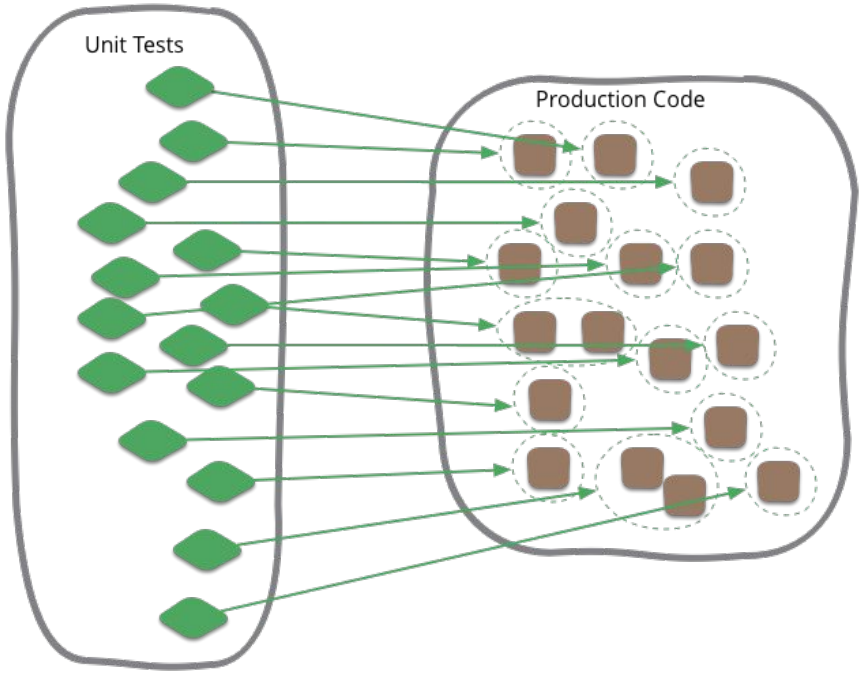
ЭКОНОМИМ ВРЕМЯ
ПОВЫШАЕМ КАЧЕСТВО

Хотите это всё? Вперёд!

Основные понятия Unit-тестирования

Unit-модуль

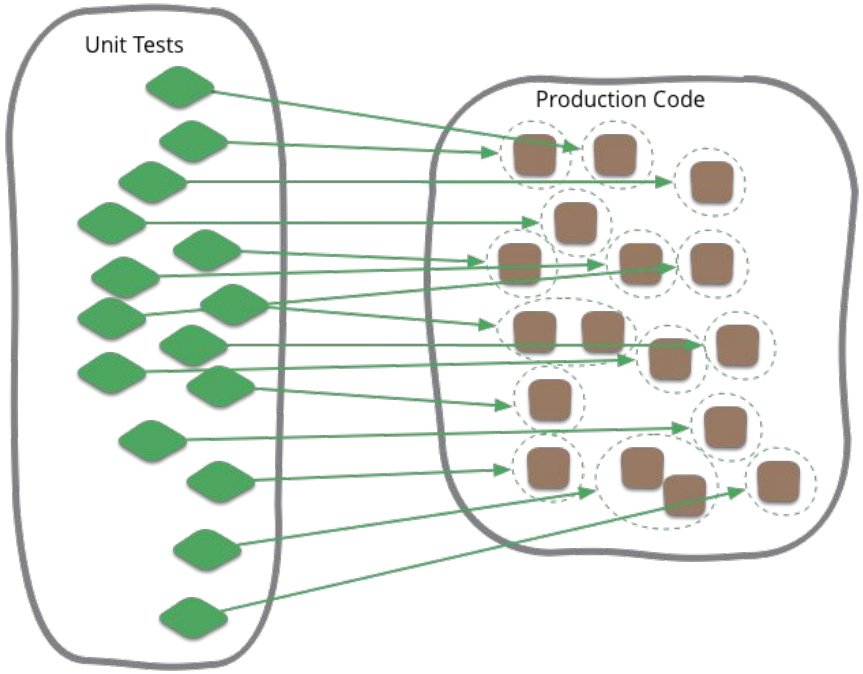
Unit-модуль - изолированная часть кода



Unit-модуль

Unit-модуль - изолированная часть кода

- класс
- метод
- набор методов
- набор классов



Unit-тестирование*

Unit-тестирование - соответствие ожидаемого поведения модуля реальному

✅ Ожидание

❌ Реальность

*Unit-тестирование также называют модульным тестированием

Уровни тестирования

Основные уровни тестирования

Модульное

ЦЕЛЬ - ПРОВЕРИТЬ, ЧТО КАЖДЫЙ МОДУЛЬ (UNIT) РАБОТАЕТ В ОТДЕЛЬНОСТИ

- Если хоть один модуль не работает - **вся система не работает**
- Если каждый модуль работает - **вся система, скорее всего, работает**

Основные уровни тестирования

Модульное

Интеграционное

ЦЕЛЬ - ПРОВЕРИТЬ, ЧТО МОДУЛИ РАБОТАЮТ ВМЕСТЕ

- Иногда проверяют взаимодействие и с внешними системами: БД и др.

Основные уровни тестирования

ЦЕЛЬ - ПРОВЕРИТЬ, ЧТО РАБОТАЕТ СИСТЕМА В ЦЕЛОМ

- Полное тестирование от **А** до **Я**
- Обычно проводится **тестерами** и **автотестерами**

Основные уровни тестирования

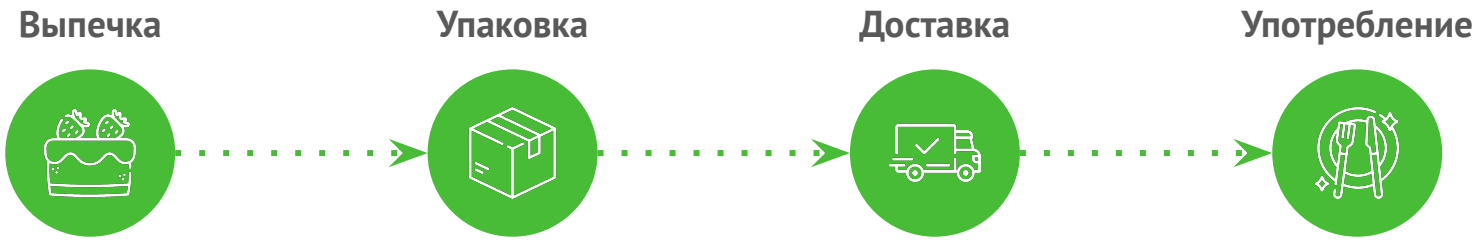
ЦЕЛЬ - ПОКАЗАТЬ РЕЗУЛЬТАТЫ ЗАКАЗЧИКУ

- Приходит заказчик и проверяет, что **всё действительно так, как он хотел**
- Либо мы сами приходим и проводим **демо-показ**

Что, где и как тестировать?

Жизненный цикл продукта

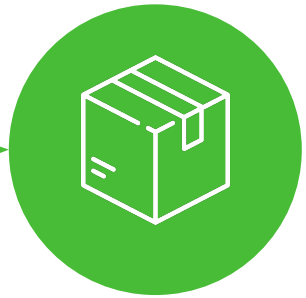
(кондитерский)



Что тестировать

Выпечка

Упаковка

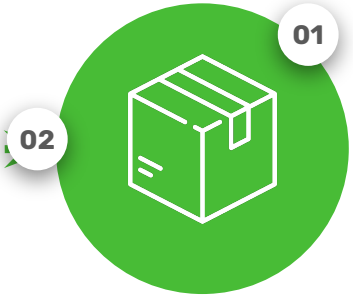


Что нам нужно протестировать?

Что тестировать

Выпечка

Упаковка



03

01 Элементы

02 Соединения

03 Канал

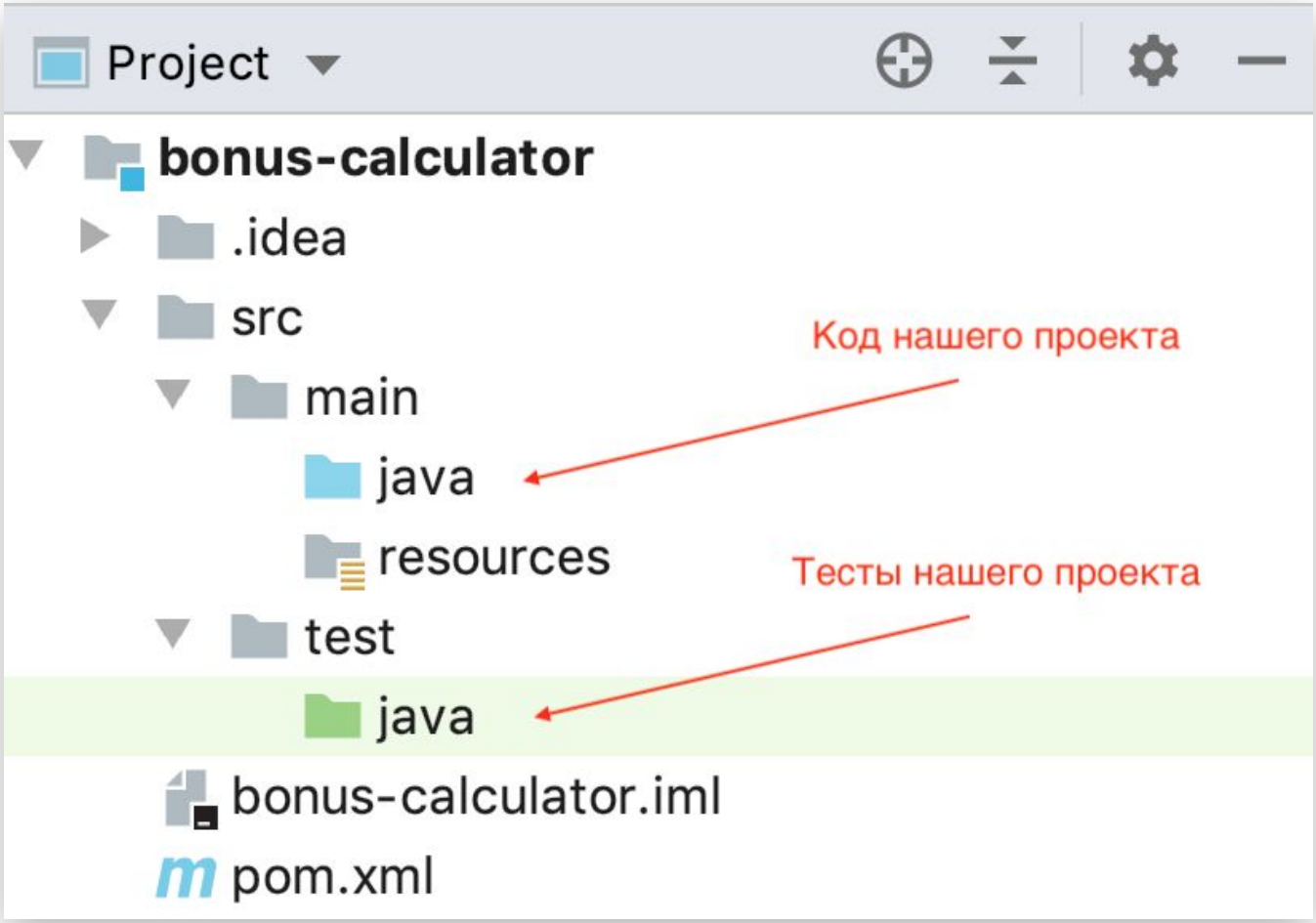
Где и как писать тесты

1. В том же классе, где и основной код
2. В классе рядом с основным кодом
3. В отдельной папке
4. В отдельном проекте

Где и как писать тесты

1. В том же классе, где и основной код
2. В классе рядом с основным кодом
3. **В отдельной папке**
4. В отдельном проекте

Где будут находиться тесты



JUnit

JUnit

JUnit - фреймворк Unit-тестирования для Java

- @Test
- @After/@AfterEach
- @Before/@BeforeEach
- Exception testing

Подключаем зависимость

Чтобы подключить зависимость, нужно совершить следующие операции:

```xml
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.1.0</version>
  <scope>test</scope>
</dependency>
```

```
dependencies {
  // .. другие зависимости
  testImplementation('org.junit.jupiter:junit-jupiter:5.6.2')
}

test {
  useJUnitPlatform()
}
```

Что тестировать

Метод класса String, который добавляет одну строку в конец другой. При передаче пустой строки - возвращает исходную

“AB”.concat(“BC”) = “ABBC”

```
public String concat(String str) {
    int otherLen = str.length();
    if (otherLen == 0) {
        return this;
    }
    int len = value.length;
    char buf[] = Arrays.copyOf(value, len + otherLen);
    str.getChars(buf, len);
    return new String(buf, true);
}
```

Как тестировать

Результат при **непустом** аргументе:

```
@Test
public void testConcat_validArgument_success() {
        // подготавливаем данные:
    final String original = "Test string ";
    final String argument = "valid";
    final String expected = "Test string valid";

        // вызываем целевой метод:
    final String result = original.concat(argument);

        // производим проверку (сравниваем ожидание и результат):
    Assertions.assertEquals(expected, result);
}
```

Как тестировать

Результат при **пустом** аргументе:

```
@Test
public void testConcat_emptyString_originalString() {
        // given:
    final String original = "Test string ";
    final String argument = "";

        // when:
    final String result = original.concat(argument);

        // then:
    Assertions.assertEquals(original, result);
}
```

Как тестировать

Результат при **null** аргументе*:

```
@Test
public void testConcat_nullArgument_throwsException() {
        // given:
    final String original = "Test string ";

        // expect:
    assertThrows(NullPointerException.class, () -> {
        original.concat(null);
         });
}
```

*Этот способ только для JUnit 5. В старых версиях по-другому

Автотест

Наш тест — это просто метод:

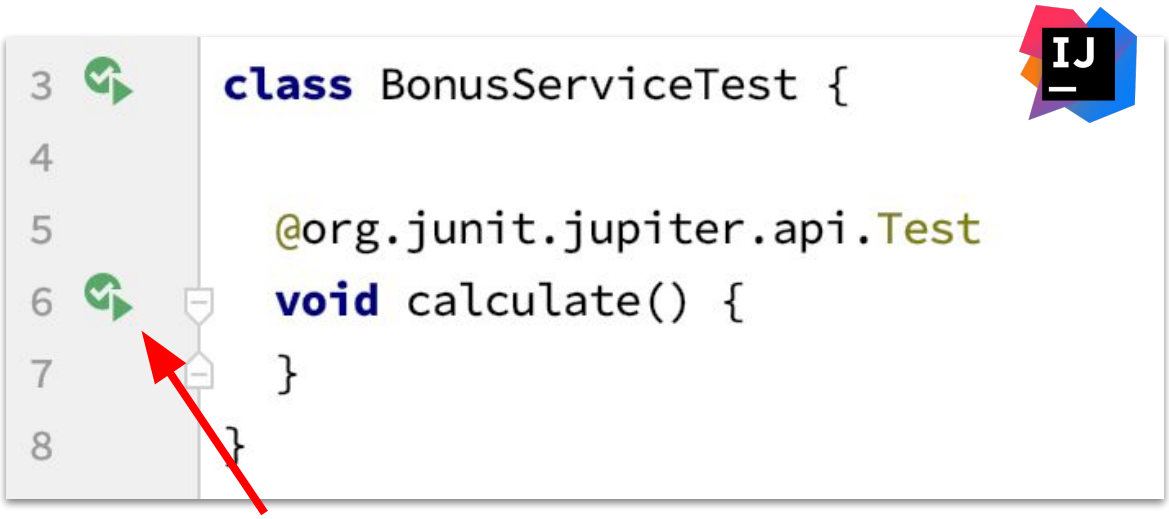
1. На месте возвращаемого типа стоит `void` — это значит метод ничего не возвращает (не нужен `return`)
2. Скобки для параметров пусты — значит метод не принимает никаких входных данных
3. Над методом стоит конструкция `@org.junit.jupiter.api.Test` — это аннотация

---

*Если ваш тест не работает - проверьте, что импорт аннотации верный

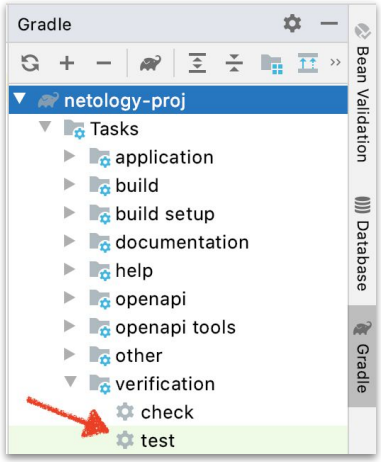
Как запустить

Чтобы запустить тест, воспользуйтесь иконкой “play”.



Как запустить

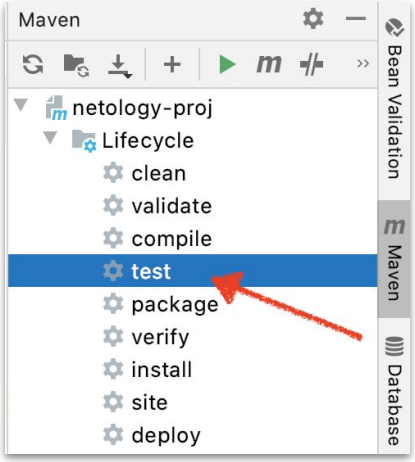
1



2

```
3  class BonusServiceTest {
4
5      @org.junit.jupiter.api.Test
6      void calculate() {
7      }
8  }
```

3



Gradle

IJ

Maven™

Как это работает

1. Когда мы запускаем тесты, запускается JUnit (как раньше запускался наш `Main`).
2. JUnit ищет все классы в каталоге `src/test/java` над методами которых стоит `@Test`.
3. Для каждого метода (с `@Test`) — создаёт объект из класса и вызывает метод (это и есть тест).
4. Для каждого вызова — проверяет результат.

Дополнительные методы

```
class NetologyTesting {
    private static long suiteStartTime;
    private long testStartTime;

    @BeforeAll
    public static void initSuite() {
        System.out.println("Running StringTest");
        suiteStartTime = System.nanoTime();
    }

    @AfterAll
    public static void completeSuite() {
        System.out.println("StringTest complete: " + (System.nanoTime() - suiteStartTime));

    }

    @BeforeEach
    public void initTest() {
        System.out.println("Starting new nest");
        testStartTime = System.nanoTime();
    }

    @AfterEach
    public void finalizeTest() {
        System.out.println("Test complete:" + (System.nanoTime() - testStartTime));
    }
}
```

Настройка фикстуры

Ещё ассерты

assertTrue(condition)	Проверить что condition (типа boolean) является true
assertFalse(condition)	Проверить что condition (типа boolean) является false
assertEquals(expected, actual)	Проверить равенство expected и actual аналогично Objects.equals(expected, actual)
assertNotEquals(expected, actual)	Проверить **не**равенство expected и actual аналогично !Objects.equals(expected, actual)
assertNull(value)	Проверить что value является null
assertNotNull(value)	Проверить что value **не** является null
assertArrayEquals(expected, actual)	Проверить равенство массивов
assertLinesMatch(expected, actual)	Проверить равенство списков строк
Assertions.*	И всё, что вы найдёте в подсказке, набрав Assertions.

Параметризованные тесты

Тесты на наборе различных параметров:

```
@ParameterizedTest
@ValueSource(strings = { "Hello", "World" })
public void testWithStringParameter(String argument) {
   Assertions.assertTrue(argument.contains("o"));
}
```

Ещё больше параметризации >>

Попробуем?

Живой пример
написания
автотеста

Что ещё нужно знать?

Что ещё нужно знать?

TDD - вначале тесты. Код по ним

Что ещё нужно знать?

TDD - вначале тесты. Код по ним

BDD - вначале критерии приемочного тестирования (и иногда системные автотесты к ним)

Что ещё нужно знать?

TDD - вначале тесты. Код по ним

BDD - вначале критерии приемочного тестирования (и иногда системные автотесты к ним)

Hamcrest - библиотека для большей читабельности + больше возможностей

```
@Test
public void containsSimply() {
   List<String> list = List.of("hello", "netology",
"world");

   assertTrue(list.contains("hello"));
   assertTrue(list.contains("netology"));
}
```

```
@Test
public void containsCool() {
   List<String> list = List.of("hello", "netology",
"world");

   assertThat(list, hasItems("hello", "netology"));
}
```

Итоги

- **Основы**
  - Тесты - это важно
  - Основные понятия, уровни тестирования
  - Дополнительные преимущества тестов
- **Примеры на JUnit**
  - различные Assertions
  - проверка выброса исключений
  - параметризованные тесты
- **Модные веяния в сфере тестов (TDD, BDD)**
- **Вспомогательные инструменты** (Hamcrest, Mockito… на след. уроке)

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

Спасибо!

нетология

Задавайте вопросы и пишите отзыв о лекции!