

Parameterized Tests & Annotations

В JUnit существуют Parameterized Tests. Они дают возможность прогонять один и тот же тест с разными данными (в т.ч. ожидаемый результат).

По аналогии с ручным тестированием, мы можем прикрепить к тест-кейсу табличку с тестовыми данными, в которой перечислить, для каких входных данных нужно прогнать этот тест-кейс

JUnit активно использует аннотации — мы уже видели, что достаточно

«повесить» на метод аннотацию `@Test`, чтобы метод стал тестом.

То же самое с параметризованными тестами — нужно написать аннотацию

`@ParameterizedTest` и сообщить JUnit, откуда брать входные данные

Аннотации — это мета-данные, прикрепляемые к коду.

Фактически, это специальные «пометки», которые программист может оставлять для того, чтобы на основе этих пометок JUnit и другие

инструменты могли делать свою работу.

В большинстве случаев перед аннотацией будет символ `@` (но не всегда).

Аннотации можно писать над классами, методами, параметрами и т.д.

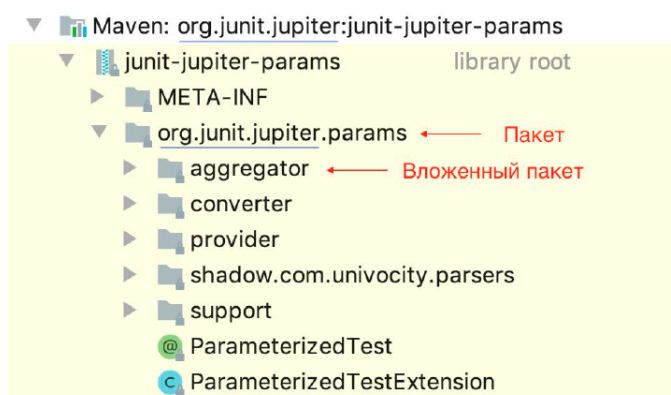
В Java все классы (и другие сущности) разделены по пакетам.

Пакет — это просто каталог (или их набор) на жёстком диске

Когда классов становится очень много, возникает две проблемы:

1. Очень трудно найти нужный.
2. Очень трудно придумывать новые имена классам (чтобы они не совпадали с уже существующими).

Часто принято в название пакета включать `GroupId` артефакта:



1. **CSV** — это не Java, здесь мы не используем `_` в числах.

2. Если встречается значение, содержащее запятую (название теста), то оно заключается в кавычки.

3. Одна строка — один набор данных.

```
'registered user, bonus under limit',100060,true,30  
'registered user, bonus over limit',100000060,true,500
```

Проще использовать excel для формата csv

Для того, чтобы использовать этот формат в наших тестах, мы можем

использовать аннотацию **@CsvSource**

Q: где мне узнать про все аннотации?

A: варианта два:

- В руководстве [на официальном сайте](#);
- Изучая пакеты библиотек:

`/** ... */` — это JavaDoc-комментарии, в которых описывается как работает эта аннотация.

Запись вида `String[]` — это массив

Строки в Java представлены классом `String` (поэтому и написано с большой буквы).

Строки — особый класс в Java, один из немногих, которые позволяют инициализировать себя без* вызова `new`

```
String testName = "registered user, bonus under limit";
```

У строк есть свой оператор `+`, который осуществляет конкатенацию (склеивание) двух строк в одну:

```
String testName = "registered user, " + «bonus under limit»;  
// Это позволяет длинные строки разбивать на несколько:  
String message = "Hello, dear User!" + "You registered on our service ...";
```

Массив — это упорядоченный набор данных фиксированной длины.

Когда нам нужно хранить набор данных (объектов или примитивов), мы можем использовать массивы:

```
// Объявление массива:  
int[] numbers; // массив из целых чисел  
String[] names; // массив из строк  
  
// Но неинициализированные переменные использовать нельзя!  
  
// Инициализация массива (вариант 1):  
int[] numbers = new int[3]; // массив из 3 чисел  
String[] names = new String[3]; // массив из 3 строк  
  
// Инициализация массива (вариант 2):  
int[] numbers = {1, 2, 3}; // массив из 3 чисел  
String[] names = {"Vasya", "Petya", "Masha"}; // массив из 3 строк
```

ОПЕРАЦИИ С МАССИВАМИ

С массивами можно выполнять две ключевые операции:

- чтение данных по индексу;
- запись данных по индексу.

```
String[] names = {"Vasya", "Petya", "Masha"}; // массив из 3 строк
// нумерация элементов идёт с нуля:
// — у «Vasya» индекс = 0
// — у «Petya» индекс = 1
// — у «Masha» индекс = 2

// Чтение по индексу:
System.out.println(names[0]);

// Запись по индексу:
names[0] = "Vasiliy Ivanovich";

// .length — свойство, хранящее длину массива
System.out.println(names.length);
```

До этого мы с вами работали только с методами, сейчас же познакомимся со свойствами (или полями объектов)*:

Т.е. обращение к свойству идёт с помощью оператора `.` после которого следует имя свойства.

В отличие от методов, мы не ставим круглые скобки — поскольку `()` после имени означают «вызов метода».

Свойство `length` предназначено только для чтения — в него ничего нельзя записать.

```
System.out.println(names.length);
```

@CSVSOURCE

```
@ParameterizedTest
@CsvSource(
    value={
        "registered user, bonus under limit,100060,true,30",
        "registered user, bonus over limit,100000060,true,500"
    },
    delimiter=',',
)
void shouldCalculate(String test, long amount, boolean registered, long expected) {
    BonusService service = new BonusService();
    // вызываем целевой метод:
    long actual = service.calculate(amount, registered);
    // производим проверку (сравниваем ожидаемый и фактический):
    assertEquals(expected, actual);
}
```

JUnit нумерует тесты с 1 а не с 0

В Java есть 4 вида циклов:

1. foreach;
2. for;
3. while;
4. do-while.

Сегодня мы рассмотрим только первый, как наиболее часто используемый

foreach

Здесь буквально «перебираются» все элементы из массива и на каждой итерации кладётся следующий элемент в переменную `purchase`.

```
// Продемонстрировать работу в дебаггере
int[] purchases = {1_000, 2_000, 500, 5_000, 2_000};
// IDEA: iter + tab
for (int purchase : purchases) {
    System.out.println(purchase);
}
```

Цикл останавливается после перебора всех элементов.

Важно: несмотря на то, что `purchase` здесь объявлена «вне блока», по факту — она видна только внутри `for`.