

# OBJECTS WITH INNER STATE

## ИНИЦИАЛИЗАЦИЯ

При создании объекта его поля инициализируются (т.е. им присваиваются) нулевые значения:

- для целых чисел — 0
- для вещественных чисел — 0.0
- для boolean — false
- для объектов — null

Для установки значения полей в Java (если поля приватные) у нас есть следующие возможности:

- инициализаторы полей
- блоки инициализации

3. конструкторы

4. методы (включая setter'ы)

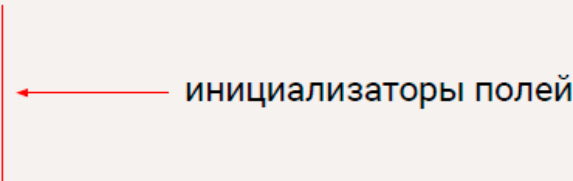
5. reflection API\*

Инициализаторы полей — это выражения, позволяющие задать начальные значения полям создаваемых объектов:

Как вы видите, не обязательно всем полям давать начальные значения. Если нас устраивают нулевые значения, то можно оставить их.

```
package ru.netology.domain.field;

public class Conditioner {
    private int id;
    private String name = "noname";
    private int maxTemperature = 30;
    private int minTemperature = 15;
    private int currentTemperature = 22;
    private boolean on;
}
```



В инициализаторах полей можно использовать не только литералы (вы ведь помните, что такое литералы?), но и:

- Оператор new (для создания новых объектов)
- Арифметические, логические и иные выражения

3. Вызовы методов

4. Обращение к собственным полям

Несмотря на то, что в инициализаторах можно многое, мы не рекомендуем усложнять: инициализаторы должны быть короткими

и понятными

В инициализаторах полей можно использовать только выражения (синтаксические конструкции, возвращающие ровно одно значение).

К выражениям не относятся:

- условия
- циклы
- блоки кода

## TESTABILITY

Важный момент: достаточно часто в классы добавляют методы для удобства тестирования, в том числе getter'ы и setter'ы.

Например, несмотря на то, что на пульте кондиционера физически нет кнопок установки температуры (кроме как +/-), мы можем добавить\* getter'ы и setter'ы для этого поля **только для удобства тестирования**.

**Примечание\*:** ни в коем случае не добавляйте самовольно код в чужие классы (написанные другими программистами) для упрощения тестирования! Обязательно согласовывайте это с ними, а ещё лучше — просите их самих добавить такой код.

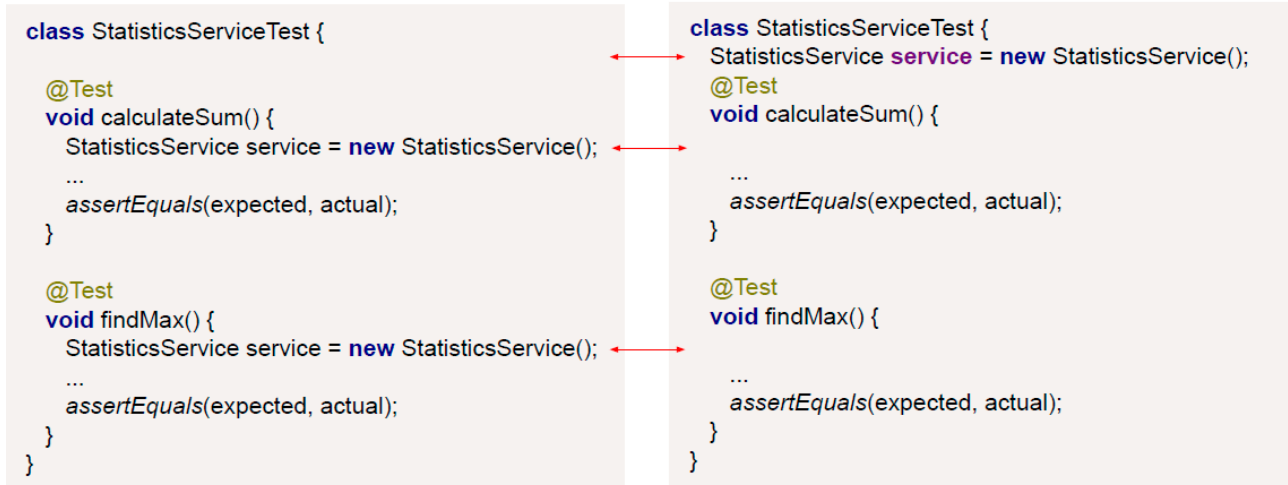
В реальной жизни мы видим то же самое: в автомобиле достаточно много датчиков (getter'ы), которые позволяют получать различную информацию о состоянии автомобиля:

В то же время есть возможности, позволяющие настраивать некоторое состояние: например, можно сбросить «дневной пробег»

(это не setter в чистом виде, но некий аналог установки состояния)

К тестам не всегда предъявляются те же требования, что и к обычному коду, но мы рекомендуем вам следовать правилам и вырабатывать единый стиль.

С точки зрения автотестеров инициализаторы полей можно использовать в тестах:



## КОНСТРУКТОРЫ

**Конструкторы (constructors)** — именованные блоки кода, очень похожие на методы. Их ключевая задача — инициализация объекта.

Давайте создадим новый package constructor, в который скопируем наш класс и скомпилируем приложение (mvn compile).

Итак, **default constructor** — это блок кода, генерируемый компилятором Java, который:

1. имеет такой же access modifier, как класс
2. не имеет возвращаемого значения
3. имеет название, соответствующее названию класса
4. не имеет параметров
5. имеет пустое тело\*

Когда мы пишем **new** Conditioner(), происходит вызов default constructor'a, который генерируется компилятором.

Именно поэтому мы не писали ничего в скобках (у дефолтного конструктора нет параметров).

Создадим свой конструктор: Alt + Insert (у IDEA есть конструктор), Ctrl + A, OK.

## NO ARGS & ALL ARGS

Такие конструкторы без параметров и со всеми параметрами, соответствующими полям, часто называют No Args Constructor и

All Args Constructor соответственно.

```
public class Conditioner {  
    ...  
  
    public Conditioner(  
        int id,  
        String name,  
        int maxTemperature,  
        int minTemperature,  
        int currentTemperature,  
        boolean on  
    ){  
        this.id = id;  
        this.name = name;  
        this.maxTemperature = maxTemperature;  
        this.minTemperature = minTemperature;  
        this.currentTemperature = currentTemperature;  
        this.on = on;  
    }  
}
```

мы немного изменили форматирование,  
чтобы уместилось на экран

Получился «один setter на всё».

Примечание\*: не обязательно делать конструктор на все поля, можно выбрать только нужные или не выбирать ни одного вообще.

Теперь мы можем сразу при создании указывать нужное состояние (и можно не вызывать по 10 раз метод увеличения температуру):

```
class ConditionerTest {  
    @Test  
    public void shouldUseConstructor() {  
        Conditioner conditioner = new Conditioner(  
            1,  
            "Winter Cold",  
            30,  
            10,  
            30,  
            true  
        );  
        assertEquals(30, conditioner.getCurrentTemperature());  
    }  
}
```

**Перегрузка** (overloading) в Java — механизм, при котором два метода (включая конструкторы) в классе\* могут иметь **одинаковые имена, но разные сигнатуры**

Сигнатура — это набор из декларации методов, состоящий из:

1. имени метода
2. списка параметров (количество, порядок и типы)

Обратите внимание:

- в сигнатуру не входит тип возвращаемого значения
- важно только количество, порядок и типы аргументов (а не их названия).

Q: Зачем это может быть нужно?

A: По нескольким причинам:

1. чтобы каждый раз не придумывать новые имена методам
2. чтобы реализовать разную логику для разных параметров
3. чтобы предоставить значение параметров по умолчанию

Q: Как Java определяет, какой из двух методов использовать?

A: Ответ на этот вопрос не так прост (целиком он описан в спецификации). В упрощённой форме — по количеству, порядку и типам аргументов при вызове ищется наиболее подходящий.

Имена параметров не учитываются, только количество, типы и порядок.

## BOILERPLATE

Q: если конструкторы не содержат никакой логики (как getter'ы и setter'ы могут её не содержать), не слишком ли утомительно каждый раз писать практически одинаковый код?

A: верное замечание, такой код называют **boilerplate code** (кусок кода, который встречается во множестве мест без изменения или с небольшими изменениями) и даже с генератором IDEA. Писать такой код малопрятно.

**Code Generation** — термин, описывающий автоматическую генерацию кода. Было бы здорово генерировать boilerplate код конструкторов и getter'ов/setter'ов.

И действительно, существует специальный инструмент (и плагин к Maven), который умеет генерировать такой код.

[Lombok](#) — библиотека, которая занимается генерацией кода.

С её использованием наш код будет выглядеть вот так:

```
@NoArgsConstructor ← Конструктор без параметров
@Data ← Конструктор с параметрами на все поля*
        getter'ы/setter'ы на все поля + ряд методов
public class Conditioner {
    private int id;
    private String name = «noname»;
    private int maxTemperature = 30;
    private int minTemperature = 15;
    private int currentTemperature = 22;
    private boolean on;
}
```

В разных командах существует разное отношение к Lombok, вплоть до полного неприятия.

Мы крайне настоятельно предостерегаем вас от подобного отношения к любым инструментам: если инструмент популярен, то

вы должны уметь им пользоваться без эмоциональной окраски

Сегодня мы поговорили о том, насколько важно иметь возможность устанавливать нужное нам состояние объектов. Это ключевой

аспект тестопригодности (testability) всей системы.

Мы узнали, что иногда следует добавлять методы, которые позволяют легко устанавливать/проверять состояние объекта

только для того, чтобы этот объект было легко тестировать