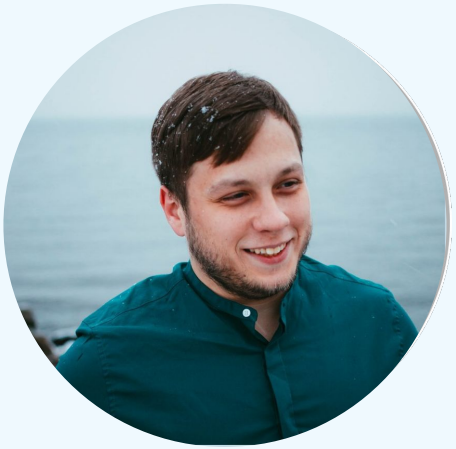
нетология

Основы ООП — структура класса

Филипп
Воронов



Филипп Воронов

Teamlead, VK Group

@Филипп Воронов

План занятия

1. Структура класса
2. Понятие поля и метода
3. Конструкторы
4. Модификатор `static`

Структура класса

Понятие класса

Прежде чем описывать понятие класса в Java, важно объяснить такой термин как парадигма программирования — это общее понятие, в котором описываются принципы построения и работы программы на конкретном языке.

Парадигм изобретено достаточно много, при разработке на Java обычно сталкиваются с основными тремя:

1. **Императивное программирование** — описывает шаг за шагом выполняемые действия.
2. **Функциональное программирование** — как можно догадаться, оперирует функциями, результатами выполнения этих функции и не хранит явно состояние программы.
3. **Объектно-ориентированное программирование** (сокращенно часто называют ООП) — оперирует описанными объектами, которые так или иначе взаимодействуют друг с другом.

Понятие класса

Язык Java создавался как объектно-ориентированный язык — это значит, что помимо примитивных типов в языке принято оперировать объектами, приближенными к реальным предметам. Например, объект из реального мира «музыкальный трек» обладает следующими характеристиками:

- исполнитель;
- альбом;
- автор;
- продолжительность;
- год записи.

Понятие класса

Для того чтобы описывать объекты в Java, были придуманы классы. Класс — это специальная структура, с помощью которой описываются будущие объекты в Java. Чтобы описать класс нужно:

1. Создать файл с именем класса и расширением Java;
2. Внутри этого файла добавить код `class` "Имя файла".

Понятие класса

Пример самого простого пустого класса в файле Record.java:

```
class Record {
}
```

Специальный оператор `class` говорит компилятору, что начинается описание класса в исходном файле.

Компилятор — программа, преобразующая файлы с исходным кодом (в языке Java — это файлы с расширением java) в файлы, исполняемые компьютером (операционной системой).

Понятие класса

Перенесем описание характеристик музыкального трека в созданный класс-файл:

```
class Record {
    String singer;
    String album;
    String author;
    int duration;
    int year;
}
```

Имена классов в Java принято называть с большой буквы (в нашем случае это Record). Имена классов, переменных, методов — должны начинаться с буквы (это требование спецификации языка).

Понятие поля и метода

Понятие полей и методов класса

Рассмотрим более подробное описание класса в Java:

1. Каждый класс описывается в пределах фигурных скобок `class Author { ... }`;
2. Каждый класс может содержать поля — это объявление характеристик объекта. Если посмотреть пример выше — то полями класса `Record` являются `singer`, `album`, `author`, `duration`, `year`;
3. Поле может быть, как примитивного, так и ссылочного типа (любой объект);
4. Класс может содержать в себе методы — функции которые может выполнять объект.

Понятие полей и методов класса

Давайте создадим новый класс музыкальная студия со следующими полями:

- название;
- владелец;
- год основания;
- количество выпущенных треков.

Какие типы данных вы будете использовать для каждого из этих полей?

Понятие полей и методов класса

Мы получим следующий класс:

```
class RecordStudio {
    String name;
    String owner;
    int foundationYear;
    int records;
}
```

Понятие полей и методов класса

В текущей записи наш класс умеет только хранить данные, давайте расширим его методом `makeNewRecord` — этот метод будет увеличить количество сделанных записей нашей студией.

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    void makeNewRecord() {
        records++;
    }
}
```

Понятие полей и методов класса

Для того чтобы описать метод нам нужно объявить:

- тип возвращаемого значения, в нашем случае это `void` который говорит, что наш метод ничего не возвращает;
- имя метода;
- в круглых скобках передаются входные параметры метода, в нашем случае их нет, поэтому там пусто;
- тело метода описывается так же как и класс в фигурных скобках `{...}`.

Понятие полей и методов класса

Теперь давайте добавим новый метод, который будет менять название `label` студии на случай ребрендинга. Этот метод должен принимать на вход новое название студии и заменять значение `label` у нашего объекта. Прежде чем начать ответьте на три вопроса:

1. Какое название метода будет хорошо описывать результат его выполнения;
2. Какое имя входного параметра будет лучше отражать его значение;
3. Насколько быстро другой разработчик поймет делает наш метод исходя из выбранных выше значений.

Решение

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    void makeNewRecord() {
        records++;
    }

    void setLabel(String newLabel) {
        label = newlabel;
    }
}
```

Конструкторы

Понятие конструктор

Только что мы научились описывать новые классы, но для работы программы нам нужно научиться использовать эти классы — создавать из них объекты.

Для того, чтобы создать новый объект из класса в Java, используется оператор `new`. При его вызове в памяти JVM (компьютера/телефона) выделяется область для хранения данных об этом объекте.

Понятие конструктор

Давайте создадим из класса `RecordStudio` новый объект (создадим свою студию).

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    void makeNewRecord() {
        records++;
    }

    void setLabel(String newLabel) {
        label = newlabel;
    }

    public static void main(String[] args) {
        new RecordStudio();
    }
}
```

Понятие конструктор

Мы добавили метод `main` (точка входа в программу), создали новый объект `RecordStudio`.

Давайте посмотрим, что происходит и откуда взялся new `RecordStudio()`, похожий на вызов метода. Если заглянуть в документацию, то можно увидеть, что после оператора new должен следовать так называемый конструктор класса.

Конструктор — это специальный метод класса, который ничего не возвращает, но описывает, каким образом должен быть собран/инициализирован класс. Например, какие должны быть значения полей у только что созданного объекта.

Конструкторов у класса может быть сколь угодно.

Понятие конструктор

Если у класса не создать конструктор в описании класса, то компилятор создаст его сам, как произошло в нашем случае с классом `RecordStudio`.

Наш класс получил конструктор без параметров по умолчанию, что это значит?

Что все поля, если они не примитивные значения, примут значения `null`.

Понятие конструктор

Вот что мы получили после компиляции нашего класса:

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    public RecordStudio() {
    }

    void makeNewRecord() {
        records++;
    }

    void setLabel(String newLabel) {
        label = newlabel;
    }

    public static void main(String[] args) {
        new RecordStudio();
    }
}
```

Понятие конструктор

`public RecordStudio()` – конструктор без параметров по-умолчанию.

Вызывая каждый раз конструктор `new RecordStudio()`

```
public static void main(String[] args) {
    new RecordStudio();
    new RecordStudio();
    new RecordStudio();
}
```

Понятие конструктор

Мы создадим музыкальные студии с одинаково пустым именем, одинаковым годом, владельцем и остальным полями. В таком виде мы не можем обращаться к созданным объектам, потому что у нас нет ссылки на каждый из этих объектов.

Ссылка на объект должна быть того же типа, что и создаваемый объект:

```
public static void main(String[] args) {
    RecordStudio captainRecords = new RecordStudio();
    RecordStudio goldmanRecords = new RecordStudio();
    RecordStudio mainstreamRecords = new RecordStudio();
}
```

Понятие конструктор

`captainRecords`, `goldmanRecords`, `mainstreamRecords` — имена ссылок на объект типа `RecordStudio`.

Чтобы менять значения полей у объекта после создания, нужно присвоить значение поля, обратившись к нему по имени.

```
public static void main(String[] args) {
    RecordStudio captainRecords = new RecordStudio();
    RecordStudio goldmanRecords = new RecordStudio();
    RecordStudio mainstreamRecords = new RecordStudio();

    captainRecords.label = "Captain Records";
    captainRecords.owner = "Unkown";
    captainRecords.foundationYear = 2010;
    captainRecords.records = 131235;

    //TODO
}
```

Понятие конструктор

Чтобы сократить код, достаточно создать конструктор, принимающий на вход параметры:

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    public RecordStudio(String label, String owner, int foundationYear, int
records) {
        this.label = label;
        this.owner = owner;
        this.foundationYear = foundationYear;
        this.records = records;
    }

    void makeNewRecord() {
        records++;
    }

    void setLabel(String newLabel) {
        label = newlabel;
    }
}
```

Понятие конструктор

Теперь, чтобы заполнить объект, при создании достаточно передать параметры новому конструктору:

```
public static void main(String[] args) {
    RecordStudio captainRecords = new RecordStudio("Captain Records", "Unkown",
2010, 131235);
}
```

Если в классе описан хотя бы 1 конструктор класса, компилятор не создаст конструктор по умолчанию, если вы хотите по прежнему его использовать, нужно явно добавить его в код создаваемого класса.

Понятие конструктор

```
class RecordStudio {
    String label;
    String owner;
    int foundationYear;
    int records;

    public RecordStudio() {
    }

    public RecordStudio(String label, String owner, int
foundationYear, int records) {
        this.label = label;
        this.owner = owner;
        this.foundationYear = foundationYear;
        this.records = records;
}
```

Чтобы внутри метода отличить параметр от поля, если они имеют одинаковые наименования, используется `this` — ссылка на объект.

Вспоминаем прошлые занятия

- Самый часто используемый класс в Java?
- Как сравниваются объекты в Java?

Модификатор `static`

Модификатор `static`

При создании объектов `RecordStudio` между собой у них нет ничего общего, кроме класса, из которого создается каждый объект, поля имеют свои собственные значения. Но что делать, если мы хотим хранить общую информацию? Например, считать количество создаваемых объектов или иметь общую переменную для всех объектов (например, константное поле).

Модификатор `static`

Можно создать отдельную переменную и считать в ней.

```
public static void main(String[] args) {
    int counter = 0;
    RecordStudio captainRecords = new RecordStudio("Captain
Records", "Unkown", 2010, 131235);
    counter++;
    RecordStudio goldmanRecords = new RecordStudio("Goldman
Records", "Unkown", 2007, 45135);
    counter++;
    RecordStudio mainstreamRecords = new RecordStudio("Mainstream
Records", "Unkown", 2005, 651235);
    counter++;
}
```

Модификатор `static`

У этого подхода есть минус: нужно отслеживать изменения в разных частях программы и увеличивать переменную `counter`.

Второй вариант — создать переменную в классе с модификатором `static` — это значит создать переменную, значение которой будет храниться не в поле созданного объекта, а принадлежать всему описанному классу.

Пример

```
class RecordStudio {
    static int counter = 0;
    String label;
    String owner;
    int foundationYear;
    int records;

    public RecordStudio() {
        counter++;
    }

    public RecordStudio(String label, String owner, int
foundationYear, int records) {
        this.label = label;
        this.owner = owner;
        this.foundationYear = foundationYear;
        this.records = records;
        counter++;
}
```

Модификатор `static`

Мы добавили `static int counter` с начальным значением `0`, а в конструкторе эта переменная будет увеличиваться при каждом создании объекта.

Для обращения к статическим полям не требуется создание объекта:

```
public static void main(String[] args) {
   RecordStudio captainRecords = new RecordStudio("Captain Records", "Unkown",
2010, 131235);
   System.out.println("Количество созданных объектов = " +
RecordStudio.counter);
}
```

Если указать оператор `static` перед методом класса, он также будет принадлежать созданному классу — это значит, что при вызове таких методов не требуется создание объекта.

оператор `static` удобно использовать с утильными методами (вспомогательные методы — utils)

Модификатор `static`

Пример утильного класса Utils с методом, который удаляет пробелы в названии студии:

```
class Utils {
    static String trimStudioName(RecordStudio studio) {
        if (studio.label != null) {
            return studio.label.trim();
        }
        return "";
    }
```

Модификатор `static`

Вызовем созданный статический метод `trimStudioName`:

```
public static void main(String[] args) {
   RecordStudio captainRecords = new RecordStudio("Captain Records", "Unkown",
2010, 131235);
   System.out.println(Utils.trimStudioName(captainRecords));
}
```

Метод `trimStudioName` не требует создания объекта, и для вызова используется сигнатура `Utils.trimStudioName(...)`.

Важно — из статических методов доступны только статические переменные, т.к. обычные переменные принадлежат конкретным объектам.

Чему мы научились

- Узнали, как создавать объекты;
- Узнали, как создавать новые конструкторы;
- Узнали, для чего используется оператор `static`.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

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

нетология

Задавайте вопросы и пишите отзыв о лекции!

Филипп Воронов

@Филипп Воронов