Содержание

3. Специализация: ООП

3.1. В предыдущем разделе

Будет рассмотрен базовый функционал языка, то есть основная встроенная функциональность, такая как математические операторы, условия, циклы, бинарные операторы. Далее способы хранения и представления данных в Java, и в конце способы манипуляции данными, то есть функции (в терминах языка называющиеся методами).

3.2. В этом разделе

Разберём такие основополагающих в Java вещи, как классы и объекты, а также с тем, как применять на практике основные принципы ООП: наследование, полиморфизм и инкапсуляцию. Дополнительно рассмотрим устройство памяти в джава.

— Класс;
— Объект;
— Статика;
— Стек;
— Куча;
— Сборщик мусора;
— Конструктор;
— Вложенный класс;
— Внутренний класс;
— Инкапсуляция;
— Наследование;
— Полиморфизм ;

3.3. Классы и объекты, поля и методы, статика

3.3.1. Классы

Что такое класс? С точки зрения ООП, **класс** определяет форму и сущность объекта и является логической конструкцией, на основе которой построен весь язык Java.

Наиболее важная особенность класса состоит в том, что он определяет новый тип данных, которым можно воспользоваться для создания объектов этого типа

То есть класс — это шаблон (чертёж), по которому создаются объекты (экземпляры класса). Для определения формы и сущности класса указываются данные, которые он должен содержать, а также код, воздействующий на эти данные. Создаем мы свои классы, когда у нас не хватает уже созданных.

Например, если мы хотим работать в нашем приложении с документами, то необходимо для начала объяснить приложению, что такое документ, описать его в виде класса (чертежа) `Document`. Указать, какие у него должны быть свойства: название, содержание, количество страниц, информация о том, кем он подписан и т.п. В этом же классе мы обычно описываем, что можно делать с документами: печатать в консоль, подписывать, изменять содержание, название и т.д. Результатом такого описания и будет класс `Document`. Однако, это по-прежнему всего лишь чертеж хранимых данных (состояний) и способы взаимодействия с этими данными.

Если нам нужны конкретные документы, а нам они обязательно нужны, то необходимо создавать **объекты**: документ №1, документ №2, документ №3. Все эти документы будут иметь одну и ту же структуру (описанные нами название, содержание, ...), с ними можно выполнять одни и те же описанные нами действия (печатать, подписать, ...), но наполнение будет разным, например, в первом документе содержится приказ о назначении работника на должность, во втором, о выдаче премии отделу разработки и т.д.

Начнём с малого, напишем свой первый класс. Представим, что необходимо работать в приложении с котами. Java ничего не знает о том, что такое коты, поэтому необходимо создать новый класс (тип данных), и объяснить что такое кот. Создадим новый файл, для простоты в том же пакете, что и главный класс программы.

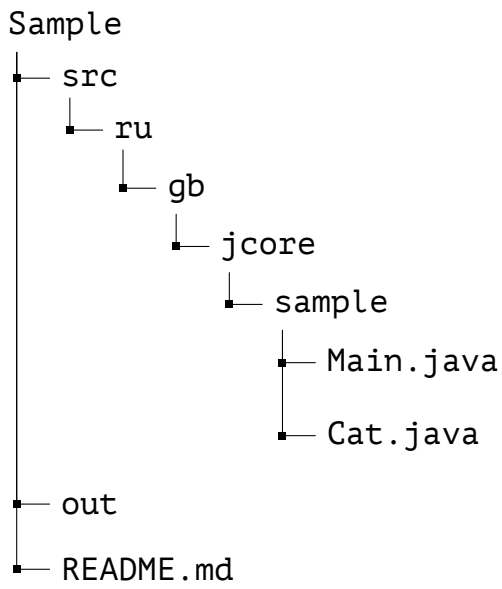

Рис. 1: Структура проекта

3.3.2. Поля класса

Начнем описывать в классе `Cat` так называемый API кота. Как известно, имя класса должно совпадать с именем файла, в котором он объявлен, т.е. класс `Cat` должен находиться в файле `Cat.java`. Пусть у котов есть три свойства: `name` (кличка), `color` (цвет) и `age` (возраст); совокупность этих свойств называется состоянием, и коты пока ничего не умеют делать. Класс `Cat` будет иметь вид, представленный в листинге 1. Свойства класса, записанные таким образом, в виде переменных, называются **полями**.

Листинг 1: Структура кота в программе

```
1  package ru.gb.jcore;
2
3  public class Cat {
4      String name;
5      String color;
6      int age;
7  }
```

Для новичка важно не запутаться, класс кота мы описали в отдельном файле, а создавать объекты и совершать манипуляции следует в основном классе программы, не может же кот назначить имя сам себе.

Мы рассказали программе, что такое коты, теперь если мы хотим создать в нашем приложении конкретного кота, следует воспользоваться оператором `new Cat();` в основном классе программы. Более подробно разберём, что происходит в этой строке, чуть позже, пока же нам достаточно знать, что мы создали объект типа `Cat` (экземпляр класса `Cat`), и запомнить эту конструкцию. Для того чтобы с ним (экземпляром) работать, можем положить его в переменную, которой дать идентификатор `cat1`. При создании объекта полям присваиваются значения по умолчанию (нули для числовых переменных и `false` для булевых).

```
Cat cat0; // cat0 = null;
cat0 = new Cat();
Cat cat1 = new Cat();
```

В листинге выше можно увидеть все три операции (объявление, присваивание и инициализацию) и становится понятно, как можно создавать объекты. Также известно, что в переменной не лежит сам объект, а только ссылка на него. Объект `cat1` создан по чертежу `Cat`, это значит, что у него есть поля `name`, `color`, `age`, с которыми можно работать: получать или изменять их значения.

Для доступа к полям объекта используется оператор точка, который связывает имя объекта с именем поля. Например, чтобы присвоить полю `color` объекта `cat1` значение «Белый», нужно выполнить код `cat1.color = "Белый";`

Операция «точка» служит для доступа к полям и методам объекта по его имени. Мы уже использовали оператор «точка» для доступа к полю с длиной массива, например. Рассмотрим пример консольного приложения, работающего с объектами класса `Cat`. Создадим двух котов, один будет белым Барсиком 4х лет, второй чёрным Мурзиком шести лет, и просто выведем информацию о них в терминал.

```
package ru.gb.jcore;

public class Main {
   public static void main(String[] args) {
      Cat cat1 = new Cat();
      Cat cat2 = new Cat();

      cat1.name = "Barsik";
      cat1.color = "White";
      cat1.age = 4;

      cat2.name = "Murzik";
      cat2.color = "Black";
      cat2.age = 6;

      System.out.println("Cat1 named: " + cat1.name +
            " is " + cat1.color +
            " has age: " + cat1.age);
      System.out.println("Cat2 named: " + cat2.name +
            " is " + cat2.color +
            " has age: " + cat2.age);
   }
}
```

в результате работы программы в консоли появятся следующие строки:

```
Cat1 named: Barsik is White has age: 4
Cat2 named: Murzik is Black has age: 6
```

Вначале мы создали два объекта типа `Cat`: `cat1` и `cat2`, соответственно, они имеют одинаковый набор полей `name`, `color`, `age`. Почему? Потому что они принадлежат одному классу, созданы по одному шаблону. Объекты всегда «знают», какого они класса. Однако каждому из них в эти поля записаны разные значения. Как видно из результата печати в консоли, изменение значения полей одного объекта, никак не влияет на значения полей другого объекта. Данные объектов `cat1` и `cat2` изолированы друг от друга. А значит мы делаем

вывод о том, поля хранятся в классе, а значения полей хранятся в объектах. Логическая структура, демонстрирующая отношения объектов и классов, в том числе в части хранения полей и их значений показана на рис. 2.

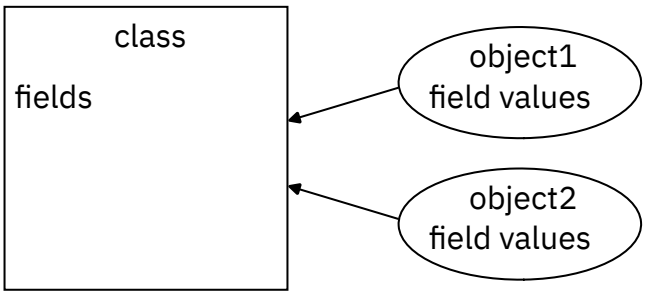

Рис. 2: Логическая структура отношения класс-объект

3.3.3. Объекты

Разобравшись с тем, как создавать новые типы данных (классы) и мельком посмотрев, как создаются объекты, нужно подробнее разобрать, как создавать объекты, и что при этом происходит. Создание объекта как любого ссылочного типа данных проходит в два этапа. Как и в случае с уже известными нам массивами.

- Сначала создается переменная, имеющая интересующий нас тип, в неё возможно записать ссылку на объект;
- затем необходимо выделить память под объект;
- создать и положить объект в выделенную часть памяти;
- и сохранить ссылку на этот объект в памяти - в нашу переменную.

Для непосредственного создания объекта применяется оператор `new`, который динамически резервирует память под объект и возвращает ссылку на него, в общих чертах эта ссылка представляет собой адрес объекта в памяти, зарезервированной оператором `new`.

```
1 Cat cat1; // cat1 = null;
2 cat1 = new Cat();
3 Cat cat2 = new Cat();
```

В первой строке кода переменная `cat1` объявляется как ссылка на объект типа `Cat` и пока ещё не ссылается на конкретный объект (первоначально значение переменной `cat1` равно `null`). В следующей строке выделяется память для объекта типа `Cat`, и в переменную `cat1` сохраняется ссылка на него. После выполнения второй строки кода переменную `cat1` можно использовать так, как если бы она была объектом типа `Cat`. Обычно новый объект создается в одну строку, то есть инициализируется.

3.3.4. Оператор new

[квалификаторы] ИмяКласса имяПеременной = **new** ИмяКласса();

Оператор `new` динамически выделяет память для нового объекта, общая форма применения этого оператора имеет вид как на врезке выше, но на самом деле справа - не имя

класса, конструкция ИмяКласса() в правой части выполняет вызов конструктора данного класса, который подготавливает вновь создаваемый объект к работе.

Именно от количества применений оператора new будет зависеть, сколько именно объектов будет создано в программе.

```
Cat cat1 = new Cat();
Cat cat2 = cat1;

cat1.name = "Barsik";
cat1.color = "White";
cat1.age = 4;

cat2.name = "Murzik";
cat2.color = "Black";
cat2.age = 6;

System.out.println("Cat1 named: " + cat1.name +
      " is " + cat1.color +
      " has age: " + cat1.age);
System.out.println("Cat2 named: " + cat2.name +
      " is " + cat2.color +
      " has age: " + cat2.age);
```

На первый взгляд может показаться, что переменной cat2 присваивается ссылка на копию объекта cat1, т.е. переменные cat1 и cat2 будут ссылаться на разные объекты в памяти. Но это не так. На самом деле cat1 и cat2 будут ссылаться на один и тот же объект. Присваивание переменной cat1 значения переменной cat2 не привело к выделению области памяти или копированию объекта, лишь к тому, что переменная cat2 содержит ссылку на тот же объект, что и переменная cat1. Это явление дополнительно подчёркивает ссылочную природу данных в языке Java.

Таким образом, любые изменения, внесённые в объект по ссылке cat2, окажут влияние на объект, на который ссылается переменная cat1, поскольку *это один и тот же объект в памяти*. Поэтому результатом выполнения кода, где мы как будто бы указали возраст второго кота, равный шести годам, станут строки, показывающие, что по обеим ссылкам оказался кот возраста шесть лет с именем Мурзика.

```
Cat1 named: Murzik is Black has age: 6
Cat2 named: Murzik is Black has age: 6
```

Множественные ссылки на один и тот же объект в памяти довольно легко себе представить как ярлыки для запуска одной и той же программы на рабочем столе и в меню быстрого запуска. Или если на один и тот же шкафчик в раздевалке наклеить два номера - сам шкафчик можно будет найти по двум ссылкам на него.

Важно всегда перепроверять, какие объекты созданы, а какие имеют множественные ссылки.

3.3.5. Методы

Ранее было сказано о том, что в языке Java любая программа состоит из классов и функций, которые могут описываться только внутри них. Именно поэтому все функции в языке

Java являются методами. А метод - это функция, являющаяся частью некоторого класса, которая может выполнять операции над данными этого класса.

Метод - это функция, принадлежащая классу

Метод для своей работы может использовать поля объекта и/или класса, в котором определен, напрямую, без необходимости передавать их во входных параметрах. Это похоже на использование глобальных переменных в функциях, но в отличие от глобальных переменных, метод может получать прямой доступ только к членам класса. Самые простые методы работают с данными объектов. Методы чаще всего формируют API классов, то есть способ взаимодействия с классами, интерфейс. Место методов во взаимодействии классов и объектов показано на рис. 3.

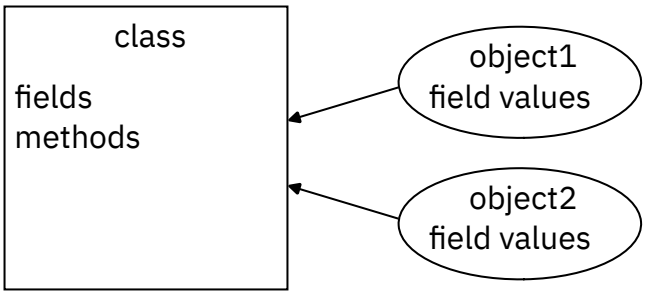

Рис. 3: Логическая структура отношения класс-объект

Вернёмся к примеру с котиками. Широко известно, что котики умеют урчать, мяукать и смешно прыгать. В целях демонстрации в описании этих действий просто будем делать разные выводы в консоль, хотя возможно и научить котика в программе выбирать минимальное значение из массива, но это было бы, как минимум, неожиданно. Итак опишем метод например подать голос и прыгать.

```
1  public class Cat {
2     String name;
3     String color;
4     int age;
5
6     void voice() {
7        System.out.println(name + " meows");
8     }
9
10    void jump() {
11       if (this.age < 5) System.out.println(name + " jumps");
12    }
13 }
```

Обращение к методам выглядит очень похожим на стандартный способом, через точку, как к полям. Теперь когда появляется необходимость позвать котика, он скажет: «мяу, я имя котика», а если в программе пришло время котику прыгнуть, он решит, прилично ли это – прыгать в его возрасте.

```
1  package ru.gb.jcore;
2
3  public class Main {
4     public static void main(String[] args) {
```

```
        Cat cat1 = new Cat();
        Cat cat2 = new Cat();

        cat1.name = "Barsik";
        cat1.color = "White";
        cat1.age = 4;

        cat2.name = "Murzik";
        cat2.color = "Black";
        cat2.age = 6;

        cat1.voice();
        cat2.voice();
        cat1.jump();
        cat2.jump();
    }
}
```

```
Barsik meows
Murzik meows
Barsik jumps
```

Как видно, Барсик замечательно прыгает, а Мурзик от прыжков воздержался, хотя попрыгать программа попросила их обоих.

3.3.6. Ключевое слово `static`

В завершение базовой информации о классах и объектах, остановимся на специальном модификаторе `static`, делающем переменную или метод «независимыми» от объекта.

> **`static`** — модификатор, применяемый к полю, блоку, методу или внутреннему классу, он указывает на привязку субъекта к текущему классу.

Для использования таких полей и методов, соответственно, объект создавать не нужно. В Java большинство членов служебных классов являются статическими. Возможно использовать это ключевое слово в четырех контекстах:

- статические методы;
- статические переменные;
- статические вложенные классы;
- статические блоки.

В этом разделе рассмотрим подробнее только первые два пункта, третий опишем чуть позже, а четвёртый потребует от нас знаний, выходящих не только за этот урок, но и за десяток следующих.

Статические методы также называются методами класса, потому что статический метод принадлежит классу, а не его объекту. Нестатические называются методами объекта. Статические методы можно вызывать напрямую через имя класса, не обращаясь к объекту и вообще объект не создавая. Что это и зачем нужно? Например, умение кота мяукать можно вывести в статическое поле, потому что, например, весной можно открыть окно, не увидеть ни одного экземпляра котов, но зато услышать их, и точно знать, что мяукают не дома и не машины, а именно коты.

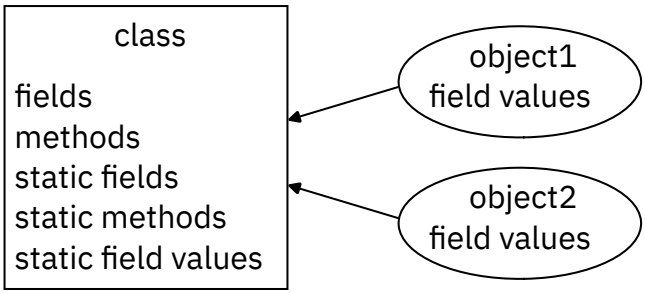

Рис. 4: Логическая структура отношения класс-объект

Аналогично статическим методам, **статические поля** принадлежат классу и совершенно ничего «не знают» об объектах.

Важной отличительной чертой статических полей является то, что их значения также хранятся в классе, в отличие от обычных полей, чьи значения хранятся в объектах.

Рисунок 4 именно в этом виде автор настоятельно рекомендует если не заучить, то хотя бы хорошо запомнить, он ещё пригодится в дальнейшем обучении и работе. Из этого же изображения можно сделать несколько выводов.

лайвкод 03-статическое-поле-код Помимо того, что статические поля - это полезный инструмент создания общих свойств это ещ§ и опасный инструмент создания общих свойств. Так, например, мы знаем, что у котов четыре лапы, а не 6 и не 8. Не создавая никакого барсика будет понятно, что у кота - 4 лапы. Это полезное поведение.

лайвкод 03-статическое-поле-ошибка Посмотрим на опасность. Мы видим, что у каждого кота есть имя, и помним, что коты хранят значение своего имени каждый сам у себя. А знают экземпляры о названии поля потому что знают, какого класса они экземпляры. Но что если мы по невнимательности добавим свойство статичности к имени кота?

03-статическое-поле-признак Создав тех же самых котов, которых мы создавали весь урок, мы получим двух мурзиков и ни одного барсика. Почему это произошло? По факту переменная у нас одна на всех, и значение тоже одно, а значит каждый раз мы меняем именно его, а все остальные коты ничего не подозревая смотрят на значение общей переменной и бодро его возвращают. Поэтому, чтобы не запутаться, к статическим переменным, как правило, обращаются не по ссылке на объект — cat1.name, а по имени класса — Cat.name.

03-статические-поля К слову, статические переменные — редкость в Java. Вместо них применяют статические константы. Они определяются ключевыми словами static final и по соглашению о внешнем виде кода пишутся в верхнем регистре.

3.3.7. Задание для самопроверки

1. Что такое класс?
2. Что такое поле класса?
3. На какие три этапа делится создание объекта?
4. Какое свойство добавляет ключевое слово static полю или методу?
   (a) неизменяемость;

(b) принадлежность классу;
(c) принадлежность приложению.

5. Может ли статический метод получить доступ к полям объекта?
   (a) не может;
   (b) может только к константным;
   (c) может только к неинициализированным.

3.4. Стек и куча

3.5. Сборка мусора

3.6. Конструкторы

3.7. Инкапсуляция

3.8. Наследование

3.9. Полиморфизм

- Также, хотелось бы отметить, что мы можем использовать -Xms и -Xmx опции JVM, чтобы определить начальный и максимальный размер памяти в куче. Для стека определить размер памяти можно с помощью опции -Xss;

```
public class Cat {
   private String name;
   private String color;
   private int age;

   public Cat() {
      System.out.println("constructor is constructing...");
      name = "Barsik";
      color = "White";
      age = 2;
   }
}
```

```
public class Cat {
   private String name;
   private String color;
   private int age;

   Cat(String n, String c, int a) {
      System.out.println("constructor is constructing...");
      name = n;
      color = c;
      age = a;
   }
}
```

```
  Cat cat1 = new Cat("Barsik", "White", 4);
  Cat cat2 = new Cat("Murzik", "Black", 6);
```

```
public class Cat {
    private String name;
    private String color;
```

```
    private int age;

    public Cat(String name, String color, int age) {
        this.name = name;
        this.color = color;
        this.age = age;
    }
}
```

ключевое слово this в Java используется только в составе экземпляра класса. Но неявно ключевое слово this передается во все методы, кроме статических (поэтому this часто называют неявным параметром) и может быть использовано для обращения к объекту, вызвавшему метод.