



Lesson 5

19.11.2020

```
public class Ex1 {
    static int a = 1111;

    static {
        System.out.println("static");
        a = a-- - --a;
    }

    {
        System.out.println("non static");
        a = a++ + ++a;
    }

    public static void main(String[] args) {
        System.out.println(a);
    }
}
```

```
public class Ex2 {  
    public static void main(String[] args) {  
        Integer i1 = 128;  
        Integer i2 = 128;  
        System.out.println(i1 == i2);  
  
        Integer i3 = 127;  
        Integer i4 = 127;  
        System.out.println(i3 == i4);  
    }  
}
```



```
public class Ex3 {  
    public static void show() {  
        System.out.println("Static method called");  
    }  
  
    public static void main(String[] args) {  
        Ex3 obj = null;  
        obj.show();  
    }  
}
```

```
public class Ex4 {  
    static int method1(int i) {  
        return method2(i *= 11);  
    }  
    static int method2(int i) {  
        return method3(i /= 11);  
    }  
    static int method3(int i) {  
        return method4(i -= 11);  
    }  
    static int method4(int i) {  
        return i += 11;  
    }  
    public static void main(String[] args) {  
        System.out.println(method1(11));  
    }  
}
```

Inheritance



you can create new type of animal
changing or adding properties

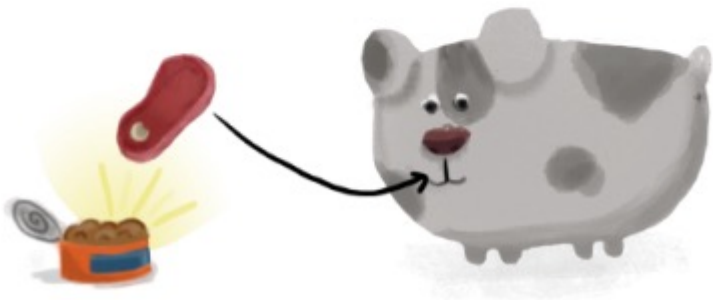


Incapsulation



every animal eats
and then poop

Polymorphism



each animal can eat
its own type of food

Функция — часть программы, имеющая собственное имя. Это имя можно использовать в программе как команду (такая команда называется вызовом функции). При вызове функции выполняются команды, из которых она состоит. Вызов функции может возвращать значение (аналогично операции) и поэтому может использоваться в выражении наряду с операциями.



Метод — это функция, являющаяся частью некоторого класса, которая может выполнять операции над данными этого класса. В языке Java вся программа состоит только из классов и функции могут описываться только внутри них. Именно поэтому все функции в языке Java являются методами.



Функции используются в программировании, чтобы уменьшить его сложность:

1. Вместо того, чтобы писать непрерывную последовательность команд, в которой вскоре перестаешь ориентироваться, программу разбивают на подпрограммы, каждая из которых решает небольшую законченную задачу, а потом большая программа составляется из этих подпрограмм (этот прием называется декомпозицией).

2. Уменьшается общее количество кода, потому что, как правило, одна функция используется в программе несколько раз.

3. Написанная однажды и всесторонне проверенная функция, может быть включена в библиотеку функций и использоваться в других программах (при этом не надо вспоминать, как была запрограммирована эта функция, достаточно знать, что она делает).

```
public static int methodName(int a, int b) {  
    // тело  
}
```


Где,

public static — модификатор;

int — возвращаемый тип;

methodName — имя метода;

int a, int b — перечень параметров.



Описание переменной выполняется по следующей схеме:

[модификаторы] Тип ИмяПеременной [=значение];

Модификаторами могут быть :


- любой из модификаторов доступа: public, protected, private, иначе уровень доступа переменной устанавливается по умолчанию пакетным.
- static – модификатор принадлежности – переменные, отмеченные этим модификатором, принадлежат классу, а не экземпляру класса и существуют в единственном числе для всех его объектов, создаются JVM в момент первого обращения к классу, допускают обращение до создания объекта класса

Обращение: **Имя класса.Имя компонента**

- final – переменная не может изменять своего начального значения, то есть, является именованной константой.
- transient – переменная не должна сохранять и восстанавливать значение при сериализации (записи в файл) объекта. Все статические переменные являются не сохраняемыми автоматически.




- Инициализация статических полей и блоков выполняется при загрузке класса
- Инициализация не статических элементов выполняется при создании объекта (при вызове конструктора)



**[модификаторы] ТипВозвращаемогоЗначения ИмяМетода (список Параметров)
[throws списокВыбрасываемыхИсключений] { Тело метода};**

Модификаторами могут быть

- любой из модификаторов доступа: public, protected, private, иначе уровень доступа метода устанавливается по умолчанию пакетным.
- static – модификатор принадлежности – методы, отмеченные этим модификатором, принадлежат классу и доступны до создания объектов. Статические методы могут оперировать только статическими переменными или локальными переменными, объявленными внутри метода. Статический метод не может быть переопределен.
- final – метод не может быть переопределен при наследовании класса.
- synchronized – при исполнении метода не может произойти переключение конкурирующих потоков
- abstract – нереализованный метод. Тело такого метода просто отсутствует. Если объявили некий метод класса абстрактным, то и весь класс надо объявить абстрактным.



Конструктор – это метод, назначение которого состоит в создании экземпляра класса.

Характеристики конструктора:

- Имя конструктора должно совпадать с именем класса;
- Если в классе не описан конструктор, компилятор автоматически добавляет в код конструктор по умолчанию;
- Конструктор не может быть вызван иначе как оператором new;
- Конструктор не имеет возвращаемого значения – так как он возвращает ссылку на создаваемый объект.
- Конструкторов может быть несколько в классе. В этом случае конструкторы называют перегруженными;

Последовательность действий при вызове конструктора

- Все поля данных инициализируются своими значениями, предусмотренными по умолчанию (0, false или null).
- Инициализаторы всех полей и блоки инициализации выполняются в порядке их перечисления в объявлении класса.
- Если в первой строке конструктора вызывается другой конструктор, то выполняется вызванный конструктор.
- Выполняется тело конструктора.

```
public class Konstr {  
    int width; // ширина коробки  
    ... int height; // высота коробки  
    ... int depth; // глубина коробки  
    ... // вычисляем объём коробки  
    ... int getVolume() {  
        ... return width * height * depth;  
        ... }  
    →  
    public static void main(String[] args) {  
        Konstr kons = new Konstr();  
        System.out.println("Объём коробки: " + kons.getVolume());  
    }  
}
```

```
public class Konstr {
    int width; // ширина коробки
    int height; // высота коробки
    int depth; // глубина коробки

    Konstr(int w, int h, int d) { // конструктор с параметрами
        width = w;
        height = h;
        depth = d;
    }
    // вычисляем объём коробки
    int getVolume() {
        return width * height * depth;
    }
    public static void main(String[] args) {
        Konstr kons=new Konstr(5,5,5); // вызов конструктора с параметрами
        System.out.println("Объём коробки: " + kons.getVolume());
        Konstr kons1=new Konst(); // ошибка, конструктор не определен
    }
```



Class vs. Object

Car class



Car
Objects



Green
Ford
Mustang
Gasoline



Red
Toyota
Prius
Electricity



Blue
Volkswagon
Golf
Deisel

Box myBox;



null

myBox = new Box();



Box

double height;

double depth;

double width;

Heap

stack

Heap

b1


b2

Box

double height;

double depth;

double width;



В языке Java при проектировании классов принято ограничивать уровень доступа к переменным с помощью модификатора `private` и обращаться к ним через геттеры и сеттеры.

Существуют правила объявления таких методов, рассмотрим их:

- Если свойство НЕ типа `boolean`, префикс геттера должно быть `get`. Например: `getName()` это корректное имя геттера для переменной `name`.
- Если свойство типа `boolean`, префикс имени геттера может быть `get` или `is`. Например, `getPrinted()` или `isPrinted()` оба являются корректными именами для переменных типа `boolean`.
- Имя сеттера должно начинаться с префикса `set`. Например, `setName()` корректное имя для переменной `name`.
- Для создания имени геттера или сеттера, первая буква свойства должна быть изменена на большую и прибавлена к соответствующему префиксу (`set`, `get` или `is`).
- Сеттер должен быть `public`, возвращать `void` тип и иметь параметр соответствующий типу переменной.
- Геттер метод должен быть `public`, не иметь параметров метода, и возвращать значение соответствующее типу свойства.



Класс Object

Является суперклассом для **всех** классов (включая массивы)

Переменная этого типа может ссылаться на **любой** объект (но не на переменную примитивного типа)

Его методы наследуются **всеми** классами

Реализует базовые операции с объектами



Получение строкового представления объекта

`String toString()`

Получение ссылки на описание класса объекта

`final Class getClass()`

Клонирование объекта (получение копии)

`protected Object clone()`

Проверка равенства объектов

`boolean equals(Object obj)`

Получение хэш-кода объекта

`int hashCode()`

Метод завершения работы с объектом

`protected void finalize()`

Методы обслуживания блокировок в многопоточных приложениях

`void wait(...), void notify(), void notifyAll()`