



Lesson 4

29.03.2021

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

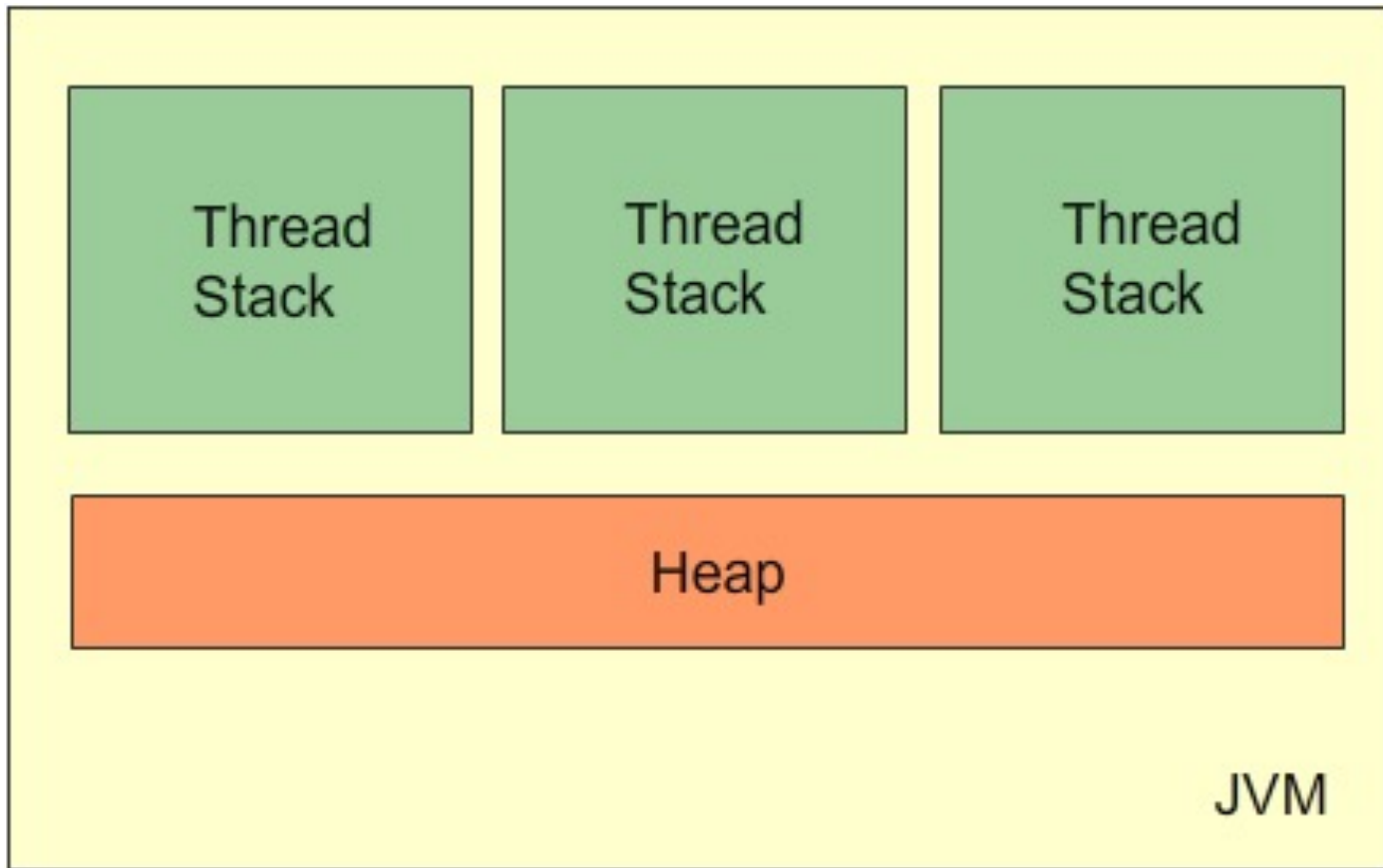
- Все поля данных инициализируются своими значениями, предусмотренными по умолчанию (0, false или null).
- Инициализаторы всех полей и блоки инициализации выполняются в порядке их перечисления в объявлении класса.
- Если в первой строке конструктора вызывается другой конструктор, то выполняется вызванный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

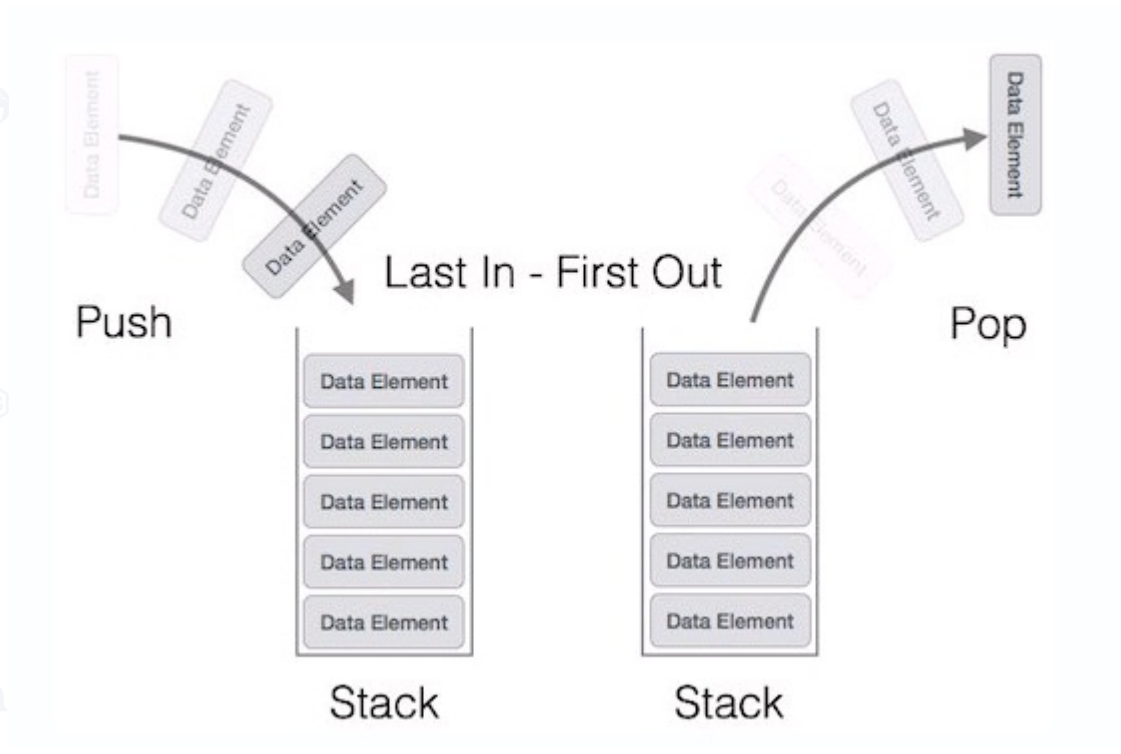
```
public class Konstr {  
    int width; // ширина коробки  
    ... int height; // высота коробки  
    ... int depth; // глубина коробки  
    ... // вычисляем объём коробки  
    ... int getVolume() {  
        ... return width * height * depth;  
        ... }  
    →  
    public static void main(String[] 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
    }
```

Java Memory Model







Стек работает по схеме LIFO (последним вошел, первым вышел). Всякий раз, когда вызывается новый метод, содержащий примитивные значения или ссылки на объекты, то на вершине стека под них выделяется блок памяти

Когда метод завершает выполнение, блок памяти, отведенный для его нужд, очищается, и пространство становится доступным для следующего метода

Основные особенности стека

Он заполняется и освобождается по мере вызова и завершения новых методов
Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы

Если память стека будет заполнена, Java бросит исключение

`java.lang.StackOverflowError`

Доступ к этой области памяти осуществляется быстрее, чем к куче

Является потокобезопасным, поскольку для каждого потока создается свой отдельный стек

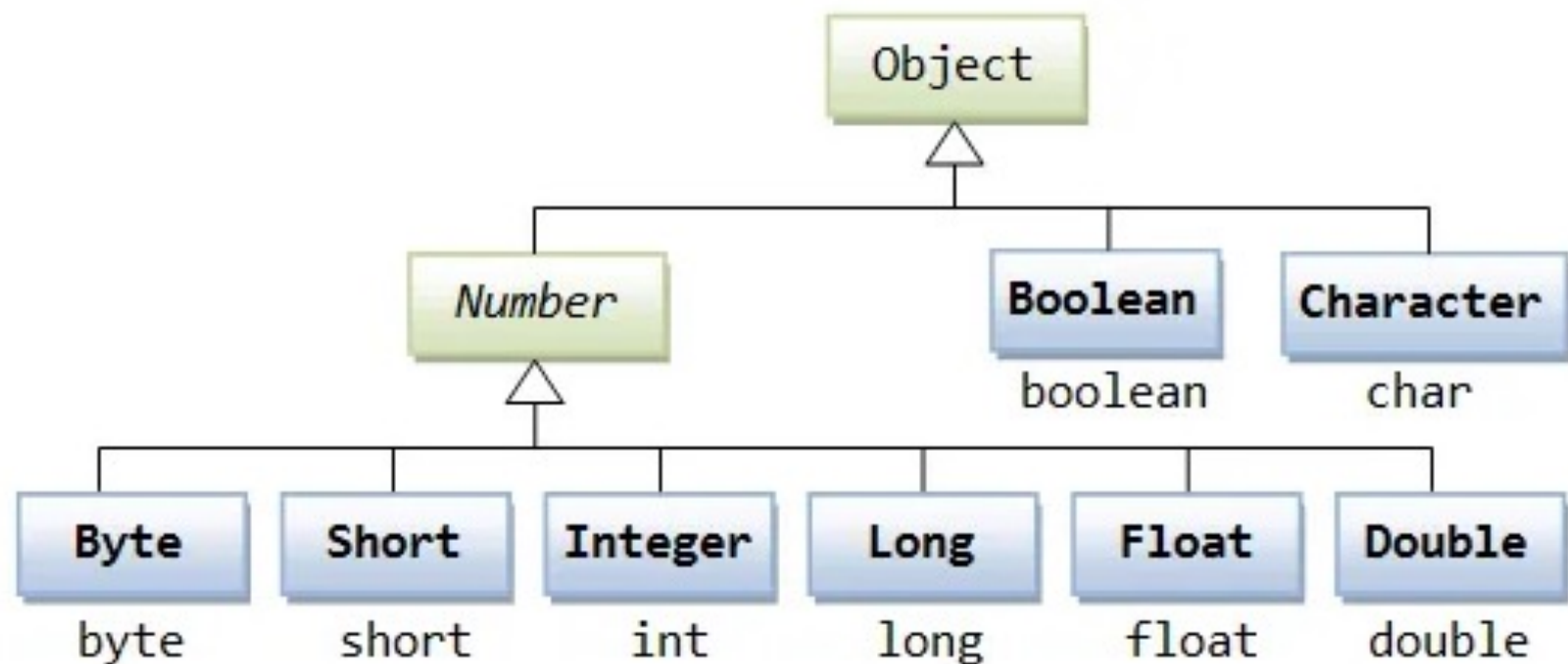
Тем не менее, Вы знаете, что Java - **объектно-ориентированный язык программирования**. Это значит, что в ней "все, что можно, представлено в виде объектов".

Поэтому, у примитивных типов есть **объекты-аналоги** - так называемые "**классы оболочки**", или "**wrapper**" (с англ. "обертка, упаковка"):

Примитивные типы

Классы оболочки

| | | |
|---------|---|-----------|
| byte | → | Byte |
| short | → | Short |
| int | → | Integer |
| long | → | Long |
| float | → | Float |
| double | → | Double |
| char | → | Character |
| boolean | → | Boolean |



boolean

byte

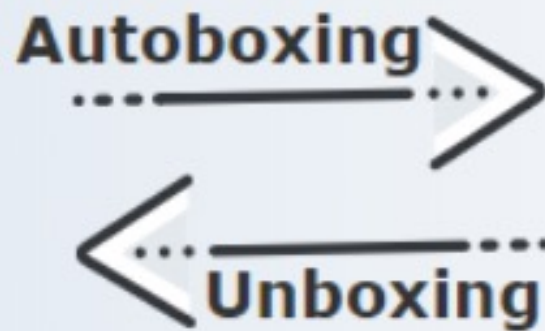
char

float

int

long

short



Boolean

Byte

Character

Float

Integer

Long

Short

Integer i = 127

Integer j = 127

127

Integer Constant pool

Integer i = new Integer(127)

Integer j = new Integer(127)

127

127

Integer i = 128

Integer j = 128

128

128

Integer i = 120

Integer j = new Integer(120)

120

120