Введение в программирование на Java

Лекция 4. Ссылочные типы данных. Строки. Массивы.

Виталий Олегович Афанасьев

02/03 декабря 2025

Массивы

Массивы (1)

Массив — структура данных, хранящая несколько значений одного типа.

Индекс — позиция элемента в массиве. Индексация во многих языках (в т.ч. в Java) начинается с 0.

Пример: массив из пяти имён.

Элемент	Элмо	Кермит	Гровер	Зелибоба	Коржик
Индекс	0	1	2	3	4

Массивы (2)

```
1 int[] a = new int[10];
2 // { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
3 a[0] = 10;
4 a[2] = 20;
5 a[9] = 500;
6  // { 10, 0, 20, 0, 0, 0, 0, 0, 0, 500 }
7
8 int[] a = new int[] { 2, 3, 5, 7, 11 };
9 int[] a = { 2, 3, 5, 7, 11 };
```

Массивы (3)

Размер массива нельзя изменить. Тем не менее, в переменную можно записать новый массив нужного размера.

```
1 int[] a = { 1, 2, 3, 4, 5 };
2 System.out.println(a.length); // 5
3
4 a = new int[] { 1, 2, 3 };
5 System.out.println(a.length); // 3
6
7 a = { 1, 2, 3 }; // ERROR
```

Массивы (4)

Вывод массива через `println` выведет не его элементы, а его тип и хэш-код (про него мы поговорим в будущих лекциях).

```
1  int[] a = { 1, 2, 3, 4, 5 };
2  System.out.println(a); // [I@5fdef03a
3
4  for (int i = 0; i < a.length; ++i)
5      System.out.println(a[i]);
6  // 1
7  // 2
8  // 3
9  // 4
10 // 5
11
12 import java.util.Arrays; // Поместить в начало программы
13 System.out.println(Arrays.toString(a)); // [1, 2, 3, 4, 5]
```

Массивы (5)

Размер массива может задаваться любым выражением (в т.ч. переменной).

```
1 Scanner scanner = new Scanner(System.in);
2 int size = scanner.nextInt();
3 int[] a = new int[size];
4 // a.length == size
5 for (int i = 0; i < a.length; ++i)
6     a[i] = i * i;
7 System.out.println(Arrays.toString(a)); // [0, 1, 4, 9, 16, ...]
```

Модель памяти Java

Стек и куча в JVM

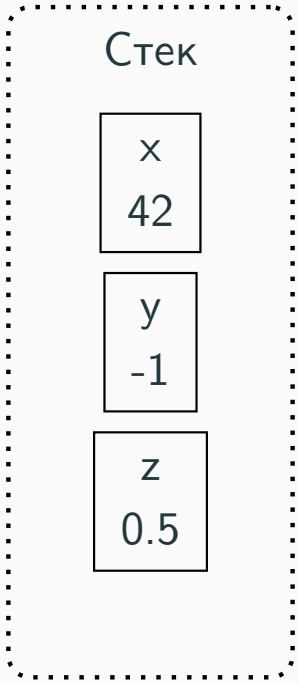
В JVM выделяются две основные области памяти программы:

- **Стек**
  - Набор стековых фреймов. Каждый фрейм хранит локальные переменные каждого метода (либо значения примитивных типов, либо ссылки на объекты составных типов)
  - Память освобождается автоматически при выходе из метода.
- **Куча**
  - Объекты составных типов и их элементы: массивы, экземпляры классов и их поля.
  - Память освобождается **сборщиком мусора** автоматически после того, как на объект перестанут ссылаться (но в произвольный момент времени!)

Замечание: не путайте стек и кучу в JVM со стеком и кучей, управлемыми ОС. Стек и куча JVM могут оказаться как в динамической, так и в статической области памяти программы.

Локальные переменные

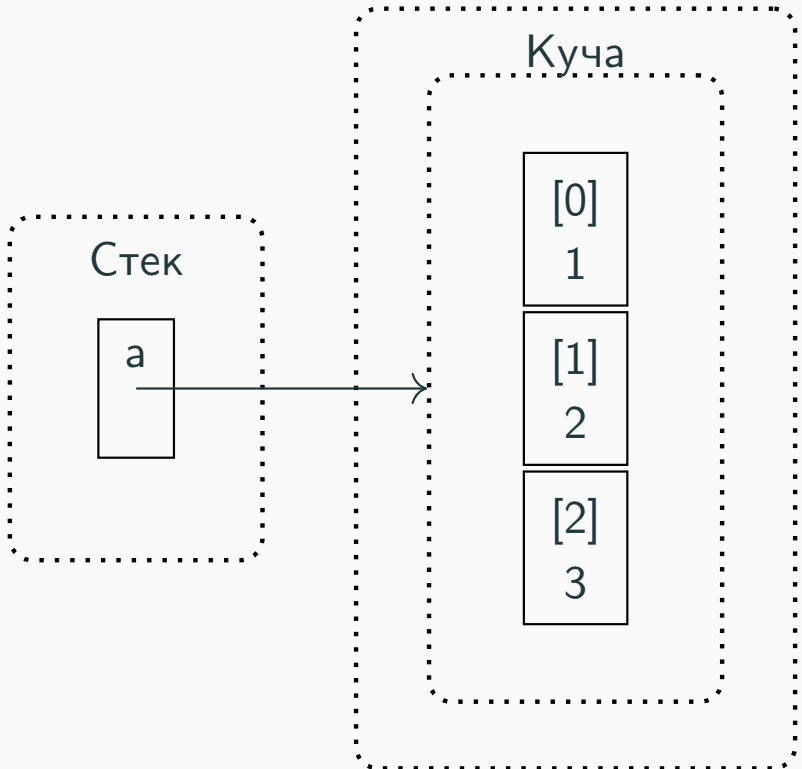
```
1 public static void main(String[] args) {
2     int x = 42;
3     byte y = -1;
4     double z = 0.5;
5 }
```



Массивы как ссылочные типы (1)

Массив — это ссылочный тип. На стеке хранится ссылка на массив, а значения массива — в куче.

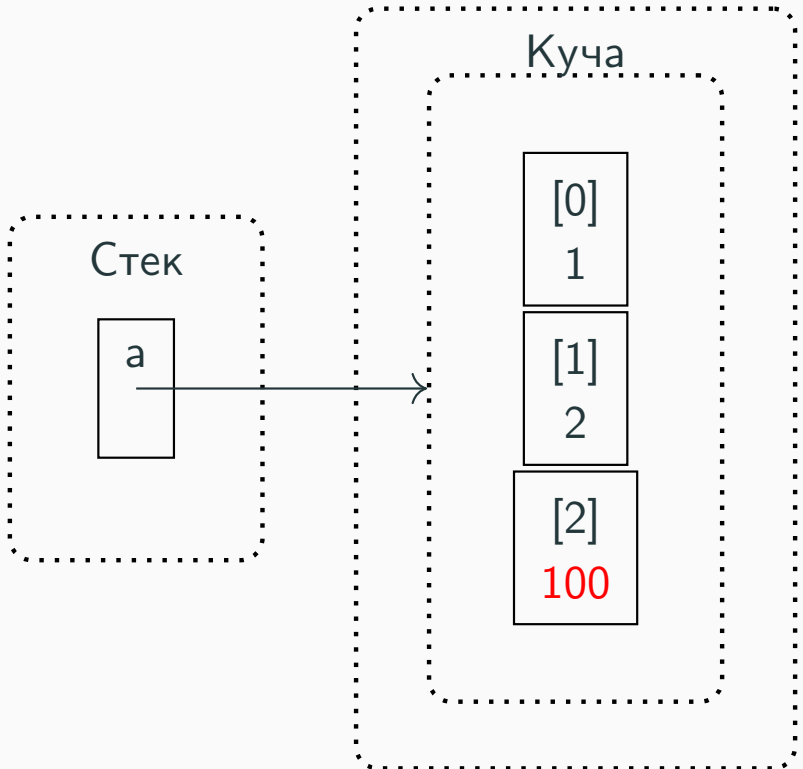
```
1 public static void main(String[] args) {
2     int[] a = { 1, 2, 3 };
3
4 }
```



Массивы как ссылочные типы (1)

Массив — это ссылочный тип. На стеке хранится ссылка на массив, а значения массива — в куче.

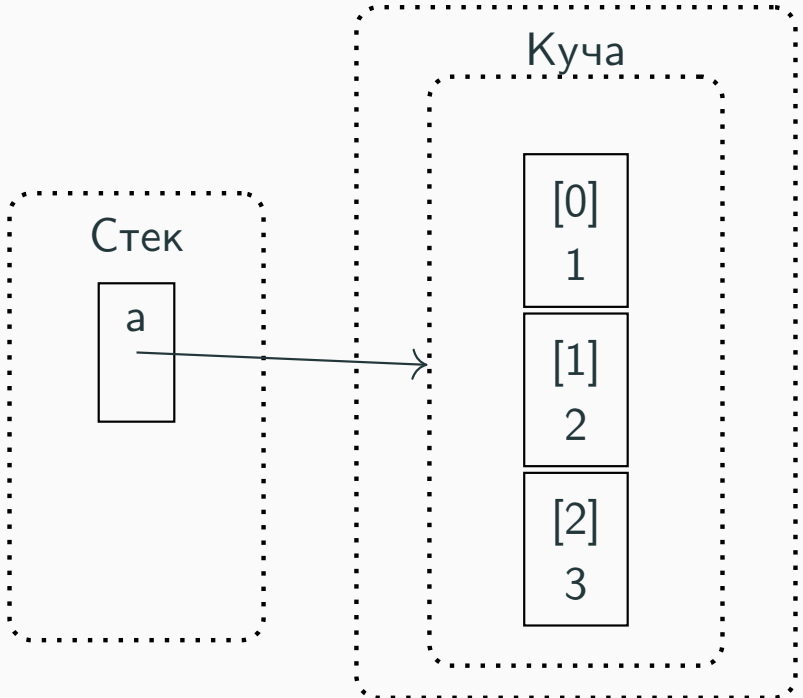
```
1 public static void main(String[] args) {
2     int[] a = { 1, 2, 3 };
3     a[2] = 100;
4 }
```



Массивы как ссылочные типы (2)

Присвоение одного массива в другую переменную не копирует его, а создаёт ссылку на существующий массив.

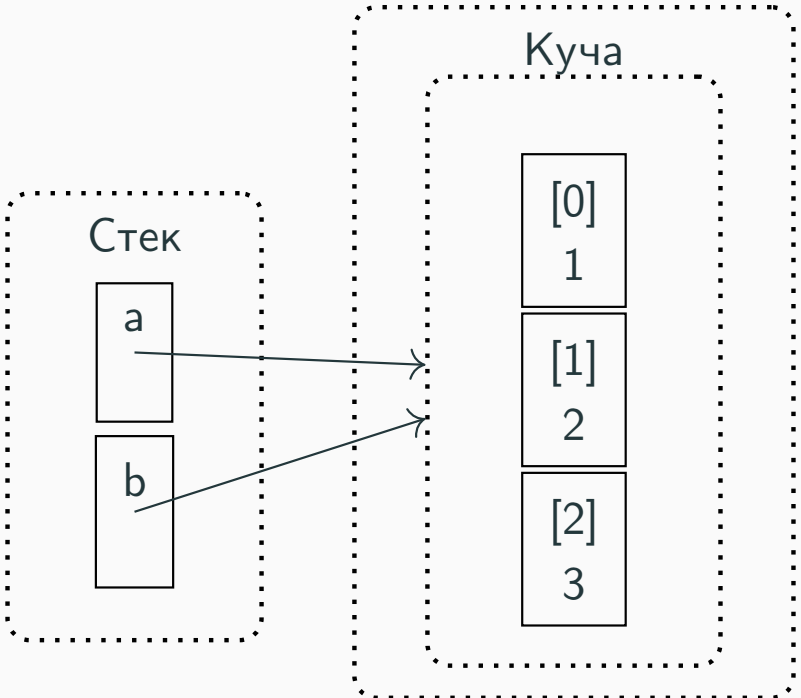
```
1 public static void main(String[] args) {
2     int[] a = { 1, 2, 3 };
3
4
5 }
```



Массивы как ссылочные типы (2)

Присвоение одного массива в другую переменную не копирует его, а создаёт ссылку на существующий массив.

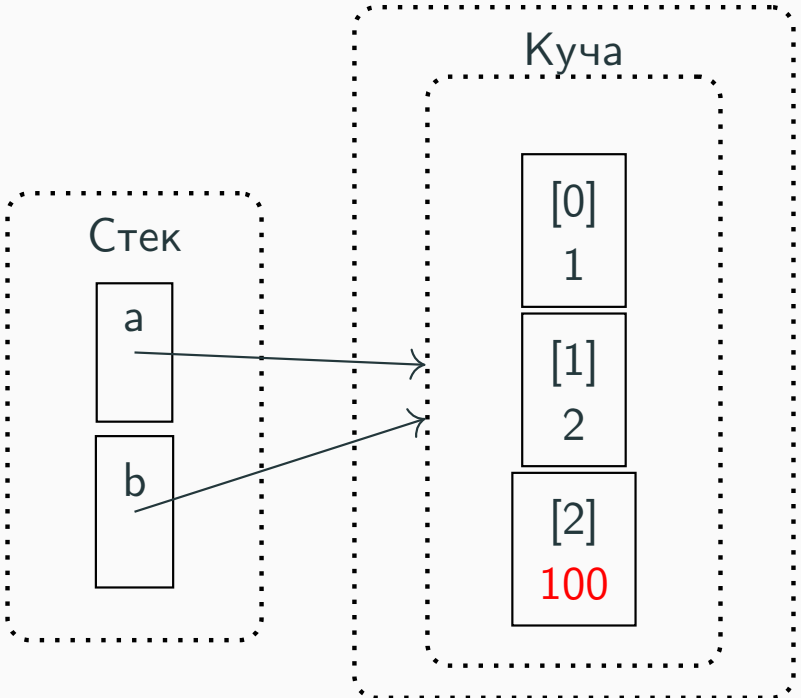
```
1 public static void main(String[] args) {
2     int[] a = { 1, 2, 3 };
3     int[] b = a; // { 1, 2, 3 }
4
5 }
```



Массивы как ссылочные типы (2)

Присвоение одного массива в другую переменную не копирует его, а создаёт ссылку на существующий массив.

```
1 public static void main(String[] args) {
2     int[] a = { 1, 2, 3 };
3     int[] b = a; // { 1, 2, 3 }
4     b[2] = 100; // a: { 1, 2, 100 }
5 }
```



Строковый тип

Строки в Java (1)

Строки в Java представлены библиотечным типом `String`.

```
1 String str1 = "Hello, World!";
2 String str2 = "Привет, Мир!";
3 String str3 = "👋, 🌍!";
```

Строки в Java (2)

У строк определено большое количество методов.

Полный список можно найти в документации.

```java
String hello = "Hello";
String world = "World";
String helloWorld = hello + ", " + world + "!";
// "Hello, World!"

int length = helloWorld.length(); // 13

char firstChar = helloWorld.charAt(0); // 'H'

int indexOf     = helloWorld.indexOf('l');     // 2
int lastIndexOf = helloWorld.lastIndexOf('l'); // 10
```

Строки в Java (3)

Сами строки в Java **неизменяемые**.

Но есть набор методов, позволяющих получить изменённую копию.

```
1  String helloWorld = "Hello, World!";
2
3  String substr = helloWorld.substring(0, 2); // "He"
4
5  String repeat = substr.repeat(3); // "HeHeHe"
6
7  String replaceS = helloWorld.replace("World", "it's me");
8  // "Hello, it's me!"
9  String replaceC = helloWorld.replace('l', ' ');
10 // "He  o, Wor d!"
11
12 String strip = "   something   ".strip(); // "something"
13
14 String toLowerCase = helloWorld.toLowerCase(); // "hello, world!"
15 String toUpperCase = helloWorld.toUpperCase(); // "HELLO, WORLD!"
```

Строки в Java (4)

Существуют т.н. **escape-последовательности** — символы, начинающиеся с обратного слеша, имеющие специальное значение.

```
1 char lineFeed = '\n';
2 char carriageReturn = '\r';
3 char tabulation = '\t';
4 char singleQuote = '\'';
5 char doubleQuote = '\"';
6 char backslash = '\\';
```

Unicode

Unicode (1)

ASCII — одна из распространённых кодировок текста.

Проблема в том, что ASCII кодирует лишь 128 символов (латинские буквы, цифры, знаки пунктуации и спец. символы).

Только около 20% населения Земли владеют английским языком.

Что же делать с символами остальных языков?

Unicode (2)

Unicode — стандарт кодирования, цель которого — унифицировать представление текста в компьютерных системах, включив в себя знаки (почти) всех письменных языков мира.

В Unicode зарезервировано ≈ 1.1 млн. символов.

В настоящий момент представлено ≈ 155 тыс.

Рекомендую к ознакомлению:

- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
- The Absolute Minimum Every Software Developer Must Know About Unicode in 2023 (Still No Excuses!)

Unicode (3)

Unicode имеет три основных способа представления:

- UTF-8 (каждый символ* — от 1 до 4 байт)
- UTF-16 (каждый символ* — 2 или 4 байта)
- UTF-32 (каждый символ* — 4 байта)

В Java исторически для представления строк используется UTF-16.

Важно: это не значит, что Java не совместима с UTF-8 или UTF-32.

Unicode (4)

Java использует UTF-16 для строк: каждый символ* — 2 или 4 байта.

Но тип `char` имеет размер в 2 байта.

Где подвох?

Unicode (5)

* Понятие "символ" довольно размыто.

Например, иероглиф 明 состоит из 日 и 月.

Является ли он одним "символом"?

Unicode оперирует следующими терминами:

- **Code unit** — единица кодирования. 1 байт для UTF-8, 2 байта для UTF-16, 4 байта для UTF-32. Значение типа `char` — это ровно один code unit.
- **Code point** — идентификационный номер элемента в Unicode. Например, латинская буква `A` имеет номер 65, а кириллическая буква `Ы` — номер 1067. Code point *обычно* соответствует печатному символу. Для UTF-16 один code point состоит либо из одного, либо из двух code unit.
- **Grapheme cluster** — последовательность code point'ов, которые при печати должны отображаться как единое целое.

Unicode (6)

Примеры (в UTF-16):

- Латинская буква `A`. Один code point, один code unit. Можно сохранить в один `char`.
- Кириллическая буква `Ы`. Один code point, один code unit. Можно сохранить в один `char`.
- Эмодзи 😎. Один code point, **два** code unit. **Нельзя** сохранить в один `char`. Можно сохранить в один `int`.
- Буква é. Один Grapheme Cluster. **Два** code point, **два** code unit. **Нельзя** сохранить в один `char`. Можно (но обычно **не нужно**) сохранить в один `int`.
- Эмодзи 🚴🏻‍♂️. **Пять** code point, **семь** code unit. Один Grapheme Cluster. **Нельзя** сохранить в один `char`. **Нельзя** сохранить в один `int`.

Unicode (8)

Метод `length` у строки возвращает количество code **unit**, а не визуальное количество символов.

```
1 "A".length(); // 1
2 "Ы".length(); // 1
3 "😎".length(); // 2
4 "é".length(); // 2
5 "🚵".length(); // 7
```

Unicode (8)

Метод `charAt` возвращает один code **unit**. Если вы не уверены, что все все символы строки имеют размер в один code unit, то использовать этот метод **опасно**.

```
1 char c0 = "é".charAt(0);
2 char c1 = "é".charAt(1);
3 System.out.println(c0); // e
4 System.out.println(c1); // ´
5 c0 = "😎".charAt(0);
6 c1 = "😎".charAt(1);
7 System.out.println(c0); // ? (некорректный символ)
8 System.out.println(c1); // ? (некорректный символ)
```

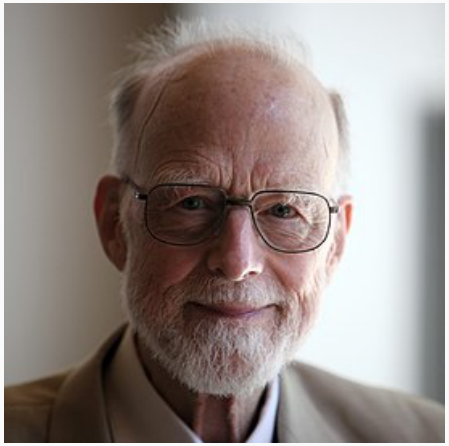
Значение по умолчанию

Значение по умолчанию

```java
int[] ints = new int[5];
// { 0, 0, 0, 0, 0 }
double[] doubles = new double[5];
// { 0.0, 0.0, 0.0, 0.0, 0.0 }
boolean[] bools = new boolean[5];
// { false, false, false, false, false }

int[][] arr2d = new int[5][];
// { null, null, null, null, null }

String[] strs = new String[5];
// { null, null, null, null, null }

String str = null;
str.substring(0, 2); // Ошибка ВЫПОЛНЕНИЯ: NullPointerException

int[] a = null;
int len = a.length; // Ошибка ВЫПОЛНЕНИЯ: NullPointerException
```

The billion-dollar mistake

*I call it my billion-dollar mistake. It was the invention of the null reference in **1965**. At that time, I was designing the first comprehensive type system for references in an object oriented language (**ALGOL W**). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But **I couldn't resist the temptation** to put in a null reference, simply because **it was so easy to implement**. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a **billion dollars of pain and damage in the last forty years**.*



Сэр Энтони Хоар
QCon, 2009