

Лекция 6

Массивы

«Множественность» реального мира

- До сих пор мы рассматривали только переменные различных типов данных (`int`, `double`, `char`, `String`, `boolean`), в которых хранились единственные значения (одной значение в одной переменной – целое число, вещественное число, символ, строка, логическое значение, соответственно)
- Но в реальном мире многие вещи состоят из множества однотипных объектов:
 - В каждой группе несколько студентов
 - На каждом курсе несколько групп
 - В неделе 7 дней
 - В рабочие дни в вашем расписании несколько предметов
 - Лес состоит из множества деревьев
 - и т.д. и т.п.

Связь с программированием

- Используя простые переменные (с единственным значением), решать задачи, связанные с обработкой множества объектов (причем число таких объектов заранее неизвестно, может меняться и т.п.), невозможно
- Поэтому во всех языках программирования есть специальные структуры данных, позволяющие в одной переменной (точнее в одном объекте) хранить множество неких других объектов

Массивы

- Простейшей структурой данных для хранения нескольких значений одновременно являются ***массивы***
- Ниже на показан массив, состоящий из 5-ти целых чисел

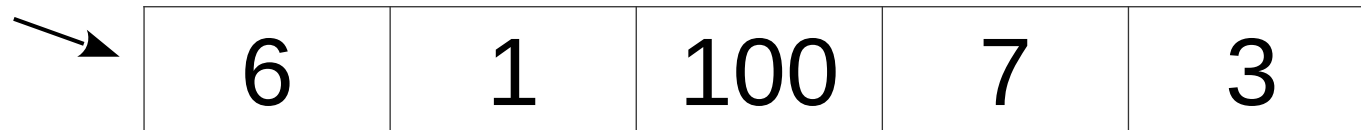
`int[] arr`



6	1	100	7	3
---	---	-----	---	---

Массивы

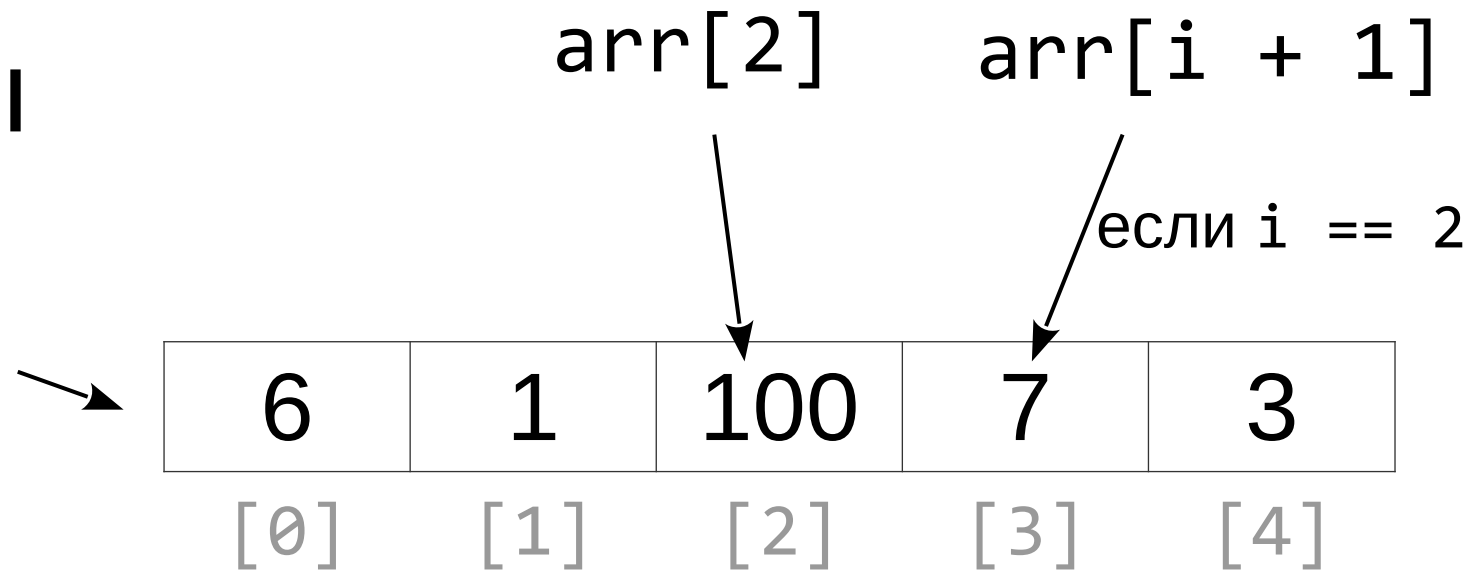
```
int[] arr
```



- **Массив** – упорядоченный набор элементов **одного типа данных**
- Если обычные переменные мы представляли как именованные клеточки в тетради или на доске, в которые можно вписывать конкретные значения, то массивы – это именованный набор из нескольких клеточек, каждая из которых может содержать свое значение (которой можно как прочитать, так и изменить)

Массивы

`int[] arr`



- **Массив** – упорядоченный набор элементов **одного типа данных**
 - Конкретный элемент массива определяется его индексом (номером)
 - Индексация (нумерация) начинается с 0, идет непрерывно (0, 1, 2 и т.д.) до (размер_массива – 1)
 - Индекс элемента задается в квадратных скобках после имени переменной-массива, например: `arr[0]`, `arr[2]`, `arr[i + 1]`
 - Конкретный массив имеет фиксированную длину, которая определяется при его создании

Объявление и создание массивов

```
int[] arr;  
arr[0] = 6;    // ошибка компиляции  
               // (переменной не присвоено значение)
```

- Чтобы работать с переменной-массивом, такой переменной необходимо присвоить какое-либо значение

```
int[] arr = null;  
arr[0] = 6;    // ошибка выполнения  
               // (arr не указывает на массив)
```

- Чтобы работать с массивом его нужно создать (или присвоить переменной-массиву уже созданный массив)

```
int[] arr = new int[5];  
arr[0] = 6;    // все хорошо
```

← создание массива

Варианты создания массива

```
// все элементы массива принимают
// значение по умолчанию (для чисел – 0)
int[] arr1 = new int[5];

// значения всех элементов задаются
int[] arr3 = new int[] { 6, 1, 100, 7, 3 };

// более короткая форма записи,
// только при объявлении массива
// ("синтаксический сахар")
double[] arr4 = { -1, 0, 1.5, 2.65 };
```


Пример: вывод массива, заполненного случайными числами

```
int[] arr = new int[10];
```

```
Random rnd = new Random();
```

```
for (int i = 0; i < arr.length; i++) {  
    arr[i] = rnd.nextInt(100); // от 0 до 99  
}
```

Проход по всем элементам массива

```
for (int i = 0; i < arr.length; i++) {  
    System.out.print((i > 0 ? ", " : "") + arr[i]);  
}  
System.out.println();
```

Еще одна версия цикла **for**

- В ситуациях, когда нам необходимо последовательно перебрать все элементы массива и при этом информация об индексе каждого элемента не важна, можно воспользоваться следующим синтаксисом (на примере суммирования всех элементов массива):

```
double[] arr = { 3, 4.6, 0.01 };
```

```
double sum = 0;
```

```
for (double item : arr) {  
    sum += item;  
}
```

Тип элементов массива



Тело цикла выполнится 3 раза (по кол-ву элементов в массиве); переменная цикла **item** последовательно примет значения: 3, 4.6, 0.01

ArrayUtils

- Многие операции с массивами в ваших практических задачах будут повторяться, поэтому в процессе лекции создадим (рассмотрим) класс **ArrayUtils**, содержащий базовые примитивы для работы с массивами, который демонстрирует принципы работы с массивами и может быть повторно использован
 - Фактически **ArrayUtils** – ваша первая повторно используемая библиотека
 - Будет использоваться во всех задачах на массивы
 - При необходимости вы можете и должны добавлять в этот класс другие примитивы работы с массивами, которые вам могут встретиться в ваших задачах
(именно примитивы, т.е. такие действия, которые могут встретиться и в других задачах)

Ввод массивов

- Массивы могут вводиться не только с консоли, но из файлов и т. д.
- Прочитать строку из Scanner (или из файла) очень просто, поэтому нам нужно реализовать методы конвертации строки в массивы (хотя бы для `int` и `double`):

```
public static int[ ] strToIntArray(String str)
```

```
public static double[ ] strToDoubleArray(String str)
```

- Как было сказано, все функции работы с массивами соберем в отдельный класс `ArrayUtils` (данный класс вы будете повторно использовать в своих проектах)

Ввод массивов

- Несколько следующих слайдов – забежание вперед с целью объяснить код методов, которые будут реализованы для ввода массивов
 - Желательно, конечно, понять, но можно пока, как было уже сказано, просто скопировать из проекта Lect6Samples класс ArrayUtils в свой проект (не забыв при необходимости изменить имя пакета) и использовать
- Конечно, часть функциональности ArrayUtils уже реализована в стандартной библиотеке Java (и уж тем более в сторонних библиотеках), но в качестве обучения полезно реализовать данную функциональность самостоятельно

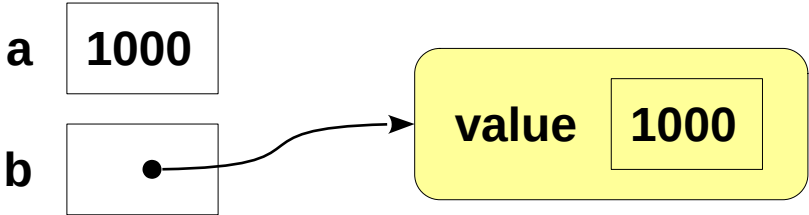
Ввод массивов

- Для считывания чисел, записанных в строке, также будем использовать Scanner (да-да, Scanner может читать данные не только с консоли, но из строки, а также из файлов и т.п.)
- Проблема в том, что при чтении чисел с помощью Scanner'a мы не знаем, сколько чисел записано в строке, пока все эти числа не прочитаем
- **А размер уже созданного массива изменить нельзя!** (можно создать новый объект массива и в него скопировать данные из старого массива, но это неэффективно)

Ввод массивов

- Поэтому следует воспользоваться списком – новым типом (классом) из стандартной библиотеки Java (Java Class Library – JCL)
 - Список – структура данных (коллекция), которая во-многом аналогична массиву, но количество элементов в которой может меняться во время выполнения программы (часто в программах используются именно списки, а не массивы)
- Проблема в том, что список и другие обобщенные классы (такие, которые можно применять с различными типами данных) не умеют работать с примитивными типами данных (так получилось исторически с целью сохранения обратной совместимости байт-кода Java при появления дженериков в Java 5)
- Для обхода данной проблемы предусмотрены классы-обертки для примитивных типов данных, которые можно использовать в списках

Классы-обертки для примитивных типов данных

- **Integer** для **int**
 - **Double** для **double**
 - **Character** для **char**
 - **Boolean** для **boolean**
 - и т. д.
- Предусмотрены, когда необходимо, чтобы числа и другие примитивные типы данных можно было представить в виде объекта
- ```
int a = 1000;
Integer b = a;
```
- 
- The diagram shows two variables, 'a' and 'b'. Variable 'a' is represented by a box containing the value '1000'. Variable 'b' is represented by a box containing a dot, which has an arrow pointing to a larger yellow rounded rectangle representing an Integer object. Inside this object, there is a smaller box labeled 'value' containing the number '1000'.
- В большинстве случаев синтаксис работы с классами-обертками совпадает с примитивными типами данных, но меньшая производительность и есть свои особенности, которые мы более подробно рассмотрим, когда будем изучать коллекции (**одна из особенностей, сравнивать на равенство == нельзя, надо использовать метод equals**)



# *toIntArray, toDoubleArray*

```
public static int[] toIntArray(String str) {
 Scanner scanner = new Scanner(str);
 scanner.useLocale(Locale.ROOT);
 scanner.useDelimiter("(\\s|[,;])+");
 List<Integer> list = new ArrayList<>();
 while (scanner.hasNext()) {
 list.add(scanner.nextInt());
 }

 // из List<Integer> можно получить Integer[]
 Integer[] arr = list.toArray(new Integer[0]);
 // Integer[] -> int[]
 return toPrimitive(arr);
}
```

Для double и других типов придется реализовать аналогичные методы

# *toPrimitive*

```
public static int[] toPrimitive(Integer[] arr) {
 if (arr == null) {
 return null;
 }
 int[] result = new int[arr.length];
 for (int i = 0; i < arr.length; i++) {
 // автоматическая распаковка из объекта
 result[i] = arr[i];
 }
 return result;
}
```

Для double/Double и других типов придется реализовать аналогичные методы

Чтобы не дублировать код, в Java существует возможность писать код, применимый к различным типам данных, — механизм **generics**, однако он не применим к примитивным типам данных, а только к классам

# *toObject*

```
public static Integer[] toObject(int[] arr) {
 if (arr == null) {
 return null;
 }
 Integer[] result = new Integer[arr.length];
 for (int i = 0; i < arr.length; i++) {
 // автоматическая упаковка в объект
 result[i] = arr[i];
 }
 return result;
}
```

Для double/Double и других типов придется реализовать аналогичные методы

# Забегаая вперед (пока можно не запоминать)

- Многие полезные методы для работы с массивами, и не только с массивами, содержатся в сторонней библиотеки **Commons Lang** проекта **Apache Commons** (в частности методы `toPrimitive`, `toObject` есть в одноименном классе `ArrayUtils` этой библиотеки)
- Библиотека **Commons Lang** и другие библиотеки проекта **Apache Commons** — одни из наиболее часто используемых сторонних библиотек в мире Java (на них обязательно стоит обратить внимание, хотя поверхностно ознакомиться, что эти библиотеки умеют, чтобы знать, где искать, когда вам похожая функциональность понадобится)
- Подключить стороннюю библиотеку к **Maven**-проекту в среде NetBeans проще простого (в дереве проекта в узле «Зависимости» в контекстном меню выбрать пункт «Добавить зависимость»)

# Алтернативная «магия»

- Начиная с версии Java 8 можно конвертировать строку в массив примитивных типов одной строкой кода следующим образом (но лучше пока таких трюков избегать, т.к. для вас они на данном этапе полностью непонятны):

```
public static int[] toIntArray(String str) {
 return Arrays.stream(str.split("(\\s|[,;])+"))
 .mapToInt(Integer::parseInt).toArray();
}
```

- Как было сказано, существует способы частично обобщить код (использовать generics) для ввода массивов разных типов, но сейчас рассматривать эти методы не имеет смысла

# Собственно ввод массива с консоли

```
public static int[] readIntArrayFromConsole(String arrName) {
 Scanner scanner = new Scanner(System.in);
 while (true) {
 try {
 if (arrName == null || arrName.length() == 0) {
 arrName = "";
 } else {
 arrName = " " + arrName;
 }
 System.out.printf("Введите массив%s:%n", arrName);
 String line = scanner.nextLine();
 return toIntArray(line);
 }
 catch (Exception e) {
 System.out.print("Вы ошиблись, попробуйте еще раз! ");
 }
 }
}

public static int[] readIntArrayFromConsole() {
 return readIntArrayFromConsole(null);
}
```

# Перевод в строку – *toString*

```
private static String toString(int[] arr, String itemFormat) {
 if (arr == null) {
 return null;
 }

 if (itemFormat == null || itemFormat.length() == 0) {
 itemFormat = "%s";
 }
 StringBuilder sb = new StringBuilder();
 for (int i = 0; i < arr.length; i++) {
 if (i > 0) {
 sb.append(", ");
 }
 sb.append(String.format(Locale.ROOT, itemFormat, arr[i]));
 }
 return sb.toString();
}

private static String toString(int[] arr) {
 return toString(arr, null);
}
```

Для double и других типов придется реализовать аналогичные методы

# Альтернативная реализация *toString*

- С использованием класса `java.util.Arrays`

```
private static String toString(int[] arr) {
 if (arr == null) {
 return null;
 }

 String str = Arrays.toString(arr); // возвр. "[a1, ...]"
 return str.substring(1, str.length() - 1);
}
```

- В данной реализации нельзя указать шаблон форматирования для элементов массива



# Ввод и вывод массива – конечный вариант с использованием разработанного нами ArrayUtils

*// ввод массива*


```
int[] arr = ArrayUtils.readIntArrayFromConsole();
```

*// обработка массива*

```
for (int i = 0; i < arr.length; i++) {
 arr[i]++;
}
```

*// вывод массива*

```
System.out.printf("\033[31mОбработанный\033[0m массив:%n%s%n",
 ArrayUtils.toString(arr, "\033[31m%3d\033[0m"));
```



Если формат не указать, будет форматирование по умолчанию

# Передача массивов в качестве параметров методам

```
public static void arrayParamDemo(int value, int[] arr) {
 value++;
 // увеличим последний элемент
 arr[arr.length - 1]++;
 // создадим новый массив
 arr = new int[] { 1, 2, 3 };
}
```

В методах можно менять содержимое элементов массива, внесенные изменения будут видны после выполнения метода

```
int a = 7;
int[] b = { 1, 10, 100 };
arrayParamDemo(a, b);
System.out.printf("a = %d\n", a); // a = 7
System.out.printf("b = { %s }\n",
 ArrayUtils.toString(b)); // b = { 1, 10, 101 }
```

Но заменить массив на другой массив нельзя (все параметры передаются по значению, т. е. в виде копии ссылки на объект)

# Примеры

# Пример.

## Поиск индекса элемента в массиве

// для массива целых чисел

```
public static int indexOf(int[] arr, int value) {
 for (int i = 0; i < arr.length; i++) {
 if (arr[i] == value) {
 return i;
 }
 }
 return -1;
}
```

// обобщенный вариант

```
public static <T> int indexOf(T[] arr, T value) {
 for (int i = 0; i < arr.length; i++) {
 if (arr[i].equals(value)) {
 return i;
 }
 }
 return -1;
}
```

# Пример.

## Поиск максимума в массиве

```
public static int indexOfMax(int[] arr) {
 if (arr == null || arr.length == 0)
 return -1;

 int indexMax = 0; // индекс максимального элемента
 // (за максимум вначале принимаем
 // первый [0] элемент массива)
 for (int i = 1; i < arr.length; i++) {
 if (arr[i] > arr[indexMax]) {
 // если нашли элемент
 // больше запомненного максимума
 indexMax = i;
 }
 }

 return indexMax;
}
```

Обобщенный метод также можно написать, если наложить ограничение `<T extends Comparable<T>>` (реализовано в `Lect6Samples`)

Можете пока не забивать себе голову

# Пример. Сортировка массива методом пузырька: суть

Суть метода заключается в последовательном сравнении соседних элементов и их последующем обмене, если левый больше правого. Таким образом направо "выдавливается" самый большой элемент

Ниже сортируется массив { 7, 9, 5, 1, 3 }

- красный цвет — элементы, которые сравниваются и при необходимости обмениваются
- синий цвет — результат после последовательности действий
- желтый цвет — элементы массива, которые отсортированы (стоят на своих местах)

|               |               |               |               |
|---------------|---------------|---------------|---------------|
| 1)            | 2)            | 3)            | 4)            |
| 7, 9, 5, 1, 3 | 7, 5, 1, 3, 9 | 5, 1, 3, 7, 9 | 1, 3, 5, 7, 9 |
| 7, 9, 5, 1, 3 | 7, 5, 1, 3, 9 | 5, 1, 3, 7, 9 | 1, 3, 5, 7, 9 |
| 7, 9, 5, 1, 3 | 5, 7, 1, 3, 9 | 1, 5, 3, 7, 9 | 1, 3, 5, 7, 9 |
| 7, 5, 9, 1, 3 | 5, 1, 7, 3, 9 | 1, 3, 5, 7, 9 |               |
| 7, 5, 1, 9, 3 | 5, 1, 3, 7, 9 |               |               |
| 7, 5, 1, 3, 9 |               |               |               |

После первого прохода, очевидно, что на последней позиции будет нужный элемент; кол-во таких проходов `arr.length - 1`

# Пример. Сортировка массива методом пузырька: реализация

```
/**
 * Сортировка методом пузырька
 */
public static void bubbleSort(int[] arr) {
 for (int i = arr.length - 1; i >= 0; i--) {
 for (int j = 0; j < i; j++) {
 if (arr[j] > arr[j + 1]) { // соседи
 // обмен
 int temp = arr[j];
 arr[j] = arr[j + 1];
 arr[j + 1] = temp;
 }
 }
 }
}
```

Обобщенный метод также можно написать, если наложить ограничение `<T extends Comparable<T>>` (реализовано в `Lect6Samples`)

Можете пока не забивать себе голову

# Сортировка с помощью стандартной библиотеки языка Java (`java.util.Arrays`)

- Самостоятельно реализованная сортировка (особенно пузырьковая) при программировании на Java имеет смысл только в целях обучения, естественно, в стандартной библиотеке Java есть реализованная сортировка

```
int[] arr = ArrayUtils.createRandomIntArray(10, -50, 51);
System.out.println("Before: " + ArrayUtils.toString(arr));
Arrays.sort(arr);
System.out.println("After: " + ArrayUtils.toString(arr));
```



# Сортировка по какому-то особому критерию

- Например, по абсолютному значению:

```
// если мы необходимо отсортировать
// по какому-то другому критерию,
// это возможно только для массивов объектов
// (в нашем случае для Integer[])
```

```
Integer[] objArr = ArrayUtils.toObject(arr);
Arrays.sort(objArr, (a, b) -> Math.abs(a) - Math.abs(b));
arr = ArrayUtils.toPrimitive(objArr);
System.out.println("After2: " + ArrayUtils.toString(arr));
```

After2: -6, 16, 16, -27, 30, 32, 42, -45, -46, -50

# Стандартная функциональность Arrays

- Очевидно, что функции, которые рассматривались выше, часто требуются для различных задач, поэтому они уже реализованы в стандартной библиотеке
- Класс `java.util.Arrays`, которым является любой массив во время работы программы, содержит несколько десятков различных свойств и функций / методов (некоторые из них будут рассмотрены ниже):

<https://msdn.microsoft.com/ru-ru/library/system.array.aspx>

- Также к классу `Array` добавляется ряд методов-расширений при подключении `Linq`:

<https://msdn.microsoft.com/ru-ru/library/system.linq.enumerable.aspx>

# Другие методы Arrays

- Java 8:  
<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>
- Java 10:  
<https://docs.oracle.com/javase/10/docs/api/java/util/Arrays.html>
  - *binarySearch*
  - *copyOf*
  - *copyOfRange*
  - *equals / deepEquals*
  - *fill*
  - *hashCode*
  - *sort / parallelSort*
  - *stream*
  - *toString*

# ArrayUtils из Apache Commons Lang

- <https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/ArrayUtils.html>
  - Изучите самостоятельно

# Совет (настоятельная рекомендация)

- Обязательно смотрите проекты-примеры к лекции (и экспериментируйте с ними):
  - Array1Samples