PRODUCT
STAR.RU

Работа
с массивами

Содержание урока

- Что такое массивы
- Объявление, инициализация, чтение и запись данных
- Многомерные массивы
- Методы для работы с массивами
- Исключения, связанные с массивами
- Java Collections и массивы

Александр Слесаренко

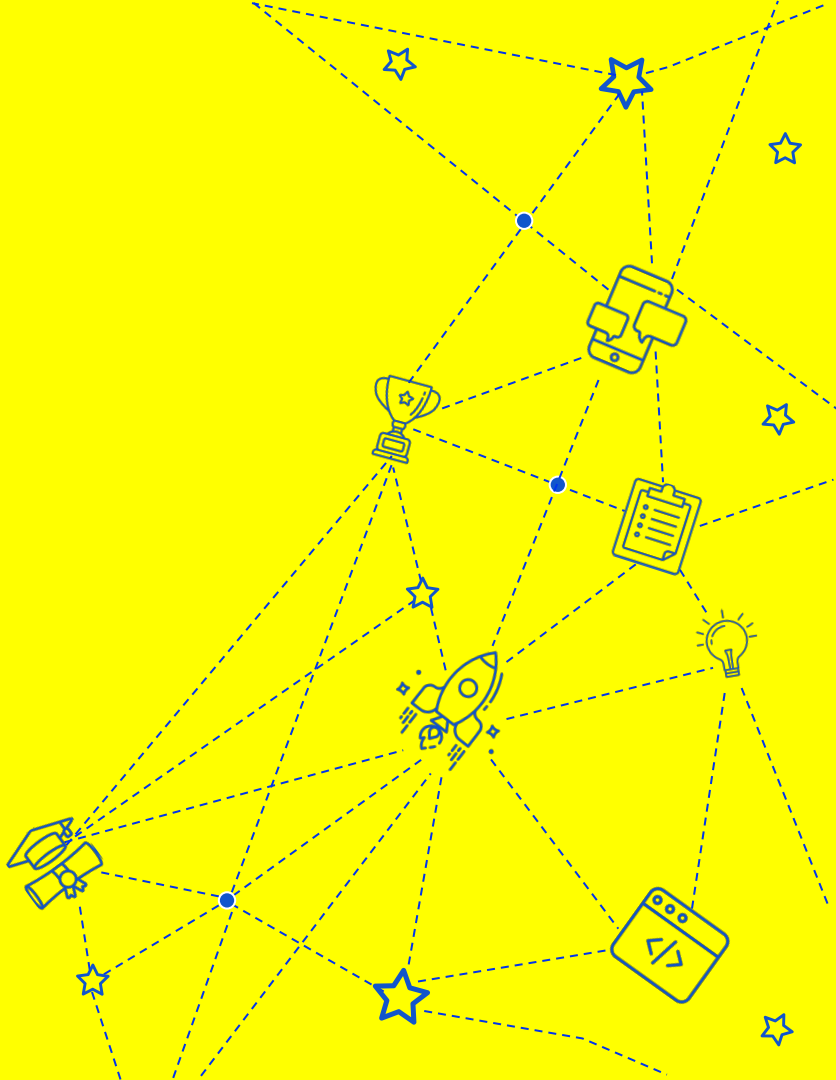
Core Developer, Ergoplatform.org

ex-Expert at Huawei Research, Moscow

ex-Principal Engineer at Huawei Research, Moscow

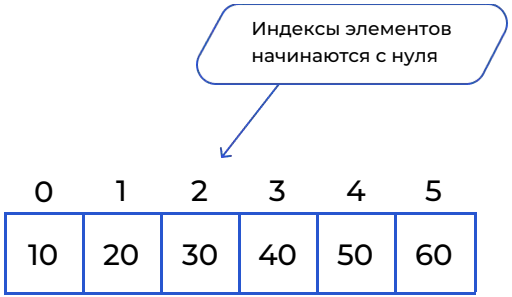
- Разработал и запустил блокчейн Ergo
- Разработал язык ErgoScript
- Разработал стандарт Ergo Pay
- Патенты и публикации

Что такое массивы



Что такое массив?

- Блок памяти с элементами **одного** типа

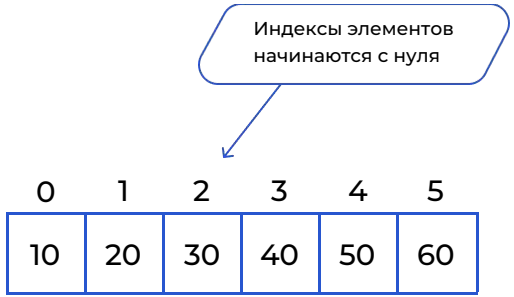


byte	- 1 байт
short	- 2 байта
char	- 2 байта
int	- 4 байта
float	- 4 байта
long	- 8 байтов
double	- 8 байтов

элемент ссылочного типа -
4 байта - на 32 битном процессоре,
4 байта - на 64 битном в режиме сжатия (когда heap size < 32Gb)
8 байта - на 64 битном (куча >= 32Gb)

Что такое массив?

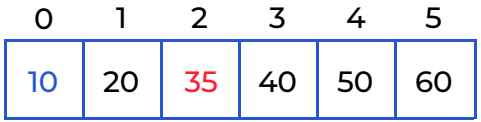
- Блок памяти с элементами **одного** типа



- Объявление в Java

```
// выражение после ‘=’ называется инициализатором
int[] ages = new int[]{10, 20, 30, 40, 50, 60};
```

Основные операции



- Чтение элемента по индексу

```
int myAge = ages[0];  // myAge == 10
```

- Запись элемента по индексу

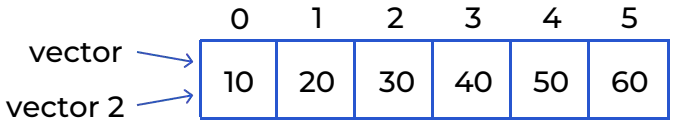
```
ages[2] = 35;  // ages == {10, 20, 35, 40, 50, 60}
```

Массивы — это ссылки

- Присваивание != копирование

```
double[] vector2 = vector;  // копируем ссылку,
но не сами данные
```

- На один массив может ссылаться много переменных



- Длина не обязана быть константой

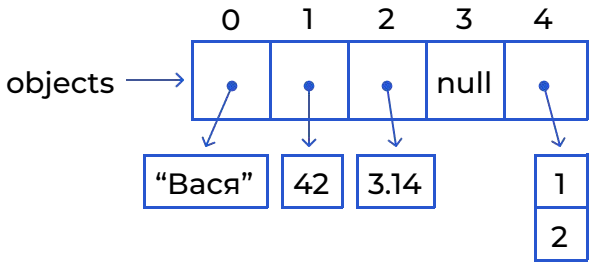
```
int N = …;  // переменная, поле объекта или параметр
double[] vector = new double[N]; // вектор длины N
```

Массивы — это ссылки

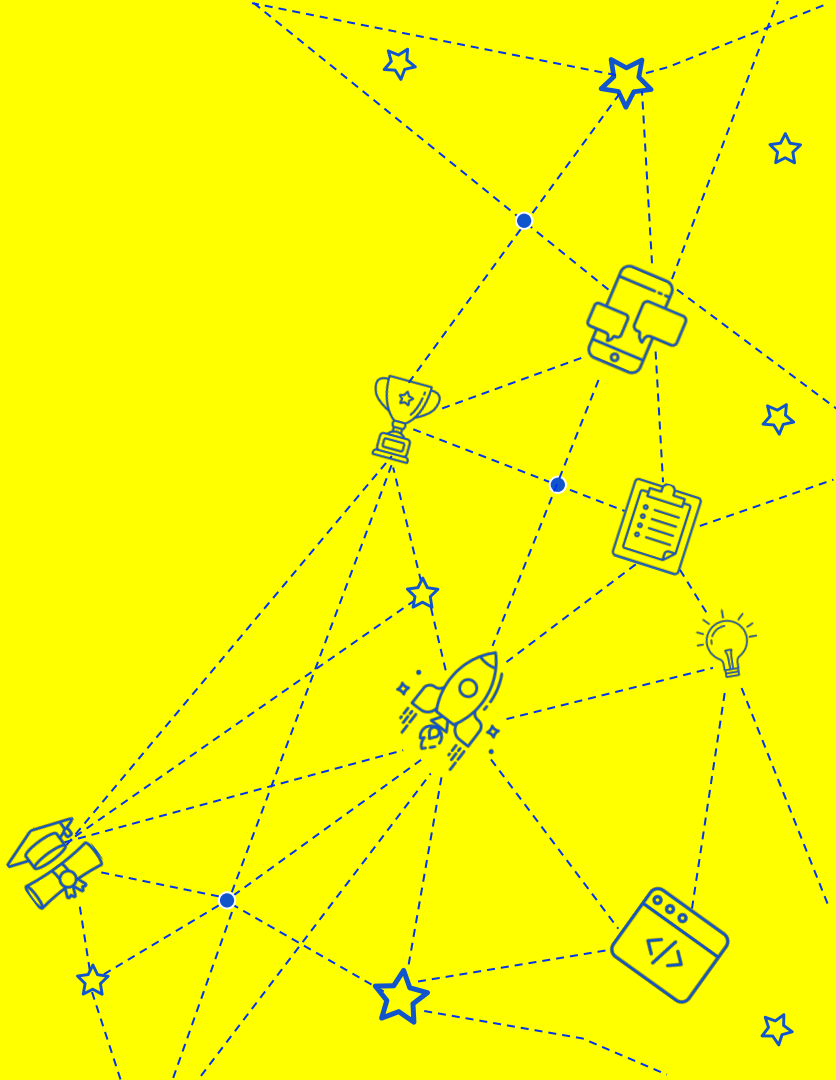
- Тип элемента массива — это общий тип для всех его элементов

```
Object[] objects = new Object[]{
  "Вася", 42, 3.14, null, new int[]{1, 2}
};
```

- Массив элементов ссылочного типа содержит ссылки
- Сами элементы хранятся отдельно в куче



Объявление, инициализация, чтение и запись данных

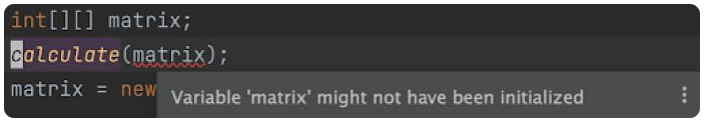


Объявление переменных массива

- Переменная без значения

```
int[] numbers;   // массив целых чисел
```

- Изначально содержит null, но использовать нельзя



Объявление переменных массива

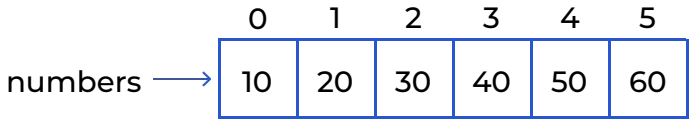
- Нужно присвоить значение

```
numbers = new int[] {10, 20, 30, 40, 50, 60}; // литерал
```

- Или явно присвоить null

```
numbers = null;
```

- Переменная массива содержит ссылку на блок памяти



Чтение элементов

- Чтение по индексу **A[i]**

```
String[] words = {"один", "два", "три"};
String word = words[1];  // word == “два”
```

- Индексы в диапазоне **[0, A.length - 1]**

```
int n = words.length;    // n == 3
```

- Новый массив заполнен значениями по умолчанию (default values)

```
int[] numbers = new int[5];

int n = numbers[3]; // n == 0 - значение по умолчанию для int
```

Чтение элементов в цикле

- Цикл for-each неявно читает элементы массива и присваивает переменной

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

```
public static double euclideanLength(double[] vector) {
        double sum = 0;

        for (double element : vector) {
            sum += element * element;
        }

        return Math.sqrt(sum);
 }
```

Запись элементов

- Новый массив заполняется нулями

```
int[] ages = new int[4];
```

- Выражение для записи
  A[i] = expr

```
ages[0] = 25;
ages[1] = 30;
ages[2] = 35;
ages[3] = 40;
```

Запись элементов

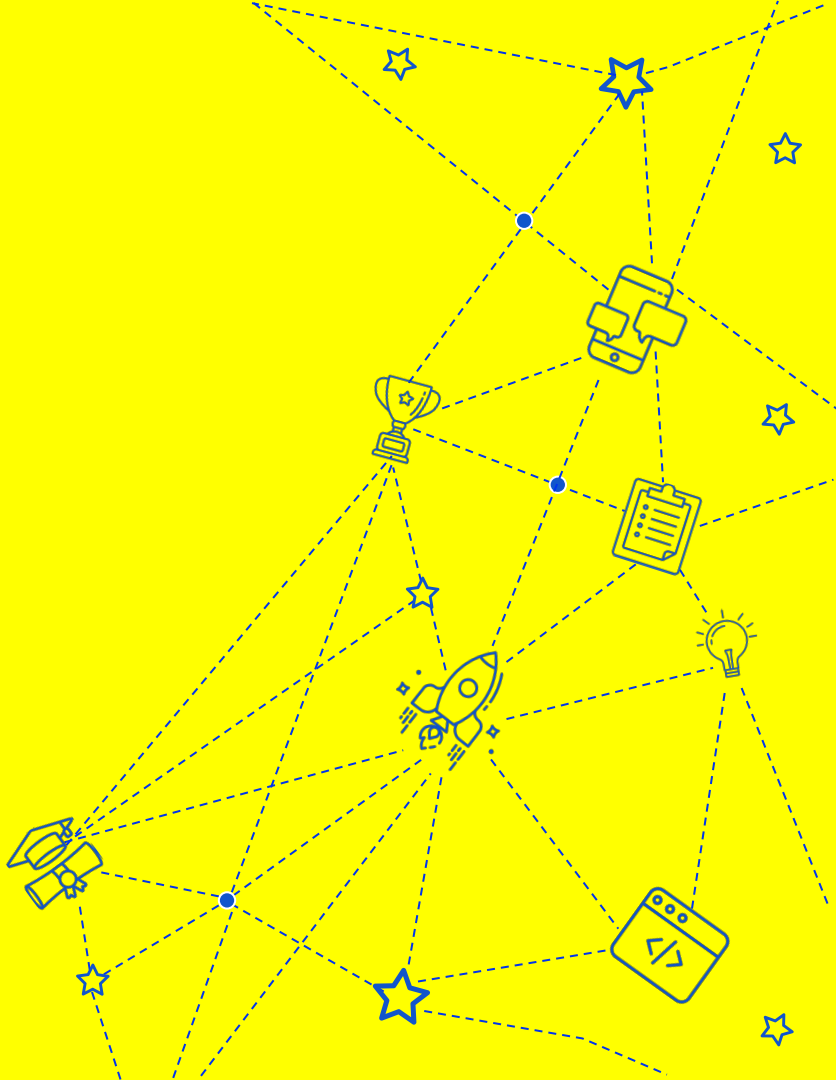
- New размещает в памяти сразу все элементы

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

- Запись можно делать в любом порядке

```
for (int i = squares.length - 1; i >= 0; i--) {
      squares[i] = i * i;
}
```

Многомерные массивы



Объявление многомерных массивов

- Объявление переменной под многомерный массив

```
float[] vector;  // размерность 1
float[][] matrix; // размерность 2
float[][][] matrices; // массив матриц (размерность 3)
```

- Типы проверяются

```
float[][] matrix = new float[4];
⓵ incompatible types: float[] cannot converted to float[][]
```

- Размер нельзя указывать в типе массива

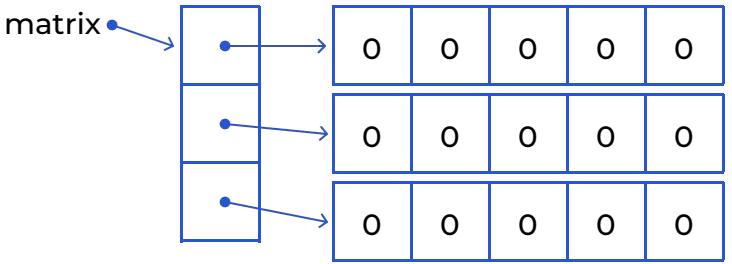
```
float [3][] matrix = new float [3][];
```

Объявление многомерных массивов

- Порядок измерений важен
  Каждая пара скобок —
  это одно измерение

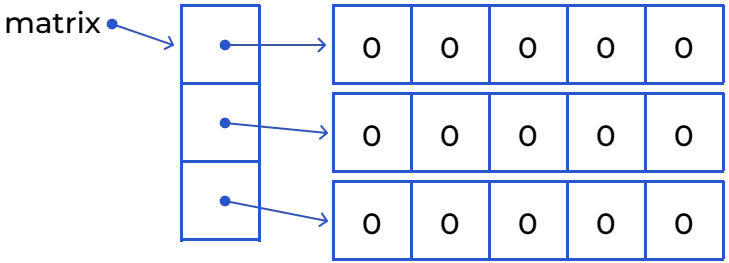
```
int nRows = 3; // количество строк матрицы
int nColumns = 5; // количество столбцов матрицы
float[][] matrix = new float[nRows][nColumns];
```

- Изначально все массивы
  заполняются нулями



Объявление многомерных массивов

- Массив размерности N хранит ссылки на массивы размерности N-1



Заполнение многомерных массивов

- Все строки одной длины если размер указан

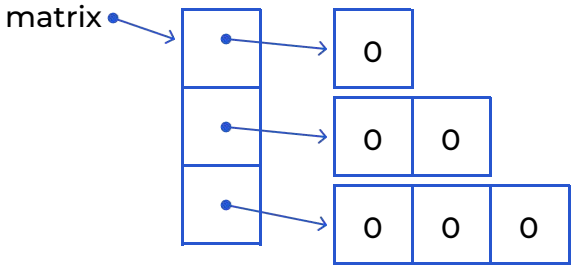
```
float[][] matrix = new float[nRows][nColumns];
```

- Нет размера — нет размещения в памяти

```
float[][] matrix = new float[n][];
for (int i = 0; i < matrix.length; i++)
    matrix[i] = new float[i + 1];
```

matrix[i] имеет тип float[]

- Можем создавать строки матрицы как угодно

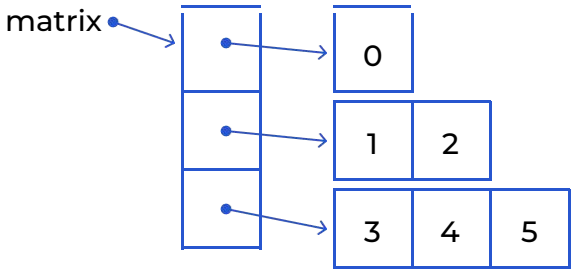


Заполнение многомерных массивов

- Длина массива хранится в поле length
- Количество итераций цикла будет зависеть от длины

```
int[][] matrix = new int[3][];
int counter = 0;
for (int iRow = 0; iRow < matrix.length; iRow++) {
      int[] row = new int[iRow + 1];
      for (int iCol = 0; iCol < row.length; iCol++) {
            row[iCol] = counter;
            counter++;
      }
      matrix[iRow] = row;
}
```

- Для каждой строки:
  — создаем
  — заполняем
  — сохраняем в матрице



Чтение данных вложенного массива

- Если массив M имеет размерность N, то M[i] будет размерности N-1

```
int[][] matrix = {
       {1, 2, 3},
       {4, 5, 6},
       {7, 8, 9}
};
int[] row = matrix[0];  // {1, 2, 3}
```

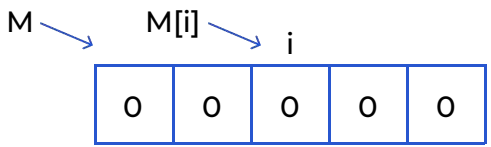
- Чтение из матрицы требует **2 шага:**
  — читаем строку (по индексу)
  — читаем элемент (по индексу)

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

System.out.println(
    "Сумма элементов главной диагонали: "
    + diagonalSum);
```

Запись данных во вложенный массив

- Что означает выражение M[i] — это ключ к пониманию записи в массив
- M[i] — это позиция элемента i внутри блока памяти на который ссылается M

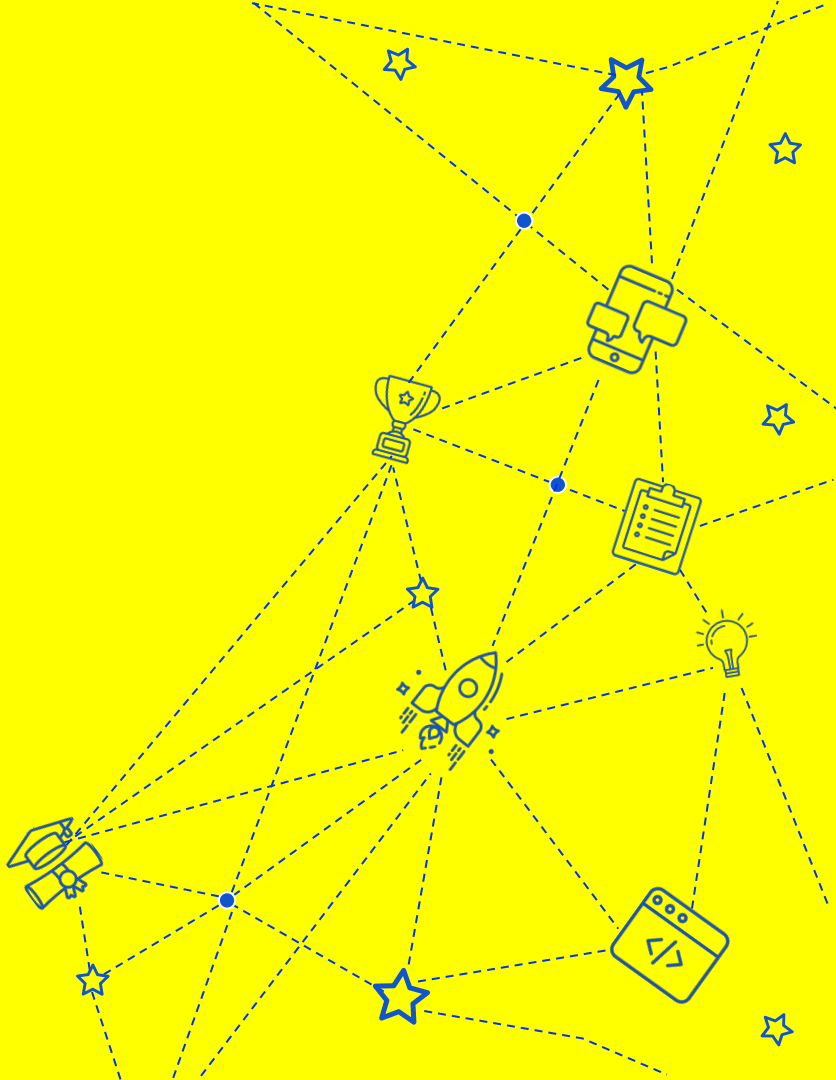


- Запись происходит в позицию последнего индекса **j**, слева от присваивания

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        matrix[i][j] = matrix[i][j] * 2;
    }
}
```

Чтение Запись

Методы для работы с массивами



Поля и методы массивов

Массивы в Java (объекты типа T[]) имеют одно поле
и несколько методов, унаследованных от класса Object

Тип значения	Поле	Описание
int	length	Длина массива (количество элементов в массиве)

Методы, унаследованные от класса Object

Тип результата	Метод	Описание
T[]	clone()	Создает копию массива
boolean	equals(Object)	Сравнивает массивы на равенство по ссылкам
int	hashCode()	Возвращает хеш-код массива
String	toString	Возвращает строковое представление массива

Полезные методы

Некоторые статические методы класса java.util.Arrays

Тип результата	Метод	Описание
boolean	equals(array1, array2)	Сравнивает два массива на равенство по элементам
void	fill(array, value)	Заполняет массив указанным значением
T[]	copyOf(original, newLength)	Копирует указанный массив в новый массив указанной длины
T[]	copyOfRange(original, 1, 4)	Копирует указанный диапазон элементов исходного массива в новый массив
void	sort(array)	Сортирует элементы массива в порядке возрастания
int	binarySearch(sortedArray, key)	Выполняет двоичный поиск указанного значения в отсортированном массиве

Метод Arrays.equals

Метод equals() сравнивает два массива на равенство по элементам

Пример:

```
int[] array1 = {1, 2, 3};
int[] array2 = {1, 2, 3};
boolean isEqual = Arrays.equals(array1, array2); // true
```

Метод Arrays.fill

Метод fill() заполняет массив указанным значением

Пример:

```
int[] intArray = new int[5];
Arrays.fill(intArray, 42); // {42, 42, 42, 42, 42}

String[] stringArray = new String[3];
Arrays.fill(stringArray, "Hello"); // {"Hello", "Hello", "Hello"}

boolean[] boolArray = new boolean[4];
Arrays.fill(boolArray, true); // {true, true, true, true}
```

Метод Arrays.copyOf

Метод copyOf() копирует содержимое массива в новый массив

Пример:

```
int[] original = {1, 2, 3};
int[] copied1 = Arrays.copyOf(original, 5); // {1, 2, 3, 0, 0}
int[] copied2 = Arrays.copyOf(original, 2); // {1, 2}

String[] stringArray = {"apple", "banana", "cherry"};
String[] copiedStringArray = Arrays.copyOf(stringArray, 4); // {"apple", "banana", "cherry", null}
```

Метод Arrays.copyOfRange

Метод copyOfRange() копирует указанный диапазон элементов исходного массива в новый массив

Пример:

```
int[] original = {1, 2, 3, 4, 5};
int[] subArray1 = Arrays.copyOfRange(original, 1, 4); // {2, 3, 4}
int[] subArray2 = Arrays.copyOfRange(original, 3, 7); // {4, 5, 0, 0}

String[] stringArray = {"apple", "banana", "cherry", "orange", "grape"};
String[] subStringArray = Arrays.copyOfRange(stringArray, 1, 3); // {"banana", "cherry"}
```

Метод Arrays.sort

Метод sort() сортирует элементы массива в порядке возрастания

Пример:

```
int[] intArray = {3, 1, 4, 1, 5, 9};
Arrays.sort(intArray); // {1, 1, 3, 4, 5, 9}

String[] stringArray = {"apple", "orange", "banana", "cherry"};
Arrays.sort(stringArray); // {"apple", "banana", "cherry", "orange"}

Person[] people = {
    new Person("Alice", 30), new Person("Bob", 25),
    new Person("Charlie", 35) };

Arrays.sort(people);
// Результат: {Person("Bob", 25), Person("Alice", 30), Person("Charlie", 35)}
```

Пример:

```
class Person implements Comparable<Person> {
        String name; int age;

@Override
        public int compareTo(Person other) {
            return Integer.compare(this.age, other.age);
        }
}
```

Метод Arrays.binarySearch

Метод binarySearch() выполняет двоичный поиск указанного значения в отсортированном массиве

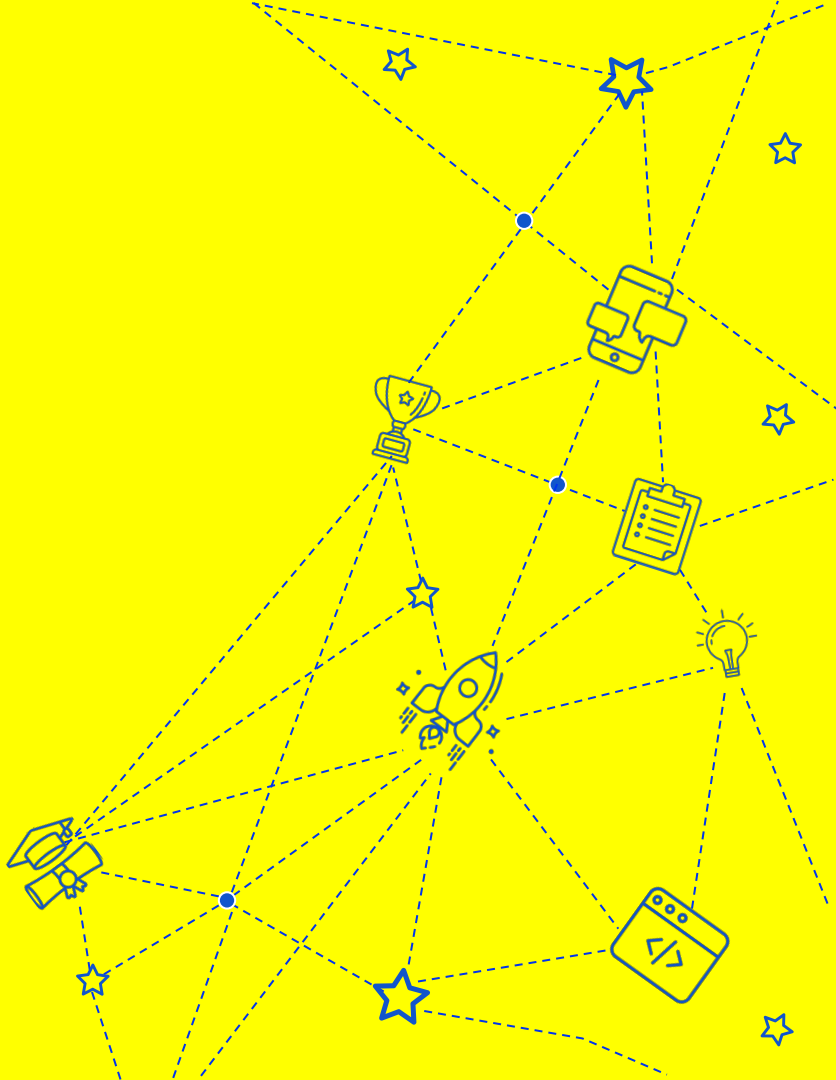
Возвращает:

- индекс i >= 0 найденного элемента
- либо отрицательное число равное (-(position) - 1)

Пример:

```
String[] sortedStringArray = {"apple", "banana", "cherry", "orange"};
int index3 = Arrays.binarySearch(sortedStringArray, "banana"); // 1
int index4 = Arrays.binarySearch(sortedStringArray, "grape"); // -4
```

Исключения, связанные с массивами

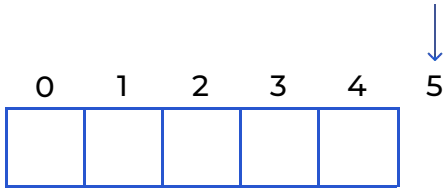


ArrayIndexOutOfBoundsException

- Возникает, когда при обращении к элементу массива указан недопустимый индекс
- Индекс i < 0 или i >= длина массива

Пример:

```
try {
	int[] numbers = new int[5];
	numbers[5] = 10; // Выход за пределы массива
} catch (ArrayIndexOutOfBoundsException e) {
	System.out.println("Ошибка: выход за пределы
массива");
}
```



NegativeArraySizeException

Исключение, которое возникает, когда при создании массива указана отрицательная размерность

Пример:

```
public class NegativeArraySizeExample {
       public static void main(String[] args) {
              try {
                 int start = …; int end = …;
                 int[] array = createSlice(start, end);
              } catch (NegativeArraySizeException e) {
                    System.out.println("Ошибка: невозможно создать массив с отрицательной длиной.");
              }
       }

       public static int[] createSlice(int start, int end) {
              int len = end - start;
              return new int[len];
       }
}
```

OutOfMemoryError

Возникает, когда виртуальная машина Java (JVM)
не может выделить достаточно памяти

Пример:

```
public class OutOfMemoryInput {
	public static void main(String[] args) {
		// create byte ByteBuffer
		String input = System.console().readLine("Enter size: ");
		int size = Integer.parseInt(input);
		try {
			long[] items = new long[size];
		} catch (OutOfMemoryError e) {
			System.out.println("Error: " + e.getMessage());
		}
	}
}
```

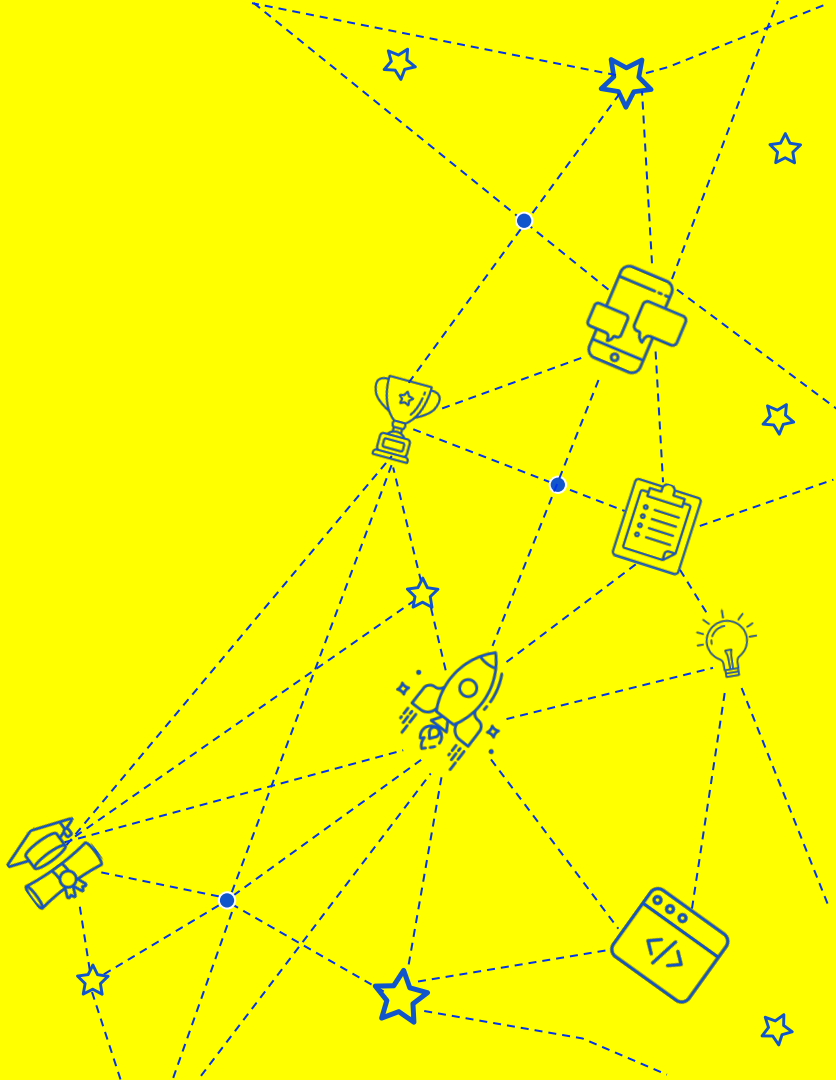
ArrayStoreException

- Возникает, при попытке сохранить в массиве значение несовместимого типа
- Совместимость определяется возможностью присваивания
- Можно проверять во время исполнения:
  objectArray.getClass()
  .getComponentType()
  .isAssignableFrom(Integer.class)

Пример:

```
Object[] objectArray = new String[5];
try {
       objectArray[0] = 42; // Несовместимость типов: Integer вместо String
} catch (ArrayStoreException e) {
       System.out.println(
         "Ошибка: несовместимость типов при присвоении элемента массиву.");
}
```

Java Collections и массивы



Зачем нужны коллекции

1. Более удобная работа с данными
2. Динамическая размерность
3. Использование дополнительных функций коллекций
4. Улучшение читаемости и поддерживаемости кода
5. Повышение производительности

Преобразование массивов в коллекции

- Используйте метод Arrays.asList

```
public class Arrays {
    ...
    public static <T> List<T> asList(T... a) {...}
    ...
}
```

- Конвертация массива объектов в список (List)

```
String[] stringArray = {"apple", "banana", "cherry"};
List<String> stringList = Arrays.asList(stringArray);
```

Преобразование коллекций в массивы

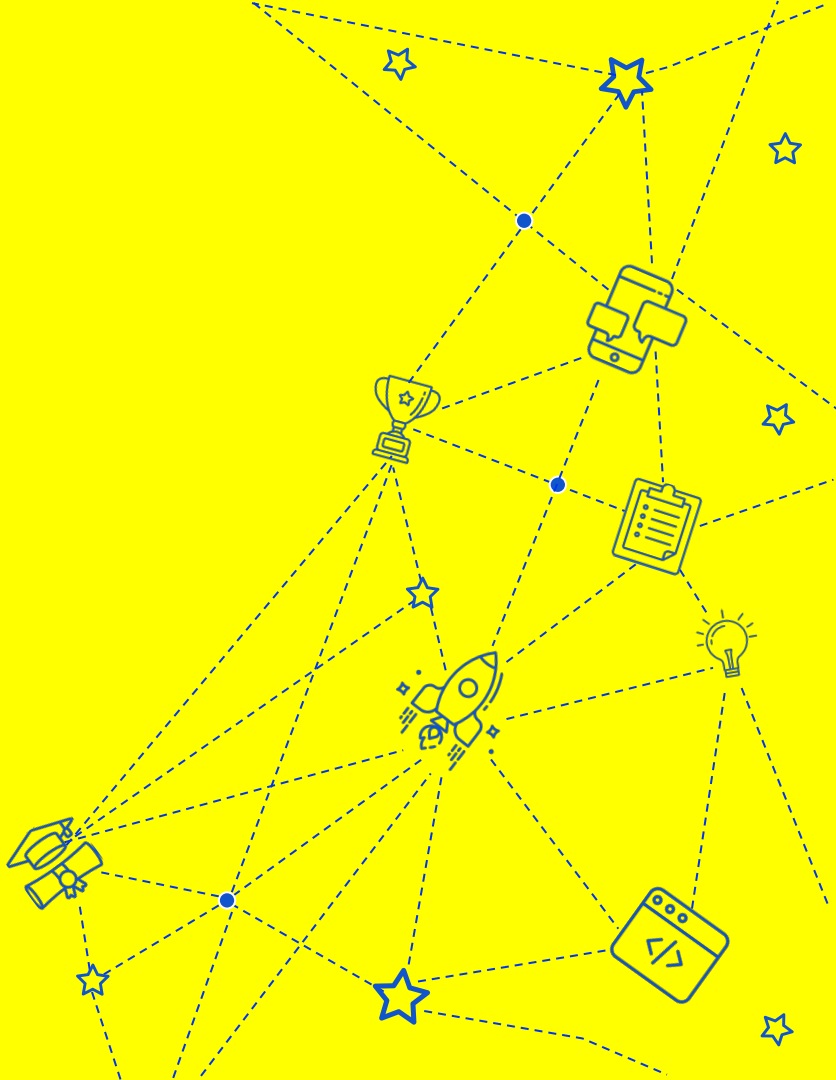
- Используйте метод интефейса Collection.toArray

```
interface Collection<E> extends Iterable<E> {
    ...
    <T> T[] toArray(T[] a);
    ...
}
```

- Конвертация списка (List) в массив объектов

```
List<String> stringList = Arrays.asList("apple", "banana", "cherry");
String[] stringArray = stringList.toArray(new String[0]);
```

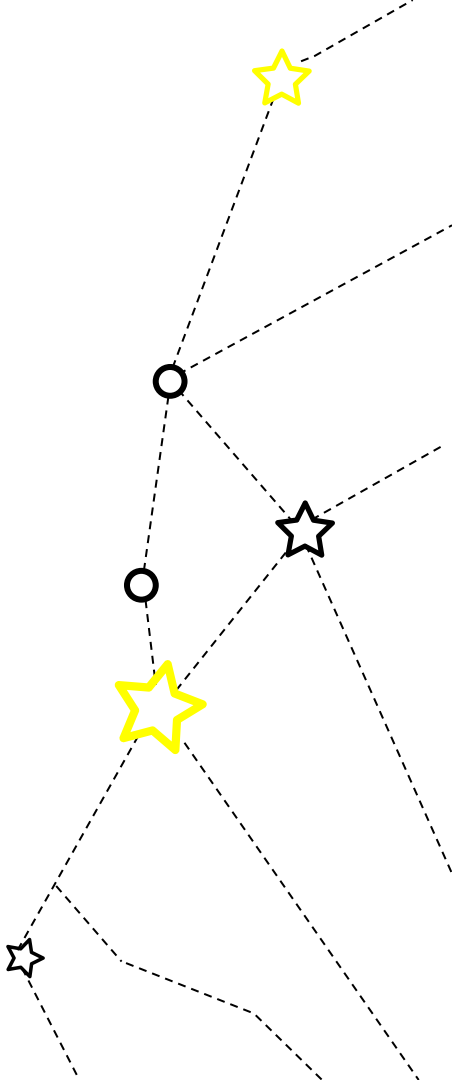
Воркшоп

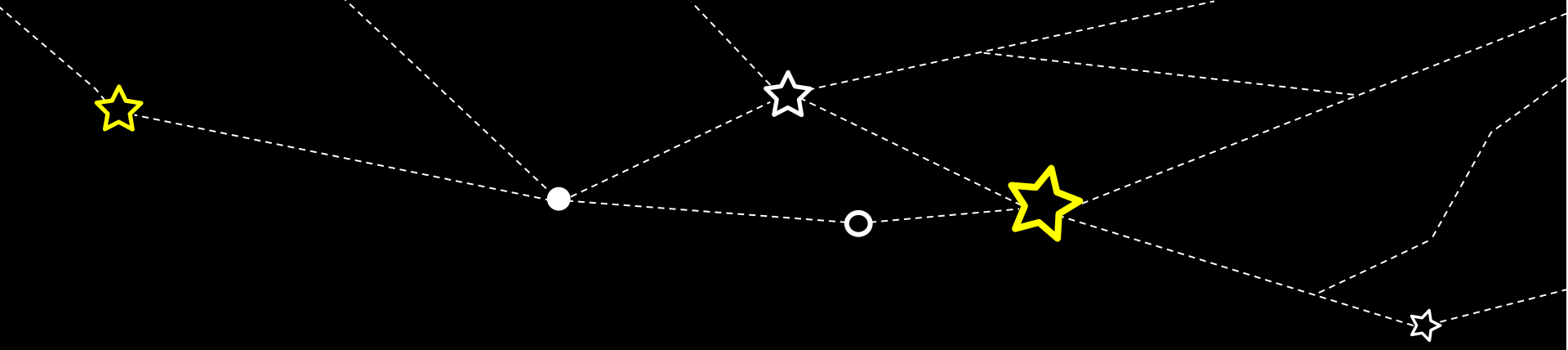


Воркшоп

Библиотека матричных операций

- Постановка задачи
- Основные операции
- Представление плотных матриц
- Представление разреженных матриц
- Операции над плотными матрицами
- Операции над разреженными матрицами





СПАСИБО ЗА ВНИМАНИЕ

 @avslesarenko

 aslesarenko

 aslesarenko

Домашнее задание

1. Сделать клон репозитория
   https://github.com/aslesarenko/java-arrays
2. Открыть проект в IntelliJ IDEA
   (достаточно бесплатной Community Edition)
3. Запустить метод Matrices.main и убедиться, что выводится ошибка java.lang.UnsupportedOperationException: Метод не реализован
4. В файле src/Matrices.java заменить все `todo()` правильной реализацией
5. Добиться того чтобы программа успешно отрабатывала