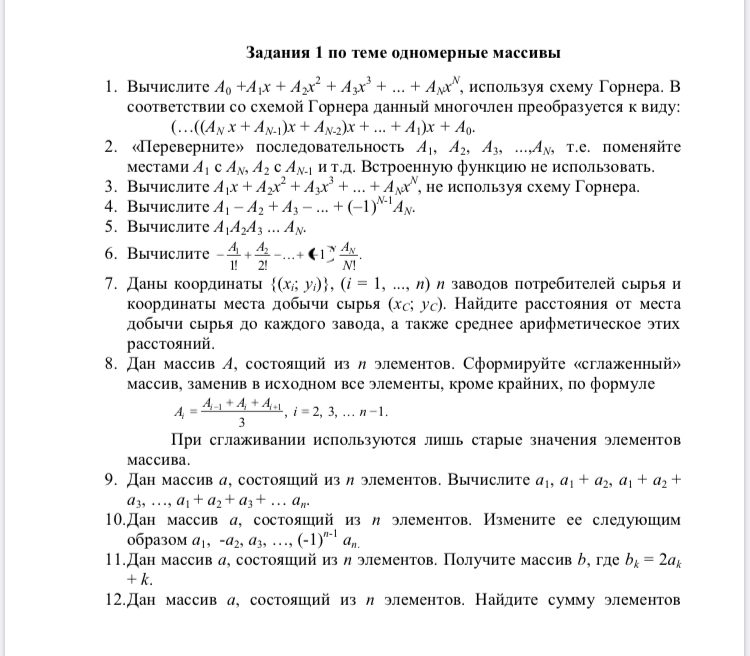
# Лабораторная работа № 2

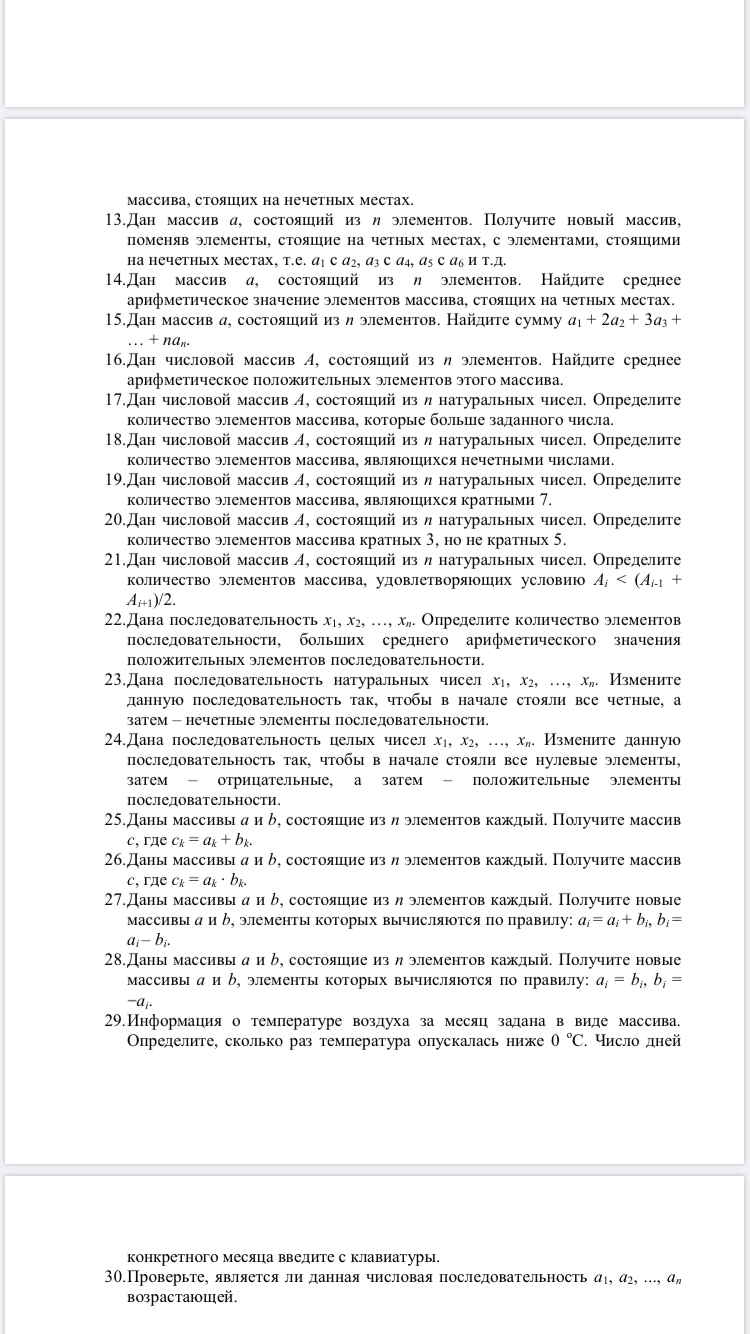
# Тема работы: «Разработка программ с использованием массивов»

**Цель работы:** Закрепить навык работы с массивами различных видов.

**Задание:** Два задания обязательны для выполнения.

Номер варианта соответствует вашему номеру по списку.

****

****

**Задания 2 по теме многомерные массивы**

1. В данной матрице определите количество столбцов, у которых элементы расставлены в порядке возрастания.
2. Дан массив *А*, состоящий из *n* натуральных чисел. Найдите элемент массива, сумма цифр которого наибольшая.
3. Дана действительная квадратная матрица *А* порядка *n*. Найдите количество строк матрицы, сумма модулей элементов которых больше 1.
4. Дана действительная квадратная матрица порядка *n*. Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 4.1).
5. Дана действительная квадратная матрица порядка *n*. Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 4.2).
6. Дана действительная квадратная матрица порядка *n*. Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 4.3).
7. Дана квадратная матрица *А* порядка *n*. Найдите номер строки матрицы, в которой больше всего единичных элементов.
8. Дана квадратная матрица *А* порядка *n*. Найдите среднее арифметическое положительных элементов каждого столбца матрицы.
9. Дана квадратная матрица *А* порядка *n*. Найдите сумму положительных элементов матрицы, стоящих под главной диагональю.
10. Дана квадратная матрица *А* порядка *n*. Найдите суммы элементов тех строк матрицы, на главной диагонали которой стоят отрицательные элементы.
11. Дана квадратная матрица *А* порядка *n*. Постройте вектор, элементы которого являются наибольшими числами каждой строки матрицы.
12. Дана квадратная матрица *А* порядка *n*. Проверьте, является ли данная матрица симметричной.
13. Дана квадратная матрица *А* порядка *n*. Проверьте, является ли матрица единичной.
14. Дана квадратная матрица *А* порядка *n*. Транспонируйте данную матрицу.
15. Дана матрица *A* порядка *n*. Найдите два наибольших элемента матрицы с указанием номеров строк и столбцов, в которых они находятся.
16. Дана матрица *A* порядка *n*. Найдите наибольший среди отрицательных элементов матрицы.
17. Дана матрица *A* порядка *n*. Отсортируйте строки матрицы в порядке возрастания наибольших элементов в каждой строке.
18. Дана матрица *A* порядка *n*. Поменяйте местами наибольший и наименьший элементы матрицы.
19. Дана матрица *A* порядка *n*. Поменяйте местами строки: первую с

последней, вторую с предпоследней и т.д.

1. Дана матрица *A* порядка *n*. Расставьте строки матрицы в порядке возрастания количества нулевых элементов.
2. Дана матрица *A* порядка *n*. Расставьте элементы каждой строки в порядке возрастания.
3. Дана матрица *A* порядка *n*. Расставьте элементы строк с четными номерами матрицы в порядке убывания.
4. Дана матрица *А*, имеющая n строк и m столбцов, содержащая оценки группы за первый семестр. Выведите номера отличников (оценки не ниже 8).
5. Даны две матрицы одинаковой размерности. Найдите разность этих матриц.
6. Даны две матрицы одинаковой размерности. Найдите сумму этих матриц.
7. Даны две последовательности *a*1, *a*2, ..., *an* и *b*1, *b*2, ..., *bn* целых чисел. Среди членов каждой последовательности нет повторяющихся чисел.
8. Получите все члены последовательности *b*1, *b*2, ..., *bn*, которые не входят в последовательность *a*1, *a*2, ..., *an*.
9. Если все элементы какой-либо строки данной матрицы равны между собой, то все элементы такой строки замените нулями.
10. Найдите наибольшую сумму модулей элементов строк заданной матрицы.
11. Определите количество строк заданной матрицы, которые упорядочены по возрастанию.
12. Сложите две треугольные матрицы порядка *n*, у которых только элементы над главной диагональю отличны от нуля.
    1. **Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **IntelliJ IDEA**.

* 1. **Основные теоретические сведения**

В любом языке программирования используются массивы, удобные для работы с большим количеством однотипных данных. Если вам нужно обработать сотни переменных, то вызывать каждую по отдельности становится муторным занятием. В таких случаях проще применить массив. Для наглядности представьте себе собранные в один ряд пустые коробки. В каждую коробочку можно положить что-то одного типа, например, котов. Теперь, даже не зная их по именам, вы можете выполнить команду **Накормить кота из 3 коробки**. Сравните с командой **Накормить Рыжика**. Чувствуете разницу? Вам не обязательно знать котов по именам, но вы всѐ равно сможете справиться с заданием. Завтра в этих коробках могут оказаться другие коты, но это не

составит для вас проблемы, главное знать номер коробки, который называется индексом.



Еще раз повторим теорию. Массивом называется именованное множество переменных одного типа. Каждая переменная в данном массиве называется элементом массива. Чтобы сослаться на определенный элемент в массиве нужно знать имя массива в соединении с целым значением, называемым индексом. Индекс указывает на позицию конкретного элемента относительно начала массива. Обратите внимание, что первый элемент будет иметь индекс 0, второй имеет индекс 1, третий - индекс 2 и так далее. Данное решение было навязано математиками, которым было удобно начинать отсчѐт массивов с нуля.

**Объявление массива**

Переменную массива можно объявить с помощью квадратных скобок:

int[] cats; // мы объявили переменную массива

Возможна и альтернативная запись:

int cats[]; // другой вариант

Здесь квадратные скобки появляются после имени переменной. В разных языках программирования используются разные способы, и Java позволяет вам использовать тот вариант, к которому вы привыкли. Но большинство предпочитает первый вариант. Сами квадратные скобки своим видом напоминают коробки, поэтому вам будет просто запомнить.

Мы пока только объявили массив, но на самом деле его ещѐ не существует, так как не заполнен данными. Фактически значение массива равно *null*.

**Определение массива**

После объявления переменной массива, можно определить сам массив с помощью ключевого слова **new** с указанием типа и размера. Например, массив должен состоять из 10 целых чисел:

cats = new int[10];

Можно одновременно объявить переменную и определить массив (в основном так и делают):

int[] cats = new int[10];

Если массив создается таким образом, то всем элементам массива автоматически присваиваются значения по умолчанию. Например, для числовых значений начальное значение будет 0. Для массива типа *boolean* начальное значение будет равно *false*, для массива типа *char* - '\u0000', для массива типа класса (объекты) - *null*.

Последнее правило может запутать начинающего программиста, который забудет, что строка типа **String** является объектом. Если вы объявите массив из десяти символьных строк следующим образом:

String[] catNames = new String[10];

То у вас появятся строки со значением *null*, а не пустые строки, как вы могли бы подумать. Если же вам действительно нужно создать десять пустых строк, то используйте, например, такой код:

for (int i = 0; i < 10; i++) catNames[i] = "";

**Доступ к элементам массива**

Обращение к элементу массива происходит по имени массива, за которым следует значение индекса элемента, заключенного в квадратные скобки. Например, на первый элемент нашего массива **cats** можно ссылаться как на **cats[0]**, на пятый элемент как **cats[4]**.

В качестве индекса можно использовать числа или выражения, которые создают положительное значение типа **int**. Поэтому при вычислении выражения с типом **long**, следует преобразовать результат в **int**, иначе получите ошибку. С типами **short** и **byte** проблем не будет, так как они полностью укладываются в диапазон **int**.

**Инициализация массива**

Не всегда нужно иметь значения по умолчанию. вы можете инициализировать массив собственными значениями, когда он объявляется, и определить количество элементов. Вслед за объявлением переменной массива добавьте знак равенства, за которым следует список значений элементов, помещенный в фигурные скобки. В этом случае ключевое слово **new** не используется:

int[] cats = {2, 5, 7, 8, 3, 0}; // массив из 6 элементов

Можно смешать два способа. Например, если требуется задать явно значения только для некоторых элементов массива, а остальные должные иметь значения по умолчанию.

int[] cats = new int[6]; // массив из шести элементов с начальным значением 0 для каждого элемента

cats[3] = 5; // четвертому элементу присвоено значение 5 cats[5] = 7; // шестому элементу присвоено значение 7

Массивы часто используют в циклах. Допустим, 5 котов отчитались перед вами о количестве пойманных мышек. Как узнать среднее арифметическое значение:

int[] mice = {4, 8, 10, 12, 16};

int result = 0;

for(int i = 0; i < 5; i++){ result = result + mice[i];

}

result = result / 5;

System.out.println("Среднее арифметическое: " + result);

Массив содержит специальное поле **length**, которое можно прочитать (но не изменить). Оно позволяет получить количество элементов в массиве. Данное свойство удобно тем, что вы не ошибѐтесь с размером массива. Последний элемент массива всегда **mice[mice.length - 1]**. Предыдущий пример можно переписать так:

int[] mice = { 4, 8, 10, 12, 16 };

int result = 0;

for (int i = 0; i < mice.length; i++) { result = result + mice[i];

}

result = result / mice.length; // общий результат делим на число элементов в массиве

System.out.println("Среднее арифметическое: " + result);

Теперь длина массива вычисляется автоматически, и если вы создадите новый массив из шести котов, то в цикле ничего менять не придѐтся.

Если вам нужно изменять длину, то вместо массива следует использовать списочный массив **ArrayList**. Сами массивы неизменяемы.

Будьте осторожны с копированием массивов. Массив - это не числа, а специальный объект, который по особому хранится в памяти. Чтобы не загромождать вас умными словами, лучше покажу на примере.

Допустим, у нас есть одна переменная, затем мы создали вторую переменную и присвоили ей значение первой переменной. А затем проверим их.

int a = 5; int b = a;

System.out.println("a = " + a + "\nb = " + b);

Получим ожидаемый результат.

a = 5

b = 5

Попробуем сделать подобное с массивом.

int[] anyNumbers = {2, 8, 11}; int[] luckyNumbers = anyNumbers; luckyNumbers[2] = 25;

mInfoTextView.setText("anyNumbers: " + Arrays.toString(anyNumbers)

+ "\nluckyNumbers: " + Arrays.toString(luckyNumbers));

Получим результат.

anyNumbers: [2, 8, 25];

luckyNumbers: [2, 8, 25];

Мы скопировали первый массив в другую переменную и в ней поменяли третий элемент. А когда стали проверять значения у обоих массивов, то оказалось, что у первого массива тоже поменялось значение. Но мы же его не трогали! Магия. На самом деле нет, просто массив остался прежним и вторая переменная обращается к нему же, а не создаѐт вторую копию. Помните об этом.

Если же вам реально нужна копия массива, то используйте метод Arrays.copyOf()

Если ваша программа выйдет за пределы индекса массива, то программа остановится с ошибкой времени исполнения **ArrayOutOfBoundsException**. Это очень частая ошибка у программистов, проверяйте свой код.

**Перебор значений массива**

Массивы часто используются для перебора всех значений. Стандартный способ через цикл **for**

int aNums[] = { 2, 4, 6 };

for (int i = 0; i < aNums.length; i++) {

String strToPrint = "aNums[" + i + "]=" + aNums[i];

}

Также есть укороченный вариант записи

for (int num : aNums) { String strToPrint = num;

}

Нужно только помнить, что в этом случае мы не имеем доступа к индексу массива, что не всегда подходит для задач. Поэтому используется только для обычного перебора элементов.

**Многомерные массивы**

Для создания многомерных массивов используются дополнительные скобки:

int[][] a = {

{ 1, 2, 3 },

{ 4, 5, 6 }

}

Также массив может создаваться ключевым словом **new**:

// трехмерный массив фиксированной длины int[][][] b = new int[2][4][4];

**Двумерный массив**

Двумерный массив - это массив одномерных массивов. Если вам нужен двумерный массив, то используйте пару квадратных скобок:

String[][] arr = new String[4][3]; arr[0][0] = "1";

arr[0][1] = "Васька";

arr[0][2] = "121987102";

arr[1][0] = "2";

arr[1][1] = "Рыжик";

arr[1][2] = "2819876107";

arr[2][0] = "3";

arr[2][1] = "Барсик";

arr[2][2] = "412345678";

arr[3][0] = "4";

arr[3][1] = "Мурзик";

arr[3][2] = "587654321";

Представляйте двумерный массив как таблицу, где первые скобки отвечают за ряды, а вторые - за колонки таблицы. Тогда пример выше представлят собой таблицу из четырѐх рядов и трѐх колонок.

|  |  |  |
| --- | --- | --- |
| 1 | Васька | 121987102 |
| 2 | Рыжик | 2819876107 |
| 3 | Барсик | 412345678 |
| 4 | Мурзик | 587654321 |

Для двумерных массивов часто используются два цикла **for**, чтобы заполнить элементы данными слева направо и сверху вниз. Напишем такой код:

int[][] twoD = new int[3][4]; // объявили двухмерный массив int i, j, k = 0;

for (i = 0; i < 3; i++)

for (j = 0; j < 4; j++) { twoD[i][j] = k; k++;

}

for (i = 0; i < 3; i++) {

for (j = 0; j < 4; j++) System.out.println(twoD[i][j] + " ");

System.out.println("\n");

}

В данном примере мы сначала заполнили двумерый массив данными, а затем снова прошлись по этому массиву для считывания данных.

Логическое представление данного двухмерного массива будет выглядеть следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| [0, 0] | [0, 1] | [0, 2] | [0, 3] |
| [1, 0] | [1, 1] | [1, 2] | [1, 3] |
| [2, 0] | [2, 1] | [2, 2] | [2, 3] |

Первое число в скобках обозначают ряд (строку), а второе число - столбец. Принято считать, что в массиве **new int[M][N]** первый размер означает количество строк, а второй - количество столбцов.

На экране после запуска примера мы увидим следующее:

0 1 2 3

4 5 6 7

8 9 10 11

При резервировании памяти под многомерный массив необходимо указать память только для первого измерения. Для остальных измерений память можно выделить отдельно.

int[][] twoD = new int[3][]; // память под первое измерение

// далее резервируем память под второе измерение twoD[0] = new int[4];

twoD[1] = new int[4]; twoD[2] = new int[4];

В данном примере особого смысла в этом нет.

Еще одна интересная особенность при создании массива связана с запятой. Посмотрите на пример.

int[][] a = {{1, 2, 3}, {4, 0, 0,},};

System.out.println(Arrays.deepToString(a));

Вроде в конце используется лишняя запятая, но еѐ наличие не приведѐт к ошибке (только одна запятая). Это бывает удобно, когда надо скопировать или вставить кусок массива в коде. Кстати, метод **deepToString()** класса Arrays очень удобен для вывода двухмерных массивов.

Чтобы совсем сбить вас с толку, приведу ещѐ один правильный пример.

Integer[] Integer[] = {{1, 2, 3}, {4, 0, 0,},}; System.out.println(Arrays.deepToString(Integer));

Квадратные скобки можно использовать двумя способами. Сначала мы поставили скобки у типа переменной, а потом у имени переменной. При этом мы использовали в качестве имени имя класса **Integer**. Однако, Java догадывается, что на этот раз используется не класс, а имя и разрешает такой синтаксис. Но лучше так не выпендриваться.

**Размер имеет значение**

Размер двумерного массива измеряется интересным способом. Длина массива определяется по его первой размерности, то есть вычисляется количество рядов.

int[][] matrix = new int[4][5]; System.out.println(matrix.length);

А если мы хотим узнать количество столбцов в ряду? Тогда указываете ряд, а затем вычисляете у него количество столбцов.

// число колонок у третьего ряда System.out.println(matrix[2].length);

Не забывайте, что в массивах ряды могут содержать разное количество столбцов.

**Сложить два массива**

Предположим, у вас есть два массива, и вам нужно их соединить и получить общий массив.

private double[] concatArray(double[] a, double[] b) { if (a == null)

return b; if (b == null)

return a;

double[] r = new double[a.length + b.length]; System.arraycopy(a, 0, r, 0, a.length); System.arraycopy(b, 0, r, a.length, b.length); return r;

}

Вместо типа *double* вы можете использовать другие типы. Вот например, пример сложения двух строковых массивов:

// метод для склеивания двух строковых массивов private String[] concatArray(String[] a, String[] b) {

if (a == null)

return b; if (b == null)

return a;

String[] r = new String[a.length + b.length]; System.arraycopy(a, 0, r, 0, a.length); System.arraycopy(b, 0, r, a.length, b.length); return r;

}

String[] week1 = new String[] { "Понедельник", "Вторник", "Среда" }; String[] week2 = new String[] { "Четверг", "Котопятница", "Субкота",

"Воскресенье" };

String[] week = concatArray(week1, week2); // будет возвращѐн массив всех семи дней недели

**Взять часть массива**

Аналогично, если вам нужно взять только часть из большого массива, то воспользуйтесь методом:

// start - с какой позиции нужно получить новый массив, отсчѐт с 0 private double[] copyPartArray(double[] a, int start) {

if (a == null)

return null;

if (start > a.length) return null;

double[] r = new double[a.length - start]; System.arraycopy(a, start, r, 0, a.length - start); return r;

}

public void onClick(View v) {

double[] digits = new double[] {6.5, 3.1, 5.72}; double[] part = copyPartArray(digits, 1);

Toast.makeText(v.getContext(), part[1] + "", Toast.LENGTH\_LONG).show();

}

Здесь вы также можете использовать другие типы вместо *double*. Вот пример использования:

private String[] copyPartArray(String[] a, int start) { if (a == null)

return null;

if (start > a.length) return null;

String[] r = new String[a.length - start]; System.arraycopy(a, start, r, 0, a.length - start); return r;

}

public void onClick(View v) {

String[] weekday = new String[] { "Понедельник", "Вторник", "Среда"

};

// нам нужен массив со второго элемента

String[] week = copyPartArray(weekday, 1); // вернѐт Вторник и Среда

// выводим второй элемент из полученного массива, т.е. Среда Toast.makeText(v.getContext(), week[1], Toast.LENGTH\_LONG).show();

}

**Перемешать элементы массива**

Бывает необходимость перемешать элементы массива в случайном порядке. В интернете нашѐл готовый метод по алгоритму Fisher–Yates (прим.: Fisher - это рыбак, который ловит рыбу, например, кот)

// Implementing Fisher–Yates shuffle static void shuffleArray(int[] ar) {

Random rnd = new Random();

for (int i = ar.length - 1; i > 0; i--) {

int index = rnd.nextInt(i + 1);

// Simple swap

int a = ar[index]; ar[index] = ar[i]; ar[i] = a;

}

}

// создадим массив и перемешаем его

int[] mSolutionArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,

13, 14 };

shuffleArray(mSolutionArray);

Log.i("Array", Arrays.toString(mSolutionArray));

Помните, что размер массива фиксируется и не может меняться на протяжении его жизненного цикла. Если вам нужно изменять, то используйте **ArrayList**, который способен автоматически выделять дополнительное пространство, выделяя новый блок памяти и перемещая в него ссылки из старого.

При выходе за границу массива происходит исключение **RuntimeException**, свидетельствующее об ошибке программиста.

**Метод arraycopy() - Копирование массива**

Стандартная библиотека Java содержит статический метод **System.arraycopy()**, который копирует массивы значительнее быстрее, чем при ручном копировании в цикле **for**.

В аргументах **arraycopy()** передается исходный массив, начальная позиция копирования в исходном массиве, приѐмный массив, начальная позиция копирования в приѐмном массиве и количество копируемых элементов. Любое нарушение границ массива приведет к исключению.

Разработчик Avi Yehuda написал программу, которая вычисляет время на копирование с помощью цикла **for** и с помощью метода **arraycopy()** на примере с миллионом элементов. Ручное копирование у него заняло 182 мс, с помощью метода **arraycopy()** - 12 мс. Разница колоссальна.

public class ArrayCopyTestActivity extends Activity { private static final int SIZE\_OF\_ARRAY = 1000000;

private long time;

/\*\* Called when the activity is first created. \*/ @Override

public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main);

Integer [] sourceArray = new Integer[SIZE\_OF\_ARRAY]; Integer [] dst = new Integer[SIZE\_OF\_ARRAY]; fillArray(sourceArray);

startBenchmark(); naiveCopy(sourceArray,dst);

stopBenchmark();

startBenchmark();

System.arraycopy(sourceArray, 0, dst, 0, sourceArray.length); stopBenchmark();

}

private void naiveCopy(Integer [] src, Integer [] dst) { for (int i = 0; i < src.length; i++) {

dst[i]=src[i];

}

}

private void fillArray(Integer [] src) { for (int i = 0; i < src.length; i++) {

src[i]=i;

}

}

private void startBenchmark() {

time = System.currentTimeMillis();

}

private void stopBenchmark() {

time = System.currentTimeMillis() - time; Log.d("array copy test", "time="+time);

}

}

**Поиск элементов строкового массива по начальным символам**

Допустим, у нас есть строковый массив и нам нужно по первым символам найти все слова, которые входят в данный массив.

// Сам метод

public static ArrayList<String> searchFromStart(String[] inputArray, String searchText) {

ArrayList<String> outputArray = new ArrayList<>();

for (int i = 0; i < inputArray.length; i++) {

if (searchText.compareToIgnoreCase(inputArray[i].substring(0, searchText.length())) == 0) {

outputArray.add(inputArray[i]);

}

}

return outputArray;

}

// Массив строк

final String[] catNamesArray = new String[] { "Рыжик", "Барсик", "Мурзик", "Мурка", "Васька", "Томасина", "Бобик", "Кристина", "Пушок",

"Дымка", "Кузя", "Китти", "Барбос", "Масяня", "Симба" };

// Применим метод. Ищем по буквам "мур":

List<String> words = searchFromStart(catNamesArray, "мур"); Toast.makeText(getApplicationContext(), words.get(0).toString() + ":" +

words.get(1).toString(), Toast.LENGTH\_SHORT).show();

Вернѐт списочный массив из двух элементов: Мурзик и Мурка.

**Класс Arrays**

Класс **java.util.Arrays** предназначен для работы с массивами. Он содержит удобные методы для работы с целыми массивами:

* **copyOf()** − предназначен для копирования массива
* **copyOfRange()** − копирует часть массива
* **toString()** − позволяет получить все элементы в виде одной строки
* **sort()** — сортирует массив методом quick sort
* **binarySearch()** − ищет элемент методом бинарного поиска
* **fill()** − заполняет массив переданным значением (удобно использовать, если нам необходимо значение по умолчанию для массива)
* **equals()** − проверяет на идентичность массивы
* **deepEquals()** − проверяет на идентичность массивы массивов
* **asList()** − возвращает массив как коллекцию

**Сортировка массива**

Сортировка (упорядочение по значениям) массива a производится методами **Arrays.sort(a)** и **Arrays.sort(a, index1, index2)**. Первый метод упорядочивает в порядке возрастания весь массив, второй — часть элементов (от индекса index1 до индекса index2). Имеются и более сложные методы сортировки. Элементы массива должны быть сравниваемы (поддерживать операцию сравнения).

Простой пример

// задаѐм числа в случайном порядке

int[] numbers = new int[]{1, 23, 3, 8, 2, 4, 4};

// сортируем Arrays.sort(numbers);

// проверяем mInfoTextView.setText(Arrays.toString(numbers));

**Сортировка массива для ArrayAdapter**

Массивы часто используются в адаптерах для заполнения данными компоненты **Spinner**, **ListView** и т.п.

Предположим, у вас есть массив строк и его нужно отсортировать перед отдачей массива адаптеру **ArrayAdapter**. Это позволит вывести строки в упорядоченном виде, например, в **ListView**:

String[] catsNames = {

"Васька",

"Кузя",

"Барсик",

"Мурзик", "Леопольд", "Бегемот", "Рыжик", "Матроскин"

};

// Сортируем перед передачей адаптеру

Arrays.sort(catsNames);

ArrayAdapter<String> adapter;

adapter = new ArrayAdapter<>( this,

android.R.layout.simple\_list\_item\_1, catsNames);

setListAdapter(adapter);

У метода **sort()** есть перегруженные версии, где можно указать диапазон массива, в пределах которого следует произвести сортировку.

**Копирование массивов**

Метод **Arrays.copyOf(оригинальный\_массив, новая\_длина)** — возвращает массив-копию новой длины. Если новая длина меньше оригинальной, то массив усекается до этой длины, а если больше, то дополняется нулями.

// первый массив

int[] anyNumbers = {2, 8, 11};

// копия второго массива

int[] luckyNumbers = Arrays.copyOf(anyNumbers, anyNumbers.length); luckyNumbers[2] = 25;

mInfoTextView.setText("anyNumbers: " + Arrays.toString(anyNumbers)

+ "\nluckyNumbers: " + Arrays.toString(luckyNumbers));

Теперь первый массив останется без изменений, а со вторым массивом делайте что хотите. Смотрим на результат.

anyNumbers: [2, 8, 11];

luckyNumbers: [2, 8, 25];

Можно создать увеличенную копию, когда копируются все значения из маленького массива, а оставшиеся места заполняются начальными значениями, например, нулями.

// три элемента

int[] small\_array = {1, 2, 3};

// создадим массив с пятью элементами

int[] big\_array = Arrays.copyOf(small\_array, 5); mInfoTextView.setText("big\_array: " + Arrays.toString(big\_array));

Получим результат:

big\_array: [1, 2, 3, 0, 0]

Метод **Arrays.copyOfRange(оригинальный\_массив, начальный\_индекс, конечный\_индекс)** — также возвращает массив-копию новой длины, при этом копируется часть оригинального массива от начального индекса до конечного –1.

// Массив из четырех элементов String[] array\_1 = {

"Васька",

"Мурзик",

"Рыжик",

"Барсик"};

// Сортировка массива

Arrays.sort(array\_1);

// Копируем первые три элемента массива во второй массив String[] array\_2 = Arrays.copyOf(array\_1, 3);

// Копируем нужные элементы из первого массива

// в диапазоне от второго элемента до последнего в третий массив String[] array\_3 = Arrays.copyOfRange(array\_1,

2, array\_1.length);

Log.i(TAG, Arrays.toString(array\_1)); Log.i(TAG, Arrays.toString(array\_2)); Log.i(TAG, Arrays.toString(array\_3));

**Метод Arrays.toString()**

Если использовать вызов метода **toString()** непосредственно у массива, то получите что-то непонятное и нечитаемое.

String[] catNames = {

"Васька",

"Мурзик",

"Рыжик",

"Барсик"};

Log.i(TAG, catNames.toString());

// Вернѐт [Ljava.lang.String;@4222bd88

Метод **Arrays.toString(массив)** возвращает строковое представление массива со строковым представлением элементов, заключенных в квадратные скобки. В примерах выше мы уже вызывали данный метод.

Метод **deepToString()** удобен для вывода многомерных массивов. Этот метод мы также уже использовали выше.

**Метод Arrays.fill() - наполнение массива одинаковыми данными**

Метод **Arrays.fill()** позволяет быстро заполнить массив одинаковыми значениями. У метода есть восемнадцать перегруженных версий для разных типов и объектов.

Метод **fill()** просто дублирует одно заданное значение в каждом элементе массива (в случае объектов копирует одну ссылку в каждый элемент):

int size = 4;

boolean[] test1 = new boolean[size]; int[] test2 = new int[size]; String[] test3 = new String[size];

Arrays.fill(test1, true); // присваивем всем true Toast.makeText(getApplicationContext(), Arrays.toString(test1),

Toast.LENGTH\_LONG).show(); Arrays.fill(test2, 9); // присваиваем всем 9

Toast.makeText(getApplicationContext(), Arrays.toString(test2), Toast.LENGTH\_LONG).show();

Arrays.fill(test3, "Мяу!"); // Ну вы поняли Toast.makeText(getApplicationContext(), Arrays.toString(test3),

Toast.LENGTH\_LONG).show();

Запустив код, вы увидите, что на экране по очереди выводятся значения:

[true, true, true, true] [9, 9, 9, 9]

[Мяу!, Мяу!, Мяу!, Мяу!]

Можно заполнить данными в нужном интервале за два прохода:

int size = 4;

String[] test3 = new String[size]; Arrays.fill(test3, "Мяу! ");

Arrays.fill(test3, 2, 3, "Гав!"); Toast.makeText(getApplicationContext(), Arrays.toString(test3),

Toast.LENGTH\_LONG).show();

Сначала массив заполнится мяуканьем кота 4 раза, а потом на третью позицию попадает слово **Гав!**:

[Мяу!, Мяу!, Гав!, Мяу!]

Как видите, метод заполняет весь массив, либо диапазон его элементов. Но получаемые одинаковые данные не слишком интересны для опытов, но пригодятся для быстрых экспериментов.

**Метод equals() - сравнение массивов**

Класс **Arrays** содержит метод **equals()** для проверки на равенство целых массивов. Чтобы два массива считались равными, они должны содержать одинаковое количество элементов, и каждый элемент должен быть эквивалентен соответствующему элементу другого массива.

Напишем код в своей учебной программе.

// Создаем два массива int[] a1 = new int[10]; int[] a2 = new int[10];

// заполняем их девятками Arrays.fill(a1, 9);

Arrays.fill(a2, 9);

mInfoTextView.setText("Сравним: " + Arrays.equals(a1, a2));

Мы создали два массива и заполнили их одинаковыми числами. При сравнении мы получим **true**. Добавим в код строчку кода, которая заменит один элемент во втором массиве:

//Изменим один элемент у второго массива a2[3] = 11;

mInfoTextView.setText("Сравним: " + Arrays.equals(a1, a2));

Теперь при сравнении будет выдаваться **false**.