Nucleotide 8 мая 2019 в 21:15

Практические задачи по Java — для курсов и прочих занятий

Java *

Практические задачи по Java — для курсов и прочих занятий

Несколько вводных слов

Последние несколько лет я читаю курс по программированию на Java. Со временем он менялся — то добавлялись, то выкидывались разные части, менялась последовательность тем, менялся подход к построению плана самих занятий, и так далее. То есть, курс совершенствовался. Одной из основных проблем, возникших при подготовке курса — это задачи. О них и пойдёт речь.

Дело в том, что каждое моё занятие состоит из двух частей. На первой я выступаю в роли лектора — рассказываю с примерами кода о какой-то новой теме (классы, наследование, дженерики и так далее). Вторая часть — практическая. Очевидно, что нет смысла просто рассуждать о программировании, надо программировать. Приоритет на занятиях — решение задач, то есть программирование чего-то как-то. Программирование на занятиях отличается от программирования дома, так как на занятиях можно задать вопрос, показать код, получить быструю оценку кода, комментарии по улучшению, исправлению написанного. Очень легко было найти задачи для самых первых занятий. Задачи на циклы, условные операторы, и ООП (к примеру, написать класс «Собака» или класс «Вектор»). Сервисы вроде leetcode позволяют даже проверить правильность решения таких задач сразу, онлайн. Но какие задачи дать студентам на занятии, которое было посвящено коллекциям? Потокам? А аннотациям? За несколько лет я придумал, или переработал несколько таких задач, и эта статья, по сути, является сборником этих задач (к некоторым задачам прилагается решение).

Конечно, все задачи уже где-то появлялись. Однако, эта статья ориентирована на преподавателей курсов по программированию (для языков, похожих на Java, большинство задач подойдёт), или тех, кто преподаёт программирование частным образом. Эти задачи можно использовать «из коробки» на своих занятиях. Изучающие Java тоже могут попробовать решать их. Но такие решения требуют сторонней проверки и оценки.

Некоторые самые простые задачи, которые уже десятилетия все используют, я тоже включил в эту статью. Пожалуй, для того, чтобы не начинать сразу с абстрактных классов.

Любые идеи и пожелания приветствуются!

Список задач

Основы

1.0. Максимальное, минимальное и среднее значение
1.1 Сортировка массива
1.2 Поиск простых чисел
1.3 Удаление из массива

Основы ООП

2.0 Проектирование и создание класса, описывающего вектор
2.1 Генерация случайного элемента с весом
2.2 Связный список

Рекурсия

3.0 Двоичный поиск
3.1 Найти корень уравнения
3.2 Бинарное дерево поиска

Наследование

4.0 Реализовать иерархию классов, описывающую трёхмерные фигуры
4.1 Реализовать иерархию классов, описывающую трёхмерные фигуры — 2
4.2 Реализовать иерархию классов, описывающую трёхмерные фигуры — 3
4.3 Реализовать иерархию классов, описывающую трёхмерные фигуры — 4

Строки

5.0 Частотный словарь букв

Абстрактные классы и интерфейсы

6.0. Конвертер температур
6.1. Stringbuilder с поддержкой операции undo
6.2. Stringbuilder с возможностью отслеживания состояния (паттерн наблюдатель)
6.4. Заполнение массива с помощью **Function**

Коллекции

7.0. Частотный словарь слов
7.1. Коллекция без дубликатов
7.2. ArrayList и LinkedList
7.3. Написать итератор по массиву
7.4. Написать итератор по двумерному массиву
7.5. Ещё более сложный итератор
7.6. Итератор по двум итераторам
7.7. Подсчёт элементов
7.8. Поменять ключи и значения в Map

Многопоточность

8.0. Состояния
8.1. Синхронизация потоков
8.2. Производитель-потребитель

Аннотации

9.0. Своя аннотация — создание и использование с помощью *reflection*

Итоговые и прочие задания

10.0. Количество дорожных ограничений
10.1. Поиск по Википедии. В консольной программе
10.2. Итоговое задание — консольная утилита для скачивания файлов по HTTP
10.3. Итоговое задание — погодный Telegram-бот
10.4. Итоговое задание — распознавание рукописных цифр

Основы

1.0. Максимальное, минимальное и среднее значение

Задача:

Заполните массив случайным числами и выведите максимальное, минимальное и среднее значение.

Для генерации случайного числа используйте метод Math.random(), который возвращает значение в промежутке [0, 1].

Решение:

```
public static void main(String[] args) {

        int n = 100;
        double[] array = new double[n];
        for (int i = 0; i < array.length; i++) {
            array[i] = Math.random();
        }

        double max = array[0]; // Массив не должен быть пустым
        double min = array[0];
        double avg = 0;
        for (int i = 0; i < array.length; i++) {
            if(max < array[i])
                max = array[i];
            if(min > array[i])
                min = array[i];
            avg += array[i]/array.length;
        }

        System.out.println("max = " + max);
        System.out.println("min = " + min);
        System.out.println("avg = " + avg);
    }
```

1.1. Реализуйте алгоритм сортировки пузырьком для сортировки массива

Решение:

```
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array.length - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    double temp = array[j];
```

```
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }

    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i]);
    }
```

1.2. Поиск простых чисел

Задача:

Напишите программу, которая выводит на консоль простые числа в промежутке от [2, 100]. Используйте для решения этой задачи оператор "**%**" (остаток от деления) и циклы.

Решение:

```
    for(int i = 2; i <= 100; i ++){
        boolean isPrime = true;

        for(int j = 2; j < i; j++){
            if(i % j == 0){
                isPrime = false;
                break;
            }
        }

        if(isPrime){
            System.out.println(i);
        }
    }
```

Или, используя циклы с метками:

```
    out_for:
    for (int i = 2; i <= 100; i++) {
        for (int j = 2; j < i; j++) {
            if (i % j == 0) {
                continue out_for;
            }
        }
        System.out.println(i);
    }
```

1.3. Удаление из массива

Задача:

Дан массив целых чисел и ещё одно целое число. Удалите все вхождения этого числа из массива

(пропусков быть не должно).

Решение:

```
    public static void main(String[] args) {
        int test_array[] = {0,1,2,2,3,0,4,2};
        /*
            Arrays.toString:
            см. https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
         */
        System.out.println(Arrays.toString(removeElement(test_array, 3)));
    }

    public static int[] removeElement(int[] nums, int val) {
        int offset = 0;

        for(int i = 0; i< nums.length; i++){
            if(nums[i] == val){
                offset++;
            } else{
                nums[i - offset] = nums[i];
            }
        }

        // Arrays.copyOf копирует значение из массива nums в новый массив
        // с длинной nums.length - offset
        return Arrays.copyOf(nums, nums.length - offset);
    }
```

Можно написать метод для «отрезания хвоста» массива и самостоятельно, но стоит отметить, что стандартный метод будет работать быстрее:

```
public static int[] removeElement(int[] nums, int val) {
        int offset = 0;

        for(int i = 0; i< nums.length; i++){
            if(nums[i] == val){
                offset++;
            } else{
                nums[i - offset] = nums[i];
            }
        }

        int[] newArray = new int[nums.length - offset];
        for(int i = 0; i < newArray.length; i++){
            newArray[i] = nums[i];
        }
        return newArray;
    }
```

Впрочем, если идти таким путём, то можно сначала создать массив нужной длины, а потом уже заполнить его:

```java
public static int[] removeElement(int[] nums, int val) {
        int count = 0;

        // Сначала вычислим длину нового массива
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != val) {
                count++;
            }
        }

        int[] newArray = new int[count];
        int offset = 0;

        // Далее всё как в прошлых решениях,
        // только запись идёт в новый массив
        for(int i = 0; i< nums.length; i++){
            if(nums[i] == val){
                offset++;
            } else{
                newArray[i - offset] = nums[i];
            }
        }

        return newArray;
    }
```

2.0. Проектирование и создание класса, описывающего вектор

Задача:

Создайте класс, который описывает вектор (в трёхмерном пространстве).

У него должны быть:

- конструктор с параметрами в виде списка координат *x, y, z*
- метод, вычисляющий длину вектора. Корень можно посчитать с помощью Math.sqrt():

$$\sqrt{x^2 + y^2 + z^2}$$

- метод, вычисляющий скалярное произведение:

$$x_1x_2 + y_1y_2 + z_1z_2$$

- метод, вычисляющий векторное произведение с другим вектором:

$$(y_1z_2 - z_1y_2, z_1x_2 - x_1z_2, x_1y_2 - y_1x_2)$$

- метод, вычисляющий угол между векторами (или косинус угла): косинус угла между векторами равен скалярному произведению векторов, деленному на произведение модулей (длин) векторов:

$$\frac{(a, b)}{|a| \cdot |b|}$$

- методы для суммы и разности:

$$(x_1 + x_2, y_1 + y_2, z_1 + z_2)$$
$$(x_1 - x_2, y_1 - y_2, z_1 - z_2)$$

- статический метод, который принимает целое число N, и возвращает массив случайных векторов размером N.

Если метод возвращает вектор, то он должен возвращать новый объект, а не менять базовый. То есть, нужно реализовать шаблон "Неизменяемый объект"

Решение:

```
public class Vector {
    // Три приватных поля
    private double x;
    private double y;
    private double z;

    // С тремя параметрами
    public Vector(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    // Длина вектора. Корень из суммы квадратов
    public double length() {
        return Math.sqrt(x * x + y * y + z * z);
    }

    // метод, вычисляющий скалярное произведение
    public double scalarProduct(Vector vector) {
        return x * vector.x + y * vector.y + z * vector.z;
    }

    // метод, вычисляющий векторное произведение
    public Vector crossProduct(Vector vector) {
        return new Vector(
                y * vector.z - z * vector.y,
                z * vector.x - x * vector.z,
                x * vector.y - y * vector.x);
    }

    // Косинус между двумя векторами
    public double cos(Vector vector) {
        // Для вычисления длины и произведения используются
        //методы multiply и length
        return scalarProduct(vector) / (length() * vector.length());
    }

    public Vector add(Vector vector) {
        return new Vector(
                x + vector.x,
                y + vector.y,
                z + vector.z
```

```
        );
    }

    public Vector subtract(Vector vector) {
        return new Vector(
                x - vector.x,
                y - vector.y,
                z - vector.z
        );
    }

    public static Vector[] generate(int n){
        Vector[] vectors = new Vector[n];
        for(int i =0; i < n; i++){
            vectors[i] = new Vector(Math.random(), Math.random(), Math.random());
        }
        return vectors;
    }

    @Override
    public String toString() {
        return "Vector{" +
                "x=" + x +
                ", y=" + y +
                ", z=" + z +
                '}';
    }
}
```

Использовать этот класс можно так:

```
public static void main(String[] args) {
        Vector[] vectors = Vector.generate(10);
        System.out.println(vectors[0]);
        System.out.println(vectors[1]);
        System.out.println(vectors[0].length());
        System.out.println(vectors[0].scalarProduct(vectors[1]));
        System.out.println(vectors[0].crossProduct(vectors[1]));
        System.out.println(vectors[0].cos(vectors[1]));
        System.out.println(vectors[0].add(vectors[1]));
        System.out.println(vectors[0].subtract(vectors[1]));
    }
```

Это решение можно обобщить и написать класс Vector для произвольной размерности:

```
public class Vector {

    // теперь не три координаты, а массив координат
    private double values[];

    public Vector(double[] values) {
        this.values = values;
    }
```

```java
// Длина вектора. Корень из суммы квадратов
public double length() {
    double sum = 0;
    for (int i = 0; i < values.length; i++) {
        sum += values[i] * values[i];
    }
    return Math.sqrt(sum);
}

// метод, вычисляющий скалярное произведение
public double scalarProduct(Vector vector) {
    double res = 0;

    for (int i = 0; i < values.length; i++) {
        res += values[i] * vector.values[i];
    }
    return res;
}

// для многомерных не определено
//    public double crossProduct(Vector vector) {
//
//    }

// Косинус между двумя векторами
public double cos(Vector vector) {
    return scalarProduct(vector) / (length() * vector.length());
}

public Vector add(Vector vector) {
    double[] another = new double[values.length];

    for (int i = 0; i < values.length; i++) {
        another[i] = values[i] + vector.values[i];
    }
    return new Vector(another);
}

public Vector subtract(Vector vector) {
    double[] another = new double[values.length];

    for (int i = 0; i < values.length; i++) {
        another[i] = values[i] - vector.values[i];
    }
    return new Vector(another);
}

// Вспомогательный метод
private static double[] generateRandomArray(int length) {
    double[] array = new double[length];
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.random();
    }
    return array;
}

public static Vector[] generate(int n, int dimension) {
    Vector[] vectors = new Vector[n];
```

```
        for (int i = 0; i < n; i++) {
            vectors[i] = new Vector(generateRandomArray(dimension));
        }
        return vectors;
    }
}
```

2.1. Генерация случайного элемента с весом

Задача:

Напишите класс, конструктор которого принимает два массива: массив значений и массив весов значений.
Класс должен содержать метод, который будет возвращать элемент из первого массива случайным образом, с учётом его веса.
Пример:
Дан массив [1, 2, 3], и массив весов [1, 2, 10].
В среднем, значение **«1»** должно возвращаться в 2 раза реже, чем значение **«2»** и в десять раз реже, чем значение **«3»**.

Решение:

```
/*
    Решение основывается на геометрической идее:
    Будем считать, что веса — это длины некоторых отрезков.
    Тогда надо "уложить" все отрезки в один общий,
    генерировать случайное значение из этого общего отрезка,
    определять в какой из наших отрезков попало значение:
    |-|--|----------|
    0-1--3----------13
              ^

 */
class RandomFromArray {
    private int[] values; // значения
    private int[] weights; // веса
    private int[] ranges; // левые границы отрезков
    private int sum; // общая длина всех отрезков

    public RandomFromArray(int[] values, int[] weights) {
        this.values = values;
        this.weights = weights;
        ranges = new int[values.length];

        // Сумма длин всех отрезков
        sum = 0;
        for (int weight : weights) {
            sum += weight;
        }

        // Заполняем ranges, левыми границами
        int lastSum = 0;
        for (int i = 0; i < ranges.length; i++) {
            ranges[i] = lastSum;
```

```
            lastSum += weights[i];
        }
    }

    /*
        Массив ranges уже заполнен, так что остаётся
        сгенерировать значение в промежутке [0;sum],
        и найти отрезок, содержащий это значение:
     */
    public int getRandom() {
        int random = (int) (Math.random() * (sum - 1));

        int ourRangeIndex = 0;
        for (int i = 0; i < ranges.length; i++) {
            if (ranges[i] > random) {
                break;
            }
            ourRangeIndex = i;
        }

        return values[ourRangeIndex];
    }
}
```

Но, так как массив ***ranges*** отсортирован, то можно (и нужно) использовать бинарный поиск:

```
public int getRandom() {
        int random = (int) (Math.random() * (sum - 1));

        int index = Arrays.binarySearch(ranges, random);
        return values[index >= 0 ? index : -index - 2];
    }
```

Есть ещё один вариант решения этой задачи. Можно создать массив, размер которого равен сумме всех весов. Тогда выбор случайного элемента сводится к генерации случайного индекса. То есть, если дан массив значений [1, 2, 3], и массив весов [1, 2, 10], то можно сразу создать массив [1, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3] и извлекать из него случайный элемент:

```
class RandomFromArray {
    private int[] extended_values; // значения

    public RandomFromArray(int[] values, int[] weights) {
        // Сумма длин всех отрезков
        int sum = 0;
        for (int weight : weights) {
            sum += weight;
        }

        extended_values = new int[sum];
        int cursor = 0;

        for(int i = 0; i < weights.length; i++){
            for(int j = 0; j < weights[i]; j++){
                extended_values[cursor++] = values[i];
```

```
            }
        }
    }

    /*
        Массив extended_values уже заполнен,
        так что остаётся сгенерировать значение в промежутке
        [0; extended_values.length)
     */
    public int getRandom() {
        int random = (int) (Math.random() * ( extended_values.length - 1));
        return extended_values[random];
    }
}
```

У этого решения есть преимущество — время извлечения случайного элемента *O(1)*, в отличии от log(n) в предыдущем решении. Однако требует много памяти:

$$O(\sum n)$$

2.2. Связный список

Еще одна задача, которую я часто даю — реализация связного списка. Её можно давать в самом простом виде (реализовать только *add()* и *get()*), а можно попросить реализовать java.util.List.
Я не буду подробно на этом останавливаться, много статей о реализации связного списка на Java есть на Хабре, к примеру эта:
Структуры данных в картинках. LinkedList

3.0. Двоичный поиск

Задача:

Напишите метод, который проверяет, входит ли в массив заданный элемент или нет.
Используйте перебор и двоичный поиск для решения этой задачи.
Сравните время выполнения обоих решений для больших массивов (например, 100000000 элементов).
Решение:

```
    /*
        Просто перебираем, пока не найдет.
        Если ничего не найдём, вернём -1
     */
    public static int bruteForce(double[] array, double key) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == key)
                return i;
        }
        return -1;
    }

    /*
        Двоичный поиск
```

```
 */
public static int binarySearchRecursively(double[] sortedArray, double key) {
    return binarySearchRecursively(sortedArray, key, 0, sortedArray.length);
}

/**
 * Вспомогательный метод для {@link #binarySearchRecursively(double[], double)}
 *
 * Будем делить отрезок пополам, но не копировать, а просто "сдвигать границы",
 * и вызывать этот же метод рекурсивно. Для этого используем low и high
 *
 * @param sortedArray сортированный массив
 * @param key искомое значение
 * @param low от какого значения ищем
 * @param high до какого значения ищем
 * @return индекс элемента
 */
private static int binarySearchRecursively
                (double[] sortedArray, double key, int low, int high) {
    int middle = (low + high) / 2; // середина

    if (high < low) { // больше делить нечего
        return -1;
    }

    if (key == sortedArray[middle]) { // если нашёлся
        return middle;
    } else if (key < sortedArray[middle]) { // ищем в левой половине
        return binarySearchRecursively(
                sortedArray, key, low, middle - 1);
    } else {
        return binarySearchRecursively( // ищем в правой половине
                sortedArray, key, middle + 1, high);
    }
}

// Вспомогательный метод для тестов
private static double[] generateRandomArray(int length) {
    double[] array = new double[length];
    for (int i = 0; i < array.length; i++) {
        array[i] = Math.random();
    }
    return array;
}

public static void main(String[] args) {
    double[] array = generateRandomArray(100000000);
    Arrays.sort(array); // нужно сначала отсортировать

    /*
        Строго говоря,
        измерять время выполнения так не совсем корректно,
        лучше использовать benchmarks
        см. https://habr.com/ru/post/349914/
        Но масштаб будет понятен
     */
    long time = System.currentTimeMillis(); // текущее время, unix-time
    bruteForce(array, 0.5);
```

```
        System.out.println(System.currentTimeMillis() - time);

        time = System.currentTimeMillis();
        binarySearchRecursively(array, 0.5);
        System.out.println(System.currentTimeMillis() - time);
    }
```

3.1. Найти корень уравнения

Задача:

Найдите корень уравнения

$$cos(x^5) + x^4 - 345.3 * x - 23 = 0$$

на отрезке [0; 10] с точностью по *x* не хуже, чем 0.001. Известно, что на этом промежутке корень единственный.
Используйте для этого метод деления отрезка пополам (и рекурсию).

Решение:

```
    // вспомогательный метод
    public static double func(double x){
        return Math.cos(Math.pow(x, 5)) + Math.pow(x, 4) - 345.3 * x - 23;
    }

    // решить уравнение
    public static double  solve(double start, double end){
        if(end - start <= 0.001){
            return start;
        }

        double x = start + (end - start) / 2;

        if(func(start) * func(x) > 0){
            return solve(x, end);
        } else {
            return solve(start, x);
        }
    }

    public static void main(String[] args) {
        System.out.println(solve(0, 10));
    }
```

Замечание: если мы хотим добиться нужной точности не по *x*, по *y*, то условие в методе следует переписать:

```
        if(Math.abs(func(start)- func(end)) <= 0.001){
            return start;
        }
```

Маленькая хитрость: учитывая, что множество значений double конечно (есть два соседних

значения, между которыми нет ни одного значения double), условие выхода из рекурсии переписать так:

```
double x = start + (end - start) / 2;
if(x == end || x == start){
    return x;
}
```

Таким образом получим максимальную точность, которую вообще можно получить, используя этот подход.

3.2. Бинарное дерево поиска

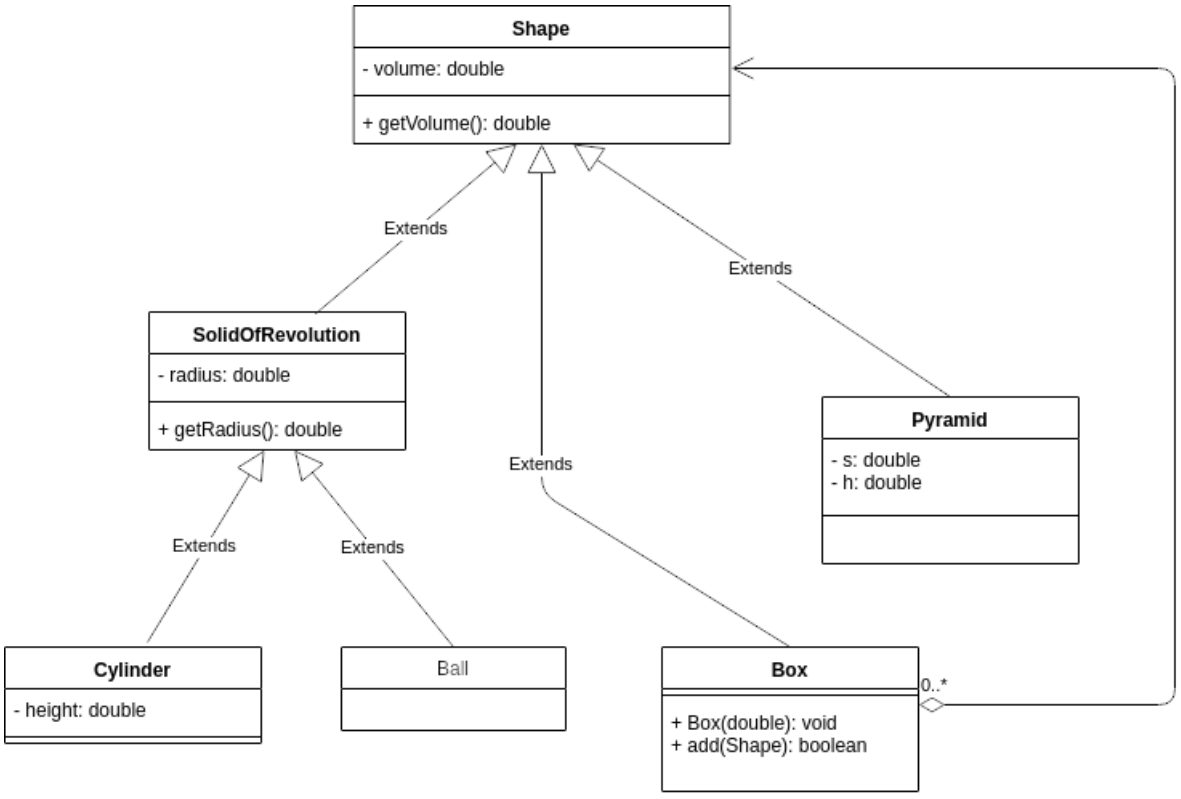
Реализация бинарного дерева поиска — отличная задача. Я обычно даю её, когда заходит разговор про рекурсию.
Об этом я тоже много писать не буду, есть много статей/реализаций самого разного вида:
Структуры данных: бинарные деревья.
Бинарное дерево, быстрая реализация
Реализация на Java хешированного бинарного дерева

Наследование

4.0. Реализовать иерархию классов, описывающую трёхмерные фигуры

Задача:

Реализуйте иерархию классов:



Класс **Box** является контейнером, он можем содержать в себе другие фигуры. Метод *add()* принимает на вход Shape. Нужно добавлять новые фигуры до тех пор, пока для них хватаем места в Box (будем считать только объём, игнорируя форму. Допустим, мы переливаем жидкость). Если места для добавления новой фигуры не хватает, то метод должен вернуть **false**.

Решение:

```
class Shape {
    private double volume;

    public Shape(double volume) {
        this.volume = volume;
    }

    public double getVolume() {
        return volume;
    }
}

class SolidOfRevolution extends Shape {
    private double radius;

    public SolidOfRevolution(double volume, double radius) {
        super(volume);
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }
}

class Ball extends SolidOfRevolution { // конкретный класс
    public Ball(double radius) {
        super(Math.PI * Math.pow(radius, 3) * 4 / 3, radius);
    }
}

class Cylinder extends SolidOfRevolution { // конкретный класс
    private double height;

    public Cylinder(double radius, double height) {
        super(Math.PI * radius * radius * height, radius);
        this.height = height;
    }
}

class Pyramid extends Shape{
    private double height;
    private double s; // площадь основания

    public Pyramid(double height, double s) {
        super(height * s * 4 / 3);
        this.height = height;
        this.s = s;
    }
}


class Box extends Shape {
    private ArrayList<Shape> shapes = new ArrayList<>();
```

```
    private double available;

    public Box(double available) {
        super(available);
        this.available = available;
    }

    public boolean add(Shape shape) {
        if (available >= shape.getVolume()) {
            shapes.add(shape);
            available -= shape.getVolume();
            return true;
        } else {
            return false;
        }
    }
}

public class Main {

    public static void main(String[] args) {
        Ball ball = new Ball(4.5);
        Cylinder cylyinder = new Cylinder(2, 2);
        Pyramid pyramid = new Pyramid(100, 100);

        Box box = new Box(1000);

        System.out.println(box.add(ball)); // ok
        System.out.println(box.add(cylyinder)); // ok
        System.out.println(box.add(pyramid)); // failed
    }
}
```

Чтобы к этой задаче не возвращаться, далее описывается еще несколько вариаций этой задачи.

4.1. Реализовать иерархию классов, описывающую трёхмерные фигуры — 2

Задача:

Реализуйте ту же иерархию классов, но сделав некоторые классы абстрактными.

Решение:

```
abstract class Shape {
    public abstract double getVolume();
}

abstract class SolidOfRevolution extends Shape {
    protected double radius;

    public SolidOfRevolution(double radius) {
        this.radius = radius;
    }

    public double getRadius() {
```

```
        return radius;
    }
}

class Ball extends SolidOfRevolution { // конкретный класс

    @Override
    public double getVolume() {
        return Math.PI * Math.pow(radius, 3) * 4 / 3;
    }

    public Ball(double radius) {
        super(radius);
    }
}

class Cylinder extends SolidOfRevolution { // конкретный класс
    private double height;

    public Cylinder(double radius, double height) {
        super(radius);
        this.height = height;
    }

    @Override
    public double getVolume() {
        return Math.PI * radius * radius * height;
    }
}

class Pyramid extends Shape {
    private double height;
    private double s; // площадь основания

    public Pyramid(double height, double s) {
        this.height = height;
        this.s = s;
    }

    @Override
    public double getVolume() {
        return height * s * 4 / 3;
    }
}


class Box extends Shape {
    private ArrayList<Shape> shapes = new ArrayList<>();
    private double available;
    private double volume;

    public Box(double available) {
        this.available = available;
        this.volume = available;
    }

    public boolean add(Shape shape) {
        if (available >= shape.getVolume()) {
```

```
            shapes.add(shape);
            available -= shape.getVolume();
            return true;
        } else {
            return false;
        }
    }

    @Override
    public double getVolume() {
        return volume;
    }
}

public class Main {

    public static void main(String[] args) {
        Ball ball = new Ball(4.5);
        Cylinder cylyinder = new Cylinder(2, 2);
        Pyramid pyramid = new Pyramid(100, 100);

        Box box = new Box(1000);

        System.out.println(box.add(ball)); // ok
        System.out.println(box.add(cylyinder)); // ok
        System.out.println(box.add(pyramid)); // failed

    }
}
```

4.2. Реализовать иерархию классов, описывающую трёхмерные фигуры — 3

Задача:

Реализуйте ту же иерархию классов, но использовав интерфейсы.
Дополнительно, студентам предлагается реализовать интерфейс **Comparable**.

Решение:

```
interface Shape extends Comparable<Shape>{
    double getVolume();

    @Override
    default int compareTo(Shape other) {
        return Double.compare(getVolume(), other.getVolume());
    }
}

abstract class SolidOfRevolution implements Shape {
    protected double radius;

    public SolidOfRevolution(double radius) {
        this.radius = radius;
    }
```

```
    public double getRadius() {
        return radius;
    }
}

class Ball extends SolidOfRevolution { // конкретный класс

    @Override
    public double getVolume() {
        return Math.PI * Math.pow(radius, 3) * 4 / 3;
    }

    public Ball(double radius) {
        super(radius);
    }
}

class Cylinder extends SolidOfRevolution { // конкретный класс

    private double height;

    public Cylinder(double radius, double height) {
        super(radius);
        this.height = height;
    }

    @Override
    public double getVolume() {
        return Math.PI * radius * radius * height;
    }
}

class Pyramid implements Shape {
    private double height;
    private double s; // площадь основания

    public Pyramid(double height, double s) {
        this.height = height;
        this.s = s;
    }

    @Override
    public double getVolume() {
        return height * s * 4 / 3;
    }
}


class Box implements Shape {
    private ArrayList<Shape> shapes = new ArrayList<>();
    private double available;
    private double volume;

    public Box(double available) {
        this.available = available;
        this.volume = available;
    }
```

```
    public boolean add(Shape shape) {
        if (available >= shape.getVolume()) {
            shapes.add(shape);
            available -= shape.getVolume();
            return true;
        } else {
            return false;
        }
    }

    @Override
    public double getVolume() {
        return volume;
    }

    public ArrayList<Shape> getShapes() {
        return shapes;
    }
}

public class Main {

    public static void main(String[] args) {
        Ball ball = new Ball(4.5);
        Cylinder cylyinder = new Cylinder(2, 2);
        Pyramid pyramid = new Pyramid(100, 100);

        Box box = new Box(1000);

        System.out.println(box.add(ball)); // ok
        System.out.println(box.add(cylyinder)); // ok
        System.out.println(box.add(pyramid)); // failed

        // Sorting:
        ArrayList<Shape> shapes = box.getShapes();
        Collections.sort(shapes); // sorted by Volume!
    }
}
```

4.3. Реализовать иерархию классов, описывающую трёхмерные фигуры — 4

Задача:

Добавьте в иерархию классов фигуру вращения для произвольной функции. Вычислять объём можно приближенное с помощью определённого интеграла. Так как объём фигуры вращения вокруг оси *x* это

$$V_x = \pi \int_a^b f^2(x)dx$$

А интеграл это

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{i=0}^{n-1} f(\xi_i)\Delta x_i$$

То можно написать реализацию метода прямоугольников:

```
class SolidRevolutionForFunction extends SolidOfRevolution {
    private Function<Double, Double> function;
    private double a;
    private double b;

    public SolidRevolutionForFunction(
                          Function<Double, Double> function, double a, double b) {
        super(b - a);
        this.function = function;
        this.a = a;
        this.b = b;
    }

    @Override
    public double getVolume() {
        double sum = 0;
        int iterations = 10000;
        double delta = (b - a)/iterations;
        for(int i = 0; i < iterations; i++){
            double x = a + ((b - a) * i/iterations);
            sum += Math.pow(function.apply(x), 2) * delta;
        }
        return Math.PI * sum;
    }
}
```

```
public static void main(String[] args) {
        Shape shape = new SolidRevolutionForFunction(new Function<Double, Double>() {
            @Override
            public Double apply(Double x) {
                return Math.cos(x);
            }
        }, 0, 10);
        System.out.println(shape.getVolume());
    }
```

Конечно, мы не учитываем точность вычислений здесь, и не подбираем количество разбиений для достижения необходимой точности, но это задача на программирование, а не численные методы, так что на занятиях мы это опускаем.

Строки

Можно найти очень много задач на строки. Я обычно даю такие:

- Напишите метод для поиска самой длинной строки в массиве.
- Напишите метод, который проверяет является ли слово палиндромом.

- Напишите метод, заменяющий в тексте все вхождения слова ~~Навальный~~ «бяка» на «[вырезано цензурой]».
- Имеются две строки. Напишите метод, возвращающий количество вхождений одной строки в другую.

Решения таких задач я описывать не буду, да и задач на строки тоже можно найти огромное количество.

Из более интересных мне нравится эта:

5.0. Частотный словарь букв русского (или английского) алфавита.

Задача:

Постройте частотный словарь букв русского (или английского) алфавита. Опустим проблему выбора и анализа корпуса языка, достаточно будет взять текст небольшой длины).

Решение:

```
/**
 * Будем перебирать все символы в строке, и
 * если это символ алфавита русского языка,
 * обновляем значение в Map.
 *
 * Потом возьмём все Map.Entry<Character, Integer>,
 * и отсортируем по ключу (Character)
 *
 * @param text - текст
 */
void buildDictionaryWithMap(String text){
    text = text.toLowerCase();

    Map<Character, Integer> map = new HashMap<>();
    for(int i = 0; i < text.length(); i++){
        char ch = text.charAt(i);

        // ё идёт отдельно от а-я
        if((ch >= 'а' && ch <= 'я') || ch == 'ё'){
            map.compute(ch, (character, integer)
                    -> integer == null ? 1 : integer + 1);
        }
    }

    ArrayList<Map.Entry<Character, Integer>> entries =
            new ArrayList<>(map.entrySet());
    entries.sort((o1, o2) -> Character.compare(o1.getKey(), o2.getKey()));
    for(Map.Entry<Character, Integer> entry : entries){
        System.out.println(entry.getKey() + " " + entry.getValue());
    }
}
```

Или так:

```
    /**
     * Вариант без Map.
     * Создадим массив нужной длины, и будем хранить
     * в соответствующих позициях количество вхождений
     * символов
     * @param text
     */
    void buildDictionary(String text){
        text = text.toLowerCase();

        int[] result = new int['я' - 'а' + 1];
        for(int i = 0; i < text.length(); i++){
            char ch = text.charAt(i);
            if(ch >= 'а' && ch <= 'я'){
                result[ch - 'а']++;
            }
        }

        for(int i = 0; i < result.length; i++){
            System.out.println((char) (i + 'а') + " = " + result[i]);
        }
    }
```

Абстрактные классы и интерфейсы

6.0. Конвертер температур

Задача:

Напишите класс BaseConverter для конвертации из градусов по Цельсию в Кельвины, Фаренгейты, и так далее. У класса должен быть метод *convert*, который и делает конвертацию.

Решение:

```
interface Converter {
    double getConvertedValue(double baseValue);
}

class CelsiusConverter implements Converter {
    @Override
    public double getConvertedValue(double baseValue) {
        return baseValue;
    }
}

class KelvinConverter implements Converter {

    @Override
    public double getConvertedValue(double baseValue) {
        return baseValue + 273.15;
    }
}
```

```
class FahrenheitConverter implements Converter {
    @Override
    public double getConvertedValue(double baseValue) {
        return 1.8 * baseValue + 32;
    }
}

public class Main {
    public static void main(String[] args) {
        double temperature = 23.5;
        System.out.println("t = " +
                new CelsiusConverter().getConvertedValue(temperature));
        System.out.println("t = " +
                new KelvinConverter().getConvertedValue(temperature));
        System.out.println("t = " +
                new FahrenheitConverter().getConvertedValue(temperature));
    }
}
```

Дополнительно можно попросить реализовать фабричный метод, как-то так:

```
interface Converter {
    double getConvertedValue(double baseValue);

    public static Converter getInstance(){
        Locale locale = Locale.getDefault();
        String[] fahrenheitCountries = {"BS", "US", "BZ", "KY", "PW"};

        boolean isFahrenheitCountry =
                Arrays.asList(fahrenheitCountries).contains(locale.getCountry());

        if(isFahrenheitCountry){
            return new FahrenheitConverter();
        } else {
            return new CelsiusConverter();
        }
    }
}
```

6.1. Stringbuilder с поддержкой операции *undo*

Задача:

Напишите свой класс StringBuilder с поддержкой операции *undo*. Для этого делегируйте все методы стандартному StringBuilder, а в собственном классе храните список всех операций для выполнения *undo()*. Это будет реализацией шаблона «Команда».

Решение:

```
/**
 * StringBuilder с поддержкой операции undo
 * java.lang.StringBuilder – класс с модификатором <b>final</b>,
 * поэтому нет наследования, используем делегирование.
```

```
 */
class UndoableStringBuilder {

    private interface Action{
        void undo();
    }

    private class DeleteAction implements Action{
        private int size;

        public DeleteAction(int size) {
            this.size = size;
        }

        public void undo(){
            stringBuilder.delete(
                stringBuilder.length() - size, stringBuilder.length());
        }
    }

    private StringBuilder stringBuilder; // делегат

    /**
     * операции, обратные к выполненным.
     * То есть при вызове append, в стек помещается
     * операция "delete". При вызове undo() она
     * будет выполнена.
     */
    private Stack<Action> actions = new Stack<>();

    // конструктор
    public UndoableStringBuilder() {
        stringBuilder = new StringBuilder();
    }

    /**
         see {@link java.lang.AbstractStringBuilder#reverse()}

        После того, как сделан reverse(),
        добавляем в стек операций обратную – тоже reverse().

        Далее таким же образом.
     */
    public UndoableStringBuilder reverse() {
        stringBuilder.reverse();

        Action action = new Action(){
            public void undo() {
                stringBuilder.reverse();
            }
        };

        actions.add(action);

        return this;
    }
```

```
public UndoableStringBuilder append(String str) {
    stringBuilder.append(str);

    Action action = new Action(){
        public void undo() {
            stringBuilder.delete(
                    stringBuilder.length() - str.length() -1,
                    stringBuilder.length());
        }
    };

    actions.add(action);
    return this;
}

// ..... остальные перегруженые методы append пишутся аналогично (см. выше)......

public UndoableStringBuilder appendCodePoint(int codePoint) {
    int lenghtBefore = stringBuilder.length();
    stringBuilder.appendCodePoint(codePoint);
    actions.add(new DeleteAction(stringBuilder.length() - lenghtBefore));
    return this;
}

public UndoableStringBuilder delete(int start, int end) {
    String deleted = stringBuilder.substring(start, end);
    stringBuilder.delete(start, end);
    actions.add(() -> stringBuilder.insert(start, deleted));
    return this;
}

public UndoableStringBuilder deleteCharAt(int index) {
    char deleted = stringBuilder.charAt(index);
    stringBuilder.deleteCharAt(index);
    actions.add(() -> stringBuilder.insert(index, deleted));
    return this;
}

public UndoableStringBuilder replace(int start, int end, String str) {
    String deleted = stringBuilder.substring(start, end);
    stringBuilder.replace(start, end, str);
    actions.add(() -> stringBuilder.replace(start, end, deleted));
    return this;
}

public UndoableStringBuilder insert(int index, char[] str, int offset, int len) {
    stringBuilder.insert(index, str, offset, len);
    actions.add(() -> stringBuilder.delete(index, len));
    return this;
}

public UndoableStringBuilder insert(int offset, String str) {
    stringBuilder.insert(offset, str);
    actions.add(() -> stringBuilder.delete(offset, str.length()));
    return this;
}

// ..... остальные перегруженные методы insert пишутся аналогично (см. выше)......
```

```
    public void undo(){
        if(!actions.isEmpty()){
            actions.pop().undo();
        }
    }

    public String toString() {
        return stringBuilder.toString();
    }
}
```

6.2. Stringbuilder с возможностью отслеживания состояния (паттерн наблюдатель)

Задача:

Напишите свой класс StringBuilder, с возможностью оповещения других объектов об изменении своего состояния. Для этого делегируйте все методы стандартному StringBuilder, а в собственном классе реализуйте шаблон проектирования «Наблюдатель».

Решение:

```
/**
    Слушатель.
    Каждый раз, когда меняется связанный с ним UndoableStringBuilder,
    вызывается метод onChange().
 */
interface OnStringBuilderChangeListener {
    void onChange(OvservableStringBuilder stringBuilder);
}

class OvservableStringBuilder {

    // Слушатель, которого надо будет уведомить
    private OnStringBuilderChangeListener onChangeListener;

    // делегат
    private StringBuilder stringBuilder;

    // Сеттер для onChangeListener
    public void setOnChangeListener(OnStringBuilderChangeListener onChangeListener) {
        this.onChangeListener = onChangeListener;
    }

    public OvservableStringBuilder() {
        stringBuilder = new StringBuilder();
    }

    private void notifyOnStringBuilderChangeListener(){
        if(onChangeListener != null){
            onChangeListener.onChange(this);
        }
    }

    public OvservableStringBuilder append(Object obj) {
```

```
        stringBuilder.append(obj);
        notifyOnStringBuilderChangeListener();
        return this;
    }

    public OvservableStringBuilder replace(int start, int end, String str) {
        stringBuilder.replace(start, end, str);
        notifyOnStringBuilderChangeListener();
        return this;
    }

    public OvservableStringBuilder insert(int index, char[] str, int offset, int len) {
        stringBuilder.insert(index, str, offset, len);
        notifyOnStringBuilderChangeListener();
        return this;
    }

    // ....... Другие методы аналогично ..........

    public String toString() {
        return stringBuilder.toString();
    }
}

/**
    Конкретная реализация OnStringBuilderChangeListener
 */
class MyListener implements OnStringBuilderChangeListener {
    /*
        Определяем метод onChange
        В него передаётся stringBuilder, который "прослушивается"
     */
    public void onChange(OvservableStringBuilder stringBuilder) {
        System.out.println("CHANGED: " + stringBuilder.toString());
    }
}

public class Main {
    public static void main(String[] strings) {
        OvservableStringBuilder UndoableStringBuilder =
                new OvservableStringBuilder();
        UndoableStringBuilder.setOnChangeListener(new MyListener());
        UndoableStringBuilder.append("Hello");
        UndoableStringBuilder.append(", ");
        UndoableStringBuilder.append("World!");
    }
}
```

6.3. Фильтр

Задача:

Напишите метод *filter*, который принимает на вход массив (любого типа) и реализацию интерфейса **Filter** с методом *apply(Object o)*, чтобы убрать из массива лишнее.
Проверьте как он работает на строках или других объектах.

Решение:
Обычно, я даю эту задачу еще до Generics, поэтому студенты пишут метод без них, используя Object:

```
interface Filter {
    boolean apply(Object o);
}

public class Main {

    public static Object[] filter(Object[] array, Filter filter) {
        int offset = 0;

        for(int i = 0; i< array.length; i++){
            if(!filter.apply(array[i])){
                offset++;
            } else{
                array[i - offset] = array[i];
            }
        }

        // Arrays.copyOf копирует значение из массива array в новый массив
        // с длинной array.length - offset
        return Arrays.copyOf(array, array.length - offset);
    }

    public static void main(String[] args) {
        String array[] =
                new String[]{"1rewf ", "feefewf", "a", null, "1"};

        String[] newArray =  (String[]) filter(array, new Filter() {
            @Override
            public boolean apply(Object o) {
                return o != null;
            }
        });
    }
}
```

Но, можно и с Generics. Тогда можно использовать стандартный Function:

```
public class Main {

    public static <T> T[] filter(T[] array, Function<? super T, Boolean> filter) {
        int offset = 0;

        for (int i = 0; i < array.length; i++) {
            if (!filter.apply(array[i])) {
                offset++;
            } else {
                array[i - offset] = array[i];
            }
        }

        // Arrays.copyOf копирует значение из массива array в новый массив
```

```
        // с длинной array.length - offset
        return Arrays.copyOf(array, array.length - offset);
    }


    public static void main(String[] args) {
        String array[] =
                new String[]{"1rewf ", "feefewf", "a", null, "1"};

        String[] newArray = filter(array, s -> s != null);
    }
}
```

6.4. Заполнение массива

Задача, немного похожая на предыдущую:
Напишите метод *fill*, который принимает массив объектов, и реализацию интерфейса Function (или своего).
Метод fill должен заполнить массив, получая новое значение по индексу с помощью реализации интерфейса Function. То есть, использовать его хочется так:

```
public static void main(String[] args) {
        Integer[] squares = new Integer[100];
        fill(squares, integer -> integer * integer); // 0, 1, 4, 9, 16 .....
    }
```

Решение:

```
public static <T> void fill(T[] objects, Function<Integer, ? extends T> function) {
        for(int i = 0; i < objects.length; i++){
            objects[i] = function.apply(i);
        }
}
```

Коллекции

7.0. Частотный словарь слов

см. Задачу про частотный словарь букв алфавита

7.1. Коллекция без дубликатов

Задача:

Напишите метод, который на вход получает коллекцию объектов, а возвращает коллекцию уже без дубликатов.

Решение:

```
    public static <T> Collection<T> removeDuplicates(Collection<T> collection) {
        return new HashSet<>(collection); // Вот и всё!
    }
```

7.2. ArrayList и LinkedList

Напишите метод, который добавляет 1000000 элементов в ArrayList и LinkedList. Напишите еще один метод, который выбирает из заполненного списка элемент наугад 100000 раз. Замерьте время, которое потрачено на это. Сравните результаты и предположите, почему они именно такие.

Решение:

```
    public static void compare2Lists() {
        ArrayList<Double> arrayList = new ArrayList<>();
        LinkedList<Double> linkedList = new LinkedList<>();
        final int N = 1000000;
        final int M = 1000;
        for (int i = 0; i < N; i++) {
            arrayList.add(Math.random());
            linkedList.add(Math.random());
        }
        long startTime = System.currentTimeMillis();
        for (int i = 0; i < M; i++) {
            arrayList.get((int) (Math.random() * (N - 1)));
        }
        System.out.println(System.currentTimeMillis() - startTime);

        startTime = System.currentTimeMillis();
        for (int i = 0; i < M; i++) {
            linkedList.get((int) (Math.random() * (N - 1)));
        }
        System.out.println(System.currentTimeMillis() - startTime);
    }
```

7.3. Написать итератор по массиву

Решение:

```
class ArrayIterator<T> implements Iterator<T>{

    private T[] array;
    private int index = 0;

    public ArrayIterator(T[] array) {
        this.array = array;
    }

    @Override
    public boolean hasNext() {
        return index < array.length;
    }

    @Override
```

```
    public T next() {
        if(!hasNext())
            throw new NoSuchElementException();
        return array[index++];
    }
}
```

7.4. Итератор по двумерному массиву

Задача:

Напишите итератор по двумерному массиву.

Решение:

```
class Array2d<T> implements Iterable<T>{
    private T[][] array;

    public Array2d(T[][] array) {
        this.array = array;
    }

    @Override
    public Iterator<T> iterator() {
        return new Iterator<T>() {

            private int i, j;

            @Override
            public boolean hasNext() {
                for(int i = this.i; i< array.length; i++){
                    for(int j = this.j; j< array[i].length; j++){
                        return true;
                    }
                }
                return false;
            }

            @Override
            public T next() {
                if(!hasNext())
                    throw new NoSuchElementException();
                T t = array[i][j];
                j++;
                for(int i = this.i; i< array.length; i++){
                    for(int j = (i == this.i ? this.j : 0); j< array[i].length; j++){
                        this.i = i;
                        this.j = j;
                        return t;
                    }
                }

                return t;
            }
        };
```

```
    }
}
```

7.5. Ещё более сложный итератор

Мне нравится эта задача. До неё доходит всего несколько студентов в группе, которые сравнительно легко справляются с предыдущими задачи.

Задача:

Дан итератор. Метод *next()* возвращает либо String, либо итератор такой же структуры (то есть который опять возвращает или String, или такой же итератор). Напишите поверх этого итератора другой, уже «плоский».

Решение на стеках:

```
public class DeepIterator implements Iterator<String> {

    private Stack<Iterator> iterators;
    private String next;
    private boolean hasNext;

    public DeepIterator(Iterator<?> iterator) {
        this.iterators = new Stack<Iterator>();
        iterators.push(iterator);

        updateNext();
    }

    @Override
    public boolean hasNext() {
        return hasNext;
    }

    private void updateNext(){

        if(iterators.empty()){
            next = null;
            hasNext = false;
            return;
        }

        Iterator current = iterators.peek();

        if (current.hasNext()) {
            Object o = current.next();
            if (o instanceof String) {
                next = (String) o;
                hasNext = true;
            } else if (o instanceof Iterator) {
                Iterator iterator = (Iterator) o;
                iterators.push(iterator);
                updateNext();
            } else {
```

```
                throw new IllegalArgumentException();
            }
        } else {
            iterators.pop();
            updateNext();
        }
    }

    @Override
    public String next() throws NoSuchElementException {

        if(!hasNext){
            throw new NoSuchElementException();
        }

        String nextToReturn = next;
        updateNext();
        return nextToReturn;
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Решение рекурсивное:

```
/**
 * @author Irina Poroshina
 */
class DeepIterator implements Iterator<String> {
    private Iterator subIter;
    private DeepIterator newIter;

    public DeepIterator(Iterator iniIter) {
        this.subIter = iniIter;
    }

    @Override
    public boolean hasNext() {
        if (subIter.hasNext()) return true;
        if (newIter != null) return newIter.hasNext();
        return false;

    }

    @Override
    public String next() {
        if(!hasNext())
            throw new NoSuchElementException();

        Object obj = null;
        if (newIter != null && newIter.hasNext()) obj = newIter.next();

        if (subIter.hasNext() && obj == null) {
```

```
            obj = subIter.next();

            if (obj instanceof Iterator && ((Iterator) obj).hasNext()) {
                newIter = new DeepIterator((Iterator) obj);

            }
        }

        if(obj instanceof Iterator){
            obj = next();
        }

        return (String) obj;
    }
}
```

7.6. Итератор по двум итераторам

Задача:

Напишите итератор, который проходит по двум итератором.

Решение:

```
/**
 * @author Irina Poroshina
 */
class ConcatIterator<T> implements Iterator<T> {

    private Iterator<T> innerIterator1;
    private Iterator<T> innerIterator2;

    public ConcatIterator (Iterator<T> innerIterator1, Iterator<T> innerIterator2) {
        this.innerIterator1 = innerIterator1;
        this.innerIterator2 = innerIterator2;
    }

    @Override
    public boolean hasNext() {
        while (innerIterator1.hasNext()) return true;
        while (innerIterator2.hasNext()) return true;
        return false;
    }

    @Override
    public T next() {
        if(!hasNext())
            throw new NoSuchElementException();

        while (innerIterator1.hasNext()) return innerIterator1.next();
        while (innerIterator2.hasNext()) return innerIterator2.next();
        return null;
    }
}
```

7.7. Подсчёт элементов

Напишите метод, который получает на вход массив элементов типа **K** (Generic) и возвращает Map<K, Integer>, где K — значение из массива, а Integer — количество вхождений в массив.
То есть сигнатура метода выглядит так:

```
<K> Map<K, Integer> arrayToMap(K[] ks);
```

Решение:

```
public static <K> Map<K, Integer> countValues(K[] ks) {
        Map<K, Integer> map = new HashMap<>();
        for (K k : ks) {
            map.compute(k, new BiFunction<K, Integer, Integer>() {
                @Override
                public Integer apply(K k, Integer count) {
                    return count == null ? 1 : count + 1;
                }
            });
        }

        return map;
    }
```

7.8. Поменять ключи и значения в Map

Напишите метод, который получает на вход **Map<K, V>** и возвращает Map, где ключи и значения поменяны местами. Так как значения могут совпадать, то тип значения в Map будет уже не **K**, а

```
Collection<K>:
```

```
Map<V, Collection<K>>
```

Решение:

```
public static <K, V> Map<V, Collection<K>> inverse(Map<? extends K, ? extends V> map){
        Map<V, Collection<K>> resultMap = new HashMap<>();

        Set<K> keys = map.keySet();
        for(K key : keys){
            V value = map.get(key);
            resultMap.compute(value, (v, ks) -> {
                if(ks == null){
                    ks = new HashSet<>();
                }
                ks.add(key);
                return ks;
            });
        }
```

```
        return resultMap;
    }
```

Многопоточность

8.0. Состояния

Задача:

Выведите состояние потока перед его запуском, после запуска и во время выполнения.

Решение:

```
        Thread thread = new Thread() {
            @Override
            public void run() {
                System.out.println(getState());
            }
        };
        System.out.println(thread.getState());
        thread.start();
        try {
            // Тут маленькая сложность есть только для вывода состояния TERMINATED:
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(thread.getState());
```

Добавим **WAITING и BLOCKED:**

```
    /**
     * Вывод состояния WAITING
     *
     * @param strings
     * @throws InterruptedException
     */
    public static void main(String[] strings) throws InterruptedException {
        Object lock = new Object();

        Thread thread = new Thread() {
            @Override
            public void run() {
                try {
                    synchronized (lock) {
                        lock.notifyAll();
                        lock.wait();
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
```

```
        synchronized (lock){
            thread.start(); // Запустим поток
            lock.wait(); // Будем ждать, пока поток не запустится
            System.out.println(thread.getState()); // WAITING
            lock.notifyAll();
            System.out.println(thread.getState()); // BLOCKED
        }
    }
```

Для **TIMED_WAITING** немного изменим тот же код:

```
    /**
     * Вывод состояния WAITING
     *
     * @param strings
     * @throws InterruptedException
     */
    public static void main(String[] strings) throws InterruptedException {
        Object lock = new Object();

        Thread thread = new Thread() {
            @Override
            public void run() {
                try {
                    synchronized (lock) {
                        lock.notifyAll();
                        lock.wait(3000);
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };

        synchronized (lock) {
            thread.start(); // Запустим поток
            lock.wait(); // Будем ждать, пока поток не запустится
            System.out.println(thread.getState()); // WAITING
        }
    }
```

8.1. Синхронизация потоков

Задача:

Напишите программу, в которой создаются два потока, которые выводят на консоль своё имя по очереди.

Решение:

```
class StepThread extends Thread {
```

```
    // общий для двух потоков lock
    private Object lock;

    public StepThread(Object object) {
        this.lock = object;
    }

    /**
     * Идея такая: выводим имя потока, потом поток засыпает,
     * перед этим уведомив другой поток, о том, что теперь его очередь.
     *
     * После вызова первым потоком lock.notify() второй поток
     * не просыпается сразу, а ждёт,
     * пока lock не будет освобождён. А когда это происходит, уже вызван
     * метод lock.wait(), и первый поток ждёт своей очереди. И так по кругу.
     */
    @Override
    public void run() {
        while (true) {
            synchronized (lock) {
                try {
                    System.out.println(getName());
                    lock.notify();
                    lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

public class Main {
    public static void main(String[] strings) {
        Object lock = new Object();
        new StepThread(lock).start();
        new StepThread(lock).start();
    }
}
```

8.2. Производитель-потребитель

Одна из классических задач по многопоточности. Дано два потока — производитель и потребитель. Производитель генерирует некоторые данные (в примере — числа). Производитель «потребляет» их.
Два потока разделяют общий буфер данных, размер которого ограничен. Если буфер пуст, потребитель должен ждать, пока там появятся данные. Если буфер заполнен полностью, производитель должен ждать, пока потребитель заберёт данные и место освободится.

Производитель:

```
// implements Runnable чтобы запускать в отдельном потоке
class Producer implements Runnable {
    // Общая очередь
    private final Queue<Double> sharedQueue;
```

```
    // Максимальный размер
    private final int SIZE;

    // Конструктор
    public Producer(Queue<Double> sharedQueue, int size) {
        this.sharedQueue = sharedQueue;
        this.SIZE = size;
    }

    @Override
    public void run() {
        // Цикл бесконечен
        while (true) {
            try {
                // В цикле вызывается метод produce
                System.out.println("Produced: " + produce());
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private double produce() throws InterruptedException {
        synchronized (sharedQueue) { // обязательно synchronized
            if (sharedQueue.size() == SIZE) {
                // Если очередь полна, то ждём
                sharedQueue.wait();
            }

            // Добавили элемент в очередь.
            double newValue = Math.random();
            sharedQueue.add(newValue);

            // Уведомили другой поток на случай, если он ждет
            sharedQueue.notifyAll();

            return newValue;
        }
    }
}
```

Потребитель:

```
// implements Runnable чтобы запускать в отдельном потоке
class Consumer implements Runnable {
    // Общая очередь
    private final Queue<Double> sharedQueue;

    public Consumer(Queue<Double> sharedQueue) {
        this.sharedQueue = sharedQueue;
    }

    @Override
    public void run() {
        while (true) {
            try {
```

```
                System.out.println("Consumed: " + consume());
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }

    // Метод, извлекающий элементы из общей очереди
    private Double consume() throws InterruptedException {
        synchronized (sharedQueue) {
            if (sharedQueue.isEmpty()) { // Если пуста, надо ждать
                sharedQueue.wait();
            }

            sharedQueue.notifyAll();
            return sharedQueue.poll();
        }
    }
}
```

Создание и запуск:

```
public static void main(String[] strings) {
        LinkedList<Double> sharedQueue = new LinkedList<>();
        int size = 4;
        Thread prodThread = new Thread(new Producer(sharedQueue, size), "Producer");
        Thread consThread = new Thread(new Consumer(sharedQueue), "Consumer");
        prodThread.start();
        consThread.start();
}
```

9.0. Своя аннотация — создание и использование

Эту задачу я обычно даю, когда речь заходит про аннотации и reflection. Заодно, можно рассказать про **Executors**, **ThreadPoolExecutor** и другие.

Задача:

Создайте свою аннотацию **Repeat** с целочисленным параметром.
Расширьте класс **ThreadPoolExecutor** и переопределите метод *execute* следующим образом: если экземпляр **Runnable** имеет аннотацию **Repeat**, то его метод *run* выполняется несколько раз (количество задается параметром в **Repeat**).

То есть, написав такой класс:

```
@Repeat(3)
class MyRunnable implements Runnable{
    @Override
    public void run() {
        System.out.println("Hello!");
    }
}
```

и использовав его:

```
public static void main(String[] strings) {
        CustomThreadPoolExecutor customThreadPoolExecutor =
                        new CustomThreadPoolExecutor(10);
        customThreadPoolExecutor.execute(new MyRunnable());
}
```

Мы должны увидеть:

```
Hello!
Hello!
Hello!
```

Решение:

```
@Retention(RetentionPolicy.RUNTIME)
@interface Repeat {
    int value();
}

class CustomThreadPoolExecutor extends ThreadPoolExecutor {
    public CustomThreadPoolExecutor(int corePoolSize) {
        // why Integer.MAX_VALUE, 0m and TimeUnit.MILLISECONDS?
        // see Executors.newFixedThreadPool(int nThreads)
        super(corePoolSize, Integer.MAX_VALUE, 0, TimeUnit.MILLISECONDS,
                new LinkedBlockingQueue<>());
    }

    @Override
    public void execute(Runnable command) {
        if (command != null) {
            Class<? extends Runnable> runnableClass = command.getClass();
            Repeat repeat = runnableClass.getAnnotation(Repeat.class);
            for (int i = 0; i < (repeat != null ? repeat.value() : 1); i++) {
                super.execute(command);
            }
        }
    }
}
```

Итоговые и прочие задания

В течение курса я даю студентам несколько сложных задач — на целое занятие. Требуется написать небольшую программу, используя ранее изученное. Кстати, тут часто возникает сложность. Решение задач, сводящихся к написанию одного метода — это одно, а придумать алгоритм, вспомнить всё, что изучали ранее и еще и написать сразу 50 строк на Java — совсем другое. Но на занятии я могу подталкивать их в нужном направлении, помогаю решать проблемы, дебажить, находить нужные классы и методы, и так далее. Несколько таких задач описаны ниже. В таком виде я и даю их своим студентам.

Кроме того, в конце курса все должны выполнить итоговое задание. То есть дома, самостоятельно, написать программу. Еще немного более сложную. Я даю возможность выбрать один из нескольких вариантов. Кстати, интересный факт — необходимо написать минимум одну программу, а можно написать сразу несколько. Кажется, я помню только одного человека, кто написал больше одной.

10.0. Количество дорожных ограничений

Небольшая задача, которая демонстрирует, как можно приложить Java к решению практических задач.

Подготовка данных:
С портала открытых данных Санкт-Петербурга загружаем данные об ограничении движения транспорта на период производства работ в формате csv.

Задача:

Требуется определить, сколько дорожных ограничений действовало в городе на определенную дату.

Программа в качестве аргумент получает два параметра:

- Путь к файлу с данными
- Дата

то есть она запускается следующим образом:

```
java TrafficBlocks "PATH_TO_CSV_FILE" dd.MM.yyyy
```

Необходимо вывести количество действовавших ограничений движения транспорта на эту дату.

Примерный алгоритм

- Обработать аргументы, переданные через командную строку
- Прочитать файл построчно
- Распарсить строки (см. https://stackoverflow.com/questions/15738918/splitting-a-csv-file-with-quotes-as-text-delimiter-using-string-split)
- Чтобы получить из строки дату, можно использовать SimpleDateFormat
- Сравнить даты

Кстати, за всё время только один человек заметил, что формат даты в данных (yyyyMMdd) такой, что их можно не парсить, а сравнивать как строки. Так что решение можно упростить. Я даю эту задачу после разговора про Date, Calendar, DateFormat, так что я про это упрощение говорю, когда они уже всё написали.

Решение я тут не привожу, их может быть много разных и каждое надо смотреть индивидуально.

10.1. Поиск по Википедии. В консольной программе

Задача:

Напишите программу, которая с консоли считывает поисковый запрос, и выводит результат поиска по Википедии. Задача разбивается на 4 этапа:

1. Считать запрос
2. Сделать запрос к серверу
3. Распарсить ответ
4. Вывести результат

Первый и четвертый пункт не сильно нуждаются в пояснении, остановимся на запросе к серверу.

Эту задачу тоже можно разбить на несколько этапов:

1. Генерация запроса
2. Запрос к серверу
3. Подготовка к обработке ответа
4. Обработка ответа

Рассмотрим это подробнее:

Генерация запроса
API предоставляет возможность делать поисковые запросы, без ключей. Вот таким, примерно, образом:

```
https://ru.wikipedia.org/w/api.php?action=query&list=search&utf8=&format=json&srsearch="Java
```

Вы можете открыть эту ссылку в браузере, и посмотреть на результат запроса.
Однако, чтобы запрос прошел удачно, следует убрать из ссылки недопустимые символы, то есть сделать Percent-encoding, он же URL Encoding.
Для этого в Java можно воспользоваться статическим методом *encode* в классе URLEncoder, вот так:

```
street = URLEncoder.encode(street, "UTF-8");
```

Вот и всё, URL готов! Осталось теперь сделать запрос к серверу…

Запрос к серверу
Для GET и POST запросов можно воспользоваться классом HttpURLConnection. Это самое простое. Просто создать, открыть соединение и получить **InputStream**. Нам его даже не надо читать, за нас это сделает Gson.
Ещё можно использовать retrofit, или что-то подобное.

Подготовка к обработке ответа

Сервер возвращает данные в формате JSON.
Но нам его не надо парсить вручную, для этого есть библиотека Gson от Google.
Примеры есть тут:
https://github.com/google/gson
https://habrahabr.ru/company/naumen/blog/228279/

Если остаётся время, можно написать получение статьи, выбранной при поиске и так далее.

10.2. Итоговое задание — консольная утилита для скачивания файлов по HTTP

Консольная утилита для скачивания файлов по HTTP… звучит знакомо? Да, это оно и есть — История одного тестового задания. Всё логично — итоговое задание курса по Java такого же уровня, как и тестовое задание на должность Junior Java разработчика.
И это действительно хорошая задача — несложная, но охватывает самые разные темы. И сразу видно, насколько автор структурирует код, использует разные подходы и паттерны, использует сам язык и стандартную библиотеку.

10.3. Итоговое задание — погодный Telegram-бот

Задача:

Напишите бота для Telegram, который будет:

- Сообщать текущую погоду и прогноз на сутки в ответ на присланное местоположение.
- Присылать прогноз погоды на сутки каждый день, если пользователь подписался на ежедневную рассылку. Подписка должна происходить с помощью команды /subscribe. Также должна быть предусмотрена возможность отписки (/unsubscribe).

Для получения прогноза можно использовать https://openweathermap.org/api.

Эта задача скорее на умение и способность разобраться в новой технологии (bot-api) и разных библиотеках. А еще настроить VPN надо! И код написать придётся, само собой.

Кстати, интересный факт — большинство студентов игнорируют словосочетание «присланное местоположение» и наличие возможности его отправки. Они пишут бота, который ожидает название города. Я не знаю почему. Это часто работает плохо, код становится немного сложнее, но они продолжают это делать.

10.4. Итоговое задание — распознавание рукописных цифр

Задача:

Реализовать программу для классификации рукописных цифр.
Эта задача уже больше ориентирована на реализацию алгоритма, умение разбираться в таковых. Обычно код у студентов получается не очень структурированный.

Описание задачи
В качестве исследуемого набора данных будет использоваться база изображений рукописных цифр MNIST. Изображения в данной базе имеют разрешение 28x28 и хранятся в виде набора значений оттенков серого. Вся база разбита на две части: тренировочную, состоящую из 50000 изображений, и тестовую — 10000 изображений.

Для решения этой задачи предлагается реализовать метод k ближайших соседей — метрический алгоритм для автоматической классификации объектов. Основным принципом метода ближайших соседей является то, что объект присваивается тому классу, который является наиболее распространённым среди соседей данного элемента.
Соседи берутся исходя из множества объектов, классы которых уже известны, и, исходя из ключевого для данного метода значения k рассчитывается, какой класс наиболее многочислен среди них. В качестве расстояния между объектами можно использовать Евклидову метрику, то есть привычное нам расстояние между точками в пространстве.

Требования

Необходимо написать программу, которая будет распознавать рукописные цифры. Должна быть возможность инициализировать некий класс данными для обучения и предоставить метод для распознавания одного изображения.

Кроме самой реализации алгоритма, следует написать код для проверки его точности (посчитать error rate). Для этого следует использовать 10000 тестовых изображений.
Кроме вычисления точности предлагается провести эксперимент: вместо евклидовой метрики использовать расстояние городских кварталов, угол между векторами или что-то еще, и проверить качество распознавания.

Дополнительно

Если всё хорошо работает, то можно ещё немного усложнить задание. Добавив, к примеру, отсев шума (выбросов) или использование метода Парзеновского окна для повышения точности.

Еще пара слов

Если у Вас есть крутые задачи, которые можно предлагать студентам (ориентировочное время решения — час-два), то делитесь ими в комментариях.

Теги: java, задачи и решения, задачи

Хабы: Java

48 Карма 0 Рейтинг

Роман @Nucleotide
Инженер-программист

ПОХОЖИЕ ПУБЛИКАЦИИ

13 апреля в 15:16

Маски, картины, тайные покупатели и анализ продаж: разбираем решения задач для Go-разработчиков

+20 4.5K 39 25 +25

23 августа 2021 в 20:54

Apache Flink и потоковая обработка данных для решения задач IoT

+10 3.6K 18 2 +2

1 июля 2021 в 14:18

Симплексный метод решения задач линейного программирования

+2 6.3K 29 6 +6

МИНУТОЧКУ ВНИМАНИЯ

Промокод — твой билет в общество потребления

Хотите рассказать о себе в наших социальных сетях?

Конкурс технических статей Технотекст 2021

ЗАКАЗЫ

Рекомплирировать ISO-файл, достать исходники и проконсультировать
5000 руб./за проект · 2 отклика · 6 просмотров

Fullstack / Frontend / Backend в IT стартап
1000 руб./в час · 16 просмотров

Создать фон для блока на сайте
5000 руб./за проект · 6 откликов · 38 просмотров

Натянуть верстку и написать код одностраничного сайта laravel
25000 руб./за проект · 5 откликов · 26 просмотров

Доработка веб приложения (js/css)
3000 руб./за проект · 3 отклика · 39 просмотров

Больше заказов на Хабр Фрилансе

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 16:00

Удивительное рядом, но оно запрещено: что такое «номерные радиостанции»

+70 21K 66 48 +48

вчера в 19:08

Как я создал собственный 3D движок и игру на нём за 20 месяцев

+65 11K 43 23 +23

вчера в 15:23

Народная дозиметрия. Бюджетный детектор радиации своими руками

+29 4.4K 59 22 +22

вчера в 15:37

[Личный опыт] Жизнь в Швеции: полное благополучие, тесты на IQ и тысячи вакансий в IT

+24 18K 66 172 +172

вчера в 20:04

Reticulum — радиопротокол для mesh-сети. Зашифрованная пиринговая связь без интернета

+22 4.4K 58 7 +7

Чатботы, трансформеры, беспилотный транспорт: экспресс-тур по городу ИИ

Мегапост

РАБОТА

Java разработчик
540 вакансий

Все вакансии

Вернуться на старую версию