Методы

[модификаторы] <тип возвращаемых данных> имяМетода(аргументы, аргументы,...) {}

Модификаторы:

public - доступный для любого другого кода.
private - доступный только членам этого класса.
protected - только при использовании наследования.
без модификатора - открыт только для кода внутри собственного пакета (будем рассматривать позже).

Если не указываем тип возвращаемых данных, то надо использовать void. В этом случае метод НЕ возвращает значение в ту точку кода, откуда он был вызван.

Если не void, то надо использовать return.
static - используются независимо от объектов класса.

Пример вызова метода

```
public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                System.out.println("Введите размер одномерного массива");
                int[] d = new int[n]; // Объявление массива d

                System.out.println("Введите массив");
                d = massiv2(n);

                String s1 = "Вот ";
                String s2 = "такая ";
                String s3 = "строка ”;
                stroka1(s1);
                stroka1(s1, s2);
                stroka1(s1, s2, s3);
                }
```

Пример вызова метода

```
static void stroka1(String st1) {
        System.out.println(st1);}

static void stroka1(String st1, String st2) {
        System.out.println(st1 + st2);}

static void stroka1(String st1, String st2, String st3) {
        System.out.println(st1 + st2 + st3);}

static void stroka1(String st1, String st2, String st3, String st4) {
        System.out.println(st1 + st2 + st3 + st4);}
static int[] massiv2(int j) {
        int[] mas = new int[j];
        Scanner vvod = new Scanner(System.in);
        for (int i = 0; i < j; i++) {
                System.out.println("Введите элемент массива [" + i + "]");
                mas[i] = vvod.nextInt();
        }
        return mas;
}
```

Метод с параметрами

```
public class ClassInfef1 {

        public static void main(String[] args) {

                String s1 = "a"; String s2 = "b"; String s3 = "c";
                String s4 = "d"; String s5 = "e"; String s6 = "f";
                String s7 = "g"; String s8 = "i";

                int kk = 1; boolean ft = false;
                System.out.println(metodInf(kk, ft, s1, s2, s3));
                System.out.println(metodInf(kk, ft, s1, s2, s3, s4));
                System.out.println(metodInf(kk, ft, s1, s2, s3, s4, s5));
                System.out.println(metodInf(kk, ft, s1, s2, s3, s4, s5, s6));
                System.out.println(metodInf(kk, ft, s1, s2, s3, s4, s5, s6, s7));
                System.out.println(metodInf(kk, ft, s1, s2, s3, s4, s5, s6, s7, s8));
        }

        static String metodInf(int k, boolean frtr, String... newStr) {
                String stroka = "";
                for (int i = 0; i < newStr.length; i++) {
                        stroka = stroka + newStr[i];
                }
                if (frtr) {
                        stroka = stroka + k;
                }
                return stroka;
        }
}
```

Задачи

1. Написать «калькулятор» на операции +,-,/,*, где пользователь выбирает операцию, вводит числа, и вызывается метод, что их считает.
2. Написать метод, что проверяет, является ли строка палиндромом (одинаково читается с двух концов). Он выдает true или false.
3. Написать меню для выполнения операций над массивами (предусмотреть повторное выполнение программы, продолжение выполнения операций над одной матрицей).
4. Посчитать пятую степень матрицы с помощью метода, который перемножает 2 матрицы.

$$c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} \quad (i = 1, 2, \ldots m;\ j = 1, 2, \ldots q).$$

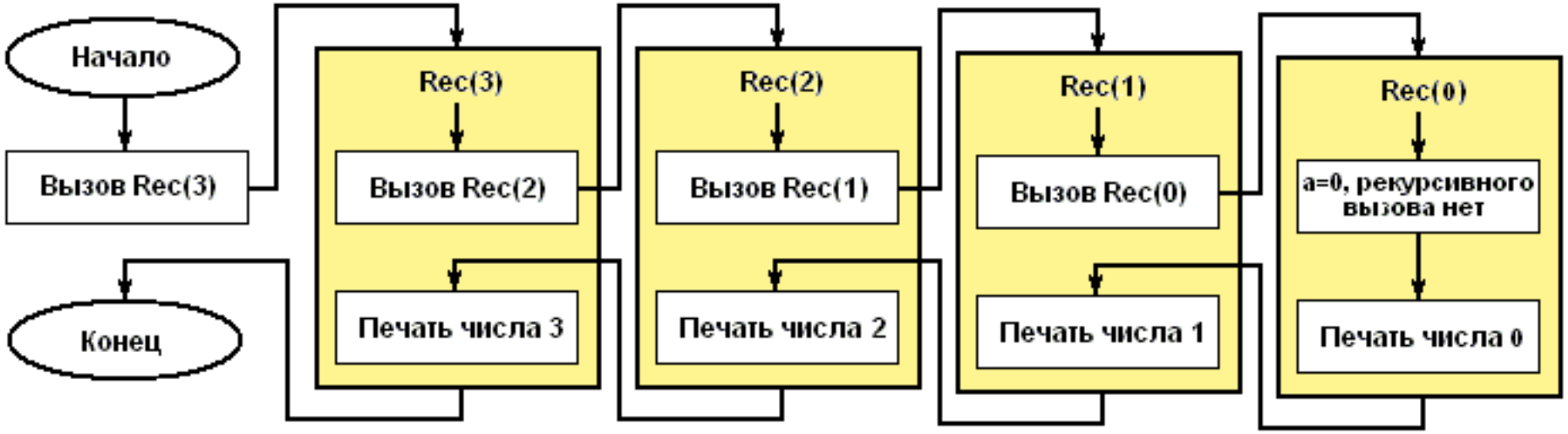
Рекурсивные алгоритмы

Метод может содержать вызов других методов. В том числе метод может вызвать сам себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов метода, просто начинает выполнять его. Без разницы, какой метод дал команду это делать.

Пример рекурсивного метода:

```
static void rec(int a){
if (a>0) {
    rec(a-1);
}
    System.out.println(a);
}
```

Рекурсивные алгоритмы



Процедура Rec вызывается с параметром a = 3. В ней содержится вызов процедуры Rec с параметром a = 2. Предыдущий вызов еще не завершился, поэтому можете представить себе, что создается еще одна процедура и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра процедуры. Количество одновременно выполняемых процедур называют глубиной рекурсии.

Четвертая вызванная процедура (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к процедуре, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все процедуры. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Задачи

1. Создать массив на N элементов со случайными элементами. Циклы не использовать.
2. Посчитать факториал числа, используя рекурсивные алгоритмы.
3. Найти число Фибоначчи под номером N, используя рекурсию.
4. Посчитать детерминант матрицы, используя рекурсию.
5. Отсортировать массив, используя рекурсию.
6. Решить задачу с помощью рекурсии.

Дана матрица нулевых элементов. В любом месте матрицы ставится одно значение 1. Нужно посчитать за сколько ходов единица «захватит мир», если каждый ход область ее владений расширяется на соседние элементы.