JAVA SCHOOL

Module 0 lesson 3

Методы. Массивы. Циклы, оператор switch.

Методы

В java большая часть кода и логики располагается в методах

Мы уже знакомы с одним методом, это **public static void main(String[] args)**

У этого метода достаточно жесткая сигнатура, которую нельзя менять, так как он является точкой входа в программу

Разберем из каких частей состоят методы в java

Методы

public - модификатор доступа, пока мы будем использовать именно этот, другие разберем немного позже

static - тоже модификатор, не всегда присутствует в сигнатуре метода, но мы пока будем использовать

void - тип возвращаемого значения, в данном случае никакого значения не возвращается

main - имя метода, может быть абсолютно произвольным

String[] args - входные параметры, в данном случае массив строк (тип **String**)

Методы

Давайте напишем код, который будет считывать два введенных два числа типа **int** *first* и *second*

```
public class Main {

    public static void main(String[] args) {
        Scanner inputFromLine = new Scanner(System.in);
        int first;
        int second;

        System.out.println("Введите первое число");
        first = inputFromLine.nextInt();

        System.out.println("Введите второе число");
        second = inputFromLine.nextInt();
```

Методы

Напишем метод который будет называться **sum**

Метод будет располагаться внутри класса **Main**, чуть ниже метода **main**

Модификаторы у метода будут **public static**

Тип возвращаемого значения **int**, так как нам нужно получить из этого метода целочисленное значение, которое будет являться суммой двух аргументов, переданных в этот метод

Поэтому объявим внутри метода переменную типа **int** с именем *result,* и присвоим значение 0. С помощью ключевого слова **return** в конце метода вернем ее

Метод будет принимать две переменные, **int** *firstNumber* и **int** *secondNumber,* несколько входных параметров нужно объявлять через запятую

Методы

Попробуем вызвать наш метод. Для сохранения вызова метода создадим еще одну переменную такого же типа как и возвращаемое значение метода, типа **int** и назовем ее *resultSum*

Вызов метода осуществляется написанием его имени и передачи параметров в круглых скобках

В нашем случае это будет **sum(first, second);**

Но если мы так вызовем метод, то не сохраним результат его выполнения, именно для этого мы создали еще одну переменную **int** *resultSum*

Для эффективного вызова метода, мы должны написать следующий код:

resultSum = **sum(first, second);**

Давайте выведем значение переменной *resultSum*

Методы

Немного модифицируем наш метод и избавимся от переменной ***resultSum***

Напишем вызов метода сразу в конструкцию **System.out.println**

```
    System.out.println("Сумма: " + sum(first, second));
}

public static int sum(int firstNumber, int secondNumber){
    int result = 0;
    result = firstNumber + secondNumber;
    return result;
}
```

Результат будет один и тот же, но зато мы сократили наш код

Методы

Давайте еще сократим наш код, избавимся от переменной *result* в методе **sum**

Вернем сразу сумму чисел, ведь результат этой операции будет тип **int**

```
public static int sum(int firstNumber, int secondNumber){
    return firstNumber + secondNumber;
}
```

И у метода возвращаемое значение **int**

Значит мы можем после слова **return** сразу написать операцию, результат которой будет тип **int**

Массив

Массив - это структура данных, хранящая набор значений, идентифицируемых по индексу

Идентифицируемых по индексу это значит что у каждого элемента в массиве есть индекс, по которому можно обратиться к элементу

Индекс	0	1	2	3	4	5	6	7
Элемент	10	381	53	1	43	6	76	64

Сейчас представлен массив, который хранит целочисленные значения (например **int**)

Индексы всегда будут начинаться с нуля и под каждым индексом может находится или не находится элемент. Значение элемента может быть абсолютно произвольным

Так же массив можно представить как коробку, внутри которой лежат в один ряд какие-то вещи одного типа, например яблоки и у каждого яблока есть свой порядковый номер, начинающийся с нуля, это будет индексом, а сами яблоки это элементы

Массив

Создать массив можно следующим образом:

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

Этот код создает массив с именем ***mass*** размеров в 10 элементов, который будет располагать в себе значения типа **int**

Индексы массива начинаются с ***0*** и так как размер равен ***10*** то индекс последнего элемента будет равен ***9***

Массив

Существует еще один способ заполнения массива, сразу при объявлении:

```
int[] mass = new int[]{0, 10, 13, 15, 8, 32};
```

Этот код создает массив с именем ***mass*** размеров в *6* элементов, который будет хранить значения типа **int** и добавили *6* значений

Такой способ не самый эффективный, мы будем использовать циклы для заполнения массива

Массив

Объявим переменную для считывания из консоли и массив с именем ***mass*** типа **int** размером ***5***

```
import java.util.Scanner;

public class MainMass {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] mass = new int[5];
    }
}
```

Массив

Давайте создадим новый класс и назовем его **MainMass**

В нем создадим метод **main** как точку входа в программу

В новом классе мы будем работать с циклами и массивами

Этими действиями немного разделим наш код

Циклы

В java существует 4 вида циклов:

```
for (объявление счетчика; условие выполнения цикла; изменение счетчика){
    код внутри цикла
}

for (объявление типа данных и переменной внутри массива : массив с типом данных){
    код внутри цикла
}

while (условие выполнения цикла){
    код внутри цикла
}

do {
    код внутри цикла
} while (условие выполнения цикла)
```

Циклы

Попробуем использовать первый цикл. В цикле заполним массив числами от 0 до 4. Цикл будет выглядеть следующим образом:

```
for (int i = 0; i < 5; i++) {
    //код цикла
}
```

В скобках первым делом мы объявляем счетчик итераций цикла и устанавливаем начальное значение, то есть ноль **int i = 0**

Далее накладываем условия выхода из цикла, это должно быть выражение, результат которого будет тип **boolean. i < 5**

Так же мы указываем как будем изменяться счетчик с каждой итерацией цикла, в нашем случае он будет увеличиваться на единицу, это операция инкремента. **i++**

Циклы

Заполнить массив можно с помощью обращения к элементам по индексу, пока что он пустой, но используя счетчик цикла, можно его заполнить, так же каждую итерацию будем выводить результат на экран

```
for (int i = 0; i < 5; i++) {
    int newElement = i + 1;
    mass[i] = newElement;
    System.out.println("Элемент под индексом: " + i +
            " Заполняется значением" + newElement);
}
```

Циклы

Модифицируем наш цикл, будем его заполнять значениями введёнными в консоль

```
for (int i = 0; i < 5; i++) {
    mass[i] = input.nextInt();
}
```

Создадим метод, который будет называться **printMass**, он ничего не будет возвращать (**void**) а принимать будет массив типа **int[]**

Внутри этого метода, создадим еще один цикл, он будет выводить каждый элемент массива в консоль

Далее вызовем этот метод в методе **main** и передадим туда заполненный массив

Циклы

Превратим цикл в методе **printMass** в цикл **foreach**, это немного модифицированный цикл **for**

```
public static void printMass(int[] mass) {
    for (int el : mass) {
        System.out.println("Значение элемента: " + el);
    }
}
```

При использовании цикла **foreach** у нас уже нет возможности посмотреть индекс, только если завести како-нибудь счетчик и увеличивать его на единицу с каждой итерацией

Цикл **foreach** используется в том случае, когда нужно пройти с первого по последний элемент, на место переменной **int el** будут последовательно подставляться элементы из массива, который описан через знак **двоеточия**

Таким образом, обращаясь к переменной **int el** мы захватим все элементы массива

Switch

Далее, познакомимся с оператором **switch case**

Эта конструкция используется достаточно редко и лучше ее избегать при написании кода, но познакомиться с ней нужно

Оператор **switch case** можно использовать когда существует множественный выбор

Например, если логика будет отталкиваться от того, какой сейчас день недели, и на каждый день недели будет выполняться разный код

Такую ситуацию можно заменить и множеством операторов **if**, но через **switch case** получится более наглядно

Switch

Создадим еще один метод, он будет называться **printDayOfWeek**

Метод будет принимать номер дня недели и выводить на экран какому дню соответствует этот номер

```
public static void printDayOfWeek(int dayNumber) {
    switch (dayNumber) {
        case 1:
            System.out.println("Понедельник");
            break;
        case 2:
            System.out.println("Вторник");
            break;
        case 3:
            System.out.println("Среда");
            break;
```

Итог

Сегодня мы с вами познакомились с методами

Научились создавать их и вызывать. В следующих занятиях мы будем сталкиваться с ними постоянно, не страшно если остались какие-то вопросы

Так же познакомились с массивами, научились ими пользоваться. Далее мы тоже будем часто говорить о них

Разобрались с циклами и конструкцией **switch case**