IT ШКОЛА SAMSUNG

Модуль 1. Основы программирования

Урок 9. Функции

SAMSUNG

Поработаем с массивом

Вспомним пример с массивом из прошлой лекции (поиск максимума):

```
public static void main(String[] args) {
      // Создаем массив
       int[] a = {5,3,4,5,5,4}

       // Ищем максимум
       int max = a[0];
       for (int i = 0; i < a.length; i++){
             if (a[i] > max){
                   max = a[i];
             }
       }
       System.out.println(max);
  }
```

А теперь с тремя.

Модифицируем пример для нахождения максимума в трех массивах (например массивы оценок по истории, физике и алгебере):

```
public static void main(String[] args) {
        // Создаем массивы
        int[] istoria = new int[] {3,4,4};
        int[] fisika = new int[] {4,4,4,5,4};
        int[] algebra = new int[] {4,4,2,4,4};
```

To be continued...

А теперь с тремя.

```
//ищем максимум массива 1
int max = istoria[0];
for (int i = 0; i < istoria.length; i++){
    if (istoria[i] > max){
        max = istoria[i];
    }
}
System.out.println(max);
```

To be continued...

А теперь с тремя.

```
//ищем максимум  массива 2
max = fisika[0];
for (int i = 0; i < fisika.length; i++){
    if (fisika[i] > max){
        max = fisika[i];
    }
}
System.out.println(max);
```

To be continued...

А теперь с тремя.

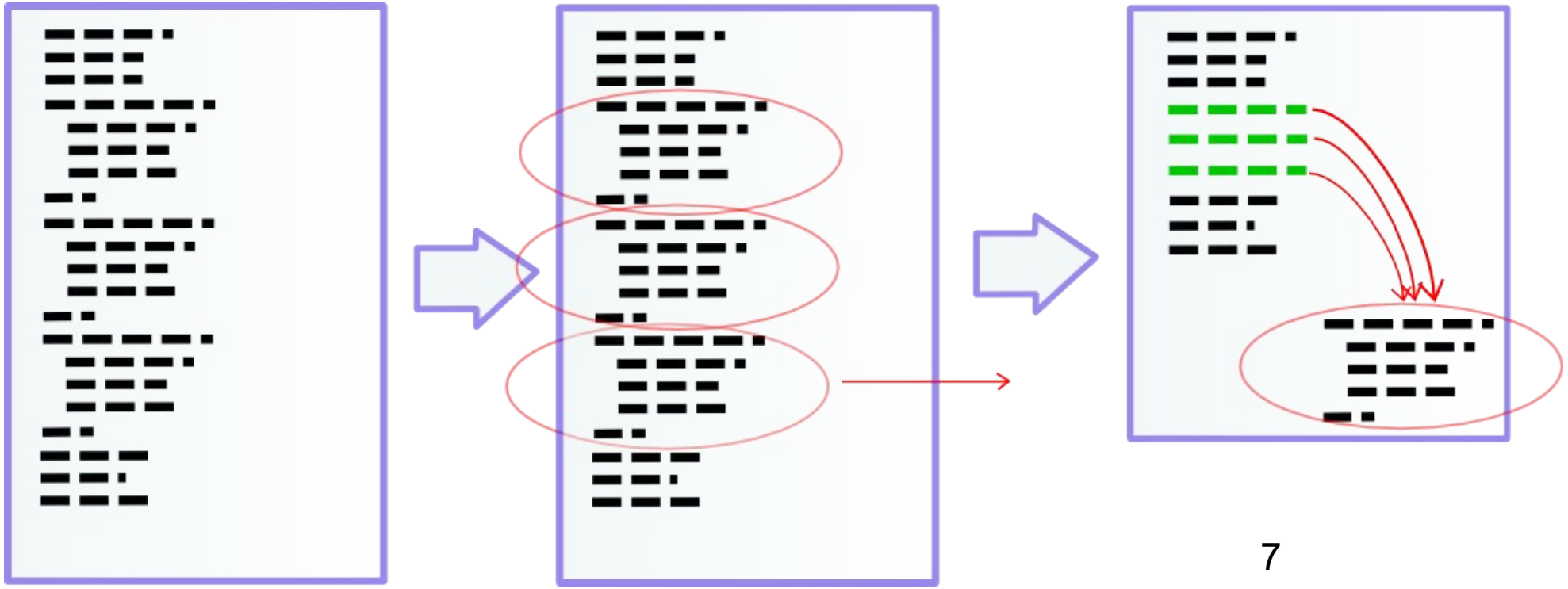
```
        //ищем максимум  массива 3
         max = algebra[0];
         for (int i = 0; i < algebra.length; i++){
              if (algebra[i] > max){
                   max = algebra[i];
              }
         }
         System.out.println(max);
    }

}
```

Закончили, НО...

Нужно больше массивов!

У любого учащегося гораздо больше предметов чем 3. Поэтому модифицируем нашу программу например на 15 массивов :-/ . А теперь прикинем – если в первом примере у нас было ~10 строк, во втором ~40 строк, то для 15 массивов нам понадобится ~200 строк. Определённо с этим надо что-то делать.

Очевидное решение – вынести в сторонку часто повторяющийся кусок программы и когда он понадобится просто выполнять (вызывать) этот кусок.



7

Функции

Примерно так появились функции.

- Таким образом, часто повторяющийся код выносят отдельно и дают ему имя (надо же как то к нему обратиться) – “объявляют” функцию. Тем более, что функций может быть несколько.
- Когда нам нужно воспользоваться функцией – мы ее «вызываем», то есть пишем ее имя и передаваемые ей значения, если нужно.
- Кроме того наша функция должна обрабатывать каждый раз, когда мы ее вызываем, разные данные, которые нас интересуют (оценки по истории,физике, алгебре). Так у функции появились входные параметры – то есть мы указываем, какие данные мы передадим функции, чтобы она с ними работала. Параметров у функции может быть несколько.
- Может быть, что некоторое значение (но только одно) функция возвращает как результат своей работы. То есть, в объявлении функции надо еще указать тип возвращаемого значения.

Функции

Для определения (объявления) функций используется следующая запись:

```
тип_данных_которые_метод_вернет ИМЯ_МЕТОДА (данные_передающиеся_методу){
   здесь размещаем тот код,
   который мы хотим, чтобы выполнялся
   при вызове этого метода
}
```

В нашем случае, давайте назовем код **maxArray** и передадим ему одну переменную - массив:

```
int maxArray(int[] ar){
   int max=ar[0];
   for(int i = 0; i < ar.length; i++){
      if(ar[i] > max){
         max = ar[i];
      }
   }
   return max;
}
```


9

Функции

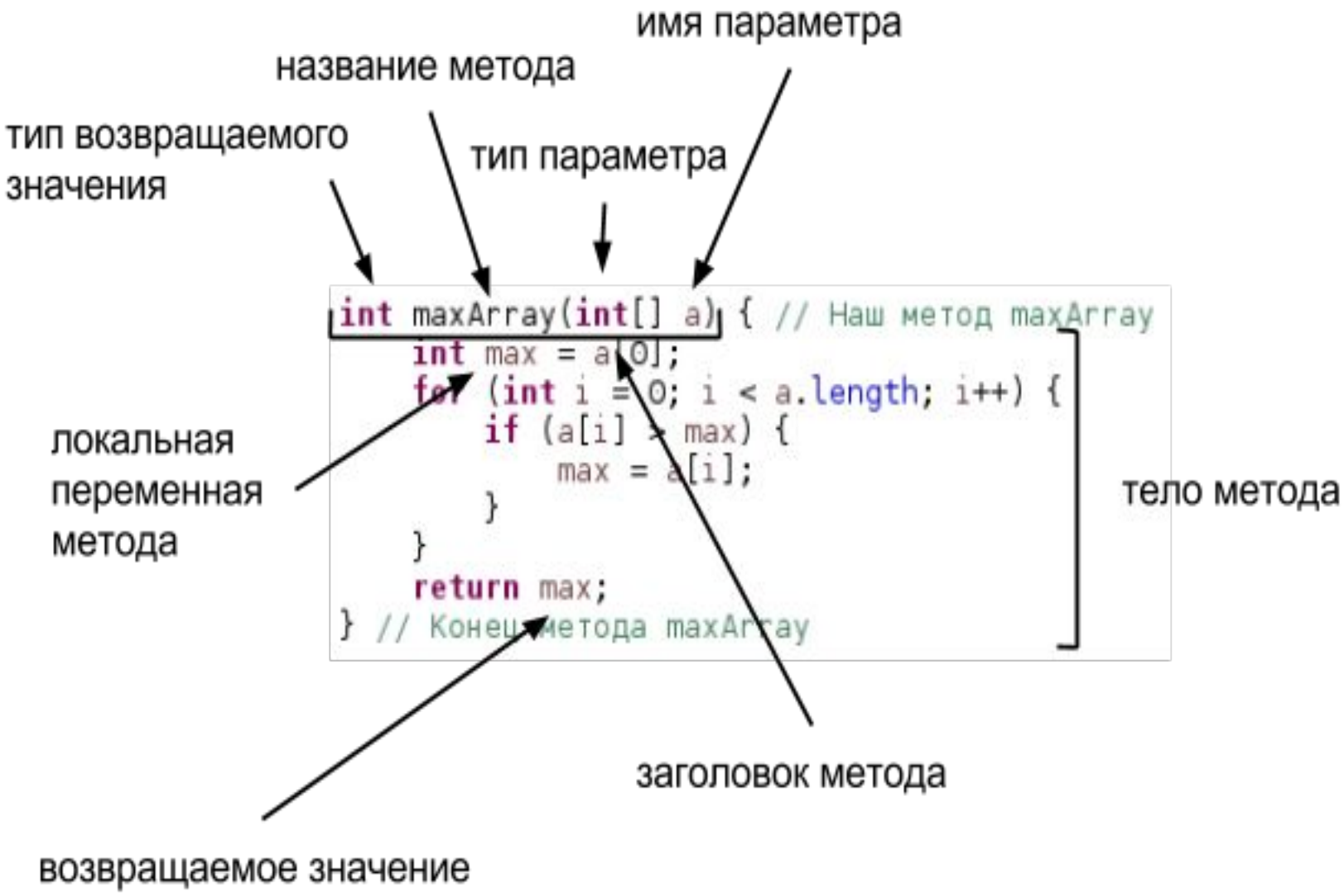
*Именованный участок кода в программировании называют термином **функция.** В объектно-ориентированных языках, а Java является именно таким языком, функции обязательно определены (прикреплены) в каком-либо классе, эта особенность (принадлежность функции классу) отразилась в том, что функцию назвает **методом**.*
Так и говорят: “у этого класса есть следующие методы” или “эти методы находятся в данном классе” или “этот метод из данного класса, определен в данном классе”. Так же могут сказать, что “метод принадлежит классу”.

Обратите внимание, что после выполнения команды **`return`**, метод завершает свою работу, причем не важно в каком месте кода произошло выполнение этой команды. Выполнили **`return`** - вышли из метода.

В функции может быть несколько операторов **`return`**. Например, если в зависимости от условия, надо вернуть разные данные, то **`return`** могут расположить внутри конструкции **`if`**.

10

Объявление функции/метода



11

Функции

Осталось разобраться, в какой части программы можно определять собственные методы. Ответ прост: в любом месте тела класса, то есть, другими словами, внутри фигурных скобок класса:

```
public class test {

static int maxArray(int[] ar){ // Наш метод maxArray
    int max = ar[0];
    for(int i = 0; i < ar.length; i++){
        if(ar[i] > max){
            max = ar[i];
        }
    }
    return max;
} // конец метода maxArray

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

To be continued...

12

Функции

```
        int[] istoria = new int[] {3,4,4};
        int[] fisika = new int[] {4,4,4,5,4};
        int[] algebra = new int[] {4,4,2,4,4};
         //находим максимумы
         int max=maxArray(istoria);
         System.out.println(max);
         max=maxArray(fisika);
         System.out.println(max);
         max=maxArray(algebra);
         System.out.println(max);
    }
}
```


13

Упражнение 1.9.1

Попробуйте самостоятельно создать метод для заполнения массива. Этот метод должен принимать массив и заполнять его случайными значениями.

Вот код решения этого упражнения:

```
public class test {

    static int maxArray(int[] ar) { // Наш метод нахожд максимума
        int max = ar[0];
        for (int i = 0; i < ar.length; i++)
            if (ar[i] > max)
                max = ar[i];
        return max;
    } // конец метода maxArray

    static int[] fillArray(int[] ar) { // Наш метод заполнения массива
        for (int i = 0; i < ar.length; i++)
            ar[i] = (int) (Math.random() * (5 + 1));
        return ar;
    } // конец метода fillArray
```


14

```
    public static void main(String[] args) {
        // Создаем массивы
        int[] istoria = new int[6];
        int[] fisika = new int[8];
        int[] algebra = new int[10];
        // Заполняем массивы
        istoria=fillArray(istoria);
        fisika=fillArray(fisika);
        algebra=fillArray(algebra);
        // Печатаем максимумы массивов
        System.out.println(maxArray(istoria));
        System.out.println(maxArray(fisika));
        System.out.println(maxArray(algebra));
    }
}
```


15

Функции

До этого урока метод **`main`** мы воспринимали, как основную программу системы, но сейчас вы знаете, что метод **main**, это просто метод класса, один из многих, отличается он лишь тем, что с него начинается выполнения программы.

С другой стороны, на каждый метод можно смотреть как на маленькую программу со своим внутренним миром: этот мир начинается с открытия скобки метода и заканчивается, когда скобка закрывается, в нем живут свои локальные переменные, которые существуют только внутри метода и не видны ни от куда больше (из других методов, например).

16

Параметры функции

Java строго сопоставляет количество определенных параметров, их тип, и очередность в вызове метода с его определением. Иначе говоря, нельзя попросить “сходить в батон и принести магазин”. Если же мы не хотим ничего передавать в метод, то круглые скобки оставляем пустые и в заголовке метода и при вызове метода.

```
static int getResult(int x, int y, int z) {
        return x + y + z;
}
```


17

Передача значений в функцию

В Java, если параметры функции примитивного типа (byte, short, int, long, float, double, boolean, char), то они всегда **передаются по значению**. Все остальные типы данных в Java - это объекты, и они **передаются по ссылке**. Более подробно отличие этих способов передачи мы рассмотрим в следующей теме.

Здесь же пока скажем только то, что при передаче параметров по значению изменения формальных параметров внутри метода никак не отражаются на фактических параметрах в вызвавшем его методе. При передаче параметра по ссылке противоположная ситуация - все изменения внутри вызываемого метода отражаются на фактическом параметре.

Передача значений в функцию

В Java есть методы, которые могут ничего не возвращать. У этих методов тип void.

```
static void fillArray(int[] ar, int maxRange){
        // Метод fillArray уже с двумя параметрами.
        for(int i = 0; i < ar.length; i++){
            ar[i] = (int) (Math.random() * (maxRange + 1));
            System.out.print("\t" + ar[i]);
        }
    } // Конец метода fillArray
```

Функция, которая ничего не возвращает - это аналог понятия “процедура” в Паскале.

19

Упражнение 1.9.3

Напишите метод для ввода массива данных, метод принимает длинну массива, а возвращает массив. Заголовок метода:

```
static int[] readIntArray(int length)
```

Решение упражнения:

```
static int[] readIntArray(int length){ // Метод readIntArray
    Scaner in = new Scaner(System.in); // Создание объекта Scaner
    int[] ar = new int[length]; // Создание массива нужной длинны
    for(int i = 0;i < ar.length;i++){// Идем циклом по массиву
        ar[i] = in.nextInt(); // Считываем очередное целое число
    }
    return ar;
} // Конец метода readIntArray
```


20

Область видимости переменных

Как вы наверно заметили, внутри каждого метода помимо формальных параметров могут определяться свои переменные, их называют **локальными переменными**. Они “живут” только внутри метода, где они были объявлены и к ним можно получить доступ извне только через уже известные нам способы получения значений из метода: либо через возвращаемый параметр либо через оператор **return**.

Более того, в `Java` **область видимости переменной** - та область откуда переменной можно пользоваться, декларируется (определяется) блоком. Если вы определили переменную внутри цикла - она видна только внутри цикла, внутри конструкции **`if`** - только в ней. Если переменную определить вне любых блоков, но внутри фигурных скобок метода, значит ее область видимости - это весь метод. Переменные, определенные на уровне класса видны всем методам. Общее правило: переменные определенные внутри блока фигурных скобок - видны всем на этом уровне и во всех вложенных блоках.

```
class Test {
    int a; /* a и b глобальные переменные. Их видимость в рамках класса*/
    int b;

    static void abobaMethod(int a, int b) {
        int aboba = a; *aboba u bebra — локальные переменные. Их видимость в рамках метода
*/
        int bebra = b;
    }
}
```

Схема написания метода:

```
[модификаторы] <тип возвращаемого значения> <название_метода> ([<параметры>]){
    <тело метода>
}
```

Метод в Java — это комплекс выражений, совокупность которых позволяет выполнить определенную операцию. Так, например, при вызове метода System.out.println(), система выполняет ряд команд для выведения сообщения на консоль.

Параметры — это входящие значения, которые передаются в метод.

void — это обозначение того, что метод (функция) не возвращает значение.

Глобальная переменная — переменная, которую можно использовать в рамках класса и даже во вне (при правильном модификаторе)

Локальная переменная — переменная с видимостью в рамках метода. Объявляется внутри метода и действует до конца метода.

Итоги

- Мы познакомились с одним из самых базовых инструментов в программировании - функциями (или в ООП - методами). Мы научились создавать методы класса, вызывать их, передавать им данные и принимать данные от них.
- Функции позволяют разбивать задачу на подзадачи. Это очень похоже на то, как человек мыслит. Если необходимо решить какую то проблему, человек не может сразу продумать все в деталях, мы сначала для себя определяем укрупненно подзадачи, а потом уже реализуем каждую. Научившись работать с функциями, мы мыслим не конкретными коснтрукциями языка (переменными, циклами и условиями), а понятиями более высокого уровня, что гораздо ближе к обычной словесной постановке задач. Например, “найти максимум в наборе данных”, “вывести данные на экран”, “ввести данные от пользователя”.
- Т.е. функции позволяют не только повторно использовать код, но и проектировать программы на более высоком уровне абстракции. Кроме того, способность вызывать из метода другие методы, позволяет порождать иерархию абстракций программы, а возможность вызывать из метода самого себя (о чем будет рассказано в дальнейшем), позволяет использовать рекурсивный подход, что зачастую упрощает алгоритм.

1. Класс содержит переменную массив int[]{0, 1, 2, 3 … 10}
Создать метод вывода чисел в обратном порядке. Т.е выводим с 10 до 0.

2. Создание аккаунта. (ИСПОЛЬЗУЙТЕ МЕТОДЫ ДЛЯ РАЗЛИЧНЫХ ДЕЙСТВИЙ)

1. Создайте возможность войти или зарегистрироваться. Создайте поле логина, пароля и две кнопки (регистрации и авторизации)

2. Создайте возможность ввести логин и пароль. Проверьте логин на пустоту. Проверьте пароль. Он должен содержать как минимум 4 символа.

3. Сохраните логин и пароль.

4. Сделайте вход. На входе надо ввести логин и пароль. Проверьте его с существующими данными.

Если данные верны, сделайте вывод сообщения («Вход выполнен успешно»), иначе («Неправильный логин и пароль.»). Отдельно проверяйте логин и пароль.

Спасибо!

SAMSUNG