1.8. Методы

> Привет!
> На связи шпаргалка урока 1.8. Методы, в которой вы найдете материал урока. Прежде чем приступить к выполнению домашнего задания, изучите шпаргалку.
> **Время прочтения**: 20 минут.

Методы

Метод — блок кода, который выполняет определенную функцию и позволяет себя переиспользовать в нескольких местах без необходимости снова и снова писать один и тот же код.

Вы могли сталкиваться с другими терминами (в школе на информатике, например), которые имеют схожий смысл. Например, функция, подпрограмма или процедура.

Безусловно, все эти слова означают примерно одно и то же, но дьявол в деталях.

Метод принадлежит какому-то объекту, т. е. может быть объявлен исключительно внутри скобок какого-то из классов.

Подпрограммы (процедуры) и функции не имеют такого ограничения и могут существовать сами по себе.

Java является объектным языком, и функционал «многоразовых» блоков кода здесь передан именно методам.

Назначение (цель) методов

Все методы в Java должны отвечать за одну определенную функцию.

Например, метод main должен запустить выполнение программы.

Он это делает путем вызова других методов, и логику в себе содержать не должен.

Следовательно, никаких вычислений в этом методе не допускается.

Его единственная задача — запуск приложения.

Возьмем другой метод, который нам тоже хорошо известен, а именно метод *println*. **System.out** — объект, которому принадлежит метод println, мы пока эту часть опустим до урока по объектам.

Единственная задача метода *println* — получить от вас данные (строка, число, символ и т. д.) и вывести эти данные в консоль операционной системы.

Если вы захотите написать свой метод, он тоже должен выполнять строго одну функцию.

Допустим, вам нужно написать приложение, которое осуществляет сбор данных от пользователя и регистрирует вас на сайте. Допустим, ВКонтакте.

Вы должны были бы написать метод, который осуществляет регистрацию и занимается только этим. Так как в Java принято декомпозировать задачи, т. е. разбивать большие задачи на маленькие части и создавать для них небольшие методы, а уже затем их вызывать в «большом брате», который бы и регистрировал пользователя.

Представим, как бы это выглядело на псевдокоде:

```
метод зарегистрироватьПользователя() {
  Строка никНейм = прочитатьЛогин();
  Строка пароль = прочитатьПароль();
  булеан успех = сохранитьВБазуДанных(никНейм, пароль);
  если (успех) {
поздравитьПользователя(никНейм);
} иначе {
  сообщитьОбОшибке();
}
}
```

Как мы видим, метод регистрирует пользователя путем вызова других методов.

Подобное решение (когда мы делим задачу на подзадачи (на маленькие методы), а не пишем весь код в одном методе) позволяет нам в дальнейшем поздравлять пользователя или сохранять данные в базу не только в случае регистрации, но и в целом в любом месте нашего приложения.

Расположение методов

Методы должны располагаться внутри фигурных скобок класса, параллельно (на одном уровне, а не внутри или где-то еще) другим методам.

Вызов метода

Чтобы в программе вызвать метод, нужно написать:

имя метода ( );

Например:

printSeparator ( );

Т. е. структура вашего класса при написании методов должна выглядеть следующим образом:

```
// Создаем файл Homework.java и объявляем в нем класс Homework
class Homework {

  // Внутри скобок класса пишем наш метод main
  public static void main(String[] args) {
      // Здесь пишем код и вызываем другие методы

      // Здесь вызываем метод task1, который объявлен ниже
        task1();

      // А здесь вызываем метод task2, который тоже объявлен ниже
        task2();
    }

  // Внутри скобок класса пишем наш самописный метод, который будет выполнять определенную функцию
  //В данном случае он группирует в себе код по первой задаче ДЗ
  public static void task1() {
      // Здесь пишем код первого задания
  }

  // Внутри скобок класса пишем еще один самописный метод, который будет выполнять определенную функцию
  //В данном случае он группирует в себе код по второй задаче ДЗ
  public static void task2() {
      // Здесь пишем код второго задания
  }
}
```

Как мы видим, чтобы вызвать метод, нужно написать его имя и поставить скобки.

Если параметров у метода нет (про них вы узнаете в этом материале дальше), скобки будут пустыми. Если при написании метода вы потребовали передать параметры, в скобках должны быть значения типов, которые вы указали в качестве параметров при объявлении метода.

Разберем сигнатуру (объявление) метода по частям

Наш с вами частый гость из домашнего задания, метод main, является отличным примером для разбора, так как содержит в себе практически всё, что может быть в объявлении метода.

Разберем сигнатуру по частям.

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

      // блок кода

}
```

Идем по порядку.

Первым стоит слово public.

public — модификатор доступа. Он определяет «места», откуда можно вызвать метод.

public является самым открытым, т. е. метод, который имеет данный модификатор, будет доступен для вызова в любом другом классе программы.

▼ **Также доступны.**

private — метод будет доступен только в том классе/файле, где был объявлен.

package-private или **default** (он же отсутствующий) — метод будет доступен в том пакете, где объявлен. Т. е. вызвать его смогут все классы, которые имеют тот же package (находятся в той же папке).

protected — метод будет доступен в пакете, а также наследникам класса, где объявлен метод (даже если они в других пакетах).

public — метод доступен из любых других мест, будь то другие пакеты, другие файлы и т. д.

О модификаторах доступа, их назначениях и наследовании вы узнаете в дальнейшем, пока рекомендуется ставить public или не ставить никакой (в этом случае метод получит default) модификатор доступа.

Далее идет слово static.

Методы бывают двух типов: статические и нестатические. Именно это отражает наличие или отсутствие ключевого слова static.

Статические — методы, которые принадлежат классу. Т. е. вам не требуется иметь объект для его вызова. Пример: Arrays.toString() или любой другой метод сущности Arrays.

Как мы помним наш опыт использования методов Arrays, нам не требуется никаких дополнительных манипуляций. Мы просто пишем слово Arrays, что является классом, а затем у этого класса вызываем нужный метод. По сути, это просто код без привязки к какому-то объекту, который просто сгруппирован внутри Arrays, а все необходимые данные мы передаем ему в скобках в виде параметра метода.

Нестатические — методы, которые принадлежат объекту. Для их использования нам нужно самим создать объект, инициализировать его и вызвать у этого объекта метод.

Обычно эти методы работают с данными конкретного объекта, который их вызывает.

Например, когда мы создаем строку и вызываем метод toUpperCase, этот метод срабатывает именно с той строкой, которая вызвала этот метод. Нам не требуется передавать ему какие-то параметры в скобках, так как он уже имеет все данные из объекта, который его вызывает.

Следующим идет слово **void**.

Это слово обозначает тип возвращаемого значения методом, т. е. результат метода.

void сообщает, что не будет возвращено ничего, т. е. метод выполняет какой-то код и не отчитывается о своем выполнении.

Если бы мы написали int, мы были бы обязаны вернуть числовой результат.

Например, нам нужно написать метод, который считает сумму двух целых чисел.

В данной ситуации нам бы потребовалось вместо void указать int.

Более подробно о возвращаемом типе поговорим в дальнейшем.

Следующим словом стоит **main**.

Это имя метода, оно может быть любым, но начинать его принято с глагола, а само имя должно описывать происходящее в методе.

Например, calcualteSum, registerUser или findMaxValue.

Только main должен оставаться неизменным, так как именно по такой сигнатуре Java ищет запускающий метод.

Затем открываются скобки, и мы видим String[] args.

Это параметр метода. При вызове метода main (его вызываете не вы, а Java) ему передается массив строк, который затем присваивается переменной args, объявленной в скобках.

Т. е. чтобы выполнить код из метод main, Java обязана сформировать массив и вызвать метод main, передав ему сформированный массив в качестве параметра. Это выглядит так,

как если бы вы сами создали массив строк arr и вызвали main(arr).

Параметры метода

Вспомним знакомые нам методы.

Например, упомянутый выше метод main.

У него в скобках сигнатуры (при объявлении) указана такая вещь, как String[] args.

Это значит, что при вызове метода вы обязаны передать ему в скобки массив строк, который внутри метода будет присвоен переменной с именем args.

Так как мы изучили массивы, класс Arrays должен быть нам знаком.

Возьмем его метод equals, который имеет два параметра-массива (мы должны передать ссылки на два массива при вызове метода equals в его круглые скобки), которые внутри метода сравниваются.

Если мы не передадим в метод два массива, IDEA выделит нам его красным, так как такого метода (без параметров) не существует.

Теперь сами попробуем реализовать метод с параметрами.

Вспомним нашу задачу на подсчет суммы элементов массива.

Мы создавали массив внутри блока кода, но давайте реализуем этот блок кода в виде метода да еще и так, чтобы массив нам приходил извне при вызове метода.

Результатом работы метода должна быть печать суммы элементов в консоль.

Создаем метод по примеру метода main. Так как вы уже знаете, что означает каждое из его слов, это не должно составить проблем.

Пусть это будет public static void метод с именем calculateSum и одним параметром int[] по имени arr.

Код будет выглядеть следующим образом:

```
public static void calculateSum(int[] arr) {
  int sum = 0;
  for (int i : arr) {
    sum += i;
  }
  System.out.println(“Сумма элементов массива: “ + sum);
}
```

В примере выше мы создали метод, который получил в виде источника данных массив откуда-то извне, присвоил его переменной с именем arr, посчитал его сумму и вывел результат в консоль.

Передача значения в метод

В метод можно передавать значение только одной переменной.

```
имя метода (значение переменной);
```

Например:

```
printSeparator (i);
```

Теперь представим, что нам нужно подсчитать сумму не массива, а двух элементов и вывести ее в консоль. Для этого нам потребуется два параметра, которые мы и будем суммировать.

Пусть это будут параметры типа int с именами a и b.

В этой ситуации метод должен выглядеть следующим образом:

```
public static void calculateSum(int a, int b) {
  int sum = a + b;
  System.out.println(“Сумма элементов: “ + sum);
}
```

Ну и для того, чтобы был пример не только с параметрами типов int и int[], сделаем что-нибудь со строкой. Например, напишем метод, который получает на вход имя, а затем печатает приветствие:

```
public static void printGreetings(String name) {
  System.out.println(“Привет, “ + name);
}
```

Давайте протестируем наши написанные выше методы.

Для этого нам нужно вызвать их в методе main:

```
public static void main (String[] args) {
  int num1 = 5;
  int num2 = 6;
  int[] arr = {1, 2, 3};
  printGreetings(“Иван”);
calculateSum(num1, num2);
calculateSum(arr);
}
```

Результат будет:

Привет, Иван

Сумма элементов: 11

Сумма элементов массива: 6

Следует заметить, что имена переменных при передаче в метод не имеют значения. Внутри они в любом случае будут присвоены тем переменным, которые объявлены в скобках при объявлении метода.

Теперь рассмотрим, как Java разобралась, какой из двух методов нужно вызывать.

Сигнатура метода

Несмотря на то, что ранее под сигнатурой называлось всё, что пишется при объявлении метода (включая public и static), сигнатурой являются только две части в объявлении метода.

Это имя и параметры.

Т. е. Java в коде выше проанализировала сигнатуры двух методов с одним именем и вызвала тот, который соответствует двум параметрам типа int.

Следовательно, даже имея одинаковые названия, методы не являются идентичными друг другу.

Такими они являются исключительно в случаях полного совпадения сигнатур.

Именно поэтому были вызваны корректные методы в зависимости от переданных параметров.

Результат метода

Как упоминалось ранее, слово, указанное перед именем метода, является возвращаемым значением.

Методы делятся на два типа:

1. Возвращающие результат (имеют перед именем метода указание типа).
2. Не возвращающие результат (имеют перед именем метода void).

Вернемся к знакомым нам ранее методам calculateSum.

В нашей реализации перед именем стояло слово void.

Это значит, что метод выполняется и никак не сигнализирует остальному миру о своем завершении.

Давайте перепишем метод так, чтобы он не печатал сумму элементов, а возвращал ее в то место, где наш метод был вызван.

Это позволит нам самим решать, что с ней делать.

Так как наш метод назван calculateSum, а не calculateAndPrintSum, следовательно, он должен считать сумму, а не производить с ней какие-то дополнительные манипуляции (как печать в нашем случае).

Давайте переделаем методы так, чтобы они возвращали результат, а не печатали его.

```
public static int calculateSum(int[] arr) {
  int sum = 0;
  for (int i : arr) {
    sum += i;
  }
  return sum;
}

public static int calculateSum(int a, int b) {
  int sum = a + b;
return sum; // Можно избавиться от промежуточной переменной и написать просто return a + b;
}
```

Давайте протестируем наши измененные версии методов.

Для этого нам нужно снова вызвать их в методе main:

```
public static void main (String[] args) {
  int num1 = 5;
  int num2 = 6;
  int[] arr = {1, 2, 3};
int sumOfAAndB = calculateSum(num1, num2);
int sumOfArrayElements = calculateSum(arr);
System.out.println(sumOfAAndB);
System.out.println(sumOfArrayElements);
}
```

Вывод:

11

6

Таким образом можно возвращать любые значения любого типа (но этот любой тип должен быть указан перед именем метода в сигнатуре).

Как вы могли заметить, для возврата значения нужно написать ключевое слово return и указать после него результат.

Оператор return

У оператора return две функции:

1. Прервать выполнение метода (эту функцию оператор выполняет всегда).
2. Вернуть результат в то место (в ту строку), где метод был вызван (эту функцию оператор выполняет в тех случаях, когда возвращаемое значение).

Рассмотрим два примера использования оператора return.

В первом примере мы добавим функционала задаче на работу с переменной clientOS, что встречалась в ДЗ по условным операторам.

Перед нами стоит необходимость написать метод, который присвоит этой самой переменной clientOS значение 0 или 1, соответствующее системе мобильного устройства пользователя.

На вход имеем название ОС, на выход должны подать 0 или 1 в зависимости от ОС.

Напомним, что 0 — iOS, 1 — Android.

```
String osName = “iOS”;
int clientOS = getClientOS(osName);

В коде выше мы подготовили данные и вызвали метод. Теперь его нужно реализовать.

public static int getClientOS(String name) {
  if (name.equals(“iOS”)) {
    return 0;
  } else {
    return 1;
  }

  /* Но так как у нас всего два возможных варианта,
 мы можем в целом отказаться от блока else и написать просто return 1;
после блока if. */
}
```

В данном примере мы увидели, как следует использовать return в условных операторах. Важно запомнить, что return должен быть во всех возможных развитиях событий, которые мы покрываем методом.

Т. е. если мы не напишем return вне блока if (или в блоке else), IDEA не даст нам выполнить такой код, так как есть ситуация (если блок if не сработает по условию), когда мы ничего не возвращаем. Это недопустимо.

Теперь пример использования return в void-методе.

Вспомним то же ДЗ по условным операторам, а конкретно задачу с определением сезона по номеру месяца.

Так как теперь мы сами умеем писать методы и передавать в них параметры извне, давайте переработаем решение этого задания. В дополнение напишем проверку полученного в параметрах числа, чтобы вместо сравнения его со всеми кейсами и в итоге вызова кода из блока default мы могли сразу проверить число и не запускать switch, если данные пришли некорректные.

```
public static void printSeason(int monthNumber) {
  if (monthNumber <= 0 || monthNumber > 12) {
    System.out.println(“Некорректное значение месяца”);
    return;
  }

  switch (monthNumber) {
    case 12:
    case 1:
    case 2:
      System.out.println("Зима");
      break;
    case 3:
    case 4:
    case 5:
      System.out.println("Весна");
      break;
    case 6:
    case 7:
    case 8:
      System.out.println("Лето");
      break;
    case 9:
    case 10:
    case 11:
      System.out.println("Осень");
  }
}
```

В примере выше мы применили оператор return, который в случае некорректного значения прервет метод и не даст коду далее выполниться.

▼ **Следует запомнить**

В случае возвращаемого результата, отличного от void, использование в коде строки “return результат;” обязательно для всех возможных вариантов развития событий.

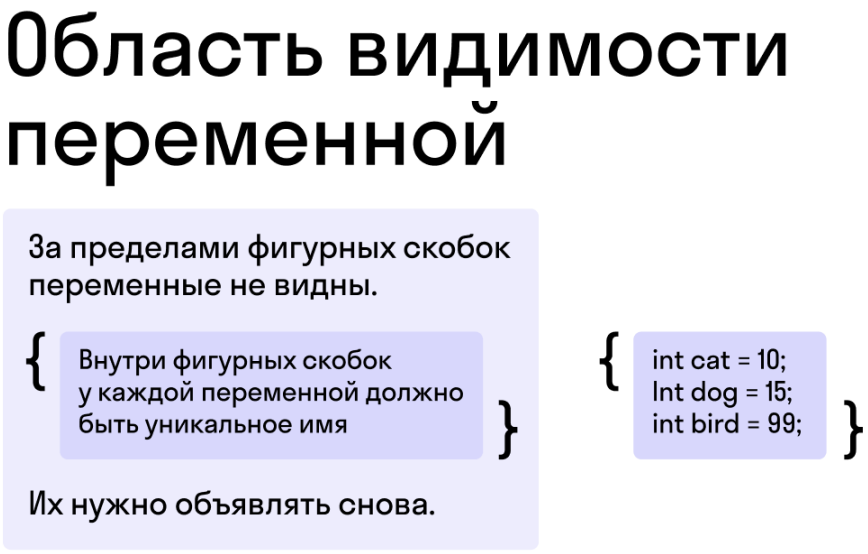
Т. е. если имеет место блок if-else, где в if возвращается результат, должен быть предусмотрен возврат результата, если блок if не будет выполнен.

В случае с методом, возвращаемый тип которого указан как void, присутствие оператора return в коде возможно для прерывания метода, но не обязательно.

Область видимости параметров (или где мы можем их использовать)

Как мы уже узнали, при создании метода с параметрами мы создаем переменные, которым присваиваем то, что было передано в скобки при вызове метода.

Эти переменные можно использовать только в скобках метода.



Например:

```
int calculateSum(int a, int b) {
  return a + b; // Переменные можно использовать
} // После этой скобки переменные a и b уже не существуют
System.out.println(a); // Здесь переменной a уже не существует
```

Поведение разных типов в параметрах метода

Как мы с вами уже слышали, в Java есть примитивные и объектные (ссылочные) типы данных.

Если говорить простым языком, то примитивом является простое значение.

Например, число или символ.

Эти типы не имеют своего поведения, возможностей и лишь используются как тип-значение.

Но есть ссылочные типы (или объекты), которые значительно сложнее. Они не просто хранят какое-то число или символ. Зачастую они имеют какие-то свойства (как length у массивов) и методы (как toUpperCase или replace у строк).

Именно за счет того, что мы не можем знать точного размера элемента (в отличие от типов, которые хранят одно значение, т. е. примитивов), а также не знаем, влезет ли он в нашу ячейку стека, мы используем специальные ссылки, которые хранят в себе исключительно адрес в памяти, где именно лежит наш объект.

Благодаря этим ссылкам мы можем взаимодействовать с объектом, который затерян где-то в глубинах памяти нашей ОС.

Т. е. когда мы создаем int i = 5; мы создаем в памяти число 5 и сразу закрепляем его за переменной i.

Когда мы создаем объект (например, строку или массив), мы пишем int[] arr = new int[20];

Здесь то, что пишется до оператора =, является созданием ссылки, которая пока ссылается на пустоту (null).

Далее мы, используя ключевое слово new, создаем сам объект, которому как раз присваиваем этот новый массив.

Теперь вернемся к нашей теме :)

Когда мы вызываем метод и передаем что-то ему в скобки, Java копирует то, что мы указали в скобках при вызове.

 Запомните эту мантру: **параметры в Java передаются по значению.**

Этот вопрос часто спрашивают на на собеседованиях.

Если это тип-значение (примитив), то копируется само значение.

Если же это ссылочный тип (объект), то копируется ссылка.

Вы уже могли догадаться, что если копируется ссылка, ссылающаяся на определенный объект, то в нашей системе будут две ссылки, которые ссылаются на один и тот же объект.

Т. е. если мы модифицируем примитив внутри метода, это значение вне метода не поменяется, но если мы модифицируем в методе объект, значение объекта вне метода поменяется тоже.

Именно так работает передача данных в метод.

Давайте напишем метод, который модернизирует два типа данных.

Первый будет примитивным, второй — ссылочным.

```
public static void main(String[] args) {
  int a = 1;
  int[] arr = {1, 2, 3};
  changeValues(a, arr);
  System.out.println(a);
  System.out.println(Arrays.toString(arr));
}

public static void changeValues(int a, int[] arr2) {
  a = 5;
  arr2[0] = 5;
}
```

Вывод:

1

[5, 2, 3]

▼ **Закрепим**

В Java все параметры (аргументы) методов передаются по значению.

Если аргументом является тип-значение (примитив), то передается копия этого значения.

Если аргументом является объект, то передается копия его ссылки.

Т. е. если вы в примере выше укажете, что arr теперь относится к новому массиву, а затем измените массив по этой переменной, изначальный массив (который вы передавали в метод) изменяться не будет, так как ссылка arr2 теперь ссылается на новый массив, созданный внутри метода changeValues, когда ссылка arr всё еще хранит в себе адрес на изначальный массив.

```
public static void changeValues(int a, int[] arr2) {
  a = 5;
  arr2 = new int[10];
  arr2[0] = 5;
}
```

Вывод:
1
[1, 2, 3]

▼ Подсказка к решению первой задачи в домашнем задании

Представим, что нам нужно реализовать метод, который проверяет, является ли число четным или нечетным и выводит результат проверки в консоль.

Если число четное, то должно быть выведено “Число четное”. Если число нечетное, то, соответственно: “Число нечетное”.

```
public static void printIsEvenNumber(int number) {
    boolean evenNumber = isEvenNumber(number);
    printIsEvenNumberResult(number, evenNumber);
}

private static boolean isEvenNumber(int number) {
    return number % 2 == 0;
}

private static void printIsEvenNumberResult(int number, boolean evenNumber) {
    if (evenNumber) {
        System.out.println("Число " + number + " четное");
    } else {
        System.out.println("Число " + number + " нечетное");
    }
}
```

> Шпаргалка урока в PDF-формате:
>
> https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c278c68f-53c8-441e-a445-c9ad03268244/1.8._.pdf