7.1 Общее представление о методах и синтаксис определения

Методы – конструкции, содержащие собой набор операторов, которые выполняют определенные действия.

В **объектно-ориентированном программировании** метод — это именованный блок кода, который объявляется внутри класса и может быть использован многократно. Если вы знакомы с процедурным программированием (Pascal, Basic), вспомните, что такое функция — по принципу работы у неё и метода много общего.

Методы в Java — это законченная последовательность действий (инструкций), направленных на решение отдельной задачи. По сути, это функции (они же процедуры, подпрограммы) более ранних, **не ООП языков**. Только эти функции являются **членами классов** и для различия с обычными функциями, согласно терминологии объектно-ориентированного программирования, называются методами.

Методы могут возвращать или не возвращать значения, могут вызываться с указанием параметров или без. **Тип возвращаемых данных указывают при объявлении метода — перед его именем**.

Метод в Java или метод Java - это набор операторов, которые выполняют некоторую конкретную задачу и возвращают результат вызывающей стороне. Метод Java может выполнять некоторую конкретную задачу, ничего не возвращая. Методы в Java позволяют нам **повторно** использовать код без повторного ввода кода. **В Java каждый метод должен быть частью некоторого класса, который отличается от таких языков, как C, C ++ и Python**.

То, каким образом описывать методы, приводится в самом начале пути изучения Java. Например, в официальном tutorial от Oracle: «Defining Methods»[1].

- Каждый метод имеет сигнатуру. Сигнатура состоит из имени метода и его входных параметров;
- Для методов должно быть указан тип возвращаемого значения (return type);
- Тип возвращаемого значения не входит в сигнатуру метода.

7.1.1Типы методов в Java

В Java есть два типа методов:

1. Предопределенные методы: в Java предопределенные методы - это методы, которые уже определены в библиотеках классов Java и называются предопределенными методами. Он также известен как метод **стандартной библиотеки или** встроенный метод. Мы можем напрямую использовать эти методы, просто вызывая их в программе в любой момент.

2. Определяемый пользователем метод: Метод, написанный пользователем или программистом, известен как **определяемый пользователем** метод. Эти методы модифицируются в соответствии с требованиями.

7.1.2 Синтаксис определения метода

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры]){
   // тело метода
}
```

Модификаторы и параметры необязательны!

```
public static void main(String[] args) {
   System.out.println("привет мир!");
}
```

Ключевые слова **public** и **static** являются модификаторами. Далее идет тип возвращаемого значения. Ключевое слово void указывает на то, что метод ничего не возвращает.

Затем идут название метода - main и в скобках параметры метода - String[] args. И в фигурные скобки заключено тело метода - все действия, которые он выполняет.

Рассмотрим следующую программу:

```
public class Program{
   public static void main (String args[]){
   }
   void hello(){
      System.out.println("Hello");
   }
   void welcome(){
      System.out.println("Welcome to Java 10");
```

[1] https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

```
    }
}
```

Программа содержит два «немых» метода. Они не вызываются! Вызов метода осуществляется в форме:

имя_метода(аргументы);

Корректная запись, вызывающая методы, выглядит так:

```
public class Program{
   public static void main (String args[]){
       hello();
       welcome();
       welcome();
   }
   static void hello(){
      System.out.println("Hello");
   }
   static void welcome(){

      System.out.println("Welcome to Java 10");
   }
}
```

Следует отметить, что чтобы **вызвать в методе main другие методы, которые определены в одном классе с методом main, они должны иметь модификатор static**.

В приведённом коде проявляется одно из преимуществ методов: мы можем вынести некоторые общие действия в отдельный метод и затем вызывать многократно их в различных местах программы.

7.2 У методов параметры или аргументы? Есть ли в java функции?

Аргументы – у функции, параметры – у методов. Аргументы – это конкретные значения, подставляемые на место параметров.

Есть ли разница между **методом** и **функцией**? Или это просто два разных названия одного и того же?

Функция и метод - это некоторый фрагмент кода, который можно вызвать по имени и, опционально, передать ему какие-либо параметры. С этой точки зрения различий никаких нет.

Функция обычно включает в себя все необходимое для своей работы, а взаимодействие с «внешним миром» осуществляет через **входные** и **выходные** параметры, а также возвращаемое значение (использование глобальных переменных скорее исключение, а чаще просто грубая ошибка). Кроме того, **функция является самостоятельной единицей** и может быть вызвана в любом месте программы практически без ограничений.

Метод же для своей работы **может** использовать **поля объекта и/или класса**, в котором определен, напрямую, без необходимости передавать их во входных параметрах. Это похоже на использование глобальных переменных в функциях, но в отличие от глобальных переменных,

метод может получать **прямой доступ только к членам класса**. **Метод привязан к классу, если он статический, или объекту и может быть вызван только через этот класс или объект**.

Из-за этих, **весьма существенных**, отличий метод и получил отдельный термин, в некотором смысле сокращение от "**Метод (способ) взаимодействия с классом или объектом**".

В целом, называть метод функцией не является ошибкой (**обратное неверно**), главное понимать концептуальные отличия этих терминов.

Это касается любых языков программирования, поддерживающих концепцию объектно-ориентированного программирования.

Поэтому в Java нет функций! Но можно организовать элементы функционального программирования!

Действительно, в Java присутствуют только классы, объекты классов и методы. Однако также присутствуют анонимные классы, то есть классы, у которых не обозначено имя. Их можно объявить в любом методе и интерфейсе. Если их интерфейс с анонимным классом объявить в каком-либо методе, **тогда с подобным интерфейсом можно делать то же самое, что и с обычной функцией в функциональных языках**. Благодаря такому свойству анонимных интерфейсов в Java возможно реализовать функциональное программирование[2].

7.2.1 Примеры применения методов

Поскольку оба вышеприведённых в пункте 7.1.2 метода не имеют никаких параметров, то после их названия при вызове ставятся пустые скобки.

С помощью параметров можно передавать различные данные в методы:

```
static void sum(int x, int y){
   int z = x + y;
   System.out.println(z);
}
**************************************************************
public class Program{
   public static void main (String args[]){
      int a = 6;
      int b = 8;
      sum(a, b);  // 14
      sum(3, a);  // 9
      sum(5, 23); // 28
   }
   static void sum(int x, int y){
      int z = x + y;
      System.out.println(z);
   }
}
**************************************************************
```

Поскольку метод sum принимает два значения типа int, то на место параметров надо передать два значения типа int. Это могут быть и числовые литералы, и переменные типов данных, которые представляют тип int или могут быть автоматически преобразованы в тип int.

[2] https://codernet.ru/articles/js/funkczionalnoe_programmirovanie_v_java_opredelenie_patternyi_i_primenenie/

Значения, которые передаются на место параметров, еще называются аргументами. Значения передаются параметрам по позиции, то есть первый аргумент первому параметру, второй аргумент - второму параметру и так далее.

****

```
public class Program{
   public static void main (String args[]){
      display("Tom", 34);
      display("Bob", 28);
      display("Sam", 23);
   }
   static void display(String name, int age){
      System.out.println(name);
      System.out.println(age);
   }
}
```

Метод **display** принимает два параметра. Первый параметр представляет тип String, а второй - тип int. Поэтому при вызове метода вначале в него надо передать строку, а затем число.

7.2.2 Параметры переменной длины

Метод может принимать параметры переменной длины одного типа. Например, нам надо передать в метод набор чисел и вычислить их сумму, но мы точно не знаем, сколько именно чисел будет передано - 3, 4, 5 или больше. Параметры переменной длины позволяют решить эту задачу:

```
public class Program{
   public static void main (String args[]){
      sum(1, 2, 3);         // 6
      sum(1, 2, 3, 4, 5);    // 15
      sum();               // 0
   }
   static void sum(int ...nums){
      int result =0;
      for(int n: nums)
         result += n;
      System.out.println(result);
   }
}
```

Троеточие перед названием параметра int ...nums указывает на то, что он будет необязательным и будет представлять массив. Мы можем передать в метод sum одно число, несколько чисел, а можем вообще не передавать никаких параметров. Причем, если мы хотим передать **несколько** параметров, то **необязательный параметр** должен указываться в **конце**:

```
public static void main(String[] args) {
   sum("Welcome!", 20,10);
   sum("Hello World!");
}
static void sum(String message, int ...nums){
   System.out.println(message);
   int result =0;
```

```
    for(int x:nums)
        result+=x;
    System.out.println(result);
}
```

7.2.3 Параметры (аргументы) методов: формальные-фактические

Формальный аргумент - *переменная* в вызываемой функции (методе).

Фактический аргумент - конкретное *значение*, присвоенное этой переменной вызывающей программой.

Фактический аргумент может быть константой, переменной или более сложным выражением. Независимо от типа *фактического аргумента* он вначале вычисляется, а затем его величина передается *функции*.

Фактический аргумент - это конкретное *значение*, которое присваивается переменной, называемой *формальным аргументом*.

7.2.4 Передача данных в метод (функцию)

Многие программисты часто путают, какие параметры в Java передаются по значению, а какие по ссылке.

Надо помнить, что для передачи данных в Java используется только метод передачи по значению. Но для повышения кругозора ниже описываются два способа – по значению и по имени (ссылке).

7.2.4.1 Передача по значению

Этот способ передачи данных в подпрограмму является основным и действует по умолчанию во многих языках. Фактический параметр **вычисляется** в вызывающей функции и его значение **передаётся на место формального параметра** в вызываемой функции. **На этом связь между фактическим и формальным параметрами прекращается**.

В качестве фактического параметра можно использовать константу, переменную или более **сложное выражение**. Передача данных по значению пригодна только для простых данных, которые являются входными параметрами.

7.2.4.2 Передача по ссылке

Параметры передаются в функцию (метод) как ссылка (адрес) на **исходную переменную**. Вызываемый метод **не создает свою копию**, а **ссылается** на исходное значение. Следовательно, **изменения, сделанные в вызываемом методе, также будут отражены в исходном значении**.

Java всегда передает параметры по значению!

7.2.4.3 Пример передачи примитивного типа в метод Java

Простой пример передачи данных представлен на рисунке 7.1. Поскольку Java передает параметры по значению, метод obrabotkaData работает с копией data. Следовательно, в исходных данных (в методе main) не произошло никаких изменений.

```java
public class ParamPassing {
    public static void  main(String[] args) {
        int data = 10;
        System.out.println("Данные до обработки в main" + data);
        obrabotkaData(data);
        System.out.println("Данные после обработки в main = " + data);
    }
    private static void obrabotkaData(int data) {
        data =data * 10;
        System.out.println("Данные после обработки в функции" + data);
    }
}
```

```
Run: ParamPassing
C:\Users\OVI\.jdks\openjdk-19\bin\java.exe "-javaagent:C:\Program Files\JetBr
Данные до обработки в main10
Данные после обработки в функции100
Данные после обработки в main = 10
```

Рисунок 7.1. Пример передачи данных в метод

7.2.4.4 Пример передачи массива в метод Java

(https://www.tutorialspoint.com/compile_java_online.php - онлайновый исполнитель кода java)

```java
public static void main(String[] args) { // Программа поэлементно выводит массив в консоль
    double[] myList = {1.9, 2.9, 3.4, 3.5}; // Создание массива
    // Вывести массив на экран
    System.out.println("Печать массива в  main построчно");
    for (double element: myList) {    // Печать массива в main
        System.out.println("element=" + element);
    }
    printArray(myList);  // Вызов метода с передачей массива myList в этот метод
}
public static void printArray(double[] massiv) { // Метод, реализующий печать массива
    System.out.println("Печать 1-го массива в методе printArray");
    for (double element: massiv) {    // Печать массива в main поэлементно
        System.out.println(element);
    }
    int dlina = massiv.length;
    // System.out.println("dlina=" + dlina);
    double[] myArray = new double[dlina];
    System.out.println("Печать 2-го массива в методе printArray");
    for (int j = 0; j <= dlina-1; j++) {
        System.out.println("j=" + j);
        System.out.println("massiv[j]=" + massiv[j]);
    }
    // Получение массива в обратном порядке переданному в метод
    int k = 0;
    for (int l = dlina-1; l >= 0 ; l=l-1) {
        // System.out.println("l= " + l);
        System.out.println(massiv[l] + " ");
        myArray[k] = massiv[l];
        k++;
    for (double value : myArray) {
        System.out.println( "value = " + value);
    }
}
}
```

Рисунок 7.2 Передача массива в метод

7.3 Ключевое слово оператор return в Java

7.3.1 Использование для возвращения значения из метода

7.3.1.1 Получение сообщения

Метод main не может возвращать значения. С помощью return можно вынести получение результата в отдельный метод. Ниже приводим пример получения сообщения из отдельного метода:

```
public class HelloWorld {
   public static void main(String []args) {
      System.out.println(getHelloMessage());
   }
   public static String getHelloMessage() {
      return "Hello World";
   }
}
```

При помощи ключевого слова return указывается возвращаемое значение, которое использовали далее в методе **println** основной программы.

В описании (определении) метода **getHelloMessage** указано, что он вернёт нам **String**. Это позволяет компилятору проверить, что действия метода согласуются с тем, каким образом он объявлен.

Естественно, тип возвращаемого значения, указанного в определении метода, может быть более широким чем тип возвращаемого из кода значения, т.е. главное, чтобы типы **приводились друг к другу**. В противном случае мы получим ошибку во время компиляции: «error: incompatible types».

7.3.1.2 Получение значения

```
public class Program{
   public static void main (String args[]){
      int x = sum(1, 2, 3);
      int y = sum(1, 4, 9);
      System.out.println(x);  // 6
      System.out.println(y);  // 14
   }
   static int sum(int a, int b, int c){
      return a + b + c;
   }
}
```

В методе в качестве типа возвращаемого значения вместо **void** используется **любой другой тип**.

В данном случае метод sum возвращает значение типа int, поэтому этот тип указывается перед названием метода. Причем если в качестве возвращаемого типа для метода определен любой другой, отличный от void, то метод **обязательно должен использовать оператор return** для возвращения значения.

При этом возвращаемое значение всегда должно иметь тот же тип, что значится в определении функции. И если функция возвращает значение типа int, то после оператора return стоит целочисленное значение, которое является объектом типа int. Как в данном случае это сумма значений параметров метода.

7.3.1.3 Использование для возврата нескольких значений из метода

Метод может использовать несколько вызовов оператора return для возвращения разных значений в зависимости от некоторых условий:

```
public class Program{
   public static void main (String args[]){
      System.out.println(daytime(7));     // Good morning
      System.out.println(daytime(13));    // Good after noon
      System.out.println(daytime(18));    // Good evening
      System.out.println(daytime(2));     // Good night
   }

   static String daytime(int hour){
      if (hour >24 || hour < 0)
         return "Invalid data";
      else if(hour > 21 || hour < 6)
         return "Good night";
      else if(hour >= 15)
         return "Good evening";
      else if(hour >= 11)
         return "Good after noon";
      else
         return "Good morning";
   }
}
```

7.3.1.4 Использование для выхода из метода

Оператор return применяется *не только* для *возвращения* значения из метода, но и для выхода из метода. В подобном качестве оператор return **применяется в методах, которые ничего не возвращают, то есть имеют тип void**:

```
public class Program{
   public static void main (String args[]){
      daytime(7);     // Good morning
      daytime(13);    // Good after noon
      daytime(32);    //
      daytime(56);    //
      daytime(2);     // Good night
   }
   static void daytime(int hour){
      if (hour >24 || hour < 0)
         return;
      if(hour > 21 || hour < 6)
         System.out.println("Good night");
      else if(hour >= 15)
         System.out.println("Good evening");
      else if(hour >= 11)
         System.out.println("Good after noon");
      else
         System.out.println("Good morning");
   }
}
```

Если переданное в метод datetime значение больше 24 или меньше 0, то просто выходим из метода. Возвращаемое значение после return указывать в этом случае не нужно.

7.4 Перегрузка методов

Механизм, позволяющий использовать методы с **одинаковым именем, но с разными типами и/или количеством параметров** называется **перегрузкой методов**.

Естественно, что и работать эти методы будут по-разному, выдавая различные результаты.

```
public class Program{
   public static void main(String[] args) {
      System.out.println(sum(2, 3));        // 5
      System.out.println(sum(4.5, 3.2));    // 7.7
      System.out.println(sum(4, 3, 7));     // 14
   }
   static int sum(int x, int y){
      return x + y;
   }
   static double sum(double x, double y){
      return x + y;
   }
   static int sum(int x, int y, int z){
      return x + y + z;
   }
}
```

Здесь определено три варианта или три перегрузки метода sum(), но при его вызове в зависимости от типа и количества передаваемых параметров система выберет именно ту версию, которая наиболее подходит.

Подчеркну автоматизм выбора требуемого метода, что обеспечивается анализом (**парсингом**) текста операторов при компиляции кода.

На включение механизма перегрузки методов влияют **количество** и **типы** параметров. **Однако различие в типе возвращаемого значения для перегрузки не имеют никакого значения**.

Например, в следующем случае методы различаются по типу возвращаемого значения:

```
public class Program{
   public static void main(String[] args) {
      System.out.println(sum(2, 3));
      System.out.println(sum(4, 3));
   }
   static int sum(int x, int y){
            return x + y;
   }

   static double sum(int x, int y){
           return x + y;
   }
}
```

Однако перегрузкой это не будет считаться. Более того такая программа некорректна и попросту не скомпилируется, так как метод с одинаковыми количеством и типом параметров определен несколько раз.

7.5 Рекурсивные методы

Рекурсивные методы, то есть, методы, которые вызывают сами себя.

```
static void up_and_down(int n) {
   System.out.println("Уровень вниз " + n);
   if (n < 4) up_and_down(n + 1);
   System.out.println("Уровень вверх " + n);
}
```

Реализовать вызов этой функции и проверить работу.

Когда вызывается какой-либо метод, он помещается в стек вызова методов (чтобы знать, какие методы в каком порядке были вызваны).

Сначала в этом стеке находится метод main(), далее, из него вызывается метод up_and_down(1).

Что происходит дальше? Этот метод выводит на экран строку «Уровень вниз 1» и проверяет условие 1<4 – да, и идет вызов такого же метода, но уже с аргументом n+1=2.

Причем работа первого метода up_and_down(1) и второго up_and_down(2) совершенно независимы друг от друга!

Второй метод также выводит строку «Уровень вниз 2» и проверяет условие 2<4. Оно срабатывает и идет вызов следующего метода up_and_down(3).

Здесь происходит все то же самое. Наконец, когда мы достигаем up_and_down(4), условие 4<4 не срабатывает и дальнейшего вызова метода up_and_down не происходит и выполняется следующая строчка метода, которая выводит строку «Уровень вверх 4».

Что будет дальше? Дальше метод up_and_down(4) завершается и вызов передается методу up_and_down(3) в соответствии со стеком вызова. Но этот метод уже выполнил первые две строчки, поэтому выполняется третья, которая выводит «Уровень вверх 3». Он также завершается, вызов переходит к методу up_and_down(2), он по аналогии выводит третью строку «Уровень вверх 2», передает управление up_and_down(1), он выводит «Уровень вверх 1» и на этом работа рекурсии завершается. Вот так работают рекурсивные методы.

Если бы у нас не было вот этого условия, то рекурсия ушла бы вниз, пока стек методов не переполнился бы (возникла бы ошибка stack overflow), поэтому при реализации рекурсивных методов нужно внимательно относиться к глубине их вызова – она не должна быть большой.

Понятие стека вызовов.

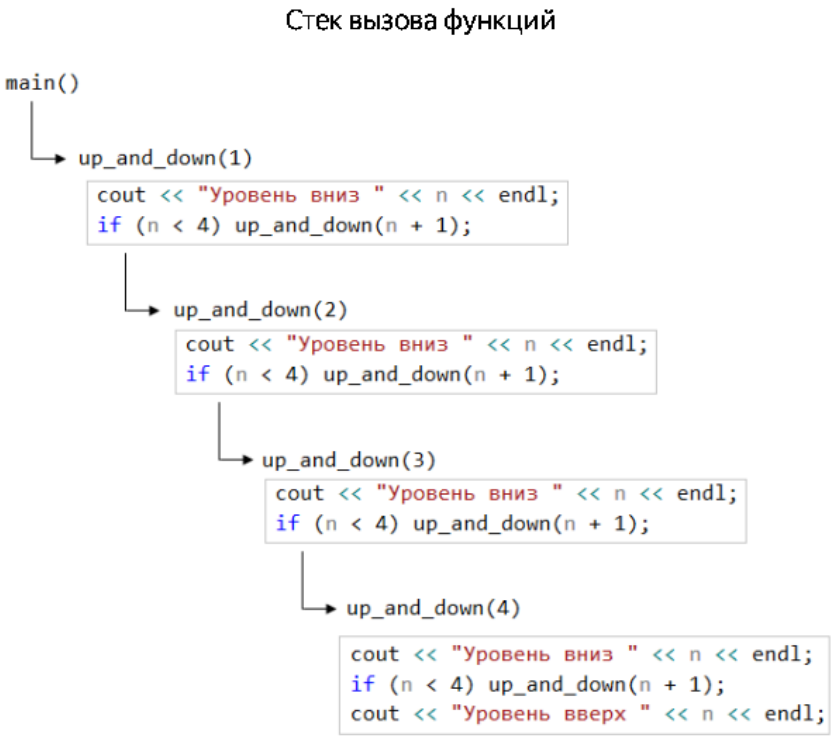

Рисунок 7.3 Схема логики работы стека вызова функций при рекурсивном вызове