Урок 4

Методы

Необходимость и области применения.

Назначение методов

Методы определяют поведение программы. Разумеется, всю логику можно разместить в главном методе main, и программа при этом будет работать. Но если нужно будет повторять действие несколько раз? Если мы не знаем конечное количество вызовов действия? Если будем просто копировать код, при изменении логики придется править его столько раз, сколько создано копий. Можно что-то упустить или ошибиться. А если неизвестно количество повторений, которое зависит от конкретной ситуации в определенное время, копирование кода не поможет.

Именно методы призваны решить данную задачу. Они содержат часть логики, которая может вызываться многократно и при каждом вызове будет работать одинаково. В таком случае изменения логики производятся только в одном месте и доступны для вызова всегда. В программировании есть принципы **DRY** (**don't repeat yourself** – не повторяйся) и **KISS** (**keep it super simple** — не усложняй). Хорошо написанный метод будет соответствовать им: содержать маленький фрагмент функциональности, вызываемый в разных местах программы не десятками строк, а одним выражением.

Методы в Java

Если переменные отвечают за хранение значений, методы хранят набор операций, выполняющих действия. Общее определение методов:

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры]){

    // тело метода

}
```

Модификаторы и параметры необязательны. Условно методы, которые не возвращают значения, называются процедурами. Они имеют ключевое слово **void** в типе возвращаемого значения.

```
static void method1(){

    System.out.println("Я метод 1");

}
```

Если тип отличен от **void**, этот метод должен по результатам работы возвращать в место его вызова значение (при помощи ключевого слова **return**). Такие методы называют функциями. Инструкция с ключевым словом **return**, за которым следует значение, соответствующее указанному возвращаемому типу, будет возвращать это значение вызвавшей метод конструкции (например, математическому выражению). Ключевое слово **return** при этом останавливает выполнение метода. Даже если тип возврата — **void**, инструкцию **return** без значения по-прежнему можно использовать, чтобы завершить выполнение метода. Без ключевого слова **return** метод будет останавливаться по достижении конца блока кода.

```
public static void main(String[] args) {
   int a = getSum();
   System.out.println(a);
}


static int getSum(){
    int x = 2;
    int y = 3;
    return x+y;
}
```

В данном примере метод **getSum** возвращает вычисленное значение в метод **main**, непосредственно в переменную **a**.

Методы могут принимать произвольное число параметров.

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

В таком случае возвращаемое значение будет меняться при вызове метода с различными параметрами. При вызове этого метода в программе необходимо передать на место параметров значения или переменные, которые соответствуют типу параметра:

```
public static void main(String[] args) {
    int result1 = getSum(4,6);
    int result2 = getSum(5,7);
    System.out.println(result1);
    System.out.println(result2);
}
```

Перегрузка методов

Java позволяет определять внутри класса два и более методов с одним именем. Но это допустимо, только когда на вход они принимают различные по типу и/или количеству параметры. Такие методы называются перегруженными, а сам процесс — перегрузкой методов.

Возвращаемые типы перегруженных методов могут быть разными, но отличий в возвращаемом типе недостаточно, чтобы Java различал версии метода. Когда Java встречает вызов перегруженного метода, просто выполняет ту его версию, параметры которой соответствуют аргументам, использованным в вызове.

```
static int multiply(int x, int y){
    return x * y;
}

static double multiply(double x){
    return x * x;
}

int a = multiply(5, 3);
int b = multiply(5.5);
```

Практическое применение

Отойдем от абстрактных примеров и напишем простое консольное приложение-калькулятор. Оно будет принимать от пользователя два аргумента и математическую операцию.

Такую программу вполне можно реализовать при помощи **main**:

```
import java.util.Scanner;


public class Calculator {

    public static void main(String[] args){

        // переменная, которая поможет нам читать ввод

        Scanner userInput = new Scanner(System.in);


        System.out.print("Введите операцию [+, -, /, *]: ");

        String operation = userInput.nextLine();


        System.out.print("Введите первый аргумент: ");

        double argument1 = userInput.nextDouble();


        System.out.print("Введите второй аргумент: ");

        double argument2 = userInput.nextDouble();

        userInput.close();


        switch(operation){

            case "+":

                System.out.println(argument1 + argument2);

                break;

            case "-":

                System.out.println(argument1 - argument2);

                break;
```

```
            case "/":
                System.out.println(argument1 / argument2);
                break;
            case "*":
                System.out.println(argument1 * argument2);
                break;
            default:
                System.err.println("Нет такой операции!");
                break;
        }
    }
}
```

- Переменная **userInput** имеет тип **Scanner**. Она поможет читать данные, вводимые пользователем. Позже мы подробно разберем этот вопрос, а пока конструкцию можно использовать *as is*. Считывание данных будет производиться при помощи методов **nextDouble** (в число) и **nextLine** (в строку);
- При использовании типа **Scanner** обязательно указываем в начале программы строку **import java.util.Scanner**. Она подключает функциональность сканера ввода к программе;
- Вывод через **System.err** (в отличие от **System.out**) будет говорить о том, что при работе программы произошла ошибка.

Вы наверняка уже видите минусы данного кода. В нем много повторяющихся операций, смешанная функциональность (и читаем данные, и обрабатываем их), валидация (проверка данных на корректность). Разделим код на методы, которые сделают его лаконичнее:

- Выделим метод подсчета результата в отдельный;
- Выделим метод чтения данных в отдельный;
- Учтем валидацию данных.

Получим:

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args){
        String operation = readStringArgument("Введите операцию [+, -, /, *]: ");
        double argument1 = readDoubleArgument("Введите первый аргумент: ");
        double argument2 = readDoubleArgument("Введите второй аргумент: ");

        if(validateCalculationData(operation)){
            calculate(operation, argument1, argument2);
        }
    }

    static double readDoubleArgument(String userText){
        Scanner userInput = new Scanner(System.in);
        System.out.print(userText);
        double argument = userInput.nextDouble();
        userInput.close();
        return argument;
    }

    static String readStringArgument(String userText){
        Scanner userInput = new Scanner(System.in);
        System.out.print(userText);
        String argument = userInput.nextLine();
        userInput.close();
        return argument;
    }

    /**
 * Created by Alex on 31.01.2018.
 */
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args){
        double argument1 = readDoubleArgument("Введите первый аргумент: ");
        double argument2 = readDoubleArgument("Введите второй аргумент: ");
        String operation = readStringArgument("Введите операцию [+, -, /, *]: ");

        if(validateCalculationData(operation)){
            calculate(operation, argument1, argument2);
        }
    }

    static double readDoubleArgument(String userText){
        Scanner userInput = new Scanner(System.in);
        System.out.print(userText);
        double argument = userInput.nextDouble();
        return argument;
    }
```

```
    static String readStringArgument(String userText){
        Scanner userInput = new Scanner(System.in);
        System.out.print(userText);
        String argument = userInput.nextLine();
        return argument;
    }

    static boolean validateCalculationData(String operation){
        boolean result = true;

                        if(!operation.equals("+")   &&   !operation.equals("-")   &&
!operation.equals("/") && !operation.equals("*")) {
            System.err.println("Введена недопустимая операция");
            result = false;
        }

        return result;
    }

    static void calculate(String operation, double arg1, double arg2){
        switch(operation){
            case "+":
                System.out.println(arg1 + arg2);
                break;
            case "-":
                System.out.println(arg1 - arg2);
                break;
            case "/":
                System.out.println(arg1 / arg2);
                break;
            case "*":
                System.out.println(arg1 * arg2);
                break;
        }
    }
}
```

Код стал объемнее, но мы добились результатов:

- Главный метод стал лаконичным и читаемым. При необходимости доработки или поиска ошибок сразу понятно, куда смотреть;
- Функционал строго разделен по слоям. Стремимся, чтобы каждый метод отвечал за свой маленький фрагмент функционала и не более;
- Методы готовы к повторному использованию.

Код, разложенный по «полочкам»-методам, становится понятнее, удобнее и легче поддается доработкам и контролю.