Урок 2

Основные конструкции

Методы, оператор switch, циклы, кодовые блоки

Методы

Общая форма объявления метода выглядит следующим образом.

```
тип_возвращаемого_методом_значения имя_метода (список_аргументов) {
   тело_метода;
}
```

Тип_возвращаемого_методом_значения обозначает конкретный тип данных (int, float, char, boolean, String и т.д.), возвращаемых методом. Если метод ничего не должен возвращать, указывается ключевое слово void. Для возврата значения из метода используется оператор return.

```
return значение;
```

Для указания имени метода служит идентификатор «имя». Список аргументов обозначает последовательность пар «тип_данных + идентификатор», разделенных запятыми. По существу, аргументы это набор данных, необходимых для работы метода. Если для работы метода не требуются аргументы, то оставляются пустые скобки - ().

Несколько примеров работы с методами.

```
public class FirstApp {
    public static void main(String[] args) {
        // для вызова метода необходимо передать ему 2 аргумента типа int,
        // результатом работы будет целое число, которое напечатается в консоль
        System.out.println(summ(5, 5));

        // для вызова метода ему не нужно передавать аргументы,
        // и он не возвращает данные (метод объявлен как void)
        printSomeText();

        // для вызова метода передаем ему в качестве аргумента строку "Java",
        // которую он выведет в консоль
        printMyText("Java");
    }

    // метод возвращает целое число, принимает на вход два целых числа
    public static int summ(int a, int b) {
        // возвращаем сумму чисел
        return a + b;
    }

    // метод ничего не возвращает, не требует входных данных
    public static void printSomeText() {
        // печатаем Hello в консоль
        System.out.println("Hello");
    }

    // метод ничего не возвращает, принимает на вход строку
    public static void printMyText(String txtToPrint) {
```

```
        // выводим строку txtToPrint в консоль
        System.out.println(txtToPrint);
    }
}
```

При объявлении всех методов внутри основного класса программы после public должно идти слово static. Если ключевое слово static будет отсутствовать, этот метод не получится вызвать из метода main(). Смысл этого ключевого слова будет пояснен на следующих занятиях в теме «Объектно-ориентированное программирование».

Оператор switch

Оператор switch позволяет делать выбор между несколькими вариантами дальнейшего выполнения программы. Выражение последовательно сравнивается со списком значений оператора switch. При совпадении выполняется набор операторов, связанных с этим условием. Если совпадений не было, выполняется блок default (блок default является необязательной частью оператора switch). Оператор break останавливает выполнение блока case, если break убрать – выполнение кода продолжится дальше.

```
switch (выражение) {
   case значение1:
      набор_операторов1;
      break;
   case значение 2:
      набор_операторов2;
      break;
   ...
   default:
      набор_операторов;
}
```

Например, последовательность if-else-if-...

```
public static void main(String[] args) {
    int a = 3;
    if (a == 1) {
        System.out.println("a = 1");
    } else if (a == 3) {
        System.out.println("a = 3");
    } else {
        System.out.println("Ни одно из условий не сработало");
    }
}
```

может быть заменена на следующее.

```
public static void main(String[] args) {
    int a = 3;
    switch (a) {
        case 1:
            System.out.println("a = 1");
            break;
```

```
        case 3:
            System.out.println("a = 3");
            break;
        default:
            System.out.println("Ни один из case не сработал");
    }
}
```

Циклы for

Циклы позволяют многократно выполнять последовательность кода.

```
for (инициализация; условие; итерация) {
   набор_операторов;
}
```

Инициализация представлена переменной, выполняющей роль счётчика и управляющей циклом (например, int i = 0;). Условие определяет необходимость повторения цикла. Итерация задаёт шаг изменения переменной, управляющей циклом.

Важно! Инициализация, условие и итерация в круглых скобках должны быть разделены символом точка с запятой (;). Использование запятых вместо точек с запятыми является ошибкой.
Правильно:

```
for (int i = 0; i < 5; i++) {
    System.out.println("i = " + i);
}
```

Неправильно:

```
for (int i = 0, i < 5, i++) {
    System.out.println("i = " + i);
}
```

Важно! После закрытой круглой скобки точки с запятой **нет**. Если вы там ее поставите, цикл будет работать некорректно.
Правильно:

```
for (int i = 0; i < 5; i++) {
    System.out.println("i = " + i);
}
```

Неправильно:

```
for (int i = 0; i < 5; i++); {
    System.out.println("i = " + i);
}
```

Выполнение цикла for продолжается до тех пор, пока проверка условия даёт истинный результат (true). Пример.

```
public static void main(String args[]) {
    for (int i = 0; i < 5; i++) {
        System.out.println("i = " + i);
    }
    System.out.println("end");
}
```

```
Результат:
i = 0
i = 1
i = 2
i = 3
i = 4
end
```

В начале каждого шага цикла проверяется условие i < 5. Если это условное выражение верно, вызывается тело цикла (в котором прописан метод System.out.println(...)), затем выполняется итерационная часть цикла. Как только условное выражение примет значение false, цикл закончит свою работу.

Пример цикла с отрицательным приращением счётчика

Ниже приведён пример цикла с отрицательным приращением цикла. Еще одной особенностью цикла является «вынос» объявления управляющей переменной до начала цикла, хотя обычно она объявляется внутри for.

```
public static void main(String args[]) {
    int x;  // объявление управляющей переменной вынесено до начала цикла
    for (x = 10; x >= 0; x -= 5) { // Шаг -5
        System.out.print(x + " ");
    }
}
Результат:
10 5 0
```

Условное выражение цикла for всегда проверяется в начале цикла. Это означает, что код в цикле может вообще не выполняться, если проверяемое условие с начала оказывается ложным. Пример.

```
public static void main(String args[]) {
    int x = 0;
    for (int count = 10; count < 5; count++) {
        x += count; // этот оператор не будет выполнен, так как 10 > 5
    }
}
```

Этот цикл вообще не будет выполняться, поскольку начальное значение переменной count больше 5. Это значит, что условное выражение count < 5 оказывается ложным с самого начала.

Цикл for с несколькими управляющими переменными

Для управления циклом можно использовать одновременно несколько переменных. В примере ниже за одну итерацию переменная i увеличивается на 1, а j уменьшается на 1.

```
public static void main(String args[]) {
    for (int i = 0, j = 10; i < j; i++, j--) {
        System.out.println("i-j: " + i + "-" + j);
    }
}
```

```
Результат:
i-j: 0-10
i-j: 1-9
i-j: 2-8
i-j: 3-7
i-j: 4-6
```

Бесконечный цикл

При использовании следующей записи цикла for можно получить бесконечный цикл. Большинство таких циклов требуют специальное условие для своего завершения.

```
for (;;) {
   // ...
}
```

Выход из работающего цикла осуществляется оператором **break** без выполнения всего кода из тела цикла, поэтому в результате нет числа 4.

```
public static void main(String[] args) {
    for(int i = 0; i < 10; i++) {
        if (i > 3) {
            break;
        }
        System.out.println("i = " + i);
    }
}
Результат:
i = 0
i = 1
i = 2
i = 3
```

Цикл foreach

Еще одной разновидностью цикла for является цикл foreach. Он используется для прохождения по всем элементам массива или коллекции, знать индекс проверяемого элемента не нужно. В приведённом ниже примере мы проходим по элементам массива sm типа String и каждому присваиваем временное имя o, то есть «в единицу времени» o указывает на один элемент массива.

```
public static void main(String[] args) {
    String[] sm = {"A", "B", "C", "D"};
    for (String o : sm) {
        System.out.print(o + " ");
    }
}
Результат:
A B C D
```

Вложенные циклы

Циклы, работающие внутри других циклов, называют вложенными. Внимательно разберите последовательность исполнения таких циклов. На всё выполнение внутреннего цикла приходится одна итерация внешнего. Пример.

```
public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            System.out.print(" " + i + j);
        }
    }
}
Результат:
 00 01 02 10 11 12 20 21 22
```

Циклы while

Цикл while работает до тех пор, пока указанное условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла. Если заранее указано условие, которое не выполняется, программа в тело цикла даже не попадет.

```
while (условие) {
   набор_операторов;
}
```

Цикл do-while очень похож на ранее рассмотренные циклы. В отличие от for и while, где условие проверялось в самом начале (предусловие), в цикле do-while оно проверяется в самом конце (постусловие). Это означает, что цикл do-while всегда выполняется хотя бы один раз.

```
do {
   набор_операторов;
while (условие);
```

Кодовые блоки

Кодовый блок представляет собой группу операторов. Для оформления в виде блока они помещаются между открывающей и закрывающей фигурными скобками. Созданный кодовый блок становится единым логическим блоком. Такой блок можно использовать при работе с if и for. Пример.

```
public static void main(String args[]) { // <- начало кодового блока main
    int w = 1, h = 2, v = 0;
    if (w < h) {                          // <- Начало кодового блока if
        v = w * h;
        w = 0;
    }                                     // <- Конец кодового блока if
}                                         // <- Конец кодового блока main
```

В данном примере оба оператора в блоке выполняются в том случае, если значение переменной w меньше значения переменной h. Эти операторы составляют единый логический блок, и ни один из них

не может быть выполнен без другого. Кодовые блоки позволяют оформлять многие алгоритмы в удобном для восприятия виде. Ниже приведён пример программы, где кодовый блок предотвращает деление на ноль.

```
public class MainClass {
   public static void main(String args[]) {
       int a = 0, b = 10, c = 0;
       if (a != 0) {
           System.out.println("a не равно нулю");
           c = b / a;
           System.out.print("b / a равно " + c) ;
       } else {
           System.out.println("a = 0. Делить на 0 нельзя");
       }
   }
}
Результат:
a = 0. Делить на 0 нельзя
```

Области видимости переменных в кодовых блоках.

```
public static void main(String args[]) { // Кодовый блок метода main()
    int x = 10;         // эта переменная доступна для всего кода в методе main
    if (x == 10) {      // Кодовый блок тела if
        int y = 20;     // Эта переменная доступна только в данном кодовом блоке
        // Обе переменные x и y доступны в данном кодовом блоке
        System.out.println("x & y: " + x + " " + y);
        x = y * 2;
    }
    // y = 100; // Ошибка! Переменная y недоступна за пределами тела if
    System.out.println("x = " + x);          // Переменная x по-прежнему доступна
}
```

Ещё один пример объявления переменных в цикле.

```
public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
        int y = -1; // переменная y пересоздается на каждом такте цикла
        System.out.println("y = " + y); // поэтому всегда выводится значение -1
        y++;
        System.out.println("y = " + y); // а тут y = 0
    }
}
```

Как чуть лучше понять циклы for

В этом разделе мы потренируемся в работе с двойными вложенными циклами. Крайне желательно не только читать что здесь написано, но и параллельно повторять код в студии. При повторении пробуйте его немного корректировать и смотреть как изменение кода влияет на получаемую картинку.

Допустим у нас есть вот такой код.

```
public class MainApp {
    public static void main(String[] args) {
        System.out.println('*');
    }
}
```

Не должно возникать сомнений, что при запуске этого кода в консоль будет выведен символ *. А что если мы хотим вывести не один символ, а например 10, и не в столбец, а в строку. Для этого слегка модифицируем код.

```
public class MainApp {
    public static void main(String[] args) {
        for (int j = 0; j < 10; j++) {
            System.out.print('*');
        }
    }
}
```

Если мы запускаем цикл for “начиная с 0” и указываем какую-либо правую границу (в данном случае 10), то это означает, что тело цикла будет выполнено указанное количество раз. Так что задачу с распечаткой строки из 10 символов * мы выполнили. Для завершения строки добавим после цикла пустой System.out.println(). Смотрим на итоговый код и результат вывода в консоль.

```
public class MainApp {
    public static void main(String[] args) {
        for (int j = 0; j < 10; j++) {
            System.out.print('*');
        }
        System.out.println();
    }
}
```

Результат:
****

Можем ли мы обернуть существующий код в еще один цикл, чтобы заставить программу печатать не одну строку, а например 5? Да, можем.

```
public class MainApp {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 10; j++) {
                System.out.print('*');
            }
            System.out.println();
```

```
        }
    }
}
```

В результате получим вот такой вывод:

```
**********
**********
**********
**********
**********
```

Смотрим еще раз на код, и думаем как его логически для себя представлять.

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 10; j++) {
                System.out.print('*');
            }
            System.out.println();
        }
    }
}
```

Фиолетовый блок кода отвечает на распечатку всего лишь одного символа без переноса строки. Желтый цикл заставляет повториться эту операцию 10 раз, получая тем самым строку длиной 10, а желтый System.out.println() заставляет выполнить перенос и возврат каретки на новую строку. И наконец зеленый цикл запускает процесс распечатки строки 5 раз.

Если внимательно посмотреть на код, то должно быть понятно что **i** представляет собой номер строки, а **j** номер столбца. Давайте рассмотрим несколько задач на двойной цикл и распечатку различных фигур.

Задача 1. Как сделать, чтобы такой двойной цикл печатал таблицу из * размером 6x6, и чтобы между звездочками стояли пробелы? Просто меняем условия работы циклов и корректируем print().

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
Результат:
* * * * * *
* * * * * *
* * * * * *
* * * * * *
* * * * * *
* * * * * *
```

Задача 2. А теперь давайте попробуем сделать так, чтобы правая половина этой таблицы состояла бы не из *, а из 0. Добавим для этого условие.

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                if (j < 3) {
                    System.out.print("* ");
                } else {
                    System.out.print("0 ");
                }
            }
            System.out.println();
        }
   }
}
```

Результат:

```
* * * 0 0 0
* * * 0 0 0
* * * 0 0 0
* * * 0 0 0
* * * 0 0 0
* * * 0 0 0
```

Задача 3. Сделать так, чтобы по краям квадрата стояли 0, а внутри он был бы заполнен *.

Для этого необходимо проверять, если номер строки или столбца равен 0 или (размер таблицы - 1), тогда ставим 0, в противном случае *.

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                if (i == 0 || i == 5 || j == 0 || j == 5) {
                    System.out.print("0 ");
                } else {
                    System.out.print("* ");
                }
            }
            System.out.println();
        }
   }
}
```

Результат:

```
0 0 0 0 0 0
0 * * * * 0
0 * * * * 0
0 * * * * 0
0 * * * * 0
0 0 0 0 0 0
```

Задача 4. Сложный пример. Как заполнить таблицу в шахматном порядке? Попробуйте разобраться как работает следующий код. Для этого вам может пригодиться листик и ручка, чтобы нарисовать

такую таблицу, и пронумеровать строки и столбцы. Напоминаем, что % это вычисление остатка от деления.

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                if ((i + j) % 2 == 0) {
                    System.out.print("* ");
                } else {
                    System.out.print("0 ");
                }
            }
            System.out.println();
        }
   }
}
```

Результат:

```
* 0 * 0 * 0
0 * 0 * 0 *
* 0 * 0 * 0
0 * 0 * 0 *
* 0 * 0 * 0
0 * 0 * 0 *
```

Задача 5. Сложный пример. Заполнить одну половину таблицы 0, а вторую *, при этом половину разделены диагональю.

```
public class MainApp {
   public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 6; j++) {
                if (i > j) {
                    System.out.print("0 ");
                } else if (i < j) {
                    System.out.print("* ");
                } else {
                    System.out.print("  ");
                }
            }
            System.out.println();
        }
   }
}
```

Результат:

```
  * * * * *
0   * * * *
0 0   * * *
0 0 0   * *
0 0 0 0   *
0 0 0 0 0
```

Разобрались? Тогда можете попробовать “зарисовать” вот такие картинки. Это не домашнее задание, а просто дополнительная тренировка.

Рисунок 1	Рисунок 2	Рисунок 3	Рисунок 4	Рисунок 5
* * * * * * * * * * * * * * * * * * * *	* *	* * * * * * * * * * * * * * *	* *	* *

Для построения рисунков вам может пригодиться вот такая схема. Например, по рисунку на схеме хорошо видно особенность координат диагональных элементов. Для каждого рисунка можете составлять такую схему.

	0	1	2	3	4	5
0	*	*	*	*	*	*
1		*				*
2			*			*
3				*		*
4					*	*
5						*

Когда пытаетесь решить любую задачу, пытайтесь найти зависимость между ячейками со звездочками и соответствующими индексами строк и столбцов.

Домашнее задание

1. Написать метод, принимающий на вход два целых числа и проверяющий, что их сумма лежит в пределах от 10 до 20 (включительно), если да – вернуть true, в противном случае – false.
2. Написать метод, которому в качестве параметра передается целое число, метод должен напечатать в консоль, положительное ли число передали или отрицательное. *Замечание: ноль считаем положительным числом.*
3. Написать метод, которому в качестве параметра передается целое число. Метод должен вернуть true, если число отрицательное, и вернуть false если положительное.
4. Написать метод, которому в качестве аргументов передается строка и число, метод должен отпечатать в консоль указанную строку, указанное количество раз;
5. * Написать метод, который определяет, является ли год високосным, и возвращает boolean (високосный - true, не високосный - false). Каждый 4-й год является високосным, кроме каждого 100-го, при этом каждый 400-й – високосный.

*Если выполнение задач вызывает трудности, можете обратиться к последней странице методического пособия. Для задач со * не нужно искать решение в интернете, иначе нет смысла их выполнять..*

Дополнительные материалы

1. К. Сьерра, Б. Бейтс Изучаем Java // Пер. с англ. – М.: Эксмо, 2012. – 720 с.
2. Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. – М.: Вильямс, 2014. – 864 с.

Используемая литература

1. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. – М.: Вильямс, 2015. – 1376 с.

Подсказки по домашнему заданию

1)
```
public static boolean within10and20(int x1, int x2) {
   …
}
```
2)
```
public static void isPositiveOrNegative(int x) {
   if (…) {
      System.out.println(…);
   } else {
      System.out.println(…);
   }
}
```
3)
```
public static boolean isNegative(int x) {
   if (…) {
      return true;
   }
   return false;
}
```
4)
```
public static void printWordNTimes(String word, int times) {
   for (...) {
      System.out.println(…);
   }
}
```
5) Если делаете задание под * вряд ли вам нужны подсказки

Вместо … подставляете ваш код. Представлены не все существующие решения, возможно вы найдете свое.

Если до этого момента все равно непонятно как подступиться к задачам (такое вполне может быть и это не страшно), давайте разберем какой-нибудь пример.

Задание: Написать метод, принимающий на вход три целых числа и проверяющий, что их сумма больше 50 (включительно), если да – вернуть true, в противном случае – false.

Разбираем задание по кускам и пишем код: ***Написать метод***

```
public static void methodName() {
}
```

Как нас и просили в задаче метод мы написали. Теперь дальше: ***принимающий на вход три целых числа***. Это значит у метода три аргумента, а раз целых числа то это вероятнее всего int. Корректируем код.

```
public static void methodName(int a, int b, int c) {
}
```

Последняя часть: ***и проверяющий, что их сумма больше 50 (включительно), если да – вернуть true, в противном случае – false.*** Если речь идет о возврате то тип метода меняется с void на какой-нибудь другой. Так как нас просят вернуть true или false, то это тип boolean. И в будущем в методе добавится оператор return.

```
public static boolean methodName(int a, int b, int c) {
}
```

Теперь решаем центральный блок задачи: ***проверяющий, что их сумма больше 50 (включительно).***

```
public static boolean methodName(int a, int b, int c) {
   int sum = a + b + c;
   if (sum >= 50) {
      return true;
   } else {
      return false;
   }
}
```

Блок if может быть написан гораздо компактнее и проще. Но если вы разбираетесь с программированием с нуля, не стоит жертвовать понятностью кода. Поэтому пока берем такой вариант.

Наш метод готов, задача почти решена, осталось дать методу нормальное имя.

```
public static boolean isSumGreaterThan50(int a, int b, int c) {
   int sum = a + b + c;
   if (sum >= 50) {
      return true;
   } else {
      return false;
   }
}
```

Задача решена! Вот мы пошагово прошлись по одной задаче, попробуйте использовать подобный подход для решения ваших задач.

Еще один штрих. Если вы только начинаете свой путь в программировании, то это может быть чуть сложным.

Приведенное выше решение можно очень сильно сократить.

Было вот так:

```
public static boolean isSumGreaterThan50(int a, int b, int c) {
   int sum = a + b + c;
   if (sum >= 50) {
      return true;
   } else {
      return false;
   }
}
```

При небольшой модификации получим такой код:

```
public static boolean isSumGreaterThan50(int a, int b, int c) {
   return a + b + c >= 50;
}
```

1. Поскольку мы сумму сразу же используем в сравнении, складывать ее в локальную переменную смысла нет.
2. Любое сравнение это логическое выражение, результатом которого будет либо true, если оно истинно, либо false, в противном случае. В данном случае, если сумма аргументов будет больше либо равна 50, то результатом логического выражения будет true, и мы true и вернем, если нет, результатом будет false, который и будет возвращен. То есть логика работы метода не будет отличаться, от той что мы получили при использовании if'ов

Если вам подобный подход кажется неочевидным, можете пока использовать if'ы. Как только тема уложится в голове, будете легко использовать более корректный подход.