Введение в программирование на Java

Лекция 3. Условные операции. Циклы.

Виталий Олегович Афанасьев

27 января 2025

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2  boolean neqTrue = (2 + 2) != 4; // false
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2  boolean neqTrue = (2 + 2) != 4; // false
3
4  boolean less      = 2 < 3; // true
5
6
7
8
9
10
11
12
13
14
15
```

Операции сравнения и логические операции (повторение)

```java
boolean eqTrue  = (2 + 2) == 4; // true
boolean neqTrue = (2 + 2) != 4; // false

boolean less      = 2 < 3; // true
boolean greaterEq = 3 >= 3; // true
```

Операции сравнения и логические операции (повторение)

```java
boolean eqTrue  = (2 + 2) == 4; // true
boolean neqTrue = (2 + 2) != 4; // false

boolean less      = 2 < 3; // true
boolean greaterEq = 3 >= 3; // true

boolean tandt = true && true;  // true
```

Операции сравнения и логические операции (повторение)

```java
boolean eqTrue  = (2 + 2) == 4; // true
boolean neqTrue = (2 + 2) != 4; // false

boolean less      = 2 < 3; // true
boolean greaterEq = 3 >= 3; // true

boolean tandt = true && true;  // true
boolean tandf = true && false; // false
```

Операции сравнения и логические операции (повторение)

```java
boolean eqTrue  = (2 + 2) == 4; // true
boolean neqTrue = (2 + 2) != 4; // false

boolean less      = 2 < 3; // true
boolean greaterEq = 3 >= 3; // true

boolean tandt = true && true;  // true
boolean tandf = true && false; // false

boolean tort = true || true;  // true
```

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2  boolean neqTrue = (2 + 2) != 4; // false
3
4  boolean less      = 2 < 3; // true
5  boolean greaterEq = 3 >= 3; // true
6
7  boolean tandt = true && true;  // true
8  boolean tandf = true && false; // false
9
10 boolean tort = true || true;  // true
11 boolean torf = true || false; // true
12
13
14
15
```

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2  boolean neqTrue = (2 + 2) != 4; // false
3
4  boolean less      = 2 < 3; // true
5  boolean greaterEq = 3 >= 3; // true
6
7  boolean tandt = true && true;  // true
8  boolean tandf = true && false; // false
9
10 boolean tort = true || true;  // true
11 boolean torf = true || false; // true
12
13 boolean nott = !true;  // false
14
15
```

Операции сравнения и логические операции (повторение)

```
1  boolean eqTrue  = (2 + 2) == 4; // true
2  boolean neqTrue = (2 + 2) != 4; // false
3
4  boolean less      = 2 < 3; // true
5  boolean greaterEq = 3 >= 3; // true
6
7  boolean tandt = true && true;  // true
8  boolean tandf = true && false; // false
9
10 boolean tort = true || true;  // true
11 boolean torf = true || false; // true
12
13 boolean nott = !true;  // false
14
15 boolean example = !(2 > 3) && (1 + 2 == 1 + 1 + 1);
```

Об отличиях логических и битовых операций (1)

Операции "и" и "или" для типа boolean существуют в двух вариантах: логическом и битовом:

```
1 boolean tandf = true && false; // false
2 boolean tandf = true & false; // false
```

Об отличиях логических и битовых операций (1)

Операции "и" и "или" для типа `boolean` существуют в двух вариантах: логическом и битовом:

```
1 boolean tandf = true && false; // false
2 boolean tandf = true & false; // false
```

Но зачем?

Об отличиях логических и битовых операций (2)

Логические операции обладают т.н. **short-circuit** логикой (короткой схемой вычисления).

Если правая часть не влияет на результат, то она не будет вычислена:

```
1  int x = 10;
2  // Левая часть false, поэтому правая не вычисляется
3  boolean fandt = false && (++x > 0);
4  System.out.println(fandt); // false
5  System.out.println(x); // 10
6
7  int x = 10;
8  // Правая часть вычисляется всегда
9  boolean fandt = false & (++x > 0);
10 System.out.println(fandt); // false
11 System.out.println(x); // 11
```

Об отличиях логических и битовых операций (3)

Что выведется в следующем коде?

```
1  int x = 10;
2  boolean first  = (x > 0) || (++x > 5);
3  boolean second = (x > 0) |  (++x > 5);
4  boolean third  = (x < 0) || (++x > 5);
5  boolean fourth = (x < 0) |  (++x > 5);
6  System.out.println(x);
7  System.out.println(first);
8  System.out.println(second);
9  System.out.println(third);
10 System.out.println(fourth);
```

Об отличиях логических и битовых операций (3)

Что выведется в следующем коде?

```
1  int x = 10;
2  boolean first  = (x > 0) || (++x > 5);
3  boolean second = (x > 0) |  (++x > 5);
4  boolean third  = (x < 0) || (++x > 5);
5  boolean fourth = (x < 0) |  (++x > 5);
6  System.out.println(x);
7  System.out.println(first);
8  System.out.println(second);
9  System.out.println(third);
10 System.out.println(fourth);
```

```
13
true
true
true
true
```

Блоки и области видимости

Блоки (1)

Блок — составной оператор, содержащий любое количество операторов, заключённых в фигурные скобки.

Каждый блок создаёт свою **область видимости**: переменные, объявленные внутри блока, доступны только в его рамках.

```
1  public static void main(String[] args) {
2      int first = 1;
3      {
4          int second = 2;
5          System.out.println(first);
6          System.out.println(second);
7      } // second 'выходит' из области видимости
8      System.out.println(first);
9      System.out.println(second); // ERROR
10 }
```

Блоки (2)

Допускается любая вложенность блоков:

```
1  public static void main(String[] args) {
2                                      // Доступные переменные:
3      int first = 1;                  // first
4      {                               // first
5          int second = 2;             // first, second
6          {                           // first, second
7              int third = 3;          // first, second, third
8              {                       // first, second, third
9                  int fourth = 4;     // first, second, third, fourth
10                 ...                 // first, second, third, fourth
11             }                       // first, second, third
12         }                           // first, second
13     }                               // first
14 }
```

Блоки (3)

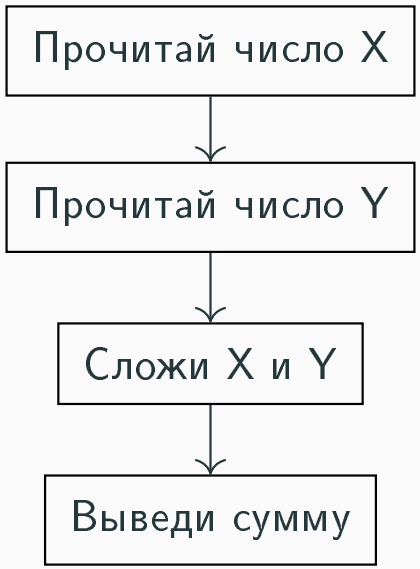
Во вложенных блоках нельзя определить две переменные с одинаковыми именами:

```
1 public static void main(String[] args) {
2     int first = 1;
3     {
4         int second = 2;
5         int first = 3; // ERROR
6     }
7 }
```

Управляющие конструкции

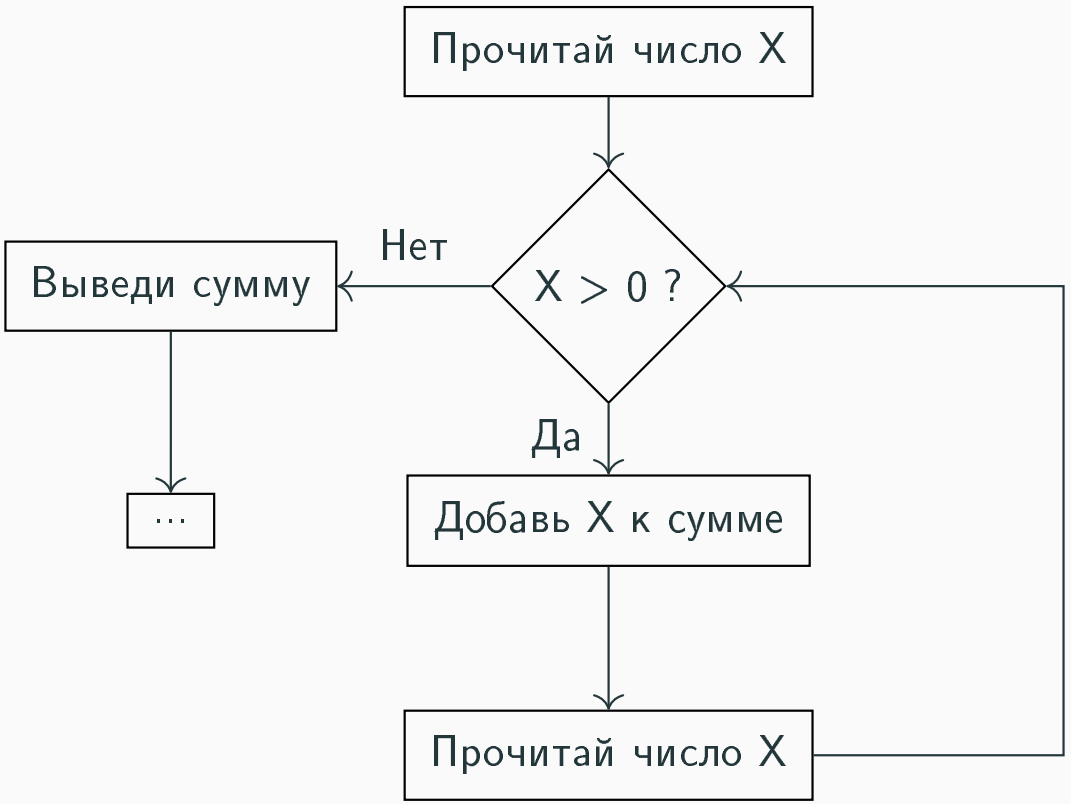
Управляющие конструкции (1)

Только самые простые алгоритмы являются линейной последовательностью шагов:



Управляющие конструкции (2)

Большинство же программ — это комбинация линейных шагов, условных и безусловных переходов:



Условный оператор

Условный оператор (1)

Один из способов выполнения операций только при определённом условии — оператор `if`.

Его базовая форма выглядит следующим образом:

```
if (условие)
    оператор
```

Условный оператор (2)

```
1 if (x % 2 == 0)
2     System.out.println("X - чётное");
```

Условный оператор (3)

Блок — это тоже оператор:

```
1 if (x % 2 == 0 && x % 3 == 0) {
2     System.out.println("X - чётное");
3     System.out.println("X - делится на 3");
4 }
```

Условный оператор (4)

Более общая форма выглядит так:

```
if (условие)
    оператор
else
    оператор
```

Оператор внутри `else` выполняется только в том случае, если условие ложно.

Условный оператор (5)

```
1 if (x < 0)
2     System.out.println("X - отрицательное");
3 else
4     System.out.println("X - неотрицательное");
```

Условный оператор (6)

Несколько конструкций `if-else` можно ставить друг за другом:

```
1 if (x < 0)
2     System.out.println("X - отрицательное");
3 else if (x % 2 == 0)
4     System.out.println("X - неотрицательное и чётное");
5 else
6     System.out.println("X - неотрицательное и нечётное");
```

Оператор switch (1)

Иногда приходится иметь цепочку условий, проверяющих выражение на равенство одной из констант:

```
1  if (x == 0)
2      System.out.println("Ноль");
3  else if (x == 1)
4      System.out.println("Один");
5  else if (x == 2)
6      System.out.println("Два");
7  else if (x == 3)
8      System.out.println("Три");
9  else if (x == 4)
10     System.out.println("Четыре");
11 else if (x == 5)
12     System.out.println("Пять");
13 ...
```

Оператор switch (2)

Для таких случаев в Java есть оператор switch:

```
switch (x) {
    case 0 -> System.out.println("Ноль");
    case 1 -> System.out.println("Один");
    case 2 -> System.out.println("Два");
    case 3 -> System.out.println("Три");
    case 4 -> System.out.println("Четыре");
    case 5 -> System.out.println("Пять");
    ...
}
```

Оператор switch (3)

Несколько ветвей с разными значениями можно объединить в одну:

```
switch (x) {
    case 0, 2, 4, 6, 8 ->
        System.out.println("Чётное");
    case 1, 3, 5, 7, 9 ->
        System.out.println("Нечётное");
}
```

Оператор switch (4)

Также есть воможность обработать все прочие значения в одной ветви при помощи `defaut`:

```
switch (x) {
    case 0, 2, 4, 6, 8 ->
        System.out.println("Чётное");
    case 1, 3, 5, 7, 9 ->
        System.out.println("Нечётное");
    default ->
        System.out.println("Какое-то ещё");
}
```

Оператор switch (5)

В данной форме оператор `switch` является **выражением** (т.е. возвращает значение).

Таким образом, это значение можно сохранить в переменную:

```
1 int value = switch (x) {
2     case "Один" -> 1;
3     case "Два" -> 2;
4     case "Три" -> 3;
5     case "Четыре" -> 4;
6     case "Пять" -> 5;
7     default -> -1;
8 };
```

Важно, чтобы **все** возможные значения были покрыты (иначе непонятно, какое значение должна иметь переменная).

Оператор switch (6)

У оператора `switch` есть и другая форма (привычная для языков C/C++). О ней рекомендуется прочитать в разделе **3.8.5 Core Java**.

Тернарный оператор (1)

Также есть тернарный оператор, который является краткой формой `if-else`, возвращающей значение.

В общей форме тернарный оператор выглядит так:

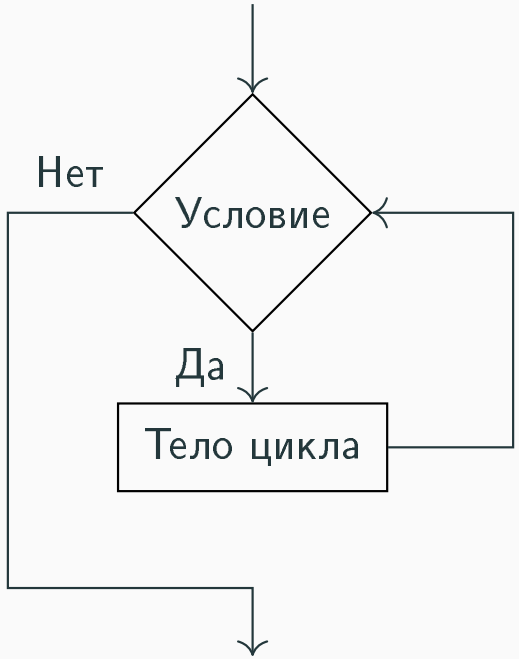
```
условие ? выражение1 : выражение2
```

Тернарный оператор (2)

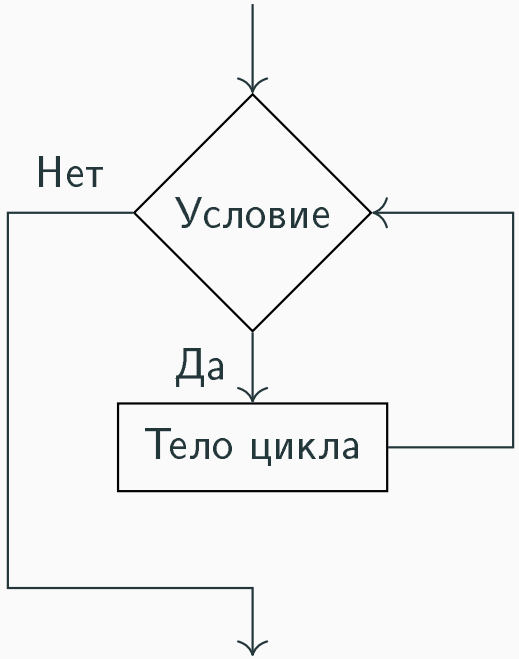
```
int min = x < y ? x : y;
```

Циклы

Цикл с предусловием (1)



Цикл с предусловием (1)



В Java данный цикл имеет следующий вид:

```
while (условие)
    оператор
```

Цикл с предусловием (2)

Решим следующую задачу: найти такую сумму $2 + 4 + 6 + \ldots > N$, что она минимальна.

Другими словами, переберём следующие суммы:

- 2
- $2 + 4$
- $2 + 4 + 6 + \ldots$

Как только сумма выйдет за пределы N — остановимся.

Цикл с предусловием (2)

Решим следующую задачу: найти такую сумму $2 + 4 + 6 + \ldots > N$, что она минимальна.

Другими словами, переберём следующие суммы:

- 2
- $2 + 4$
- $2 + 4 + 6 + \ldots$

Как только сумма выйдет за пределы N — остановимся.

```
1 int n = ...;
2 int sum = 0;
3 int current = 2;
4 while (sum <= n) {
5     sum += current;
6     current += 2;
7 }
8 System.out.println(sum);
```

Цикл с предусловием (3)

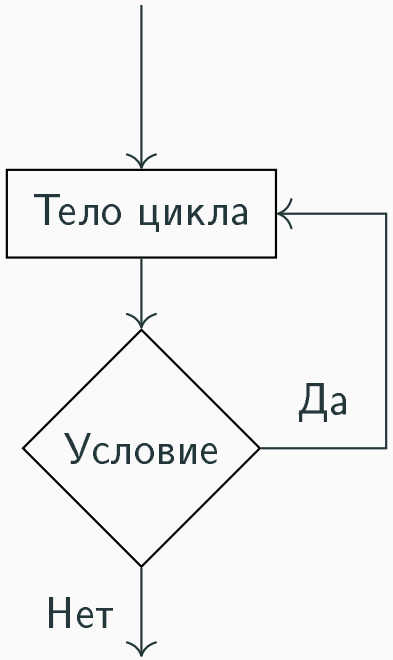
Разберём код по шагам при $N = 35$.

```
1 int sum = 0;
2 int current = 2;
3 while (sum <= n) {
4     sum += current;
5     current += 2;
6 }
```

Итерация		1	2	3	4	5	6	
sum	0	2	6	12	20	30	42	42
current	2	4	6	8	10	12	14	14

Цикл с постусловием (1)

Иногда полезно иметь такой цикл, который бы выполнял первую итерацию всегда, а условие бы проверялось в конце итерации, а не в начале.



Цикл с постусловием (2)

Для таких случаев существует цикл с постусловием:

```
do
    оператор
while (условие);
```

Цикл с постусловием (3)

Решим задачу о нахождении количества цифр в числе N.

```
1 int n = ...;
2 int numDigits = 0;
3 do {
4     ++numDigits;
5     n /= 10;
6 } while (n > 0);
```

Цикл с постусловием (4)

Разберём код по шагам при $N = 123$.

```
1 int n = ...;
2 int numDigits = 0;
3 do {
4     ++numDigits;
5     n /= 10;
6 } while (n > 0);
```

Итерация		1	2	3	
n	123	12	1	0	0
numDigits	0	1	2	3	3

Цикл с постусловием (5)

```
1 int n = ...;
2 int numDigits = 0;
3 do {
4     ++numDigits;
5     n /= 10;
6 } while (n > 0);
```

Сравните с решением, использующим цикл с предусловием:

```
1 int n = ...;
2 int numDigits = 0;
3 if (n == 0)
4     numDigits = 1;
5 while (n > 0) {
6     ++numDigits;
7     n /= 10;
8 }
```

Цикл со счётчиком (1)

Наиболее часто встречающийся вариант цикла — цикл со счётчиком.

В общей форме выглядит следующим образом:

```
for (оператор_инициализатор; условие; оператор_шаг)
    оператор_тело
```

Этот цикл выполняет операторы в следующей последовательности:

1. оператор_инициализатор
2. условие?
3. оператор_тело
4. оператор_шаг
5. условие?
6. оператор_тело
7. оператор_шаг
8. условие?
9. ...

Цикл со счётчиком (2)

Вывод квадратов целых чисел от 1 до 10:

```
1 for (int i = 1; i <= 10; ++i)
2     System.out.println(i * i);
```

Последовательность шагов:

1. `int i = 1`
2. `i <= 10`?
3. `System.out.println(i * i)`
4. `++i`
5. `i <= 10`?
6. `System.out.println(i * i)`
7. `++i`
8. `i <= 10`?
9. ...

Цикл со счётчиком (3)

Можно заметить, что цикл со счётчиком на самом деле является циклом с предусловием.

Следующий цикл:

```
for (оператор_инициализатор; условие; оператор_шаг)
    оператор_тело
```

Можно переписать следующим образом:

```
оператор_инициализатор
while (условие) {
    оператор_тело
    оператор_шаг
}
```

Тем не менее, если цикл вырождается в последовательный перебор какой-то последовательности, рекомендуется использовать цикл `for`.

Оператор `continue`

Оператор `continue` позволяет немедленно перейти к следующей итерации цикла.

```
1 for (int i = 1; i <= 5; ++i) {
2     System.out.print("Step ");
3     if (i % 2 == 0) continue;
4     System.out.println(i * i);
5 }
```

Вывод:

```
Step 1
Step Step 9
Step Step 25
```

Оператор `break`

Оператор `break` позволяет немедленно прервать цикл.

```
1 for (int i = 1; i <= 5; ++i) {
2     System.out.print("Step ");
3     if (i % 2 == 0) break;
4     System.out.println(i * i);
5 }
```

Вывод:

```
Step 1
Step
```

Операторы continue и break с метками

Операторы continue и break позволяют использовать метки, на которые должен осуществляться переход. Это полезно, если требуется выполнить выход из вложенного цикла.

```
1 outerloop: while (true) {
2     for (int i = 1; i <= 5; ++i) {
3         System.out.print("Step ");
4         if (i >= 4) break outerloop;
5         System.out.println(i);
6     }
7 }
```

Вывод:

```
Step 1
Step 2
Step 3
Step
```