БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

ЯЗЫК JAVA

ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОГРАММОЙ

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

МИНСК
2022

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1
К78

Утверждено на заседании кафедры
веб-технологий и компьютерного моделирования
8 ноября 2022 г., протокол № 3

Рецензент
кандидат физико-математических наук,
доцент *Г. А. Расолько*

Кравчук, А. С.

К78 Язык Java. Операторы управления программой : учеб. материалы / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2022. – 32 с.

Рассматриваются условные конструкции языка Java, различные виды циклов и операторы перехода, а также организация вычислений с заданной точностью. Издание ориентировано как на тех, кто не имеет опыта практического программирования на языке Java, так и на тех, кто хотел бы систематизировать и улучшить свои знания. В каждой теме приводится необходимый теоретический материал и код программ, что существенно ускоряет усваивание материала, а также способствует более квалифицированному подходу к программированию.

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1

Оглавление

Введение

Все операторы управления программой могут быть условно разделены на следующие категории:

- операторы выбора, к которым относятся оператор условия `if` и оператор выбора `switch`;
- операторы цикла (`for, while, do while`);
- операторы перехода (`break, continue, return`).

Условные операторы

Одним из фундаментальных элементов многих языков программирования являются условные конструкции. Данные конструкции позволяют направить работу программы по одному из путей в зависимости от определенных условий.

В языке Java используются следующие условные конструкции: `if..else` и `switch..case`.

Оператор if()

Формат оператора:

- полная форма:

```
if (условие) инструкция1/блок1;
else инструкция2/блок2;
```

- сокращенная форма:

```
if (условие) инструкция1/блок1;
```

Выполнение оператора `if` начинается с вычисления *условия*, результатом которого может являться значение *только* булевского типа. Далее выполнение осуществляется по следующей схеме:

- если выражение *истинно* (имеет значение `true`), то выполняется `инструкция1/блок1`,
- если выражение ложно (имеет значение `false`), то выполняется `инструкция2/блок2`.

После этого управление передается на следующую после `else` инструкцию.

Пример.

```
if (i < j) i++;
else {
    j = i - 3;
    i++;
}
```

В сокращенной форме оператора, в любом случае, в том числе и когда *условие* ложно, то выполняется следующая за `if` инструкция.

Простейший пример с ветвлением программы

Проверим меньше ли заданной константы `LIMIT` введенное с клавиатуры число.

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4     public static void main (String args[]){
5        final int LIMIT = 10;
6        String str = "Введите число: ";
7        String report1 = "Число < LIMIT\n";
8        String report2 = "Число >= LIMIT\n";
9        // ввод числа с консоли
10       System.out.print(str);
11       Scanner objIn = new Scanner(System.in);
12       float number = objIn.nextFloat();
13       objIn.close();
14       //оператор if
15       if (number < LIMIT) {
16           // если введенное число меньше 10
17           System.out.print(report1);
18       }
19       else {
20           // иначе
21           System.out.print(report2);
22       }
23    }
24 }
```

Результат выполнения программы:

```
Введите число: 5
Число < LIMIT
```

Задача о попадании в круг

Написать программу, определяющую попадают ли введенные координаты в область круга с центром в начале координат и радиусом, определенным константой `R` (Рисунок 1), объявленной с помощью служебного слова `final`.

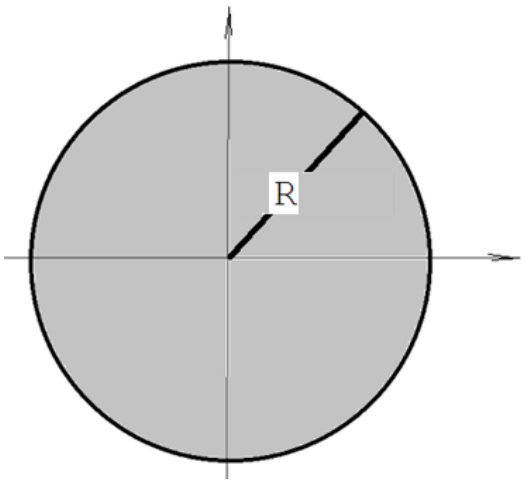

Рисунок 1 – Круг радиуса `R` с центром в начале координат

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4     public static void main (String args[]){
5        final float R = 1.5f;
6        String str = "Введите X и Y координаты:\n";
7        String report1 = "Точка лежит в круге\n";
8        String report2 = "Точка вне круга\n";
9        // ввод числа с консоли
10       System.out.print(str);
11       Scanner objIn = new Scanner(System.in);
12       float x = objIn.nextFloat();
13       float y = objIn.nextFloat();
14       objIn.close();
```

```
15          boolean condition = x * x + y * y <= R * R;
16          //оператор if
17          if (condition) {
18              // если введенное число меньше 10
19              System.out.print(report1);
20          }
21          else {
22              // иначе
23              System.out.print(report2);
24          }
25      }
26  }
```

Результат работы программы:

```
Введите X и Y координаты:
1 2
Точка вне круга
```

Попадание в четверть круга

Обычным для определения условий попадания в геометрическую область любой конфигурации является использование простейших подобластей, пересечение которых дает искомую область. В данном случае четверть круга (Рисунок 2) является пересечением круга (Рисунок 1) и двух полуплоскостей `x < 0` и `y > 0`.

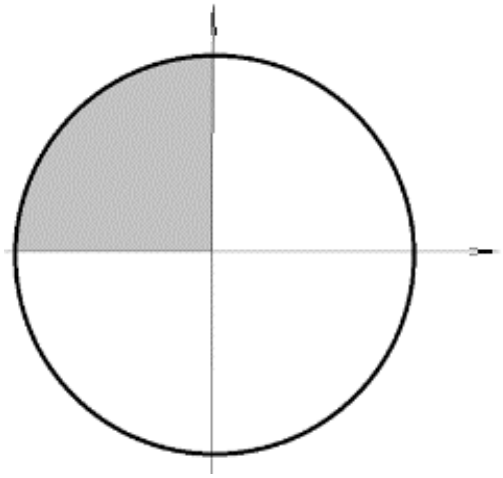

Рисунок 2 – Четверть круга с центром в начале координат

Пример.

```java
import java.util.Scanner;

public class MyClass{
    public static void main (String args[]){
        final float R = 1.5f;
        String str = "Введите X и Y координаты:\n";
        String report1 = "Точка лежит в области\n";
        String report2 = "Точка вне области\n";
        // ввод числа с консоли
        System.out.print(str);
        Scanner objIn = new Scanner(System.in);
        float x = objIn.nextFloat();
        float y = objIn.nextFloat();
        objIn.close();
        // константу можно определять через переменные
        final boolean AREA1 = x * x + y * y <= R * R;
        final boolean AREA2 = x <= 0;
        final boolean AREA3 = y >= 0;

        // оператор if
        if (AREA1 & AREA2 & AREA3) {
            // если введенное число меньше 10
            System.out.print(report1);
        }
        else {
            // иначе
            System.out.print(report2);
        }
    }
}
```

Результат работы программы:

```
Введите X и Y координаты:
-1
1
Точка лежит в области
```

Лестница `if-else-if`

Одной из распространенных конструкций любого языка является лестница `if-else-if`.

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4     public static void main (String args[]){
5        final int LIMIT = 10;
6        String str = "Введите число: ";
7        String report1 = "Число < LIMIT\n";
8        String report2 = "Число = LIMIT\n";
9        String report3 = "Число > LIMIT\n";
10       // ввод числа с консоли
11       System.out.print(str);
12       Scanner objIn = new Scanner(System.in);
13       float number = objIn.nextFloat();
14       objIn.close();
15       //оператор if-else-if
16       if (number < LIMIT) {
17          // если введенное число меньше 10
18          System.out.print(report1);
19       }
20       else if (number == LIMIT){
21          // иначе если
22          System.out.print(report2);
23       }
24       else {
25          // иначе
26          System.out.print(report3);
27       }
28    }
29 }
```

Результат работы программы:

```
Введите число: 5
Число < LIMIT
```

Следующий пример лестницы `if-else-if` имеет более сложный вид. Это элементарный пример организации консольного калькулятора.

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4     public static void main (String args[]){
5        final int LIMIT = 10;
6        // final перед объектами класса String
7        final String DECL1 = "Введите знак бинарной " +
8                             "операции ";
9        final String DECL2 ="Введите последовательно "+
10                          "значения обоих операндов\n";
11       final String REPORT = "Результат вычислений: ";
12       // ввод знака и двух чисел с консоли
13       Scanner objIn = new Scanner(System.in);
14       System.out.print(DECL1);
15       // специфика ввода символа
16       final char operationSing =
17                                  objIn.next().charAt(0);
18       System.out.print(DECL2);
19       float x = objIn.nextFloat();
20       float y = objIn.nextFloat();
21       // компилятор не пропустит если result не будет
22       // проинициализировано, т.к. это значение
23       // выводится на экран в конце программы
24       float result = 0;
25       //оператор if-else-if
26       if (operationSing == '-') {
27          result = x - y;
28       }
29       else if (operationSing == '+') {
30          result = x + y;
31       }
32       else if (operationSing == '*') {
33          result = x * y;
34       }
35       else if (operationSing == '/') {
36          result = x / y;
37       }
38       else {
39           System.out.print(false);
40           return;
41       }
```

```
42          System.out.print(REPORT + result);
43      }
44  }
```

Результат работы программы:

```
Введите знак бинарной операции +
Введите последовательно значения обоих операндов
1 2
Результат вычислений: 3.0
```

Оператор выбора `switch()`

Другим способом организации выбора из множества различных вариантов является использование специального оператора выбора `switch`. Формат оператора следующий:

```
switch (выражение)
{
    case константное-выражение1:
        списокИнструкций1;
        break;
    case константное-выражение2:
        списокИнтерукций2;
        break;
    …
    default:
        списокИнструкций;
        break;
}
```

Выражение, следующее за ключевым словом `switch` в круглых скобках, может быть любым выражением, допустимым в языке Java, *значение которого должно быть* одного из целых типов (`byte, short, char, int`) или их оберток (`Byte, Short, Character, Integer`). Также допустимо использовать тип `String` и перечисление (`enum`). Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора `switch` состоит из нескольких инструкций, начинающихся ключевым словом `case` с последующим константным выражением.

Обычно в качестве константного выражения используются целые или символьные константы (не может содержать переменные или вызовы функций). Все константные выражения в операторе `switch` должны быть уникальны. Кроме инструкций, помеченных ключевым словом `case`. Оператор `switch` *может* содержать, но только один, фрагмент, помеченный ключевым словом `default`.

Список инструкций может быть пустым либо содержать одну инструкцию или более. В операторе `switch` не требуется заключать в фигурные скобки последовательность инструкций. *Для окончания действий по одному из вариантов* `case` *присутствует оператор* `break`*, который не обязателен.*

Схема выполнения оператора `switch` следующая: вычисляется выражение в круглых скобках; вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами `case`; если одно из константных выражений совпадает со значением выражения, то управление передается на инструкцию, соответствующую совпавшему варианту значения `case`, если ни одно из константных выражений не равно выражению, то управление передается на инструкцию, помеченную ключевым словом `default`, а в случае его отсутствия управление передается на следующую после `switch` инструкцию.

Пример.

```
1  import java.util.Scanner;
2
3  public class MyClass{
4      public static void main (String args[]){
5          String declaration = "Введите символ: ";
6          String report = "Результат работы switch: ";
7          // ввод символа с консоли
8          System.out.print(declaration);
9          Scanner objIn = new Scanner(System.in);
10         // специфика ввода символа
11         final char CH = objIn.next().charAt(0);
12         int i = 0;
13          switch (CH)
14         {
15             case 'a': i += 2; break;
16             case 'b': i *= 3; break;
17             case 'c': i /= 2; break;
18             // 102 – десятичный код символа 'f'
19             case 102: i -= 5; break;
20             default: ; break; // пустой оператор
21         }
22         System.out.print(report + i);
23     }
24 }
```

Результат выполнения программы:

```
Введите символ: f
Результат работы switch: -5
```

Замечание.

Получить десятичное число, соответствующее коду заданного символа с клавиатуры достаточно просто достаточно использовать инструкцию вида `System.out.print((int)'символ');`

Оператор `switch`:

- допускает использование без ключевого слова `break`;
- позволяет объединять повторяющиеся метки через запятую;
- применять лямбда-выражения;
- может возвращать значения.

Требования к оформлению условных операторов

Для `if`:

- После `if` должен идти наиболее вероятный вариант действий, а после else – наименее.
- Желательно использовать конструкцию `else` (даже с пустой инструкцией), либо ее отсутствие следует комментировать.
- Очень сложные условия следует выносить в отдельные функции
- В случае вложенных `if` первыми должны проверяться наиболее общие условия.
- Следует рассмотреть не имеет ли смысл заменить лестницу `if-else-if` на `switch`.
- Инструкции, выполняющиеся по условию, не следует располагать в одной строке с `if` или `else`.
- Не рекомендуется сравнивать переменные с плавающей точкой с нулем.

Для `switch`:

- все перечисляемы значения должны быть выстроены в логическом порядке (наиболее вероятные варианты размещаются раньше остальных).
- действия для любого из пунктов должны быть простыми, если необходимо, то следует организовать последовательность действий в виде пользовательской функции;
- `default` следует использовать для выполнения альтернативных действий, а также для выполнения действий по умолчанию;
- `default` следует использовать для вывода сообщения об ошибках.
- `default` рекомендуется использовать всегда даже если кроме пустой инструкции никакой последовательности действий нет.

Операторы циклов

При выполнении программы нередко возникает необходимость неоднократного повторения однотипных вычислений над различными данными. Для этих целей используются циклы. Цикл – участок программы,

в котором одни и те же вычисления реализуются неоднократно над различными значениями одних и тех же переменных (объектов).

Для организации циклов в Java используются следующие операторы:

- `for`,
- `while`,
- `do while`.

Оператор цикла `for`

Цикл `for` является циклом с параметрами и обычно используется в случае, когда известно точное количество повторов вычислений.

Оператор `for` – это наиболее общий способ организации цикла. Он имеет следующий формат:

```
for(выражение1; условие; выражение2) блок/инструкция;
```

`Выражение1` обычно используется для установления начального значения переменных, управляющих циклом. `Выражение2` обычно определяет изменение переменной, управляющей циклом после каждого выполнения *инструкции (или блока) составляющего тело цикла*. `Условие-выражение` – это конструкция, определяющая условие, при котором тело цикла (`блок/инструкция;`) будет выполняться.

`Выражение1` и `выражение2` могут содержать несколько инструкций, разделенных запятыми.

Пример.

```
1 public class Program{
2     public static void main (String args[]){
3         int i;
4         float b;
5         for(i = 1, b = -1; i <= 5; i++, b--)
6             // блок (составной оператор) не оформлен
7             System.out.print(" " + i * i + " " + b);
8     }
9 }
```

Результат работы программы:

```
1 -1.0 4 -2.0 9 -3.0 16 -4.0 25 -5.0
```

При инициализации начальных значений и в секции `выражение1` могут использоваться не только неименованные константы (литералы), но и арифметические выражения из переменных, значение которых уже определено к моменту передачи управления на цикл `for()`.

В разделе `выражение1` заголовка цикла `for()` допустимо объявлять параметр цикла и другие переменные, в этом случае объявленные параметры должны быть проинициализированы. При этом сколько бы не было объявлено переменных в разделе `выражение1` все они должны быть одного и того же типа.

Таким образом наименование типа в разделе `выражение1` используется только один раз в самом начале перечисления параметров.

Пример.

```
1 public class Program{
2     public static void main (String args[]){
3         for(int i = 1, b = -1; i <= 5; i++, b--)
4             System.out.print(" " + i * i + " " + b);
5     }
6 }
```

Результат работы программы такой же, как и в предыдущем случае.

Замечание.

Если хоть один параметр, объявленный в заголовке, ранее был уже объявлен, то возникнет ошибка компиляции, сообщающая о повторном объявлении параметра.

Переменные, объявленные в заголовке цикла, являются локальными и за пределами цикла не видны.

Схема выполнения оператора `for`:

1. вычисляется `выражение1`;
2. вычисляется `условие`, если значение `условия` имеет значение `true`, то выполняется тело цикла (`блок/инструкция;`);
3. вычисляется `выражение2`;
4. далее переход к пункту 2 данного алгоритма;

Выполнение оператора цикла прекращается, когда при очередной повторной проверке `условия` в п.2 получено условие равное `false`. В этом случае управление передается на инструкцию, следующую за оператором `for`.

Проверка условия всегда выполняется перед выполнение тела цикла. Это значит, что тело цикла может ни разу не выполниться, если `условие` сразу будет ложным.

Суммирование квадратов натуральных чисел до 10 включительно

Необходимо вычислить сумму вида:

$$\sum_{i=1}^{10} i^2 = 1 + 4 + \cdots + i^2 + \cdots + 100$$

Пример.

```
1  public class Program{
2      public static void main (String args[]){
3          int Sum = 0;
4          for(int i = 1; i <= 10; i++) {
5              Sum = Sum + i * i;       //или Sum += i * i;
6          }
7          System.out.print("Sum = " + Sum);
8      }
9  }
```

Результат работы программы:

```
Sum = 385
```

Замечание.

Синтаксисом цикла `for()` в языке Java допускаются и другие конструкции. В целом они полностью перекликаются с оформлением соответствующего оператора цикла в C++.

Вычисление в цикле факториала целого числа

Вычислим в цикле $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.

Пример.

```
1  import java.util.Scanner;
2
3  public class Program {
4      public static void main (String args[]) {
5          String call = "Введите натуральное число: ";
6          String report = "Факториал = ";
7
8          System.out.print(call);
9          Scanner objIn = new Scanner(System.in);
10         int n = objIn.nextInt();
11         objIn.close();
12         int product = 1;
13         for(int i = 2; i <= n; i++) {
14             product = product * i;
15         }
16         System.out.print(report + product);
17     }
18 }
```

Результат работы программы:

```
Введите натуральное число: 5
Факториал = 120
```

Двойной факториал целого числа

Вычислим в цикле $n!!$ (двойной факториал). Двойной факториал различается в определении для четного и нечетного числа n:

- для нечетного - $n!! = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot n$;
- для четного - $n!! = 2 \cdot 4 \cdot 6 \cdot \ldots \cdot n$.

Для решения поставленной задачи следует использовать не увеличение параметра цикла `i`, а его уменьшение причем уменьшение не на `1`, а на `2`.

Пример.

```
1  import java.util.Scanner;
2
```

```java
public class Program {
    public static void main (String args[]) {
        String call = "Введите натуральное число: ";
        String report = "Двойной факториал = ";

        System.out.print(call);
        Scanner objIn = new Scanner(System.in);
        int n = objIn.nextInt();
        objIn.close();
        int product = 1;
        for(int i = n; i > 1; i = i - 2) {
            product = product * i;
        }
        System.out.print(report + product);
    }
}
```

Результат работы программы:

```
Введите натуральное число: 5
Двойной факториал = 15
```

Возведение в целую степень переменной x в цикле

Вычислим в цикле $x^n = \underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{n \text{ раз}}$.

Пример.

```java
import java.util.Scanner;

public class Program {
    public static void main (String args[]) {
        String call = "Введите x и целую степень n: ";
        String report = "x в степени n = ";

        System.out.print(call);
        Scanner objIn = new Scanner(System.in);
        // ввод двух чисел
        double x = objIn.nextDouble();
        int n = objIn.nextInt();
        objIn.close();
```

```
        double product = 1;
        for(int i = 0; i < n; i++) {
            product = product * x;
        }
        System.out.print(report + product);
    }
}
```

Результат выполнения программы:

```
Введите x и целую степень n: 2 3
x в степени n = 8.0
```

Вычисление в цикле выражения с меняющимся знаком в зависимости от четности/нечетности степени

Вычислим в цикле выражение $(-1)^n \cdot x^n$.

Пример.

```
import java.util.Scanner;

public class Program {
    public static void main (String args[]) {
        String call = "Введите x и целую степень n: ";
        String report = "x в степени n = ";

        System.out.print(call);
        Scanner objIn = new Scanner(System.in);
        // ввод двух чисел
        double x = objIn.nextDouble();
        int n = objIn.nextInt();
        objIn.close();
        double product = 1;
        for(int i = 0; i < n; i++) {
            product = product * (-1) * x;
        }
        System.out.print(report + product);
    }
}
```

Результат выполнения программы:

```
Введите x и целую степень n: 2 3
x в степени n = -8.0
```

Оператор цикла `while`

Оператор цикла `while` называется циклом с предусловием и имеет следующий формат:

```
while (условие) блок/инструкция;
```

В качестве `условия` допускается использовать любое условное выражение языка Java, в качестве тела – любую инструкцию, в том числе пустую инструкцию или блок.

Схема выполнения оператора `while` следующая: вычисляется `условие`:

- если выражение ложно, то выполнение оператора `while` заканчивается и выполняется следующую по порядку инструкцию;
- если выражение истинно, то выполняется тело оператора - `блок/инструкция;`
- управление передаются на проверку условия.

Перепишем пример вычисления $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ с помощью оператора цикла `while`.

Пример.

```
1  import java.util.Scanner;
2
3  public class Program {
4     public static void main (String args[]) {
5        String call = "Введите натуральное число: ";
6        String report = "Факториал = ";
7
8        System.out.print(call);
9        Scanner objIn = new Scanner(System.in);
10       int n = objIn.nextInt();
11       objIn.close();
12       int product = 1;
13       int i = 2;
```

```
14          while(i <= n) {
15              product = product * i;
16              i++;
17          }
18          System.out.print(report + product);
19      }
20  }
```

Результат работы программы:

```
Введите натуральное число: 3
Факториал = 6
```

Схема бесконечного цикла

С помощью цикла `for`:

```
...
for(;true;) {
    //тело цикла
}
...
```

С помощью цикла `while`:

```
...
while(true) {
    //тело цикла
}
...
```

Организация бесконечного цикла может понадобиться разработчику для организации так называемой «защиты от дурака». Она заключается в бесконечном продолжении цикла, пока пользователь не введет значения, которые удовлетворяют ограничениям на используемый далее в вычислениях алгоритм.

Конкретные примеры реализации защиты будут рассмотрены ниже после того, как будут обсужден оператор `break`.

Оператор цикла do while

Оператор цикла `do while` называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид:

```
do {
     СписокИнструкций;
}
while (условие);
```

Схема выполнения оператора `do while`:

- выполняется тело цикла (`СписокИнструкций`);
- проверяется `условие`: если условие `ложно`, то выполнение оператора `do while` заканчивается и управление программой передается на следующую по порядку инструкцию; если выражение `истинно`, то управление передается на ключевое слово `do` и далее еще раз выполняется `СписокИнструкций`.

Вложенные циклы

Операторы `while`, `for` и `do while` могут быть вложенными.

Пример.

```
while(условие) {
    // списокИнструкций1;
    while(условие) {
        // списокИнструкций2;
    }
     // списокИнструкций3;
}
```

Требования к оформлению операторов циклов

- Рекомендуется даже для циклов с телом в виде одной инструкции использовать операторные скобки (оформлять в виде блока).
- Циклы без тела (с пустой инструкцией) не желательны.
- Инструкции, имеющие отношение к управлению параметром цикла, следует группировать в конце или начале тела цикла.

- Следует следить, чтобы цикл завершался при любых обстоятельствах.
- Условия, при которых завершается выполнение цикл должны быть максимально простыми.
- Количество уровней вложенности циклов не должно превышать трех.
- Желательно, чтобы цикл был такой длины, чтобы целиком помещался в поле экрана монитора.
- В заголовке оператора `for` должны участвовать только инструкции, определяющие начальное значение и порядок изменения параметра цикла.
- В теле цикла `for` не желательно изменять счетчик.
- Присвоение начальных значений переменных цикла `while` должно выполняться непосредственно перед заголовком цикла.
- По возможности следует избегать цикла `do while`.

Операторы перехода

Оператор `break`

Оператор `break` изменяет поток управления, он прерывает выполнение последовательности действий в блоке, в частности, в теле цикла. Его целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла.

Формат:

```
break;
```

Управление передается на инструкцию, следующую за блоком, в котором используется `break`.

Рассмотрим пример использования `break` в бесконечном цикле проверки условия корректности ввода данных.

Пример.

```
1  import java.util.Scanner;
2
3  public class Program {
4      public static void main (String args[]) {
5          String call ="Введите точность(0< число< 1): ";
```

```
6           String report =
7               "Точность удовлетворяет условиям и равна ";
8           double eps;
9
10          while(true) {
11              System.out.print(call);
12              Scanner objIn = new Scanner(System.in);
13              eps = objIn.nextDouble ();
14              objIn.close();
15              if ((0 < eps) & (eps < 1)) break;
16          }
17          System.out.print(report + eps);
18      }
19  }
```

Результат работы программы:

```
Введите точность(0< число< 1): 0.005
Точность удовлетворяет условиям и равна 0.005
```

break как замена goto

В Java нет оператора безусловного перехода `goto`. Для этого оператора `break` имеет особую форму записи:

```
break МЕТКА;
```

Обычно метка — это имя идентификатор, указывающий на инструкцию или блок кода. Управление программой после выполнения данного оператора передается на инструкцию/блок, перед которым указана метка в формате: `МЕТКА:` .

Замечание.
Конструкция не используется.

Оператор continue

Оператор `continue`, в отличие от `break` передает управление на проверку условия продолжения цикла.

Формат оператора:

```
continue;
```

Организация вычислений с точностью

Суммирование отрезка ряда с точностью

При реализации многих численных методов точность вычислений зависит от числа шагов. Однако за какое именно число шагов будет достигнута приемлемая точность, заранее сказать трудно и желательно, чтобы программа сама определяла, когда следует остановиться. Например, это касается алгоритма вычисления значения любой из математических функций (например, `sin`, `cos`, …), используя разложение в ряд Тейлора.

Рассмотрим один из простейших вариантов ряда Тейлора:

$$\frac{1}{1-x} = \sum_{0}^{\infty} x^i = 1 + x^2 + x^3 + x^4 + \cdots$$

Чем большее количество членов ряда будет просуммировано, тем точнее будет вычислено значение функции, представленной этим рядом. Пусть требуется вычислить до 2-го знака после запятой, т.е. приемлемая погрешность $\varepsilon = 5 \cdot 10^{-3}$. Для этого достаточно суммировать члены ряда до тех пор, пока очередной член ряда не окажется меньше ε, т.е. вычисления суммы проводить пока $x^i > \varepsilon$.

Достаточным условием сходимости практически всех рядов Тейлора является малость модуля аргумента x относительно 1, т.е. должно выполняться условие $|x| < 1$.

В отличии от ранее рассмотренных примеров в цикле выполняется не одно действие (накопление суммы или возведение в степень), а сразу два: число x возводится в степень и результат суммируется. Этот прием существенно уменьшает как количество операций, так и время вычисления значения функции.

Пример.

```
1  import java.util.Scanner;
2
3  public class Program {
4      public static void main (String args[]) {
5          String callX = "Введите x (0< |x|< 1): ";
6          String callEps ="Введите eps (0< eps< 0.05): ";
7          String report = "Сумма ряда равна ";
8          double x;
9          double eps;
```

```
10          Scanner objIn = new Scanner(System.in);
11          // раздел проверки корректности данных
12          while(true) {
13              // сообщение о вводе x
14              System.out.print(callX);
15              // ввод переменной x
16              x = objIn.nextDouble ();
17              // условие на переменную x
18              boolean xTrue = (-1 < x) & (x < 1);
19              // сообщение о вводе точности
20              System.out.print(callEps);
21              // ввод точности eps
22              eps = objIn.nextDouble ();
23              // условие на точность eps
24              boolean epsTrue = (0 < eps) & (eps < 0.05);
25              // проверка условий и выход из беск. цикла
26              if (epsTrue & xTrue) break;
27              System.out.print("Попробуйте еще раз…\n");
28          }
29          objIn.close();
30          // алгоритм вычисления суммы с точностью
31          double sum = 0,
32                 xPower = 1;
33          while( Math.abs(xPower) > eps) {
34              sum = sum + xPower; //суммирование чл. ряда
35              xPower = xPower * x; //возведение в степень
36          }
37          // вывод результата
38          System.out.print(report + sum);
39      }
40  }
```

Вычисление значения ряда с точностью описывается таблицей (Таблица 1).

Таблица 1 – Вычисление отрезка ряда с точностью

Переменная/ действие	Значения			
sum	0	0 + 1 = 1	1 + x	1+x + x^2
xPower	1	1 · x = x	x · x = x^2	x^2 · x = x^3
xPower > eps	true	true	true	true

Результат работы программы:

```
Введите x (0< |x|< 1): 0.5
Введите eps (0< eps< 0.05): 0.0005
Сумма ряда равна 1.9990234375
```

Точность вычислений согласно рекуррентным уравнениям

Рекуррентными соотношениями или уравнениями называются уравнения вида:

$$x_n = F(x_{n-1}, \dots, x_0)$$

где $F(\ \ )$ - некоторая функция, x_i – значение, вычисленное на i-ом шаге ($i = \overline{1,n}$). Таким образом, в рекуррентной записи каждое следующее значение вычисляется по предыдущим значениям. Поэтому одним из основных практических вопросов является вопрос выбора как начального x_0, так и других необходимых значений для запуска алгоритма.

Если вычисления производятся в соответствии с рекуррентными соотношениями, то используется другой способ поставить связанное с точностью условие прекращения вычислений. Оно заключается в том, что вычисления прекращаются, если изменение вычисляемой величины на очередном шаге $n + 1$ меньше заданной величины ε:

$$|x_{n+1} - x_n| < \varepsilon.$$

Также условие можно наложить на относительное изменение:

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| < \varepsilon.$$

При этом необходимо особо подчеркнуть, что за вычисленное значение принимается x_n (но не x_{n+1}).

В качестве примера следует рассмотреть рекуррентную формулу Ньютона вычисления корня k-ой степени из числа a (т.е. $\sqrt[k]{a}$):

$$x_{n+1} = \frac{k-1}{k} x_n - \frac{a}{k \cdot {x_n}^{k-1}},$$

где $x_0 = a$.

Пример:

```java
import java.util.Scanner;

public class Program {
    public static void main (String args[]) {
        String callA = "Введите положительное a: ";
        String callK = "Введите целую степень корня: ";
        String callEps ="Введите eps (0< eps< 0.05): ";
        String report = "Корень из a равен ";
        double a, eps;
        int k;
        // раздел проверки корректности данных
        Scanner objIn = new Scanner(System.in);
        while(true) {
            // сообщение о вводе a
            System.out.print(callA);
            // ввод переменной a
            a = objIn.nextDouble ();
            // условие на переменную x
            boolean aTrue = 0 < a;
            // сообщение о вводе степени корня k
            System.out.print(callK);
            // ввод степени корня k
            k = objIn.nextInt();
            // условие на степень корня k
            boolean kTrue = 0 < k;
            // сообщение о вводе точности
            System.out.print(callEps);
            // ввод точности eps
            eps = objIn.nextDouble ();
            // условие на точность eps
            boolean epsTrue = (0 < eps) & (eps < 0.05);
            // проверка условий и выход из беск. цикла
            if (aTrue & kTrue & epsTrue) break;
            System.out.print("Попробуйте еще раз…\n");
        }
        objIn.close();
        // алгоритм вычисления корня с точностью
        double xn = a;
        double xnn = (k - 1) / (double) k * xn +
                     a / (k * Math.pow(xn, k - 1));
        while(Math.abs(xnn - xn) > eps) {
            xn = xnn; // переход к следующей итерации
            xnn = (k - 1) / (double) k * xn +
                  a / (k * Math.pow(xn, k - 1));
        }
```

```
46          // вывод результата
47          System.out.print(report + xn);
48      }
49  }
```

Результат работы программы:

```
Введите положительное a: 4
Введите целую степень корня: 2
Введите eps (0< eps< 0.05): 0.0005
Корень из a равен 2.0000000929222947
```

Литература

1. Блинов, И.Н. Java from EPAM / И.Н. Блинов, В.С. Романчик. - Минск: Четыре четверти, 2020. - 560 с.
2. Руководство по языку программирования Java/ METANIT.COM - [Электронный ресурс], URL: https://metanit.com/java/tutorial/ (дата обращения: 08.06.22)
3. Самоучитель по Java с нуля/ Vertex Academy - [Электронный ресурс], URL: https://vertex-academy.com/tutorials/ru/samouchitel-po-java-s-nulya/ (дата обращения: 08.06.22)
4. Java — Самоучитель / ProgLang - [Электронный ресурс], URL: http://proglang.su/java (дата обращения: 08.06.22)
5. 30 лучших онлайн курсов Java программирования / HashNets - [Электронный ресурс], URL: https://hashnets.com/30-лучших-онлайн-курсов-на-платформе-java/ (дата обращения: 08.06.22)
6. Учебники по Java/ Oracle - [Электронный ресурс], URL: https://docs.oracle.com/javase/tutorial/index.html (дата обращения: 08.06.22)

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

ЯЗЫК JAVA

ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОГРАММОЙ

**Учебные материалы
для студентов II курса
специальности 1-31 03 08 «Математика
и информационные технологии
(по направлениям)»**

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 25.11.2022. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 1,86. Уч.- изд. л. 1,76. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.