Лекция 4

Условный оператор
Циклы
Примеры

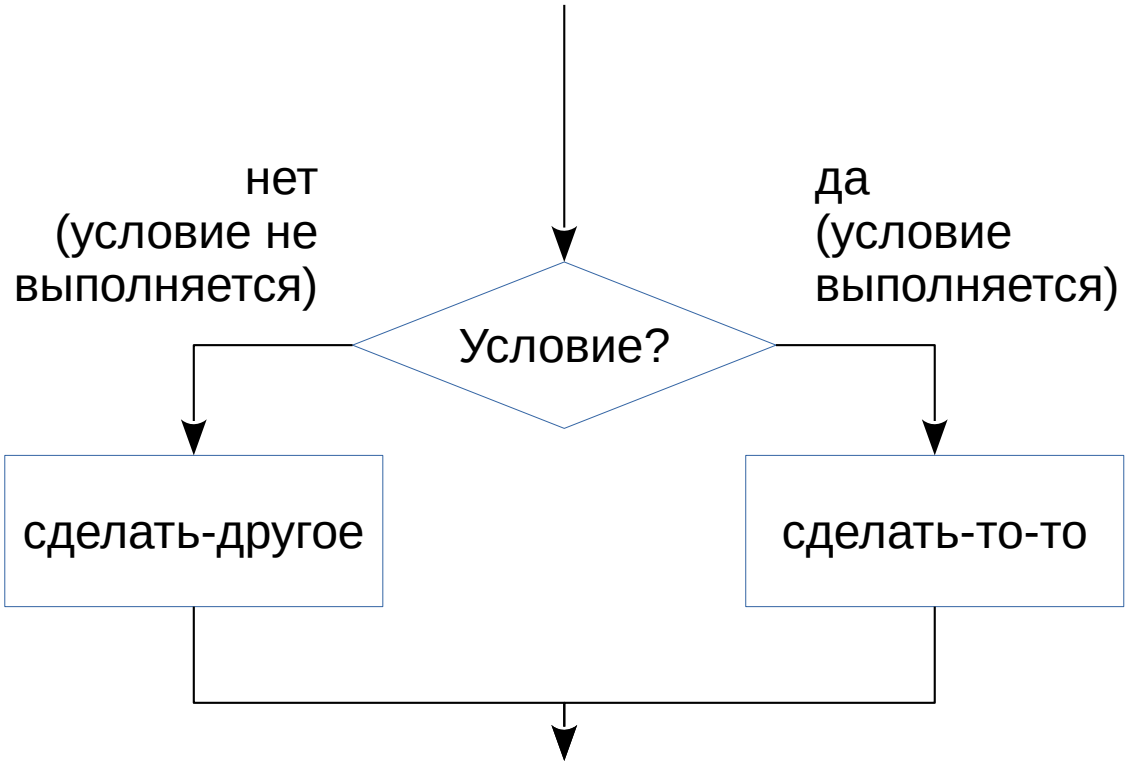
Условный оператор

- До сих пор (на лекциях) мы рассматривали (за исключением программы решения квадратного уравнения) только линейные программы (в которых действия всегда выполняются одно за другим последовательно — 1, 2, 3 и т. д.)
- В реальном мире многие действия выполняются только при соответствующих условиях (т.е. иногда, когда условия подходящие, выпоняются; условие не подходящие — не выполняются)
  - Корни квадратного уравнения стоит вычислять, если D >= 0
  - Если есть пельмени — готовим пельмени, иначе пьем чай
  - Приходим пересдавать экзамен, только если экзамен не сдан
- Условный оператор позволяет реализовать подобную логику в программах

Картинка в тему :)

Общий вид условного оператора

```
if (условие)
    сделать-то-то;
else
    сделать-другое;
```



Или (эквивалентно):

```
if (условие) {
    сделать-то-то;
} else {
    сделать-другое;
}
```

Согласно рекомендациям оформления кода Java даже единственный оператор в then и else принято обрамлять скобками

Варианты условного оператора

```
if (условие) {
    сделать-то-то;
}
```

Неполная форма условного оператора
(отсутствует секция else)

Желательны

```
if (условие) {
    сделать-то-то;
} else {
    сделать-другое;
}
```

Обязательны

```
if (x < 0) {
    сделать-одно;
    сделать-второе;
    сделать-что-то-ещё;
}
```

```
if (Math.abs(d) < EPS) {
    сделать-одно;
    сделать-второе;
} else {
    сделать-другое;
    и-ещё-кое-что;
}
```

- Если сделать-то-то состоит из нескольких действий, то надо в обязательном порядке использовать составной оператор (несколько действий, объединенных фигурными скобками - см. справа)

Варианты условного оператора

```
if (условие) {
    сделать-то-то;
}
```

Неполная форма условного оператора
(отсутствует секция else)

```
if (условие) {
    сделать-то-то;
} else {
    сделать-другое;
}
```

```
if (x < 0) {
    сделать-одно;
    сделать-второе;
    сделать-что-то-ещё;
}
```

```
if (Math.abs(d) < EPS) {
    сделать-одно;
    сделать-второе;
} else {
    сделать-другое;
    и-ещё-кое-что;
}
```

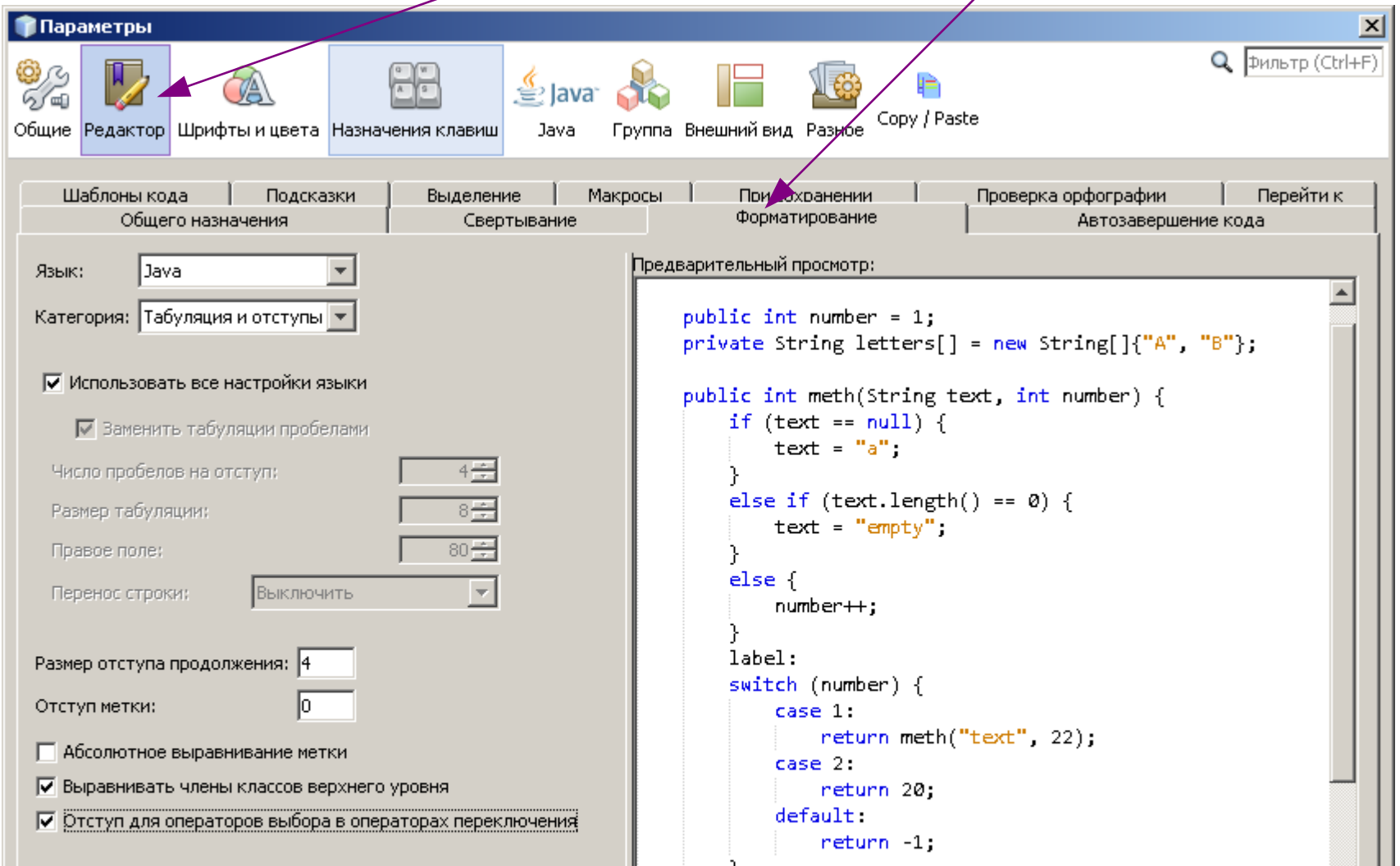
- Действия, которые выполняются в зависимости от условия, являются являются «вложенными» относительно условного оператора, а также инструкций программы, которые следуют до условного оператора и после
- Такие «вложенные» конструкции для читаемости программы (выделении структуры) обязательно оформляются увеличением отступа от левого края
- Естественно, действия — с новой строки

Про отступы

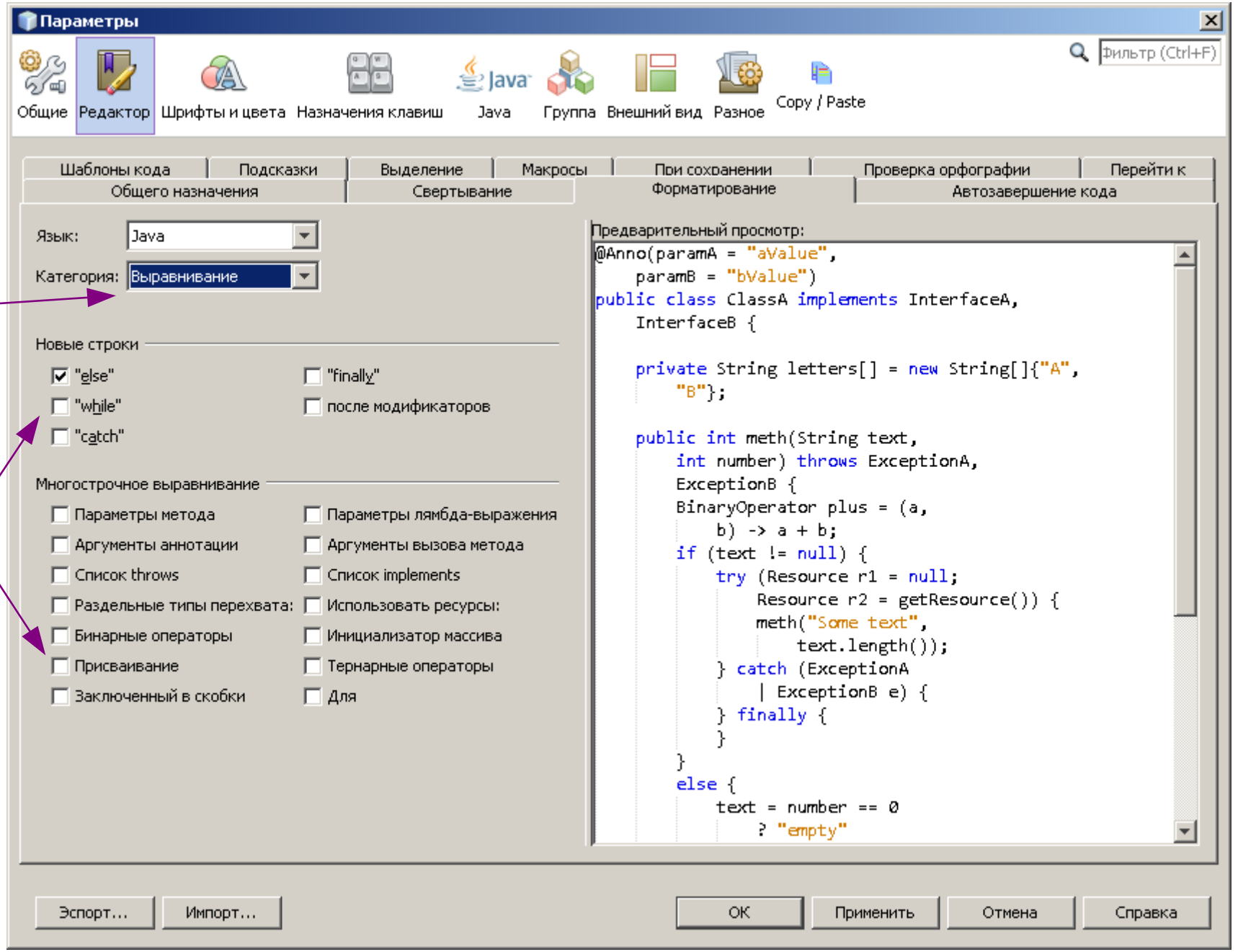
- Отступы можно выделять как знаком табуляции (символ с кодом 9), так и соответствующим кол-вом пробелов
- Любой современный редактор можно настроить, на сколько символов сдвигать табуляцию
- В одной модуле (файле) отступы должны быть оформлены единообразно – либо табуляции, либо пробелы
  - В противном случае при просмотре / редактировании программы в редакторе с другими настройками (кол-вом пробелов на табуляцию) текст программы «поплывет» (нарушится видимая структура)
- NetBeans по умолчанию заменяет табуляции пробелами

Про отступы

- В среде NetBeans можно настроить отступы и другие параметры форматирования кода:
  - Меню: Сервис -> Параметры; раздел: Редактор; закладка: Форматирование (в разных версиях NetBeans этот порядок может несколько отличаться):



Настройки форматирования кода



Группы параметров

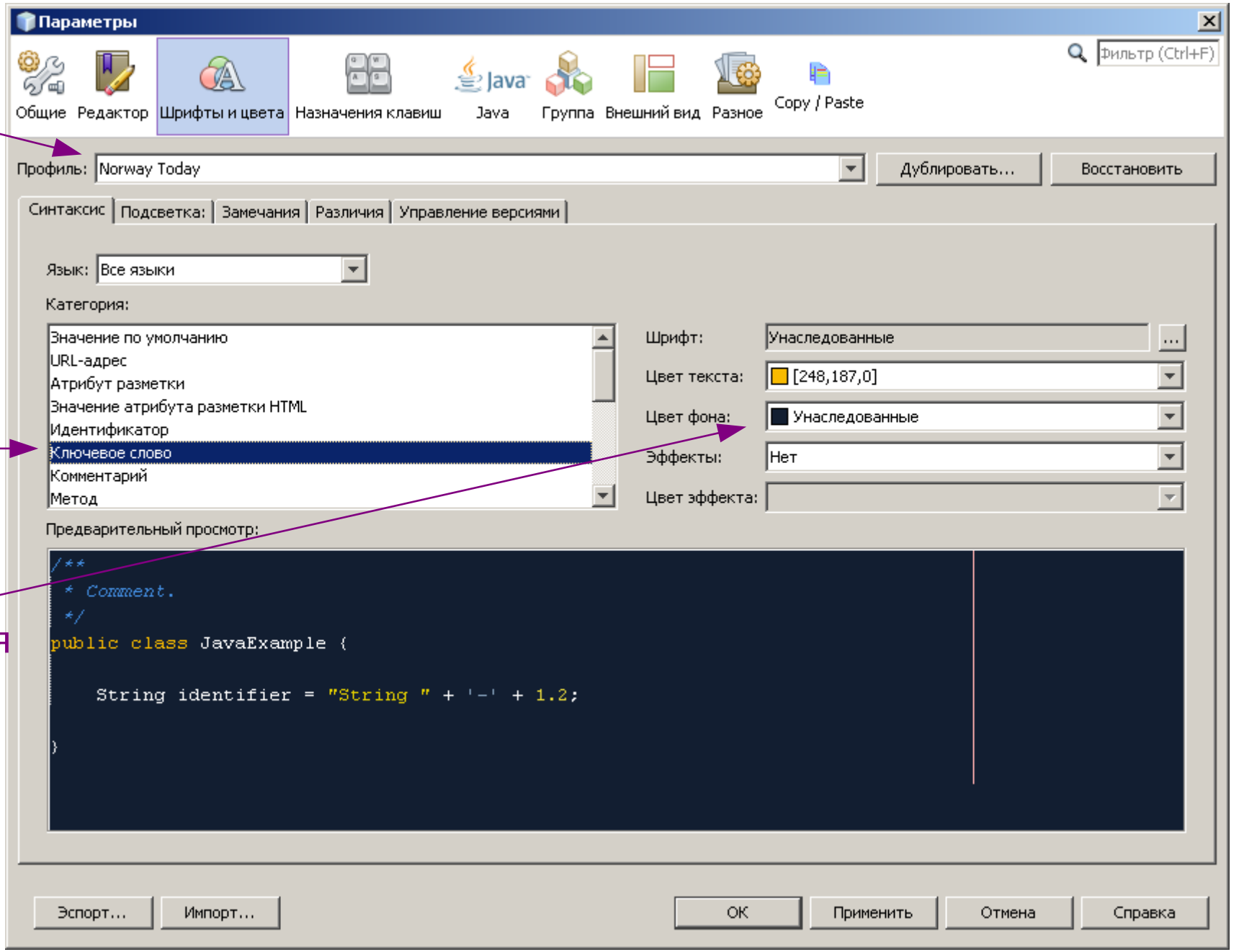
Отдельные параметры

Цветовая схема

Профиль (цветовая схема)

Отдельные элементы языка

Настройки отображения



Решение квадратного уравнения

- Решение квадратного уравнения включает информацию:
  - кол-во корней
  - x1 (если корни есть)
  - x1 (если два корня)
- Опишем класс для представления решения:

```
class QuadrEquationSolution {

    public int n;
    public double x1;
    public double x2;

    public QuadrEquationSolution() {
    }

    public QuadrEquationSolution(int n, double x1, double x2) {
        this.n = n;
        this.x1 = x1;
        this.x2 = x2;
    }
}
```

Решение квадратного уравнения

```
/* Функция для решения квадратного уравнения,
 * возвращает объект QuadrEquationSolution,
 * содержащий n (кол-во корней), x1 и x2
 */
public static QuadrEquationSolution solveQuadrEquation(
    double a, double b, double c
) {
    double d = b * b - 4 * a * c;
    if (Math.abs(d) < EPS) {  // d == 0, одно решение (провер. первым)
        double x = -b / (2 * a);
        return new QuadrEquationSolution(1, x, x);
    } else {
        if (d < 0) {  // нет решений
            return new QuadrEquationSolution(0, 0, 0);
        } else {  // два решения (последний вариант)
            QuadrEquationSolution res = new QuadrEquationSolution(2, 0, 0);
            res.x1 = (-b - Math.sqrt(d)) / (2 * a);
            res.x2 = (-b + Math.sqrt(d)) / (2 * a);
            return res;
        }
    }
}
```

Еще раз про отступы

- На предыдущем слайде вложенный IF по формальным правилам выделения отступами вложенных конструкций смещен вправо
- Однако в данном случае с точки зрения логики человека имеют место быть три равноценных относительно друг друга варианта выполнения программы (один корень, два корня, нет корней)
- В таких случаях (равноценности более двух вариантов хода выполнения программы) допустимо (и даже желательно) отказаться от формальных правил оформления отступов и записать программу в следующем виде (оформив все три варианта одинаковым смещением от края):

```
if (Math.abs(d) < EPS) {  // d == 0, одно решение (провер. первым)
    double x = -b / (2 * a);
    return new QuadrEquationSolution(1, x, x);
} else if (d < 0) {  // нет решений
    return new QuadrEquationSolution(0, 0, 0);
} else {  // два решения (последний вариант)
    QuadrEquationSolution res = new QuadrEquationSolution(2, 0, 0);
    res.x1 = (-b - Math.sqrt(d)) / (2 * a);
    res.x2 = (-b + Math.sqrt(d)) / (2 * a);
    return res;
}
```

Решение квадратного уравнения (более удачное форматирование)

```
/* Функция для решения квадратного уравнения,
 * возвращает объект QuadrEquationSolution,
 * содержащий n (кол-во корней), x1 и x2
 */
public static QuadrEquationSolution solveQuadrEquation(
    double a, double b, double c
) {
    double d = b * b - 4 * a * c;
    if (Math.abs(d) < EPS) {  // d == 0, одно решение (провер. первым)
        double x = -b / (2 * a);
        return new QuadrEquationSolution(1, x, x);
    } else if (d < 0) {  // нет решений
        return new QuadrEquationSolution(0, 0, 0);
    } else {  // два решения (последний вариант)
        QuadrEquationSolution res = new QuadrEquationSolution(2, 0, 0);
        res.x1 = (-b - Math.sqrt(d)) / (2 * a);
        res.x2 = (-b + Math.sqrt(d)) / (2 * a);
        return res;
    }
}
```

Решение квадратного уравнения (ввод данных, решение, распечатка)

```
/* Программа решения квадратного уравнения с вводом данных и печатью ответ
 */
public static void quadrEquationDemo() {
    double a = readDoubleValueFromConsole("a"),
        b = readDoubleValueFromConsole("b"),
        c = readDoubleValueFromConsole("c");

    QuadrEquationSolution sol = solveQuadrEquation(a, b, c);

    if (sol.n == 0) {
        System.out.println("Нет решшений");
    } else if (sol.n == 1) {
        System.out.printf("Одно решение: x1 = %.03f%n", sol.x1);
    } else {
        System.out.printf("Два решения: x1 = %1$.03f, x2 = %2$.03f%n",
            sol.x1, sol.x2);
    }
}
```

Тип треугольника
(решение без применения мозга)

```
/* Функция для определения типа треугольника по 3-м сторонам, возвращает
 *   0 - треугольник не существует
 *   1 - остроугольный
 *   2 - прямоугольный
 *   3 - тупоугольный
 */
public static int getTriangleType(double a, double b, double c) {
    if (a >= c + b || b >= a + c || c >= a + b) {
        return 0;
    }
            // EPS – константа 1E-9 (объявлена выше)
    if (Math.abs(a * a + b * b - c * c) < EPS
            || Math.abs(a * a + c * c - b * b) < EPS
            || Math.abs(b * b + c * c - a * a) < EPS)
        return 2;
    } else if (a * a + b * b >= c * c
            || a * a + c * c >= b * b
            || b * b + c * c >= a * a) {
        return 1;
    } else {
        return 3;
    }
}
```

Пример демонстрирует запись составных условий в условном операторе (один из возможных вариантов форматирования)

Пример демонстрирует, что вместо записи вложенных условных операторов часто удобно рассматривать некоторые случаи отдельно (невозможность существования треугольника) и «отбрасывать» их до выполнения «основного» кода

Тип треугольника
(включаем голову и вспоминаем о декомпозиции)

- Вместо того, чтобы рассматривать всевозможные случаи, часто удобно предварительно подготовить данные
  - В нашем конкретном случае добиться, чтобы c >= a и c >= b
  - Возможный вариант:

```
// если бы мы знали массивы,
// записали бы все в массив и отсортировали

double a2 = Math.min(Math.min(a, b), c);
double c2 = Math.max(Math.max(a, b), c);
double b2 = a + b + c - a2 - c2;
a = a2; b = b2; c = c2;
```

Тип треугольника
(тип возвращаемого значения)

- Вместо того, чтобы возвращать целое значение и где-то помнить об его интерпретации (1 – остроугольный и т. п.), более правильным было описать тип данных, который включал бы только допустимые значения (и был бы самодокументированным)
- В Java для этого предусмотрены перечисления

```
/* Перечисление видов треугольника
 */
enum TriangleType {
    NOT_TRIANGLE,  // не треугольник
    ACUTE,         // остроугольный
    RIGHT,         // прямоугольный
    OBTUSE         // тупоугольный
}
```

В перечислениях можно описывать и методы, но мы пока эту возможность рассматривать не будем

Тип треугольника
(конечный вариант решения)

```
public static TriangleType getTriangleType(double a,
        double b, double c) {
    // если бы мы знали массивы,
    // записали бы все в массив и отсортировали
    double a2 = Math.min(Math.min(a, b), c);
    double c2 = Math.max(Math.max(a, b), c);
    double b2 = a + b + c - a2 - c2;
    a = a2; b = b2; c = c2;

    if (c >= a + b) {
        return TriangleType.NOT_TRIANGLE;
    }

    if (Math.abs(a * a + b * b - c * c) < EPS) {
        return TriangleType.RIGHT;
    } else if (a * a + b * b >= c * c) {
        return TriangleType.ACUTE;
    } else {
        return TriangleType.OBTUSE;
    }
}
```

Возвращать строку вида "прямоугольный" и т.п. было бы категорически неправильно, т.к. это относится к представлению результата, а не к собственно результату (а представление может быть на разных языках и т.п.)

Оператор switch – частный случай составного условного оператора

```
/* Программа определения вида треугольника с вводом данных и печатью ответа
 */
public static void triangleTypeDemoWithCase() {
    double a = readDoubleValueFromConsole("a"),
        b = readDoubleValueFromConsole("b"),
        c = readDoubleValueFromConsole("c");

    TriangleType type = getTriangleType(a, b, c);
    switch (type) {
    case NOT_TRIANGLE:
        System.out.println("Не треугольник");
        break;
    case ACUTE:
        System.out.println("Остроугольный");
        break;
    case RIGHT:
        System.out.println("Прямоугольный");
        break;
    case OBTUSE:
        System.out.println("Тупоугольный");
        break;
    default:  // невозможная ситуация, но для демонстрации
        System.out.println("Ошибка программы");
        break;
    }
}
```

- Используется, когда надо сравнивать одно значение с множеством констант (чисел или перечислений)
- Считается, что в некоторых случаях улучшает читаемость
- Каждый case должен заканчиваться break
- Можно написать несколько case с разными значениями подряд
- Используется довольно редко

Тернарный оператор (?:) — частный случай условного оператора

```
double a = readDoubleValueFromConsole("a"),
       b = readDoubleValueFromConsole("b");

double max;
if (a > b) {
    max = a;
} else {
    max = b;
}

// условный оператор можно переписать в виде
// (выражение принимает a при выполнении условия,
//  b – в противном случае)
double max = (a > b) ? a : b;
```

Скобки в данном случае не обязательны, но улучшают читаемость кода

Тернарный оператор, читаемость

- Тернарный оператор при грамотном использовании сокращает код и улучшает читаемость
- Может быть вложенным (в таком виде по поводу читаемости — вопрос спорный, однако может быть применен при соответствующем форматировании — пример ниже)

- Спорное в плане читаемости применение тернарного оператора

```
return c >= a + b ?
            TriangleType.NOT_TRIANGLE :
            Math.abs(a * a + b * b - c * c) < EPS ?
                TriangleType.RIGHT :
                a * a + b * b >= c * c ?
                    TriangleType.ACUTE :
                    TriangleType.OBTUSE
;
```

Запомнить и осознать

- Перефразируя известную фразу известного человека:
  - Читаемость
  - Читаемость
  - И еще раз читаемость

- Тем более, что с каждым разом вы будете писать все более сложные программы с различными вариантами вложенности условных операторов и циклов

Циклы

- В реальном мире многие действия выполняются многократно (какое-то конкретное кол-во раз, до достижения нужного результата)
  - (Готовим салат) порезать 5 огурцов
  - Пока не отчислили и не сдан экзамен, каждый вечер учить предмет и приходить на каждую пересдачу

- Циклы позволяют реализовать подобную логику в программах

Циклы в Java

- С предусловием (while)
  - while (условие) действие
- С постусловием (do)
  - do действие while (условие)
- for
  - for (инициализация; условие; след. шаг) действие

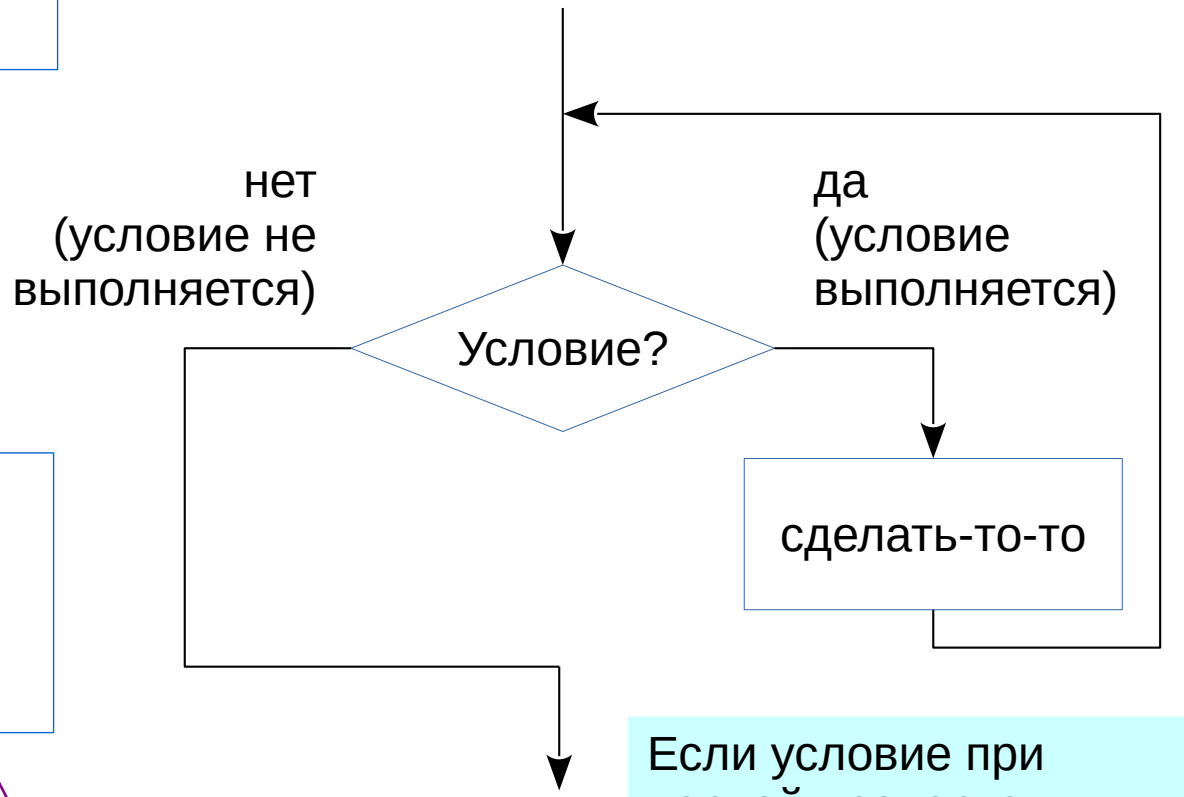
Общий вид цикла while

```
while (условие)
    сделать-то-то;
```

тело цикла

Или (эквивалентно):

```
while (условие) {
    сделать-то-то;
}
```



Согласно рекомендациям оформления кода Java даже единственный оператор в then и else принято обрамлять скобками

Если условие при первой проверке окажется ложным (не выполнится), то цикл (тело цикла) не выполнится ни одного раза

Варианты while

```
while (условие)
    сделать-то-то;
```

- Если сделать-то-то состоит из нескольких действий, то надо обязательно использовать составной оператор (несколько действий, объединенных фигурными скобками - см. справа) — правила такие же, как для условного оператора

```
while (x > 0) {
    сделать-одно;
    сделать-второе;
    сделать-что-то-ещё;
}
```

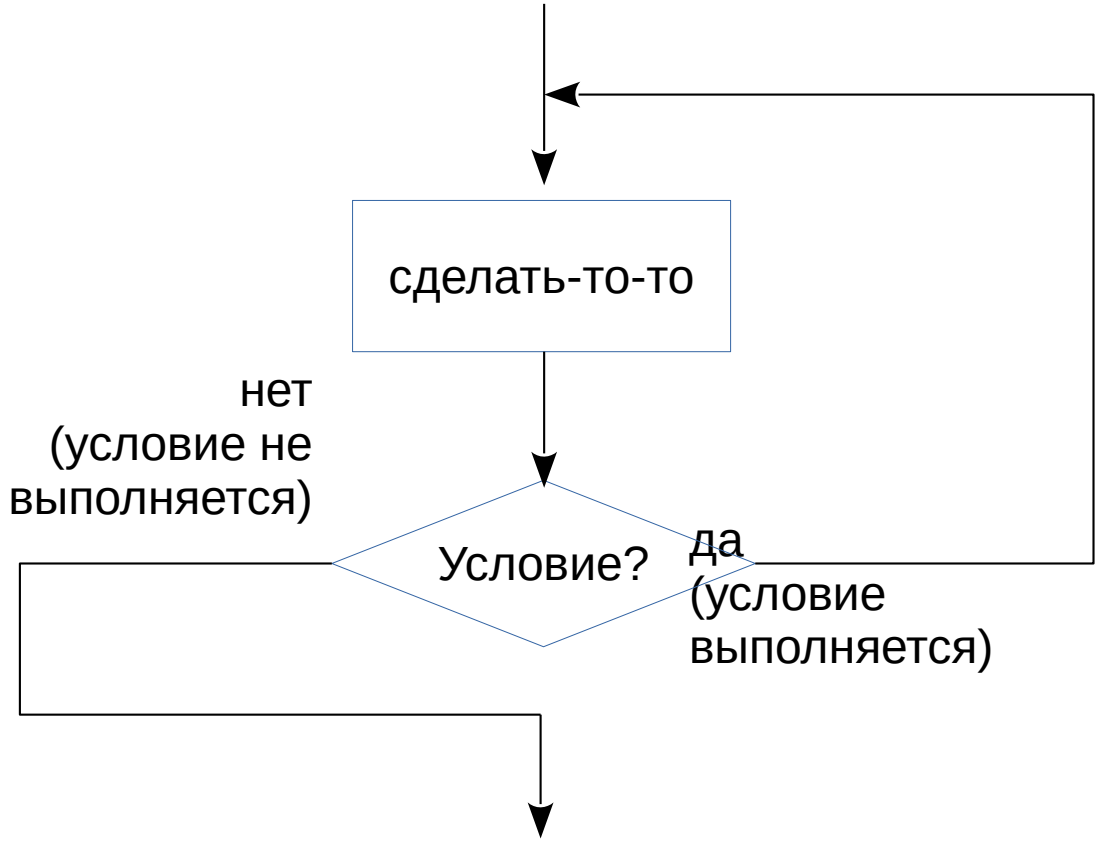
```
int a = 0;
// напечатает числа от 0 до 9
while (a < 10) {
    System.out.println(a);
    a++;  // a = a + 1
}

// напечатает числа от 10 до 1
while (a > 0) {
    System.out.println(a--);
}
```

Общий вид цикла do

```
do
    сделать-то-то
while (условие);
```

тело цикла



Или (эквивалентно):

```
do {
    сделать-то-то
} while (условие);
```

Цикл (тело цикла) всегда выполнится хотя бы один раз

Используется относительно редко

Пример цикла do – угадай число

```
/* Программа, предлагающая пользователю угадать число от 1 до 100
*/
public static void guessNumberDemo() {
    Scanner input = new Scanner(System.in);
    Random rnd = new Random();
    int rightNum = rnd.nextInt(100) + 1;

    System.out.println("Угадайте число от 1 до 100:");
    int stepsCount = 0;
    int num;
    do {
        stepsCount++;
        System.out.print(": ");
        num = input.nextInt();
        if (rightNum > num) {
            System.out.println("    Загаданное число больше");
        }
        else if (rightNum < num) {
            System.out.println("    Загаданное число меньше");
        }
        else {
            System.out.printf("    Угадали за %d попыток!%n", stepsCount);
        }
    }
    while (num != rightNum);
}
```

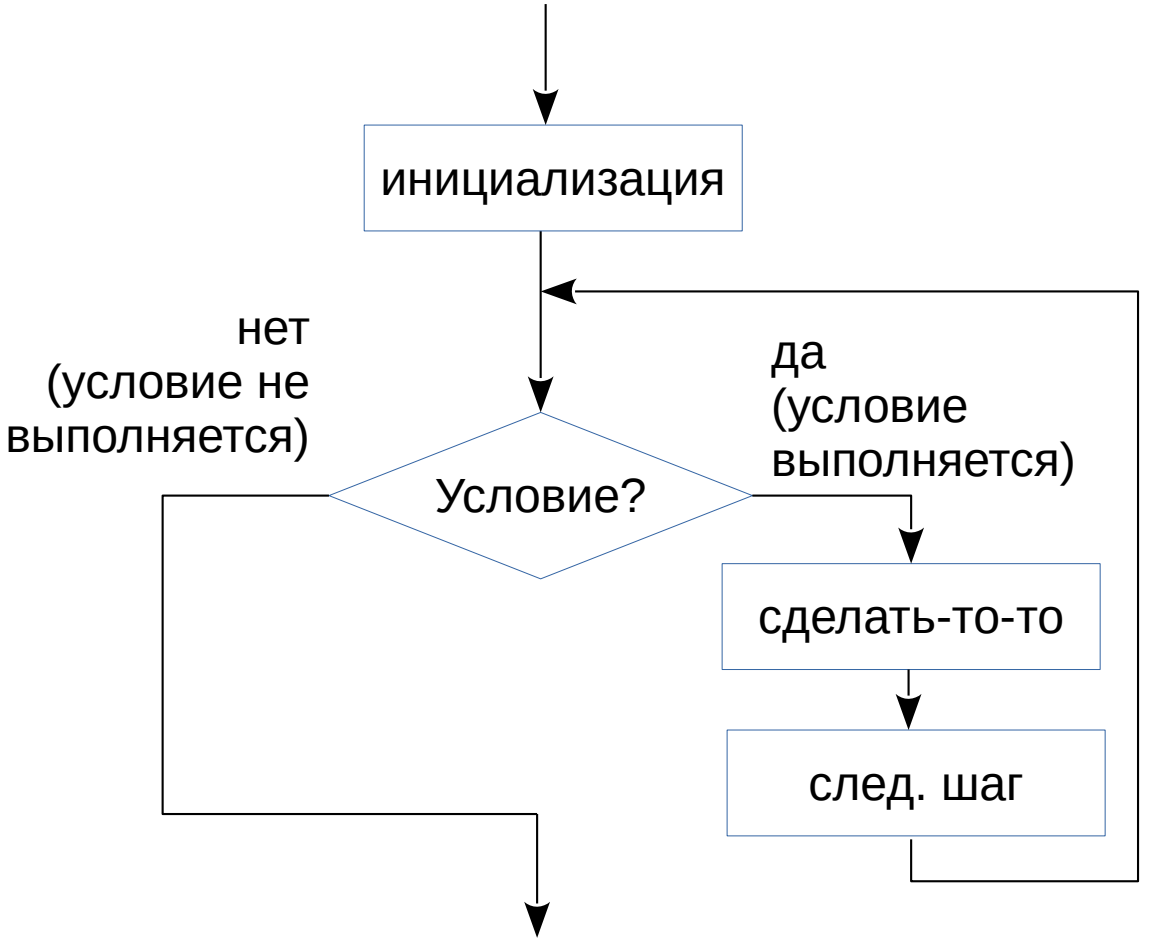
Для возможности использования класса Random необходимо импортировать java.util.Random

Общий вид цикла do

```
for (инициализация; условие; след. шаг)
    сделать-то-то;
```

тело цикла

Если условие при первой проверке окажется ложным (не выполнится), то цикл (тело цикла) не выполнится ни одного раза



Цикл for

- Типичным вариантом использования цикла for является итерация заранее известное кол-во раз
- В секции инициализации можно объявлять новые переменные

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- В секции инициализации и след. шаг возможны несколько действий, записанных через запятую:

```
int x, y;
for (x = 2, y = 100000; y > x; x *= 2, y /= 10) {
    System.out.printf("x = %d, y = %d%n", x, y);
}
```

- Любая секция for может быть опущена

```
// a, b - параметры
for ( ; a != 0 && b != 0; ) {
    if (a > b) {
        a %= b;
    } else {
        b %= a;
    }
}
int r = a + b;
System.out.println(r);
```

Алгоритм Эвклида нахождения НОД

Операторы break и continue

- break прерывает выполнение цикла
- continue прерывает выполнение тела цикла и запускает следующую итерации цикла

```
System.out.println("before loop");
for (int i = 0; ; i++) {
    if (i > 10) {
        break;
    }
    if (i % 2 == 0) {
        continue;
    }
    System.out.println(i);
}
System.out.println("after loop");
```

Напечатает:

```
before loop
1
3
5
7
9
after loop
```

Запомнить и осознать

- Не пытаться кого-то удивить своим знанием синтаксиса
  - Пишите максимально просто и понятно
  - Используйте различные «фишки» языка только если они делают код понятнее, например:

```
while (true) { ... }
```

все же понятнее, чем

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

Пример: треугольник из звездочек высоты h

- См. код в проекте Lect4Samples

```
/* печать трегольника из "звездочек" высоты h
*/
public static void printStarTriangle1(int h) {
    PrintStream out = System.out;
    for (int i = 0; i < h; i++) {
        for (int j = 0; j <= i; j++) {
            out.print("*");
        }
        out.println();
    }
}
```

Необходимо импортировать java.io.PrintStream

Пример: бинарное представление int

- См. код в проекте Lect4Samples

```
/* Собственная реализация представления int в бинарном виде в виде строки
*/
public static String intToBinV0(int n) {
    String result = "";
    for (int i = Integer.SIZE - 1; i >= 0; i--) {
        result += (n >> i) & 1;
    }
    return result;
}

/* Собственная реализация представления int в бинарном виде в виде строки
 * (при множественной конкатенации строк эффективнее использовать
 *  StringBuilder)
 */
public static String intToBin(int n) {
    StringBuilder result = new StringBuilder();
    for (int i = Integer.SIZE - 1; i >= 0; i--) {
        result.append((n >> i) & 1);
    }
    return result.toString();
}
```

Пример: собственная реализация sqrt

- См. код в проекте Lect4Samples

```
/* Собственная реализация sqrt
*/
public static double sqrt(double x) {
    final double EPS = 1E-9;

    if (x < 0) {
        return Double.NaN;
    }

    // double a = (x < 1) ? 0 : 1,
    //        b = (x < 1) ? 1 : x;
    double a = 1, b = x;
    if (x < 1) {
        a = 0;
        b = 1;
    }
```

продолжение:

```
    while (b - a > EPS) {
        double c = (a + b) / 2;
        if (c * c > x) {
            b = c;
        } else {
            a = c;
        }
    }

    return (a + b) / 2;
}
```

Пример: функция getStringPart

- См. код в проекте Lect4Samples

```
public static String getStringPart(String str, int partIndex) {
    int from = 0;
    for (int i = 0; i <= partIndex; i++) {
        int commaPos = str.indexOf(',', from);
        if (commaPos >= 0) {
            if (i == partIndex) {
                return str.substring(from, commaPos - from).trim();
            } else {
                from = commaPos + 1;
            }
        } else {
            if (i == partIndex) {
                return str.substring(from).trim();
            } else {
                return null;
            }
        }
    }
    // чтобы просто не ругался компилятор (или если i < 0)
    return null;
}
```

Совет (настоятельная рекомендация)

- Обязательно смотрите проекты-примеры к лекции (и экспериментируйте с ними):
  - Lect4Samples