Занятие 2

Anastasiya Solodkaya, Denis Stepulenok

LevelUP

17 ноября 2016 г.

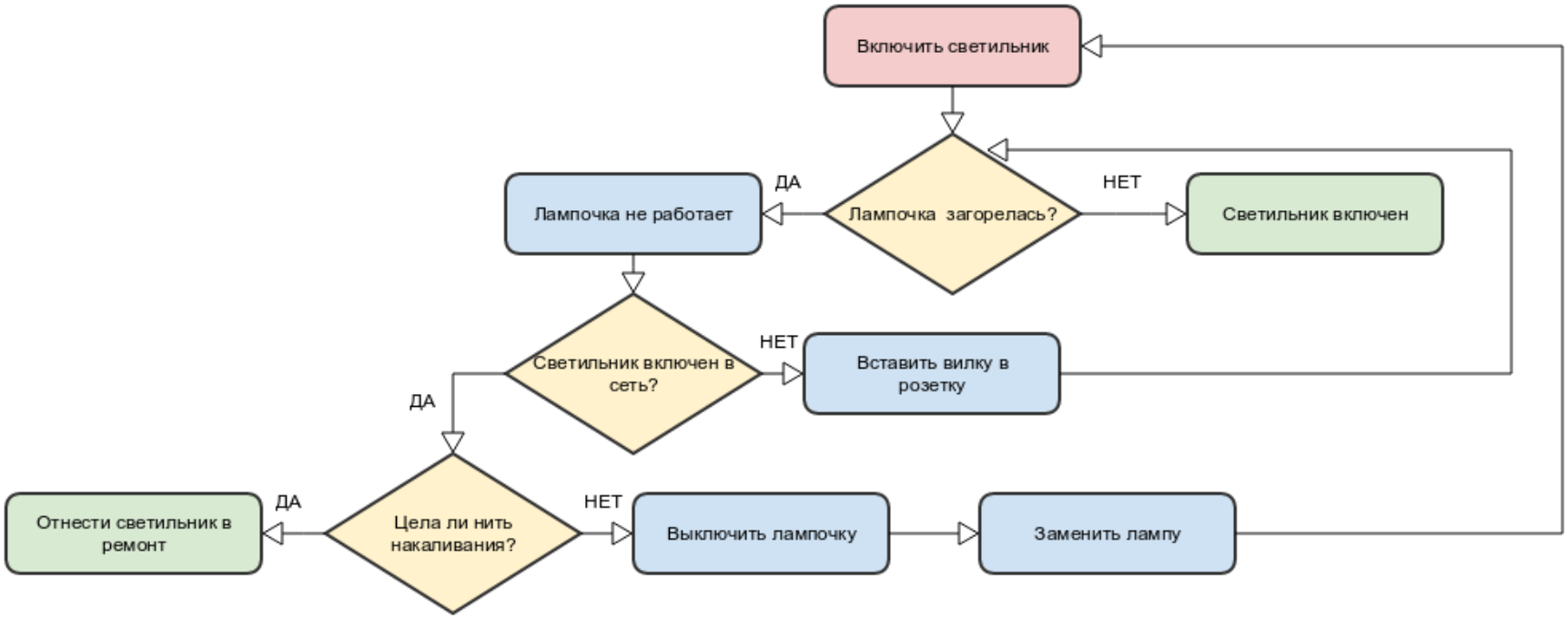
1. Контроль потока управления. Условный переход.
2. For Loop
3. Логические операции
4. Вложенные конструкции и область видимости.
5. Преобразование типов - явное и неявное.
6. Строки: создание и сложение строк.
7. Фундаментальные алгоритмы: знакомство.
8. Первый алгоритм: поиск минимума.

1. Контроль потока управления. Условный переход.
2. For Loop
3. Логические операции
4. Вложенные конструкции и область видимости.
5. Преобразование типов - явное и неявное.
6. Строки: создание и сложение строк.
7. Фундаментальные алгоритмы: знакомство.
8. Первый алгоритм: поиск минимума.

Контроль потока управления.

- Control flow
- Порядок, в котором выполняются инструкции, выражения, вызовы функций и так далее.
- Одна из отличительных черт процедурного программирования
- На диаграмме часто изображают граф, который называют графом потока управления.

Граф потока управления



Конструкции потока управления

- Условные конструкции (if-then, if-then-else, switch)
- Циклы (for-loop, while-loop, do-while-loop)
- Ветвления (break, continue, return)

Условный переход.

Условный переход

if-then, if-then-else, switch

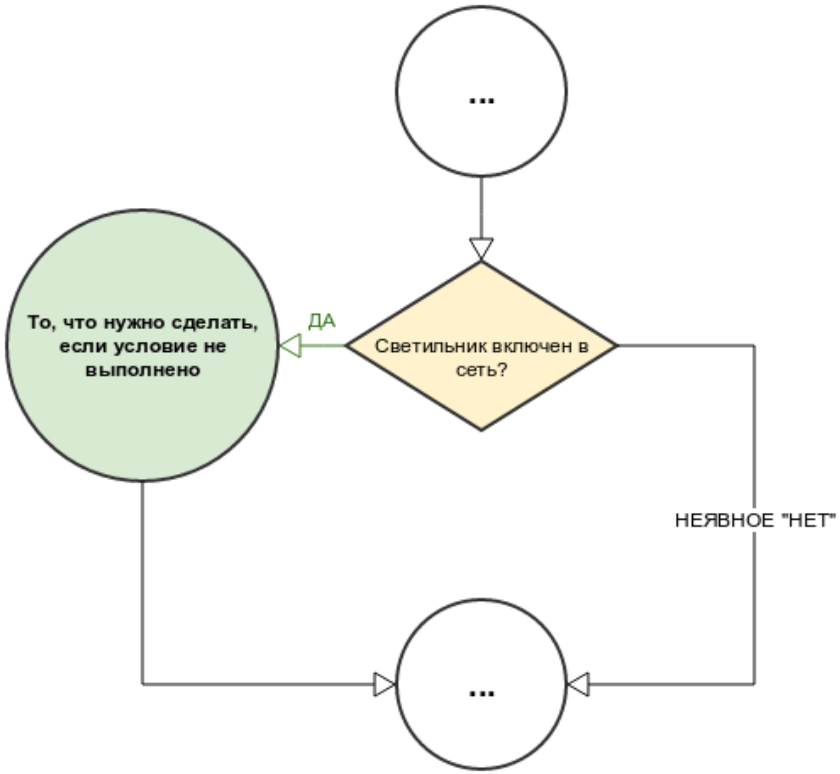
If-Then

Общая форма:

```
if (boolean expression) {
    ... // do something
}
```

Пример:

```
int a = 6;
int b = 0;
if (b != 0) {
    System.out.println(a / b);
}
```



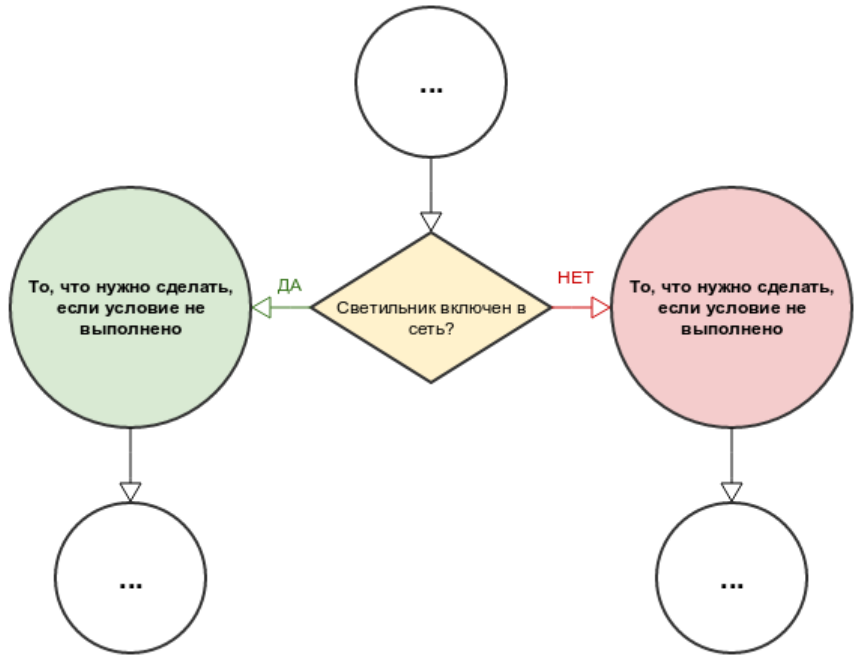
If-Then-Else

Общая форма:

```
if (boolean expression) {
} else {
}
```

Пример:

```
int a = 6;
int b = 0;
if (b != 0) {
    System.out.println(a / b);
} else {
    System.out.println("b=0");
}
```

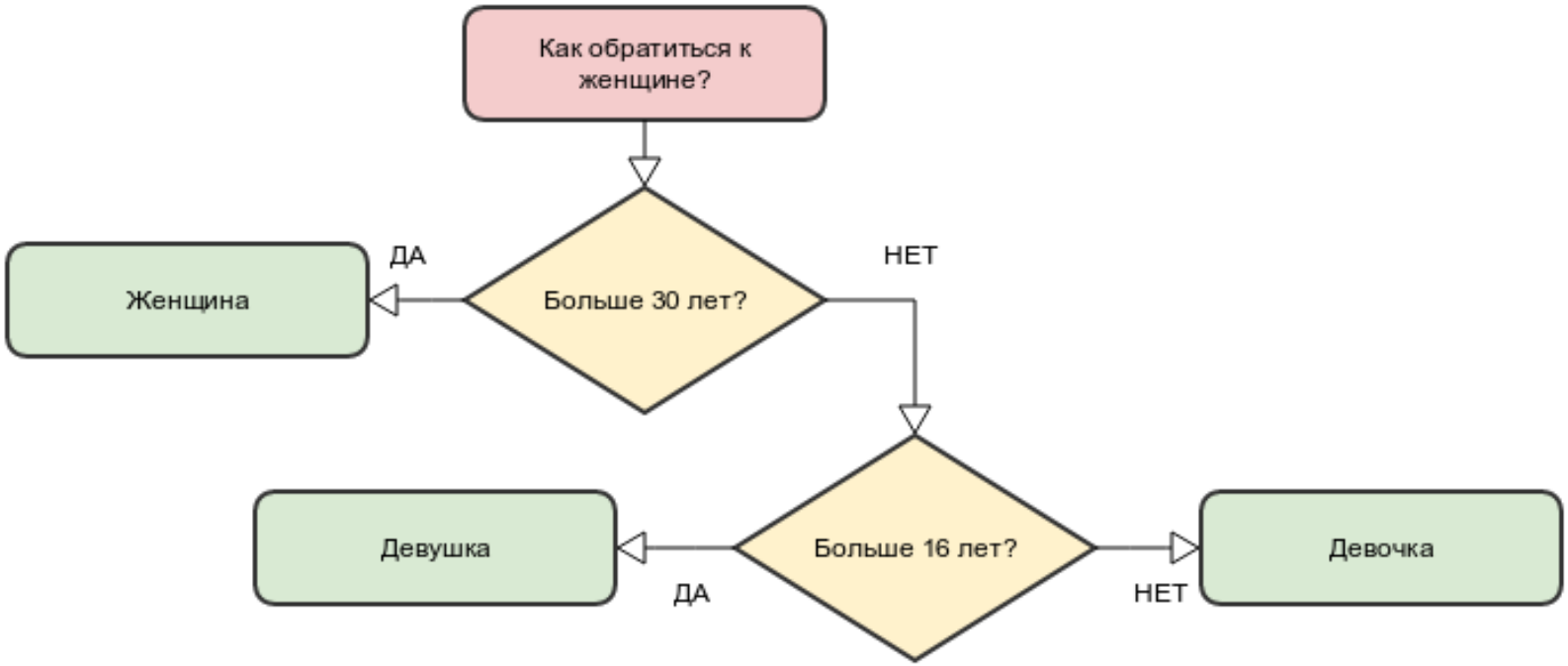


If-Then-Else-If-Then-Else

Пример:

```
int a = 6;
int b = 0;
if (a == 0) {
    System.out.println("no roots");
} else if (b != 0) {
    System.out.println("x = " + a / b);
} else {
    System.out.println("x = all numbers");
}
```

If-Then-Else-If-Then-Else



Сокращенная запись

- Фигурные скобочки можно опускать, если у вас код в ветвлении - всего одно предложение (т.е. с одной ;).
- Огромное количество вложенных конструкций - не приветствуется. То есть если у вас три уровня вложенности, то это повод задуматься.

Стандартные проблемы

- Огромное количество вложенных конструкций - не приветствуется. То есть если у вас три уровня вложенности, то это повод задуматься.
- Не стоит писать такой код:

```
boolean b = ...;

if(b == true) {
    ...
}
```

Вместо этого мы используем **if(b)**, **if(!b)** и прочие конструкции

Стандартные проблемы

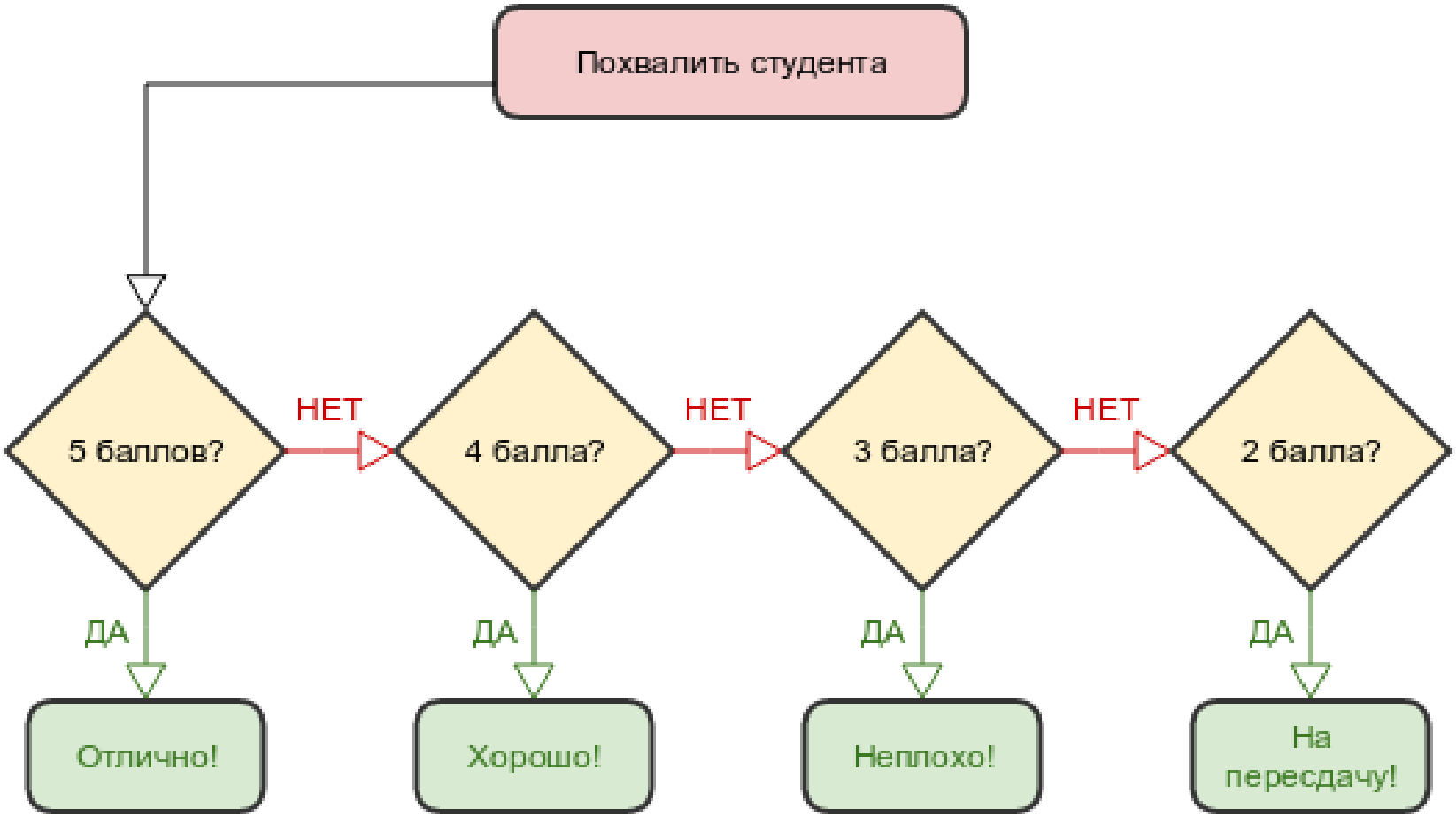
- А такой код просто выглядит некрасиво:

```
if (a == 0)
    System.out.println("YES");
else {
    int b = 19 % a;
    int c = 19 / a;
    System.out.println(b + c);
}
```

Switch

- Принято считать, что switch выбирает один из предложенных вариантов
- На самом деле это не всегда так (и это может послужить источником ошибок!)
- Работает с
  - **byte**, **short**, **char**, **int**
  - **String**
  - **Byte**, **Short**, **Character**, **Integer**
  - **Enum** types

Switch


Похвалить студента
5 баллов?
НЕТ
4 балла?
НЕТ
3 балла?
НЕТ
2 балла?
ДА
Отлично!
ДА
Хорошо!
ДА
Неплохо!
ДА
На пересдачу!

Switch

- ключевое слово **switch**
- переменная, значению которой ищем соответствие
- тело в скобочка { и }:
  - ключевое слово **case**, значение и **:**
  - после **двоеточия** - действия для выполнения. Фигурные скобочки вокруг них возможны, но не обязательны (и не приняты).
  - возможно ключевое слово **break**, после которого нет действий и которое "возвращает"нас за пределы **switch**
  - возможен case 'по умолчанию', который обхозначается ключевым словом **default**

Switch - простой

```
int a = some_number % 3;
switch (a) {
    case 0:
        System.out.println("Mod = 0");
        break;
    case 1:
        System.out.println("Mod = 1");
        break;
    case 2:
        System.out.println("Mod = 2");
        break;
}
```

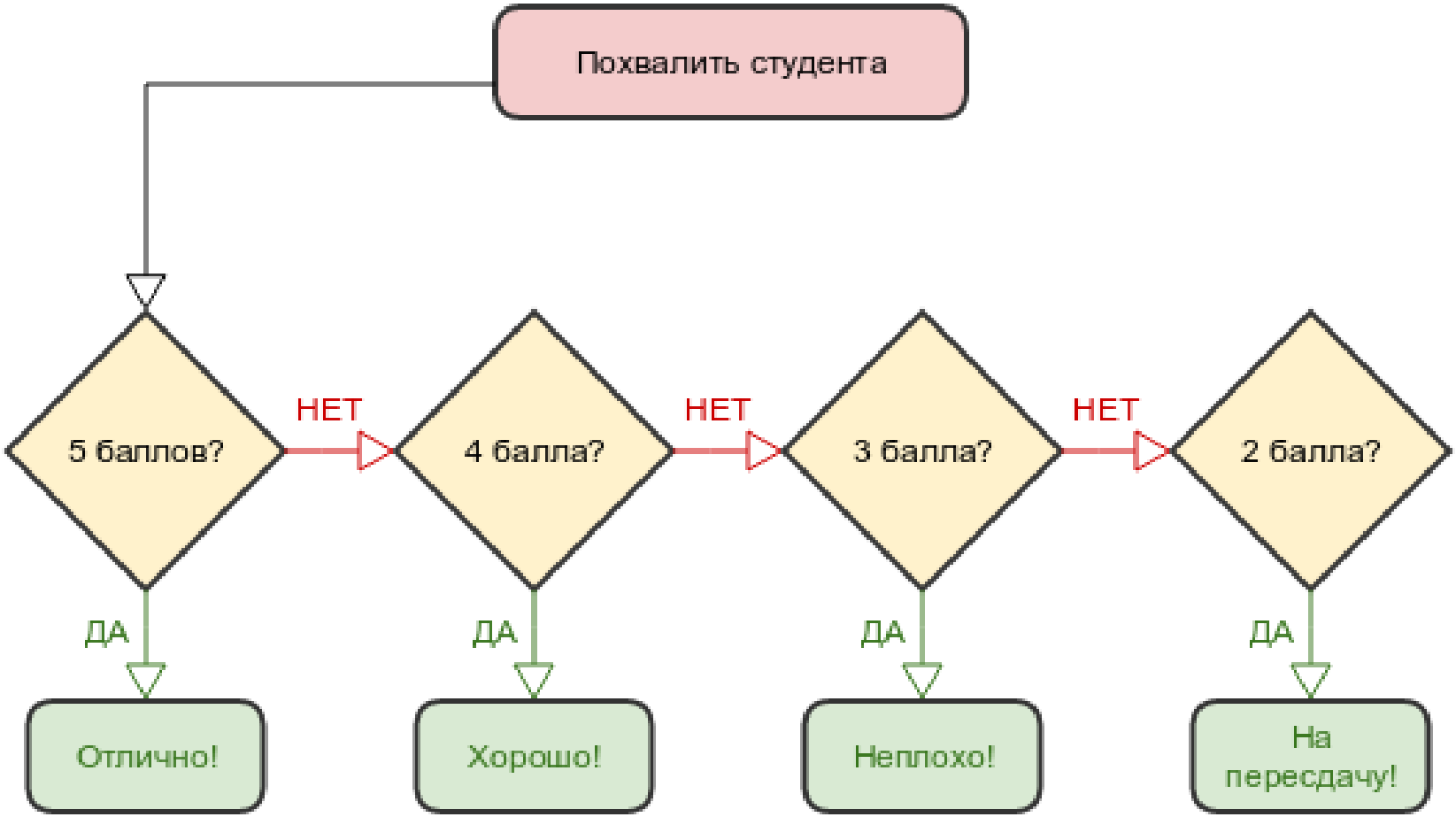
Switch - с default действием

```
int a = some_number % 100;
switch (a) {
    case 0:
        System.out.println("Mod = 0");
        break;
    case 1:
        System.out.println("Mod = 1");
        break;
    dafault: // 'default case'
        System.out.println("Mod >= 2");
        break;
}
```

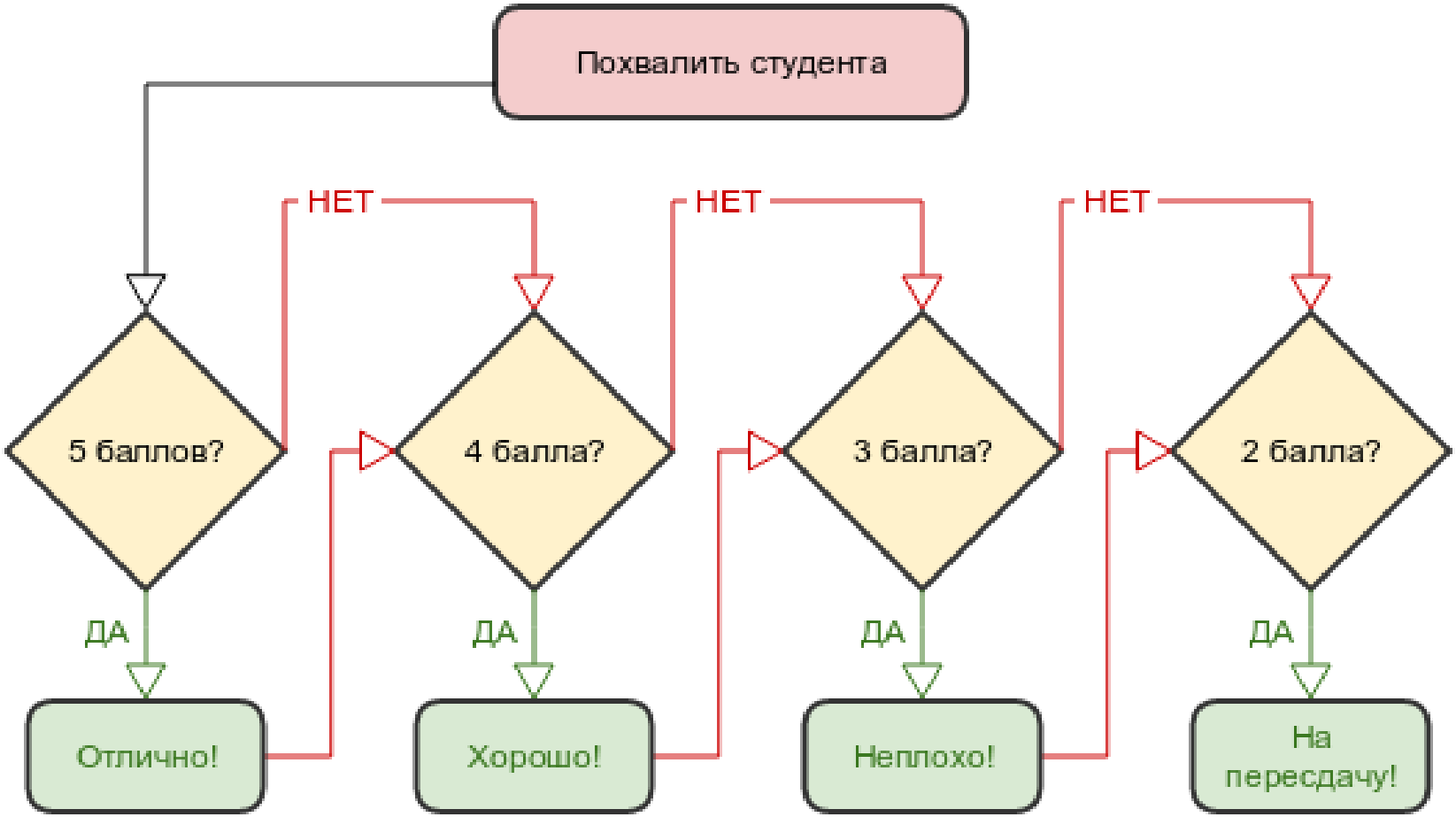
Switch - break

- Но самые интересные вещи начинаются, когда мы забываем или намеренно не ставим слово **break** (или **return**).
- Часто служит источником неприятных ошибок
- Это связано с тем, что **switch** - это на самом деле не "выбор"

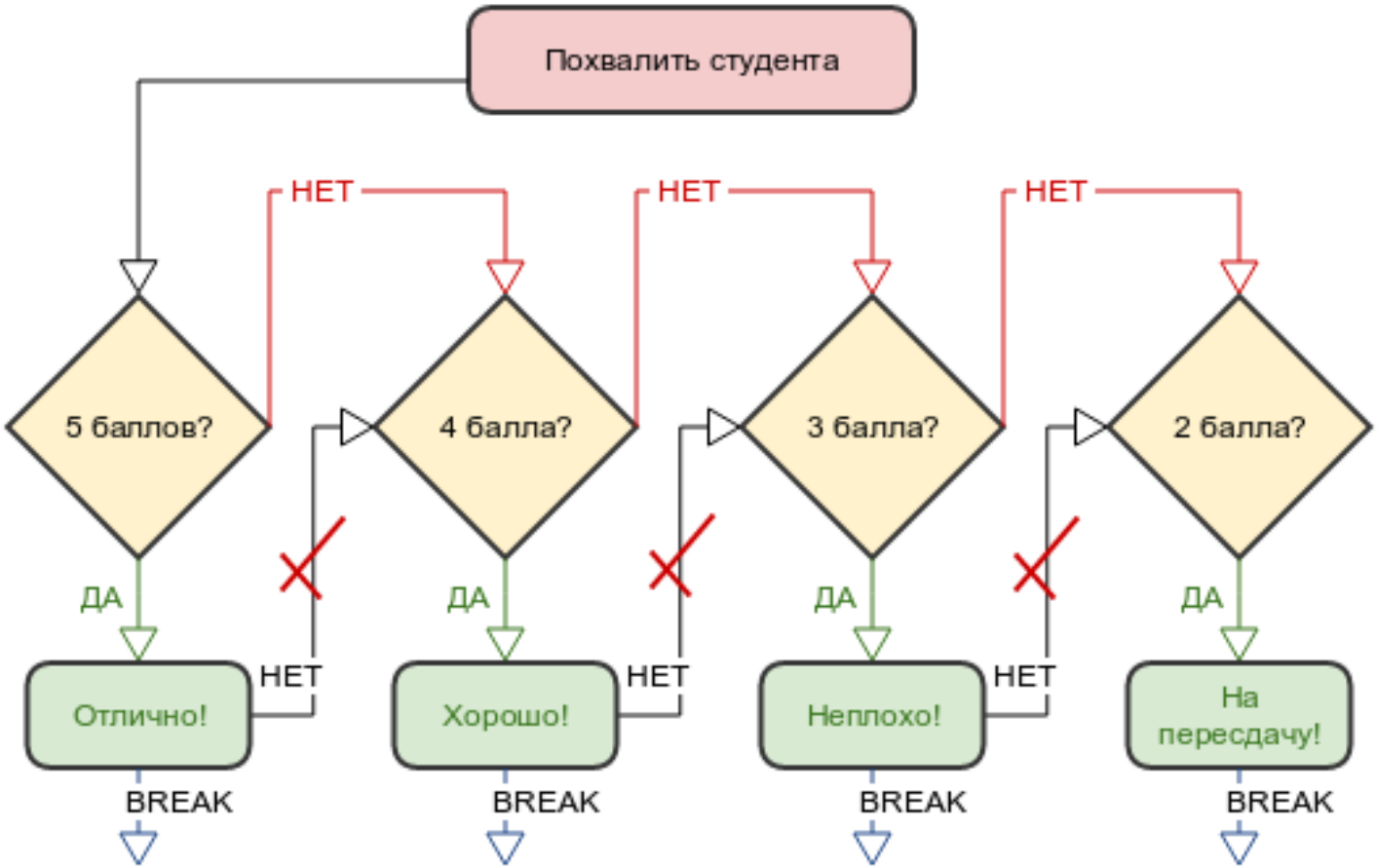
Switch - наивное понимание



Switch без break



Switch без break



Switch - стандартная ситуация (намеренная)

```
int a = some_number % 100;
switch (a) {
     case 0:
     case 1:
          System.out.println("Mod < 2");
          break;
     dafault: // 'default case'
          System.out.println("Mod >= 2");
          break;
}
```

То есть в случае, если $a = 0$ или $a = 1$, выведет **Mod < 2**.

Switch - вероятно "потерянный"break

```
int a = some_number % 100;
switch (a) {
     case 0:
          System.out.println("Mod = 0");
     case 1:
          System.out.println("Mod = 1");
          break;
     dafault: // 'default case'
          System.out.println("Mod >= 2");
          break;
}
```

То есть в случае, если $a = 0$, выведет и **Mod = 0**, и **Mod = 1**.

1 Контроль потока управления. Условный переход.

2 For Loop

3 Логические операции

4 Вложенные конструкции и область видимости.

5 Преобразование типов - явное и неявное.

6 Строки: создание и сложение строк.

7 Фундаментальные алгоритмы: знакомство.

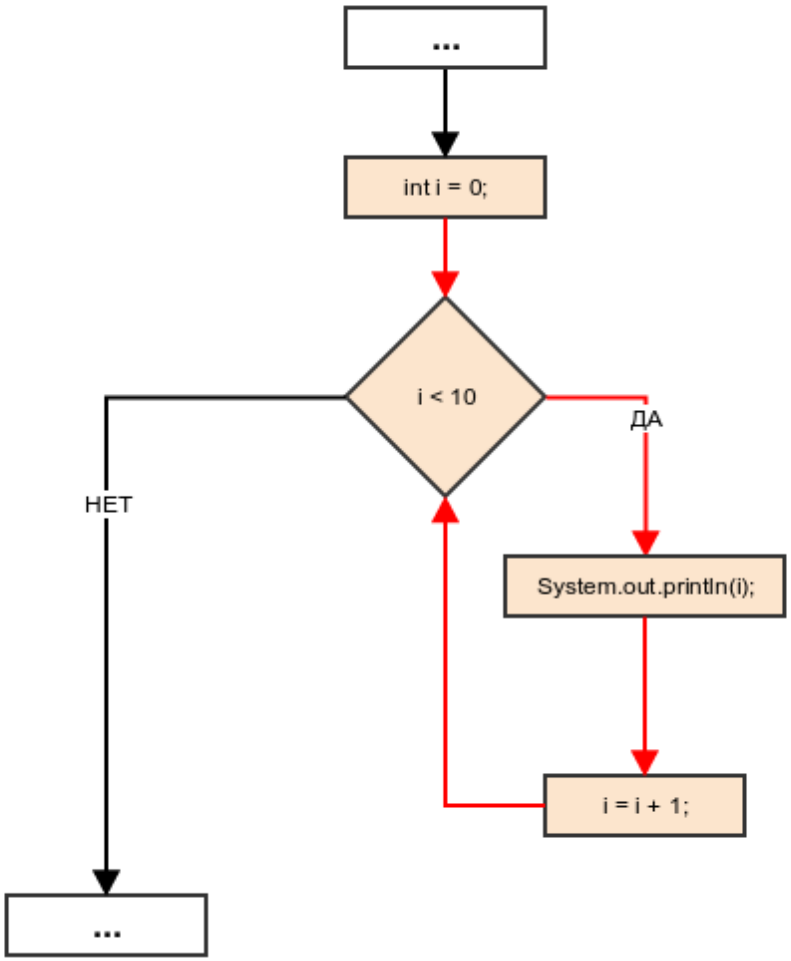
8 Первый алгоритм: поиск минимума.

Простейший цикл

- Зачем нужен цикл? Например, нам надо перебрать учеников класса...
- Циклы в java - for-loop, do-while, while
- Выглядит это так:

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

Простейший цикл




```
...

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

...
```

Простейший цикл - результат

```
0
1
2
3
4
5
6
7
8
9
```

Цикл в цикле

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        ...
    }
}
```

Пример - таблица умножения

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        table[i][j] = (i + 1) * (j + 1);
    }
}
```

1 Контроль потока управления. Условный переход.

2 For Loop

3 Логические операции

4 Вложенные конструкции и область видимости.

5 Преобразование типов - явное и неявное.

6 Строки: создание и сложение строк.

7 Фундаментальные алгоритмы: знакомство.

8 Первый алгоритм: поиск минимума.

Условные операторы

- && - логическое **И**
- || - логическое **ИЛИ**
- ! - логическое **НЕ**
- ?: - **тернарный оператор**

Тернарный оператор

- If-Then-Else в краткой форме
- Не стоит использовать для больших конструкций
- Вложенные тернарные операторы - зло

```
my_variable = boolean_expression ? if_is : else_is;
```

Тернарный оператор

```
int x = 18;
String greeting = x > 30 ? "Woman" : "Girl";
System.out.println("Hello, " + greeting);
```

Вложенный тернарный оператор

```
int x = 18;
String greeting = x > 30 ? (x > 60 ? "Grandma" : "Woman") :
    "Girl";
System.out.println("Hello, " + greeting);
```

Логическое И

```
boolean expr = boolean_expr_1 && boolean_expr_2;
```

- true && true = true
- false && true = false
- true && false = false
- false && false = false

Логическое ИЛИ

```
boolean expr = boolean_expr_1 || boolean_expr_2;
```

- true || true = true
- false || true = true
- true || false = true
- false || false = false

Логические И и ИЛИ - порядок вычисления

```
boolean expr1 = boolean_expr_1 || boolean_expr_2;
boolean expr2 = boolean_expr_1 && boolean_expr_2;
```

- Первым вычисляется левое выражение. Если по нему уже можно определить результат, то следующее выражение вычисляться не будет.
- Логическое И - если первое выражение false , то результат false и второе вычисляться не будет.
- Логическое ИЛИ - если первое выражение true , то результат true и второе вычисляться не будет.

Логические И и ИЛИ - порядок вычисления

Следующий код не сгенерирует **NullPointerException**, несмотря на вызов **equals**.

```
a = null;
if (a == null || a.equals(b)) {
    ... // do something
}

if (a != null && a.equals(b)) {
    ... // do something else
}
```

Логическое НЕ

```
boolean expr = !boolean_expr_1;
```

- !true = false
- !false = true

Логическое НЕ

Следующий код не сгенерирует **NullPointerException**, несмотря на вызов **equals**.

```
a = false;
if (!a) {
    ... // do something
}
```

Комбинации условий

```
boolean a = false;
boolean b = true;
boolean c = true;
if (!(!a && b || c)) {
    ... // do something
}
```

Результат вычисляется на основе приоритетов операций.

Условные операторы: приоритеты

- оператор НЕ (!)
- оператор логического И (&&)
- оператор логического ИЛИ (||)
- тернарный оператор (?:)

На самом деле порядок определен над всеми операциями, а не только логическими, и его надо знать

Условные операторы: приоритеты

Шпаргалка

$! \rightarrow \&\& \rightarrow || \rightarrow ?:$

- $!true \,||\, true = ?$
- $(!true\ ) \,||\, true = ?$
- $!(true \,||\, true\ ) = ?$
- $true \,||\, false \,\&\&\, false = ?$
- $true \,||\, (false \,\&\&\, false\ ) = ?$
- $(true \,||\, false\ ) \,\&\&\, false = ?$
- $false \,\&\&\, false\ ?\ false : true = ?$
- $false \,\&\&\, (false\ ?\ false : true\ ) = ?$

Условные операторы: приоритеты

Шпаргалка

$! \rightarrow \&\& \rightarrow || \rightarrow ?:$

- $!true \,||\, true = true$
- $(!true\ ) \,||\, true = true$
- $!(true \,||\, true\ ) = false$
- $true \,||\, false \,\&\&\, false = true$
- $true \,||\, (false \,\&\&\, false\ ) = true$
- $(true \,||\, false\ ) \,\&\&\, false = false$
- $false \,\&\&\, false\ ?\ false : true = true$
- $false \,\&\&\, (false\ ?\ false : true\ ) = false$

1 Контроль потока управления. Условный переход.

2 For Loop

3 Логические операции

4 Вложенные конструкции и область видимости.

5 Преобразование типов - явное и неявное.

6 Строки: создание и сложение строк.

7 Фундаментальные алгоритмы: знакомство.

8 Первый алгоритм: поиск минимума.

Variable scope

```
int i = 0;

if (i == 0) {
    long t = 15;
    System.out.println(i + ", " + t);
}

System.out.println(i + ", " + t);
```

Variable scope

```
int i = 0;

if (i == 0) {
    long t = 15;
    System.out.println(i + ", " + t);
}

System.out.println(i);
```

Области видимости (процедурные)

- метод
- блок операторов:
  - цикл
  - условный оператор if/then/else
  - switch
  - отделенный скобками блок кода
  - и другие

Области видимости

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: параметры метода

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменные внутри метода

```
public static void myMethod(int k, boolean bl) {
    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";
    for (int i = 0; i < a.length; i++) {
        long b = 90;
        for (int j = 0; j < a[i].length; j++) {
            long c = b % 16;
            if(c % 7 > 1){
                String p = "123" + s + b;
            }
            {
                char chr = 14;
            }
        }
    }
}
```

Области видимости: переменная цикла

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменная внутри цикла

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменная цикла

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменная внутри цикла

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменная внутри условия

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: переменная внутри блока

```
public static void myMethod(int k, boolean bl) {

    int[][] a = {{0, 1}, {2, 3}};
    String s = "my string";

    for (int i = 0; i < a.length; i++) {

        long b = 90;

        for (int j = 0; j < a[i].length; j++) {

            long c = b % 16;

            if(c % 7 > 1){
                String p = "123" + s + b;
            }

            {
                char chr = 14;
            }

        }

    }

}
```

Области видимости: if-then-else

Как вы думаете, доступна ли переменная **p** внутри блока **else**?

```
int i = 10;

if (i > 3) {
    String p = "A";
} else {
    int a = 0;
}
```

Области видимости: if-then-else

```
int i = 10;

if (i > 3) {
    String p = "A";
} else {
    int a = 0;
}
```

Области видимости: if-then-else

Как вы думаете, можно ли использовать переменную **p** за пределами блока?

```
int i = 10;

if (i > 3) {
    String p = "A";
} else {
    String p = "B";
}

    System.out.println(p);
```

Области видимости: switch

Как вы думаете, можно ли использовать переменную **b** в блоке **case: 4**?

```
int a = new Random().nextInt() % 5;

switch (a) {
    case 0:
        int b = 10;
    case 1:
    case 2:
    case 3:
    case 4:
        System.out.println(b);
}
```

Области видимости: switch

Как вы думаете, можно ли использовать переменную **b** в блоке **case: 4**?

```
int a = new Random().nextInt() % 5;

switch (a) {
    case 0:
        int b = 10;
        break;
    case 1:
    case 2:
    case 3:
    case 4:
        int b = 8;
        System.out.println(b);
}
```

Области видимости: switch

Переменная **b** 'видна' вплоть до конца блока **switch**, даже если после нее указан **break** или **return**. Однако, она может оказаться неинициализированной, потому не может быть использована в блоке **case 4**. Из-за того, что она там 'видна', объявить ее повторно нельзя.

Области видимости: switch

It works!

```
int a = new Random().nextInt() % 5;

switch (a) {
    case 0:
        int b = 10;
        break;
    case 1:
    case 2:
    case 3:
    case 4:
        b = 8;
        System.out.println(b);
}
```

Никогда не пишите такой код!

1 Контроль потока управления. Условный переход.

2 For Loop

3 Логические операции

4 Вложенные конструкции и область видимости.

5 Преобразование типов - явное и неявное.

6 Строки: создание и сложение строк.

7 Фундаментальные алгоритмы: знакомство.

8 Первый алгоритм: поиск минимума.

Приведение типов

- int → long, long → int
- Integer → int, int → Integer
- Object → String, String → Object

Приведение типов

Возможные операции (при выполнении определенных условий)

	Примитивы	**Объекты**
Примитивы	Расширение/Сужение	Autoboxing
Объекты	Unboxing	Down-casting / Up-casting

Приведение типов - зачем?

- Например, у вас есть код, написанный вашим коллегой из соседнего отдела. Этот код работает с типом **Integer**, например

```
public Integer calulateDelta(Integer value) {
    ...
}
```

- В свою очередь в вашем коде все операции производятся с типом **int**:

```
public void runSomething(){
    ...
    int t = someMethod();
    int delta = calculateDelta( ? );
    ...
}
```

Приведение типов - зачем?

- Например, у вас есть хранилище свойств файловой системы. Это могут быть даты, права доступа, свободное место и так далее. Вам необходим метод, который возвращает свойство по имени:

```
public Object getProperty(String name) {
    ...
}
```

- Если бы вы знали, что метод возвращает только стого дату, но код выглядел бы как

```
public Date getProperty(String name) {
    ...
}
```

Приведение типов - зачем?

- Первая форма преобразования здесь - независимо от того, что за объект мы возвращаем, мы видим его как **Object**. То есть мы приводим все к **Object** - и свободное место (**long**), и дату последней модификации (**Date**).
- Вторая форма преобразования:

```
Object property = getProperty('timestamp');
Date lastModified = (Date) property;
```

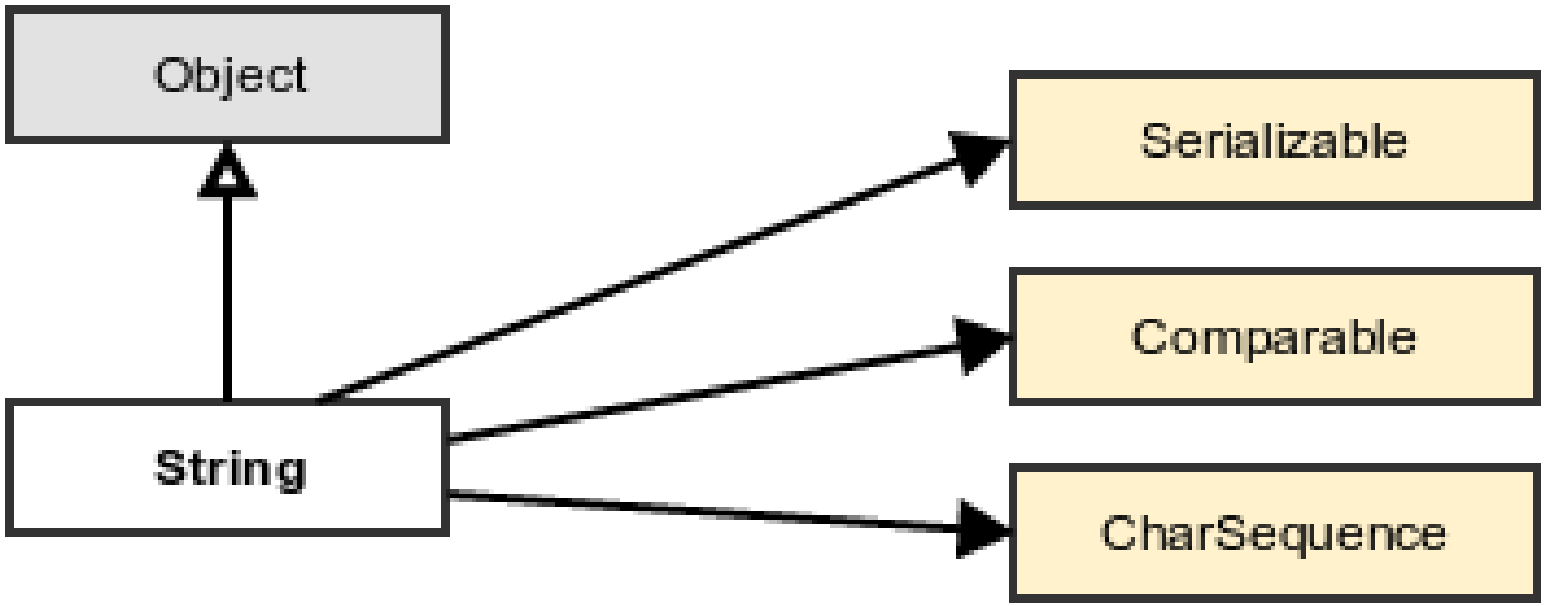
Примитивы и обертки

- int и Integer
- long и Long
- float и Float
- double и Double
- short и Short
- byte и Byte
- char и Character
- boolean и Boolean

Autoboxing & Unboxing

```
int k = 10;
Integer f = k; // autoboxing: int -> Integer
int p = f; // unboxing: Integer -> int
```

Down-casting и Up-casting



Down-casting и Up-casting

```
Object o = "sdf";
String s = (String) o; // down-casting

String a = "abc";
Object b = a; // up-casting
```

В случае, если приведение совершить нельзя, down-casting падает с ошибкой **ClassCastException**

Расширение и сужение

Расширение и сужение

Примитивы можно так же приводить друг к другу.

Расширение

byte $\rightarrow$ short $\rightarrow$ int $\rightarrow$ long $\rightarrow$ float $\rightarrow$ double

Сужение

double $\rightarrow$ float $\rightarrow$ long $\rightarrow$ int $\rightarrow$ short $\rightarrow$ byte

Расширение и сужение

Сужение требует указания типа (опасная операция):

```
long l = 9L;
int i = (int) l;
```

Расширение производится без прямых указаний (безопасная операция):

```
int i = 9;
long l = i;
```

1 Контроль потока управления. Условный переход.

2 For Loop

3 Логические операции

4 Вложенные конструкции и область видимости.

5 Преобразование типов - явное и неявное.

6 Строки: создание и сложение строк.

7 Фундаментальные алгоритмы: знакомство.

8 Первый алгоритм: поиск минимума.

Строки

- Тип **String**
- Набор символов
- Неизменяемый объект - все, что вы делаете со строкой порождает другую строку, а не изменяет старую.
- Не является примитивом
- Является **объектом**

```
String s = "abc";
```

Строки: операции

- Сложение строк:

```
String s0 = "a";
String s1 = "b";

String s = s0 + s1; // "ab"
```

- Сложение строк с другими типами:

```
String s0 = "a" + 5; // "a5"
String s1 = "b" + null; // "bnull"
String s2 = "a" + 't'; // "at"
```

- Различные методы, определенные в классе **String**

Сравнение строк

```
String s0 = "abc";
String s1 = "def";

boolean sameObject = s0 == s1;
boolean isEquals = s0.equals(s1);
```

Нюансы создания строк

Java устроена так, что строки **s0**, **s1** и **s2** будут указывать на один и тот же объект (не будет повторного создания):

```
String s0 = "abc";
String s1 = "abc";
String s2 = "ab" + "c";

System.out.println(s0 == s1); // true!
System.out.println(s0 == s2); // true!
```

Это исключительная особенность класса **String** в java language.
Старайтесь не пользоваться ей.

Нюансы создания строк

Однако в следующем примере **s0** и **s1** уже не указывают на один и тот же объект. Компилятор здесь не определит, что надо сразу s1 инициализировать в **"abc"**.

```
String s0 = "abc";
String s1 = "ab";
s1 = s1 + "c";

System.out.println(s0 == s1); // false!
```

Нюансы создания строк

То же самое будет в следующих случаях.

```
String s0 = "abc";
String s1 = "ab" + new String("c");
String s2 = new String("abc");

System.out.println(s0 == s1); // false!
System.out.println(s0 == s2); // false!
```

Сложение строк с другими типами

- Можно сложить с любым другим объектом
- При сложении объекты преобразуются в строки как будто бы с помощью метода **java.util.Objects.toString(...)**
- При этом для объектов используется стандартный метод объекта **toString()**
- На самом деле используются соответствующие методы из класса **java.lang.invoke.StringConcatenationFactory**.
- Стоит быть осмотрительными в случае с числами **double** и **float**: их стандартное представление вряд ли подходит для вывода на экран:

```
System.out.println("result: " + (2.0 / 3));
// result: 0.6666666666666666
```

Сложение строк: порядок операций

- Порядок операций в выражении определяется соответственно **приоритету** (Занятие 1).
- Этот код выполнится без проблем (операция / имеет приоритет выше, чем +):

```
System.out.println("result: " + 2.0 / 3);
```

- Этот код не скомпилируется (операция « имеет приоритет ниже, чем + и не определена для строк):

```
System.out.println("result: " + 2 << 1);
```

Сложение строк: порядок операций

- В случае, если приоритет операций одинаковый, то имеет значение порядок, в котором операции используются (слева направо):
- В случае, если у нас две операции сложения, то результат просто будет отличаться:

```
System.out.println("result: " + 2 + 3); // result: 23
System.out.println(2 + 3 + " - result"); // 5 - result
```

- В этом же случае приоритеты операций разные, но код просто не скомпилируется:

```
System.out.println("result: " + 2 - 3); // why?
```

- А этот код работает:

```
System.out.println(2 - 3 + " - result"); // -1 - result
```

1. Контроль потока управления. Условный переход.
2. For Loop
3. Логические операции
4. Вложенные конструкции и область видимости.
5. Преобразование типов - явное и неявное.
6. Строки: создание и сложение строк.
7. Фундаментальные алгоритмы: знакомство.
8. Первый алгоритм: поиск минимума.

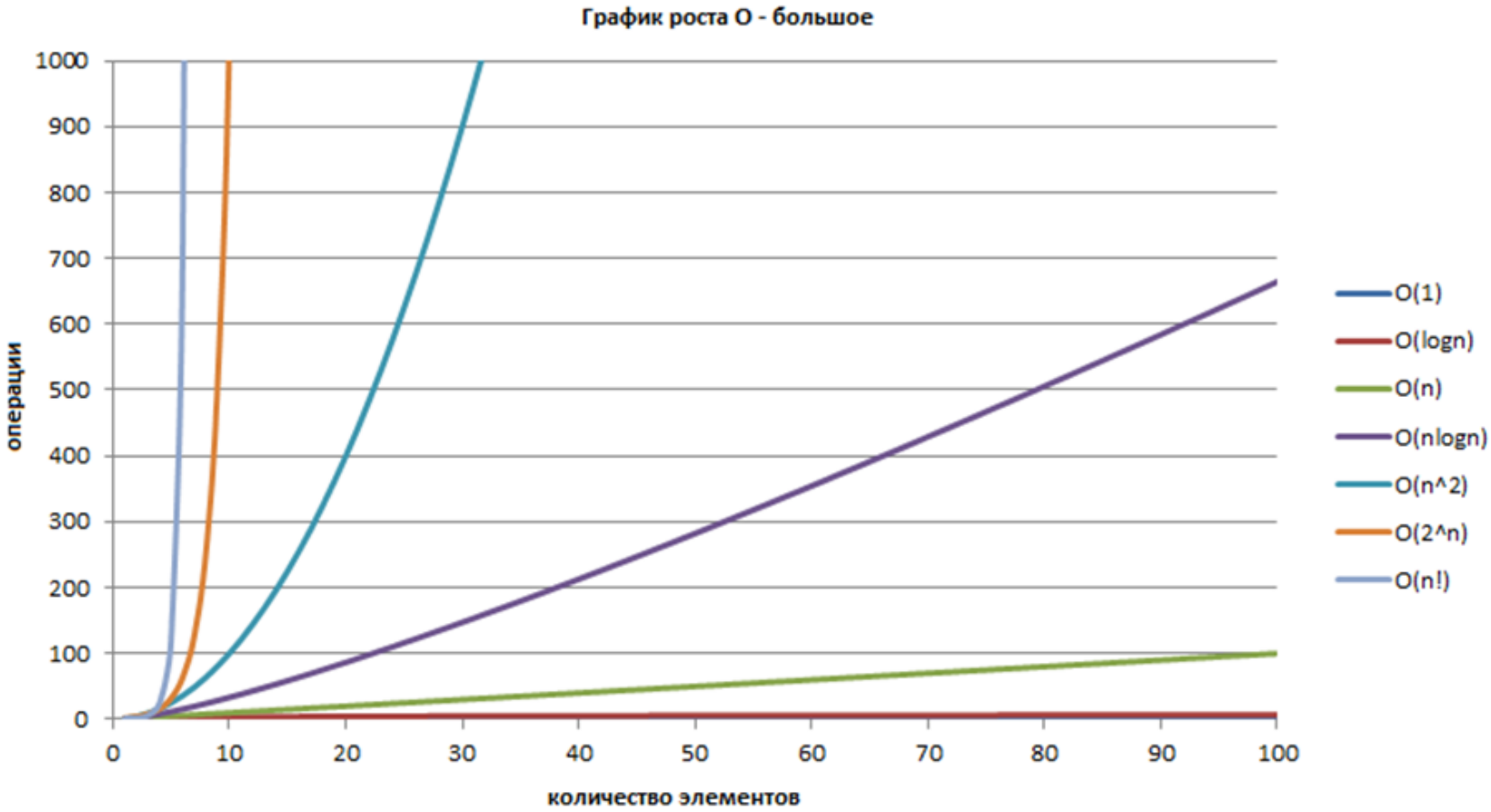
Фундаментальные алгоритмы

- Сортировка
- Поиск
- Алгоритмы хранения и доступа к данным
- Алгоритмы на графах
- Комбинаторные алгоритмы
- Шифрование, хэширование и многое другое

Оценка алгоритмов

- Скорость исполнения. Рост времени работы в зависимости от характера и размера входных данных.
- Требования к памяти
- Сложность описания на языке программирования
- + свойства, характерные для той или иной задачи

Асимптотическое поведение



Сортировка

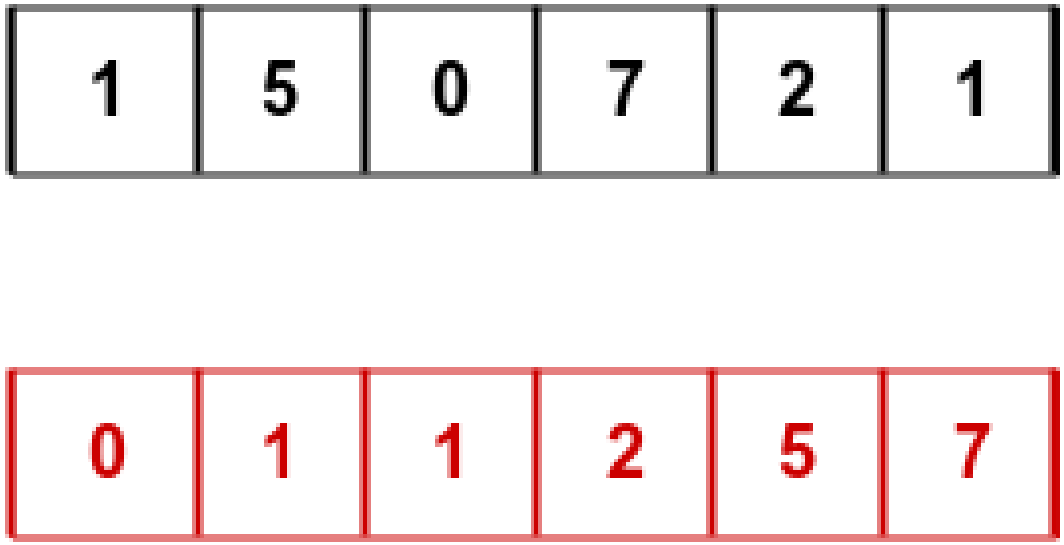
Неформальная постановка задачи

Есть набор входных данных, над которыми определен линейный порядок (элементы можно сравнивать).
Задача: упорядочить набор в порядке возрастания или убывания.

Примеры использования

- Отсортировать учеников по фамилиям
- Отсортировать учеников по успеваемости
- Отсортировать учеников по возрасту

Сортировка



Поиск элемента в наборе

Неформальная постановка задачи

Есть набор входных данных, необходимо найти какой-то конкретный элемент или же определить, что его нет в наборе.

Примеры

- Найти человека по номеру телефона
- Найти клиента, у которого сегодня день рождения
- Найти выплату за определенный месяц

Поиск элемента в наборе

1. Контроль потока управления. Условный переход.
2. For Loop
3. Логические операции
4. Вложенные конструкции и область видимости.
5. Преобразование типов - явное и неявное.
6. Строки: создание и сложение строк.
7. Фундаментальные алгоритмы: знакомство.
8. Первый алгоритм: поиск минимума.

Поиск минимума

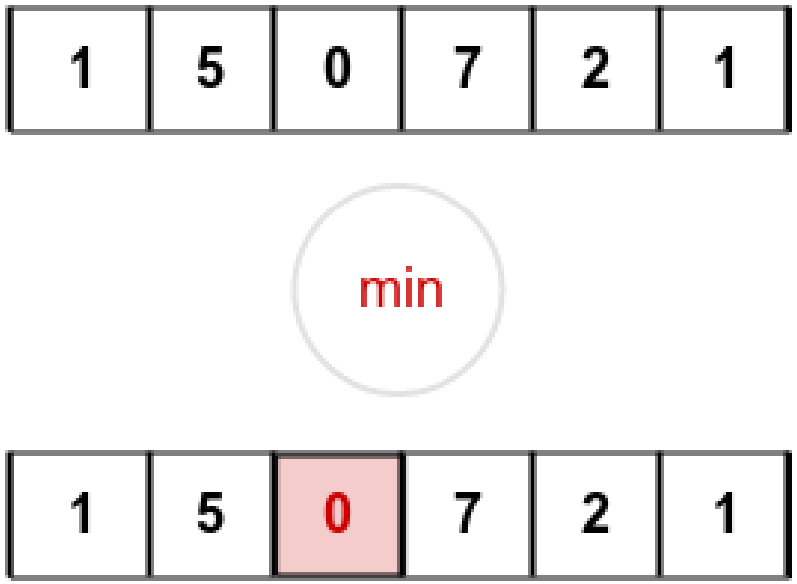
Неформальная постановка задачи

Есть набор входных данных, для которых задан линейный порядок (элементы можно сравнивать).
Задача: найти минимальный элемент

Примеры

- Найти самого младшего ученика
- Найти ученика с наихудшей успеваемостью
- Найти клиента с самой большой задолженностью

Поиск минимума



Алгоритм поиска минимума

Условие

Найти минимум в массиве **A**

Algorithm 1 Поиск минимума

```
1: function FindMinimum(A)
2:     min := A[0]
3:     for i = 1 to length(A) do
4:         if A[i] < min then
5:             min = A[i]
6:         end if
7:     end for
8:     return min
9: end function
```

Алгоритм поиска минимума

```
int[] A = new int[]{5, 8, 1, 14};

int min = A[0];
for (int i = 1; i < A.length; i++) {
    if (A[i] < min) {
        min = A[i];
    }
}

System.out.println(min);
```

Алгоритм поиска минимума

```
String[] A = new String[]{"B", "BC", "A", "a"};

String min = A[0];
for (int i = 1; i < A.length; i++) {
    if (A[i].compareTo(min) < 0) {
        min = A[i];
    }
}

System.out.println(min);
```

Поиск максимума

Неформальная постановка задачи

Есть набор входных данных, для которых задан линейный порядок:

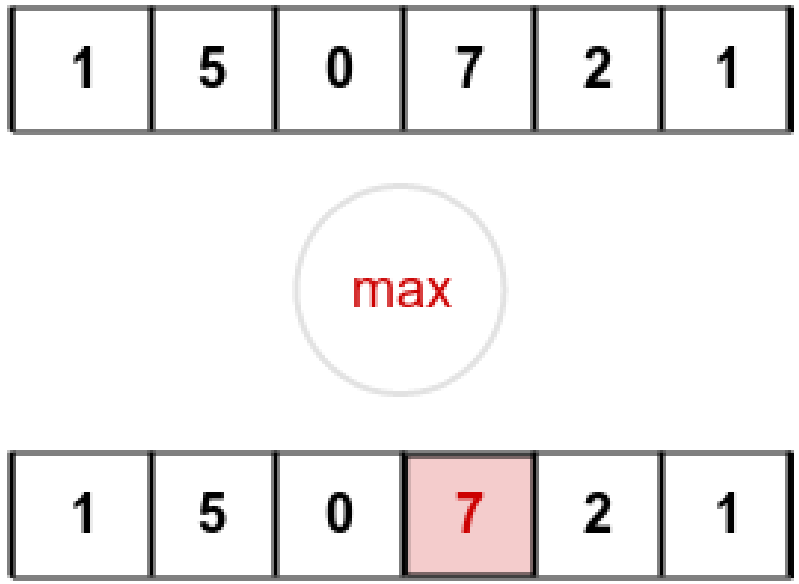
- для всех элементов x и y выполняется одно из условий: $x < y$, $x > y$ или $x = y$
- если $x < y$ и $y < z$, то $x < z$

Задача: найти максимальный элемент

Примеры

- Найти самого старшего ученика
- Найти ученика с лучшей успеваемостью
- Найти месяц с наибольшей суммарной выручкой

Поиск максимума



Алгоритм поиска максимума

Самостоятельная работа

Алгоритм поиска максимума

Условие

Найти максимум в массиве **A**

Algorithm 2 Поиск минимума

```
1: function FindMaximum(A)
2:     max := A[0]
3:     for i = 1 to length(A) do
4:         if A[i] > max then
5:             max = A[i]
6:         end if
7:     end for
8:     return max
9: end function
```

Алгоритм поиска минимума

```
int[] A = new int[]{5, 8, 1, 14};

int max = A[0];
for (int i = 1; i < A.length; i++) {
    if (A[i] > max) {
        max = A[i];
    }
}

System.out.println(max);
```