Программирование на языке Java

Алгоритмы и программы

Знакомство с Знакомство с Java

Переменные

Программирование на языке Java

Алгоритмы и программы

Программирование

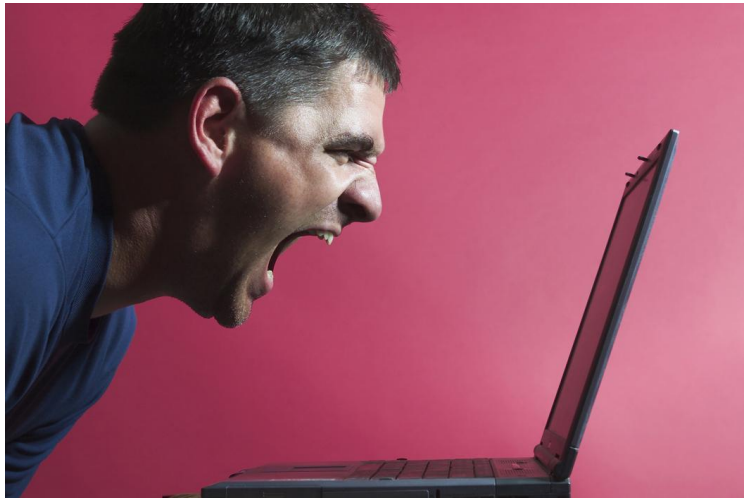
«Любой человек должен:
уметь сменить пеленку,
составить план вторжения,
заколоть свинью,
вести корабль,
построить дом,
написать сонет,
подвести счета,
построить стену,
снять мясо с костей,
утешить умирающего,
отдать приказ,
выполнить приказ,
действовать вместе и в одиночку,
решать уравнения,
анализировать новую проблему,
разбросать навоз,
запрограммировать компьютер,
приготовить вкусное блюдо,
биться и победить,
и умирать с достоинством.

Специализированны лишь насекомые.»

Р. Хайнлайн.

Достаточно времени для любви, или жизни Лазаруса Лонга, 1973

Программирование



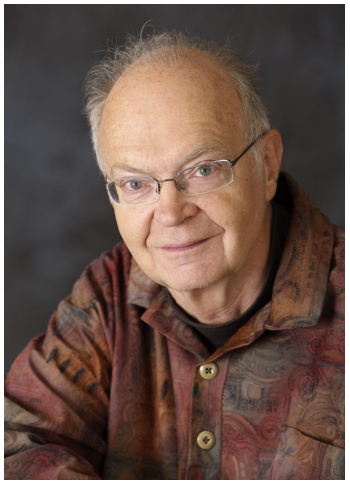
Скажи компьютеру что делать

Программирование:

- не только для экспертов;
- естественный, приносящий удовольствие творческий опыт;
- приносит новые достижения;
- путь в новый мир интеллектуальных соревнований.

Вызовы программирования

- Нужно узнать, на что способны компьютеры.
- Необходимо выучить язык программирования



Дональд Кнут

Вместо того, чтобы представлять себе, что наша главная задача – обучить компьютер тому, что делать, давайте сосредоточимся на объяснении людям того, что мы хотим, чтобы компьютер делал.

Алгоритм

Алгоритм – точный набор инструкций, описывающий порядок действий исполнителя для достижения результата решения задачи за конечное время.

Свойства алгоритма

- **дискретность**: состоит из отдельных шагов (команд)
- **понятность**: должен включать только команды, известные исполнителю (входящие в СКИ)
- **детерминированность (определенность)**: при одинаковых исходных данных всегда выдает один и тот же результат
- **конечность**: заканчивается за конечное число шагов
- **массовость**: может применяться многократно при различных исходных данных
- **корректность**: дает верное решение при любых допустимых исходных данных
- **результативность**: завершает работу определёнными результатами

Задача

Является ли формула алгоритмом для вычисления числа π

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right) = 4\sum_{i=0}^{\infty}\frac{(-1)^i}{2i+1}$$

Не соблюдаются свойства:

- **конечности**
- **массовости**

Программа

Программа – запись алгоритма на формальном языке (часто употребляется как синоним термина алгоритм, но это не совсем так).

Алгоритм – основная идея, метод, схема решения задачи.

Программа – конкретная реализация алгоритма, которая может быть скомпилирована и выполнена на компьютере.

Команды – 1

Команда – описание действий, которые должен выполнить исполнитель.

- откуда взять исходные данные?
- что нужно с ними сделать?

Исполнитель должен уметь выполнять все команды, составляющие алгоритм.

Множество возможных команд **конечно** и изначально **строго задано**.

Действия, выполняемые по этим командам, называются **элементарными**.

Команды – 2

У каждого исполнителя есть конечный набор элементарных команд (действий), оперирующих элементарными объектами, которых также конечное число.

Способы записи алгоритма. Словесный – 1

Словесная запись – описание последовательных этапов обработки данных в произвольном изложении на естественном языке.

Недостатки:

- отсутствие строгой формализации;
- многословность записи;
- допускают неоднозначность толкования отдельных предписаний.

Способы записи алгоритма. Словесный – 2

Пример.

1. задать два числа;
2. если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. определить большее из чисел;
4. заменить большее из чисел разностью большего и меньшего из чисел;
5. повторить алгоритм с шага 2.

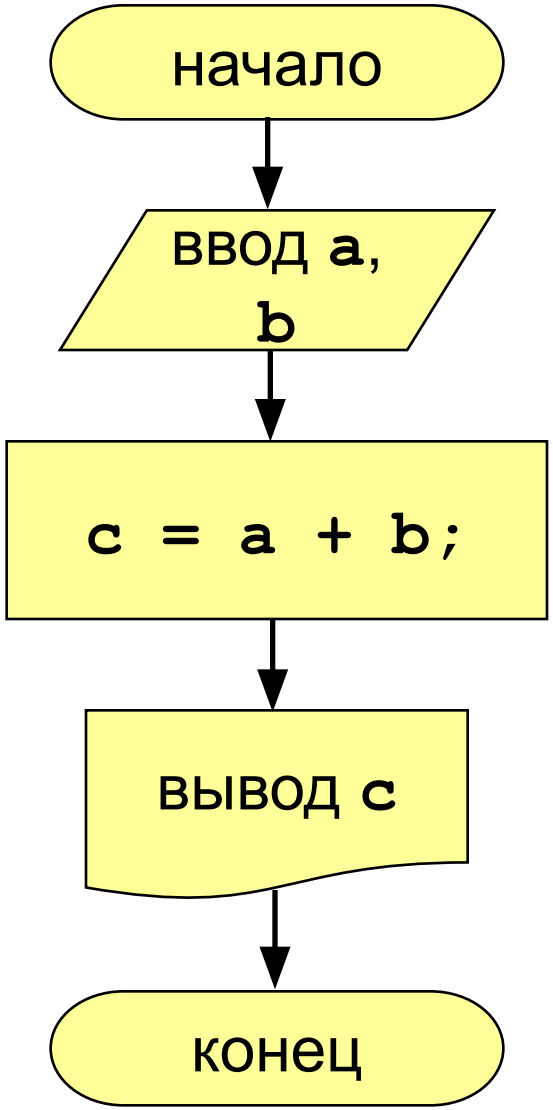
Способы записи алгоритма. Графический – 1

Графическое представление – алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

Такое графическое представление называется **блок-схемой**.

Способы записи алгоритма. Графический – 2

Пример.



Способы записи алгоритма. Псевдокод – 1

Псевдокод – полуформализованное описание алгоритмов на условном алгоритмическом языке, включающее в себя как элементы ЯП, так и фразы естественного языка, общепринятые математические обозначения и др.

Псевдокод – промежуточный язык между естественными и формальными языками.

Способы записи алгоритма. Псевдокод – 2

Пример.

```
IF вы счастливы THEN
    улыбайтесь
ELSEIF вам грустно
    хмурьтесь
ELSE
    сохраняйте обычное выражение
лица
ENDIF
```

Способы записи алгоритма. Программа

Программа – запись на языке программирования.

Пример программы на Java

```
public class First {
public static void main (String[] args)
   {
   System.out.print("Hello, World!");
   }
}
```

Основные качества программ

- **Корректность (правильность)** – реализация корректного алгоритма решения исходной задачи.
- **Эффективность** – уменьшение времени работы программы.
- **Понятность и модифицируемость**
- **Удобство эксплуатации**
- **Надежность**
- **Удобство сопровождения**

Писать понятные и легко модифицируемые программы существенно легче, чем правильные и эффективные.

Правила написания программного кода

1. Использовать осмысленные имена для всех переменных, отличных от счетчиков;
2. Константам, отличным от нуля и единицы присваивать имена;
3. Соблюдать принятый в языке стиль написания имен (имена классов с прописной буквы, переменных и методов – со строчной, констант – полностью из прописных);
4. Добавлять краткие и понятные комментарии.
5. Применять форматирование текста (лесенка – упорядочивание програм
повышения его читабель

Автоматическое форматирование:
NetBeans – Alt + Shift + F
IntelliJ IDEA – Ctrl + Alt + L

Этапы разработки программ – 1

1. Постановка задачи
2. Анализ и исследование задачи, модели
3. Разработка алгоритма
4. Программирование
5. Тестирование и отладка
6. Анализ результатов решения задачи
7. Сопровождение программы

Этапы разработки программ – 2

1. Постановка задачи:

- сбор информации о задаче;
- формулировка условия задачи;
- определение конечных целей решения задачи;
- определение формы выдачи результатов;
- описание данных (их типов, диапазонов величин, структуры и т.п.).

Этапы разработки программ – 3

2. Анализ и исследование задачи, модели:

- анализ существующих аналогов;
- анализ технических и программных средств;
- разработка математической модели;
- разработка структур данных.

Этапы разработки программ – 4

3. Разработка алгоритма:

- выбор метода проектирования алгоритма;
- выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
- выбор тестов и метода тестирования;
- проектирование алгоритма.

Этапы разработки программ – 5

4. Программирование:

- выбор языка программирования;
- уточнение способов организации данных;
- запись алгоритма на выбранном языке программирования.

Этапы разработки программ – 6

5. Тестирование и отладка:

- синтаксическая отладка;
- отладка семантики и логической структуры;
- тестовые расчеты и анализ результатов тестирования;
- совершенствование программы.

Этапы разработки программ – 7

6. Анализ результатов решения задачи:

- уточнение в случае необходимости математической модели с повторным выполнением этапов 2 — 5

7. Сопровождение программы:

- доработка программы для решения конкретных задач;
- составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

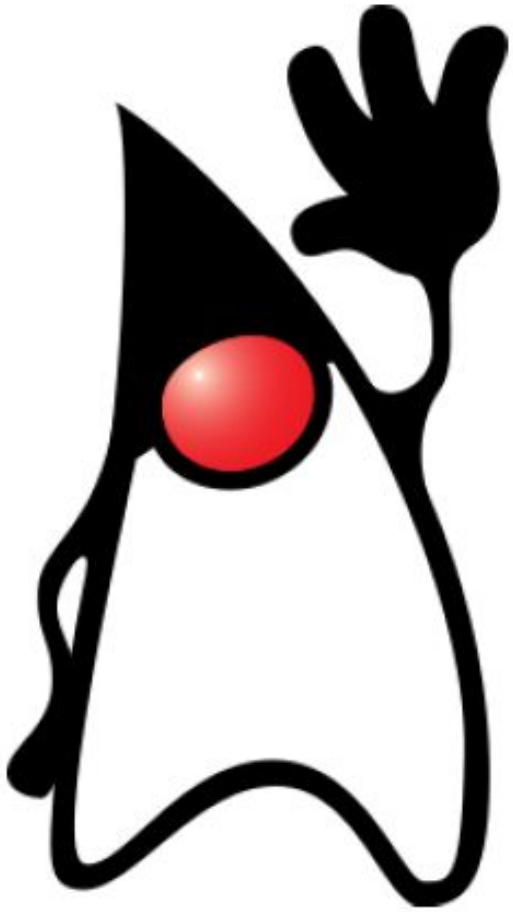
Программирование на языке Java

Знакомство с Java

Языки программирования

- **Машинно-ориентированные (низкого уровня)** – каждая команда соответствует одной команде процессора (ассемблер).
- **Языки высокого уровня** – приближены к естественному (английскому) языку, легче воспринимаются человеком, **не зависят от конкретного компьютера**
  - *для обучения*: Basic, ЛОГО, Pascal
  - *профессиональные*: Java, C, C++
  - *для задач искусственного интеллекта*: Prolog, LISP
  - *для Интернета*: JavaScript, Java, Perl, PHP, ASP.Net, Ruby


Java™



Язык Java

Java – объектно-ориентированный язык программирования, разработанный Sun Microsystems в 1995 г.

Почему стоит изучать Java?

Один из самых популярных и востребованных ЯП.
Индекс TIOBE (https://www.tiobe.com/tiobe-index/)
Рейтинг CFF языков программирования (https://m.habr.com/company/it-grad/blog/422057/)

На Java пишут:

- высоконагруженные системы;
- корпоративные приложения;
- настольные приложения;
- программы и игры для телефонов, в том числе под Android
- апплеты для смарт-карт

- Язык развивается и совершенствуется (версия Java 18 вышла в июле 2022 г.)

Почему стоит изучать Java?

Java — это не только язык программирования, но и. . .

- обширная стандартная библиотека;
- сторонние библиотеки и фреймворки;
- инструменты разработки (сборка, тестирование);
- методология ООП, паттерны проектирования;
- платформа для альтернативных языков; (Clojure, Groovy, JRuby, Jython, Kotlin, Scala).

Виртуальная машина и байт-код – 1

Традиционный подход:

исходный код → машинный код → процессор

- программа работает только на той платформе, под которую она скомпилирована

Подход Java:

исходный код → байт-код виртуальной машины → виртуальная машина → процессор

- программа работает на любой платформе, где есть виртуальная машина Java
- **“Write once, run anywhere!”** («Написано однажды, работает везде!»)

Виртуальная машина и байт-код – 2

Программы транслируются в байт-код, выполняемый виртуальной машиной Java (JVM = Java Virtual Machine).

Байт-код – машинно-независимый код низкого уровня, генерируемый транслятором и исполняемый виртуальной машиной.

Большинство инструкций байт-кода эквивалентны одной или нескольким командам ассемблера.

Виртуальная машина и байт-код – 3

Виртуальная машина Java (JVM) – основная часть исполняющей системы Java, интерпретирует и исполняет байт-код Java.

JVM доступны для многих аппаратных и программных платформ, что обеспечивает **кросс-платформенность** Java.

Виртуальная машина и байт-код – 4

Насколько быстро работает виртуальная машина?

- Интерпретация байткода на порядок (10–20 раз) медленнее исполнения аналогичного машинного кода. . .

- но есть Just-In-Time компиляция – виртуальная машина компилирует байткод в машинный код (используется с JDK 1.1)
- а также HotSpot адаптивный оптимизирующий JIT-компилятор (используется с JDK 1.3)
- в результате Java 8 всего в 1.5–2 раза медленнее C, а в некоторых тестах не хуже или даже быстрее!

Сборка мусора

Подход C/C++:

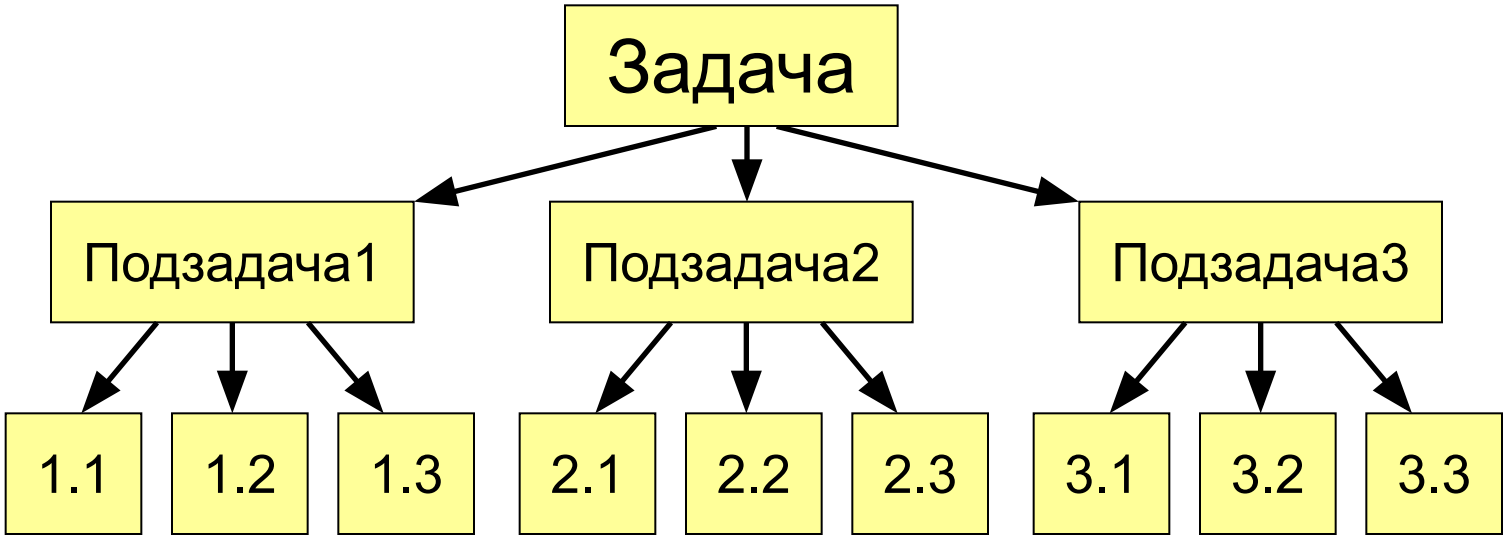
- выделил память → поработал → освободил память
- всё управление памятью в руках программиста

Подход Java:

- выделил память → поработал → молодец
- виртуальная машина считает ссылки на каждый объект
- освобождает память, когда ссылок больше нет

Разработка программ на Java

- разработка программ «сверху вниз»



- разнообразные структуры данных (массивы, коллекции: списки, отображения, множества)
- объектно-ориентированный подход


HELLO
WORLD

Из чего состоит программа?

```
public class <имя класса>
   {
   public static void main(String[]
args)
         {
      /* основная программа */
      }
   }
```

Комментарии, заключенные в «скобки» /* */ не обрабатываются

Простейшая программа

имя класса должно совпадать с именем файла

void = «пустой» основной метод не возвращает никакого результата

главный (основной) метод класса всегда имеет имя ***main***

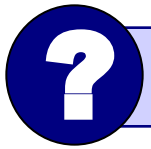
```
public class <имя класса>
    {
    public static void main(String[] args)
        {
        /* основная программа */
        }
    }
```

начало метода

В среде разработки написать **psvm** и нажать **Tab**

«тело» метода (основная часть)

конец метода



Что делает эта программа?

Что происходит дальше?

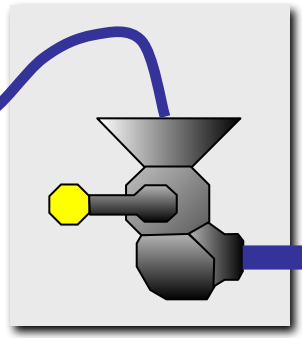
текст программы на Java

First.java

```
public class
{
...
}
```

исходный файл

транслятор



First.class

```
ЪБzЦ2?|ё3БКа
n/36ШпIC+И -
ЦЗ_5 МуРЧ б
s6bд^:/@:лЖ1_
```

байт-код

!

- **по исходному файлу можно восстановить байт-код**
- **байт-код можно выполнить на JVM**

Вывод текста на экран

стандартная функция вывода

```
public class HelloWorld
{
   public static void main (String[] args)
      {
      System.out.print("Привет!");
      }
}
```

вызов стандартного метода ***print*** (вывод)

этот текст будет выведен на экран

Переход на новую строку

последовательность \n
код 10
переход на новую строку

```
public class HelloWorld
{
public static void main (String[] args)
   {
   System.out.print(" Привет,\nВася!");
   }
}
```

на экране:

```
Привет,
Вася!
```

Задания

1. Вывести на экран текст "лесенкой"

```
Вася
    пошел
        гулять
```

2. Вывести на экран рисунок из букв

```
  Ж
 ЖЖЖ
ЖЖЖЖЖ
ЖЖЖЖЖЖЖ
НН  НН
ZZZZZ
```

Программирование на языке Java

Переменные

Что такое переменная?

Переменная – это ячейка в памяти компьютера, которая имеет имя и хранит некоторое значение.

- Значение переменной может меняться во время выполнения программы.
- При записи в ячейку нового значения старое стирается.

Типы переменных

- **int** – целое число в интервале $[-2^{31}, 2^{31}-1]$ (4 байта)
- **float** – вещественное число, *floating point* (4 байта)
- **char** – символ, *character* (2 байта)

Из чего состоит программа?

Переменная – изменяющаяся величина, имеющая имя (ячейка памяти).

Метод – вспомогательный алгоритм, описывающий некоторые действия (например, рисование окружности) или выполняющий вычисления (например, вычисление квадратного корня, **sin**).

Имена классов, переменных, методов

В Java имена могут включать

- Символы алфавита (латиница A-Z, кириллица А-Я и т.д.)

заглавные и строчные буквы различаются

- цифры

имя не может начинаться с цифры

- знак подчеркивания _, знак $

Имена НЕ могут включать

- пробелы
- скобки, знаки +, =, !, ? и др.

Какие имена правильные??

AXby R&B 4Wheel Вася “PesBarbos”
TU154 [QuQu] _ABBA A+B

Ключевые слова Java – 1

Ключевые слова в сочетании с синтаксисом операций и разделителями образуют основу языка Java.

Ключевые слова **нельзя** использовать в качестве имен переменных, классов, методов.

Ключевые слова Java – 2

`abstract`	`continue`	`for`	`new`	`switch`
`assert`	`default`	`goto`	`package`	`synchronized`
`boolean`	`do`	`if`	`private`	`this`
`break`	`double`	`implements`	`protected`	`throw`
`byte`	`else`	`import`	`public`	`throws`
`case`	`enum`	`instanceof`	`return`	`transient`
`catch`	`extends`	`int`	`short`	`try`
`char`	`final`	`interface`	`static`	`void`
`class`	`finally`	`long`	`strictfp`	`volatile`
`const`	`float`	`native`	`super`	`while`

Ключевые слова `const` и `goto` зарезервированы, но не используются.

Зарезервированные слова Java

`true, false, null` – зарезервированные слова в Java, нельзя использовать в качестве имен переменных, классов и т.п.

Объявление переменных

Объявить переменную – определить ее имя, тип, начальное значение, и выделить ей место в памяти.

```
public static v
{
i
f
int Tu104, Il86=23, Yak42;
float x=4.56f, y, z;
char c, c2='A', m;
}
```

Целочисленная переменная **a**

вещественные переменн

целая и дробная части отделяются **точкой**

целые переменные **Tu104**, **Il86** и **Yak42**

вещественные переменные **x**, **y** и **z**
x = 4,56

символьные переменные **c**, **c2** и **m**
c2 = 'A'

Переменные

Переменная – это величина, имеющая имя, тип и значение. Значение переменной можно изменять во время работы программы.

Объявление переменных (выделение памяти):

```
int a, b;
double Q;
char s1, s2;
```

Простые типы данных (primitive)

char { символьный (один символ) }

можно использовать русские буквы!

byte, short, int, long {целые типы}

float, double { вещественные типы }

целая и дробная часть отделяются точкой

boolean { логический }

может принимать два значения:

- true (истина, «да»)
- false (ложь, «нет»)

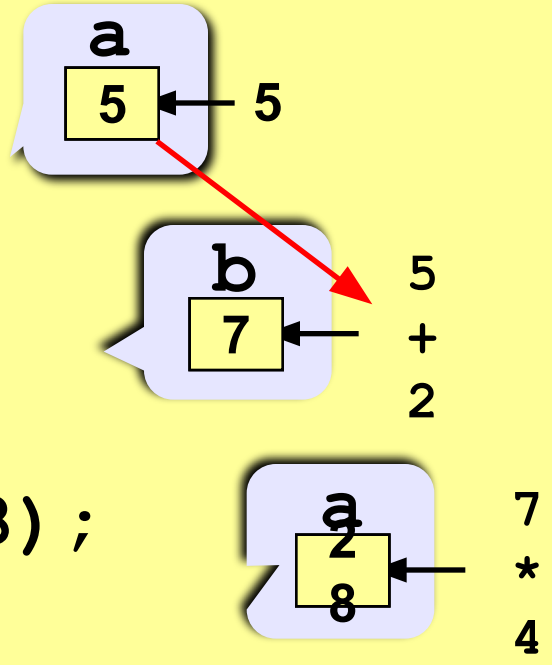
Как изменить значение переменной?

Оператор – это команда языка программирования высокого уровня.

Оператор присваивания служит для изменения значения переменной.

Пример:

```
{
int a, b;

   a = 5;
   b = a + 2;
   a = (a + 2)*(b - 3);
 }
```



Оператор присваивания

куда

что

<имя переменной> = <выражение>;

Арифметическое выражение может включать

- имена переменных
- знаки арифметических операций:

  + - * / %

  умножение

  деление

  остаток от деления

- вызовы методов
- круглые скобки ( )

Для чего служат круглые скобки?

Какие операторы записаны неверно?

```
public static void main (String[] args)
{
 int a, b;
 double  x, y;
   a = 5;
   10 = x;
   y = 7,8;
   b = 2.5;
   x = 2*(a + y);
   a = b + x;
}
```

имя переменной должно быть слева от знака **=**

целая и дробная часть отделяются **точкой**

нельзя записывать вещественное значение в целочисленную переменную

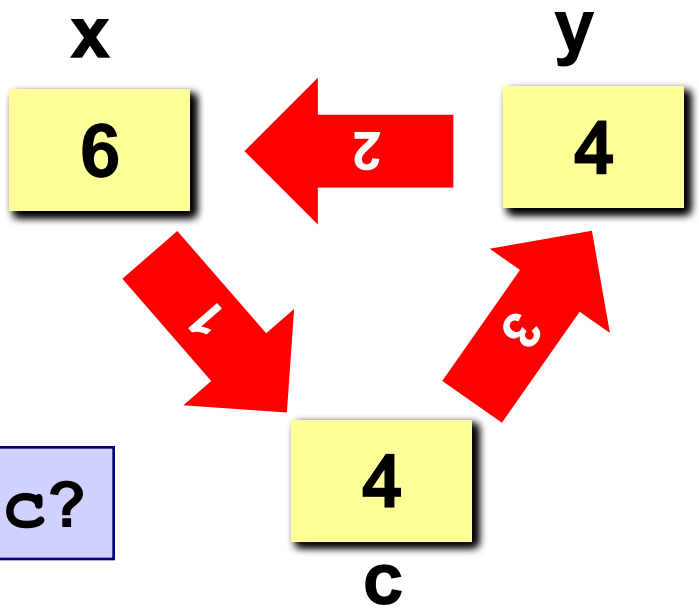
Обмен значений переменных – 1

Задача: поменять местами содержимое двух чашек.

Задача: поменять местами содержимое двух ячеек памяти.

```
x = y;
y = x;
```

```
c = x;
x = y;
y = c;
```



Можно ли обойтись без c?

Обмен значений переменных – 2

```
public static void main(...) {
   int x = in.nextInt();
   int y = in.nextInt();
   x = x + y;
   y = x - y;
   x = x - y;
}
```

Можно ли обойтись без вспомогательной переменной при обмене 3 переменных?

Особенность деления в Java

При делении целых чисел остаток отбрасывается!

```
public static void main(…)
{
int a = 7;
float x;
x = a / 4;
x = 4 / a;
x = (float)a / 4;
x = 1.*a / 4;
}
```

1

0

1.75

1.75

Сокращенная запись операций в Java

полная запись	сокращенная запись
`a = a + 1;` инкремент	`a++;`
`a = a + b;`	`a += b;`
`a = a - 1;` декремент	`a--;`
`a = a - b;`	`a -= b;`
`a = a * b;`	`a *= b;`
`a = a / b;`	`a /= b;`
`a = a % b;`	`a %= b;`

Ручная прокрутка программы

```
public static void
 main(…) {
   int a, b;
   a = 5;
   b = a + 2;
   a = (a + 2)*(b – 3);
   b = a / 5;
   a = a % b;
   a++;
   b = (a + 14) % 7;
}
```

a	b
?	?
5	
	7
28	
	5
3	
4	
	4

Порядок выполнения операций

- вычисление выражений в скобках
- умножение, деление, % слева направо
- сложение и вычитание слева направо

```
2  3  5  4   1    7  8   6
z  = (5*a*c+3*(c-d))/a*(b-c)/ b;
```

$$z = \frac{5ac + 3(c-d)}{ab}(b-c)$$

$$x = \frac{a^2 + 5c^2 - d(a+b)}{(c+d)(d-2a)}$$

```
2  6  3  4  7  5   1    12   8  11  10
x =(a*a+5*c*c-d*(a+b))/((c+d)*(d-2*a));
```

Сложение двух чисел

Задача. Ввести два целых числа и вывести на экран их сумму.

Простейшее решение:

```
import java.util.Scanner;
public static void main (…) {
Scanner in = new Scanner (System.in);
 int a, b, c;
 System.out.print("Введите a");
 a = in.nextInt();
 System.out.print("Введите b");
 b = in.nextInt();
 c = a + b;
 System.out.println(c);
 }
```

Подключение пакета

Создание экземпляра класса

Чтение из входного потока целого числа

Ввода данных с клавиатуры

```
Scanner in = new Scanner (System.in);
a = in.nextInt ();     /* ввод целого
 значения и присваивание переменной a */

b = in.nextDouble (); /* ввод
 вещественного значения и присваивание
 переменной b */
```

Оператор вывода

```
System.out.print ( a );    /* вывод
                  значения переменной a */

System.out.println ( a ); /* вывод
                  значения переменной a
                  и переход на новую
                  строчку /*
System.out.println ( “Привет!” ); /*
                  вывод текста */
System.out.println( “Ответ: ” + c );
          /* вывод текста и значения
          переменной c */
System.out.println ( a + “+” + b+ “=”
          + c );
```

Полное решение

```
 int a, b, c;
System.out.println("Введите число a");
 a = in.nextInt();
System.out.println("Введите число b");
 b = in.nextInt();
 c = a + b;
 System.out.println(a+ "+                c );
```

Протокол:

```
Введите число a
25
Введите число b
30
25+30=55
```

компьютер

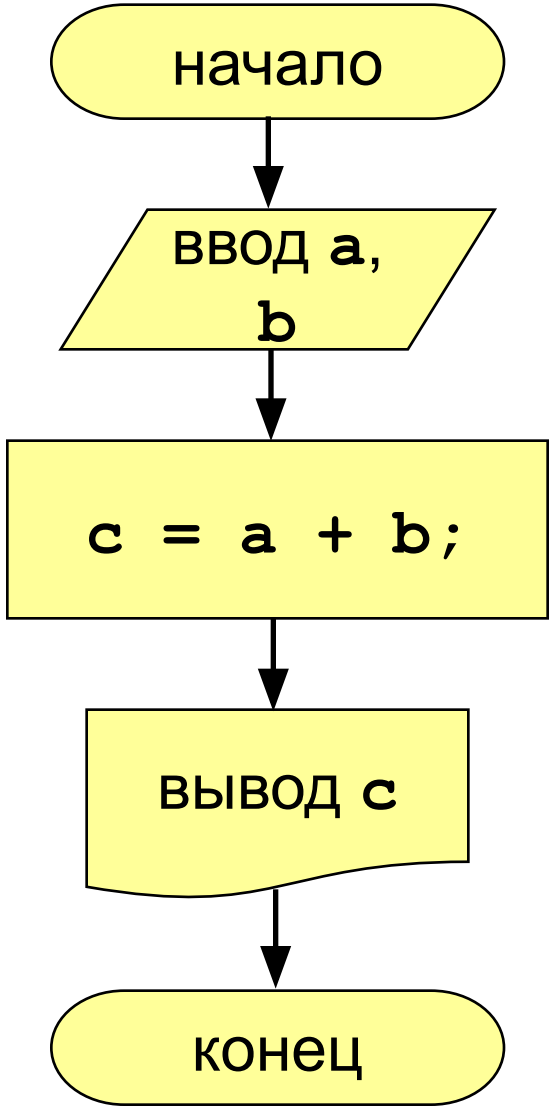
пользователь

компьютер

пользователь

компьютер

Блок-схема линейного алгоритма



блок «начало»

блок «ввод»

блок «процесс»

блок «вывод»

блок «конец»

Задания

1. **Ввести три числа, найти их сумму и произведение.**

   Пример:

```
Введите три числа:
4   5   7
4+5+7=16
4*5*7=140
```

2. **Ввести три числа, найти их сумму, произведение и среднее арифметическое.**

   Пример:

```
    Введите три числа:
4   5   7
4+5+7=16
4*5*7=140
(4+5+7)/3=5.33
```

Программирование на языке Java

6. Форматный вывод

Программирование на языке Java

Тема 6. Форматный вывод

Сложение двух чисел

Задача. Ввести два целых числа и вывести на экран их сумму.

Простейшее решение:

```
int a, b, c;
a = in.nextInt();
b = in.nextInt();
c = a + b;
System.out.print(a+"+"+b+"="+c);
```

Форматный вывод

Форматный вывод – вывод в различные потоки значений **разных типов**, отформатированных согласно заданному формату (шаблону).

Формат определяется составленной по специальным правилам строкой.

Форматный вывод в Java

```
System.out.printf(<форматная строка>,
                  <список аргументов>);
```

Форматная строка – символьная строка, которая задает шаблон вывода аргументов.

Форматная строка состоит из:

- символов, которые копируются в выходной поток;
- спецификаторов формата, определяющих способ, в соответствии с которым должны отображаться последующие аргументы.

Спецификатор формата

Спецификатор формата начинается со знака процента с последующим спецификатором преобразования.

Пример. Спецификатор формата для десятичного целого числа – **%d**.

Форматный вывод. Пример

printf – форматный вывод

Форматная строка

Целое число подставляется из переменной a

```
System.out.printf("%d", a);
```

a – имя переменной

```
a
```

```
12
```

12 – значение переменной **a**

Спецификаторы формата – 1

Спецификатор	Применяемое преобразование
`%a, %A`	Шестнадцатеричное с плавающей точкой
`%b, %B`	Булевское
`%c`	Символ
`%d`	Десятичное целое число
`%e, %E`	Научная нотация
`%f`	Десятичное с плавающей точкой
`%g, %G`	Либо %e, либо %f, смотря что короче
`%o`	Восьмеричное целое
`%n`	Символ перевода строки
`%s, %S`	Строка
`%x, %X`	Шестнадцатеричное целое
`%%`	Вставляет символ %

Спецификаторы формата – 2

Некоторые спецификаторы имеют заглавную и прописную формы. При использовании заглавной формы буквы отображаются в верхнем регистре.

Форматирование целых чисел – 1

вывести целое число и перевод строки

это число взять из переменной **c**

```
System.out.printf ("%d%n", c);
```

```
System.out.printf ("Результат: %d",
c);
```

```
System.out.printf("%d+%d=%d%n", a, b, c
);
```

форматная строка

список аргументов

```
System.out.printf("%d+%d=%d%n",a, b,
a+b);
```

арифметическое выражение

Форматирование целых чисел – 2

```
System.out.printf ("16-ая с/с %x", 196);
```

```
16-ая с/с c4
```

```
System.out.printf ("16-ая с/с %X", 196);
```

```
16-ая с/с C4
```

```
System.out.printf ("8-ая с/с %o", 196);
```

```
8-ая с/с 304
```

Форматирование вещественных чисел

```
double x = 12345.6789;
System.out.printf ("%f",
x);
```

```
12345,678900
```

минимальное число позиций, **6 цифр** в дробной части

```
System.out.printf ("%e",
x);
```

```
1.234568e+04
```

Научная нотация: $1{,}23456 \cdot 10^4$

```
System.out.printf ("%E",
x);
```

```
1.234568E+04
```

Спецификаторы %n и %%

Отличаются от других тем, что они не соответствуют аргументу. Представляют собой управляющие последовательности:

%n – вставляет перевод строки

%% – вставляет знак процента.

```
System.out.printf ("Копирование файла
%nПеремещение на %d%% завершено",
88);
```

```
Копирование файла
Перемещение на 88% завершено
```

Указание минимальной ширины поля – 1

Спецификатор минимальной ширины – целое число, помещенное между символом `%` и кодом преобразования формата.

Спецификатор минимальной ширины дополняет вывод пробелами, обеспечивая заданную минимальную длину.

Если строка или число получаются длиннее, чем заданный минимум, то число выводится полностью.

По умолчанию, дополнение осуществляется пробелами.

Указание минимальной ширины поля – 2

```
int x = 1234;
System.out.printf ("%d\n", x);
```

```
1234
```

минимальное число позиций

```
System.out.printf ("%9d\n",
x);
```

```
     1234
```

всего 9 позиций

Указание минимальной ширины поля – 3

Чтобы дополнить число лидирующими нулями, нужно поместить 0 перед спецификатором ширины поля.

```
System.out.printf ("%09d\n",
x);
```

```
000001234
```

всего 9 позиций, пустые заполнены нулями

Указание минимальной ширины поля – 4

```
double x = 10.12345;
System.out.printf("|%f|%n|%12f|%n|%012f|",
x, x, x);
```

```
|10,123450|
|   10,123450|
|00010,123450|
```

Указание точности – 1

Спецификатор точности может быть применен к спецификаторам формата **`%f`**, **`%e`**, **`%g`** и **`%s`**.

Спецификатор точности следует за спецификатором минимальной ширины поля (если таковой имеется) и состоит из точки с последующим целым числом.

Указание точности – 2

Спецификатор точности для данных с плавающей точкой (**%f** или **%e**) определяет количество отображаемых десятичных разрядов.

%10.4f – число в 10 символов шириной с 4 разрядами после запятой.

При использовании **%g** точность определяется количеством значащих десятичных разрядов.

Точность по умолчанию – 6 знаков после запятой.

Указание точности – 3

```
double x = 12345.6789;
System.out.printf ("%10.3f", x);
```

```
 12345,679
```

всего 10 позиций, **3 цифры** в дробной части

```
System.out.printf ("%10.2e", x);
```

```
  1.23e+04
```

всего 10 позиций, **2 цифры** в дробной части мантиссы

Вопрос. Как вывести `00012345,68`

```
System.out.printf ("%011.2f", x);
```

Указание точности – 4

Для строк спецификатор точности задает максимальную ширину поля. Если строка длиннее максимальной ширины, конечные символы усекаются.

```
System.out.printf ("%.15s%n",
"Форматировать в Java очень просто");
```

```
Форматировать в
```

15 символов

Флаги формата – 1

Флаги формата позволяют управлять различными аспектами преобразования.

Все флаги формата – одиночные символы, которые следуют за знаком `%` в спецификаторе формата.

Флаги формата – 2

Флаг	Эффект
-	Выравнивание влево
0	Вывод дополняется нулями вместо пробелов
Пробел	Положительным числам предшествует пробел
+	Положительным числам предшествует знак +
,	Числовые значения, включающие групповые разделители
(	Отрицательные числовые значения заключены в скобки

Внимание! Не все флаги применимы ко всем спецификаторам формата.

Выравнивание вывода

По умолчанию весь вывод выравнивается вправо.

Для выравнивания по левому краю, нужно поставить знак минус после сразу после %.

```
System.out.printf ("|%10.2f|%n",123.123);
System.out.printf ("|%-10.2f|%n",123.123);
```

```
|    123,12|
|123,12    |
```

Флаги пробела, +, 0 и (– 1

Данные флаги работаю со знаком числа:

```
System.out.printf ("%d%n",100);
System.out.printf ("% d%n",100);
System.out.printf ("%+d%n",100);
System.out.printf ("%05d%n",100);
System.out.printf ("%(d%n",100);
```

```
100
 100
+100
00100
100
```

Флаги пробела, +, 0 и (– 2

Данные флаги работаю со знаком числа:

```
System.out.printf ("%d%n",-100);
System.out.printf ("% d%n",-100);
System.out.printf ("%+d%n",-100);
System.out.printf ("%05d%n",-100);
System.out.printf ("%(d%n",-100);
```

```
-100
-100
-100
-0100
(100)
```

Флаг запятая

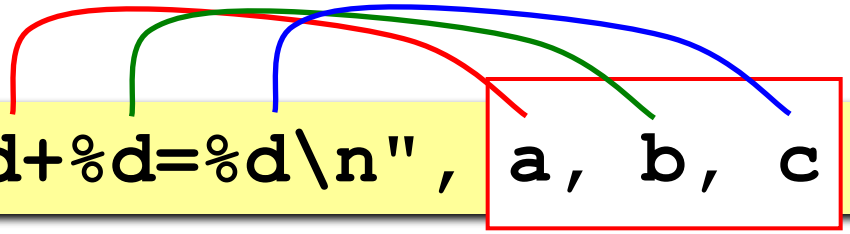
При отображении больших чисел удобно использовать разделители групп. Например 1234567 читается легче в виде 1 234 567. Для добавления спецификаторов группирования служит флаг запятая.

```
System.out.printf ("%,.2f", 4356783497.34);
```

```
4 356 783 497,34
```

Использование индекса аргументов – 1

Обычно порядок аргументов и спецификаторов совпадает (слева направо), т.е. первый спецификатор относится к первому аргументу, второй – ко второму и т.д.



```
System.out.printf("%d+%d=%d\n", a, b, c
);
```

Используя индексы аргументов, можно управлять тем, к какому из аргументов относится спецификатор формата.

Использование индекса аргументов – 2

Индекс аргумента следует за `%` в спецификаторе формата и имеет вид **`n$`**, где **`n`** – индекс нужного аргумента, начиная с 1.

```
System.out.printf("%3$d %1$d %2$d",
                  10, 20, 30);
```

```
30 10 20
```

Использование индекса аргументов – 3

Преимущество индексирования аргументов: повторное использование аргумента.

```
System.out.printf("%1$d в шестнадцатеричном
        формате это %1$X%n", 255);
```

```
255 в шестнадцатеричном формате это FF
```

Полное решение

Задача. Ввести два целых числа и вывести на экран их сумму.

```
int a, b, c;
  a = in.nextInt();
  b = in.nextInt();
  c = a + b;
  System.out.printf("%d+%d=%d", a, b, c);
}
```

Программирование на языке Java

8. Целые типы данных

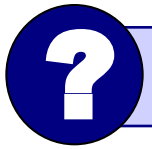
Программирование на языке Java

Тема 8. Целые типы данных

Типы данных

Тип данных – множество значений и набор операций, которые можно применять к этим значениям.

В математике множество целых чисел бесконечно, в компьютерных программах это не так.



Почему?

Целые типы данных – 1

В Java 4 целых типа данных: **`byte`**, **`short`**, **`int`** и **`long`**.

Все целые типы в Java представляют значения со знаком – положительные и отрицательные.

Ширина целочисленного типа представляет собой занимаемый объем памяти.

Целые типы данных – 2

Тип	Размер	Min	Max
`byte` (байт)	8 бит	-128	+127
`short` (короткое целое)	16 бит	-2^{15}	$2^{15}-1$
`int` (целое число)	32 бита	-2^{31}	$2^{31}-1$
`long` (целое число двойного размера)	64 бита	-2^{63}	$2^{63}-1$

Объявление переменных целого типа

Объявить переменную – определить ее имя, тип, начальное значение, и выделить ей место в памяти.

```
public static void main()
{
byte a;
short b, c;
int d;
long x=4, y, z;
}
```

Операции над величинами целого типа

+ – сложение
- – вычитание
* – умножение
/ – деление нацело
% – получение остатка от деления

Недопустимые операции

Деление на 0 и вычисление остатка от деления на 0 невозможны

```
int x = 1 / 0;
```

Ошибка времени выполнения

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
```

Особенность деления в Java

При делении целых чисел остаток отбрасывается!

```
public static void main()
{
int a = 7;

double x;

x = a / 4;

x = 4 / a;

x = (double)a / 4;

x = 1. * a / 4;

x = a / 4 * 1.;
}
```

1.0

0.0

1.75

1.75

1.0

Примеры выполнения операций / и %

```
 19 / 4     =  4
-19 / 4     = -4
 19 / (-4)  = -4
-19 / (-4)  =  4
```

```
 19 % 4     =  3
-19 % 4     = -3
 19 % (-4)  =  3
-19 % (-4)  = -3
```

Определение цифр числа

Задача. Вывести на экран число сотен, десятков и единиц трехзначного числа.

```
public static void main()
{
int x = in.nextInt();
int one = x % 10;
int dec = (x / 10) % 1
int hun = (x / 100) % 10;
System.out.printf(……);
}
```

456

6

45

4

5

4

Целочисленные константы – 1

Пример целочисленных констант: 1, 2, 42, 93, ...

В числовых константах используются 4 вида представления:

- **десятичное**;
- **двоичное (начиная с Java 8)**
  обозначаются ведущим нулем и символом **b**:
  `0b1001, 0B11, 0b120`
- **восьмеричное**
  обозначаются ведущим нулем:
  `054, 0123, 091`
- **шестнадцатеричное**
  обозначаются ведущим нулем и символом **x**:
  `0X54, 0x1Ab, 0X91, 0xQwerty`

Целочисленные константы – 2

Целочисленные константы создают значение типа **`int`**.

Для создания константы типа **`long`** компилятору нужно явно указать тип, для этого к константе дописывают строчную или прописную букву **`L`**.

```
long x, y;
x = 0x7fffffffffffffffL;
y = 923789344394779L;
```

Целочисленные константы – 3

Начиная с Java 7 в описании константы можно использовать символы «_» (подчеркивание)

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes =
       0b11010010_01101001_10010100_10010010;
```

Целочисленные константы – 4

Символ «_» (подчеркивание) можно использовать в любом месте, кроме следующих

- вначале или конец числа
- перед суффиксом **L**
- В позициях, где ожидается строка цифр

```
int x1 = _52;
int x2 = 5_2;
int x3 = 52_;
int x4 = 5_______2;
int x5 = 0_x52;
int x6 = 0x_52;
int x7 = 0x5_2;
int x8 = 0x52_;
```

Инкремент и декремент

Инкремент – операция, увеличивающая переменную на единицу.

```
x = 5;
x++;
```

6

Декремент – операция, уменьшающая переменную на единицу.

```
x = 5;
x--;
```

4

Инкремент и декремент работают быстрее, чем обычное прибавление единицы, т.к. для их вычисления используются отдельные низкоуровневые команды, выполняемые на аппаратном уровне.

Префиксная и постфиксная формы

Оператор инкремента можно записывать с обоих сторон:

- **префиксная форма** (прекремент) – оператор «++» или «--» записывается перед переменной

```
++x;
--x;
```

```
y = ++x;
```

=

```
x = x+1;
y = x;
```

1. выполняется операция,
2. вычисляется результат.

- **постфиксная форма** (посткремент) – оператор «++» или «--» записывается после переменной

```
x++;
x--;
```

```
y = x++;
```

=

```
y = x;
x = x+1;
```

1. вычисляется результат,
2. выполняется операция.

Префиксная и постфиксная формы

x	Выражение	Итоговое y	Итоговое x
5	`y = x++;`	5	6
5	`y = ++x;`	6	6
5	`y = x--;`	5	4
5	`y = --x;`	4	4

Инкремент и декремент. Задание

Вычислите значение переменной

```
x = 5; p = 5; q = 5;
w = 5; k = 5;

p *= x++;
q /= ++x;
w += --x;
k += x--;
```

x	5
p	25
q	0
w	11
k	11

Переполнение

В Java возможна ситуация переполнения

Тип int занимает 32 бита, минимум -2^{31}, максимум $2^{31}-1$

```
int x = 2147483647;
x++;
```

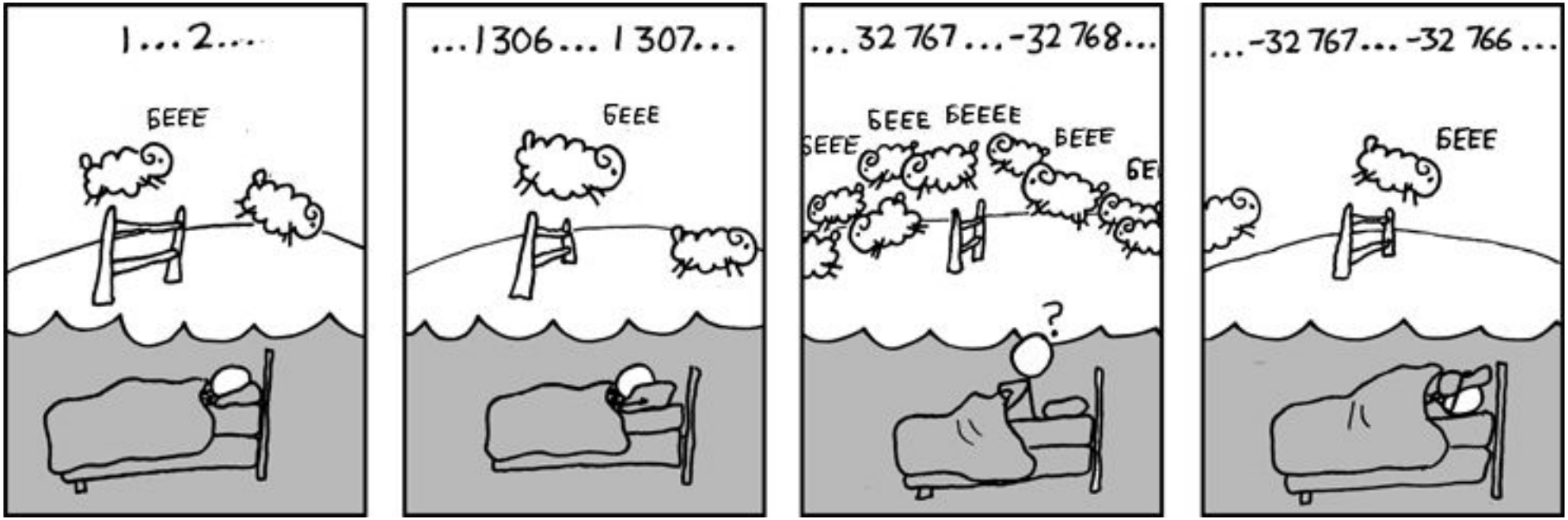
$2^{31} - 1$

-2^{31}

Java не сообщит об ошибке переполнения!

Будьте внимательны при работе с числами, близкими к максимальному или минимальному значению типа.

Переполнение



Задача

Задача. Отобразить текущее время в формате Часы:минуты:секунды, например 13:19:08, если метод **`System.currentTimeMillis()`** возвращает количество прошедших миллисекунд с начала эпохи Unix (01-01-1970 00:00:00) по Гринвичу

- 1 секунда = 1000 миллисекунд
- для нашего часового пояса нужно прибавить 9 часов
- Какой тип данных будем использовать?

Программирование на языке Java

9. Типы с плавающей точкой
10. Методы класса Методы класса Math

Программирование на языке Java

Тема 9. Типы с плавающей точкой

Типы с плавающей точкой – 1

Числа с плавающей точкой, (в математике действительные или вещественные числа), используются при вычислениях, которые требуют получения результата с точностью до определенного десятичного знака.

Пример. Вычисление квадратного корня, трансцендентных функций (**`sin()`**, **`cos()`**, …).

В Java существует два типа с плавающей точкой: **`float`** и **`double`** (числа одинарной и двойной точности).

Типы с плавающей точкой – 2

Стандарт **IEEE754**

Число представлено в виде $\pm m \cdot 2^e$,

где m – мантисса, e – порядок (экспонента)

Тип	Бит	Знак	Мантисса	Порядок	Min	Max
float	32	1	23	8	1.4e-045	3.4e038
double	64	1	52	11	4.9e-324	1.8e308

Типы с плавающей точкой – 3

Тип **`float`** используется, когда требуется дробная часть без **особой точности**, например, для представления денежных сумм в рублях и копейках.

Применение типа **`double`** наиболее рационально, когда требуется сохранение точности множества последовательных вычислений или манипулирование большими числами.

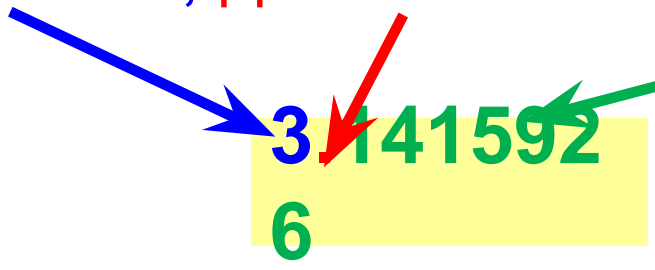
Все трансцендентные математические функции (**`sin()`**, **`cos()`**, **`sqrt()`**,…) возвращают значения типа **`double`**.

Константы с плавающей точкой – 1

Числа с плавающей точкой представляют десятичные значения с дробной частью.

Стандартная форма записи десятичного числа состоит из:

целого числа; десятичной точки; дробной части.

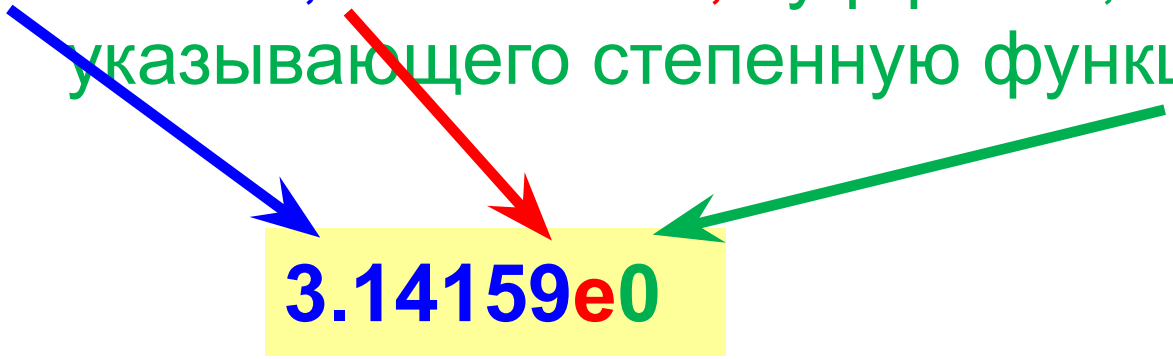


Константы с плавающей точкой – 2

Научная форма записи десятичного числа

состоит из:

мантиссы; символа E, суффикса,
указывающего степенную функцию числа 10



Константы с плавающей точкой – 3

Задача. Записать в стандартной форме

1.44e-6 = 0.00000144

0.832e8= 83200000.0

0.000034e7= 340.0

0.00524e-1= 0.000524

Константы с плавающей точкой – 4

По умолчанию в **Java** константам с плавающей точкой присвоен тип **double**.

Для указания константы типа **float**, к ней нужно дописать символ **F** или **f**.

```
float x;
x = 23.48f;
```

Также существует суффикс **d** или **D**

```
double y = 3D;
```

Особые случаи: бесконечность

- Деление положительного числа на 0.0 дает +∞
- Деление отрицательного числа на 0.0 дает −∞
- Переполнение дает +∞ или −∞, в зависимости от направления

```
double posInfinity =  1.0 / 0.0;
double negInfinity = -1.0 / 0.0;
double x = posInfinity + 1;
```

Какое значение примет x?

Особые случаи: NaN

- Деление 0.0 на 0.0 дает **NaN** (Not a Number – не число)
- Любая арифметическая операция с NaN дает NaN
- NaN != NaN

```
double nan =  0.0 / 0.0;
nan = posInfinity + negInfinity ;
```

Значение NaN

К получению **NaN** приводит:

- все математические операции с NaN;
- деление нуля на ноль;
- деление бесконечности на бесконечность;
- умножение нуля на бесконечность;
- сложение бесконечностей с противоположными знаками;
- вычисление квадратного корня отрицательного числа;
- логарифмирование отрицательного числа.

Точность вычислений – 1

- Для любого a вещественного типа существует ε>0: **a + ε == a**

```
double t = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +
           0.1 + 0.1 + 0.1 + 0.1 + 0.1;
System.out.println(t);
```

```
0.9999999999999999
```

Почему?

- Из-за погрешности вычислений, поэтому при сравнении вещественных чисел проверяют не равенство (a==b), а близость этих чисел

$$|a - b| < \varepsilon$$

Почему по модулю?

Точность вычислений – 2

- При сравнении вещественных чисел проверяют не равенство (a==b), а близость этих чисел

$$|a - b| < \varepsilon$$

```
double t = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +
           0.1 + 0.1 + 0.1 + 0.1 + 0.1;
System.out.println(t == 1);
double eps = 1e-10;
System.out.println(Math.abs(t - 1) < eps);
```

```
false
true
```

Модификатор `strictfp`

- Java использует математический сопроцессор (FPU – Floating Point Unit) для вычислений с плавающей точкой;
- Регистры FPU могут быть шире 64 бит
- Результаты вычислений могут отличаться

- Модификатор **`strictfp`** включает режим строгой совместимости, результаты будут идентичны на любом процессоре

Ввод с клавиатуры

Аналогично вводу целых чисел

```
Scanner in = new Scanner(System.in);
System.out.println("Введите x");
float x = in.nextFloat();
System.out.println("Введите y");
double y = in.nextDouble();
```

```
Введите x
12,567
Введите y
0,00034
```

Внимание! При считывании с клавиатуры используется разделитель запятая

Почему запятая?

Форматный вывод – 1

Спецификаторы формата **%f, %e, %g**.

```
double x = 12345.6789;
System.out.printf ("%f",
x);
```

```
12345,678900
```

минимальное число позиций, **6 цифр** в дробной части

```
System.out.printf ("%e",
x);
```

```
1.234568e+04
```

Научная нотация: $1{,}23456 \cdot 10^4$

Форматный вывод. Указание точности

```
double x = 12345.6789;
System.out.printf ("%10.3f", x);
```

всего 10 позиций, **3 цифры** в дробной части

```
 12345,679
```

```
System.out.printf ("%10.2e",
x);
```

```
  1.23e+04
```

всего 10 позиций, **2 цифры** в дробной части мантиссы

Задача

Задача. Вычислить площадь круга заданного радиуса.

```
double pi, s, r;
r = in.nextDouble();
pi = 3.1415926;
s = pi * r * r;


System.out.printf("Площадь = %f", s);
```

Считываем с клавиатуры

Приблизительное значение π

Вычисление площади круга

Программирование на языке Java

Тема 10. Методы класса Math

Класс Math

Разработчику на **Java** доступно множество готовых (или библиотечных) классов и методов, полезных для использования в собственных программах.

Наличие библиотечных решений позволяет изящно решать множество типовых задач.

Класс **Math** содержит методы, которые используются в геометрии и тригонометрии, а также некоторые методы общего назначения.

Константы класса Math

~3,14

~2,72

2 константы типа **`double`**:

`Math.PI` – число π с точностью в 15 десятичных знаков.

`Math.E` – основание натурального логарифма с точностью в 15 десятичных знаков.

```
System.out.println(Math.PI);
System.out.println(Math.E);
```

```
3.141592653589793
2.718281828459045
```

Прямые трансцендентные методы

Метод	Описание
`double sin(double arg)`	Возвращает синус угла **arg**, переданного в **радианах**
`double cos(double arg)`	Возвращает косинус угла **arg**, переданного в **радианах**
`double tan(double arg)`	Возвращает тангенс угла **arg**, переданного в **радианах**

Прямые трансцендентные методы. Пример

```
System.out.println(Math.sin(Math.PI/2));
System.out.println(Math.cos(Math.PI/2));
System.out.println(Math.tan(Math.PI/4));
```

```
1.0
6.123233995736766E-17
0.9999999999999999
```

Значение близкое к нулю

Значение близкое к единице

Почему не 0 и 1 ?

Обратные трансцендентные методы

Метод	Описание
`double asin(double arg)`	Возвращает угол, синус которого равен **`arg`**.
`double acos(double arg)`	Возвращает угол, косинус которого равен **`arg`**.
`double atan(double arg)`	Возвращает угол, тангенс которого равен **`arg`**.
`double atan2(double x, double y)`	Возвращает угол, тангенс которого равен **`x/y`**.

Обратные трансцендентные методы. Пример

```
System.out.println(Math.asin(1)*2);
System.out.println(Math.acos(1));
System.out.println(Math.atan(0));
System.out.println(Math.atan2(1,1)*4);
```

```
3.141592653589793
0.0
0.0
3.141592653589793
```

Гиперболические методы

Метод	Описание
`double sinh(double arg)`	Возвращает гиперболический синус угла **`arg`**, переданного в **радианах**.
`double cosh(double arg)`	Возвращает гиперболический косинус угла **`arg`**, переданного в **радианах**.
`double tanh(double arg)`	Возвращает гиперболический тангенс угла **`arg`**, переданного в **радианах**.

Экспоненциальные методы

Метод	Описание
`double exp(double arg)`	Возвращает экспоненту **arg**.
`double log(double arg)`	Возвращает натуральный логарифм **arg**.
`double log10(double arg)`	Возвращает логарифм по основанию 10 от **arg**.
`double pow(double y, double x)`	Возвращает **y** в степени **x**
`double sqrt(double arg)`	Возвращает квадратный корень из **arg**.

Экспоненциальные методы. Пример

```
System.out.println(Math.exp(1));
System.out.println(Math.exp(2));
System.out.println(Math.log(1));
System.out.println(Math.log(Math.E));
System.out.println(Math.log10(1000));
System.out.println(Math.pow(2, 3));
System.out.println(Math.sqrt(25));
```

```
2.7182818284590455
7.38905609893065
0.0
1.0
3.0
8.0
5.0
```

Абсолютное значение

Метод	Описание
`int abs(int arg)`	Возвращает абсолютное значение **arg**.
`long abs(long arg)`	Возвращает абсолютное значение **arg**.
`float abs(float arg)`	Возвращает абсолютное значение **arg**.
`double abs(double arg)`	Возвращает абсолютное значение **arg**.

Чем эти методы отличаются?

Абсолютное значение. Пример

```
System.out.println(Math.abs(5));
System.out.println(Math.abs(-5));
System.out.println(Math.abs(10.3));
System.out.println(Math.abs(-10.3));
```

```
5
5
10.3
10.3
```

Методы округления

Метод	Описание
`double ceil(double arg)` потолок	Возвращает наименьшее целое число, которое больше **`arg`**.
`double floor(double arg)` пол	Возвращает наибольшее целое число, которое меньше или равно **`arg`**.
`int round(float arg)`	Возвращает **`arg`**, округленное до ближайшего **`int`**.
`long round(double arg)`	Возвращает **`arg`**, округленное до ближайшего **`long`**.

Методы округления. Пример

потолок

```
System.out.println(Math.ceil(5.4));
System.out.println(Math.floor(5.4));
System.out.println(Math.round(5.4));
System.out.println(Math.round(5.6));
System.out.println(Math.round(5.5));
```

```
6.0
5.0
5
6
6
```

пол

Максимум

Метод	Описание
`int max(int x, int y)`	Возвращает большее из двух чисел **x** и **y**.
`long max(long x, long y)`	Возвращает большее из двух чисел **x** и **y**.
`float max(float x, float y)`	Возвращает большее из двух чисел **x** и **y**.
`double max(double x, double y)`	Возвращает большее из двух чисел **x** и **y**.

Минимум

Метод	Описание
`int min(int x, int y)`	Возвращает меньшее из двух чисел **x** и **y**.
`long min(long x, long y)`	Возвращает меньшее из двух чисел **x** и **y**.
`float min(float x, float y)`	Возвращает меньшее из двух чисел **x** и **y**.
`double min(double x, double y)`	Возвращает меньшее из двух чисел **x** и **y**.

Максимум и минимум. Пример

```
System.out.println(Math.max(2, 4));
System.out.println(Math.min(2, 4));
System.out.println(Math.max(10.3, 4));
System.out.println(Math.min(10.3, 4));
```

```
4
2
10.3
4.0
```

Почему 4.0, а не 4?

Как вычислить максимум из трех чисел?

Вспомогательные функции

Метод	Описание
`double toDegrees(double angle)`	Преобразует радианы в градусы. Переданный в **`angle`** угол должен быть указан в радианах. Возвращается результат в градусах.
`double toRadians(double angle)`	Преобразует градусы в радианы. Переданный в **`angle`** угол должен быть указан в градусах. Возвращается результат в радианах.

Вспомогательные функции. Пример

```
System.out.println(Math.toDegrees(Math.PI));
System.out.println(Math.toDegrees(Math.PI/4))
;
System.out.println(Math.toRadians(180));
System.out.println(Math.toRadians(90));
```

```
180.0
45.0
3.141592653589793
1.5707963267948966
```

Псевдослучайные числа

Метод **`Math.random()`** возвращает псевдослучайное вещественное число из промежутка [0;1).

```
System.out.println(Math.random());
System.out.println(Math.random());
System.out.println(Math.random());
```

```
0.8701659383706429
0.5194884184661862
0.3324845299964946
```

Случайные числа

Целые числа в заданном интервале – 1

Целые числа в интервале [0,n-1]:

```
(int)(Math.random()* n);
```

Примеры:

```
x = (int)(Math.random () * 100); // [0,99]
x = (int)(Math.random () * z);   // [0,z-1]
```

Целые числа в интервале [a,b]:

```
x = (int)(Math.random()*(b - a + 1)) + a;
// [a,b]
```

Целые числа в заданном интервале – 2

Задача. Получить случайное число в интервале от -10 до 10.

```
int x = (int)(Math.random() * 21) - 10;
```

Методы класса Math. Задача – 1

```
System.out.println(Math.abs(-2.33));
System.out.println(Math.round(Math.PI));
System.out.println(Math.round(9.5));
System.out.println(Math.round(9.5-0.001))
;
System.out.println(Math.ceil(9.4));
double c = Math.sqrt(3*3 + 4*4);
System.out.println(c);
double s1 = Math.cos(Math.toRadians(60));
System.out.println(s1);
```

```
2.33
3
10
9
10.0

5.0

0.5
```

Методы класса Math. Задача – 2

Записать в стандартной форме записи числа

-12.3E+2 **= -1230.0**

-0.8E-6 **= -0.0000008**

1E+3 **= 1000.0**

+1E-6 **= 0.000001**

Методы класса Math. Задача – 3

Какие круглые скобки можно убрать, не изменив порядка вычисления выражений

```
(a+b)/c
a+b/c
a/(b*c)
x1/x2*y
Math.sqrt(p)*q/r
a-b-c-d-e
(a-b)-(c-d)-e
```

Методы класса Math. Задача – 4

Записать следующие выражения на Java

x^5

```
Math.pow(x,5)
```

$\cos^8 x^4$

```
Math.pow(Math.cos(Math.pow(x,4)),8
)
```

$\log_{10}(x/5)$

```
Math.log10(x/5
)
```

$|x^{-3}|$

```
Math.abs(Math.pow(x,-3)
)
```

2^{x+1}

```
Math.pow(2,x+1
)
```

$\sin 8°$

```
Math.sin(Math.toRadians(8))
```

Методы класса Math. Задача – 5

Определить типы выражений

```
double x, y, z;
int i, j, k;
x+y*i;
i+j-k;
i/j+x;
i*x+j*y;
```

double

int

double

double

Задача

Задача. Дано целое число **x**. Вывести количество цифр данного числа.

```
int x;
x = in.nextInt();
double count = Math.floor(Math.log10(x)) +
1;
System.out.println(count);
```

Что плохо?

Программирование на языке Java

11. Логический тип данных

Программирование на языке Java

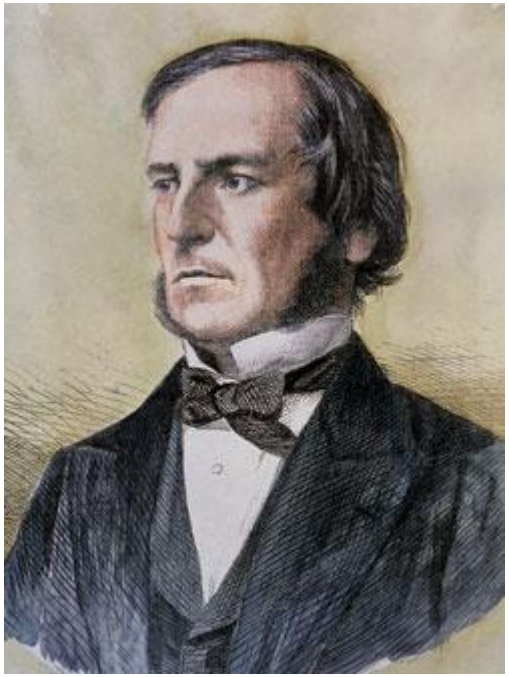
Тема 11. Логический тип данных

Логический тип данных

Логический тип данных используется для переменных, которые принимают всего два значения: **`true`** (истина) и **`false`** (ложь).

В Java логический тип называется : **`boolean`**.

Этот тип назван в честь Джорджа Буля



Почему нельзя использовать целый тип вместо логического, обозначив 0 – ложь, а 1 – истина?

Объявление логической переменной

```
boolean a = true;
boolean b = false;
boolean c = 5 > 6;
boolean d = 1 <= 3;
```

false

true

Логические операторы – 1

Логические операции работают только с операндами типа **`boolean`**.

`&` – конъюнкция (И, логическое умножение)

`|` – дизъюнкция (ИЛИ)

`^` – исключающее ИЛИ (XOR)

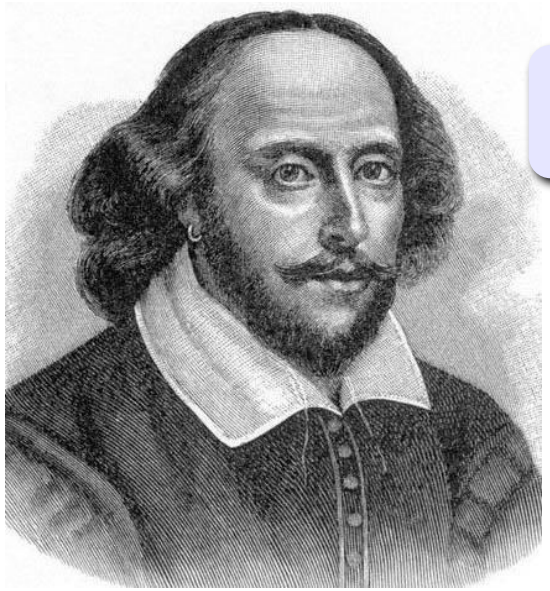
`||` – замыкающее ИЛИ

`&&` – замыкающее И

`!` – отрицание

Логические операторы – 2

A	B	A | B	A & B	A ^ B	!A
false	false	false	false	false	true
false	true	true	false	true	true
true	false	true	false	true	false
true	true	true	true	false	false



To be or not to be

```
2b | !2b
```

Операторы сравнения

Оператор	Описание	Пример	Значение
==	равно	1+1 == 2	true
!=	Не равно	3.2 != 2.5	true
<	Меньше	10 < 5	false
>	Больше	10 > 5	true
<=	Меньше или равно	126 <= 100	false
>=	Больше или равно	5.0 >= 5.0	true

Логические операторы. Пример

```
boolean a = true;
boolean b = false;
boolean c = a | b;
boolean d = a & b;
boolean e = a ^ b;
boolean f = (!a & b) | (a & !b);
boolean g = !a;

System.out.printf("a = %b", a);
```

true

false

true

true

false

Замыкающие логические операторы

A	B	A | B	A & B
false	false	false	false
false	true	true	false
true	false	true	false
true	true	true	true

```
true  |  x  = true
false &  x  = false
```

При использовании форм `||` и `&&` Java не будет вычислять значение второго операнда, если результат выражения можно определить по значению первого операнда.

Применение замыкающих операторов

Применяются, когда значение второго операнда зависит от значения первого.

```
int n = in.nextInt();
int del = in.nextInt();
boolean a = del != 0 & n / del > 10;
```

Если `del` = 0, то получим ошибку деление на 0.

Как исправить?

```
boolean a = del != 0 && n / del > 10;
```

Если `del`=0, то вторая часть выражения не будет вычисляться.

Закон де Моргана

Выражение	Отрицание выражения	Отрицание выражения
`a && b`	`!(a && b)`	`!a || !b`
`a || b`	`!(a || b)`	`!a && !b`

Пример.

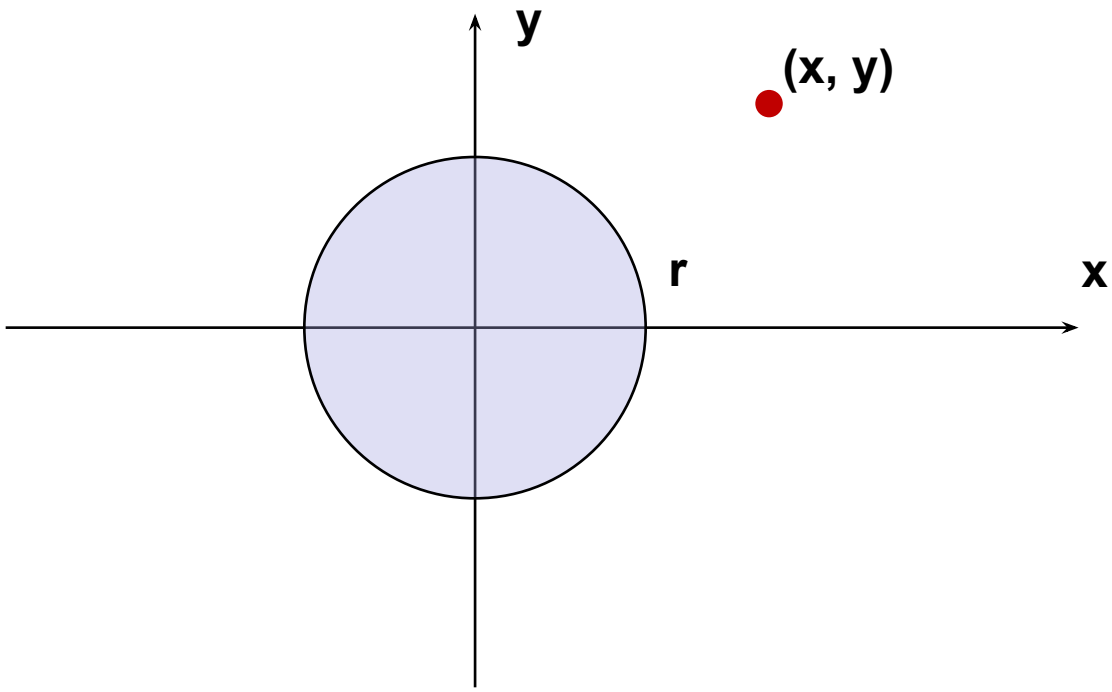
```
boolean z = !(x == 7 && y > 3);
z         = x != 7 || y <= 3;
```

Применение типа `boolean`

- Можно вычислить сложное логическое выражение и использовать в программе
- Использование логических переменных делает код более удобочитаемым

Задача

Задача. Вывести логическое значение **`true`**, если приведенное высказывание для предложенных исходных данных является истинным, и значение **`false`** в противном случае. Даны целые числа **`x`**, **`y`** и **`r`**, проверить попадает ли точка с координатами **`(x,y)`** в круг с центром в начале координат и радиусом **`r`**.



Задача

```
int x, y, r;
x = in.nextInt();
y = in.nextInt();
r = in.nextInt();
boolean result = x*x + y*y <= r*r;
System.out.printf("Ответ: %b", result);
```

Задача

Задача. Дано: прямоугольник со сторонами, параллельными осям координат, задан координатами абсцисс вертикальных сторон **(x1, x2)** и ординатами горизонтальных **(y1, y2)**; точка **M(x, y)**;
Найти: находится ли точка внутри прямоугольника, снаружи, или лежит на границе.

```
boolean inside, outside, bound;
inside = (x > x1) && (x < x2) &&
          (y > y1) && (y < y2);
outside = (x < x1) || (x > x2) ||
          (y < y1) || (y > y2);
bound = !inside && !outside;
```

Программирование на языке Java

Программирование на языке Java

Тема 12. Ветвления

Разветвляющиеся алгоритмы

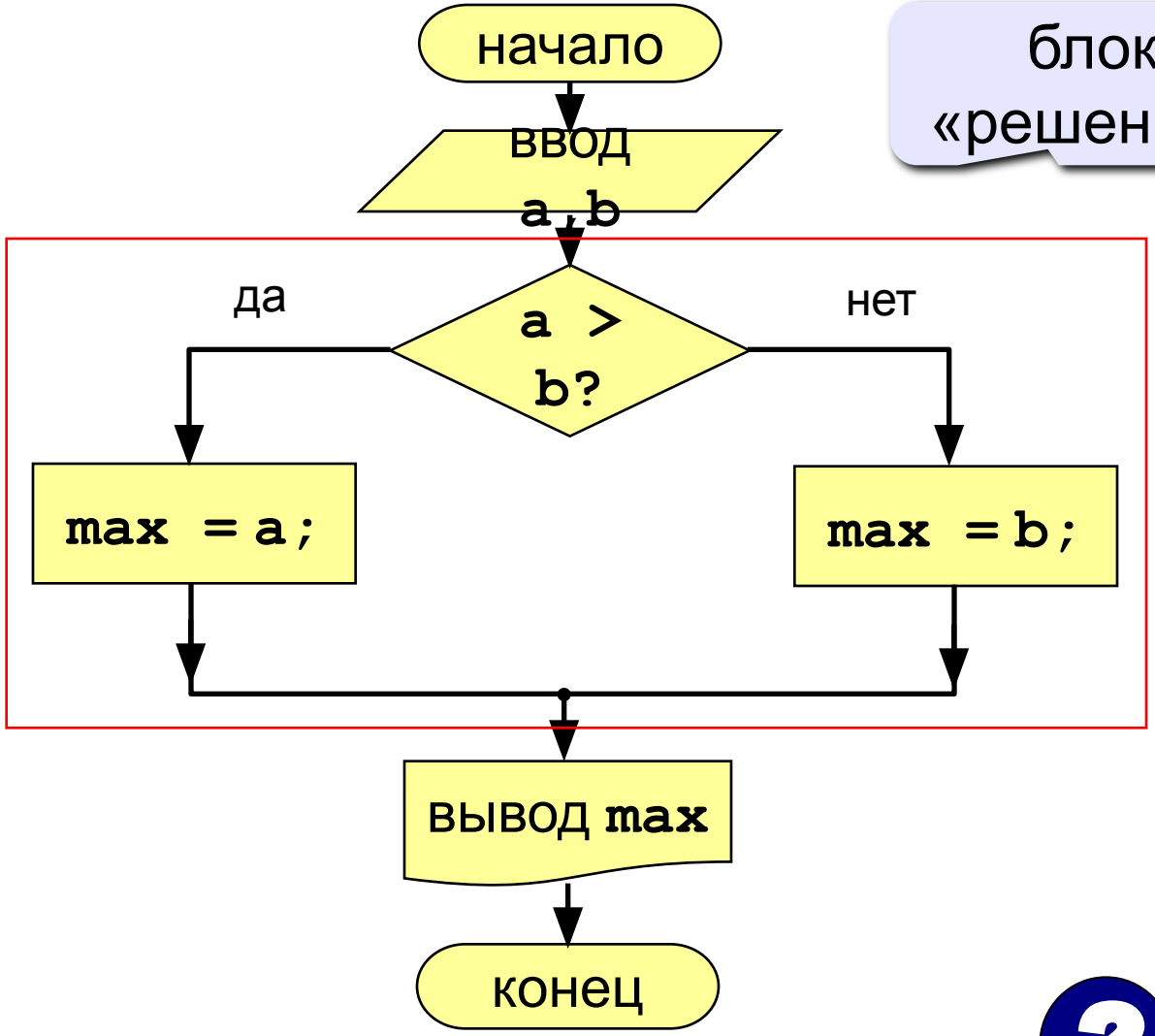
Задача. Ввести два целых числа и вывести на экран наибольшее из них.

Идея решения: надо вывести на экран первое число, если оно больше второго, или второе, если оно больше первого.

Особенность: действия исполнителя зависят от некоторых условий (***если … иначе …***).

Алгоритмы, в которых последовательность шагов зависит от выполнения некоторых условий, называются **разветвляющимися.**

Вариант 1. Блок-схема



блок «решение»

полная форма ветвления

? Если a = b?

Вариант 1. Программа

```
…
 int a, b, max;
a = in.nextInt();
b = in.nextInt();
if (a > b) {
    max = a;
  }
  else {
    max = b;
  }
   System.out.printf ("Наибольшее число %d",
 max);
```

полная форма условного оператора

Условный оператор

```
if (<условие>) {
   <что делать, если условие верно>
}
else {
   <что делать, если условие неверно>
}
```

Особенности:

- вторая часть (***else*** …) может отсутствовать (неполная форма)
- если в блоке один оператор, можно убрать фигурные скобки

Что неправильно?

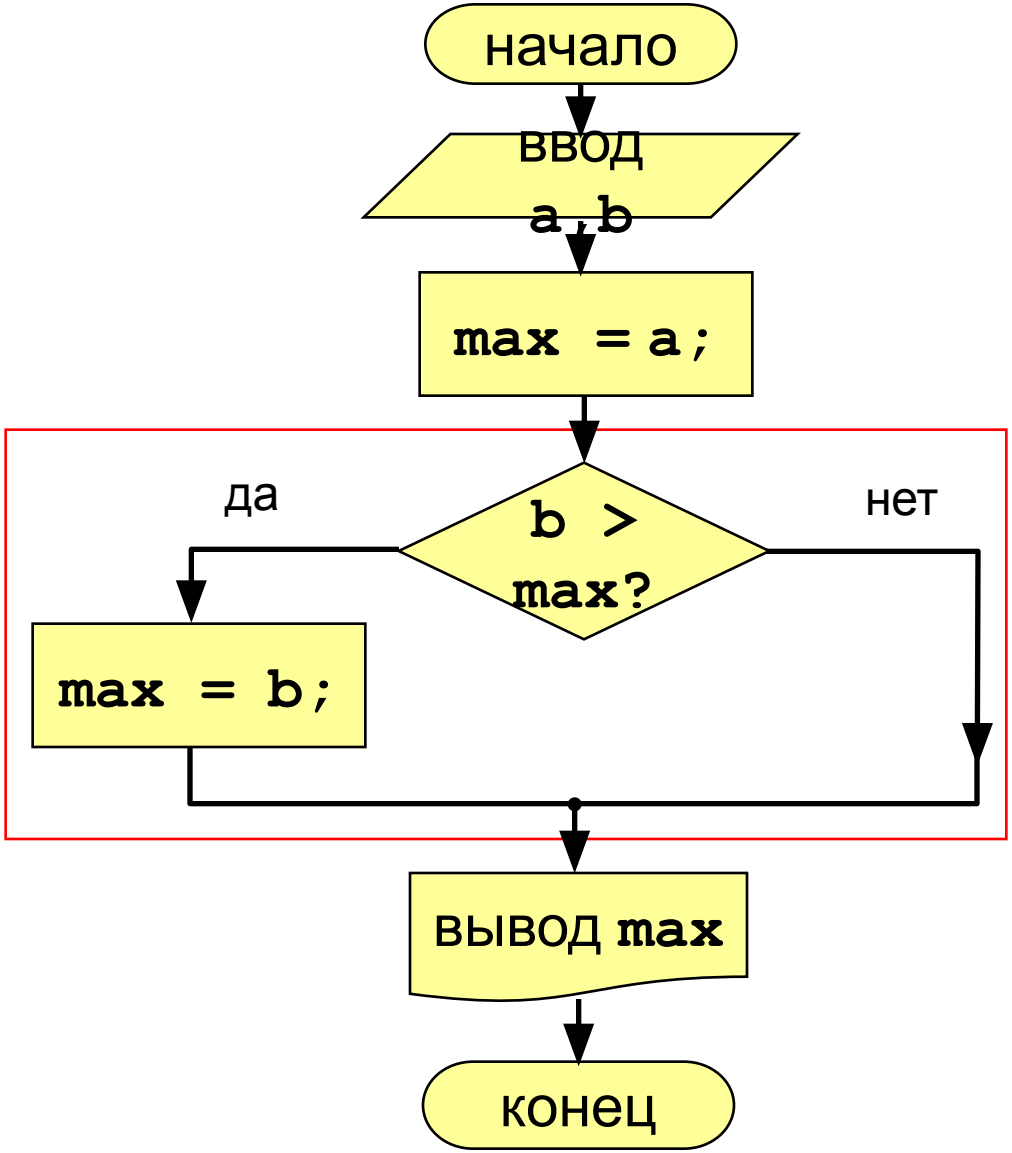
```
if (a > b) {
  a = b;
}
else {
  b = a;
}
```

```
if (a > b) {
  a = b; }
else {
  b = a;
}
```

```
if (a > b) {
  a = b;
}
else {
  b = a;
}
```

```
if (a > b) {
  a = b;
}
else {
  b = a;
}
```

Вариант 2. Блок-схема



неполная форма ветвления

Вариант 2. Программа

```
…
int a, b, max;
a = in.nextInt();
b = in.nextInt();
max = a;
    if (b > max)
     max = b;
System.out.printf ("Наибольшее число
 %d", max);
```

неполная форма условного оператора

Вариант 2Б. Программа

```
…
int a, b, max;
a = in.nextInt();
b = in.nextInt();
    max = b;
    if ( a > max )

        max = a;
System.out.printf ("Наибольшее число
 %d", max);
```

Что неправильно?

```
if (a > b)
      a = b;
else b = a;
```

```
if (a > b) {
   a = b;
}
else b = a;
```

```
if (a > b)
      a = b;
else b = a;
```

```
if (b >= a)
   b = a;
```

Ветвление и логические переменные

Условие может быть представлено переменной логического типа (**boolean**).

```
boolean test;
// ...
if (test) {
    <что делать, если test = true>
 }
 else {
    <что делать, если test = false>
}
```

Тернарный оператор ?

Некоторые небольшие конструкции **if-else** можно заменить специальным тернарным оператором.

Общий вид тернарного оператора:

```
<логическое выражение> ? <выражение1> :
 <выражение2>
```

Алгоритм работы:

1. Вычисляется логическое выражение.
2. Если логическое выражение истинно, то вычисляется значение выражения выражение 1, в противном случае — значение выражения выражение 2.
3. Вычисленное значение возвращается.

Тернарный оператор. Пример

Задача. Дано две переменных, найти наименьшую из них.

```
int a, b, min;
// ввод данных
min = (a < b) ? a : b;
```

Задания

1. Ввести три числа и найти наибольшее из них.

Пример:

```
Введите три числа:
4   15   9
Наибольшее число 15
```

2. Ввести пять чисел и найти наибольшее из них.

Пример:

```
Введите пять чисел:
4    15    9    56    4
Наибольшее число 56
```

Задания

3. Ввести числа. Упорядочить их по возрастанию.

Пример:

```
Введите два числа:
7   4
Упорядоченная последовательность: 4   7
```

4. Вычисление функции по взаимноисключающим веткам

$$y = \begin{cases} x, & x < 2 \\ x^2, & 2 < x < 3 \\ 1 - x, & x \geq 3 \end{cases}$$

Задания

5. Ввести три числа и найти среднее из них при условии, что числа попарно не равны.

Пример:

```
Введите три числа:
4   15   9
Среднее число 9
```

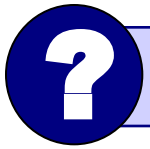
Программирование на языке Java

Тема 13. Сложные условия

Сложные условия

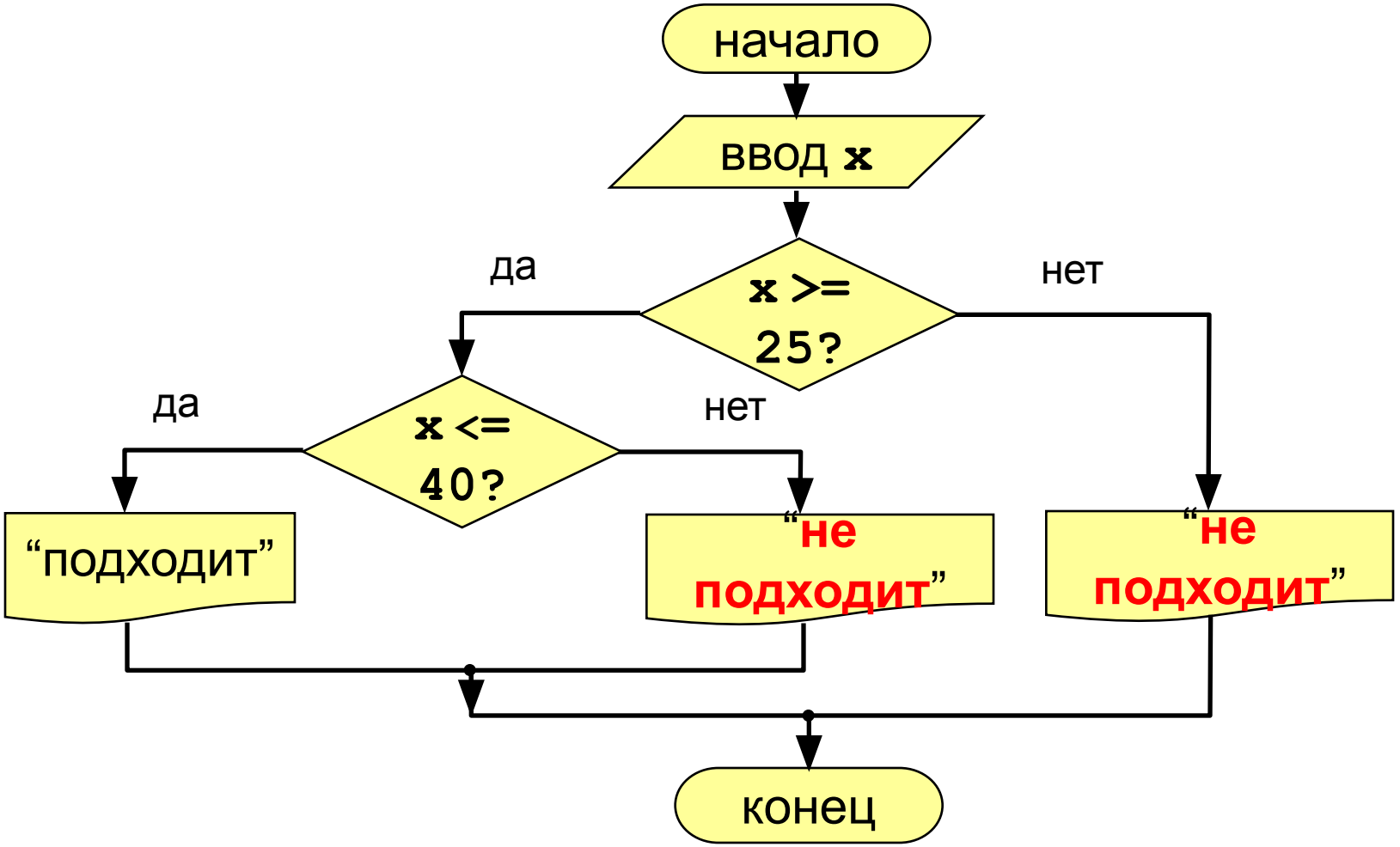
Задача. Фирма набирает сотрудников от 25 до 40 лет включительно. Ввести возраст человека и определить, подходит ли он фирме (вывести ответ «подходит» или «не подходит»).

Особенность: надо проверить, выполняются ли два условия одновременно.



Можно ли решить известными методами?

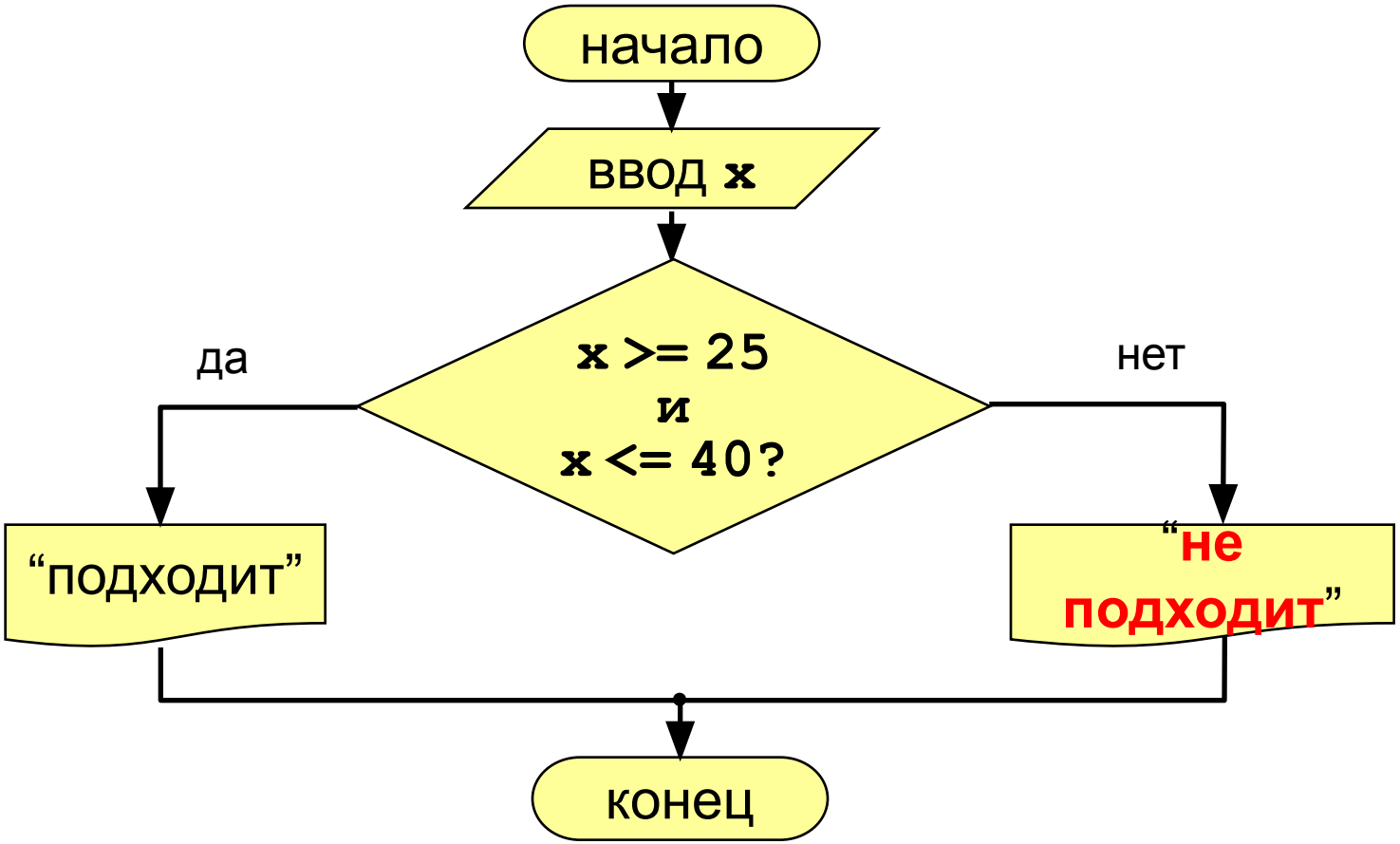
Вариант 1. Алгоритм



Вариант 1. Программа

```
…
 int x = in.nextInt();
if (x >= 25) {
  if (x <= 40)
     System.out.print(“Подходит”);
  else
     System.out.print(“Не подходит”);
  }
else
System.out.print(“Не подходит”);
```

Вариант 2. Алгоритм



Вариант 2. Программа

```
…
 int x;
 x = in.nextInt();

if (x >= 25 && x <= 40)
   System.out.println (“Подходит”);
else System.out.println (“Не
 подходит”);
}
```

сложное условие

Сложные условия

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью **логических операций**:

- **!** – НЕ (отрицание, инверсия)
- **&&** – И (логическое умножение, конъюнкция, одновременное выполнение условий)
- || – ИЛИ (логическое сложение, дизъюнкция, выполнение хотя бы одного из условий)
- ^ – исключающее или (XOR)

Простые условия (отношения)

< <= > >= == !=

равно

не равно

Сложные условия

Порядок выполнения (приоритет = старшинство)

- выражения в скобках
- `!` (НЕ, отрицание)
- `&` (И)
- `^` (XOR, исключающее ИЛИ)
- `|` (ИЛИ)
- `&&` (замыкающее И)
- `||` (замыкающее ИЛИ)

Пример

```
      2    1      6     3     5     4
if ( !(a > b) || c != d && b == a)
```

Сложные условия

Истинно или ложно при `a = 2; b = 3; c = 4;`

```
!(a > b)                          true
(a < b) && (b < c)                true
!(a >= b) || (c == d)             true
(a < c) || (b < c) && (b < a)     true
!(a < b) && (b > c)               false
```

Для каких значений `x` истинны условия:

```
(x < 6) && (x < 10)
(x < 6) && (x > 10)
(x > 6) && (x < 10)
(x > 6) && (x > 10)
(x < 6) || (x < 10)
(x < 6) || (x > 10)
(x > 6) || (x < 10)
(x > 6) || (x > 10)
```

$(-\infty, 6)$	$x < 6$
$\varnothing$	
$(6, 10)$	
$(10, \infty)$	$x > 10$
$(-\infty, 10)$	$x < 10$
$(-\infty, 6) \cup (10, \infty)$	
$(-\infty, \infty)$	
$(6, \infty)$	$x > 6$

Задания

1. Ввести номер месяца и вывести название времени года.

   Пример:

```
Введите номер месяца:
4
весна
```

2. Ввести возраст человека (от 1 до 150 лет) и вывести его вместе с последующим словом «год», «года» или «лет».

   Пример:

```
Введите возраст:
24
Вам 24 года
```

```
Введите возраст:
57
Вам 57 лет
```

Задания

3. Ввести день, месяц и год рождения, а также текущие день, месяц и год. Выведите на экран количество полных лет на текущую дату.

Пример:

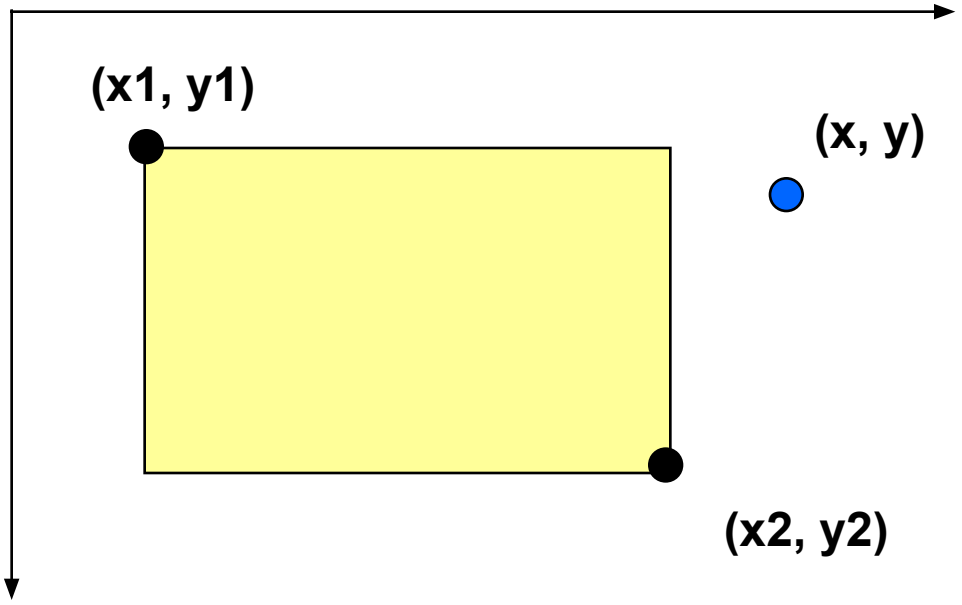
```
Введите дату рождения:
30.11.1996
Введите текущую дату:
29.10.2013
Ответ: вам 16 лет.
```

Пример:

```
Введите дату рождения:
30.09.1996
Введите текущую дату:
29.10.2013
Ответ: вам 17 лет.
```

Задания

4. Дан прямоугольник со сторонами, параллельными осям координат, он задан координатами абсцисс вертикальных сторон (x1, x2) и ординатами горизонтальных (y1, y2); точка M(x, y);
Найти: находится ли точка внутри прямоугольника, снаружи, или лежит на границе.



Пользователь может ввести некорректные начальные значения, например, x1 > x2

Программирование на языке Java

20. Оператор выбора

Программирование на языке Java

Тема 20. Оператор выбора

Оператор выбора

Задача: Ввести номер месяца и вывести количество дней в этом месяце для невисокосного года.

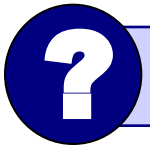
Решение: Число дней по месяцам:

28 дней – 2 (февраль)

30 дней – 4 (апрель), 6 (июнь), 9 (сентябрь), 11 (ноябрь)

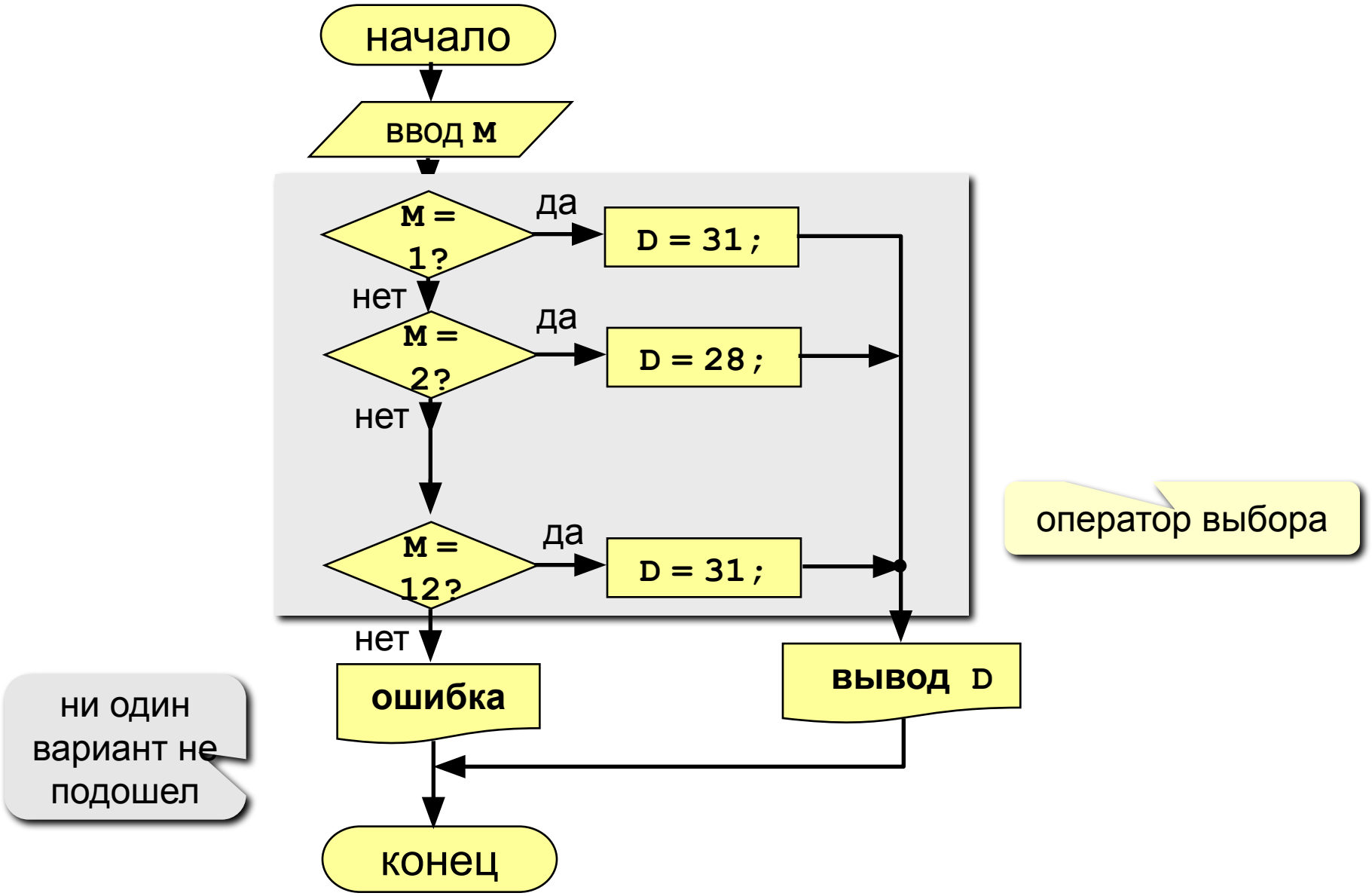
31 день – 1 (январь), 3 (март), 5 (май), 7 (июль), 8 (август), 10 (октябрь), 12 (декабрь)

Особенность: Выбор не из двух, а из нескольких вариантов в зависимости от номера месяца.



Можно ли решить известными методами?

Алгоритм


начало
ввод M
M = 1?
да
D = 31;
нет
M = 2?
да
D = 28;
нет
M = 12?
да
D = 31;
нет
ошибка
вывод D
конец
оператор выбора
ни один вариант не подошел

Оператор выбора

Оператор `switch` – оператор выбора в Java, который предоставляет простой способ направления потока выполнения команд по различным ветвям кода в зависимости от значения управляющего выражения.

Эффективнее, чем применение длинных последовательностей **`if-else`**.

Общая форма оператора switch

```
switch (выражение) {
  case значение1:
    // последовательность операторов
    break;
  case значение2:
    // последовательность операторов
    break;
  ...
  ...
  case значениеN:
    // последовательность операторов
    break;
  default:
    // последовательность операторов,
    // выполняемая по умолчанию
}
```

Константа

Необязательный оператор

Выполнение оператора выбора

1. Значение выражения сравнивается с каждым из значений констант в операторах **`case`**.
2. При совпадении, выполняется блок кода, следующего за данным оператором **`case`**.
3. Если значение ни одной из констант не совпало со значением выражения, программа выполняет оператор **`default`**.

Оператор **`break`** внутри последовательности **`switch`** служит для прерывания последовательности операторов. Как только программа доходит до **`break`**, она продолжает выполнение с первой строки кода, следующей за оператором **`switch`**.

Программа

```
…
 int M, D;
 M = in.nextInt();
switch ( M ) {
   case 2:  D = 28; break;
   case 4: case 6: case 9: case 11:
            D = 30; break;
   case 1: case 3: case 5: case 7:
   case 8: case 10: case 12:
            D = 31; break;
   default: D = -1;
   }
 if (D > 0)
     System.out.printf("В этом месяце %d дней.", D);
 else System.out.print("Неверный номер месяца.");
```

выйти из **switch**

ни один вариант не подошел

Особенности

Особенности:

- после **switch** может быть имя переменной или арифметическое выражение целого типа (**int**, **short**, **long**)

```
switch ( i+3 ) {
  case 1: a = b; break;
  case 2: a = c;
  }
```

или символьного типа (**char**)

- **нельзя** ставить два **одинаковых** значения:

```
switch ( x ) {
  case 1: a = b; break;
  case 1: a = c;
  }
```

Пример

```
int i = in.nextInt(); // положительное
switch(i) {
case 0:
case 1:
case 2:
case 3:
  System.out.println("i меньше 4");
  break;
case 4:
case 5:
case 6:
case 7:
  System.out.println("i меньше 8");
  break;
default:
  System.out.println ("i равно или больше 8");
```

Вложенные операторы `switch`

Оператор **`switch`** можно использовать в последовательности операторов внешнего оператора **`switch`**.

Такой оператор называют **вложенным оператором `switch`**.

Поскольку оператор **`switch`** определяет собственный блок, каких-либо конфликтов между константами **`case`** внутреннего и внешнего операторов **`switch`** не происходит.

Вложенные switch. Пример

```
switch(count) {
  case 1:
    switch(target) {
      case 0:
        System.out.println("target
        break;
      case 1:
        System.out.println("target равна 1");
        break;
      }
  break;
  case 2:
   ...
```

Вложенный оператор **switch**

Конфликты с внешним оператором **switch** отсутствуют

Итоги

1. Оператор **`switch`** отличается от оператора **`if`** тем, что он может выполнять проверку только равенства, в то время как оператор **`if`** может вычислять булевское выражение для любых типов. То есть оператор **`switch`** ищет только соответствие между значением выражения и одной из констант **`case`**.
2. Никакие две константы **`case`** в одном и том же операторе **`switch`** не могут иметь одинаковые значения.
3. Как правило, оператор **`switch`** эффективнее набора вложенных операторов **`if`**.

Задания

1. **Ввести номер месяца и вывести его название.**

   Пример:

```
Введите номер месяца:
-2
Такого месяца не существует
```

```
Введите номер месяца:
2
Февраль
```

2. **Ввести номер месяца и номер дня, вывести число дней, оставшихся до Нового года.**

   Пример:

```
Введите номер месяца:
12
Введите день:
25
До Нового года осталось 6 дней.
```

Программирование на языке Java

14. Циклы с известным числом шагов

Программирование на языке Java

Тема 15. Циклы с известным числом шагов

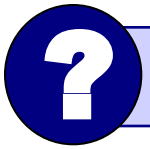
Циклы

Цикл – это многократное выполнение одинаковой последовательности действий.

- цикл с **известным** числом шагов
- цикл с **неизвестным** числом шагов (цикл с условием)

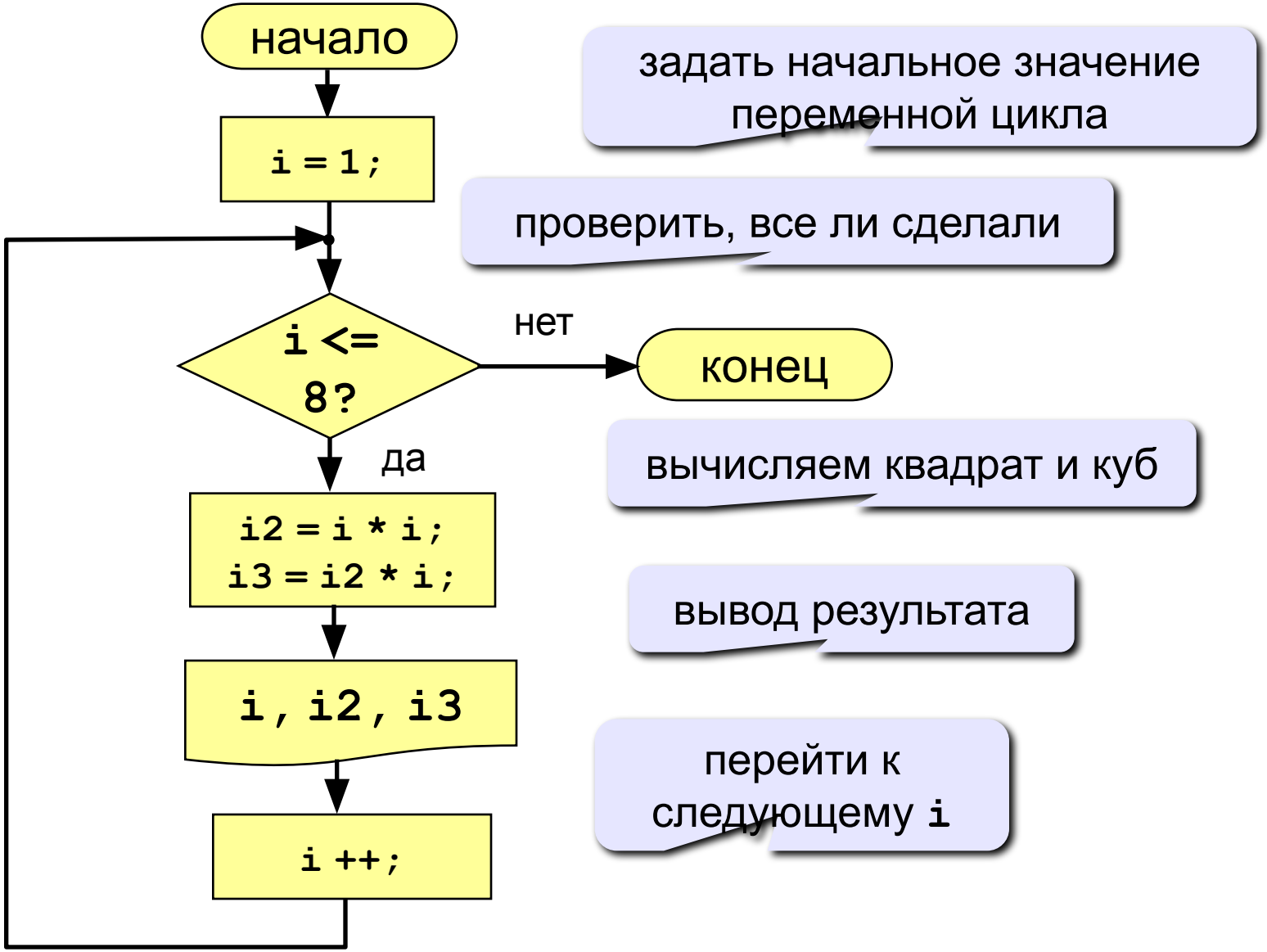
Задача. Вывести на экран квадраты и кубы целых чисел от 1 до 8 (от **`a`** до **`b`**).

Особенность: одинаковые действия выполняются 8 раз.

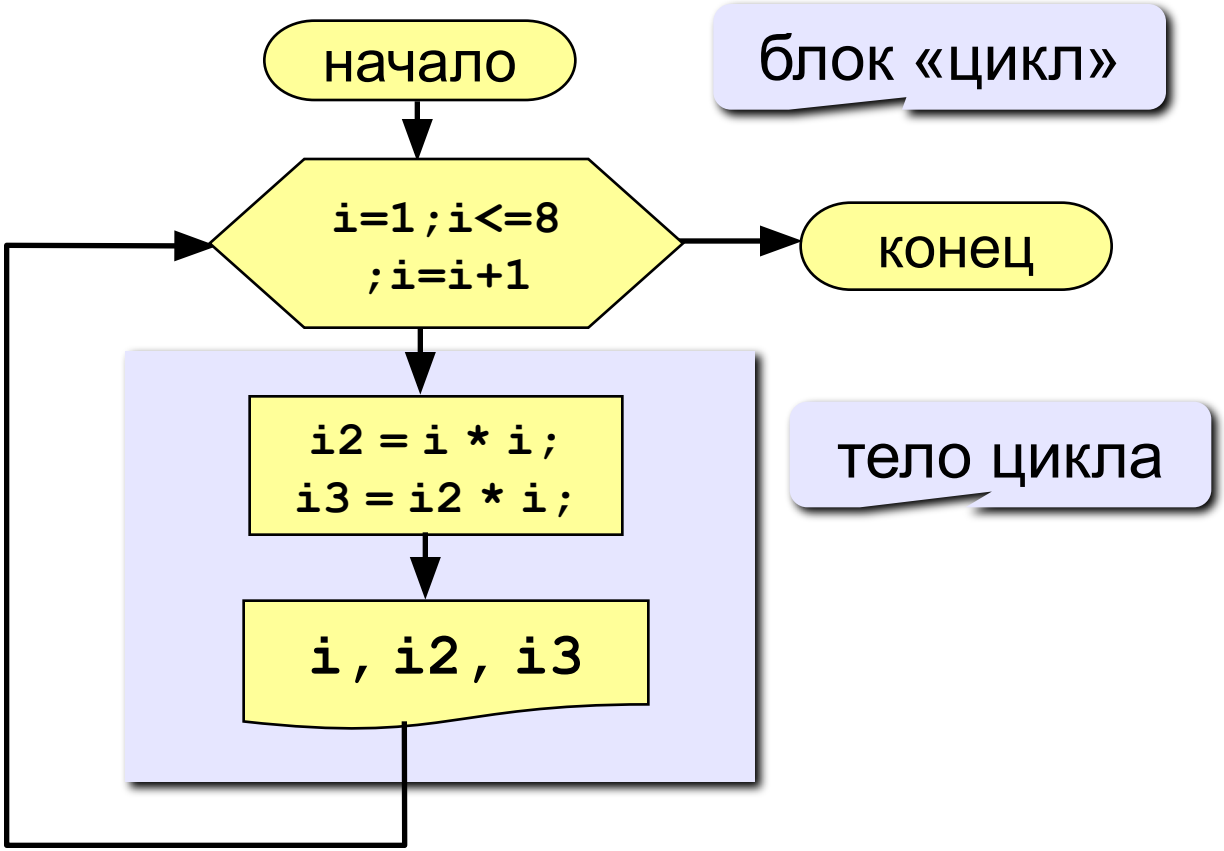


Можно ли решить известными методами?

Алгоритм


начало
i = 1;
задать начальное значение переменной цикла
проверить, все ли сделали
i <= 8?
нет
конец
да
вычисляем квадрат и куб
i2 = i * i;
i3 = i2 * i;
вывод результата
i, i2, i3
перейти к следующему i
i ++;

Алгоритм (с блоком «цикл»)



Программа

```
public static void main(String args[])
{
int i, i2, i3;
for (i=1; i<=8; i++)
  {
  i2 = i*i;
  i3 = i2*i;
  System.out.printf("%d %d %d",i,i2,i3);
  }
}
```

начальное значение

заголовок цикла

условие нахождения в цикле

цикл

циклическая переменная

шаг цикла i=i+1

начало цикла

тело цикла

конец цикла

Цикл с уменьшением переменной

Задача. Вывести на экран квадраты и кубы целых чисел от 8 до 1 (в обратном порядке).

Особенность: переменная цикла должна уменьшаться.

Решение:

```
for (i=8; i>=1 ; i-- )
  {
    i2 = i*i;
    i3 = i2*i;
    System.out.printf( ... );
  }
```

Цикл с переменной

```
for (начальные значения;
      условие продолжения цикла;
      изменение на каждом шаге)
  {
  // тело цикла
  }
```

Примеры:

```
for (a=2; a<b; a+=2) { ... }
```

```
for (a=2, b=4; a<b; a+=2) { ... }
```

```
for (a=1; c<d; x++) { ... }
```

```
for (; c<d; x++) { ... }
```

```
for (; c<d; ) { ... }
```

Цикл с переменной

Особенности:

- ***условие*** проверяется в начале очередного шага цикла, если оно ложно цикл не выполняется;
- ***изменения*** (третья часть в заголовке) выполняются в конце очередного шага цикла;
- если ***условие*** никогда не станет ложным, цикл может продолжаться бесконечно (**зацикливание**)

```
for(i=1; i<8; i++) { i--; }
```

Не рекомендуется изменять значение циклической переменной в теле цикла!

- если в теле цикла только один оператор, **скобки** {} можно не ставить:

```
for(i=1; i<8; i++) a+=b;
```

Цикл с переменной

Особенности:

- после выполнения цикла **во многих системах** устанавливается первое значение переменной цикла, при котором нарушено условие:

```
for (i=1; i<=8; i++)
    System.out.print("Привет");
System.out.print(i);
```

i=9

```
for (i=8; i>=1; i--)
    System.out.print("Привет");
System.out.print(i);
```

i=0

Использование циклов

Советы:

- При написании программ используйте **форматирование «лесенкой».**

```
for (i=1; i<=8; i++) {
System.out.print("Привет");
System.out.print(i);
}
```

В NetBeans:
Ctrl + Shift + F

```
for (i=1; i<=8; i++) {
    System.out.print("Привет");
    System.out.print(i);
}
```

- Используйте **отладку программ** для поиска логических ошибок в программах.

Сколько раз выполняется цикл?

```
a=1;
for(i=1; i<4; i++) a++;
```

a = 4

```
a=1;
for(i=1; i<4; i++) a=a+i;
```

a = 7

```
a=1; b=2;
for(i=3; i>=1; i--)a+=b;
```

a = 7

```
a=1;
for(i=1; i>=3; i--)a=a+1;
```

a = 1

```
a=1;
for(i=1; i<=4; i--)a++;
```

зацикливание

Как изменить шаг?

Задача. Вывести на экран квадраты и кубы нечётных целых чисел от 1 до 9.

Особенность: переменная цикла должна увеличиваться на 2.

Решение:

```
for (i=1; i<=9;i++) {
    if( i % 2 == 1 ) {
        i2 = i*i;
        i3 = i2*i;
        System.out.printf( ... );
    }
}
```

выполняется только для нечетных **i**

Что плохо?

Как изменить шаг? – II

Идея: Надо вывести всего 5 чисел, переменная **i** изменяется от 1 до 9, с каждым шагом цикла **i** увеличивается на 2.

Решение:

```
for (i=1;i<=9; i = i+2 )     {
   i2 = i*i;
   i3 = i2*i;
   System.out.printf( ... );
   }
```

Как изменить шаг? – III

Идея: Надо вывести всего 5 чисел, переменная **k** изменяется от 1 до 5. **Зная k, надо рассчитать i.**

k	1	2	3	4	5
i	1	3	5	7	9

$i = 2k-1$

Решение:

```
for (k=1; k<=5; k++) {
    i = 2*k - 1;
    i2 = i*i;
    i3 = i2*i;
    System.out.printf( ... );
}
```

Задания

1. Ввести a и b и вывести квадраты и кубы чисел от a до b.

   Пример:

```
Введите границы интервала:
4 6
   4   16   64
   5   25  125
   6   36  216
```

2. Вывести квадраты и кубы 10 чисел следующей последовательности: 1, 2, 4, 7, 11, 16, …

   Пример:

```
   1     1     1
   2     4     8
   4    16    64
        ...
  46  2116 97336
```

Задания

3. Ввести на экран таблицу умножения.

Прием накопления суммы

Задача. Просуммировать целые числа от 1 до 100.

Идея: переменной, в которую записывается сумма, присвоим значение 0. В цикле на каждом шаге прибавим к этой переменной очередное число.

Решение:

```
S = 0;
for (i=1; i<=100;i++)
   S = S + i;
System.out.print(S);
```

Обнуление переменной

Прибавление очередного элемента суммы

Буратино подарили три яблока. Два он съел. Сколько яблок осталось у Буратино?

Неизвестно, сколько осталось, так как не сказано, сколько яблок было у него до того, как ему подарили три новых.

Мораль: не забывайте обнулить переменные.

Прием накопления произведения

Задача. Вычислить факториал числа n.

Факториалом целого числа n называется произведение всех целых чисел от 1 до n. Обозначается n!

n! = 1*2*3*…*n

Идея: переменной, в которую записывается произведение, присвоим значение 1. В цикле на каждом шаге умножим эту переменную на очередное число.

Решение:

```
P = 1;
for (i = 2; i <= n; i++)
   P = P * i;
System.out.print(P);
```

Комбинация обоих приемов – 1

Задача. Вычислить значение выражения 1!+2!+3!+…+n!

Идея: в теле цикла, осуществляющего суммирование, производить вычисление факториала:

```
s = 0;
  for (i=1; i<=n; i++) {
   p = 1;
     for (k=1; k<=i; k++)
      p =p*k;
     s = s + p;
  }
```

Вложенный цикл

Что плохо?

Комбинация обоих приемов – 2

Задача. Вычислить значение выражения 1!+2!+3!+…+n!

Идея: при вычислении факториала на каждом шаге получается факториал все большего целого числа. Эти «промежуточные» результаты однократного вычисления факториала и можно суммировать

```
s = 0; p = 1;
  for (i=1; i<=n; i++) {
   p = p * i;
      s = s + p;
  }
```

Вычисляем очередное значение

Прибавляем его к сумме

Задания

1. **Найдите сумму нечетных чисел от 1 до N.**

   Пример:

   ```
   Введите N:
   10
   Сумма равна 25
   ```

2. **Напишите программу, вычисляющую значение выражения** $1 + x + x^2 + x^3 + \ldots + x^{10}$.

   Пример:

   ```
   Введите x:
   2
   Сумма равна 2047
   ```

Рекуррентные соотношения

Зачастую результат вычислений на каждом шаге цикла должен зависеть от результата вычислений на предыдущем шаге. Обобщенным математическим выражением этой идеи являются **рекуррентные соотношения**.

Задача: задано рекуррентное соотношение $x_{n+1} = 2 - x_n^2$

начальное значение $x_0 = 0$. Найдите x_5

```
s = 0;
  for (i=1; i<=5; i++) {
    s = 2 - s * s;
  }
```

Последовательности

Примеры:

- 1, 2, 3, 4, 5, …
- 1, 2, 4, 7, 11, 16, …
- 1, 2, 4, 8, 16, 32, …
- $\frac{1}{2}, \frac{1}{2}, \frac{3}{8}, \frac{1}{4}, \frac{5}{32}, \ldots$ $\quad \frac{1}{2}, \frac{2}{4}, \frac{3}{8}, \frac{4}{16}, \frac{5}{32}, \ldots$

$a_n = n$

$a_1 = 1,\ a_{n+1} = a_n + 1$

$a_1 = 1,\ a_{n+1} = a_n + n - 1$

$a_n = 2^{n-1}$

$a_1 = 1,\ a_{n+1} = 2a_n$

$$a_n = \frac{b_n}{c_n}$$

$b_1 = 1,\ b_{n+1} = b_n + 1$

$c_1 = 2,\ c_{n+1} = 2c_n$

Задания

Придумайте рекуррентные соотношения для последовательностей:

a) **0, 5, 10, 15, …**

b) **1, 1, 1, 1, …**

c) **1, -1, 1, -1, …**

d) **1, -2, 3, -4, 5, -6, …**

e) **2, 4, 16, 256, …**

f) **0, 1, 2, 3, 0, 1, 2, 3, 0, …**

g) **1!, 3!, 5!, 7!, …**

h) $1,\ a,\ a^2,\ a^3,\ a^4,\ \ldots$

i) $1,\ \frac{a}{1!},\ \frac{a^2}{2!},\ \frac{a^3}{3!},\ \ldots$

j) $1,\ 1+a,\ 1+a+a^2,\ 1+a+a^2+a^3,\ \ldots$

Перменные-флаги

Переменная флаг – это, как правило, переменная логического типа, значение которой сигнализирует о состоянии вычислительного процесса.

Задача: Пользователь вводит 10 чисел. Требуется проверить, упорядочены ли они по возрастанию, и передать эту информацию с помощью переменной флага.

```
int x, x2;
boolean isGrowing = true;
x = in.nextInt();
for (int i = 2; i <= 10; i++) {
     x2 = x;
   x = in.nextInt();
   isGrowing = isGrowing && (x > x2);
     }
```

Перменные-счетчики

Часто требуется подсчитать, сколько раз во время вычислений наступает то или иное событие.

Для этого вводится вспомогательная переменная, которой в начале присваивается нулевое значение, а после каждого наступления события она увеличивается на единицу. Такая переменная называется **счетчиком**.

Перменные-счетчики. Задача

Задача: Пользователь вводит 10 чисел. Определить, сколько из них являются одновременно чет положительными.

Обнуление счетчика

```
int x, counter = 0;
for (int i = 1; i <= 10; i++) {
   x = in.nextInt();
   if (x%2 == 0 && x > 0)
      counter++;
   }
System.out.println(counter);
```

Увеличение значения счетчика

Задания

1. Вводится число N и последовательность, состоящая из N целых чисел. Нужно найти максимальный элемент последовательности и вывести сколько раз он встречается.

   Пример:

```
Введите N:
10
Введите последовательность:
4 7 3 5 1 7 4 3 5 7
Максимальное значение: 7
Число вхождений: 3
```

Программирование на языке Java

15. Циклы с условием

Программирование на языке Java

Тема 16. Циклы с условием

Цикл с неизвестным числом шагов

Пример: Отпилить полено от бревна. Сколько раз надо сделать движения пилой?

Задача: Ввести целое число (<2000000) и определить число цифр в нем.

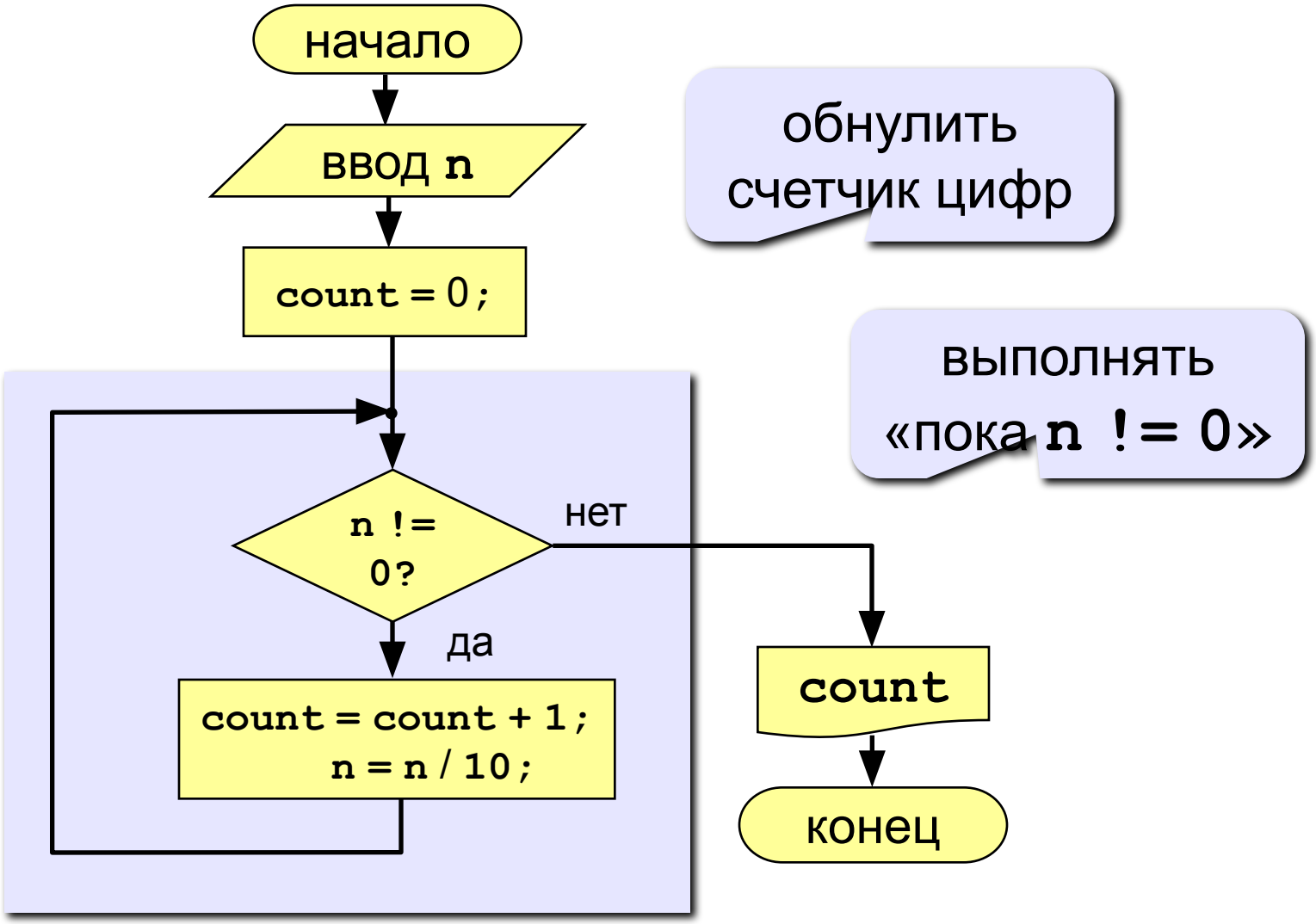
Идея решения: Отсекаем последовательно последнюю цифру, увеличиваем счетчик.

n	count
123	0
12	1
1	2
0	3

Проблема: Неизвестно, сколько шагов надо сделать.

Решение: Надо остановиться, когда `n = 0`, т.е. надо делать «пока `n != 0`».

Алгоритм


начало
ввод n
count = 0;
n != 0?
нет
да
count = count + 1;
n = n / 10;
count
конец
обнулить счетчик цифр
выполнять «пока n != 0»

Программа

```
int n, count;
System.out.print("Введите целое число");
n = in.nextInt();
count = 0;
  while (n != 0) {
     count = count + 1;
     n = n / 10;
   }
  System.out.printf("В числе %d нашли %d
 цифр%n", n, count);
}
```

ВЫПОЛНЯТЬ «пока n != 0»

Цикл с условием

```
while (<условие>) {
   тело цикла
}
```

Особенности:

- можно использовать сложные условия:

```
while (a<b && b<c) {
   тело цикла
}
```

- если в теле цикла только один оператор, фигурные скобки {} можно не писать:

```
while (a < b)
   a++;
```

Цикл с условием

Особенности:

- условие пересчитывается **каждый раз** при входе в цикл;
- если условие на входе в цикл ложно, цикл не выполняется ни разу:

```
a = 4; b = 6;
while (a > b)
   a = a - b;
```

- если условие никогда не станет ложным, программа **зацикливается**

```
a = 4; b = 6;
while (a < b)
   d = a - b;
```

Сколько раз выполняется цикл?

```
a = 4; b = 6;
while (a < b) a = a + 1;
```

2 раза
a = 6

```
a = 4; b = 6;
while (a < b) a = a + b;
```

1 раз
a = 10

```
a = 4; b = 6;
while (a > b) a = a + 1;
```

0 раз
a = 4

```
a = 4; b = 6;
while (a < b) b = a - b;
```

1 раз
b = -2

```
a = 4; b = 6;
while (a < b) a = a - 1;
```

зацикливание

Распространенная ошибка

```
int i=1;
while (i<10)   {
   println(i);
}
```

зацикливание: программист забыл добавить увеличение счетчика

Замена for на while и наоборот

```
for (i=1;i<=10;i++)
{тело цикла}
```

```
i = 1;
while (i<= 10)
{тело цикла
 i++;}
```

```
for (i=a;i>= b;i--)
{тело цикла}
```

```
i = a;
while (i >= b)
{тело цикла
  i--; }
```

В Java замена цикла for на while возможна всегда.

Задания

1. Напечатайте все нечетные числа от 3 до 25.

2. Ввести целое число и найти сумму его цифр.

Пример:

```
Введите целое число:
1234
Сумма цифр числа 1234 равна 10.
```

3. Ввести целое число и определить, верно ли, что в его записи есть две одинаковые цифры.

Пример:

```
Введите целое число:
1234
Нет.
```

```
Введите целое число:
1224
Да.
```

Вычисление номера шага

Цикл с условием может использоваться и при определении номера шага.

Пример. Коммерсант, имея стартовый капитал **k** рублей, занялся торговлей, которая ежемесячно увеличивает капитал на **p** процентов. Через сколько лет он накопит сумму **s**, достаточную для покупки собственного магазина?

```
int p, n; double k, x, s;
// ввод данных
x=k;
n=0;
  while (x<s) {
    x=x*(1+p/100.);
    n++; }
printf("%d лет и %d месяцев", n/12, n%12);
```

Начальная сумма

Обнуляем счетчик шагов

Пока сумма меньше s

Пересчитываем сумму и увеличиваем счетчик

Вычисления с заданной точностью

При реализации многих численных методов точность вычислений зависит от числа шагов.

Иногда невозможно заранее определить за какое число шагов будет достигнута приемлемая точностью.

Пример. Синус можно разложить в так называемый ряд Тейлора:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Чем большее количество членов ряда будет просуммировано, тем точнее будет вычислен синус. Пусть требуется вычислить до 5-го знака после запятой. То есть приемлемая погрешность 10^{-5} .

Вычисления с заданной точностью

```
double eps = 1e-5;
double x = in.nextDouble();
double p = x, s = x;
int n = 2;
while (Math.abs(p) > eps) {
        p = -p * x * x / (n * (n + 1));
        s = s + p;
        n += 2;
}
System.out.printf("sin(x) = %f", s);
```

Вычисления с заданной точностью

При вычислении рекуррентных соотношений можно прекратить вычисления, если изменение вычисляемой величины на очередном шаге меньше заданной величины:

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

Последовательности

Примеры:

- 1, 2, 3, 4, 5, …
- 1, 2, 4, 7, 11, 16, …
- 1, 2, 4, 8, 16, 32, …
- $\frac{1}{2}, \frac{1}{2}, \frac{3}{8}, \frac{1}{4}, \frac{5}{32}, \ldots$ $\quad \frac{1}{2}, \frac{2}{4}, \frac{3}{8}, \frac{4}{16}, \frac{5}{32}, \ldots$

$a_n = n$

$a_1 = 1,\ a_{n+1} = a_n + 1$

$a_1 = 1,\ a_{n+1} = a_n + n - 1$

$a_n = 2^{n-1}$

$a_1 = 1,\ a_{n+1} = 2a_n$

$$a_n = \frac{b_n}{c_n}$$

$b_1 = 1,\ b_{n+1} = b_n + 1$

$c_1 = 2,\ c_{n+1} = 2c_n$

Последовательности

Задача: найти сумму всех элементов последовательности,

$$1,\ -\frac{1}{2},\ \frac{2}{4},\ -\frac{3}{8},\ \frac{4}{16},\ -\frac{5}{32},\ldots$$

которые по модулю больше 0,001:

$$S = 1 - \frac{1}{2} + \frac{2}{4} - \frac{3}{8} + \frac{4}{16} - \frac{5}{32} + \ldots$$

Элемент последовательности (начиная с №2):

$$a = z\,\frac{b}{c}$$

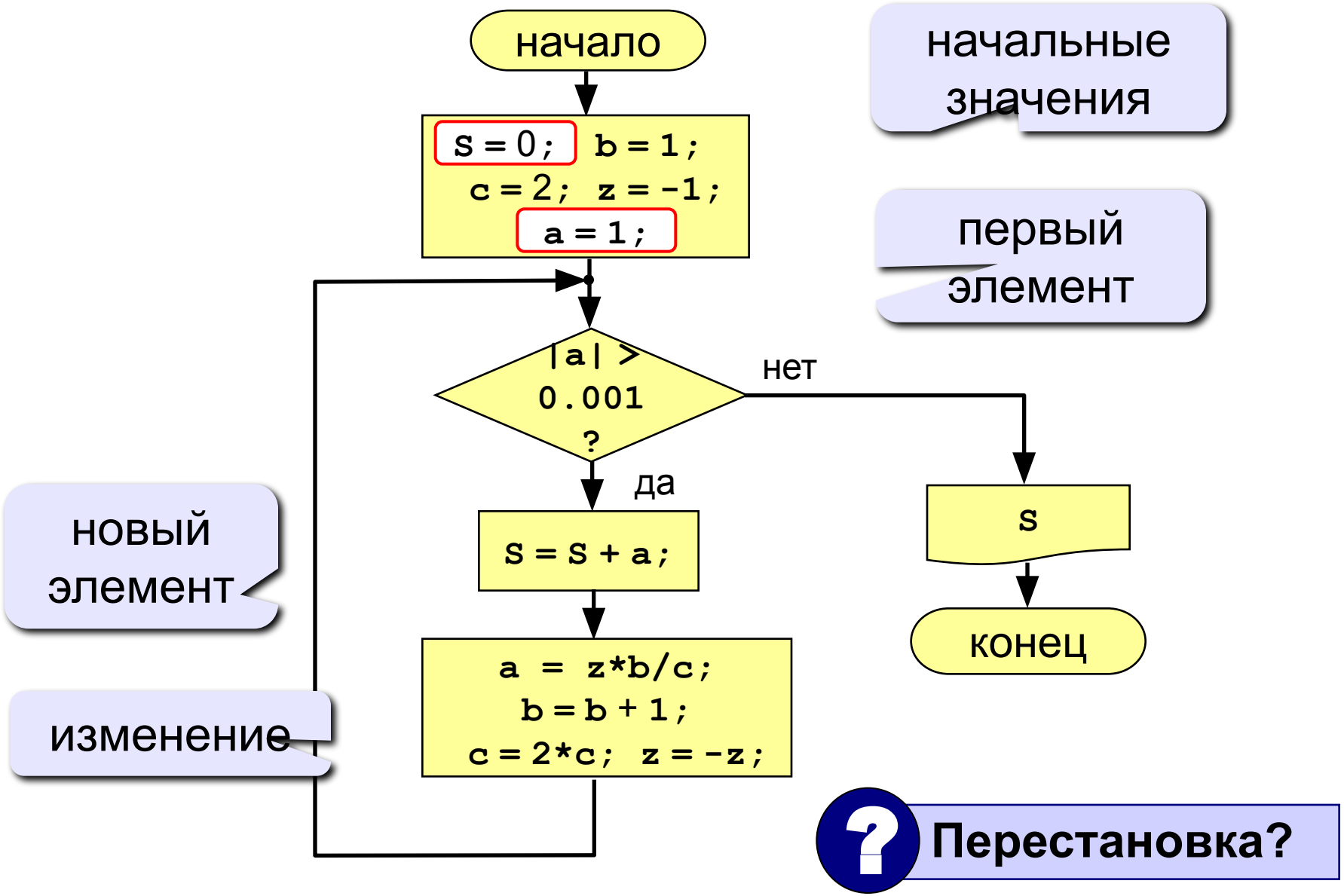
n	1	2	3	4	5	...
b	1	2	3	4	5	...
c	2	4	8	16	32	...
z	-1	1	-1	1	-1	...

```
b = b+1;
```

```
c = 2*c;
```

```
z = -z;
```

Алгоритм


начало
s = 0; b = 1;
c = 2; z = -1;
a = 1;
начальные значения
первый элемент
|a| > 0.001 ?
нет
да
s
конец
S = S + a;
новый элемент
a = z*b/c;
b = b + 1;
c = 2*c; z = -z;
изменение
Перестановка?

Программа

```
public static void                    rgs[])
{
  int b, c, z;
  float S, a, b;
  S = 0; z = -1;
  b = 1; c = 2; a = 1;
  while (a > 0.001 || a < -0.001) {
    S += a;
    a = z*b / c;
    z = -z;
    b++;
    c*=2;
    }
  System.out.printf ("S = %d%n", S);
}
```

чтобы не было округления при делении

значения

Выполняется пока |a|>0.001

увеличение суммы

расчет элемента последовательности

Что плохо?

Задания

1. Найти сумму элементов последовательности с точностью 0,001:

$$S = 1 + \frac{2}{3 \cdot 3} - \frac{4}{5 \cdot 9} + \frac{6}{7 \cdot 27} - \frac{8}{9 \cdot 81} + \ldots$$

Ответ:

```
S = 1.157
```

2. Найти сумму элементов последовательности с точностью 0,001:

$$S = 1 + \frac{2}{2 \cdot 3} - \frac{4}{3 \cdot 9} + \frac{6}{5 \cdot 27} - \frac{8}{8 \cdot 81} + \frac{10}{13 \cdot 243} - \ldots$$

Ответ:

```
S = 1.220
```

Цикл с постусловием

Задача: Ввести целое **положительное** число (<2000000) и определить число цифр в нем.

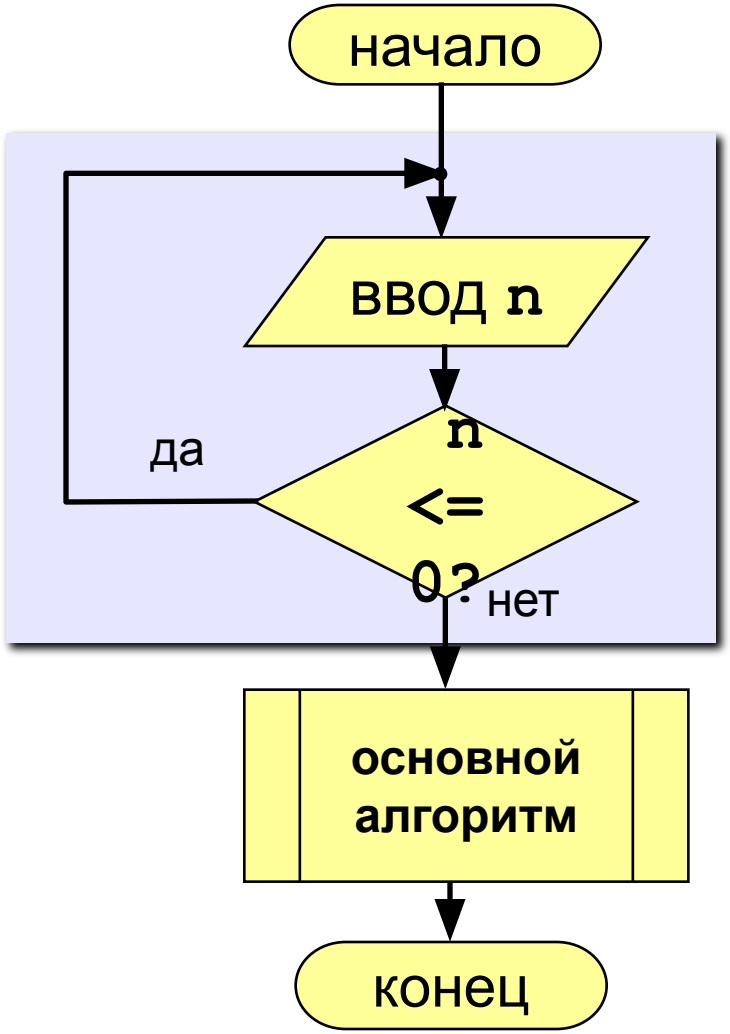
Проблема: Как не дать ввести отрицательное число или ноль?

Решение: Если вводится неверное число, вернуться назад к вводу данных (цикл!).

Особенность: Один раз тело цикла надо сделать в любом случае => проверку условия цикла надо делать в конце цикла (цикл с **постусловием**).

Цикл с постусловием – это цикл, в котором проверка условия выполняется в конце цикла.

Цикл с постусловием: алгоритм



тело цикла

условие **продолжения цикла**

блок «типовой процесс»

Программа

```
...
int n;
do {
    System.out.print("Введите положит.
число");
   while( n<=Int(); }
while  0); <= 0);

... { основной алгоритм }
```

условие **ПРОДОЛЖЕНИЯ ЦИКЛА**

Особенности:

- тело цикла всегда выполняется хотя бы один раз
- после слова **`while`** ставится условие **ПРОДОЛЖЕНИЯ** цикла

Сколько раз выполняется цикл?

```
a = 4; b = 6;
do { a++; } while (a<=b);
```

3 раза
a = 7

```
a = 4; b = 6;
do { a += b; } while ( a<=b );
```

1 раз
a = 10

```
a = 4; b = 6;
do { a += b; } while ( a>=b );
```

зацикливание

```
a = 4; b = 6;
do b=a-b; while ( a>=b );
```

2 раза
b = 6

```
a = 4; b = 6;
do a+=2; while ( a>=b );
```

зацикливание

Задания (с защитой от неверного ввода)

1. **Ввести натуральное число и определить, верно ли, что сумма его цифр равна 10.**

Пример:

```
Введите число >= 0:
-234
Нужно положительное число.
Введите число >= 0:
1234
Да
```

```
Введите число >= 0:
1233
Нет
```

2. **Ввести натуральное число и определить, какие цифры встречаются несколько раз.**

Пример:

```
Введите число >= 0:
2323
Повторяются: 2, 3
```

```
Введите число >= 0:
1234
Нет повторов.
```

Программирование на языке Java

Тема 21. Статические методы

Функции и библиотеки

Модульное программирование

- **Организация программы в виде независимых модулей, которые выполняют совместную работу.**
- **Почему? Проще делиться и повторно использовать код для создания больших программ.**
- **Возможность независимо разрабатывать отдельные части больших программ.**

Библиотека – набор функций. Обычно библиотека связана с какой-то предметной областью.

Алгоритм

Алгоритм – последовательность шагов для решения некоторой задачи.

Пример алгоритма изготовления печения:

- **Смешайте сухие ингредиенты;**
- **Взбейте сахар и масло;**
- **Добавьте во взбитую массу яйца;**
- **Добавьте сухие ингредиенты;**
- **Разогрейте духовку до температуры 180 градусов;**
- **Установите таймер на 10 минут;**
- **Разложите печенье на противень;**
- **Выпекайте печенье;**
- **Смешайте ингредиенты для украшения**
- **…**

Проблемы с алгоритмами

- **Недостаток структурированности – много шагов.**
- **Избыточность: рассмотрим алгоритм изготовления двух порций печенья**
  - **Смешайте сухие ингредиенты;**
  - **Взбейте сахар и масло;**
  - **Добавьте во взбитую массу яйца;**
  - **Добавьте сухие ингредиенты;**
  - **Разогрейте духовку до температуры 180 градусов;**
  - **Установите таймер на 10 минут;**
  - **Разложите печенье на противень;**
  - **Выпекайте печенье;**
  - **Разогрейте духовку до температуры 180 градусов;**
  - **Установите таймер на 10 минут;**
  - **Разложите печенье на противень;**
  - **Выпекайте печенье;**
  - **Смешайте ингредиенты для украшения**

Структурированный алгоритм

- **Структурированный алгоритм разбивает решение задачи на отдельные логические шаги:**

1. **Приготовление теста:**
   - **Смешайте сухие ингредиенты;**
   - **Взбейте сахар и масло;**
   - **Добавьте во взбитую массу яйца;**
   - **Добавьте сухие ингредиенты.**
2. **Выпечка:**
   - **Разогрейте духовку до температуры 180 градусов;**
   - **Установите таймер на 10 минут;**
   - **Разложите печенье на противень;**
   - **Выпекайте печенье;**
3. **Украшение печенья:**
   - **Смешайте ингредиенты для украшения**
   - **…**

Избавление от избыточности

Хорошо структурированный алгоритм может описывать повторяющиеся действия без избыточности.

1. **Приготовление теста:**
   - Смешайте сухие ингредиенты;
   - …
2. **Выпекание:**
   - Разогрейте духовку до температуры 180 градусов;
   - …
3. **Выпекание:**
   - Повторить шаг 2
4. **Украшение печенья:**
   - Смешайте ингредиенты для украшения
   - …

Статические методы

Статический метод – конструкция Java для создания вспомогательных алгоритмов, подпрограмм.

Во многих языках программирования статические методы называются функциями и процедурами.

Каждый статический метод — последовательность операторов, которые при вызове статического метода выполняются один за другим.

Слово статические отличает эти методы от методов экземпляров (будут рассмотрены позже в теме объектно-ориентированное

Статические методы

Метод может быть многократно вызван из разных частей программы.

Процедурная декомпозиция – представление разрабатываемой программы в виде совокупности вызывающих друг друга подпрограмм.

При вызове метода управление передается на соответствующий участок программного кода. После выполнения метода осуществляется возврат на оператор основной программы, следующий за вызовом метода.

Примеры методов

- **Встроенные методы Java, например `Math.random()`, `Math.abs()`, `Math.min()`.**
- **Методы чтения данных `in.nextInt()`, `in.nextDouble()`**
- **Методы, определенные программистом, например метод `main()`.**

Библиотека

- Создание статического метода это как добавление новой команды Java.
- Библиотека – набор функций (статических методов).

Объявление статического метода

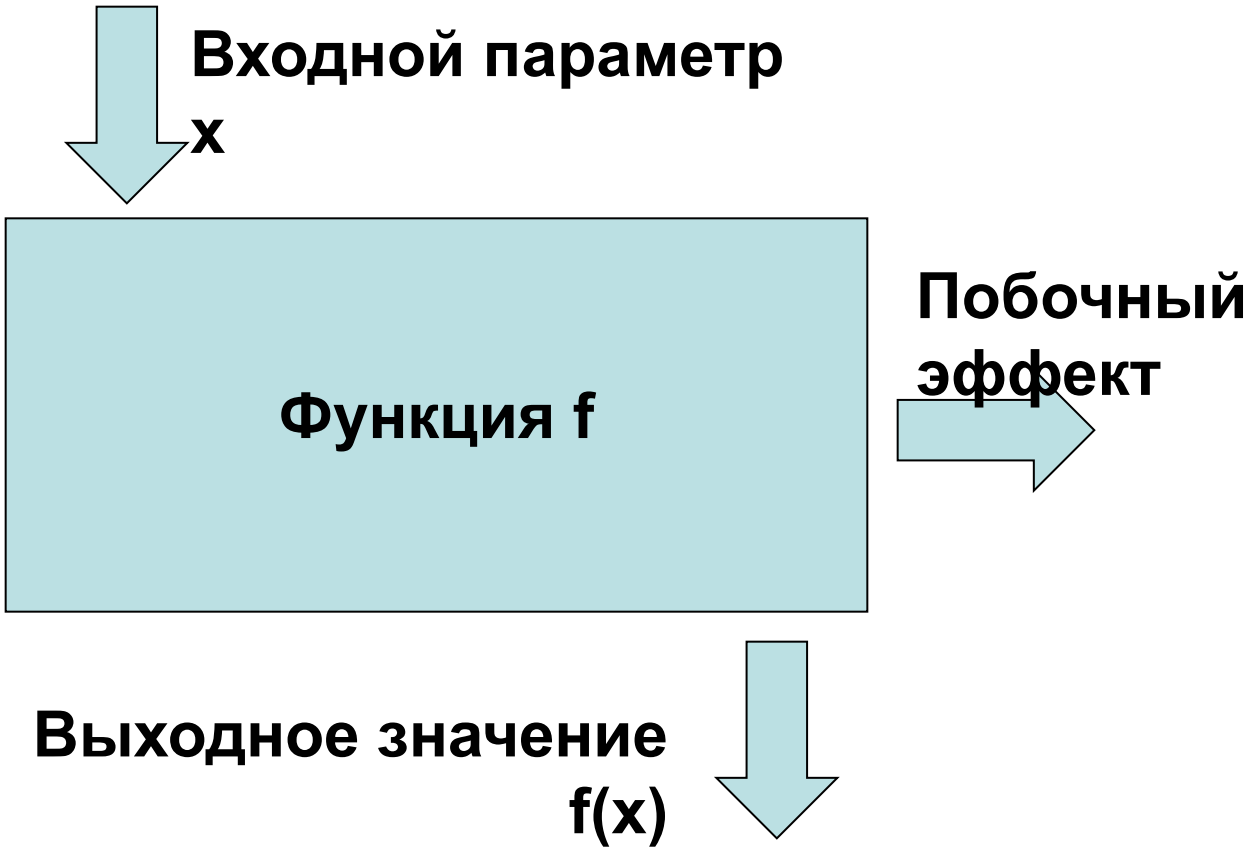
Метод содержит вычисление, которое определено как последовательность операторов.

Метод принимает **аргументы** (значения заданных типов данных, их может быть 0 и больше) и на основе этих аргументов:

- вычисляет **возвращаемое значение** определенного типа данных или
- вызывает **побочный эффект**, который зависит от аргументов (например, вывод значения).

Статический метод `main()` является примером метода второго типа (не возвращает

Функция (статический метод)



Определение статического метода

Каждый статический метод состоит из

сигнатуры –

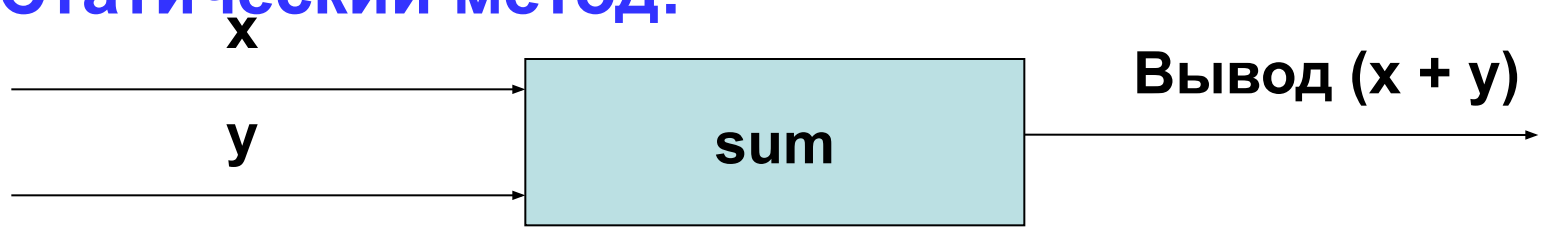
- ключевые слова `public static`,
- тип возвращаемого значения
- имя метода
- последовательность аргументов, каждый с объявленным типом в круглых скобках.

тела – последовательность операторов, заключенная в фигурные скобки.

Пример статического метода

Задача: составить метод, который вычисляет сумму двух значений выводит результат на экран.

Статический метод:




```
public static void sum (int x, int y) {
   int z = x + y;
   System.out.println(z);
}
```

Сигнатура метода

Тело метода

Описание метода, не возвращающего значение

Особенности:

- в сигнатуре ставится ключевое слово `void`, которое означает, что метод не возвращает значения

```
public static void sum (int x, int y)
```

- в сигнатуре описываются формальные параметры, они обозначаются именами с указанием типа параметра

```
public static void qq ( int a, float x )
```

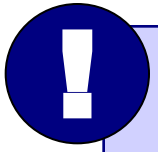
Описание метода, не возвращающего значение

Особенности:

- можно объявлять и использовать локальные переменные

```
public static void test(int a, int b)
{
  float x, y;
...
}
```

локальные переменные



Локальные переменные недоступны в основной программе и других методах

Вызов статического метода

Вызов статического метода – это его имя, за которым в скобках следуют выражения, задающие значения аргументов, разделенные запятыми.

При вызове метода его переменные аргументов инициализируются значениями соответствующих выражений из вызова.

Программа

```
public static void sum(int x,int y)
{
  ...
}
public static void main(String[] args)
{
 int a, b, c = 0;
 a = in.nextInt();
 b = in.nextInt();
sum (a,  b);
}
```

формальные параметры

фактические параметры

вызов метода

Использование статических методов

1. **Спроектировать (обдумать) алгоритм**
   - Посмотреть на структуру, отследить какие команды повторяются
   - Выделить основные части алгоритма
2. **Объявить методы (записать их в программе)**
   - Упорядочить выражения в группы, дать каждой группе имя.
3. **Вызвать методы**
   - Метод `main()` будет вызывать остальные методы для решения задачи.

Проектирование алгоритма

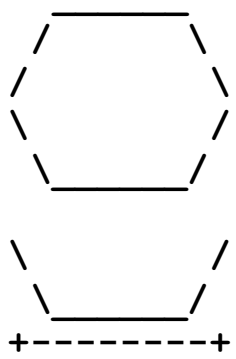
```
// Шаг 1: Приготовление теста
System.out.println("Смешайте сухие ингредиенты");
System.out.println("Взбейте сахар и масло");
System.out.println("Добавьте во взбитую массу яйца");
System.out.println("Добавьте сухие ингредиенты");

// Шаг 2а: Выпекание печенья (первый противень)
System.out.println("Разогрейте духовку");
System.out.println("Установите таймер на 10 минут");
System.out.println("Разложите печенье на противень");
System.out.println("Выпекайте печенье");

// Шаг 2б: Выпекание печенья (второй противень)
System.out.println("Разогрейте духовку ");
System.out.println("Установите таймер на 10 минут ");
System.out.println("Разложите печенье на противень");
System.out.println("Выпекайте печенье");


// Шаг 3: Украшение печенья
System.out.println("Смешайте ингредиенты для украшения");
System.out.println("Украсьте печенье");
```

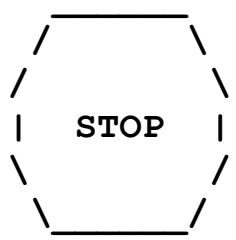
Итоговый алгоритм

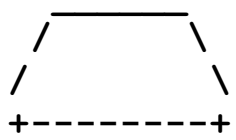
```
public static void main(String[] args) {
    makeBatter();
    bake();         // 1-ый противень
    bake();         // 2-ой противень
    decorate();
}

// Шаг 1: Приготовление теста
public static void makeBatter() {
    System.out.println("Смешайте сухие ингредиенты");
    System.out.println("Взбейте сахар и масло");
    System.out.println("Добавьте в массу яйца");
    System.out.println("Добавьте сухие ингредиенты");
}

// Шаг 2: Выпекание одного противня печенья
public static void bake() {
    System.out.println("Разогрейте духовку");
    System.out.println("Установите таймер на 10 минут");
    System.out.println("Разложите печенье на противень");
    System.out.println("Выпекайте печенье");
}

// Шаг 3: Украшение
public static void decorate() {
    System.out.println("Смешайте ингредиенты для украшения");
    System.out.println("Украсьте печенье");
}
```

Задача

Задача. Напишите программу, которая выводит на экран следующие фигуры.







Идея решения 1

Неструктурированная версия

- Создать метод `main`
- Скопировать ожидаемый вывод в программу, окружить оператором `System.out.println`

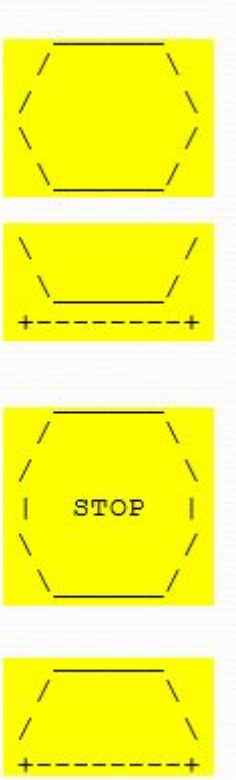
Вариант решения 1

```
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+--------+");
    }
}
```

Идея решения 2

Структурированная версия с избыточностью

– Выделить печать каждой фигуры в отдельный метод



Создадим методы:

- `printEgg`
- `printTeaCup`
- `printStopSign`
- `printHat`

Вариант решения 2

```
public class Figures2 {
    public static void main(String[] args) {
        printEgg();
        printTeaCup();
        printStopSign();
        printHat();
    }

    public static void printEgg() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
    }

    public static void printTeaCup() {
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println("+--------+");
        System.out.println();
    }
    ...
```

Вариант решения 2

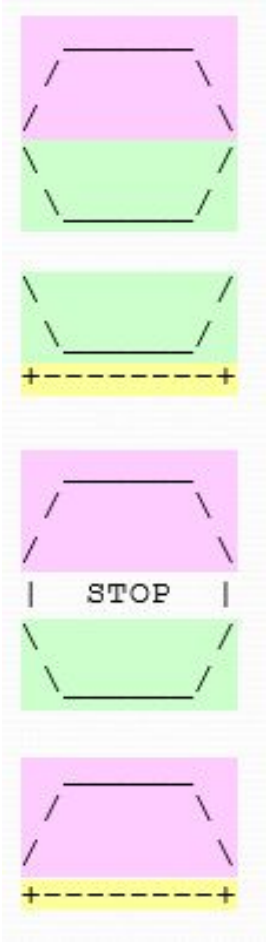
```
...

    public static void printStopSign() {
        System.out.println("        _______");
        System.out.println("  /       \\");
        System.out.println("/          \\");
        System.out.println("|  STOP    |");
        System.out.println("\\          /");
        System.out.println("  \\_____/");
        System.out.println();
    }

    public static void printHat() {
        System.out.println("        _______");
        System.out.println("  /       \\");
        System.out.println("/          \\");
        System.out.println("+--------+");
    }
}
```

Что плохо?

Идея решения 3

Структурированная версия без избыточности

– Выделить избыточность в выводе, создать методы без повторений



Создадим методы:

- `printEggTop` – используется в 3 фигурах
- `printEggBottom` – используется в 3 фигурах
- `printLine` – используется в 2 фигурах

Вариант решения 3

```
public class Figures3 {
    public static void main(String[] args) {
        printEgg();
        printTeaCup();
        printStopSign();
        printHat();
    }

    // Рисует верхнюю часть фигуры Яйцо
    public static void printEggTop() {
        System.out.println("      _______");
        System.out.println("  /~~~~~~~\\\");
        System.out.println("/           \\\");
    }

    // Рисует нижнюю часть фигуры Яйцо
    public static void printEggBottom() {
        System.out.println("\\           /");
        System.out.println(" \\_________/");
    }

    // Рисует фигуру Яйцо
    public static void printEgg() {
        printEggTop();
        printEggBottom();
        System.out.println();
    }

    ...
```

Вариант решения 3

```
...
    // Рисует фигуру Чашка
    public static void printTeaCup() {
        printEggBottom();
        printLine();
        System.out.println();
    }
    // Рисует знак Стоп
    public static void printStopSign() {
        printEggTop();
        System.out.println("|  STOP  |");
        printEggBottom();
        System.out.println();
    }
    // Рисует фигуру Шляпа
    public static void printHat() {
        printEggTop();
        printLine();
    }
    // Рисует линию
    public static void printLine() {
        System.out.println("+--------+");
    }
}
```

Процедурная декомпозиция (структурирование)

Правило. **Когда вы можете четко разделить задачи в своей программе – сделайте это.**

Глобальные и локальные переменные

Глобальные переменные описываются в классе (вне методов).

Локальные переменные создаются в теле метода, они существуют только в течение времени выполнения метода, определяются при его вызове и «исчезают» после завершения работы метода.

Область видимости

Областью видимости переменной называется та часть программы, которая может обращаться к этой переменной по имени.

В Java: область видимости переменных, объявленных в блоке, ограничивается строками этого блока. Такие переменные называют локальными.

Хороший стиль программирования – использование преимущественно локальных переменных.

Пример. Область видимости

```
public static void printSum(int x, int y)
{
 int sum = x + y;
 print(sum);
}
```

Область видимости переменных x, y, sum

```
public static void main(String[] args) {
   int x, sum;
   x = in.nextInt();
   sum = in.nextInt();
   printSum( x, sum );
}
```

Область видимости переменных x, sum

Формальные и фактические параметры

Список формальных параметров указывается в сигнатуре метода.

```
public static void sum (int x, int y)
```

Каждый такой параметр является локальным (т.е. к нему можно обращаться только в пределах данного метода).

Фактические параметры – параметры, которые передаются методу при обращении к нему.

```
int a = in.nextInt(), b = in.nextInt();
sum (a, b);
sum (1, 10);
```

Внимание! Количество и типы формальных и фактических параметров должны совпадать.

Что неправильно?

Какое имя лучше дать методу?

```
public static void sum (int x, int y, int z) {
  int u = x * y * z;
  System.out.printf("%d*%d*%d=%d", x, y, z, u);
 }
public static void main (String[] args) {
  sum (1,2,3);
 }
```

Метод, возвращающий значение

Метод, возвращающий значение – это вспомогательный алгоритм, результатом работы которого является некоторое значение.

Примеры:

- вычисление модуля числа, $\sqrt{x}$
- расчет значений по сложным формулам
- ответ на вопрос (простое число или нет?)

Зачем?

- для выполнения одинаковых расчетов в различных местах программы
- для создания общедоступных библиотек методов

Метод, возвращающий значение

Задача: составить метод, который вычисляет и возвращает наибольшее из двух значений

Метод:

формальные параметры

```
public static int max ( int a, int b )
{
  if ( a > b ) return a ;
  else         return b ;
}
```

return - вернуть результат

Метод, возвращающий значение

Особенности:

- в сигнатуре указывается тип результата

```
public static int max ( int a, int b )
```

- В сигнатуре описываются формальные параметры, они обозначаются именами и типами

```
public static float qq ( int a, float x )
```

Метод, возвращающий значение

Особенности:

- **Метод возвращает единственное значение, но может содержать несколько операторов возврата.**
- **Java-метод может возвратить только одно значение – того типа, который объявлен в сигнатуре метода.**
- **Управление возвращается в вызывающую программу как только в методе достигается первый оператор `return`.**

Программа

```
public static int max ( int a, int b )
{
  ...
}
public static void main(String[] args)
{
 int a, b, c;
 a = in.nextInt ();
 b = in.nextInt();
 c = max ( a, b );
 System.out.printf("max = %d", c);
}
```

формальные параметры

фактические параметры

вызов метода

Логические методы

Задача: составить метод, который определяет, верно ли, что заданное число – простое.

Особенности:

- ответ – логическое значение: `true` (да) или `false` (нет)
- результат метода можно использовать как логическую величину в условиях (`if, while`)

Алгоритм: считаем число делителей в интервале от 2 до N-1, если оно не равно нулю – число составное.

```
int count = 0;
for (int i = 2; i < N; i ++)
  if ( N % i == 0 ) count ++;
if ( count == 0 )
     // число N простое
else // число N составное
```

Как улучшить?

Логические методы

```
public static boole
{
int count = 0, i;
for (i = 2; i*i <= N; i++)
  if (N % i == 0) count ++;
return (count == 0);
}
```

перебор только до $\sqrt{N}$

Как улучшить?

```
if (count == 0) return true;
else            return false;
```

```
public static void main(String[] args)
{
  int N;
  N = in.nextInt();
 if ( isPrime(N) )
     System.out.printf ("%d - простое число", N);
  else System.out.println ("%d - составное число", N);
}
```

вызов метода

Стиль и использование методов

- Тщательно структурируйте ваш код
- Избегайте избыточности кода
- Следуйте соглашениям по именованию методов
- Используйте комментарии для описания поведения кода

Задача 1

Что будет выведено на экран при запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        int j = i * i * i;
        return j;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задача 2

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        int i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

Ошибка! Попытка объявить уже объявленную переменную `i`

Задача 3

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        i = i * i * i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

Ошибка! Отсутствует возвращаемое значение

Задача 4

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задача 5

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        return i * i * i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задания

1. Написать метод, который возвращает сумму всех чисел от 1 до N и привести пример его использования.

Пример:

```
Введите число:
100
Ответ: сумма чисел от 1 до 100 = 5050
```

2. Написать метод, который принимает в качестве параметров два целых числа и возвращает наибольшее значение модуля числа.

Пример:

```
Введите число: 4
Введите число: -5
Ответ: 5
```

Задания

3. Написать метод, который принимает в качестве параметра четыре значения целого типа `x1, y1, x2, y2` и возвращает расстояние между точками `(x1,y1)` и `(x2,y2)`.

Пример:

```
Введите координаты:
0 3 4 0
расстояние между точками (0, 3) и (4, 0) равно 5
```

4. Написать метод, который принимает в качестве параметра два значения типа: мантиссу и порядок и возвращает десятичную запись этого числа.

Пример:

```
Введите мантиссу: 6,23
Введите порядок: 5
Ответ: 623000.0
```

Программирование на языке Java

Тема 22. Перегрузка методов

Перегрузка методов

Сигнатура метода – совокупность его имени и набора формальных параметров.

Java позволяет создавать несколько методов с одинаковыми именами, но разными сигнатурами.

Создание метода с тем же именем, но с другим набором параметров называется перегрузкой.

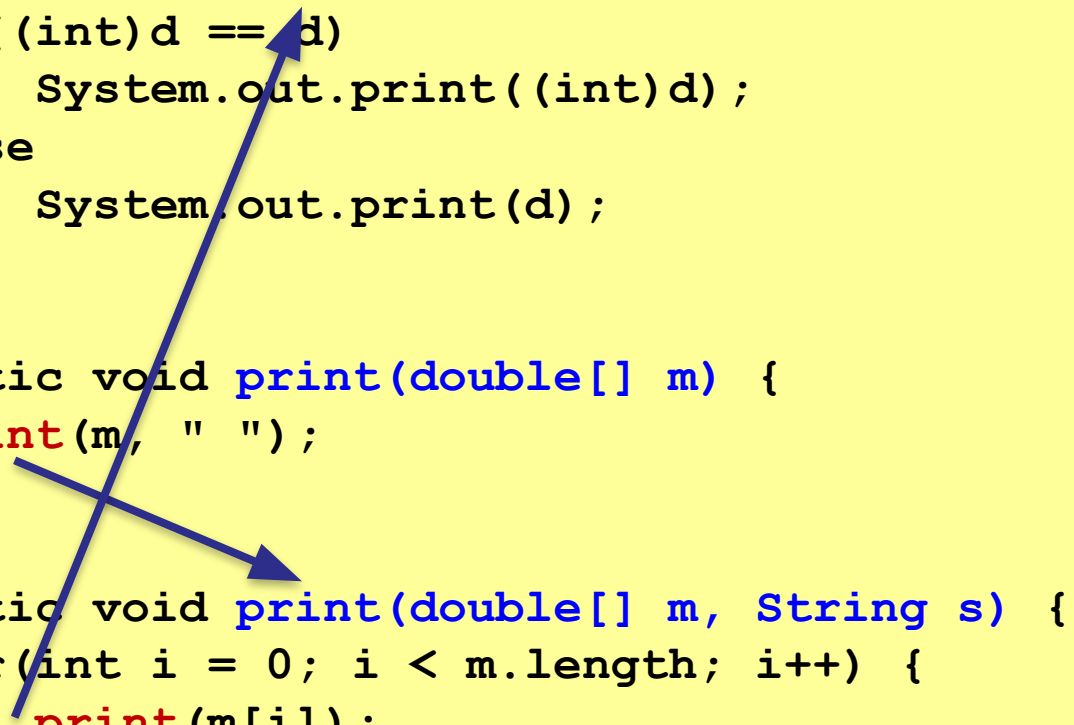
Какой из перегруженных методов должен выполняться при вызове, Java определяет на основе фактических параметров.

Перегрузка методов. Пример – 1

```
pubilc void print(double a) {
 System.out.println(a);
}
pubilc void print(String a) {
  System.out.println(a);
}
pubilc void print(int[] a) {
  for (int i=0; i<a.length; i++) {
    System.out.printf("%d ",a[i]);
  }
  System.out.println("");
}
...
int a = 5;
int [] m = {1, 2, 8, 3}
String s = "Мир";
print (a) // работает исходный метод
print (a + s); // 5 мир, работает первая перегрузка
print (m); // 1 2 8 3
print (m + a); // ошибка
```

Перегрузка методов. Пример – 2

```
public static void print() {
        System.out.println();}

public static void print(double d) {
        if((int)d == d)
            System.out.print((int)d);
        else
            System.out.print(d);
}

public static void print(double[] m) {
        print(m, " ");
}

public static void print(double[] m, String s) {
        for(int i = 0; i < m.length; i++) {
            print(m[i]);
            System.out.print(s);
        }
}
```



Перегрузка методов. Пример – 2

```
public static void main(String[] args) {
        double[] a = {1.0, 2.71, 3.14, 15, -5, 92, 0.5};
        double p = 3.0;
        int k = 13;
        print(p);
        print();
        print(a);
        print();
        print(a,", ");
        print();
        print(k);
    }
```

```
3
1 2.71 3.14 15 -5 92 0.5
1, 2.71, 3.14, 15, -5, 92, 0.5,
13
```

Программирование на языке Java

7. Отладка программы

Программирование на языке Java

Тема 7. Отладка программы

Отладка программ

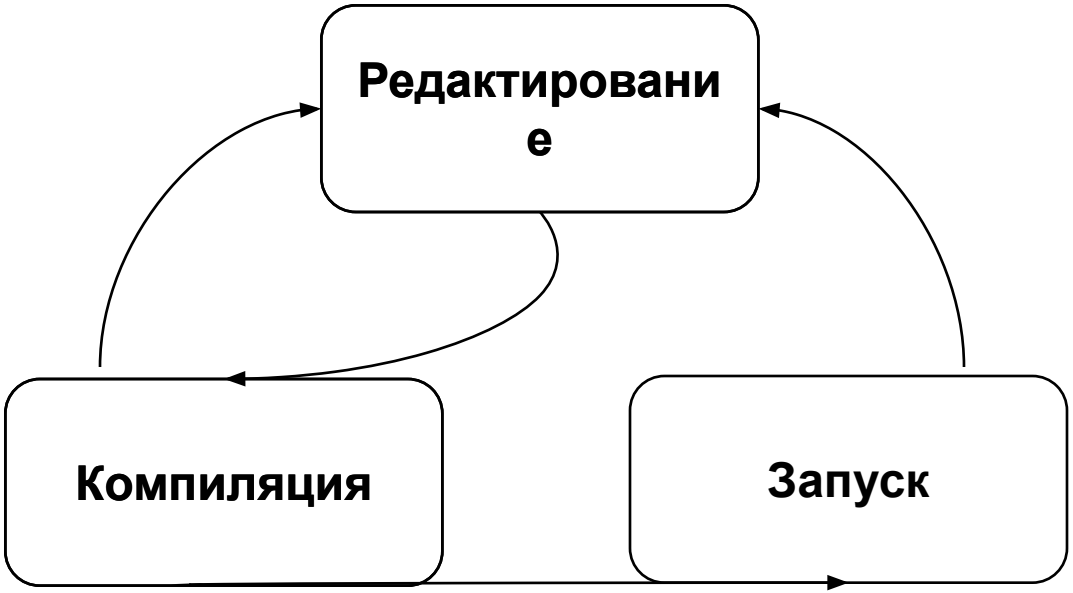
Отладка – поиск и исправление ошибок в программе.

Большая часть времени при разработке программного обеспечения на любом языке программирования – отладка.

Ошибки и отладка

Bug = *моль, жучок*

Debugging = *процесс устранения багов (ошибок)*



Отладка программ

Условия и циклы существенно увеличивают число возможных исходов.

Структура программы	Без циклов и условий	N условий	1 цикл
Число возможных последовательностей выполнения	1	2^n	∞

Большинство программ содержат по несколько условных операторов и циклов.

Отладка программ

Методы:

- **вывод сигнальных сообщений**
- **отключение части кода** (в комментарии)
- **трассировка** – пошаговое выполнения программы (выполнить одну строчку программы и остановиться)
- **точки останова** – выполнение программы останавливается при достижении отмеченных строк (переход в пошаговый режим)
- просмотр и изменение **значений переменных** в пошаговом режиме

Вывод сигнальных сообщений

```
…
 int i, X;
 System.out.print(“Введите X”);
 X = in.nextInt();

System.out.printf("Введено X=%d\n", X);

for(i=1; i<10; i++)
   {
System.out.printf(“Цикл:i=%d, X=%d\n", i, X);
   ...
   }
System.out.printf("После цикла: X=%d\n", X);
 ...
```

Отключение части кода (комментарии)

```
public static void main()
{
 int i, X;
  X = in.nextInt();
// X *= X + 2;
 for(i=1; i<10; i++) X *= i;
/* while ( X > 5 ) {
   ...
   } */
 ...
}
```

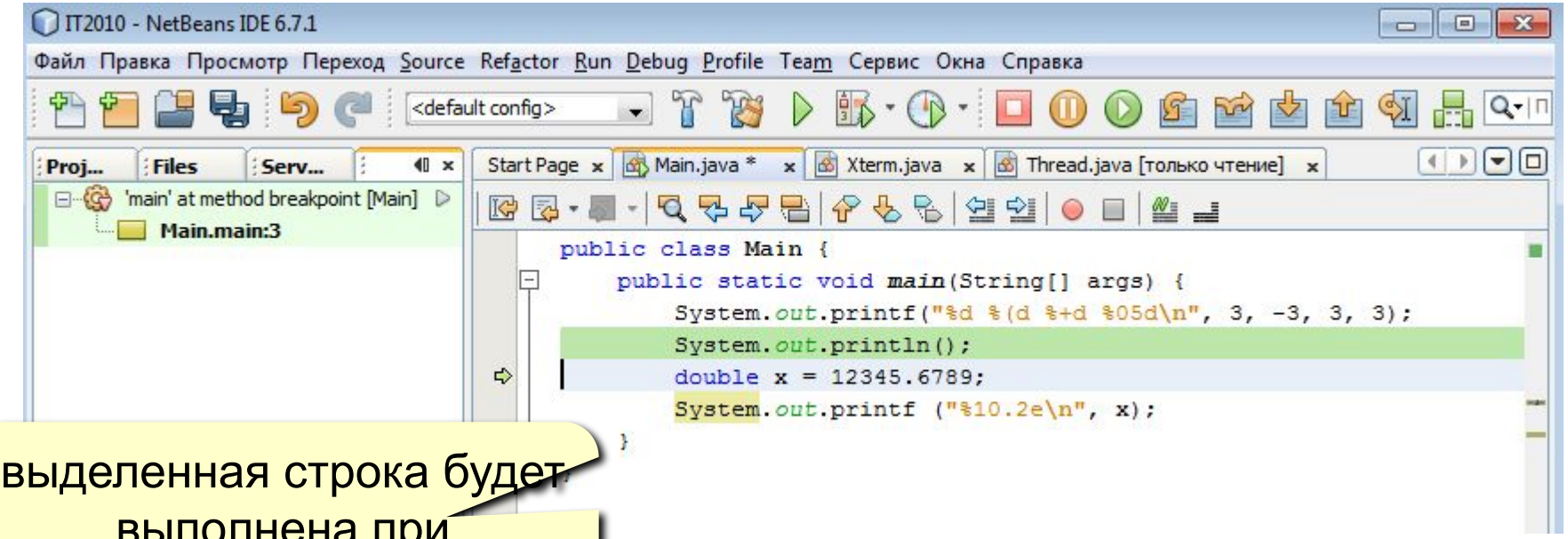
комментарий до конца строки //

закомментированный блок /* … */

Трассировка (пошаговое выполнение)

F7 – войти в процедуру или функцию

F8 – выполнить 1 строчку и остановиться



выделенная строка будет выполнена при **следующем** нажатии **F8**

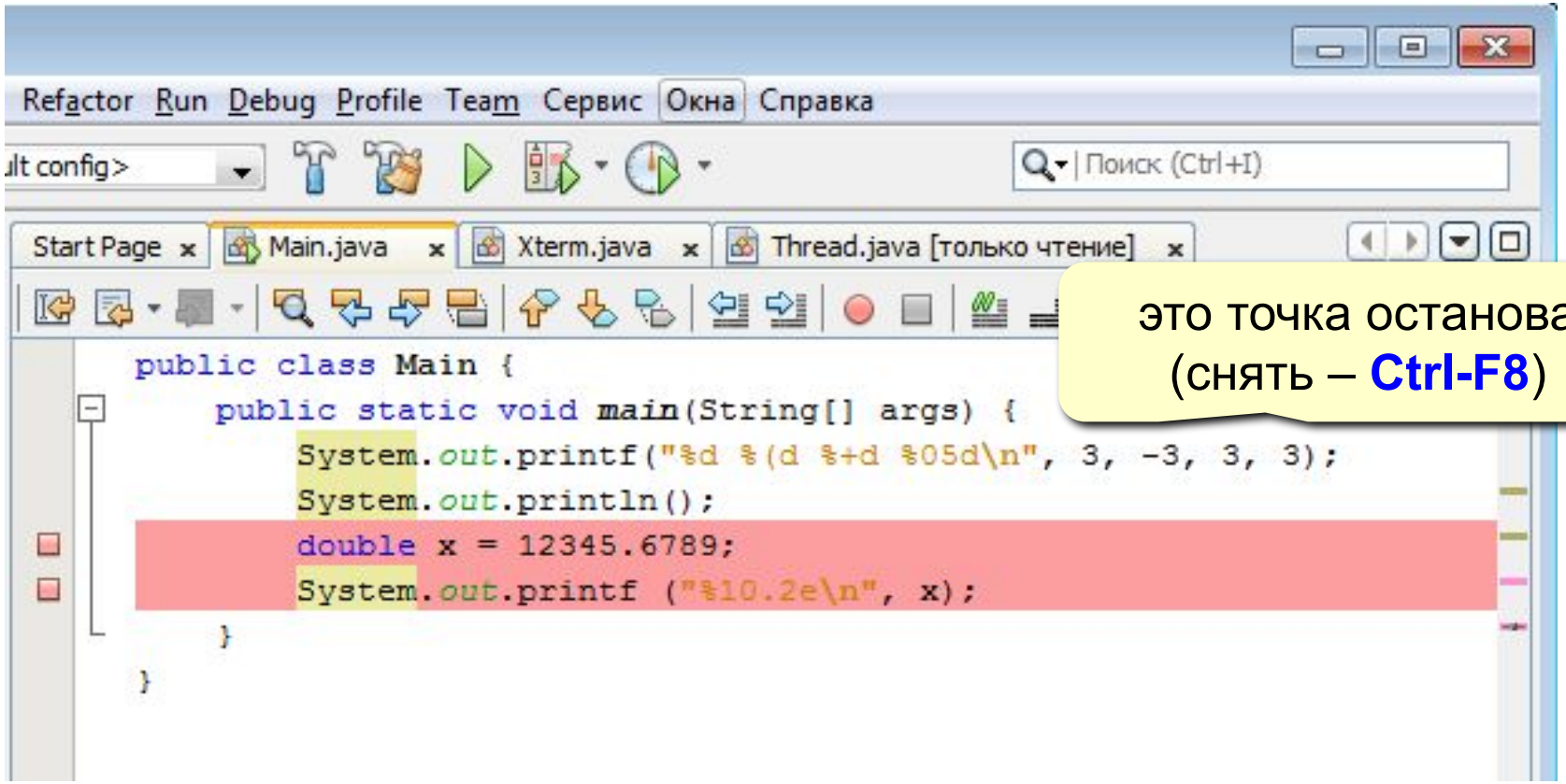
F5 – продолжить выполнение программы

Точки останова

F4 – выполнить непрерывно до той строчки, где стоит курсор (1 раз)

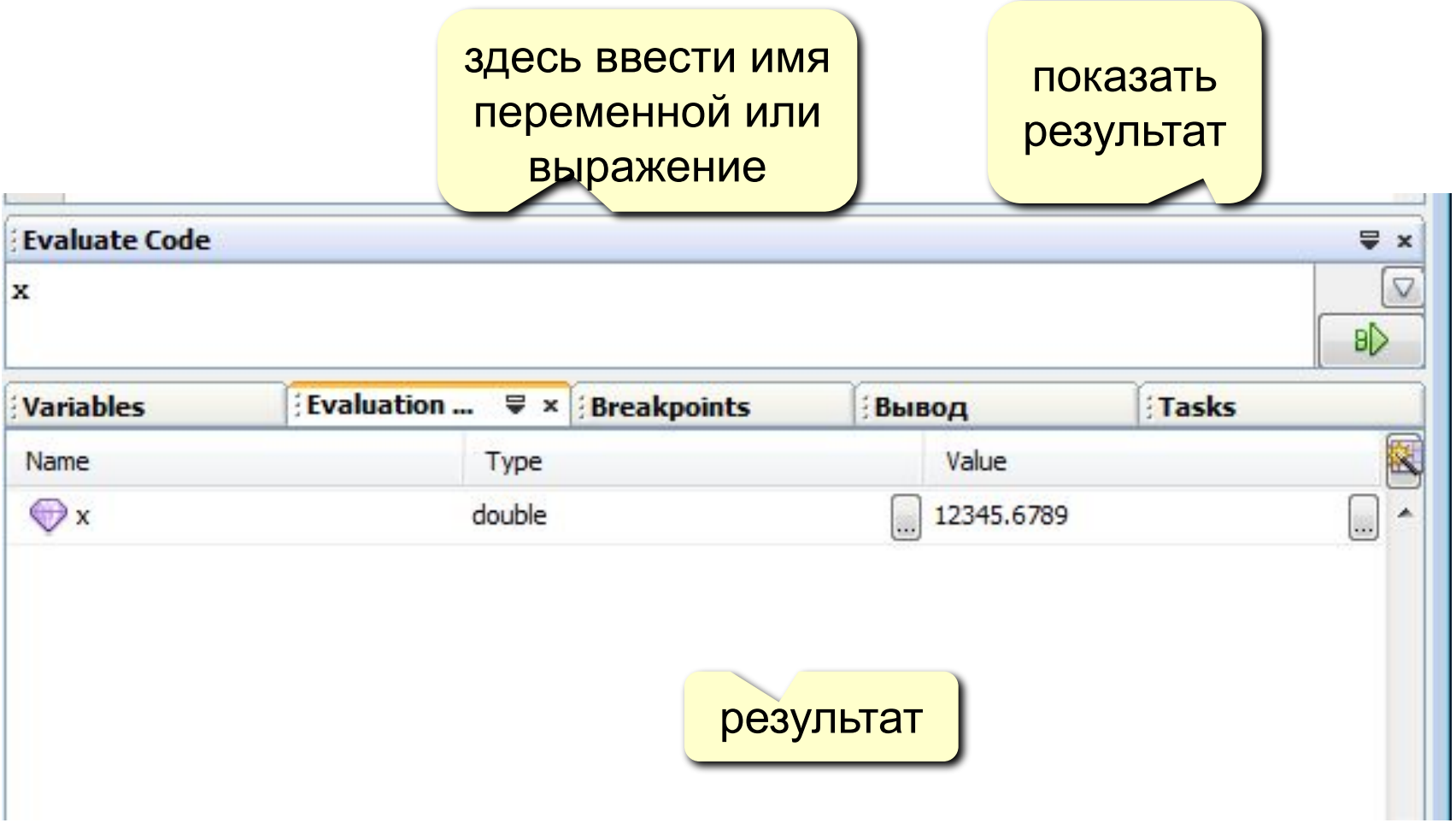
Ctrl-Shift-F5 – запустить отладчик

Ctrl-F8 – установить/снять точку останова



Просмотр значений переменных

Ctrl-F9 – открыть окно просмотра переменных



Виды ошибок

Синтаксические – ошибки в синтаксисе Java (эквивалент орфографических ошибок в естественном языке). Такие ошибки отслеживаются компилятором.

Логические возникают, когда код программы не соответствует поставленному заданию.

Ошибки времени выполнения возникают при выполнении программы (неверный ввод данных, деление на ноль, переполнение памяти и т.п.)

Синтаксические ошибки – 1

Имя файла не соответствует имени класса

```
WrongFileName.java:1: class Hello is
public,
should be declared in a file
named Hello.java
public class Hello {
^
1 error
```

Синтаксические ошибки – 2

Неверно записано имя метода

```
System.out.pruntln("Hello, world!");



Hello.java:3: cannot find symbol
symbol : method pruntln(java.lang.String)
location: class java.io.PrintStream
System.out.pruntln("Hello, world!");
           ^
1 error
```

Синтаксические ошибки – 3

Отсутствует точка с запятой

```
System.out.println("Hello, world!")



1 errorMissingSemicolon.java:4: ';' expected
System.out.println("Hello, world!");
^
1 error
```

Синтаксические ошибки – 4

Отсутствует ключевое слово

```
Bug4.java:1: class, interface,
or enum expected
public Bug4 {
       ^
1 error
```

```
Bug5.java:2: invalid method declaration;
return type required
public static main(String[] args) {
                ^
1 error
```

Синтаксические ошибки – 5

Не закрыта символьная строка

```
System.out.println("Hello, world!);


Hello.java:3: unclosed string literal
System.out.println("Hello, world!);
                   ^
Hello.java:4: ')' expected
  }
  ^
2 errors
```

Пример отладки программы

Задача. Дано число n, необходимо разложить его на простые множители.

Примеры. 98 = 2 × 7 × 7

3 757 208 = 2 × 2 × 2 × 7 × 13 × 13 × 397

11 111 111 111 111 111 = 2 071 723 × 5 363 222 357

Метод решения.

Рассмотрим каждое целое число i меньше n.

Пока i делит n нацело

Выведем i

Заменим n на n/i.

Пример отладки программы

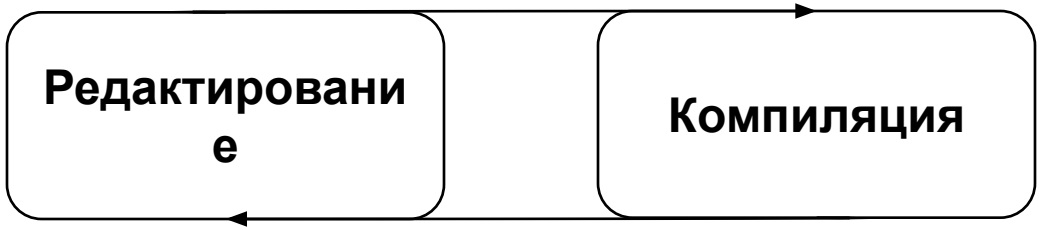
Дана программа

```
import java.util.Scanner;
public class Factor
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      long n = in.nextLong()
      for (i = 0; i < n; i++)
      {
         while (n % i == 0)
            System.out.print(i + " ")
            n = n / i
      }
   }
}
```

Отладка: синтаксические ошибки

Ваша программа соответствует синтаксису языка программирования Java?

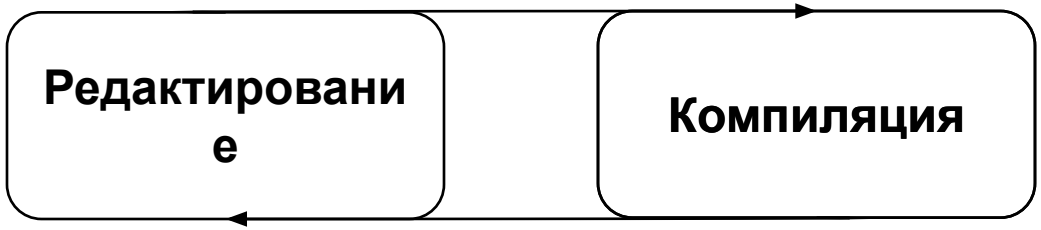
- Компилятор Java поможет найти ошибку
- Найдите первое сообщение об ошибке, которое выдает компилятор, исправьте
- Повторите
- Результат: исполняемые файл Factors.class



Отладка: синтаксические ошибки

Ваша программа соответствует синтаксису языка программирования Java?

- Компилятор Java поможет найти ошибку
- Найдите первое сообщение об ошибке, которое выдает компилятор, исправьте
- Повторите
- Результат: исполняемые файл Factors.class



Отладка: синтаксические ошибки

Дана программа

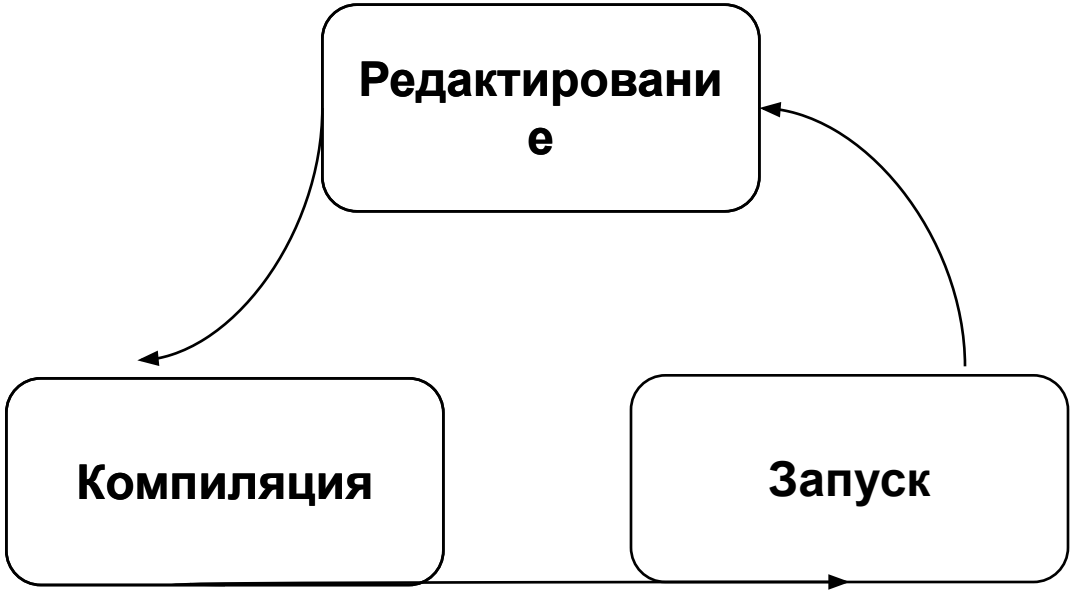
```
import java.util.Scanner;
public class Factor
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      long n = in.nextLong();
      for (int i = 0; i < n; i++)
      {
         while (n % i == 0)
            System.out.print(i + " ");
            n = n / i;
      }
   }
}
```

В программе остались ошибки

Отладка: ошибки времени выполнения

Выполняет ли ваша программа то, чего вы от нее ждете?

- Запустите программу и проверьте
- Найдите первую ошибку времени выполнения
- Исправьте ее и повторите



Отладка: ошибки времени выполнения

```
import java.util.Scanner;
public class Abs {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        long n = in.nextLong();
        for (int i = 2; i < n; i++) {
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }
    }
}
```

В программе остались ошибки

Программирование на языке Java

Тема 41. Работа с файлами (ввод и вывод)

Класс `Scanner`

`Scanner` – класс, который читает форматный ввод и преобразует его в бинарную форму.

`Scanner` позволяет читать данные с клавиатуры, из файла на диске, из строки.

Класс Scanner. Чтение с клавиатуры

Общий вид:

```
Scanner in1 = new Scanner(System.in);
Scanner in2 = new Scanner(System.in, “cp1251”);
```

Класс Scanner. Чтение из строки

Общий вид:

```
Scanner in =
    new Scanner("10 99,88 сканирование это
просто");
int a = in.nextInt();          // 10
double b = in.nextDouble();  // 99.88
String s = in.next(); // "сканирование"
```

Класс Scanner. Чтение из файла

Общий вид:

```
File file = new File ("in.txt");
Scanner in1 = new Scanner(file);
Scanner in2 = new Scanner(file, "cp1251");
```

Внимание! Для того, чтобы работать с классом `File`, нужно подключить пакет `java.io` с помощью команды:

```
import java.io.*;
```

Внимание! В методе `main` нужно указать исключение ввода-вывода, которое может генерироваться этим методом

```
public static void main(String[] args)
            throws IOException {
```

Основы сканирования

`Scanner` читает лексемы из некоторого источника (с клавиатуры, из строки, из файла), который указан при создании объекта `Scanner`.

Лексема – порция ввода, отделенная набором разделителей, которыми по умолчанию являются пробелы.

Процедура сканирования

1. Определите, доступен ли специфический тип ввода вызовом одного из методов класса `Scanner` `hasNextX()`, где `X` – нужный тип данных.
2. Если ввод доступен, читайте его одним из методов класса `Scanner` `nextX()`.
3. Повторяйте процесс до завершения ввода.

Пример. Чтение целых чисел с клавиатуры

```
Scanner in = new Scanner (System.in);
    int i;
    while(in.hasNextInt()) {
        i = in.nextInt();
        // ...
    }
```

Цикл `while` остановится, как только следующая лексема окажется не целым числом.

Некоторые методы `hasNext` – 1

Метод	Описание
`boolean hasNext()`	Возвращает **`true`**, если доступна для чтения лексема любого типа.
`boolean hasNextBoolean()`	Возвращает **`true`**, если доступно для чтения значение типа **`boolean`**.
`boolean hasNextByte()`	Возвращает **`true`**, если доступно для чтения значение типа **`byte`**.
`boolean hasNextDouble()`	Возвращает **`true`**, если доступно для чтения значение типа **`double`**.
`boolean hasNextFloat()`	Возвращает **`true`**, если доступно для чтения значение типа **`float`**.
`boolean hasNextInt()`	Возвращает **`true`**, если доступно для чтения значение типа **`int`**.

Некоторые методы `hasNext` – 2

Метод	Описание
`boolean hasNextLine()`	Возвращает **`true`**, если доступна строка ввода.
`boolean hasNextLong()`	Возвращает **`true`**, если доступно для чтения значение типа **`long`**.
`boolean hasNextShort()`	Возвращает **`true`**, если доступно для чтения значение типа **`short`**.

Некоторые методы `next` – 1

Метод	Описание
`String next()`	Возвращает следующую лексему любого типа из входного источника.
`boolean nextBoolean()`	Возвращает следующую лексему как значение типа **`boolean`**.
`byte nextByte()`	Возвращает следующую лексему как значение типа **`byte`**.
`double nextDouble()`	Возвращает следующую лексему как значение типа **`double`**.
`float nextFloat()`	Возвращает следующую лексему как значение типа **`float`**.
`int nextInt()`	Возвращает следующую лексему как значение типа **`int`**.

Некоторые методы `next` – 2

Метод	Описание
`String nextLine()`	Возвращает следующую строку ввода.
`long nextLong()`	Возвращает следующую лексему как значение типа **`long`**.
`short nextShort()`	Возвращает следующую лексему как значение типа **`short`**.

Пример. Чтение с клавиатуры

```
Scanner in = new Scanner (System.in);
int count = 0; double sum = 0;
while (in.hasNext()) {
    if (in.hasNextDouble()) {
        sum += in.nextDouble();
        count++; }
    else
      break;
}
System.out.printf("Среднее = %f", sum / count);
```

Пример. Чтение из файла – 1

Рассмотрим ту же самую задачу, но с чтением данных из файла. Пусть имеется файл `in.txt`, который находится в папке проекта.

`in.txt`

```
2  3,4  5  6  7,4  9,1  10,5
```

Пример. Чтение из файла – 2

Подгружаем пакет для работы с классом `File`

Метод `main` может генерировать исключения ввода-вывода

```
import java.util.*;
import java.io.*;
public class Main {
public static void main(String[] args)
                throws IOException {
File file = new File("in.txt");
Scanner in = new Scanner (file, "cp1251");

int count = 0;
double sum = 0;
```

Обращение к файлу `in.txt`

Пример. Чтение из файла – 3

```
while (in.hasNext()) {
    if (in.hasNextDouble()) {
        sum += in.nextDouble();
        count++; }
    else
        break;
}
System.out.printf("Среднее = %f", sum / count);
}}
```

Дальнейший код повторяет пример с чтением с клавиатуры

Класс PrintWriter

PrintWriter – класс, который применяется для записи файла.

Внимание! Для того, чтобы работать с классом PrintWriter, нужно подключить пакет java.io с помощью команды

```
import java.io.*;
```

Внимание! В методе main нужно указать исключения ввода-вывода, которые могут генерироваться этим методом

```
public static void main(String[] args)
            throws IOException {
```

Конструкторы класса `PrintWriter`

Общий вид:

```
PrintWriter out = new PrintWriter(<имя файла>);
```

Каждый раз при новой записи предыдущие данные будут стираться.

Некоторые методы `PrintWriter`

Метод	Описание
`void close()`	Закрывает поток. Последующие попытки записи генерируют исключения **`IOException`**.
`void print(тип x)`	Записывает значение **`x`** в выходной поток
`void println(тип x)`	Записывает значение **`x`** и перевод строки в выходной поток
`void printf("форматная строка", список_аргументов)`	Записывает в выходной поток отформатированную строку

Пример. Запись в файл

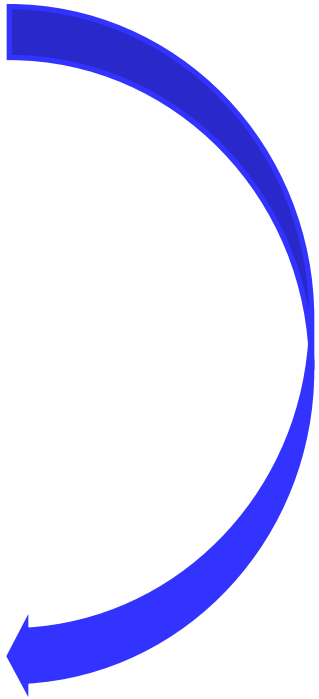
```
import java.io.*;
public class Main {
public static void main(String[] args)
                throws IOException {
  PrintWriter out = new PrintWriter("out.txt");
  out.print ("2 3,4 5 6 7,4 9,1 10,5");
  out.close();
 }
}
```

Файл `out.txt` будет помещен в папке проекта.

Пример. Чтение и запись – 1

student.in

```
Дмитрий 24
Петр 23
Ольга 22
```



student.out

```
Имя: Дмитрий ; возраст: 24
Имя: Петр; возраст: 23
Имя: Ольга; возраст: 22
```

Пример. Чтение и запись – 2

```
import java.util.*;
import java.io.*;
public class Main {
  public static void main(String[] args)
              throws IOException {
   File file = new File(“student.in");
   Scanner in = new Scanner(file);
   PrintWriter out =
                 new PrintWriter(“student.out");
```

Пример. Чтение и запись – 3

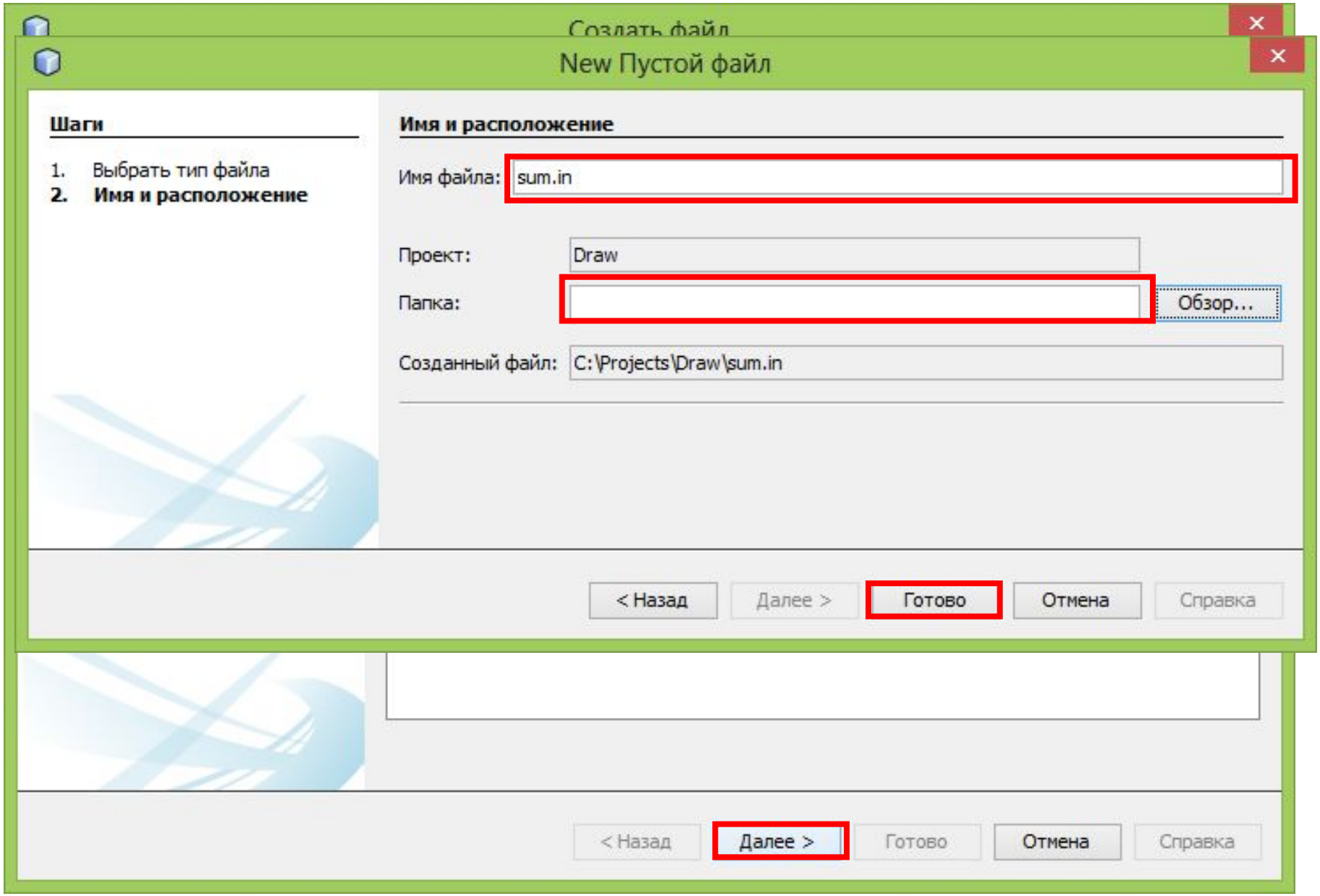
```
 while (in.hasNext()) {
  String name = in.next();
  int age = in.nextInt();
  out.printf("Имя: %s; возраст: %d\n", name,
age);
 }
 in.close();
 out.close();
 }
}
```

Создание текстовых файлов в NetBeans

Для создания текстовых файлов

1. Вызовите меню «Файл» - «Создать файл»,
2. Выберите категорию «Прочее», тип файла «Пустой файл».
3. Задайте имя файла с расширением, например sum.in
4. В поле «Папка» должно быть пусто.
5. Нажмите кнопку «Готово».
6. В открывшемся окне отредактируйте входные данные для программы и сохраните их.

Создание текстовых файлов в NetBeans



Задание

Задача 1. Напишите программу, которая считывает 2 целых числа из файла и выводит сумму этих чисел в другой файл.

Задача 2. Напишите программу, которая считывает все целые числа из файла и выводит сумму этих чисел в другой файл.

Класс FileWriter

FileWriter – класс, который применяется для записи файла.

Внимание! Для того, чтобы работать с классом FileWriter, нужно подключить пакет java.io с помощью команды

```
import java.io.*;
```

Внимание! В методе main нужно указать исключение ввода-вывода, которое может генерироваться этим методом

```
public static void main(String[] args)
            throws IOException {
```

Конструкторы класса `FileWriter`

Общий вид:

Каждый раз при записи предыдущие данные будут удаляться из файла.

```
FileWriter out = new FileWriter(<имя файла>);
```

Новые данные будут дописываться в конец файла.

```
FileWriter out = new FileWriter(<имя файла>,
                                true);
```

Некоторые методы `FileWriter`

Метод	Описание
`void close()`	Закрывает поток. Последующие попытки записи генерируют **`IOException`**.
`void write(String str)`	Записывает строку **`str`** в выходной поток

Пример. Запись в файл

```
import java.io.*;
public class Main {
public static void main(String[] args)
                throws IOExceptio
  FileWriter out = new FileWriter("out.txt");
  out.write("2 3,4 5 6 7,4 9,1 10,5 end");
  out.close();
 }
}
```

Запись данных в файл

Поток закрыт

Файл `out.txt` будет помещен в папку проекта.

Класс Formatter

Formatter – класс, который предлагает преобразования формата, позволяющие отображать числа, строки в любом виде.

Общий вид:

```
Formatter fmt = new Formatter();
fmt.format(<форматная строка>,
      <список аргументов>);
```

Внимание! Для того, чтобы работать с классом Formatter, нужно подключить пакет java.util.Formatter

Пример использования класса Formatter

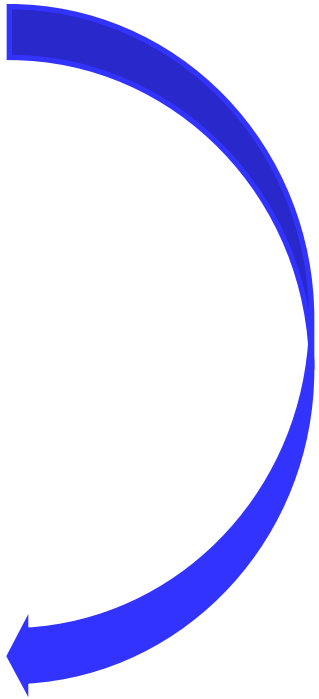
```
Formatter fmt = new Formatter();
fmt.format("Форматировать %s очень
          просто: %d, %f",
        "с помощью Java", 10, 98.5);
```

Объект Formatter, содержащий строку «Форматировать с помощью Java очень просто: 10, 98,500000»

Пример. Чтение и запись – 1

in.txt

```
Василий 24
Петр 23
Анна 24
```



out.txt

```
Имя: Василий; возраст: 24
Имя: Петр; возраст: 23
Имя: Анна; возраст: 24
```

Пример. Чтение и запись – 2

```
import java.util.*;
import java.io.*;
public class Main {
  public static void main(String[] args)
              throws IOException {
    File file = new File("in.txt");
    Scanner in = new Scanner(file, "cp1251");
    FileWriter out = new FileWriter("out.txt");
    Formatter fmt = new Formatter();
    String s;
```

Пример. Чтение и запись – 3

```
 while (in.hasNext()) {
   String name = in.next();
   int age = in.nextInt();
   fmt.format("Имя: %s; возраст: %d\n",name,age);
 }
 in.close();
 s = fmt.toString();
 out.write(s);
 out.close();
 }
}
```

Программирование на языке Java

Тема 45. Преобразование типов

Java – строго типизированный язык

Java – язык строго типизированный язык. Компилятор и виртуальная машина всегда следят за работой с типами, гарантируя надежность выполнения программы.

Однако, есть случаи, когда нужно осуществить то или ...вание для реализации логики ...

Преобразование к `long`

`int`

Сложение `int` и `char`

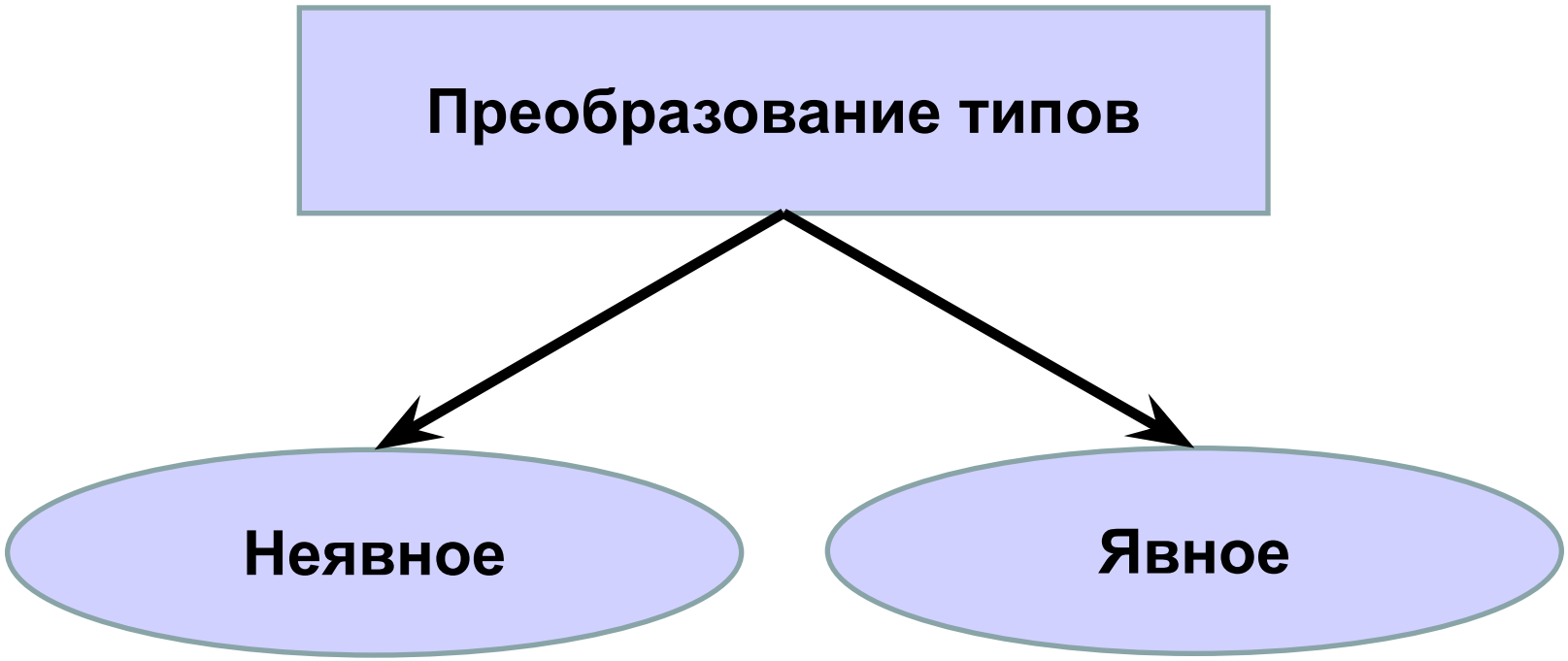
Сложение `int` и `long`

Преобразование к `double`

```
a = 5 + 'A' + a;
System.out.print("a="+Math.round(a/2.0));
```

Преобразование к `String`

Преобразование типов



Неявное преобразование типов

Выполняется в случае, если выполняются условия:

1. **Оба типа совместимы;**
2. **Длина целевого типа больше длины исходного типа.**

Пример. Преобразование от `byte` к `int`.

Явное преобразование типов

Общая форма явного преобразования

```
(<целевой тип>) <значение>
```

1. Если длина целевого типа меньше длины исходного типа, то происходит преобразование с сужением.
2. Если значение переменной вещественного типа присваивается переменной целого типа, то выполняется усечение (отбрасывается др часть).

Сужение

Пример 1. Преобразование от `int` к `short`.

Пример 2. Преобразование от `float` к `int`.

Усечение

Преобразование типов. Пример

```
int b = 1;
byte a = b;
byte c = (byte)-b;
int i = c;
```

Ошибка!

Явное преобразование переменной типа `int` к типу `byte`

Автоматическое неявное преобразование переменной типа `byte` к типу `int`

Виды приведений

- тождественное (`identity`);
- расширение примитивного типа (`widening primitive`);
- сужение примитивного типа (`narrowing primitive`);
- расширение объектного типа (`widening reference`);
- сужение объектного типа (`narrowing reference`);
- преобразование к строке (`String`);
- запрещенные преобразования (`forbidden`).

Тождественное преобразование

В Java преобразование выражения любого типа к точно такому же типу всегда допустимо и успешно выполняется.

Для чего нужно такое преобразование?

- Любой тип в Java может участвовать в преобразовании, хотя бы в тождественном.

Тип boolean можно привести только к boolean.

- Облегчение чтения кода для разработчика.

```
print(getCity().getStreet().getHouse().getFlat());

print((Flat)(getCity().getStreet().getHouse().
   getFlat()));
```

Преобразование типов

Простой тип, расширение	Объектный тип, расширение
Простой тип, сужение	Объектный тип, сужение

Расширение простого типа

Расширение простого типа – переход от менее емкого типа к более емкому.

Это преобразование **безопасно** – новый тип вмещает все данные, которые хранились в старом.

Не происходит потери данных.

Например, от типа `byte` (1 байт) к типу `int` (4 байта)

```
byte b = in.nextByte();
int a = b;
```

Типы данных

Тип	Размер	Min	Max
`byte` (байт)	8 бит	-128	+127
`short` (короткое целое)	16 бит	-2^{15}	$2^{15}-1$
`int` (целое число)	32 бита	-2^{31}	$2^{31}-1$
`long` (целое число двойного размера)	64 бита	-2^{63}	$2^{63}-1$
`char` (символьный)	16 бит	0	$2^{16}-1$
`float` (вещественные)	32 бита	1.4e-45	3.40e+038
`double` (вещественные двойной точности)	64 бита	4.9e-324	1.79e+308

Расширения простого типа

- от `byte` к `short, int, long, float, double`
- от `short` к `int, long, float, double`
- от `char` к `int, long, float, double`
- от `int` к `long, float, double`
- от `long` к `float, double`
- от `float` к `double`

19

Сколько существует расширяющих преобразований простого типа?

Почему?

Нельзя провести расширение

- от типов `byte` и `short` к типу `char`
- от типа `char` к типу `short`

Искажения при расширении

У дробных типов значащих разрядов (разрядов, отведенных на представление мантиссы) меньше, чем у некоторых целых типов, поэтому возможны **искажения значений** при:

- приведении значений `int` к типу `float`;
- приведении значений типа `long` к типу `float` или `double`.

```
long a=111111111111L;
float f = a;
System.out.println(f);
a = (long) f;
System.out.print(a);
```

```
1.11111111e11
```

```
111111110656
```

Сужения простого типа

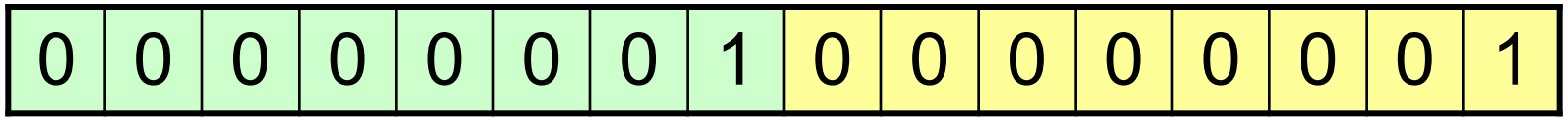
Сужение простого типа – переход осуществляется от более емкого типа к менее емкому.

Риск потерять данные! При сужении целочисленного типа все старшие биты отбрасываются.

257

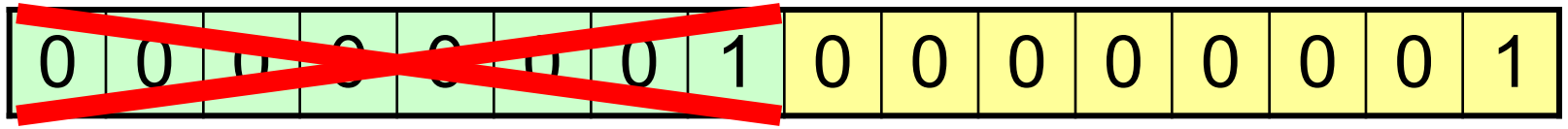
16 бит

```
short a = in.nextShort();
```



8 бит

```
byte b = (byte)a;
```




```
System.out.print(b);
```

1

Сужения простого типа

- от `byte` к `char`
- от `short` к `byte`, `char`
- от `char` к `byte`, `short`
- от `int` к `byte`, `short`, `char`
- от `long` к `byte`, `short`, `char`, `int`
- от `float` к `byte`, `short`, `char`, `int`, `long`
- от `double` к `byte`, `short`, `char`, `int`, `long`, `float`

Сужения простого типа. Примеры

```
System.out.print((byte)383);

System.out.print((byte)384);

System.out.print((byte)-384);
```

127

-128

-128

Кроме значения может быть потерян знак

```
char c=40000;
System.out.print((short)c);
```

-25536

Сужение дробного типа до целочисленного

Не число (`Not A Number`)

1. Дробное значение преобразуется в `long`, если целевым типом является `long`, или в `int` – в противном случае, т.е. дробная часть отбрасывается.
   - Если исходное число – `NaN`, то результат 0.
   - Если исходное число – положительная или отрицательная бесконечность, то результат – максимальное или минимальное значение выбранного типа.
   - Если дробное число конечной величины, но после округления получилось слишком большое по модулю число выбранного типа, то результатом будет максимальное или минимальное число выбранного типа.
   - Если результат округления уложился в диапазон типа, то он и будет результатом.

Сужение дробного типа до целочисленного

2. Производится дальнейшее сужение от выбранного целочисленного типа к целевому, если это необходимо.

Сужение дробного типа. Пример

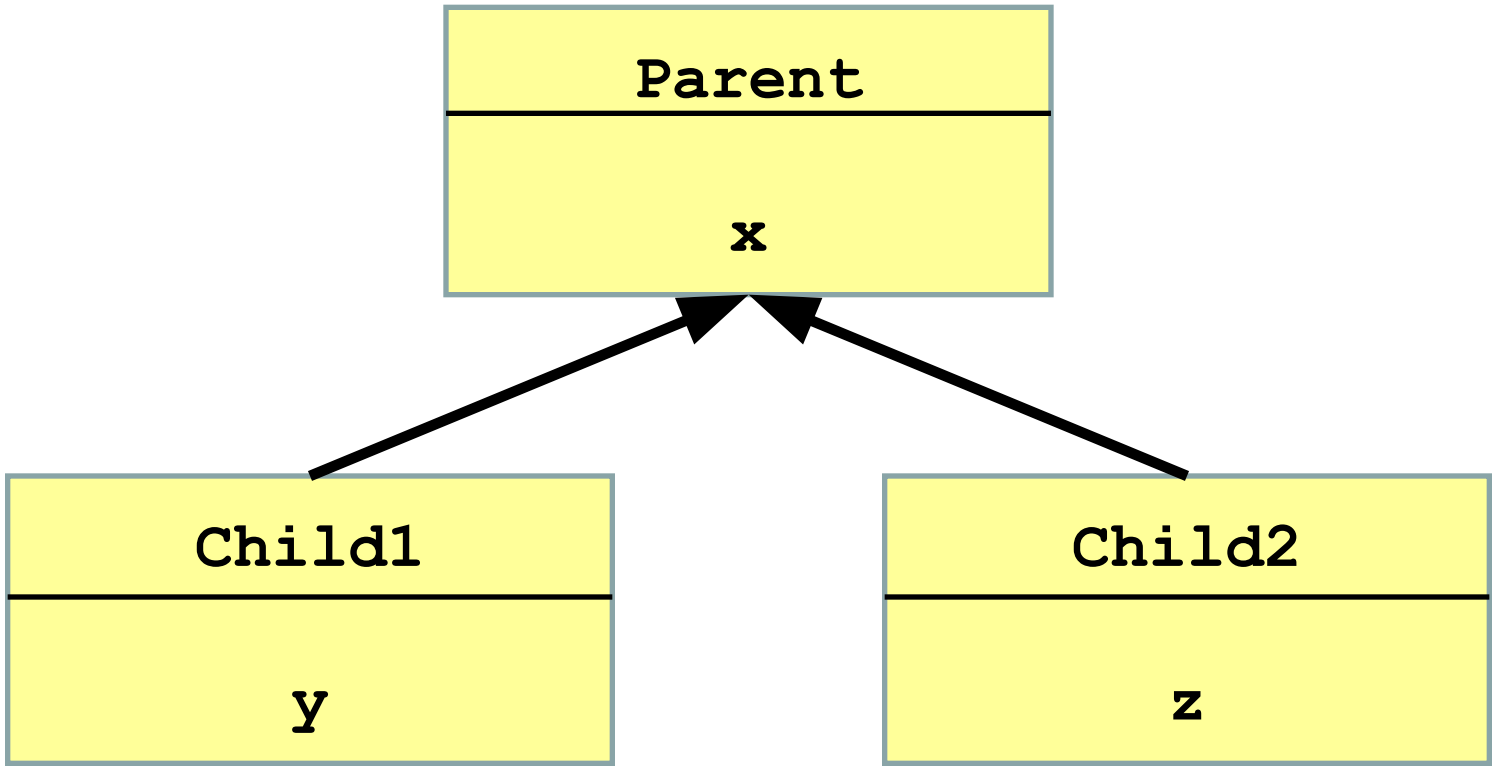
```
float fmin = Float.NEGATIVE_INFINITY;
float fmax = Float.POSITIVE_INFINITY;
print("long:"+(long)fmin+".."+(long)fmax);
print("int:"+(int)fmin+".."+(int)fmax);
print("short:"+(short)fmin+".."+(short)fmax);
print("char:"+(char)fmin+".."+(char)fmax);
print("byte:"+(byte)fmin+".."+(byte)fmax);
```

```
long: -9223372036854775808..9223372036854775807
int: -2147483648..2147483647
short: 0..-1
char: 0..65535
byte: 0..-1
```

Преобразование объектных типов

```
// Объявляем класс Parent
class Parent {
int x;
}
// Объявляем класс Child и наследуем
// его от класса Parent
class Child1 extends Parent {
int y;
}
// Объявляем второго наследника
// класса Parent - класс Child2
class Child2 extends Parent {
int z;
}
```

Схема наследования



Расширение объектных типов

Расширение – переход от более конкретного типа к менее конкретному, т.е. переход от детей к родителям.

Производится JVM автоматически.

Пример. Преобразование от наследника (`Child1`, `Child2`) к родителю (`Parent`).

```
Parent p1 = new Child1();
Parent p2 = new Child2();
```

Внимание! С помощью ссылок `p1` и `p2` можно обращаться только к полю `x`, а поля `y` и `z` недоступны.

Расширение объектных типов

1. От класса `A` к классу `B`, если `A` наследуется от `B` (важным частным случаем является преобразование от любого ссылочного типа к `Object`);
2. От `null`-типа к любому объектному типу.

Значение `null` может принять переменная любого ссылочного типа. Это означает, что ее ссылка никуда не указывает, объект отсутствует.

```
Parent p = null;
```

Пустая ссылка

Сужение объектных типов

Сужение – переход от менее конкретного типа к более конкретному, т.е. переход от родителей к детям.

Внимание! Такой переход не всегда возможен

```
Parent p = new Child1();
Child1 c = (Child1)p;

Parent p2 = new Child2();
Child1 c2 = (Child1)p2;
```

Успешное преобразование

Ошибка!

Оператор instanceof

Оператор `instanceof` проверяет принадлежность объекта указанному типу, возвращает значение типа `boolean`.

Объект

Класс

```
p instanceof Parent
```

```
Parent p = new Child1();
if (p instanceof Child1) {
  Child1 c = (Child1)p; }
Parent p2 = new Child2();
if (p2 instanceof Child1) {
  Child1 c = (Child1)p2; }
Parent p3 = new Parent();
if (p3 instanceof Child1) {
  Child1 c = (Child1)p3; }
```

Преобразование выполнено

Преобразование не выполнено

Преобразование не выполнено

Сужение объектных типов

1. От класса `A` к классу `B`, если `B` наследуется от `A` (важным частным случаем является сужение типа `Object` до любого другого ссылочного типа).

Преобразование к строке

Любой тип может быть преобразован к строке, т.е. к экземпляру класса `String`.

- Числовые типы записываются в текстовом виде без потери точности представления.
- Булевская величина приводится к строке "`true`" или "`false`" в зависимости от значения.
- Для объектных величин вызывается метод `toString`(). Если метод возвращает `null`, то результатом будет строка "`null`".
- Для `null-значения` генерируется строка "`null`".

Запрещенные преобразования

Внимание! Попытка осуществить запрещенное преобразование вызовет ошибку компиляции.

Запрещенные преобразования:

- Переходы от любого объектного типа к примитивному
- Переходы от примитивного типа к объектному
- Тип `boolean` нельзя привести ни к какому другому типу, кроме `boolean` (за исключением приведения к строке).
- Невозможно привести друг к другу типы, находящиеся не на одной, а на соседних ветвях дерева наследования
- и другие.

Применение приведений

1. Присвоение значений переменным (`assignment`).
2. Вызов метода. Это преобразование применяется к аргументам вызываемого метода или конструктора. Явное приведение. В этом случае явно указывается, к какому типу требуется привести исходное значение. Допускаются все виды преобразований, кроме приведений к строке и запрещенных.
3. Оператор конкатенации производит преобразование к строке своих аргументов.
4. Числовое расширение (`numeric promotion`). Числовые операции могут потребовать изменения типа аргумента(ов).

Автоматическое повышение типов

Правила повышения типа:

1. **Тип значения `byte`, `short` и `char` повышаются до `int`.**
2. **Если один из операндов имеет тип `long`, то тип всего выражения повышается до `long`.**
3. **Если один из операндов имеет тип `float`, то тип всего выражения повышается до `float`.**
4. **Если один из операндов имеет тип `double`, то тип всего выражения повышается до `double`.**

Программирование на языке Java

Тема 46. Битовые операции

Битовые операции

Битовые операции

Битовые операции — некоторые операции над цепочками битов.

В программировании, как правило, рассматриваются лишь некоторые виды этих операций:

- **логические побитовые операции;**
- **битовые сдвиги.**

В `Java` битовые операции могут применяться к целочисленным типам: `byte, char, short, int, long`.

Побитовые операции в Java – 1

Операция	Описание
~	Побитовая унарная операция NOT (Не, инверсия)
&	Побитовое AND (И)
|	Побитовое OR (ИЛИ)
^	Побитовое XOR (исключающее ИЛИ)
>>	Сдвиг вправо
>>>	Сдвиг вправо с заполнением нулями (без учета знака)
<<	Сдвиг влево
&=	Побитовое AND с присваиванием
|=	Побитовое OR с присваиванием

Побитовые операции в Java – 2

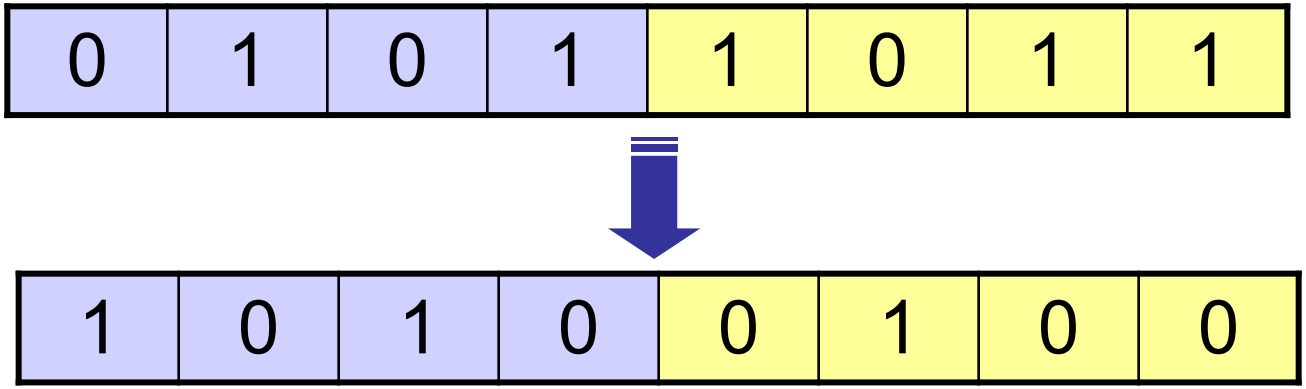
Операция	Описание
^=	Побитовое исключающее OR с присваиванием
>>=	Сдвиг вправо с присваиванием
>>>=	Сдвиг вправо с заполнением нулями (без учета знака) с присваиванием
<<=	Сдвиг влево с присваиванием

Побитовые логические операции

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Инверсия (операция НЕ)

Инверсия – это замена всех «0» на «1» и наоборот.




```
int n = 91;
n = ~n;
print(n);
```

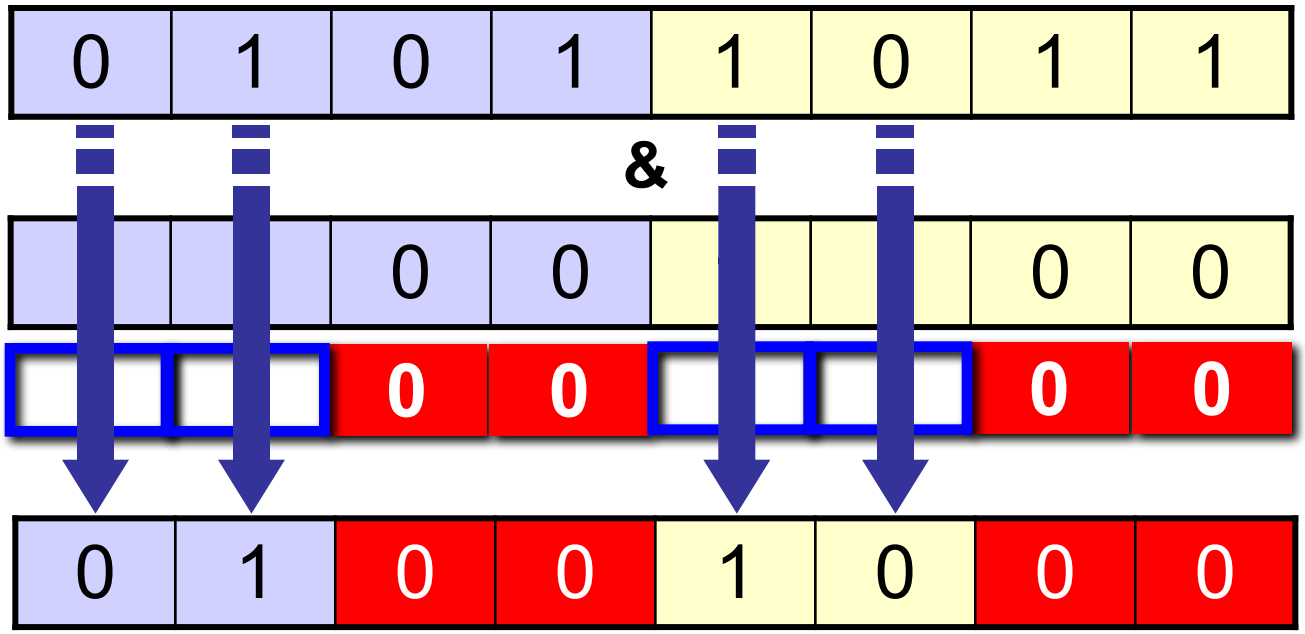
-92

Операция И

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Обозначения:

И, ∧, & (Java)



маска

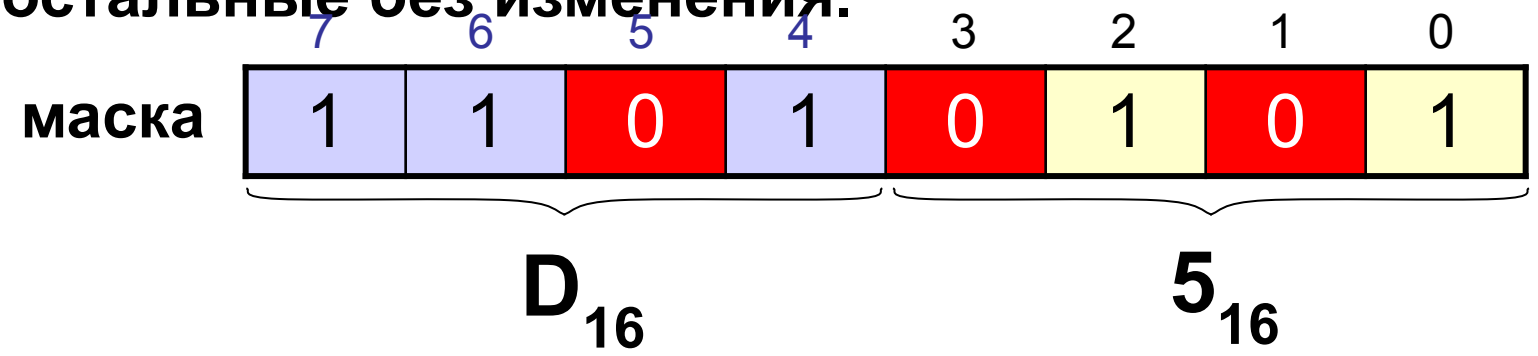
x & 0 =0
x & 1 =x

$5B_{16}$ & CC_{16} = 48_{16}

Операция И – обнуление битов

Маска: обнуляются все биты, которые в маске равны «0».

Задача: обнулить 1, 3 и 5 биты числа, оставив остальные без изменения.

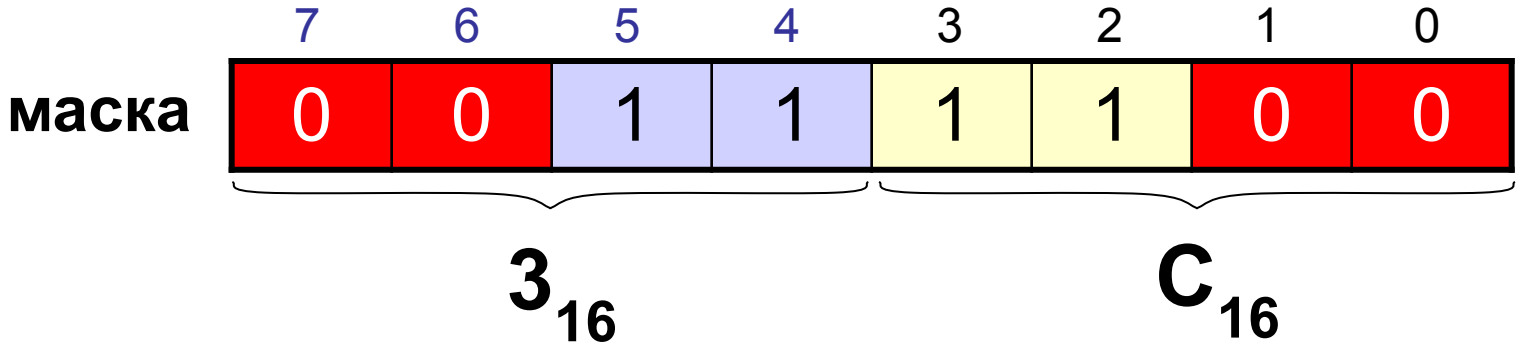



```
int n;
n = n & 0xD5;
```

Запись шестнадцатеричного числа

Операция И – проверка битов

Задача: проверить, верно ли, что все биты 2…5 – нулевые.



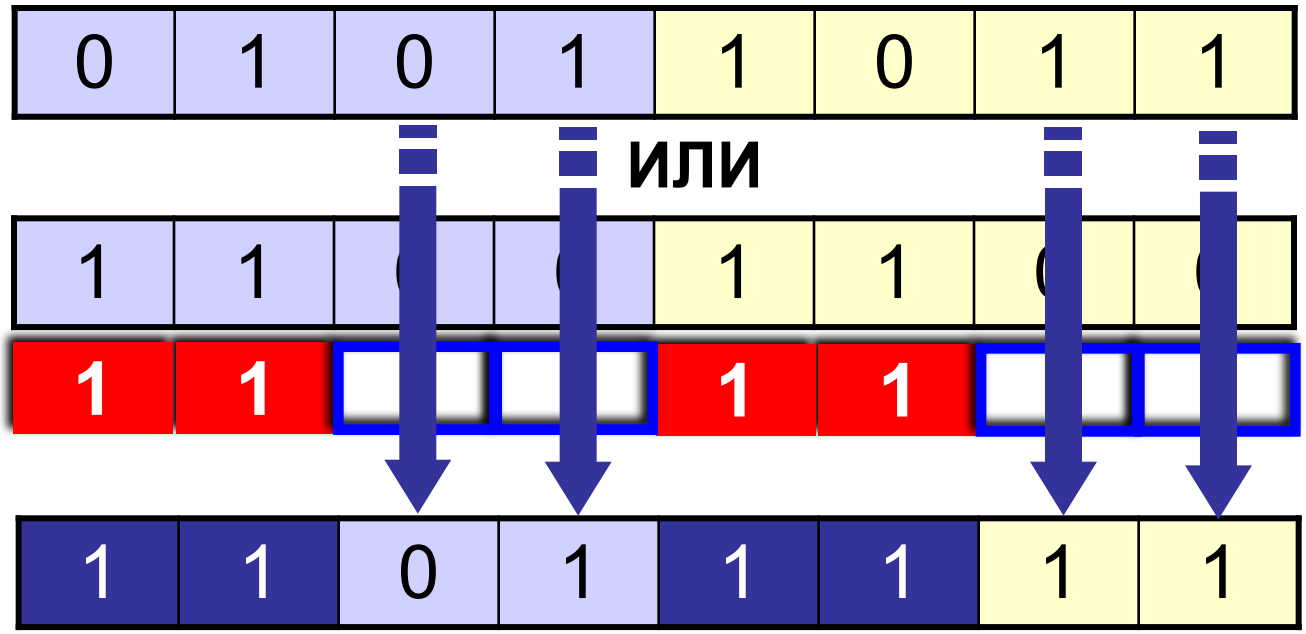

```
if ( n & 0x3C == 0 )
    print (“Биты 2-5 нулевые.”);
else
    print(“В битах 2-5 есть ненулевые.”);
```

Операция ИЛИ

A	B	A или B
0	0	0
0	1	1
1	0	1
1	1	1

Обозначения:

ИЛИ, ∨, | (Java)



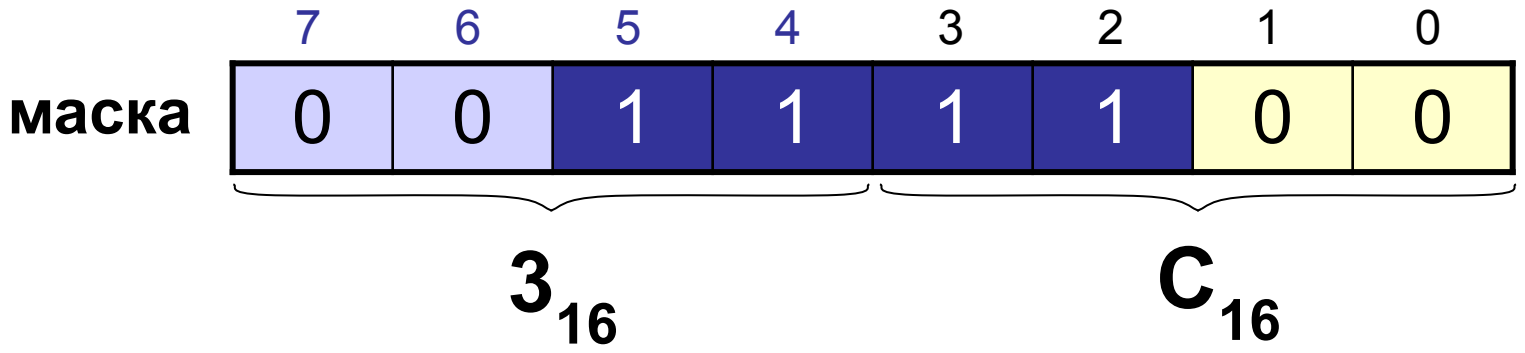
x ИЛИ 0 =x
x ИЛИ 1 =1

маска

$5B_{16} \mid CC_{16} = DF_{16}$

Операция ИЛИ – установка битов в 1

Задача: установить все биты 2…5 равными 1, не меняя остальные.




```
n = n | 0x3C;
```

Операция «исключающее ИЛИ»

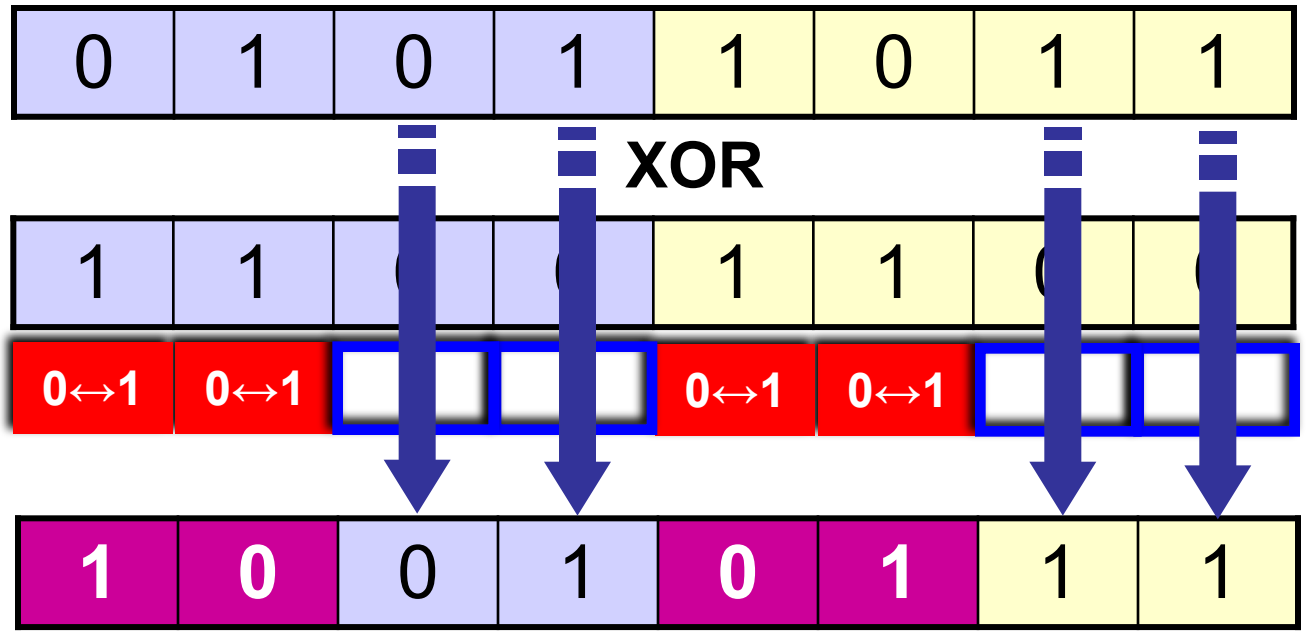
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Обозначения:

⊕, XOR, ^ (Java)

x XOR 0 = x

x XOR 1 = НЕ x

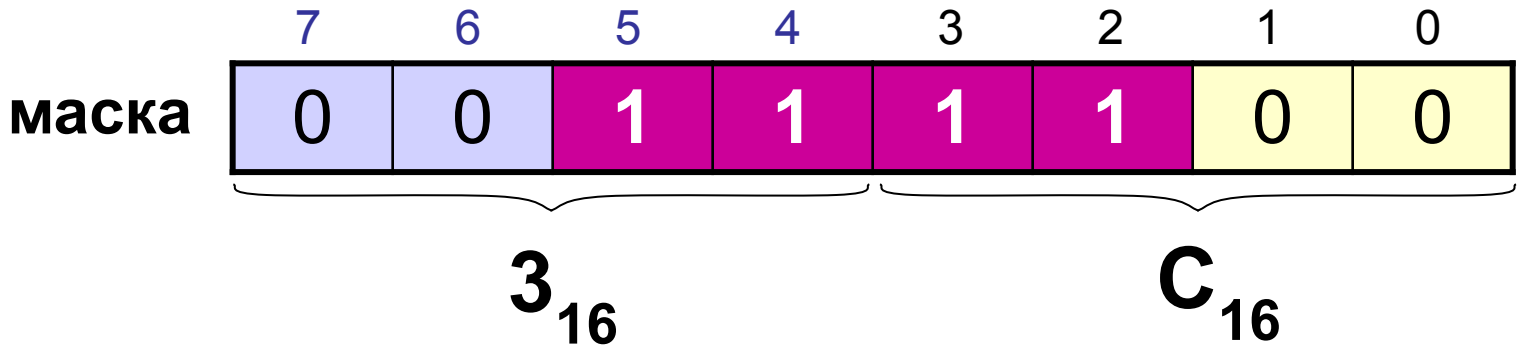


маска

$5B_{16}$ ^ CC_{16} = 97_{16}

«Исключающее ИЛИ» – инверсия битов

Задача: выполнить инверсию для битов 2…5, не меняя остальные.




```
n = n ^ 0x3C;
```

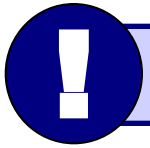
«Исключающее ИЛИ» – шифровка

```
(0 xor 0) xor 0 = 0
(1 xor 0) xor 0 = 1
```

```
(0 xor 1) xor 1 = 0
(1 xor 1) xor 1 = 1
```

код (шифр)

```
(X xor Y) xor Y = X
```



«Исключающее ИЛИ» – обратимая операция

Шифровка:

выполнить для каждого байта текста операцию XOR с байтом-шифром.

Расшифровка: сделать то же самое с тем же шифром.

Побитовые логические операции. Пример

```
int a = 3;
int b = 6;

int c = a | b;

int d = a & b;

int e = a ^ b;

int f = (~a & b) | (a & ~b);

int g = ~a & 0x0f;
```

0011

0110

7 (0111_2)

2 (0010_2)

5 (0101_2)

5 (0101_2)

12 (1100_2)

15 (1111_2)

Битовые сдвиги

При сдвиге значения битов копируются в соседние биты по направлению сдвига.

В зависимости от обработки крайних битов различают следующие сдвиги:

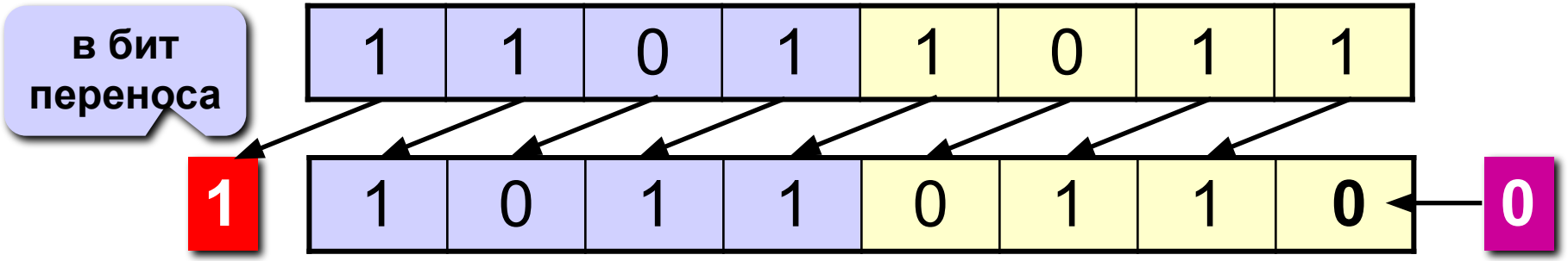
- **логический;**
- **циклический;**
- **арифметический.**

Логический сдвиг

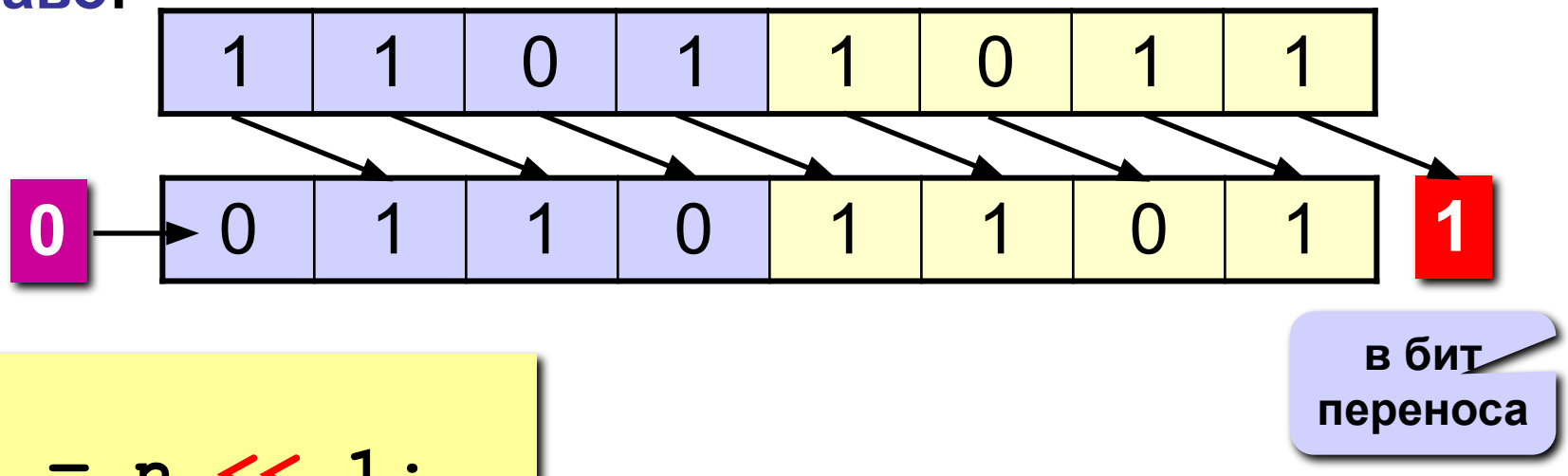
При логическом сдвиге значение последнего бита по направлению сдвига теряется (копируясь в бит переноса), а первый приобретает нулевое значение.

Логический сдвиг

Влево:



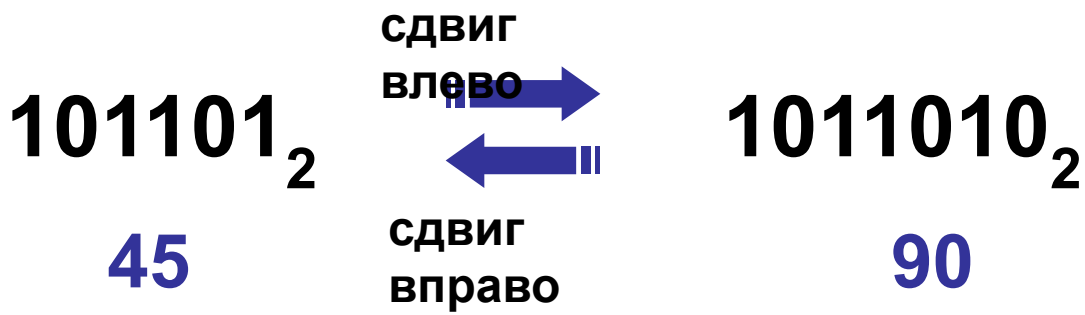
Вправо:




```
n = n << 1;
n = n >>> 1;
```

Логический сдвиг

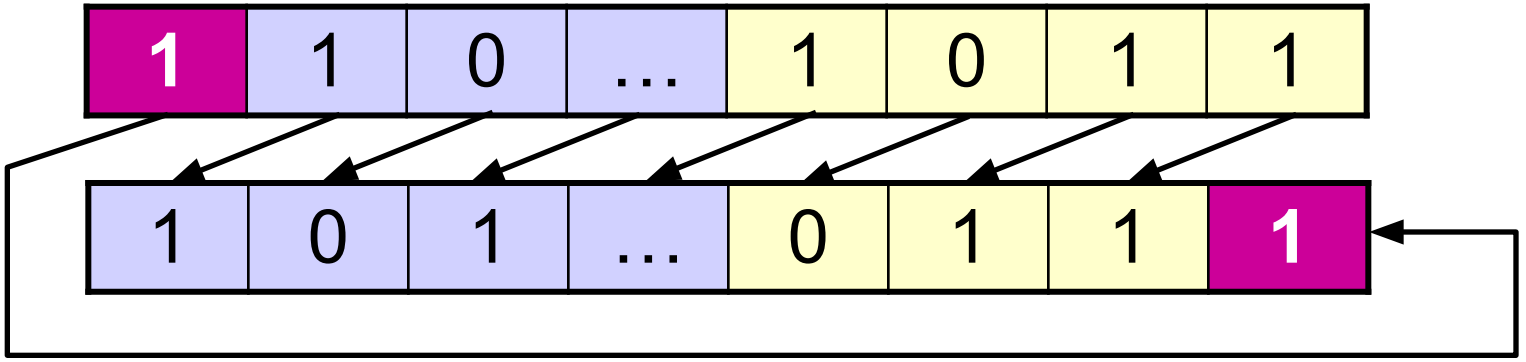
Какой арифметической операции равносилен логический сдвиг влево (вправо)? При каком условии?



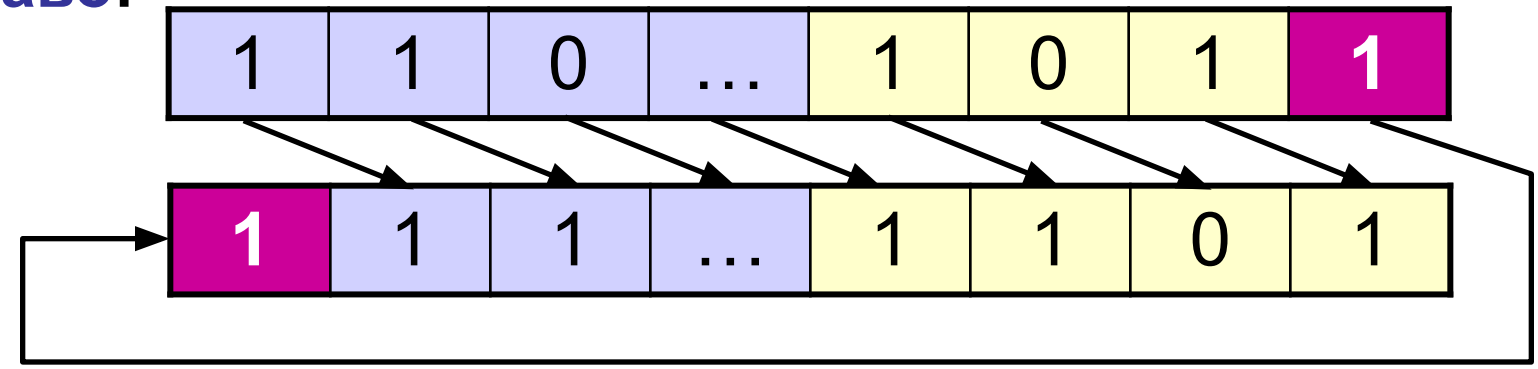
Логический сдвиг влево (вправо) – это быстрый способ умножения (деления без остатка) на 2.

Циклический сдвиг

Влево:



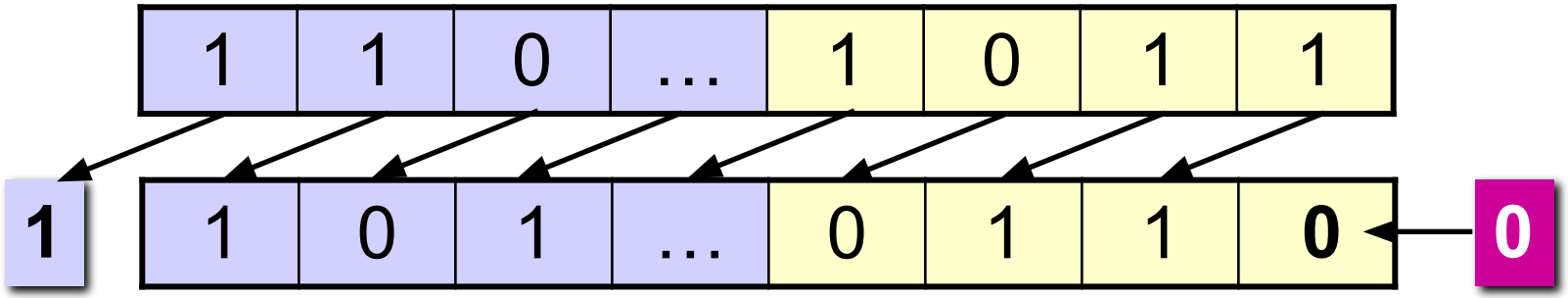
Вправо:



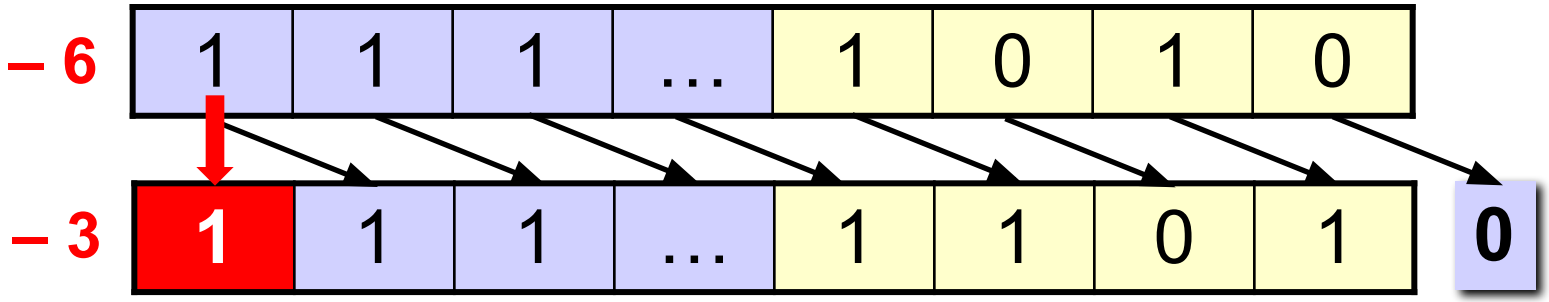
В языке `Java` циклический сдвиг не реализован

Арифметический сдвиг

Влево (= логическому):



Вправо (знаковый бит не меняется!):




```
n = -6;
n = n >> 1;
```

-3

Задача

Задача: в целой переменной n (32 бита) закодирована информация о цвете пикселя в RGB:

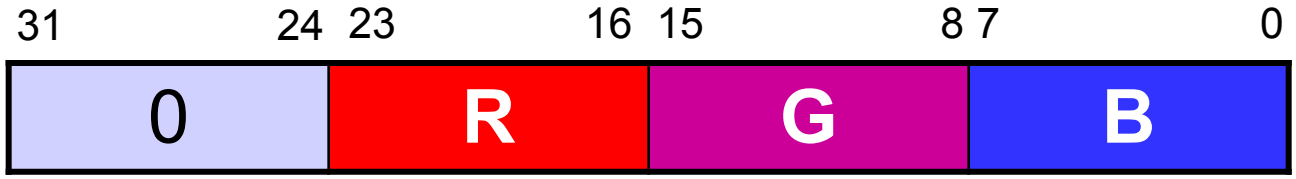
Выделить в переменные R, G, B составляющие цвета.

Вариант 1:

1. Обнулить все биты, кроме G.
   Маска для выделения G: $0000FF00_{16}$
2. Сдвинуть вправо так, чтобы число G передвинулось в младший байт.

```
G = (n & 0xFF00) >> 8;
```

Задача

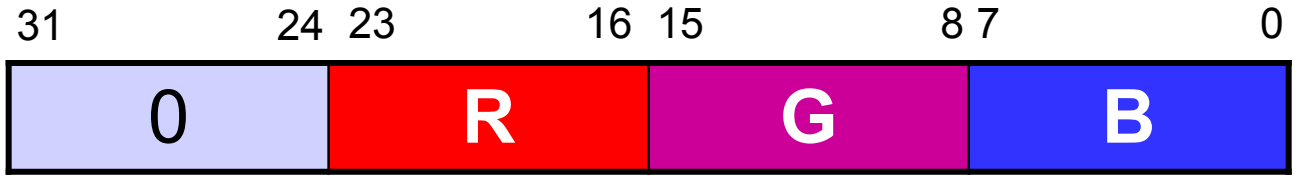


Вариант 2:

1. **Сдвинуть вправо так, чтобы число G передвинулось в младший байт.**
2. **Обнулить все биты, кроме G.**
   Маска для выделения G: $000000FF_{16}$

```
G = (n >> 8) & 0xFF;
```

Задача




```
R =

B =
```

Автоматическое повышение типов

Внимание! При работе с типами `byte, short, int` происходит автоматическое повышение типа до

```
byte a = 64, b, c;
int i;
i = a << 2;
b = (byte) (a << 2);
c = a << 2;
```

256

0

Ошибка!
Несоответствие типов

Задачи

1. Запишите x&y, используя | и ~
2. Выведите n-ый байт заданного числа x (нумерация справа налево, начиная с 0)
3. Запишите наименьшее отрицательное число в дополнительной кодировке, не используя Integer.MIN_VALUE
4. Выясните достаточно ли n бит для представления числа x
5. Вычислите $x/2^n$ и $x*2^n$, не используя операции умножения и деления
6. Дано число x, вычислите -x без обращения знака
7. Выясните является ли число x неотрицательным

Пример задачи на экзамене – 1

Запишите значения x, y и что будет выведено на экран

```
double x = 2./0;
double y = -1/0.;
System.out.print(x+y);
```

Пример задачи на экзамене – 2

Запишите значение b и что будет выведено на экран

```
long m = -130;
byte b = (byte)-m;
System.out.print("b" + b);
```

Пример задачи на экзамене – 3

Запишите значения b, c и что будет выведено на экран

```
int a = -125;
int b = (a>>2);
int c = (a<<2);
System.out.println((byte)(b + c));
```

Пример задачи на экзамене – 4

Запишите значения b, c и что будет выведено на экран

```
int a = -10;
int b = (a >>> 2);
int c = (a << 2);
System.out.println((byte) (b + c));
```

Пример задачи на экзамене – 5

Запишите значения b, c и что будет выведено на экран

```
int a = -10;
int b = ~256;
int c = a ^ b;
System.out.println((byte) c);
```