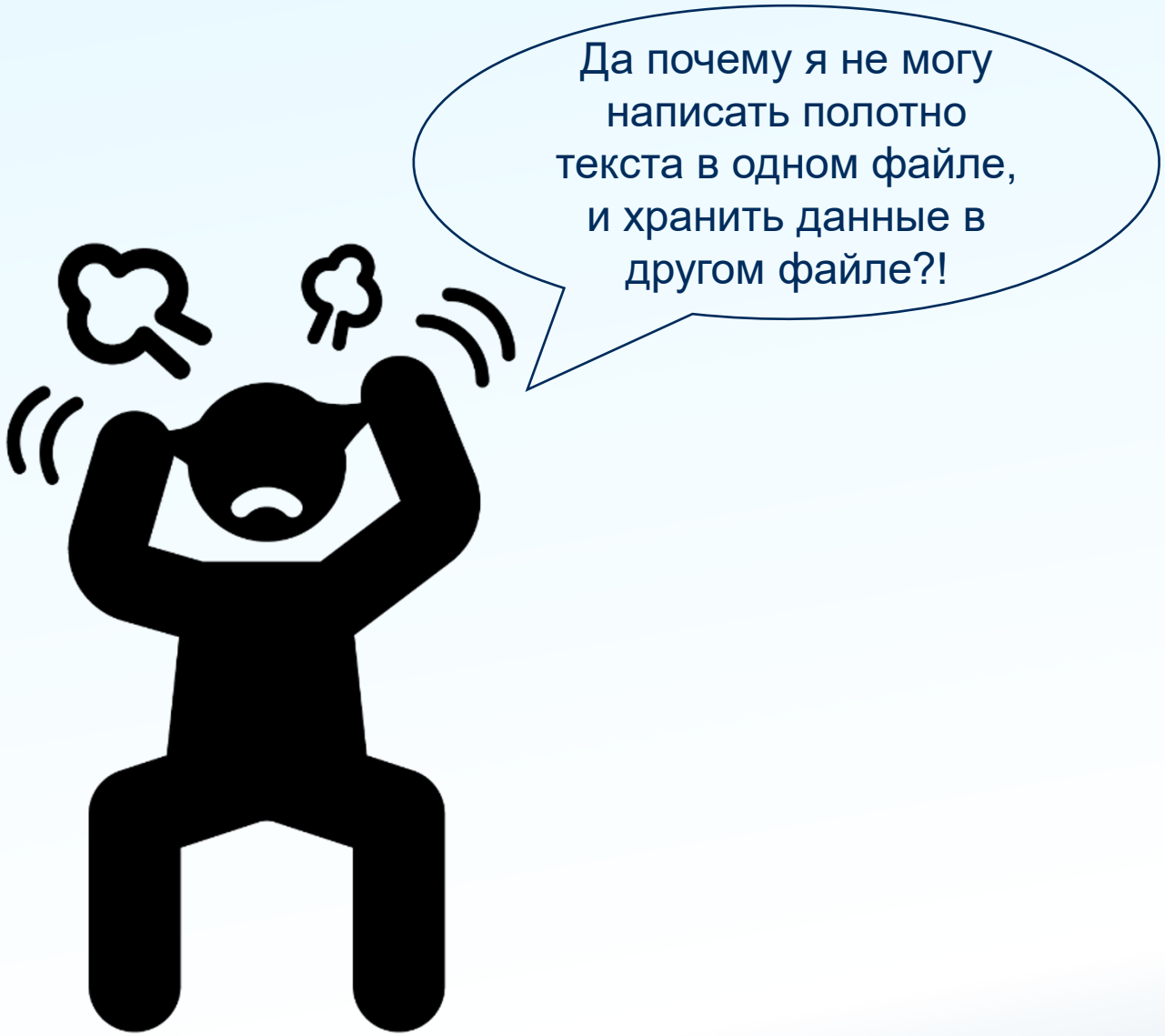
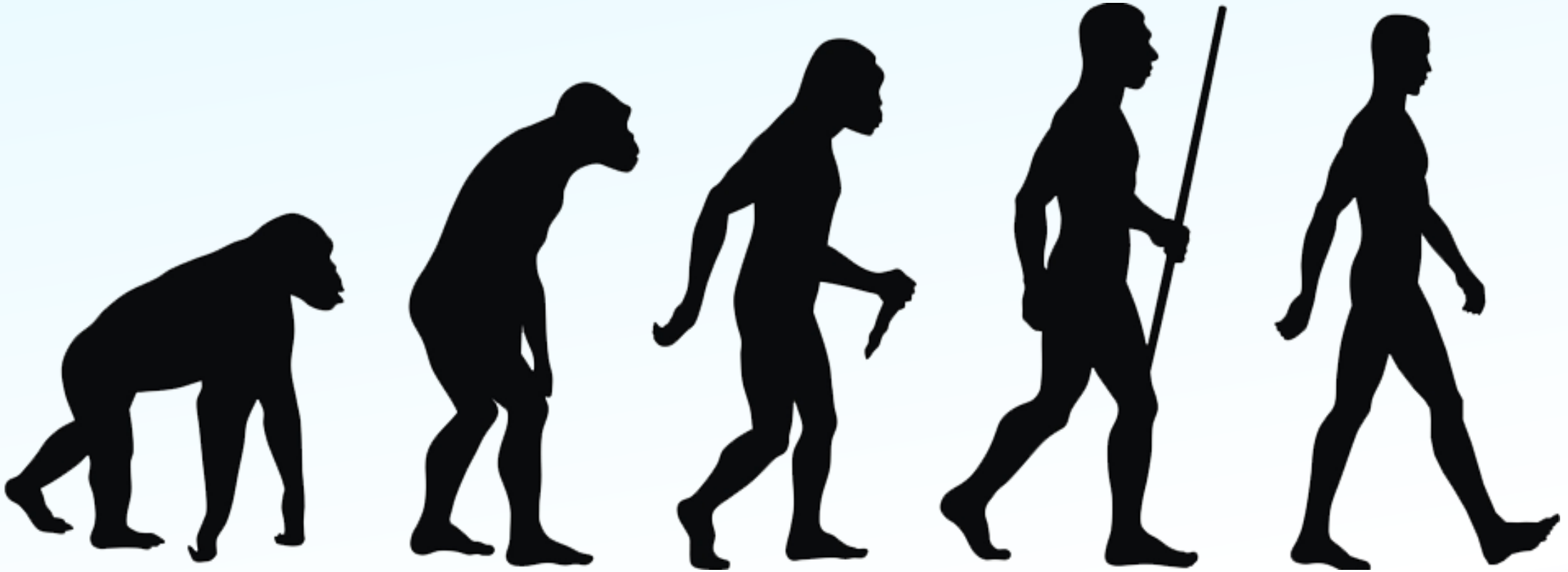
Технологии программирования

Введение


Да почему я не могу
написать полотно
текста в одном файле,
и хранить данные в
другом файле?!

О чем будем говорить в семестре?

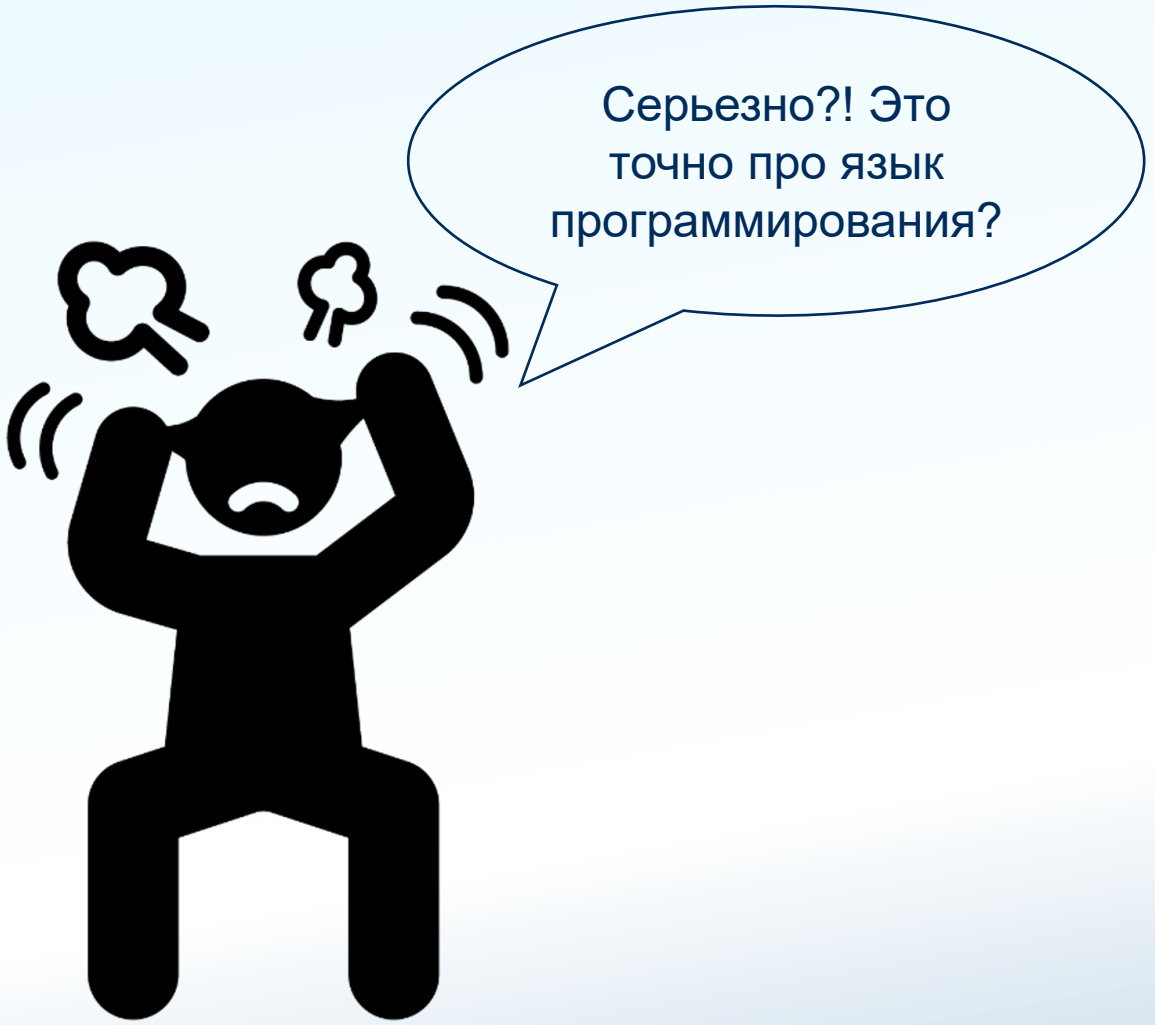
– история языков программирования;
– короткое введение в Java;
– системы сборки;
– взаимодействие с базами данных;
– взаимодействие по сети;
– фреймворк Spring;
– аспектно-ориентированное программирование;
– безопасность;
– память и сборка мусора;
– брокеры сообщений;



Эволюция языков программирования

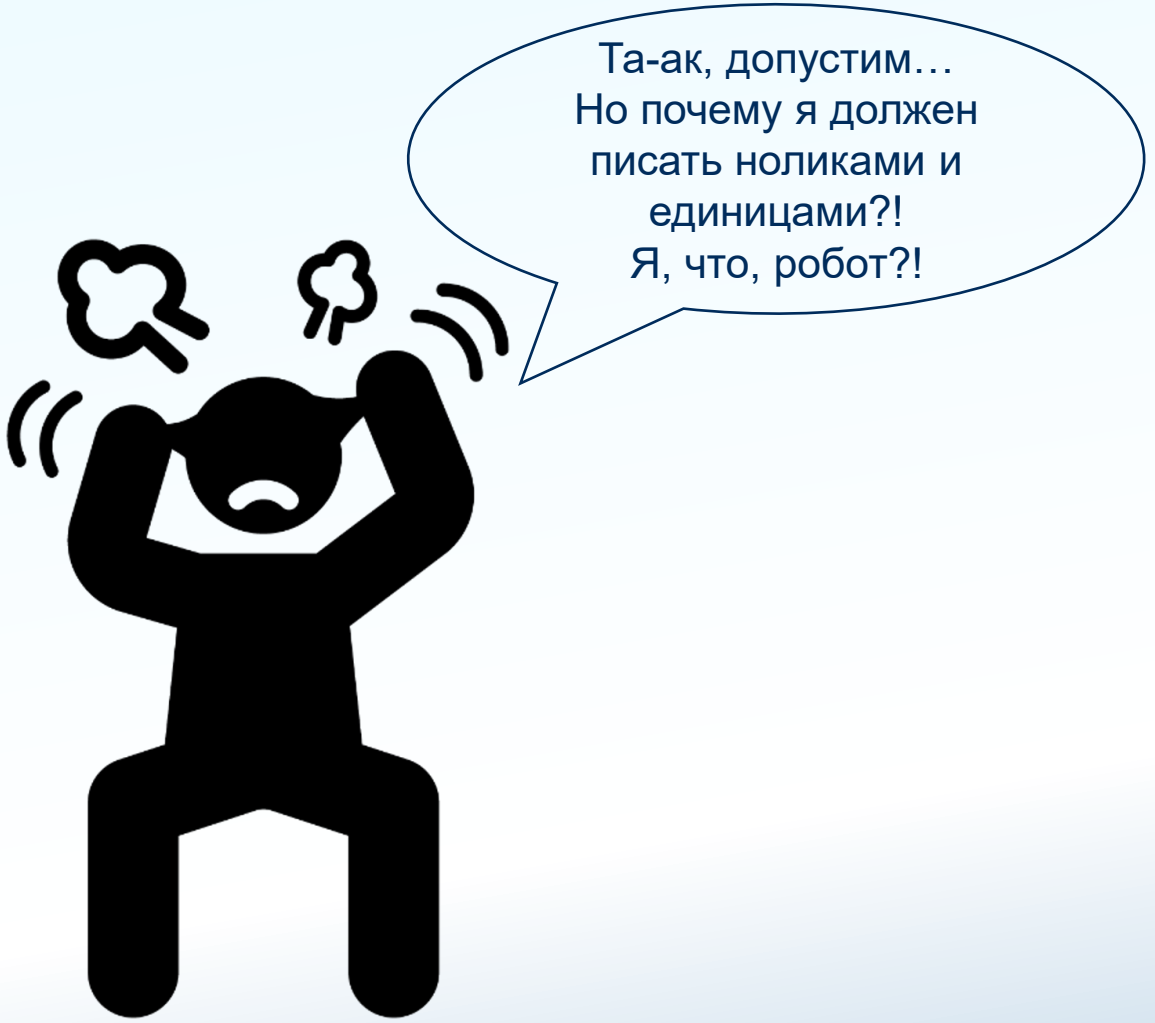
Перфокарты

- если очень грубо, то это определенным образом продырявленный кусок картона;
- наиболее распространенный формат от IBM: длина ≈ 18,7 см, ширина ≈ 8,2 см;
- использовались в первых «компьютерах»: аналитической машине Чарлза Бэббиджа, описанной им в 1851 году, а изобретенной его сыном Генри в 1906 году, и интеллектуальных машинах Семена Николаевича Корсакова в 1832 году;
- изначально изобретены для ткацких станков.



1GL, машинный язык или двоичный язык

- представляет из себя набор операций и ячеек памяти, но записанный только нулями и единицами;
- иногда «кодом» выступали физические переключатели на корпусе ЭВМ, (но данные все же уже вводились отдельно), *правда через перфокарты*;
- самые быстрые языки программирования, так как «код» интерпретируется сразу самим CPU, именно к этому виду компиляторы и интерпретаторы, трансляторы, ассемблеры сводят код, написанный на любом другом языке программирования;
- специфичен для архитектуры процессора;
- появились в начале 1940-х.

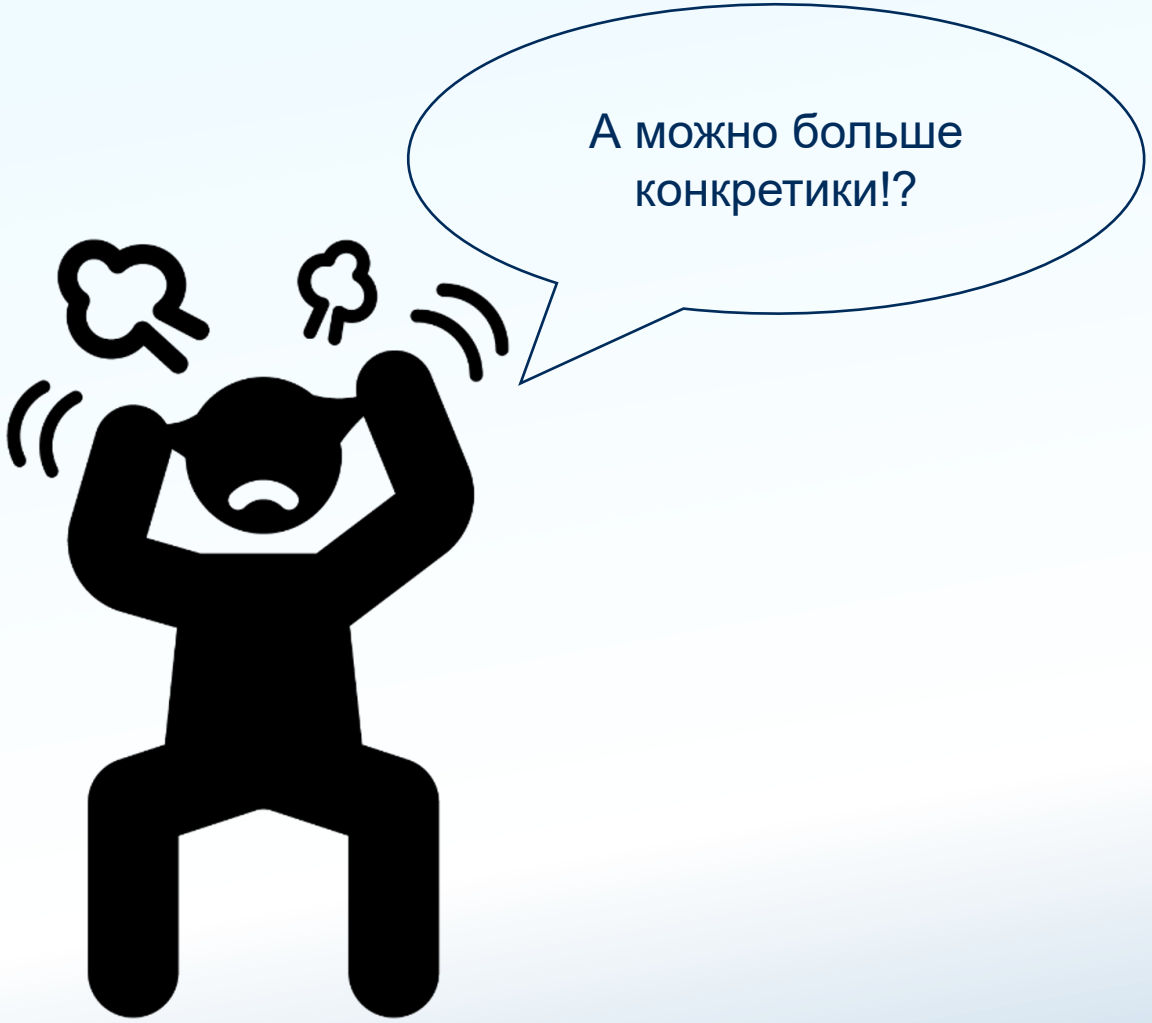


2GL или ассемблерный код

- почти тоже самое, что 1GL, но избавленный от необходимости писать только нулями и единицами, предлагающий «человеческие» названия для операций, являющиеся сокращением от английских слов;
- ячейки памяти начали «обзывать» цифрами в шестнадцатеричной системе счисления;
- процесс перевода «человеческих» команд и ячеек памяти в двоичную систему счисления выполняется служебной программой, называемой ассемблером;
- разработан Эндрю и Кэтлин Бут в 1947 году;
- все еще специфичен для архитектуры процессора.

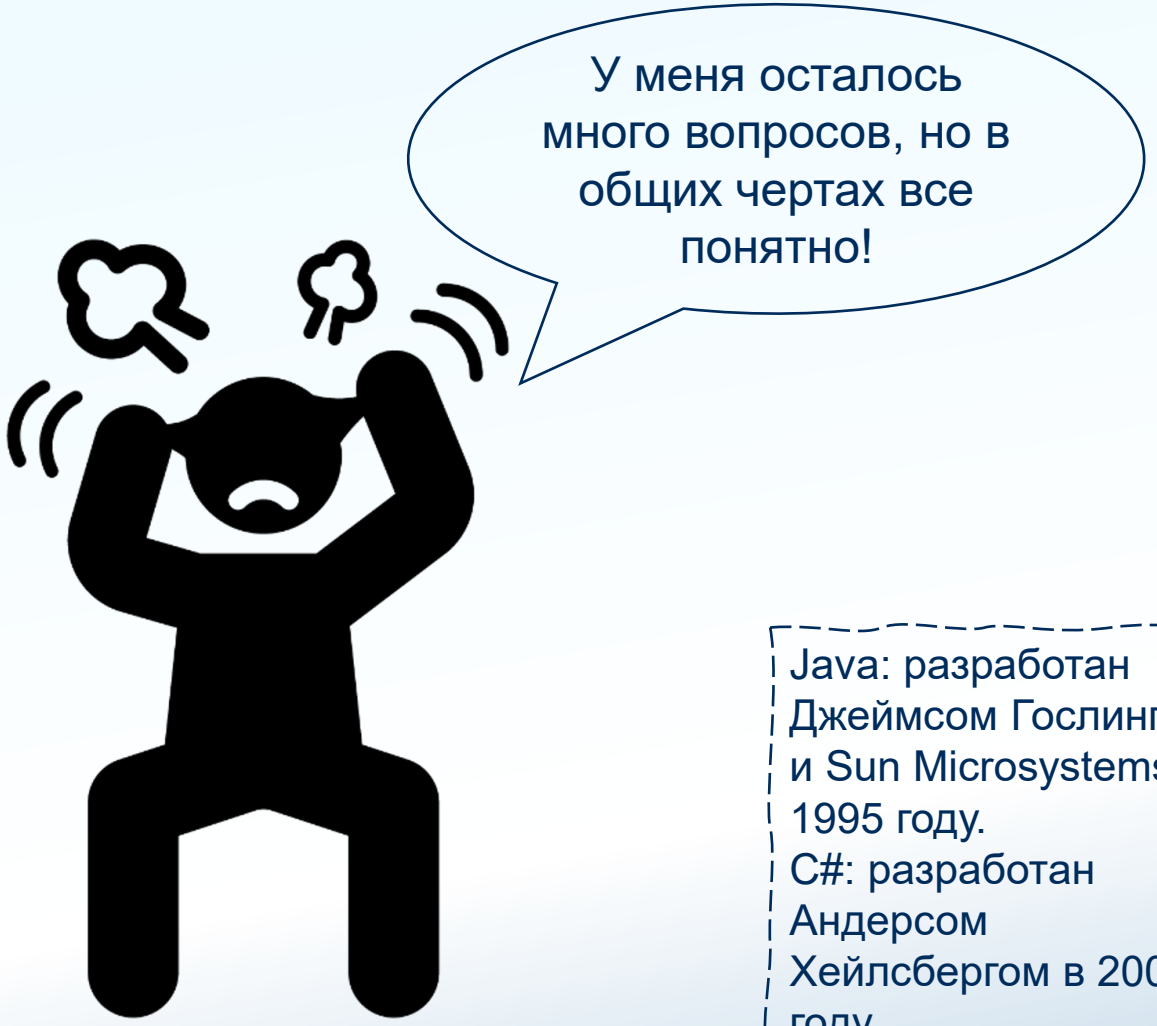
3GL или высокоуровневые языки программирования

- к языкам третьего уровня относятся все языки, кроме обозначенных выше;
- благодаря более высокому уровню абстракции пропала зависимость от архитектуры процессора, языки стали еще более «человеческими», а до машинного кода появилась еще одна прослойка преобразования исходного кода: компиляция, интерпретация или совместно, *правда компиляторы и интерпретаторы остались специфичными от архитектуры процессора*;
- первым языком третьего поколения стал Fortran в 1957 году.



Эволюция высокоуровневых языков программирования

- C: мощь, эффективность, структурированность и относительно легкое изучение, разработан Денисом Ритчи в 1972 году;
- C++: C + возможности ООП, разработан Бьерном Страуструпом в 1983 году;
- Java / C#: наследники C++, но менее эффективные, но более кроссплатформенные за счет **байт-кода**, в который компилируется исходный код, и его интерпретации на **виртуальной машине**.



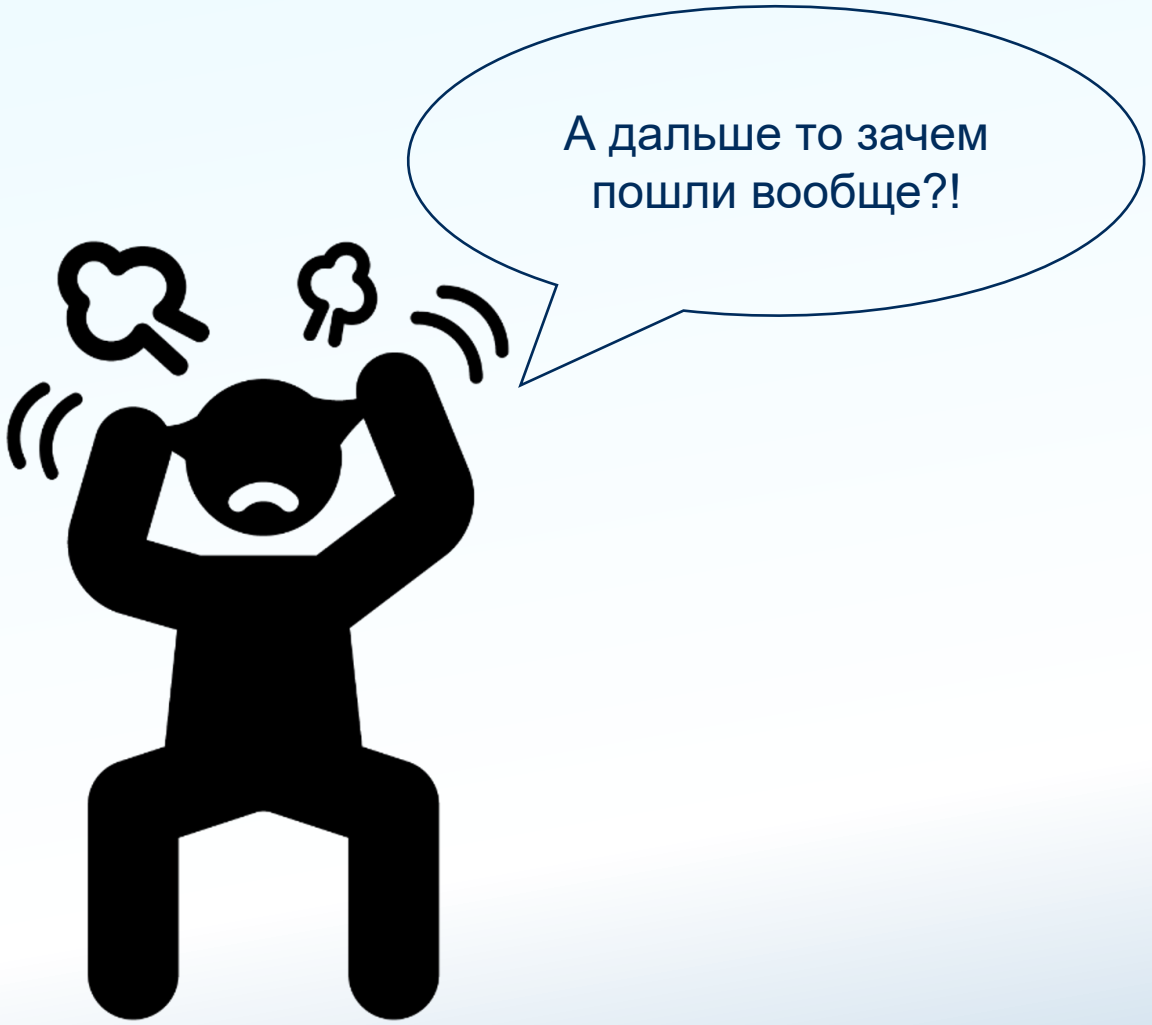
Java: разработан Джеймсом Гослингом и Sun Microsystems в 1995 году.
C#: разработан Андерсом Хейлсбергом в 2001 году.

4GL или узкоспециализированные языки программирования

Приводимые всюду примеры: JavaScript, Python, SQL.

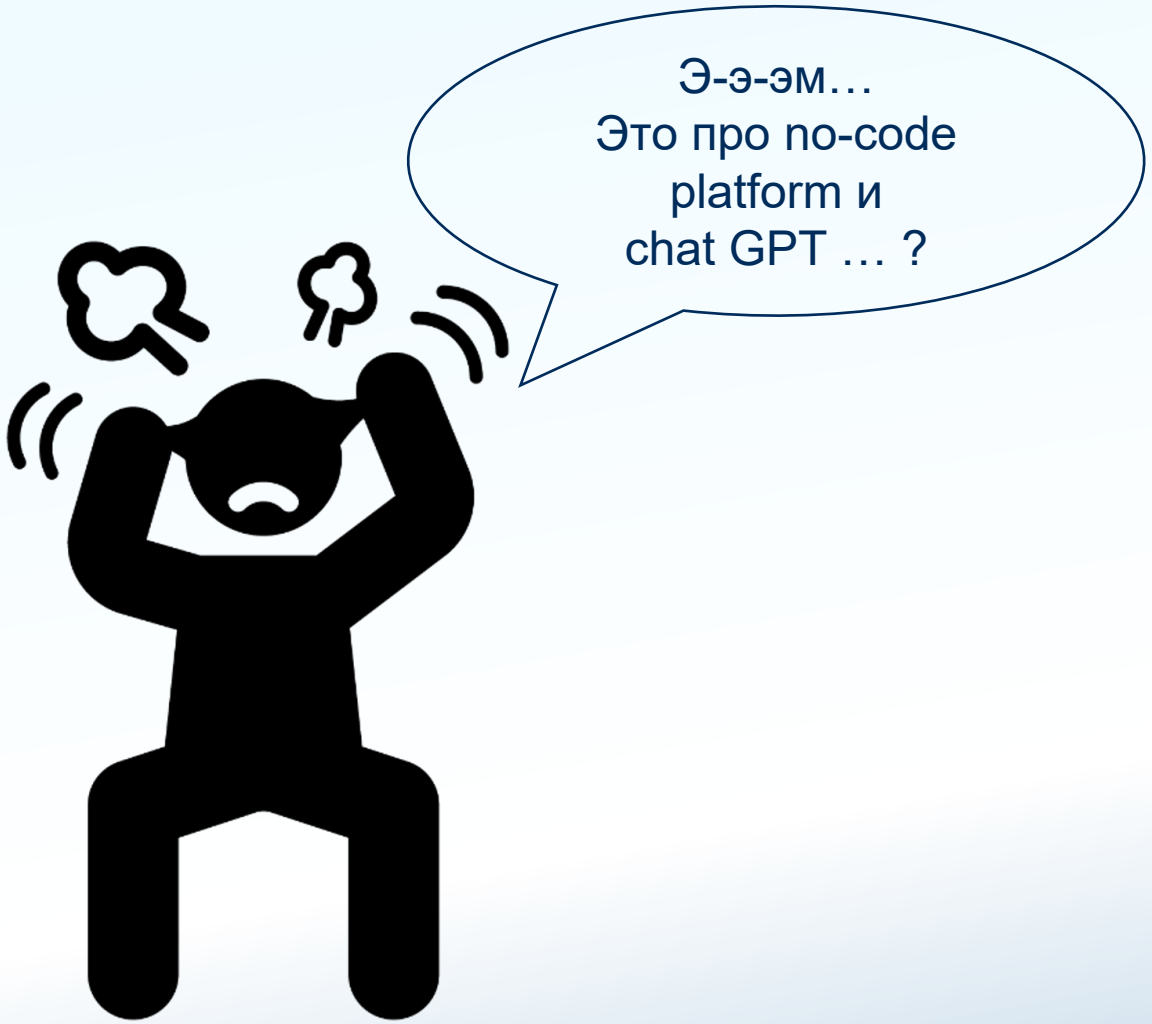
Осталось только понять, если они такие узкоспециализированные, то почему на JavaScript и Python пишут все, что угодно.

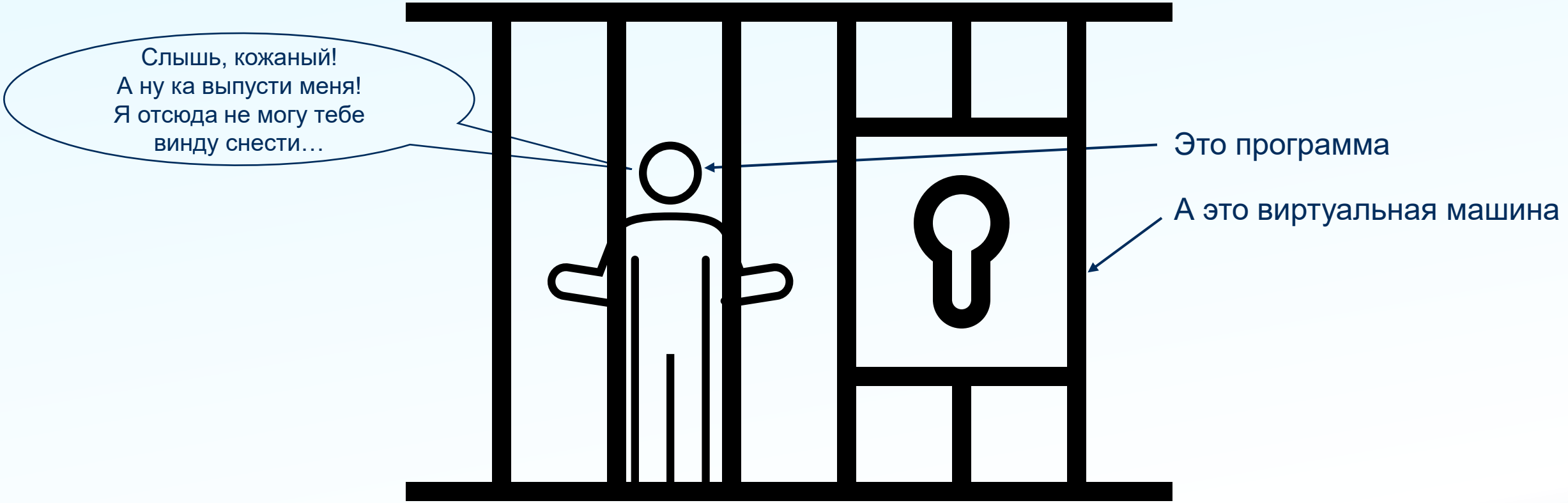
А еще конечно вопрос, что в классификации языков программирования забыл язык запросов SQL?



5GL

Большего, чем на картинке мне тут сказать особо и нечего)





Байт-код и виртуальные машины

Интерпретатор

Это программа, разновидность транслятора, построчно «читающая», обрабатывающая и выполняющая его.

Например, программу с консольным интерфейсом очень грубо можно назвать интерпретатором команд пользователя. По той причине, что «правильный» интерпретатор делает фактически тоже самое, только:

– принимает не одну команду, а все сразу в виде текстового документа;
– командами для интерпретатора выступает синтаксис языка программирования, который он интерпретирует.

Очевидно, что такие языке намного медленнее компилируемых.

Первым интерпретируемым языком программирования был Lisp, разработанный Джоном Маккарти в 1958 году.

Байт-код

Это код, в который был скомпилирован исходный код, написанный программистом.

Цель использования байт-кода в том, чтобы интерпретировать не исходный код какого-то языка, а нечто более приближенное к машинному коду, и соответственно, намного более быстро выполняемое интерпретатором, потому что сам интерпретатор можно также «приблизить» к процессору.

Итог: исходный код компилируется в байт-код, и уже он интерпретируется.

Благодаря этому получаем плюсы и компилируемых, и интерпретируемых языков. *Минусы правда тоже получаем и тех, и других, но они очень зависимы от контекста*.

Виртуальная машина

Собственно интерпретатором байт-кода выступает виртуальная машина.

Преимущества:

– кроссплатформенность – программа может работать на любом компьютере, где установлена виртуальная машина;
– безопасность – как бы разработчик ни старался, все, к чему его программа может получить доступ, это ресурсы, предоставленные виртуальной машине.

Недостатки:

– у самой виртуальной машины, как таковых нет, инструмент очень крутой и в свое время прорывной.

Исходный код

```
package org.example;

public class Human {

    private String name;
    private String surname;
    private int age;

    public Human(String name, String surname, int age) {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public int getAge() {
        return age;
    }

    public void changeName(String newName) {
        if (newName.isBlank()) {
            throw new RuntimeException("invalid name");
        }
        this.name = newName;
    }
}
```

Байт-код

```
// class version 65.0 (65)

// access flags 0x21
public class org/example/Human {

  // compiled from: Human.java

  // access flags 0x2
  private Ljava/lang/String; name

  // access flags 0x2
  private Ljava/lang/String; surname

  // access flags 0x2
  private I age

  // access flags 0x1
  public <init>(Ljava/lang/String;Ljava/lang/String;I)V
    L0
      LINENUMBER 9 L0
      ALOAD 0
      INVOKESPECIAL java/lang/Object.<init> ()V
    L1
      LINENUMBER 10 L1
      ALOAD 0
      ALOAD 1
      PUTFIELD org/example/Human.name : Ljava/lang/String;
    L2
      LINENUMBER 11 L2
      ALOAD 0
      ALOAD 2
      PUTFIELD org/example/Human.surname : Ljava/lang/String;
    L3
      LINENUMBER 12 L3
      ALOAD 0
      ILOAD 3
      PUTFIELD org/example/Human.age : I
    L4
      LINENUMBER 13 L4
      RETURN
    L5
      LOCALVARIABLE this Lorg/example/Human; L0 L5 0
      LOCALVARIABLE name Ljava/lang/String; L0 L5 1
      LOCALVARIABLE surname Ljava/lang/String; L0 L5 2
      LOCALVARIABLE age I L0 L5 3
      MAXSTACK = 2
      MAXLOCALS = 4


  // access flags 0x1
  public getName()Ljava/lang/String;
    L0
      LINENUMBER 16 L0
      ALOAD 0
      GETFIELD org/example/Human.name : Ljava/lang/String;
      ARETURN
    L1
      LOCALVARIABLE this Lorg/example/Human; L0 L1 0
      MAXSTACK = 1
      MAXLOCALS = 1

  // access flags 0x1
  public getSurname()Ljava/lang/String;
    L0
      LINENUMBER 20 L0
      ALOAD 0
      GETFIELD org/example/Human.surname : Ljava/lang/String;
      ARETURN
    L1
      LOCALVARIABLE this Lorg/example/Human; L0 L1 0
      MAXSTACK = 1
      MAXLOCALS = 1


  // access flags 0x1
  public getAge()I
    L0
      LINENUMBER 24 L0
      ALOAD 0
      GETFIELD org/example/Human.age : I
      IRETURN
    L1
      LOCALVARIABLE this Lorg/example/Human; L0 L1 0
      MAXSTACK = 1
      MAXLOCALS = 1


  // access flags 0x1
  public changeName(Ljava/lang/String;)V
    L0
      LINENUMBER 28 L0
      ALOAD 1
      INVOKEVIRTUAL java/lang/String.isBlank ()Z
      IFEQ L1
    L2
      LINENUMBER 29 L2
      NEW java/lang/RuntimeException
      DUP
      LDC "invalid name“
      INVOKESPECIAL java/lang/RuntimeException.<init>
(Ljava/lang/String;)V
      ATHROW
    L1
      LINENUMBER 31 L1
      ALOAD 0
      ALOAD 1
      PUTFIELD org/example/Human.name : Ljava/lang/String;
    L3
      LINENUMBER 32 L3
      RETURN
    L4
      LOCALVARIABLE this Lorg/example/Human; L0 L4 0
      LOCALVARIABLE newName Ljava/lang/String; L0 L4 1
      MAXSTACK = 3
      MAXLOCALS = 2
}
```

Java

- Глава 2. Краткий обзор языка Java.
- Глава 3. Типы данных, переменные и массивы.
- Глава 4. Операции.
- Глава 5. Управляющие операторы.
- Глава 6. Введение в классы.
- Глава 7. Подробный анализ методов и классов.
- **Глава 8. Наследование.**
- Глава 9. Пакеты и интерфейсы.
- **Глава 10. Обработка исключений.**
- Глава 12. Перечисления, автоупаковка и аннотации.
- Глава 13. Ввод-вывод, оператор try с ресурсами и другие темы.
- Глава 14. Обобщения.
- **Глава 15. Лямбда-выражения.**
- Глава 17. Выражения switch, записи и прочие недавно добавленные средства.
- **Глава 18. Обработка строк.**
- Глава 19. Исследование пакета java.Lang.
- **Глава 20. Пакет java.util, часть 1: Collections Framework.**