

# 1. Основни понятия

Лекционен курс “Програмиране на Java”  
проф. д-р Станмир Стоянов

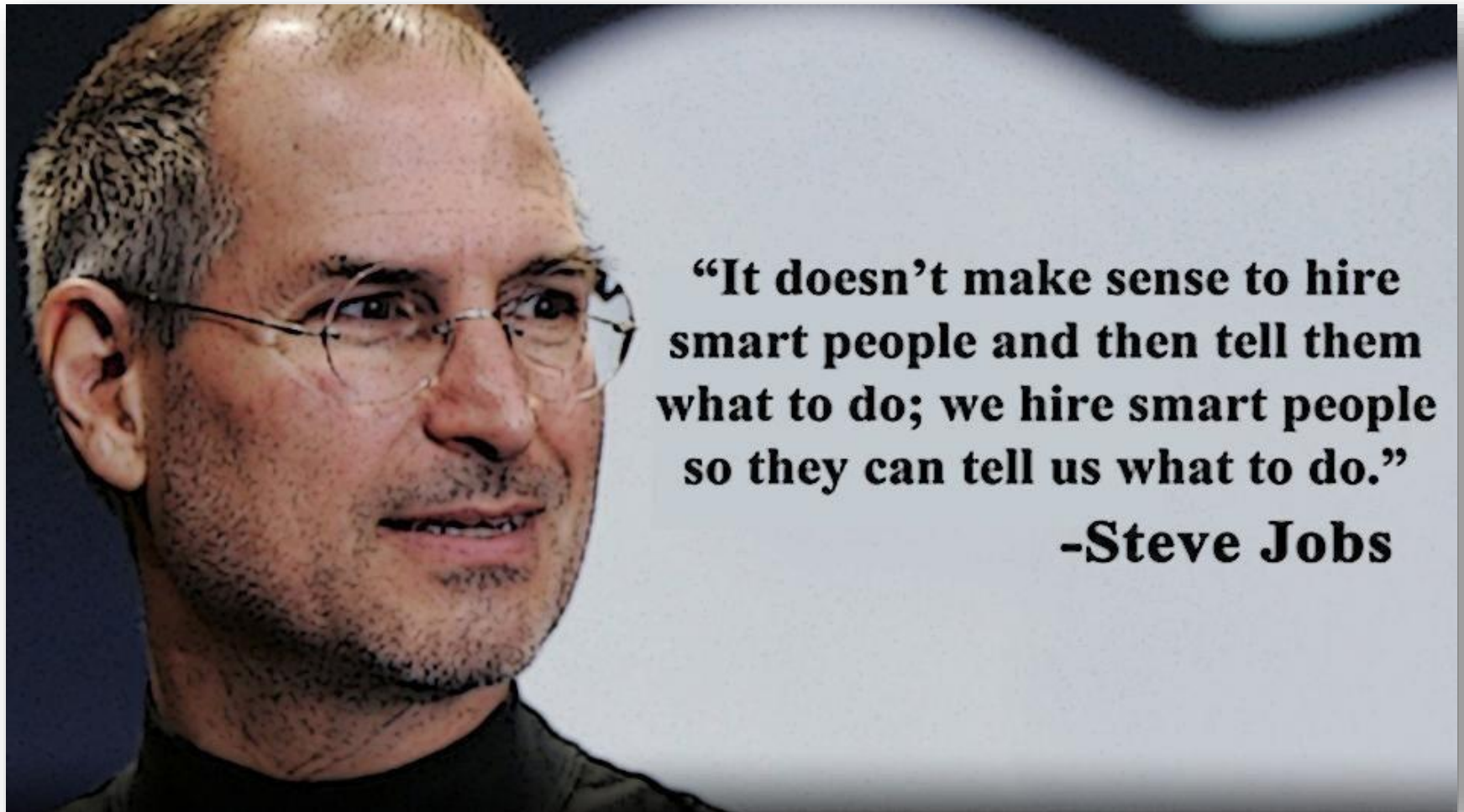
# Структура на лекцията

---

- ▶ Въвеждащ пример
- ▶ Алгоритми
- ▶ Езици за програмиране
- ▶ Формални граматики и езици

# Защо?

---



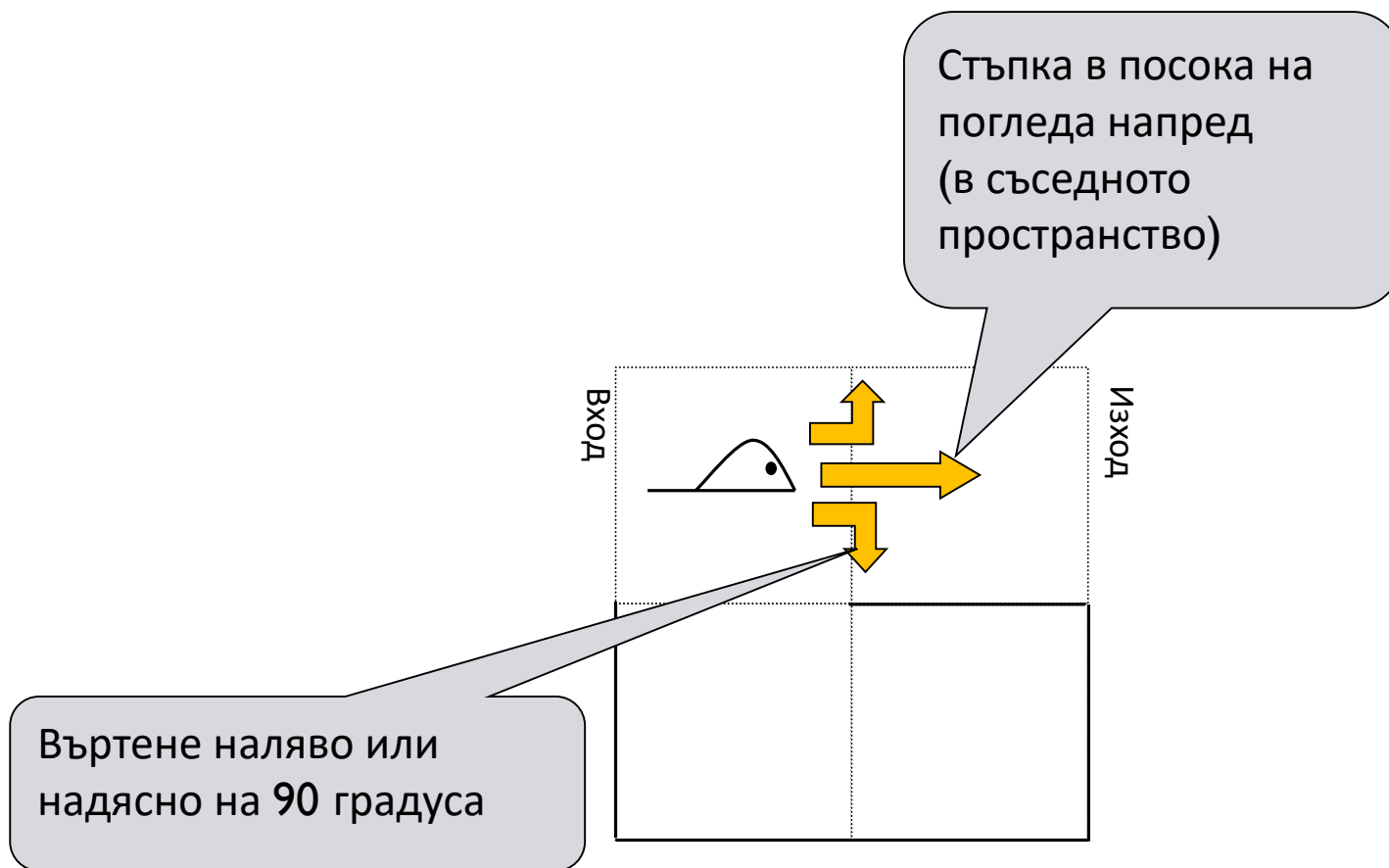
# Въвеждащ пример: 'мишка в лабиринт'

---

- ▶ Задача:

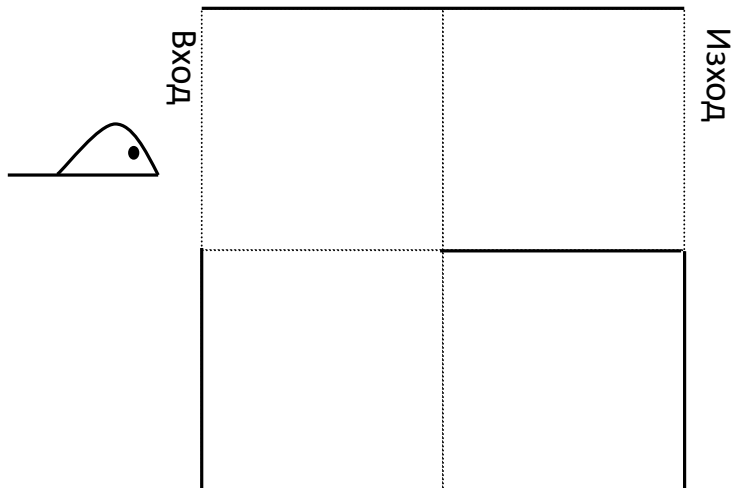
Да се разработи програма, която симулира движението на една мишка през лабиринт от входа до изхода.

# Възможно движение на мишката



# Вариант 1

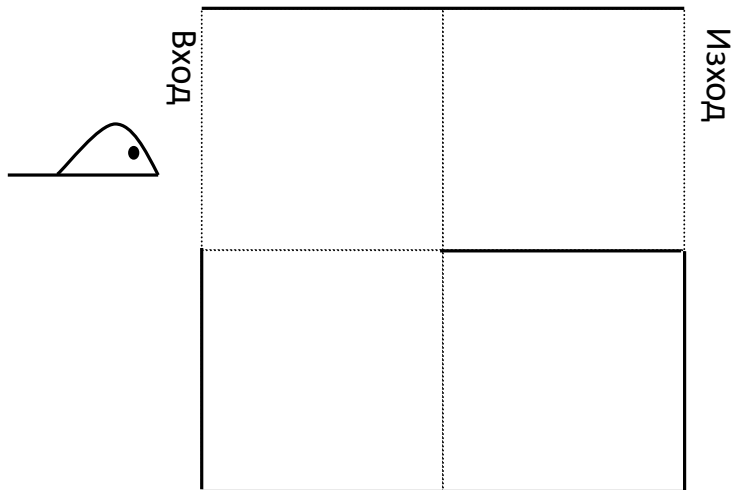
Това алгоритъм ли е ?



```
step forward  
turn right  
step forward  
turn left  
step forward  
turn left  
step forward  
turn right  
step forward
```

## Вариант 2

Това алгоритъм ли е ?



```
step forward  
step forward  
step forward
```

# Алгоритъм

---

- ▶ **Още не е алгоритъм:**

- ▶ Няма метод за постъпково изчисляване последователността от движения на основата на елементарни операции

- ▶ **Принцип:**

- ▶ Движение на мишката винаги така, че стената да се намира от дясната ѝ страна.

- ▶ **Елементарни операции:**

- ▶ `step forward`
- ▶ `turn left`
- ▶ `turn right`
- ▶ `facing a wall?`
- ▶ `outside the Labyrinth?`



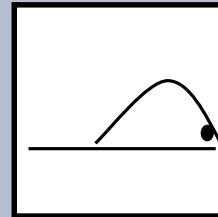
# Алгоритъм: като псевдопрограма

---

```
if мишката е още в лабиринта:
    обърни надясно;
    if (стена по посока на погледа)
        then
            обърни наляво;
            if (стена по посока на погледа):
                then
                    обърни надясно и тествай ... (цикъл)
        else
            направи стъпка напред
```

# По-строг псевдокод: една стъпка

Невъзможно:



```
IF (NOT outside the Labyrinth?)  
  BEGIN /*do next step*/  
    turn right;  
    WHILE (facing a wall?) DO  
      turn left;  
    ENDWHILE  
    step forward;  
  END
```

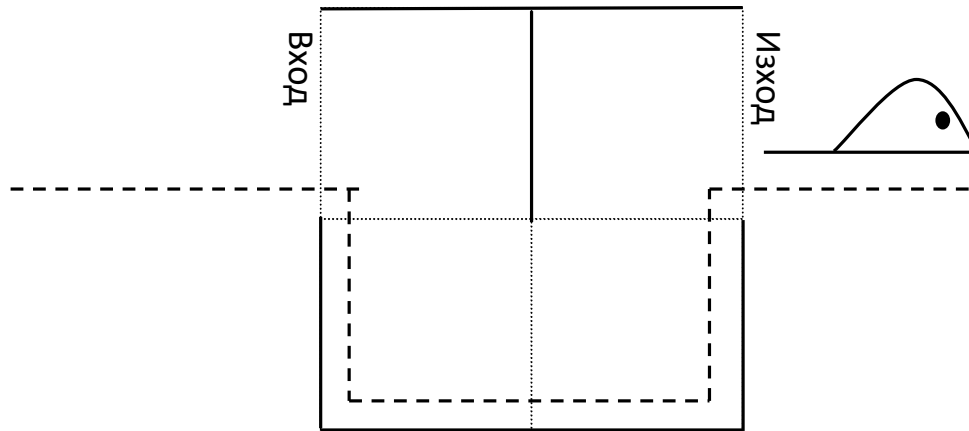
# По-строг псевдокод: общ алгоритъм

---

```
step forward; // by entry
WHILE (NOT outside the Labyrinth?)
  BEGIN // do next step
    turn right;
    WHILE (facing a wall?) DO
      turn left;
    ENDWHILE
    step forward;
  END
ENDWHILE
```

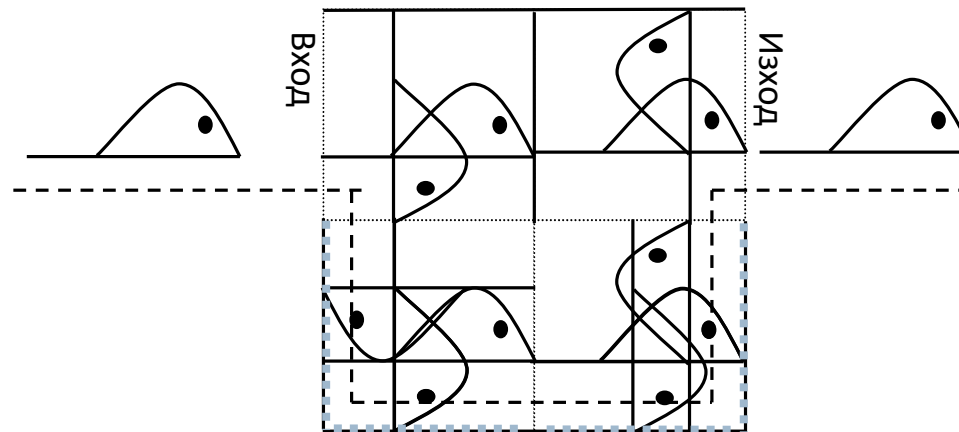
# Движения на мишката, следвайки алгоритъма: първи пример

---



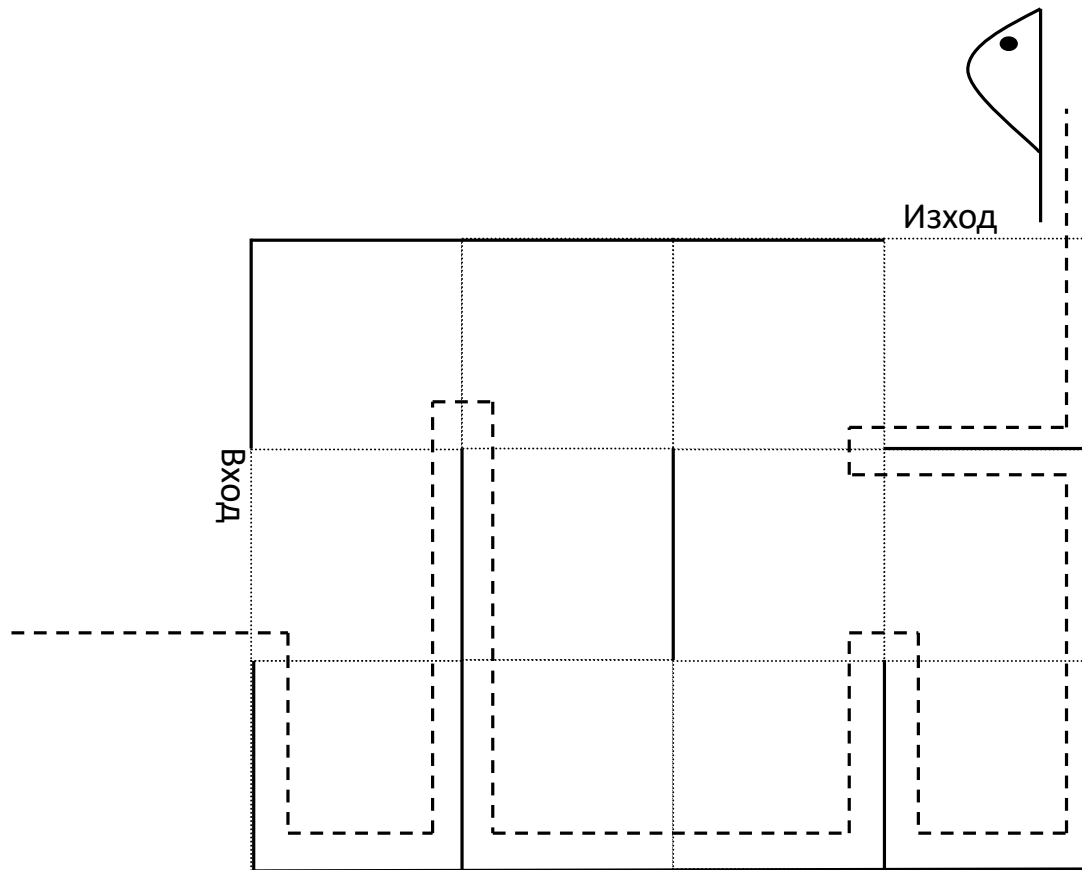
# Движения на мишката, следвайки алгоритъма: първи пример в детайли

---

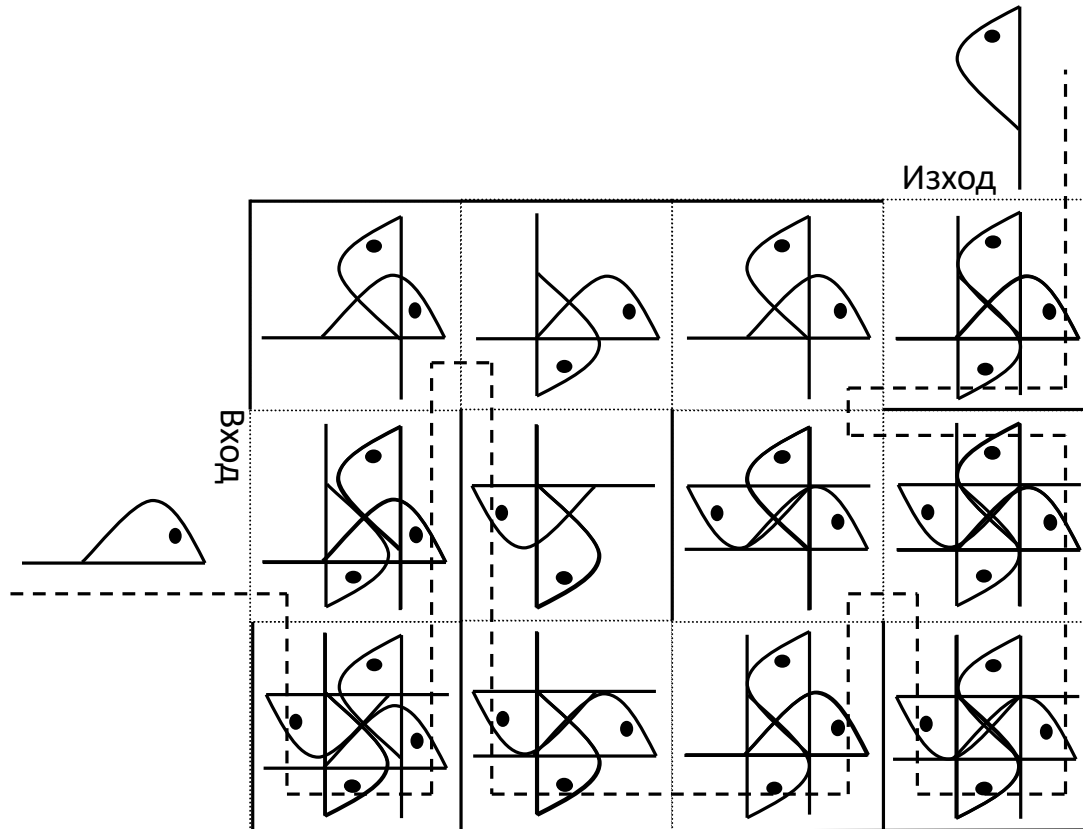


# Движения на мишката, следвайки алгоритъма: втори пример

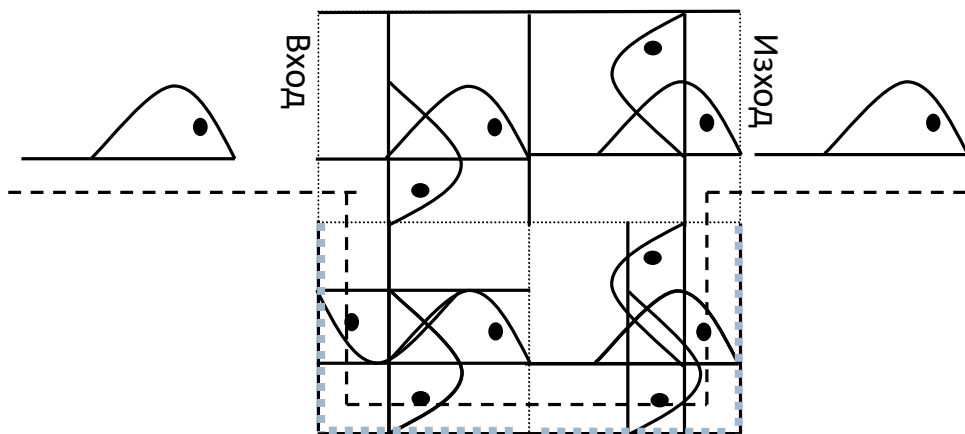
---



## Движения на мишката, следвайки алгоритъма: втори пример в детайли



# Движения на мишката: графично и текстово представяне



```
step forward  
turn right  
step forward  
turn right  
turn left  
turn left  
step forward  
turn right  
turn left  
turn left  
step forward  
turn right  
step forward
```



# Програми решават проблеми

---

Програма = Данни + Алгоритъм

Изчислителна рецепта

Какво е алгоритъм?

# Какво е алгоритъм?

---

## Алгоритъм:

Метод за изчисляване търсени стойности от дадени стойности, който е представен като прецизно крайно описание, базирано на постъпково изпълнение на елементарни (и изчислими) обработващи операции.

# Какво е необходимо?

---

- ▶ Точно предписание за изчисление:
  - ▶ Елементарни отделни стъпки
  - ▶ Последователност от отделни стъпки
  - ▶ Прецизно крайно описание
  - ▶ Крайно изчисление
    - ▶ т.е. прекъсване след краен брой стъпки
- ▶ Дефиницията трябва да се прецизира:
  - ▶ Каква точно информация трябва да се достави за формулиране на конкретни алгоритми?

# Прецизиране на понятието „Алгоритъм“

---

- ▶ Какво са елементарни, изчислими операции?
- ▶ Как могат да бъдат комбинирани?
- ▶ Как се описват?

# Пример за алгоритъм

## Стъпка 0:

Въведи  $n$

## Стъпка 1:

Ако  $n < 0$  :  
изведи "Грешка: отрицателно число" , STOP

## Стъпка 2:

$result := 1$

## Стъпка 3:

Ако  $n = 0$  :  
изведи "резултат:"  $result$ , STOP

## Стъпка 4:

$result := result * n$ ,  
 $n := n - 1$ ,  
по-нататък в стъпка 3

Какво се  
изчислява?

### Проблеми:

- Кои са елементарните операции?
- Как се комбинират?

# Отговори за примера

---

→ Изчислява се:  $n! = 1 * 2 * \dots * n$

→ Елементарни операции:

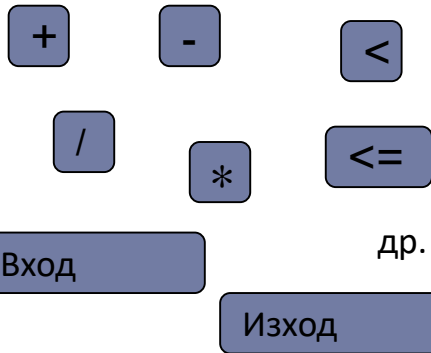
- вход, изход
- проверки:  $< =$
- аритметика:  $*$ ,  $-$
- присвояване на стойност:  $n := n + 1$
- STOP

→ Комбиниране на елементарни оператори:

- последователно изпълнение на стъпките  
(стъпка 0 → стъпка 1 ...)
- условно изпълнение (ако ...)
- повторение на стъпки (стъпка 4 → стъпка 3)

# Изкуството програмиране: разработване на алгоритми . . .

Елементарни операции:



Възможности за комбиниране

Условно изпълнение:

if B ...

Повторение:

while B ...

Последователност:

S1; S2; ...

Алгоритъм:

Стъпка 1:  
Стъпка 2:  
...

Програма:

```
class T {  
    if ...  
    while ...  
}
```

# Форми за представяне на алгоритми

---

- ▶ Текст
- ▶ Диаграма
- ▶ Псевдо-код
- ▶ Компютърна програма
- ▶ ...
- ▶ За абстрактни изследвания (теория на изчислимостта):
  - ▶ Рекурсивни функции
  - ▶ TURING-машини
  - ▶ Крайни автомати
  - ▶ ...



# Проблемни области

---

## ▶ **Представяне:**

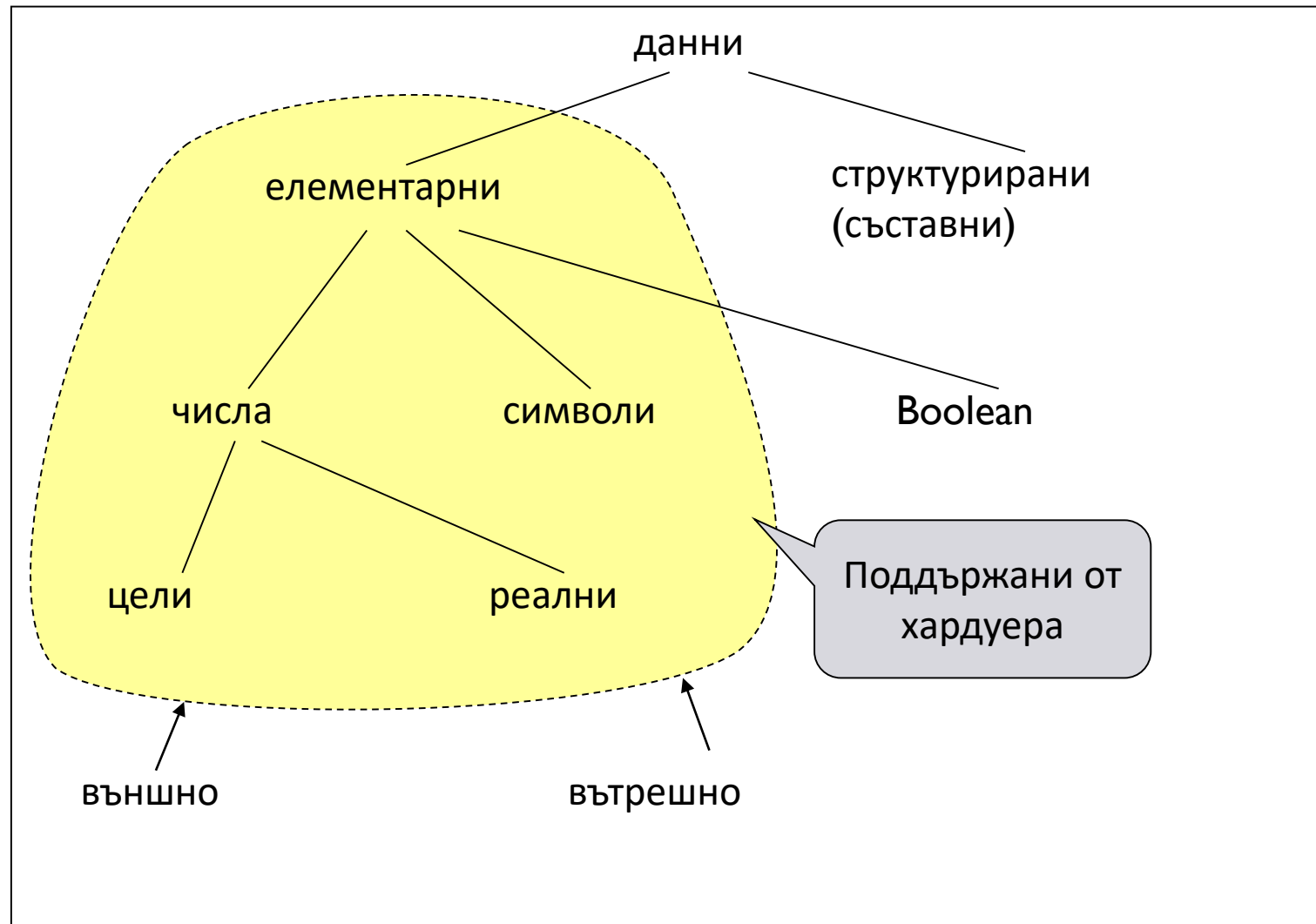
- ▶ външно - съобразено с хората, четима форма
- ▶ вътрешно - последователности от битове: 00110 ...

## ▶ **Вид структуриране:**

- ▶ елементарни данни - числа, символи
- ▶ съставни (структурирани) данни

Програма = данни + алгоритми

# Класификация на данните



адрес	съдържание
60225	...
60226	00110011 01000110 01110000 10000001
60227	10110011 11100110 01110000 00011001
60228	00110011 00000110 01110000 11000001
60229	00000011 00000110 01110000 10000001
60230	10110011 00000110 01110000 01100001
60231	01110011 00000110 01110000 00000101
60232	01100111 00000001 01101011 01000101
60233	00111011 00000110 01110000 11100001
...	...
...	...
280461	00000000 00000000 00000000 00000001
...	...
...	...
440204	00000000 00000000 00000000 00000101
...	...

Вътрешна форма:

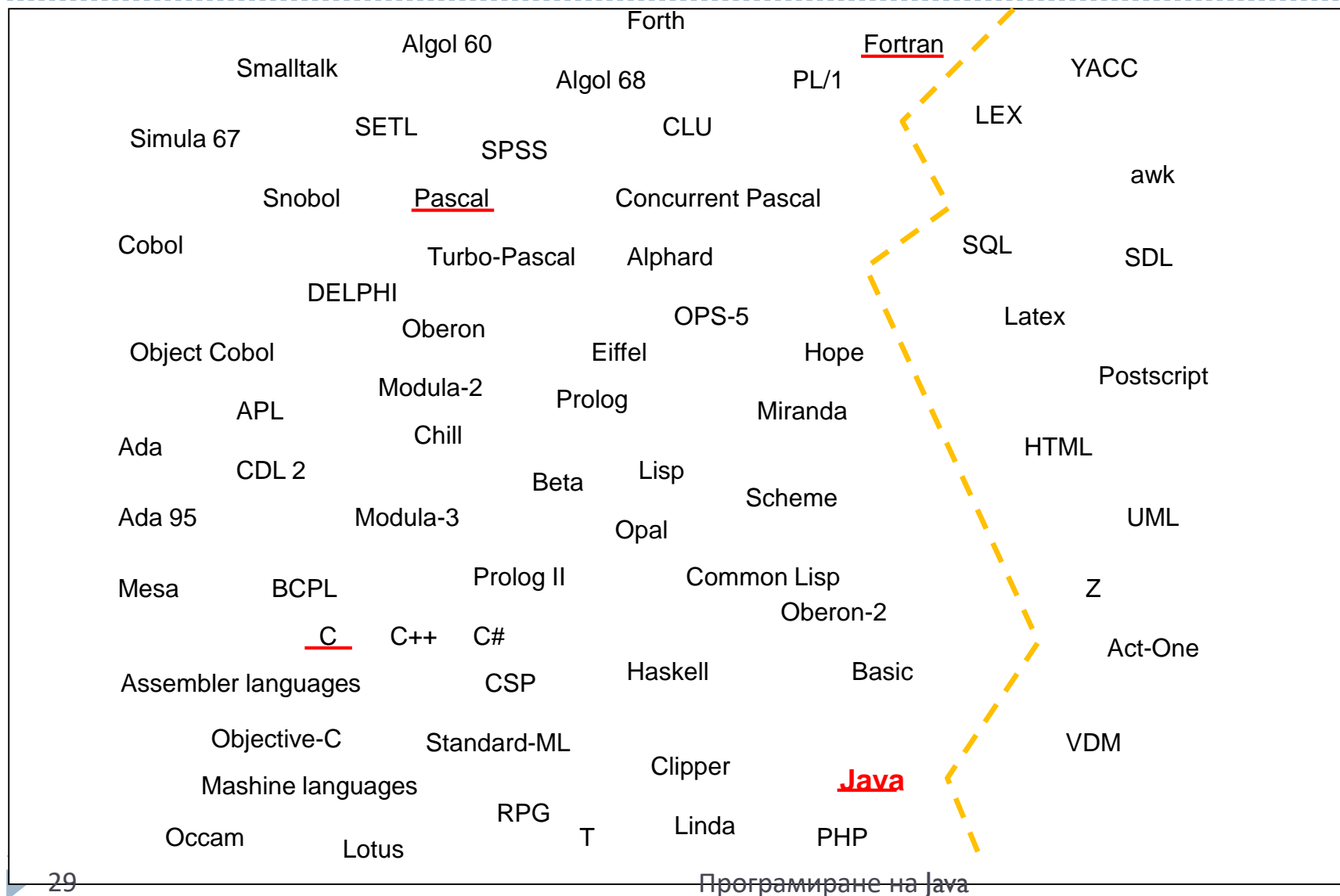
- Съдържанието на паметта
- Ограничена стойност на числата в програмите
- Проблеми на закръгляване на реалните числа

# Преглед на езиците за програмиране

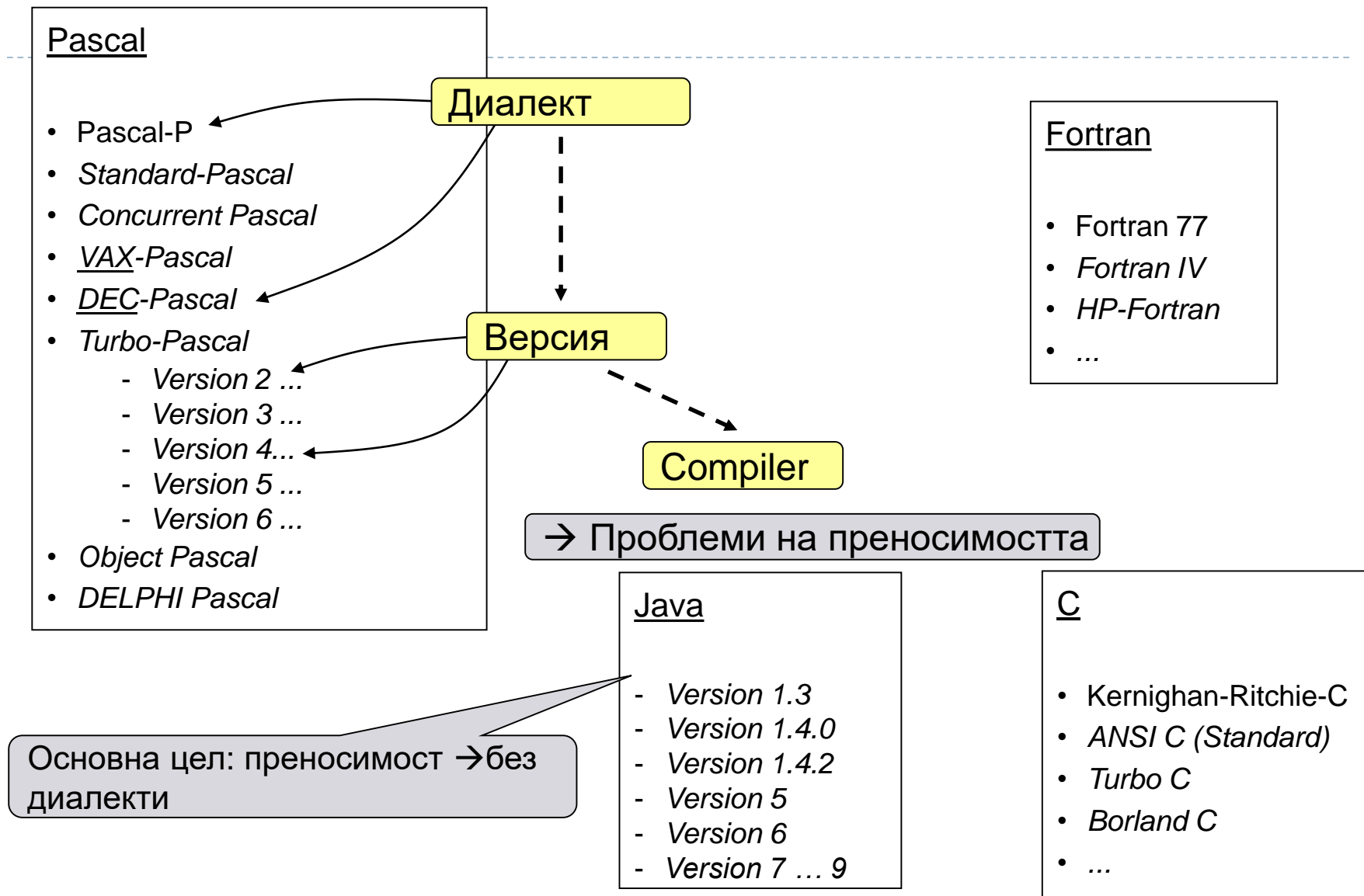
---

- ▶ Езици, диалекти, версии
  - ▶ ... и основният проблем на разнообразието: преносимост
- ▶ Класификация:
  - ▶ Как може да се предложи някаква подредба в голямото многообразие?
- ▶ Класификация според:
  - ▶ Приложната област
  - ▶ Историята
  - ▶ Генерациите на езиците
  - ▶ Парадигмите
  - ▶ Разпространението
- ▶ Основен въпрос: трябва ли да се владеят възможно много езици?

# Избрани езици за програмиране



# Езици за програмиране и варианти



# Кой език за програмиране ?

---

- ▶ Широко разпространен
- ▶ Най-добър
- ▶ Модерен
- ▶ Трябва ли да се усвояват по възможност много езици за програмиране?
- ▶ Как може да се предложи някаква подредба в голямото многообразие?

# Езици за програмиране и приложения

---

- ▶ Научно-технически области (напр. физика): Fortran, C
- ▶ Комерсиални области (банки, управление,...): Cobol
- ▶ Изкуствен интелект (напр. експертни системи, ...): Prolog, Lisp, Haskell ...
- ▶ Системен софтуер (компилатори, операционни системи, ...): C, C++, Ada, Java, CDL 2
- ▶ Реално-време: Ada95, Pearl
- ▶ Бази данни: SQL
- ▶ Телекомуникации: Chill, Ada95
- ▶ Статистически проблеми: SPSS
- ▶ Публикации / текстове: Latex, Postscript
- ▶ Генератори на компилатори: Lex, Yacc
- ▶ Формални спецификации: Z
- ▶ Софтуерни архитектури: UML

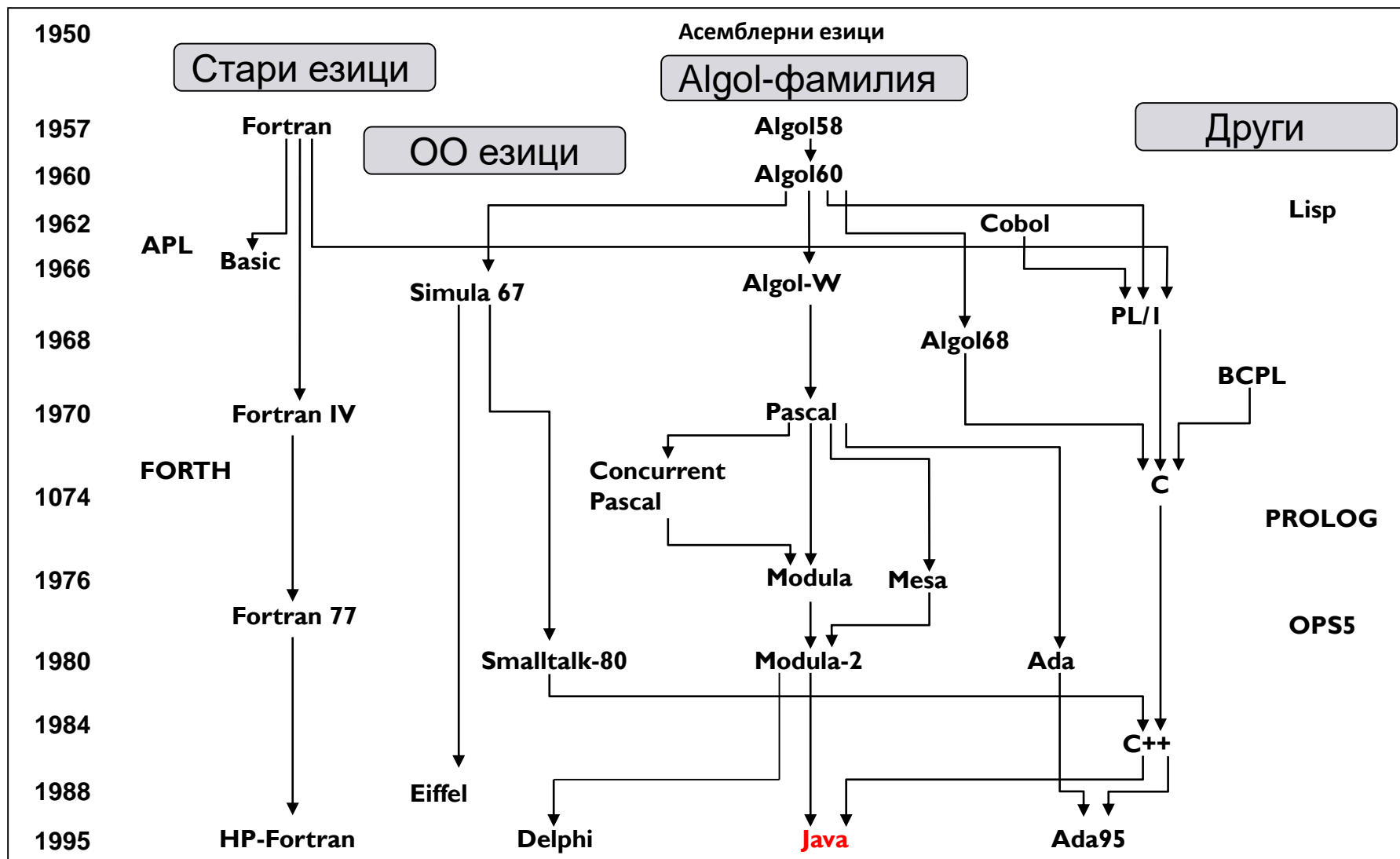
## **Внимание:**

много езици са универсално приложими

(C, C++, Ada, Java, PL/I, Pascal ...)



# Развитие на езиците за програмиране



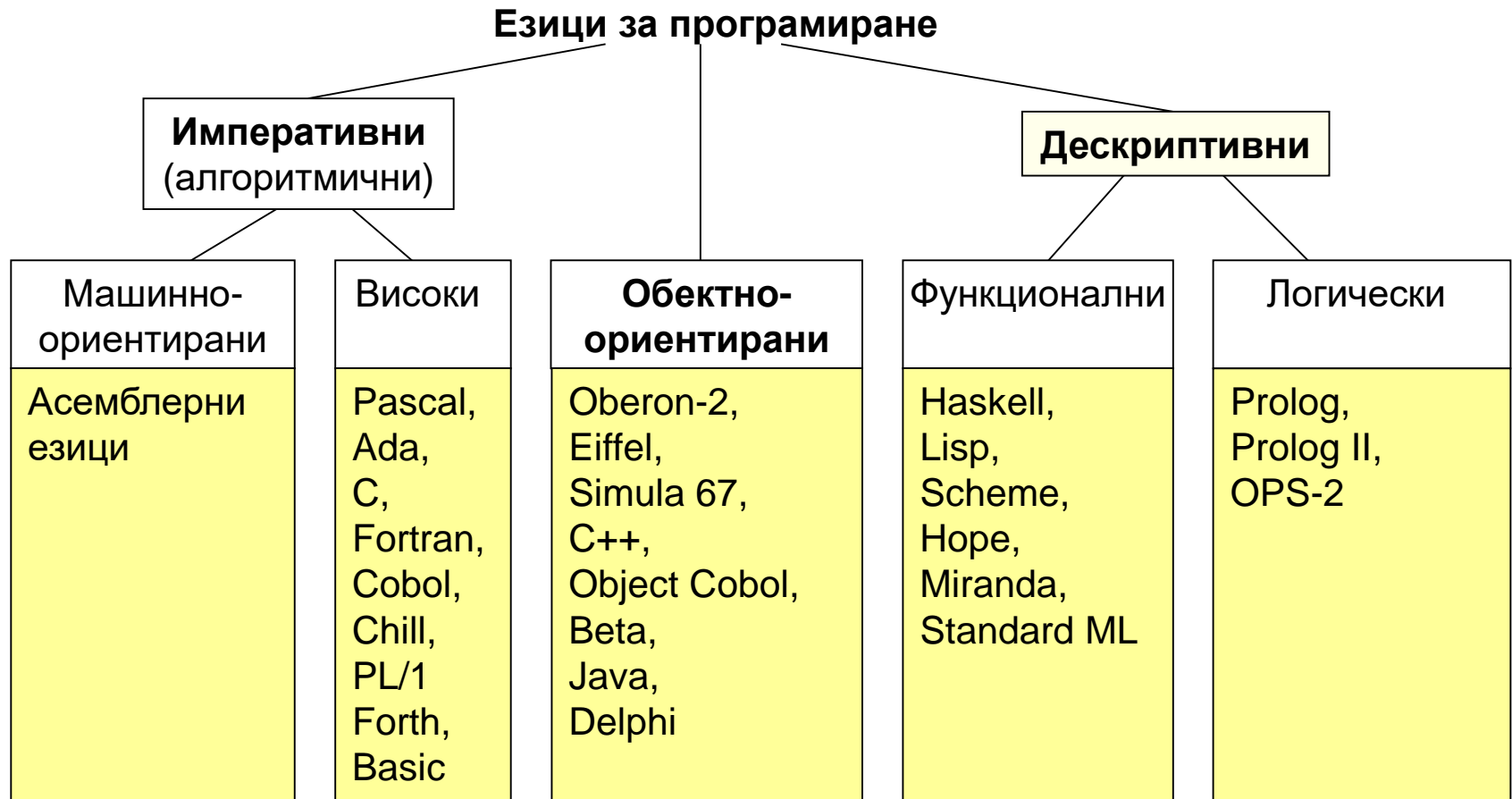
# Генерации на езиците за програмиране

---

- ▶ 1. генерация: Машинни езици
- ▶ 2. генерация: Асемблерни езици
- ▶ 3. генерация: Алгоритмични и обектно-ориентирани езици от високо ниво Pascal, Ada, C, Basic, Java ...
- ▶ 4. генерация: Електронни таблици, езици за БД SQL, Lotus, SuperCalc, dBase ...
- ▶ 5. генерация: Езици на изкуствения интелект Prolog, Lisp, OPS-5 ... Deskriptivни езици

→ ниво на абстракция

# Модели за програмиране (парадигми)



Внимание: смесени форми

- Java: OO, императивен
- Common Lisp: OO, функционален, императивен

Програмиране на Java

# Модел за програмиране: императивен

---

- ▶ Ориентиран към понятието за алгоритъм
  - ▶ Програмите се разглеждат като последователности от оператори за изпълнение
- ▶ Насочени към von-Neumann-архитектура:
  - ▶ Операторите променят данните в паметта на компютъра
- ▶ Базови концепции:
  - ▶ Променлива: за съхраняване на данни
  - ▶ Оператор: за обработка на стойности на променливи
- ▶ Метод за структуриране:
  - ▶ Подалгоритми, т. е. процедури, методи, функции

# Модел за програмиране : OO

---

Обект

Единица от:

- Данни (променливи)
- Оператори/алгоритми (методи, процедури ...) за обработка на данни

**Клас = форма за описание на подобни обекти**

# Модел за програмиране: логически

Пример:

## Програма

Логическо описание на проблеми:

- Описание на релации между 'данни' / 'обекти от реалния свят,
- Посредством предикатна логика

```
parent (maja, maria).  
parent (ivan, maria).
```

```
sister (X, Y) :-  
    parent (X, Z),  
    parent (Y, Z),  
    female (X).
```

```
female (maria).  
female (maja).
```

```
male (ivan).
```

# Трябва ли да се усвояват по възможност много езици: Резюме

---

- ▶ Овладяването на възможно повече езици не е от съществено значение, а по-скоро владееене на:
  - ▶ Концепции (оператори, типове ...)
  - ▶ Техники за програмиране (сортиране, търсене ...)
  - ▶ Изграждане на компоненти (модуларизиране: декомпозиране на големи програми)
  - ▶ Парадигми (ОО, императивна, логическа, функционална)
  - ▶ Методи за разработване на комплексен софтуер (Software Engineering)
  - ▶ Теоретични основи на програмирането
  - ▶ Работа в екип
  - ▶ Знания за съответната приложна област

# Трябва ли да се усвояват по възможност много езици: Резюме

---

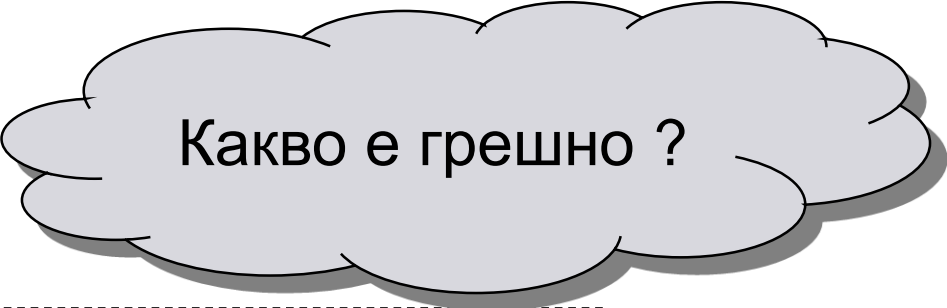
- ➔ Трябва да сме в състояние, целенасочено да вникнем в един нов език за програмиране!
- ➔ Представителите на основните езикови класове трябва да се усвояват активно:  
Java (C++), SQL, UML, HTML, XML, Prolog, Lisp ...



# Формални граматики, синтаксис, BNF

---

- ▶ Цел: създаването на коректни програми да бъде точно дефинирано



Какво е грешно ?

```
class T1 {  
    public static main (...) {  
        { x = 2 }  
    }  
}
```

# Аспекти на коректността на програмите

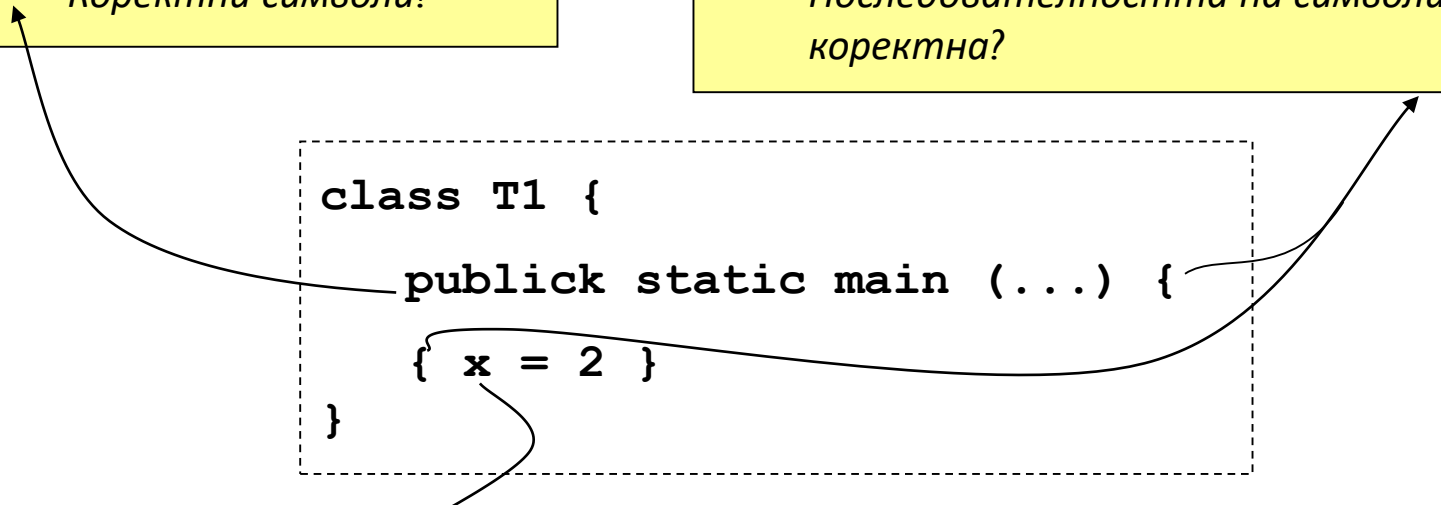
Лексика:

*Коректни символи?*

Контекстно-независим синтаксис:

*Последователността на символите  
коректна?*

```
class T1 {  
    public static main (...) {  
        { x = 2 }  
    }  
}
```



Контекстно-зависим синтаксис:

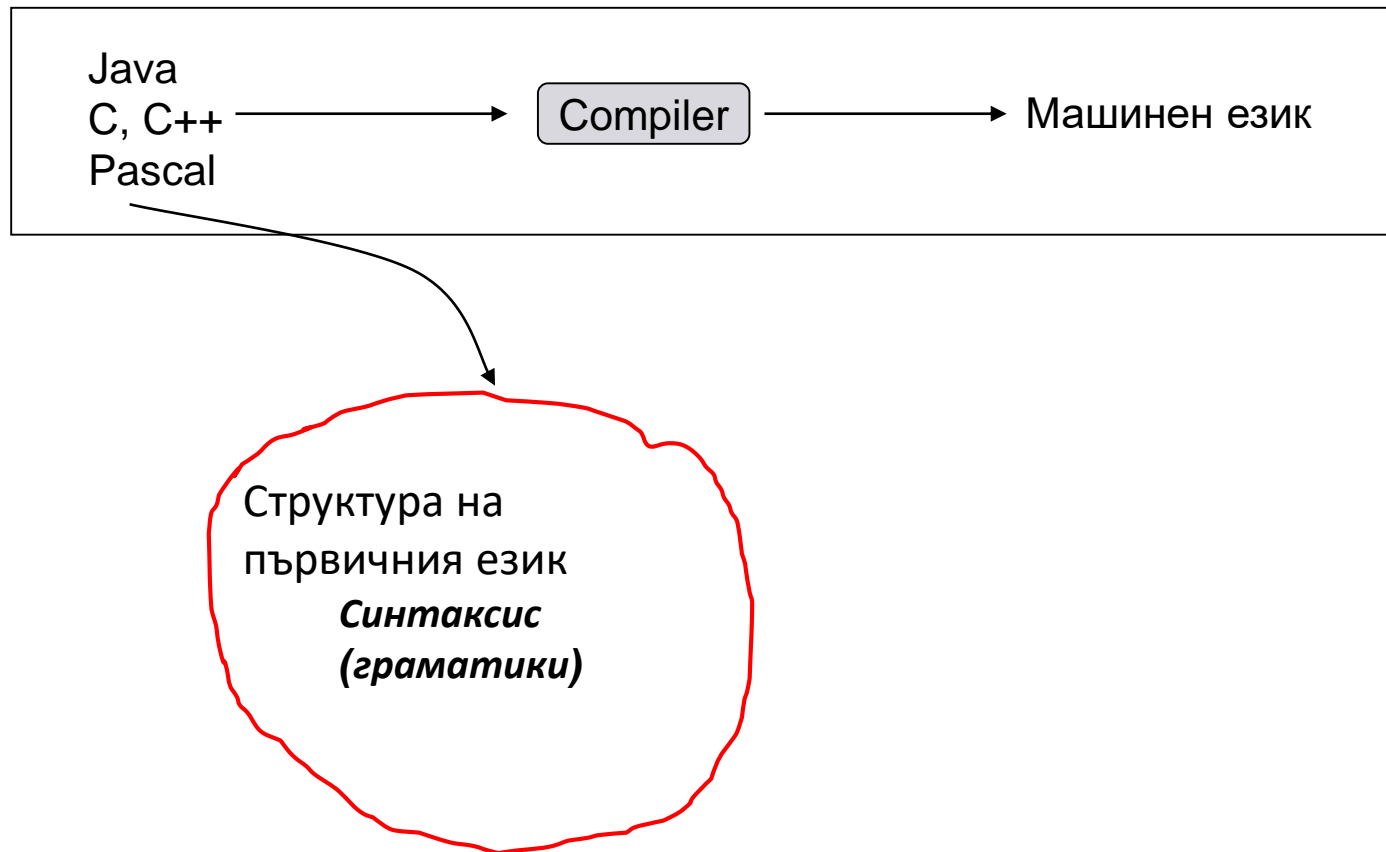
*Символите коректно свързани в  
околната среда?*

Семантика:

*Грешки при изпълнението ?  
Обработката на  
програмата коректна?*

# Анализ на грешките от компонентите на компилатора

---



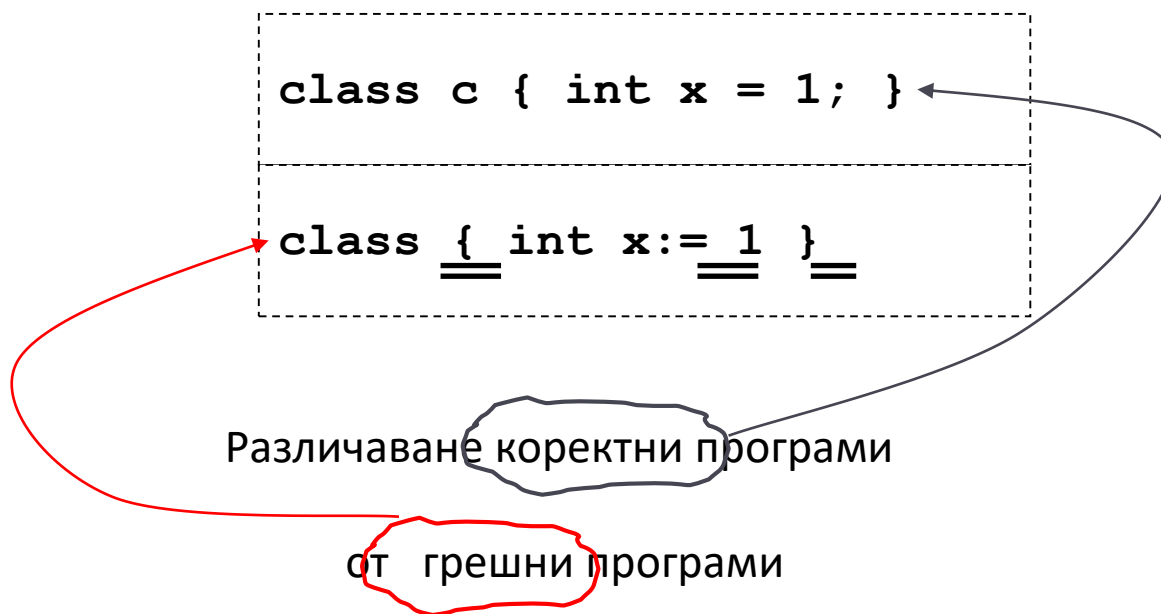
# Дефиниция на синтаксиса на езиците за програмиране

---

- ▶ **Контекстно-свободен синтаксис с контекстно-свободни граматики (EBNF)**
  - ▶ Напълно формализирани
- ▶ **Контекстно-зависим синтаксис**
  - ▶ Вербално задаван
  - ▶ Напр. , всеки идентификатор трябва да бъде деклариран преди неговото използване

# Задача на контекстно-свободните граматики

---



**Благодаря за вниманието!**

Край лекция 1. “Основни понятия”