Объектно-ориентированное программирование

Object-oriented programming

VI. Возникновение понятия “объект”

Objects

Что значит “объектно-ориентированный”?

- инкапсуляция
- полиморфизм
- наследование

"A language is considered *object-based* if it directly supports **data abstraction** and **classes**. An *object-oriented* language is one that is object-based but also provides support for **inheritance** and **polymorphism**."

***G. Booch** et al.*

"Object-Oriented Analysis, Design and Implementation"

Откуда есть пошли “объекты”

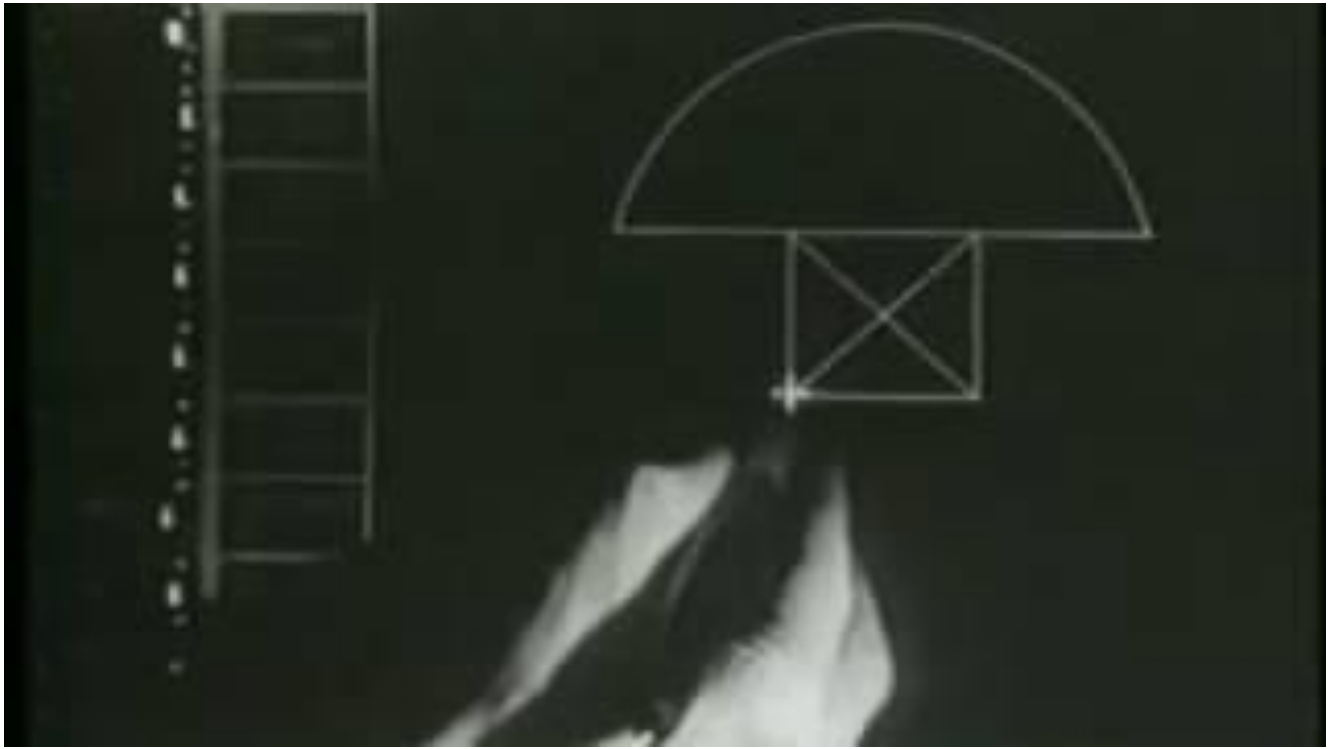
- Алан Кей в 1967 году работает над системой FLEX
  - компилируемый язык программирования
  - под влиянием **Algol 60**, **Simula 67**, **Sketchpad**
- “процессо-ориентированный” подход
  - базовая исполняющая сущность – “описание процесса”
  - создает и поддерживает операции над сегментами данных в памяти
  - операции должны поддерживать параллельное исполнение
- необходимость в поддержке языком рекурсивных вызовов, “событий” и т.п.

Объект – это …

описание вычислительного **процесса** или **семейства процессов**, которое **скрывает** свои **данные** от внешней среды и может работать независимо от других объектов, взаимодействуя с ними с помощью **сообщений**, отправленных согласно какому-то **протоколу**

Sketchpad – a man-machine graphical communication system

https://www.youtube.com/watch?v=5RyU50qbvzQ



Основные возможности системы

Master drawing – единый чертеж фигуры, который используется для создания копий, при этом изменение “мастера” отражается на копиях.

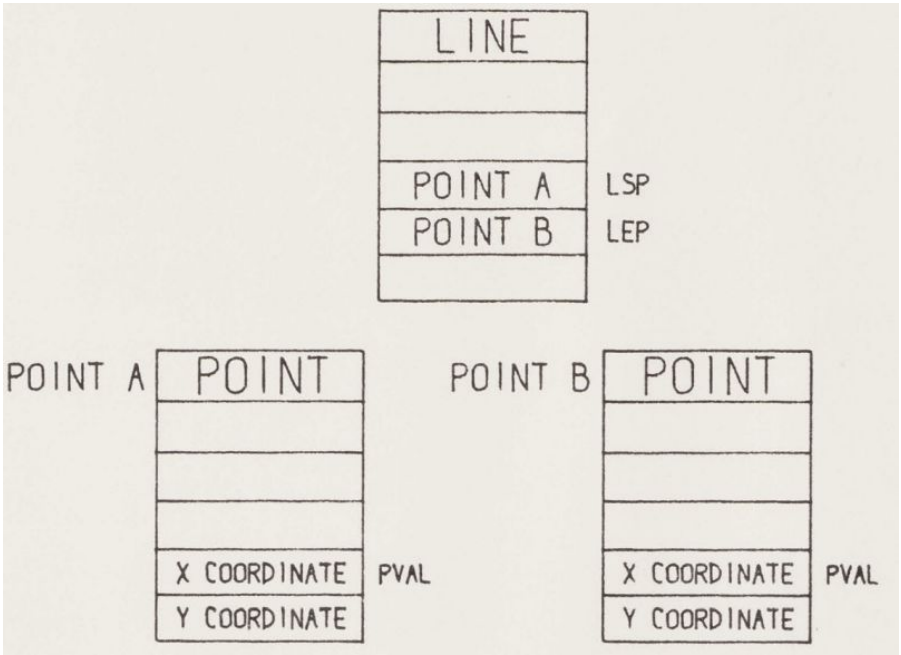
Экземпляр – рекурсивная структура данных – вектор, обладающая топологическими свойствами, которая может содержать другие экземпляры; используется для отображения фигур чертежа.

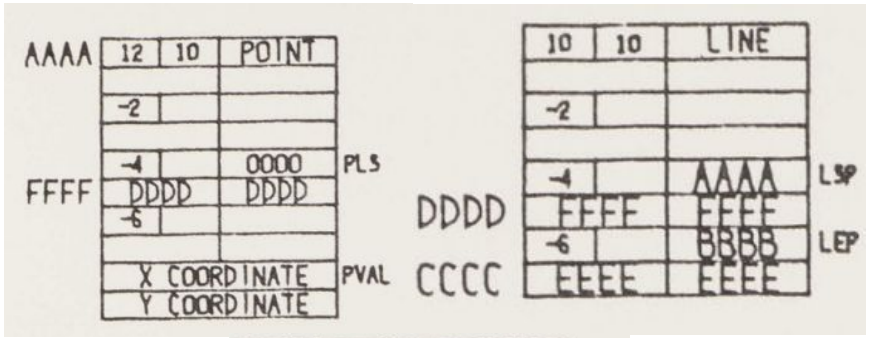
Атомарные ограничения – базовые отношения между фигурами (напр., требование чтобы шестиугольник вписывался в круг), которые насаждаются системой автоматически.

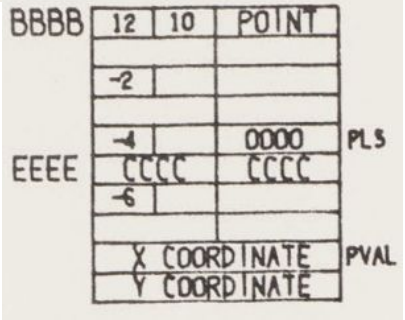
Наследование – возможность хранить информацию о фигурах в “иерархиях”, вершиной которых является “обобщенная структура данных”.

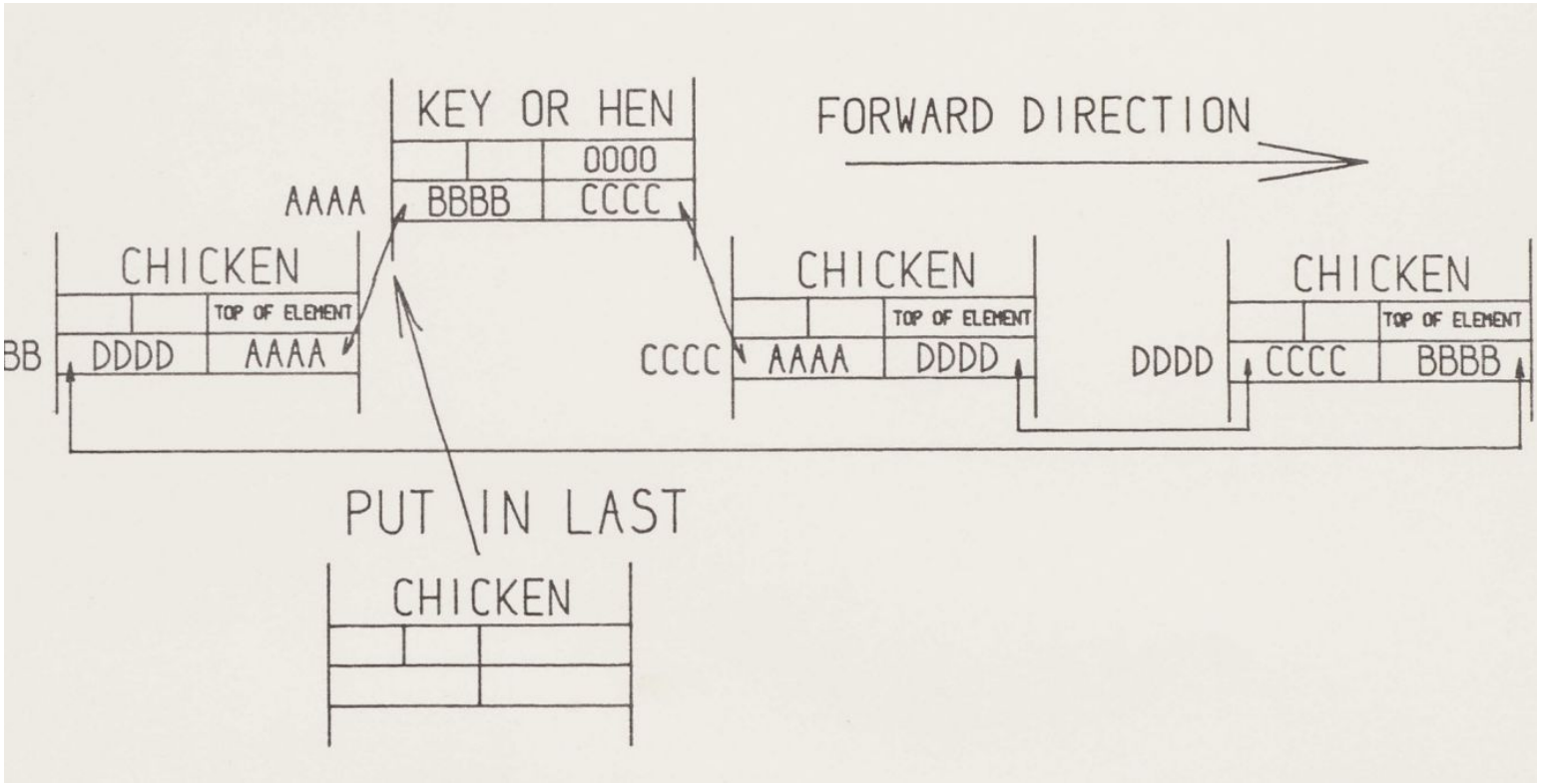
Обобщенные операции – части системы, отвечающие за поведение всех сущностей на экране, независимо от их “типа”.

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf

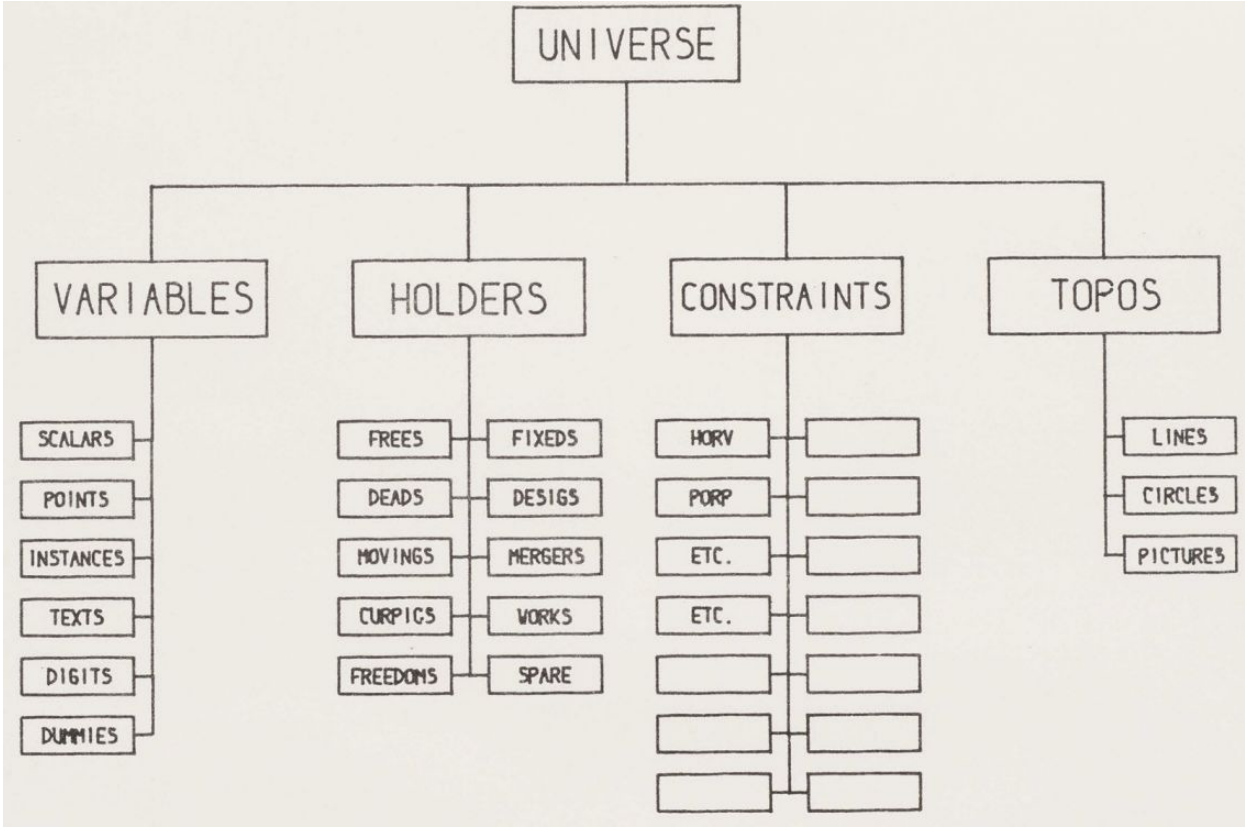

LINE
POINT A
POINT B
LSP
LEP
POINT A
POINT
X COORDINATE
Y COORDINATE
PVAL
POINT B
POINT
X COORDINATE
Y COORDINATE
PVAL


AAAA
12
10
POINT
-2
-4
0000
PLS
FFFF
DDDD
DDDD
-6
X COORDINATE
Y COORDINATE
PVAL
10
10
LINE
-2
-4
AAAA
LSP
DDDD
FFFF
FFFF
-6
BBBB
LEP
CCCC
EEEE
EEEE


BBBB
12
10
POINT
-2
-4
0000
PLS
EEEE
CCCC
CCCC
-6
X COORDINATE
Y COORDINATE
PVAL

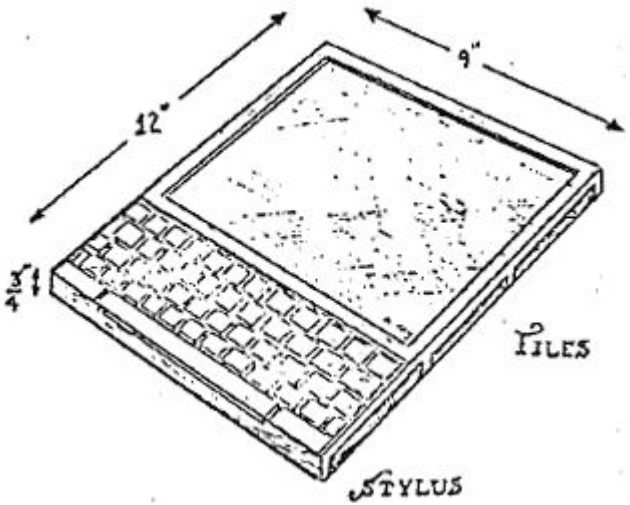

KEY OR HEN
0000
AAAA
BBBB
CCCC
FORWARD DIRECTION
CHICKEN
TOP OF ELEMENT
BB
DDDD
AAAA
CHICKEN
TOP OF ELEMENT
CCCC
AAAA
DDDD
CHICKEN
TOP OF ELEMENT
DDDD
CCCC
BBBB
PUT IN LAST
CHICKEN

24	4	VARIABLES	TYPE
-2		0000	SPECB
TYPEWRITER CODE NAME			NAME
SUBROUTINE ENTRY			DISPLAY
FIT SCOPE AROUND IT			HOWBIG
APPLY TRANSFORMATION			MOVIT
24.16..			SIZE
NORMAL PICTURE KIND			KIND
FOUR COMPONENTS			TUPLE
VALUE AT IVAL			VARLOC



Связь Sketchpad с ООП

- Понятие “класса” и “экземпляра” (основаны на базе “плексов” в Algol 68; напрямую повлиял на появление классов и объектов в языках SIMULA и Smalltalk)
- Концепция Dynabook (на рис.) была основана на Sketchpad (реализована в Smalltalk)
- Понятие “абстрактного типа данных” (проникло в систему через работы Ч. Хоара, почти одновременно с SIMULA, а оттуда – в C++)
- Ранняя реализация связного списка для автоматической очистки памяти



Simula

- симуляции, основанные на **событиях** (event-based)
- события обрабатываются в “**очередях**”
  - новые события попадают в очередь
  - первое событие обрабатывается (симулируется)
  - все события, сгенерированные этой симуляцией, помещаются в конец очереди
  - цикл продолжается до тех пор, пока очередь не опустеет
- очереди должны быть **обобщенными** структурами данных
- идея “процесса”: **классы**, ссылки и сопрограммы (coroutines)

https://www.cs.tufts.edu/comp/150FP/archive/kristen-nygaard/hopl-simula.pdf

Simula

```
Q := make_queue(first_event);
repeat
    remove next event e from Q
    simulate event e
    place all events generated by e on Q
until Q is empty
```

Simula

Отличия от ALGOL 60:

+ классы и ссылки, наследование
+ передача аргументов по ссылке
+ ввод/вывод данных (**char**, **text**)
+ сопрограммы (асинхронное программирование)
- параметры по умолчанию передаются по значению
- статические переменные
- строки (в пользу типа **text**)

Simula

Класс – процедура из структуры данных, включающей в себя информацию о вызове процедуры на стеке вместе с возвращаемым значением, с возможностью хранения указателя на конкретную структуру.

Объект – сама структура данных – **экземпляр** класса, хранящая указатели на код процедур и все необходимые для этого локальные переменные, таким образом определяя **замыкание***.

Simula

"We needed **subclasses** of processes **with actions**, not only of pure data records."

K. Nygaard

```
class Point; real x, y;
begin
    boolean procedure equals(p); ref(Point) p;
        if p =/= none then
        equals := abs(x - p.x) + abs(y - p.y) < 0.00001;
    real procedure distance(p); ref(Point) p;
        if p == none then error
        else distance := sqrt((x - p.x)**2 + (y - p.y)**2);
end
```

Как это может выглядеть в C

```
struct Point {
    double x, y;
    bool (*equals)(struct Point *this, struct Point *p);
    double (*distance)(struct Point *this, struct Point *p);
};

bool procedure_equals(struct Point *this, struct Point *p) {
    if (p != NULL) {
        return (fabs(this->x - p->x) + fabs(this->y - p->y)) < 0.00001;
    }
    return false;
}

double procedure_distance(struct Point *this, struct Point *p) {
    assert(p != NULL);
    return sqrt(pow(this->x - p->x, 2) + pow(this->y - p->y, 2));
}
```

Smalltalk

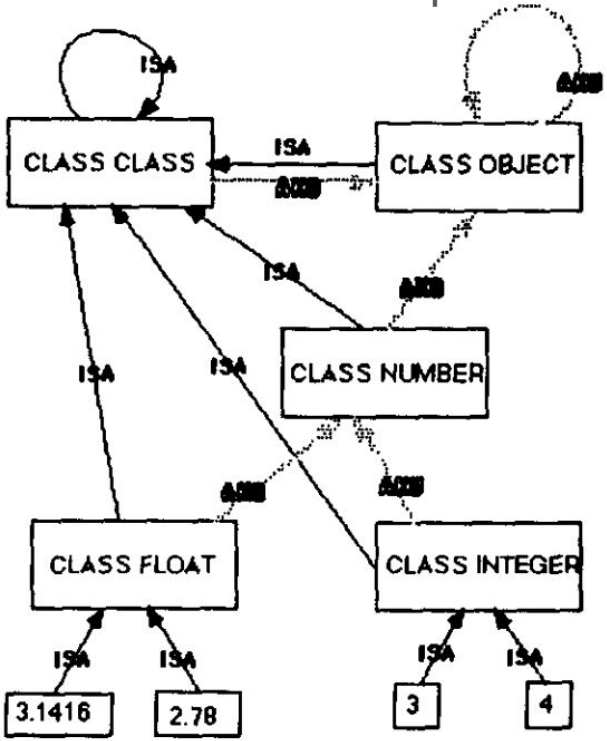
"What I got from Simula was that you could now *replace bindings and assignment with **goals***. The last thing you wanted any programmer to do is mess with **internal state** even if presented figuratively. Instead, the **objects** should be presented as sites of *higher level behaviors* more appropriate for use as dynamic components."

A. Kay

https://dl.acm.org/doi/pdf/10.1145/234286.1057828

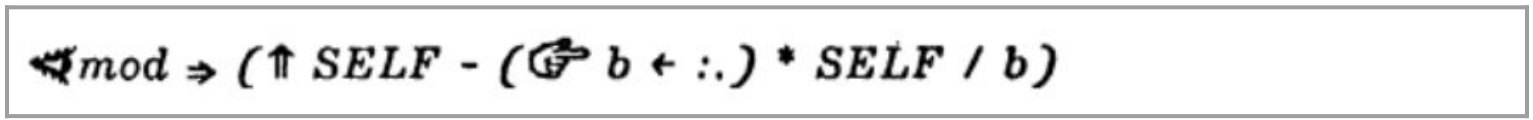
Smalltalk

Все является объектом, даже классы и “литералы”.



Smalltalk

```
5 mod 3
```



Smalltalk

```
5 (mod 3)
```

объект

сообщение для объекта

```
mod 3
```

объект

сообщение для объекта

```
3
```

объект

```
mod ⇒ ((b ← :.) is number ⇒ (⇑ SELF - b * SELF / b)
                error ('non-numeric operand'))
```

Smalltalk

```
to if exp
        ((☞exp ← :) ⇒ (◁ then ⇒ (☞exp ← :.  ◁else ⇒ (☝. exp) exp)
                                error ☞(no then ))
          ◁ then ⇒ (☝. ◁else ⇒ (☞exp ← :) false)
          error ☞(no then)) !

☞val ← if a > 10 then 4 else (if a < 10 then (-4) else 0)!
```

```
to while Cond Exp
        (☞Cond ← ☝.
          ◁ do.
          ☞Exp ← ☝.
           repeat (apply Boolean to Cond ⇒ (Exp eval) done))!

☞str ← stream!
while (kbck and  ((☞t ← kbd) ≠ 13))
    do (str ← t)!
```

Smalltalk

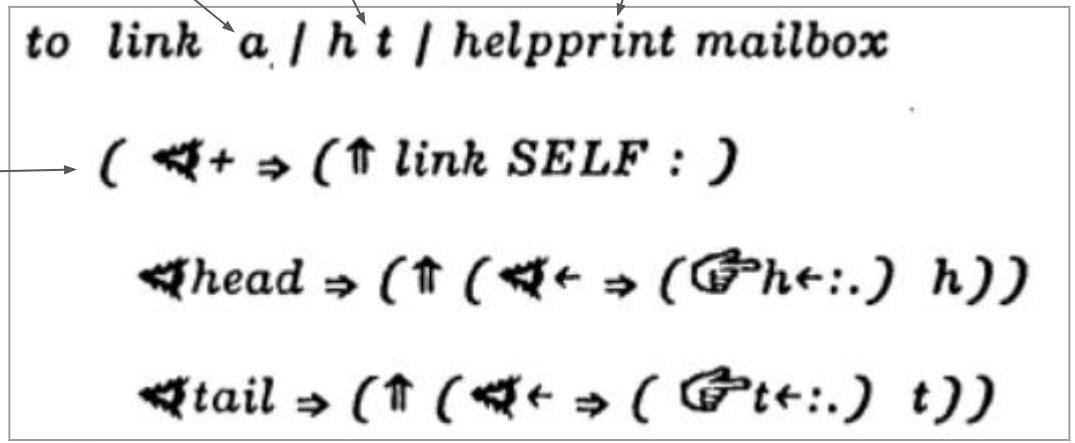
Структура класса:

временные переменные

переменные экземпляра

переменные класса

таблица методов и ответов на сообщения



Smalltalk

Структура класса **True**:

```
to True | truthValue |
(
    @ifTrue => (trueBlock (ifFalse (falseBlock (^trueBlock value))))!
    @ifFalse => (falseBlock (ifTrue (trueBlock (^trueBlock value))))!
    @ifTrue => (trueBlock (^trueBlock value))!
    @ifFalse => (falseBlock (^nil))!
    @not => (^false)!
    @& :aBoolean => (^aBoolean)!
    @| :aBoolean => (^true)!
    @eqv :aBoolean => (^aBoolean)!
    @xor :aBoolean => (^aBoolean not)!
    @and :aBlock => (^aBlock value)!
    @or :aBlock => (^true)!
    @printOn :aStream => (aStream nextPutAll :'true')!
)!
```

Что значит “объектно-ориентированный”?

- ~~инкапсуляция~~ (абстрактные типы данных)
- ~~полиморфизм~~ (алгебраическое свойство системы типов)
- ~~наследование~~ (не все “OO”-языки реализуют наследование)
- делегация? (forwarding, делегирование сообщений другим объектам)
- виртуальные функции (dynamic look-up)
- обобщение (абстракция) (скрытая реализация, открытый интерфейс)
- подмена типов (subtyping)

Что значит “объектно-ориентированный”?

“Can you define a new kind of integer, put your new integers into rectangles, ask the system to blacken a rectangle, and have everything work?”

D. Ingalls

Что значит “объектно-ориентированный”?

“Can you define a new kind of integer, put your new integers into rectangles, ask the system to blacken a rectangle, and have everything work?”

D. Ingalls

“A **Lisp** programmer knows the value of everything, but the cost of nothing.”

A. Perlis

Что значит “объектно-ориентированный”?

“OOP to me means only **messaging**, **local retention** and **protection** and **hiding of state-process**, and extreme **late-binding** of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I'm not aware of them.”

A. Kay

http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en

https://www.youtube.com/watch?v=0WYgKc00J8s