ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция № 1 / **01**
11.02.2019 г.

РЕАЛИИ НАШЕГО ДНЯ

- ООП везде. Из **10** самых популярных языков программирования **7** поддерживают ООП на уровне синтаксиса.
  http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

- Программы на C++ работают быстро.

О КУРСЕ

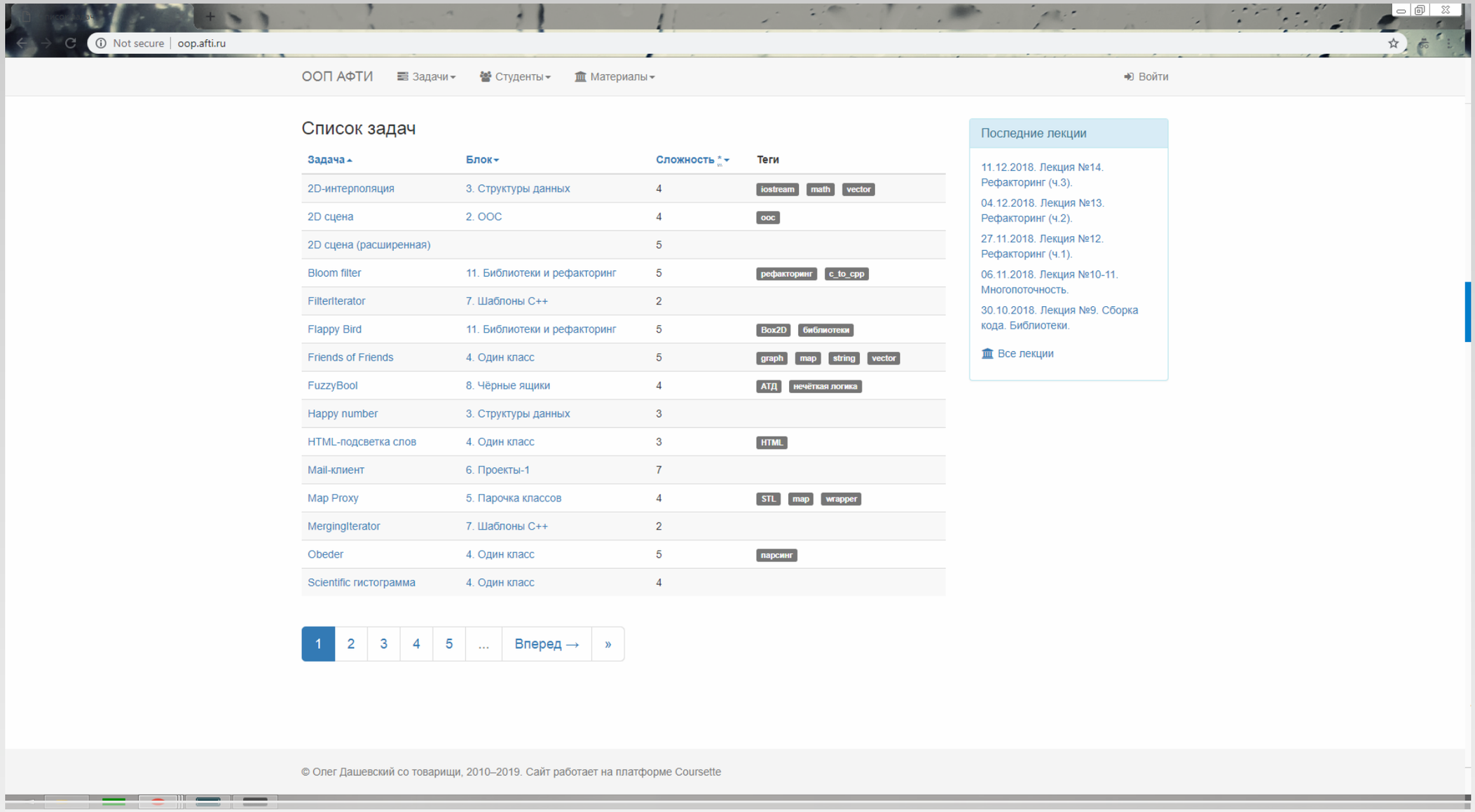
- Две основных темы:
  - C++.
  - Принципы ООП.
- Орг. часть:
  - «Как на ОПК».
  - *I часть:* проект, теория, семинары => диф. зачет + экзамен.
  - *II часть:* проект, теория, семинары, I часть => экзамен.

О КУРСЕ

Семинаристы:

Барамия Денис Александрович
baramiyadenis@gmail.com

Шишкин Дмитрий Юрьевич
shish@sl.iae.nsk.su



http://oop.afti.ru/

ЛАДНО, КРУТО!

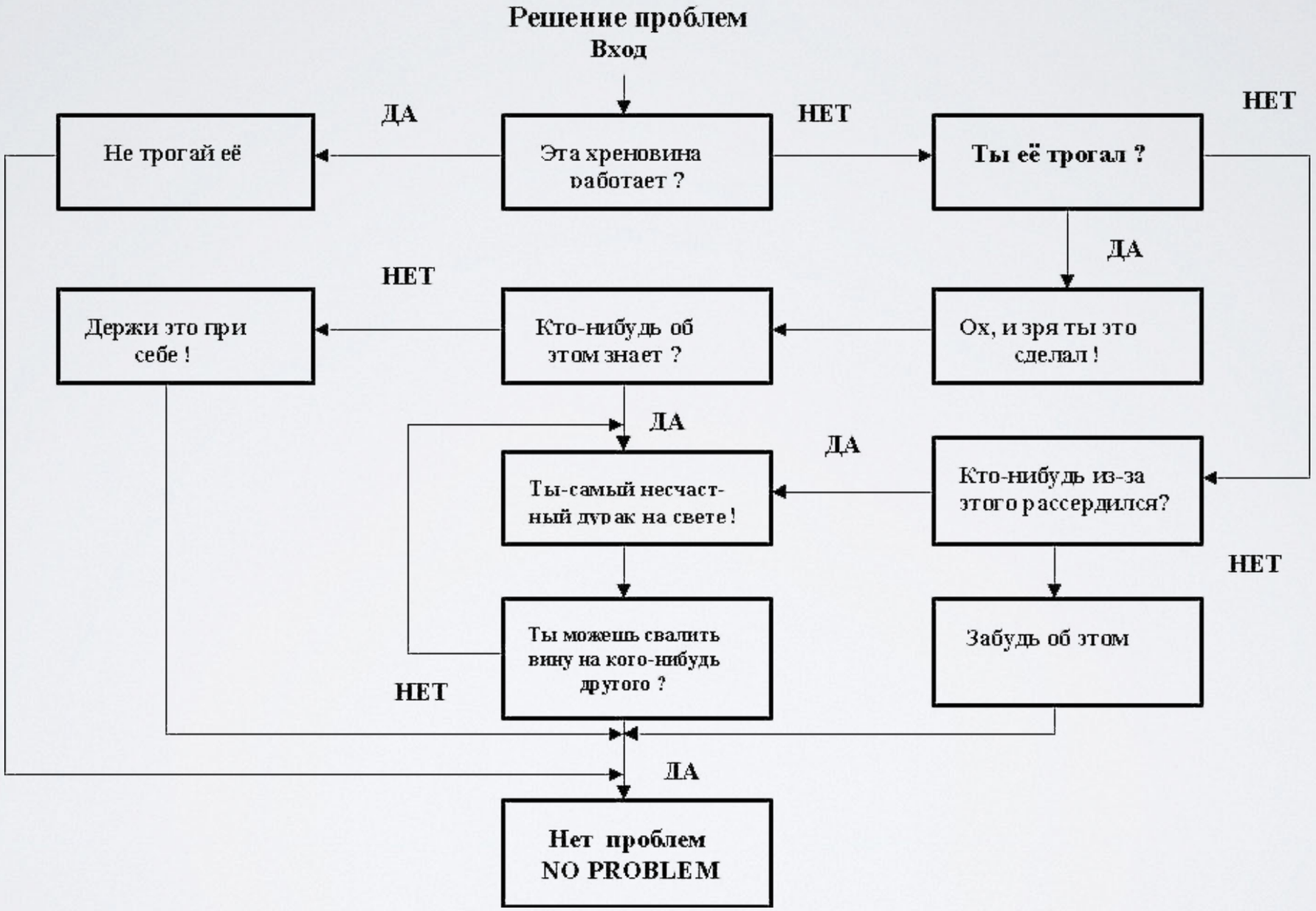
А ЧТО ТАКОЕ ООП?


"Object-oriented programming
is an exceptionally bad idea
which could only have
originated in California."
-- Edsger Dijkstra

ПАРАДИГМЫ

- Известные нам парадигмы:
  - императивная (алгоритмическая);
  - функциональная;
  - логическая.

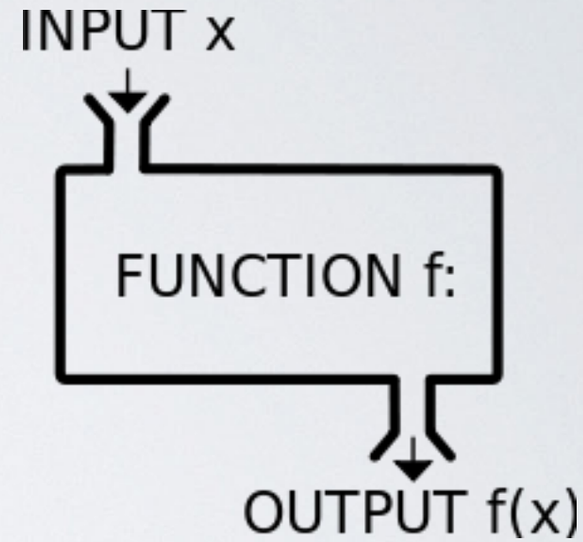
ИМПЕРАТИВНАЯ ПАРАДИГМА



последовательность действий

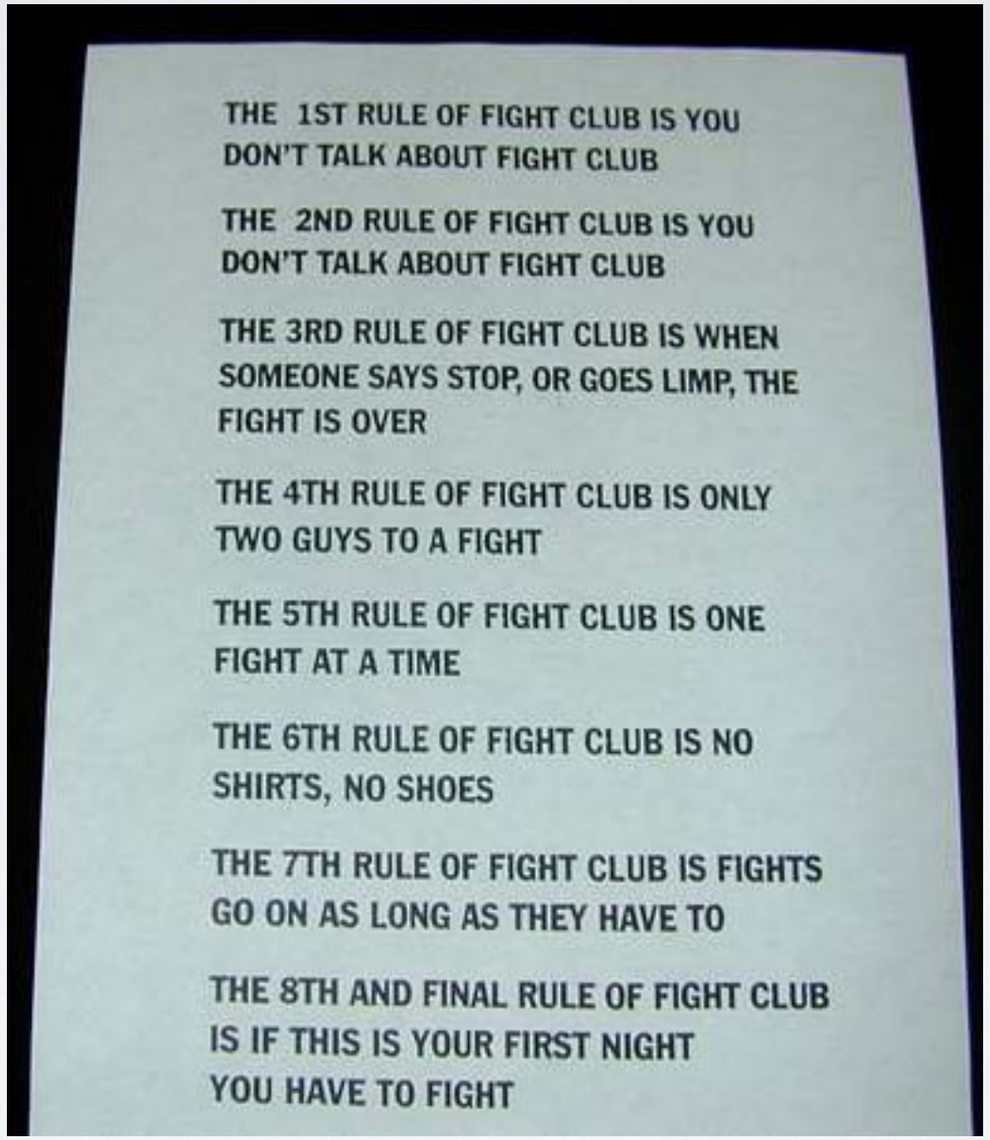
ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА

- result = f1(a, b) + f2(b, c)
- Данные являются неизменяемыми.
- Последовательность действий не важна.
- Текущее состояние (≈ слепок памяти) не важно.



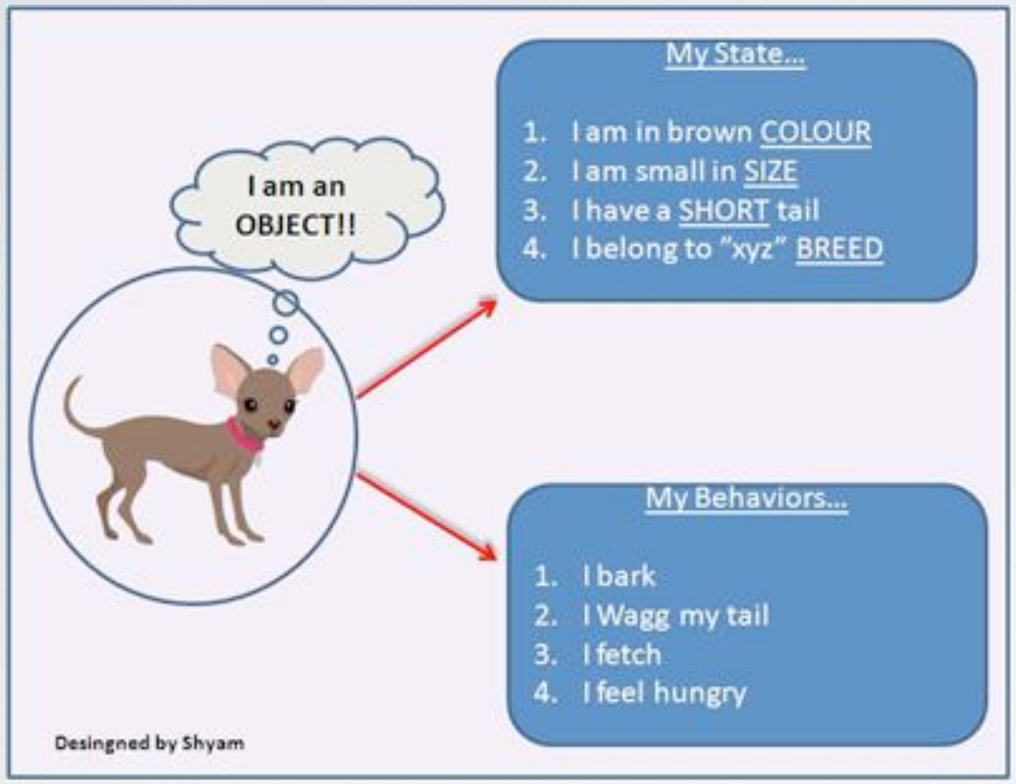
данные и преобразования над ними

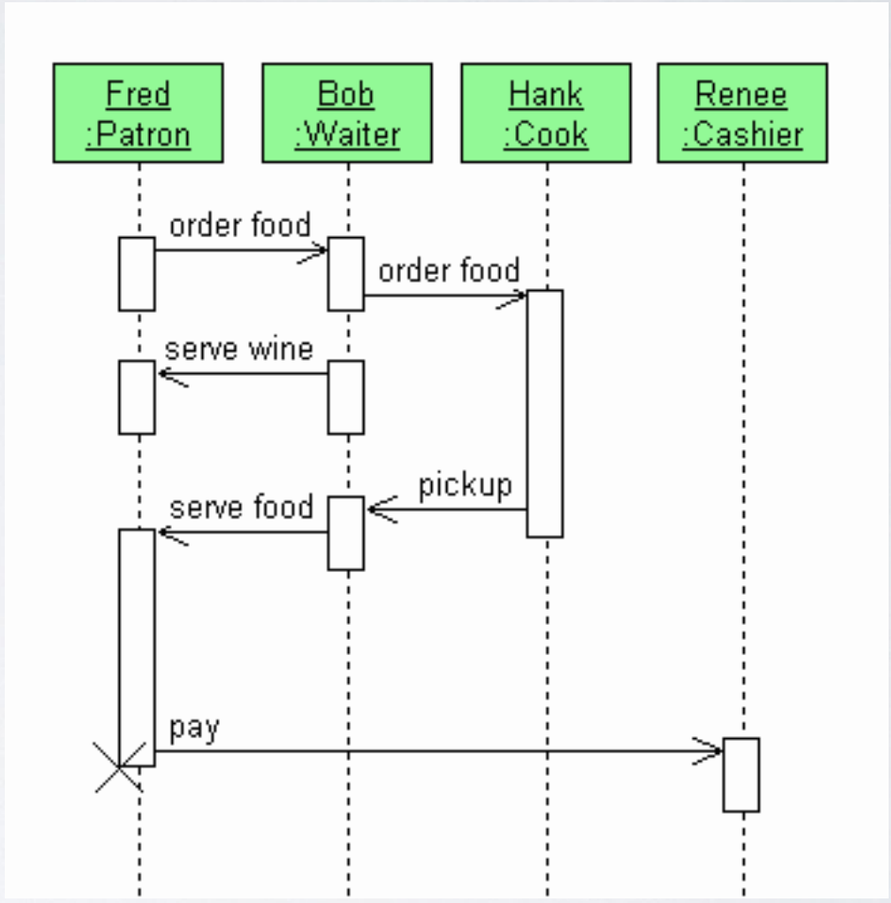
ЛОГИЧЕСКАЯ ПАРАДИГМА



предикаты, правила

О. О. ПАРАДИГМА





объекты, их «поведение» и взаимодействие

ОБЪЕКТЫ

- Программы так или иначе моделируют мир.
- В мире есть «объекты», т. е. некие отдельные сущности.
- Пусть в программах тоже будут объекты!
- Для нас важны **свойства** и **поведение** объекта исходя из решаемой задачи !!

ВЗАИМОДЕЙСТВИЕ ОБЪЕКТОВ

- Важен момент **рождения** объекта и время его **жизни**.

- Принцип **иерархии**: есть более глобальные объекты, которые живут дольше и порождают менее глобальные объекты.

- Принцип **черного** ящика: взаимодействие с любым объектом через интерфейс.

КАКОЙ ЯЗЫК САМЫЙ ООП-ИСТЫЙ?


КАКОЙ ЯЗЫК ЛУЧШЕ ВСЕГО
ПОДХОДИТ ДЛЯ ООП?
ООП - ЭТО КОНЦЕПЦИЯ


ЧЕТЫРЕ КИТА ООП

Абстракция

Полиморфизм

Наследование

Инкапсуляция

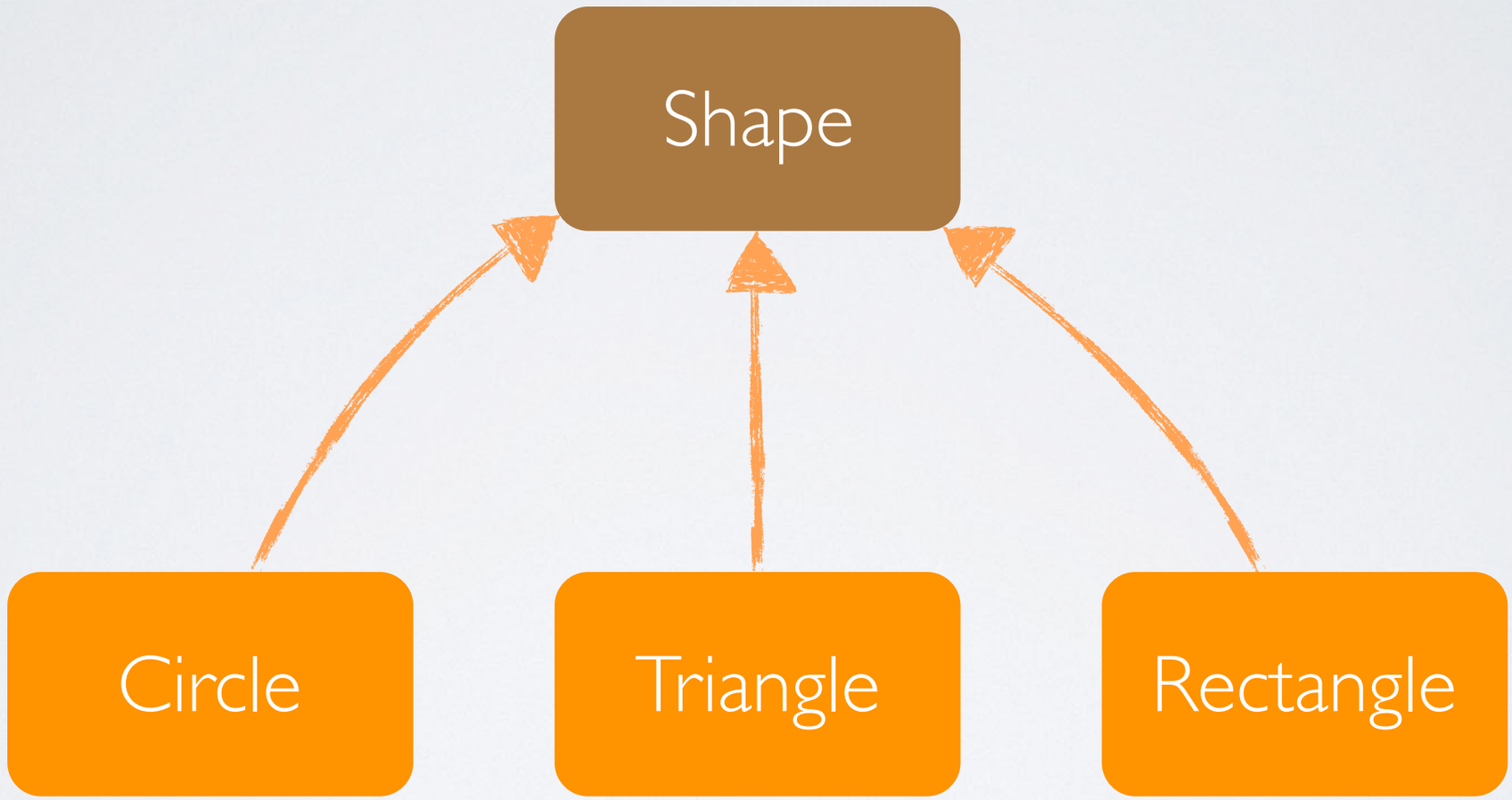
ИНКАПСУЛЯЦИЯ

- Данные сокрыты.
- Детали реализации сокрыты.
- Наружу торчит только интерфейс.

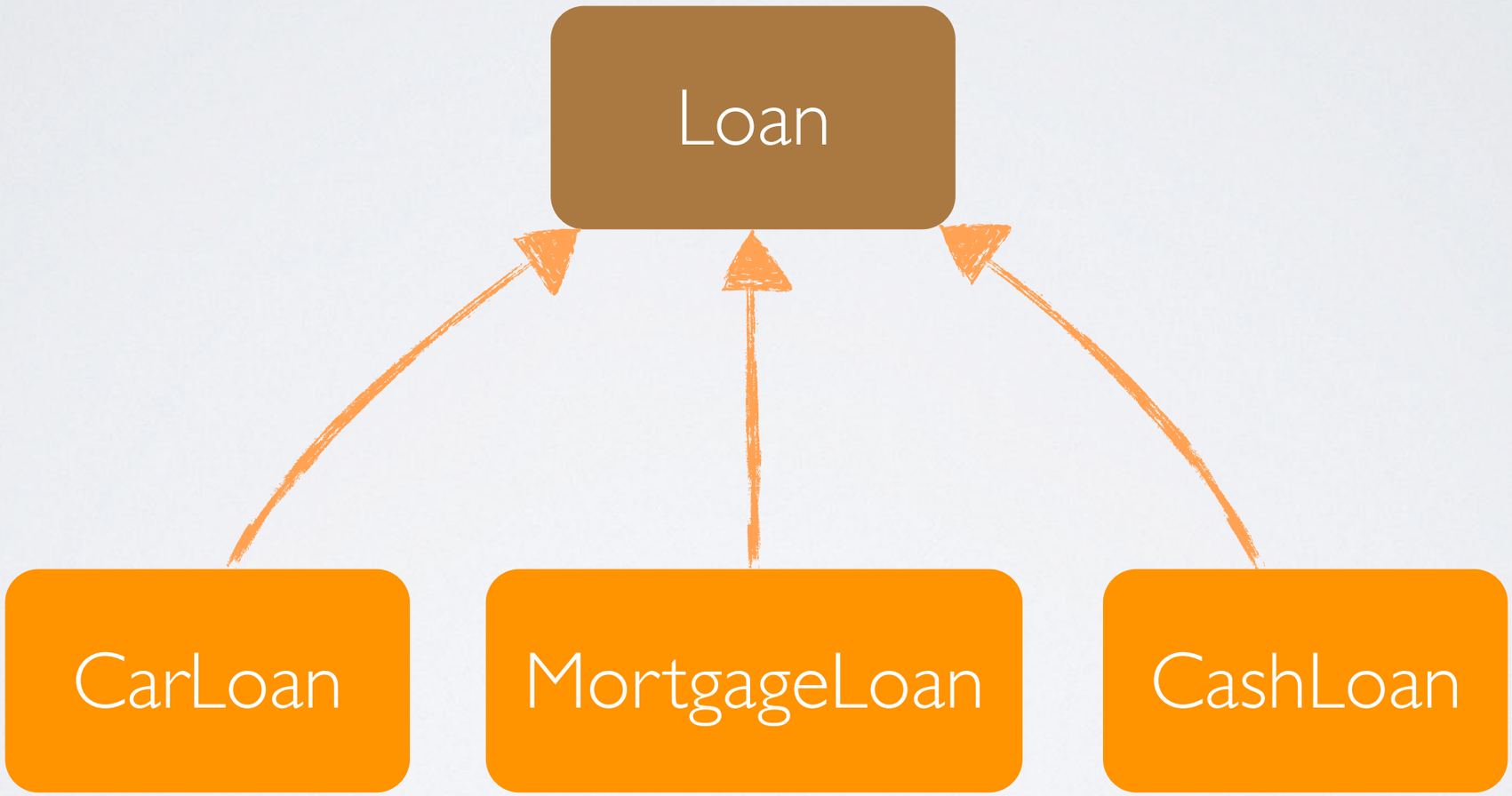
НАСЛЕДОВАНИЕ

- «**Этот** объект ведет себя так же, как **тот**, только немного по-другому»
- Поведение класса **A** наследуется от класса **B** (с изменениями).
- **B** — базовый класс (суперкласс).
- **A** — производный класс (подкласс).

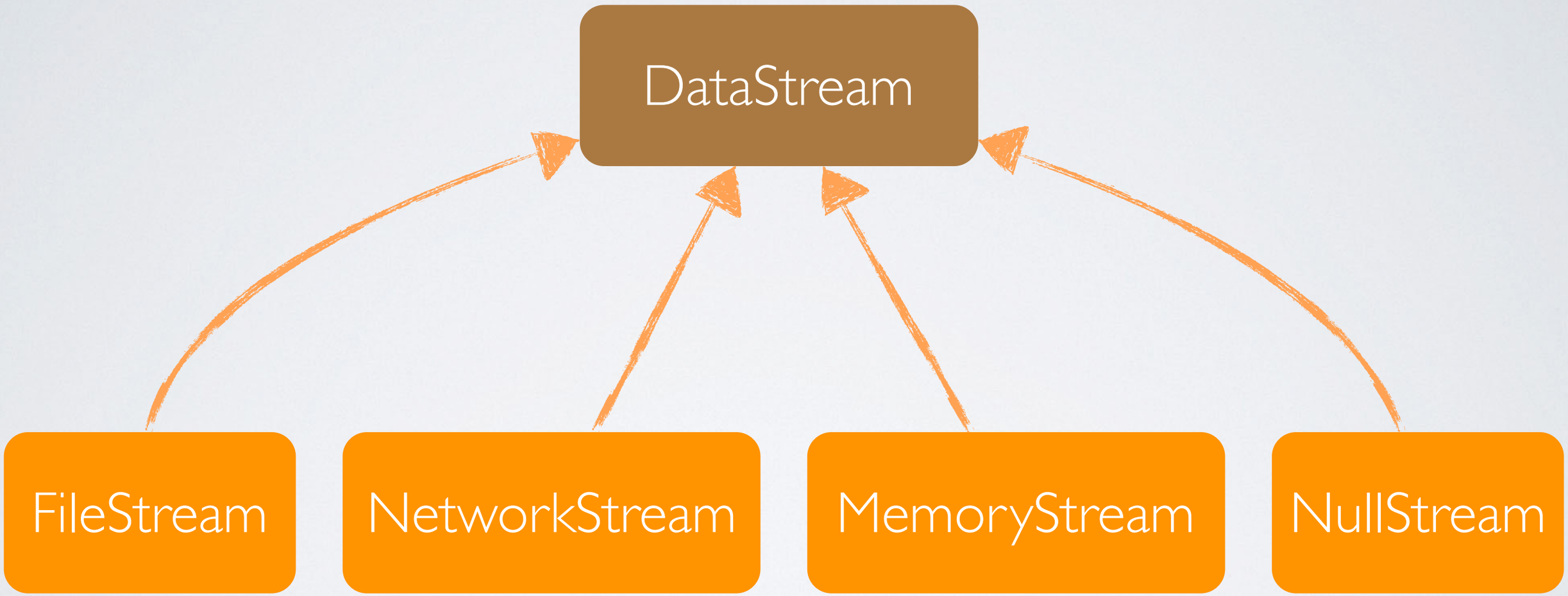
ГЕОМЕТРИЧЕСКИЕ ФИГУРЫ



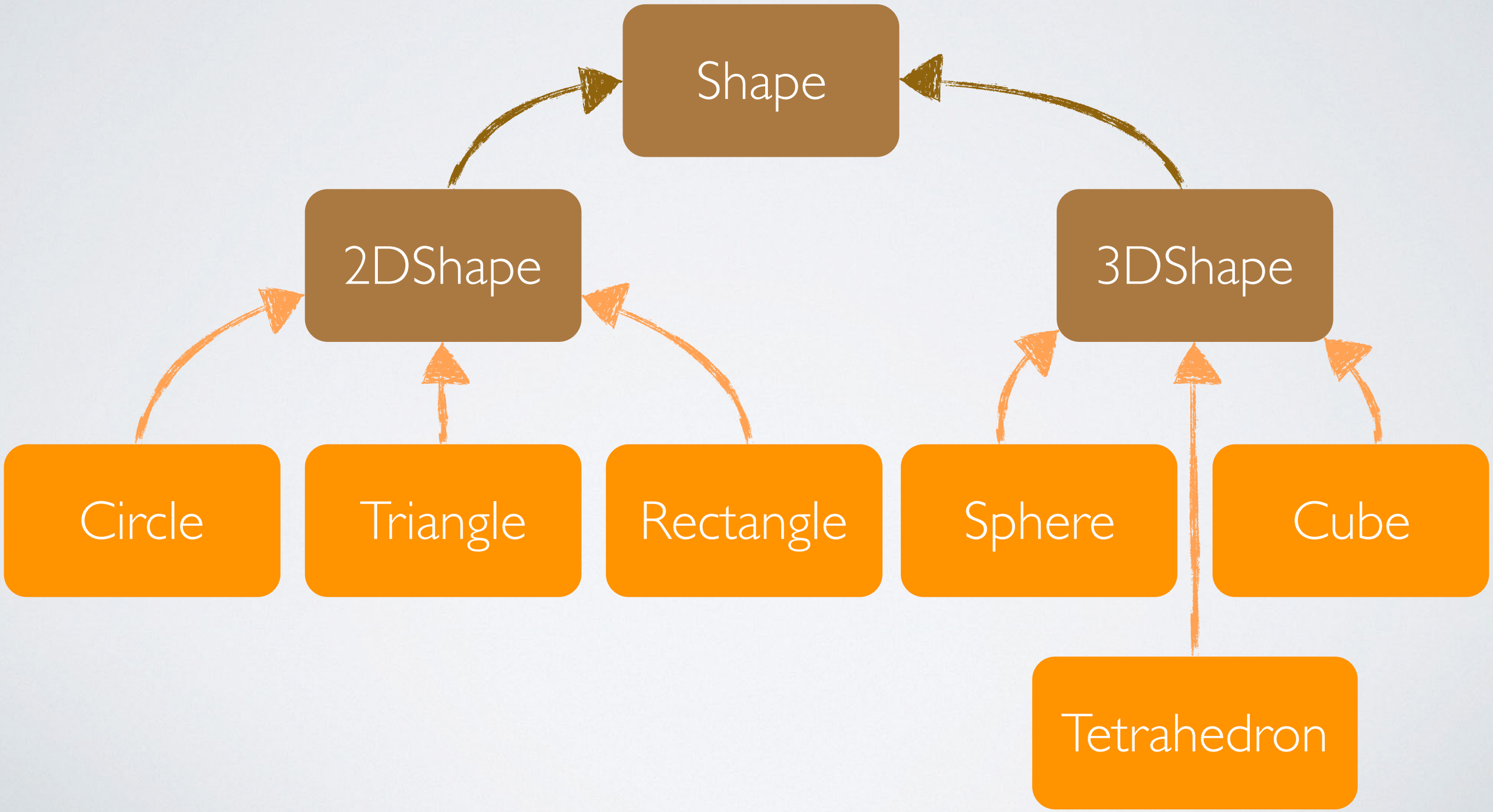
БАНКОВСКИЕ КРЕДИТЫ



ПЕРЕДАЧА ДАННЫХ



ИЕРАРХИЯ НАСЛЕДОВАНИЯ



ПРИНЦИПЫ НАСЛЕДОВАНИЯ

- Принцип «**A** является **B**» (*is-a*). Круг (**Circle**) является плоской фигурой (**2DShape**), которая является просто фигурой (**Shape**). Обратное неверно!

- Нельзя путать с «**A** содержит **B**» (*has-a*). Это не наследование, а композиция!

НЕКОТОРЫЕ ПРОБЛЕМЫ

- Square **is a** Rectangle? (Circle **is a** Ellipse?)

- Будет ли удачным ходом сделать класс **Square** наследником **Rectangle**?

- Ответ: **нет**, если объект может изменяться. Методы **stretchWidth** и **stretchHeight** не годятся для **Square**.

- Ответ: **да**, если объект не будет изменяться после создания.

ValueObject

ПОЛИМОРФИЗМ

Один интерфейс, много реализаций.

Полиморфизм

```
int abs(int x){}

double abs(double x){}

float abs(float x){}

long abs(long x){}

long long abs(long long x){}
```

Без полиморфизма

```
int abs(int x){}

double fabs(double x){}

float fabsf(float x){}

long labs(long x){}

long long llabs(long long x){}
```

АБСТРАКЦИЯ

Абстракция фокусирует внимание на существенных характеристиках объекта с точки зрения наблюдателя

Абстракция кота

Значимое поведение

- Мяукнуть
- Поесть
- Потребовать погладить
- Погулять

Незначимое поведение

- Рвать обои
- Испортить тапки

Вспомним немного C

СТРУКТУРЫ

```c
// point.h
struct Point {
   int x, y;
};
```

не очень удобно

```c
// main.c
int main() {
   struct Point point = {1, 1};
   printf("Point(%d, %d);\n",
      point.x, point.y);
   return 0;
};
```

```c
// point.h
typedef struct {
   int x, y;
} Point;
```

получше

```c
// main.c
int main() {
   Point point = {1, 1};
   printf("Point(%d, %d);\n",
      point.x, point.y);
   return 0;
};
```

EXTERN

```c
// file.h
extern double PI; // <- declaration

// file.c
#include "file.h"
double PI = 3.14159265359; // <- definition

// main.c
#include "file.h"

int main() {
   printf("PI = %.11f;\n", PI);
   return 0;
};
```

STATIC

```c
// file.h
int counter();

// file.c
#include "file.h"

int counter() {
   static int count = 0;
   return count++;
}
```

Область видимости ограничена блоком, в котором определен static

```c
// main.c
#include "file.h"

int main() {
   printf("%d\n", counter());
   printf("%d\n", counter());
   return 0;
};
```

INCLUDE GUARD

```c
// point.h
#ifndef POINT_H
#define POINT_H

struct Point {
   int x, y;
};

#endif//POINT_H
```

```c
// circle.h
#include "point.h"
```

```c
// main.c
#include "point.h"
#include "circle.h"
```

PRAGMA ONCE

```c
// point.h
#pragma once

struct Point {
   int x, y;
};

// circle.h
#include "point.h"

// main.c
#include "point.h"
#include "circle.h"
```

УКАЗАТЕЛИ НА ФУНКЦИИ

```c
#include <stdio.h>

void main()
{
    int (*func_ptr)();                          /* Объявление указателя */
    func_ptr = printf;                          /* Присваивание */
    (*func_ptr)("Printf is here!\n");           /* Вызов функции */
}
```

```c
void qsort(void *base, size_t nitems,
           size_t item_size,
           int (*compar)(void *p1, void *p2));


void *bsearch(void *key, void *base,
           size_t nitems, size_t item_size,
           int (*compar)(void *p1, void *p2));
```

- **base** — указатель на начало массива.
- **nitems** — количество сортируемых элементов.
- **item_size** — размер одного элемента в байтах.
- **compar** — функция сравнения. Возвращает **−1**, **0** или **1**. **p1** и **p2** — указатели на сравниваемые элементы.

```c
#include <stdio.h>
#include <stdlib.h>

char *names[] = {
    "Vasya",
    "Petya",
    "Semyon Semenych"
};


int name_compare(void *p1, void *p2) {
    char *n1 = *(char **)p1;
    char *n2 = *(char **)p2;

    return strcmp(n1, n2);
}

int main()
{
    int i;
    qsort(names,                                    // void *base
          sizeof (names) / sizeof (char *),         // size_t n_items
          sizeof (char *),                          // size_t item_size
          name_compare);                            // int (*compar)(void *p1, void *p2)

    for (i = 0; i < 3; ++i)
        puts(names[i]);
}
```

ВИДЫ ПАМЯТИ В С

- глобальная (статическая); ← - - - - - - - static
- автоматическая (стековая); ← - - - - - - - stack
- динамическая (**malloc** / **free**). ← - - - - - - - heap

ПАМЯТЬ В С-ПРОГРАММЕ

Скомпилированный код

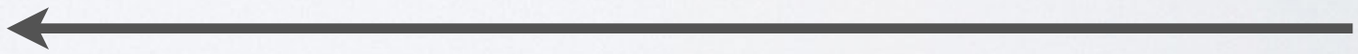
Только чтение
Размер фиксирован

Глобальная память (вкл. static)

Чтение / запись
Размер фиксирован
Время жизни переменных: максимально

Свободный стек

Занятый стек



Чтение / запись
Размер динамический, обычно ограничен сверху
(задается в свойствах линкера)
Время жизни переменных = время жизни фрейма

СТЕКОВЫЕ ФРЕЙМЫ

```
void MyFunction3(int x, int y, int z)
{
  int a, int b, int c;
  a = 10;
  ...
  return;
}
```

```
_MyFunction3:
  push ebp
  mov ebp, esp
  sub esp, 12 ; sizeof(a) + sizeof(b) + sizeof(c)
  ; x = [ebp + 8], y = [ebp + 12], z = [ebp + 16]
  ; a = [ebp - 4] = [esp + 8], b = [ebp - 8] = [esp + 4],
  ; c = [ebp - 12] = [esp]
  mov [ebp -  4], 10
  ...
  mov esp, ebp
  pop ebp
  ret 12 ; sizeof(x) + sizeof(y) + sizeof(z)
```

стек растет в направлении меньших адресов памяти

ebp+16	Аргумент **z**
ebp+12	Аргумент **y**
ebp+8	Аргумент **x**
ebp+4	Адрес возврата
ebp	Старый **ebp**
ebp−4	Переменная **a**
ebp−8	Переменная **b**
esp = ebp−12	Переменная **c**
esp + 4	Свободная ячейка
....	Свободная ячейка

```
MyFunction3(10, 5, 2);  →
```

```
push 2
push 5
push 10
call _MyFunction3
```

ПЕРЕМЕННОЕ ЧИСЛО АРГУМЕНТОВ

```
#include <stdarg.h>

double average(int n, ...)
{
    va_list myList;
    va_start(myList, n);

    // ...

    va_end(myList);
}

average(5, 1.0, 2.0, 3.0, 4.0, 5.0);
```

ebp+44	Аргумент 5.0
ebp+36	Аргумент 4.0
ebp+28	Аргумент 3.0
ebp+20	Аргумент 2.0
ebp+12	Аргумент 1.0
ebp+8	Аргумент n
ebp+4	Адрес возврата
ebp	Старый ebp
ebp−4	Переменная myList
....	

```c
#include <stdarg.h>

double average(int n, ...)
{
    va_list myList;
    va_start(myList, n);                          // Инициализация

    int numbersAdded = 0;
    double sum = 0;

    while (numbersAdded < n) {
        double number = va_arg(myList, double); // Очередное число
        sum += number;
        numbersAdded++;
    }

    va_end(myList);

    return sum / numbersAdded;                    // Вычисляем среднее
}
```

УЧИМСЯ ПРАВИЛЬНО ОФОРМЛЯТЬ КОД

https://tproger.ru/translations/learn-basic-c-coding-rules-from-open-source-projects/

КОНЕЦ ПЕРВОЙ ЛЕКЦИИ