Объектно-ориентированное программирование

Object-oriented programming

VIII. Полиморфизм

Polymorphism

Что такое ООР?

https://youtube.com/clip/UgkxMjTVfez-AB1gsyvoIpeprL2pc5YxvGjB?si=VIhZaa0uoTkcql6n

"I think that **object orientedness** is almost as much of a **hoax** as **artificial intelligence**."

A. Stepanov

“Полиморфизм” в Smalltalk 72

receiver	*message*
a	| + *b*
3	| + *4*

“kitty”	| + *“kat”*	=>	*“kittykat”*
$\begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$	| + 4	=>	$\begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$

“This led to a style of finding **generic behaviors for message symbols**. "**Polymorphism**" is the official term (*I believe derived from Strachey*), but it is not really apt as its original meaning applied only to functions that could take more than one type of argument. ”

A. C. Kay

"In more sophisticated programming languages, however, we use the **type** to tell us what sort of object we are dealing with (i.e., to **restrict** its **range** to one sort **of object**). We also expect the compiling system to **check** that we have not made silly **mistakes** (such as multiplying two labels) and to **interpret** correctly **ambiguous symbols** (such as +) which mean different things according to the types of their operands. We call ambiguous **operators** of this sort **polymorphic** as they have several forms depending on their arguments."

C. Strachey

"Fundamental Concepts in Programming Languages", 1967

https://reed.cs.depaul.edu/jriely/447/assets/articles/strachey-fundamental-concepts-in-programming-languages.pdf

"The evolution of languages from **untyped** universes to **monomorphic** and then **polymorphic** type systems[...], mechanisms for polymorphism such as **overloading**, **coercion**, **subtyping**, and **parameterization**[...], a unifying framework for polymorphic type systems[...] in terms of the typed λ-calculus augmented to include **binding of types** **by quantification** as well as **binding of values** **by abstraction**."

L. Cardelli

"On Understanding Types, Data Abstraction, and Polymorphism", 1985

http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf

Приведение типов и перегрузка

```
3   + 4
3.0 + 4
3   + 4.0
3.0 + 4.0
```

Приведение типов и перегрузка

```
3    +  4
3.0  +  4
3    +  4.0
3.0  +  4.0
```

- оператор “+” перегружен четыре раза
- оператор “+” перегружен два раза: для целых и дробных чисел
- оператор “+” не перегружен, а определен только для дробных чисел

Приведение типов и перегрузка

3	+	4
4	+	3
“3”	+	“4”
“4”	+	“3”

Приведение типов и перегрузка

```
3     +   4    // 7
4     +   3    // 7
“3”   +   “4”  // “34”
“4”   +   “3”  // “43”
```

Перегружать операторы так, как оператор **+** перегружен для строк в **C++**, в подавляющем большинстве случаев не следует. Такая перегрузка оператора игнорирует математические свойства фундаментальных операций и выработанную интуицию о природе математических операций.

Обобщенные арифметические операции в Scheme

```
> (+ 1 2 3)              # Exact Integer
6
> (+ 1 (/ 2 3))          # Exact Rational
5/3
> (+ 1 (sqrt 2))         # Inexact Real
2.414213562373095
> (+ 1 (sqrt -2))        # Inexact Complex
1+1.4142135623730951i
> (+ 1 (sqrt -4))        # Exact Complex
1+2i
```

Виды полиморфизма

- Универсальный
  - параметризация
  - включение (inclusion, sub-typing)
- Специальный (ad-hoc)
  - перегрузка (overloading)
  - приведение (coercion)

Приведение типов (ad-hoc)

Явное:

```
auto i = (int)abs(negative);
// или
auto i = int(abs(negative));
// или
auto i = static_cast<int>(abs(negative));
```

Неявное:

```
int *p = malloc(sizeof(int) * 10);
```

Управление неявным приведением классов в C++

- Создание подходящего **конструктора**, который принимает аргумент заданного типа
- Перегрузка **оператора присваивания**, которая принимает аргумент заданного типа
- Перегрузка **оператора приведения** типа

Управление неявным приведением классов в C++

```
class A { … };

struct B {
    // приведение из типа A (конструктор)
    B(const A &x) {}
    // приведение из типа A (присваивание)
    B& operator=(const A &x) { return *this; }
    // приведение к A
    operator A() { return A(); }
};
```

https://cplusplus.com/doc/tutorial/typecasting/

Полиморфизм под-типов (tagged union)

```
enum tag { Char, Int, Double };

union types {
    char char_;
    int int_;
    double double_;
};

struct variant {
    tag tag_;
    types type_;
};
```

```
void f(const variant &var) {
    switch(var.tag_) {
    case Char:
        // версия для var.type_.char_
        break;
    case Int:
        // версия для var.type_.int_
        break;
    case Double:
        // версия для var.type_.double_
        break;
    default:
        break;
    }
}
```

Полиморфизм под-типов (abstract base class)

```
class abstract_data_t {
public:
    virtual iterator find(int) = 0;
    …
};

class vector: public abstract_data_t {
public:
    iterator find(int) override;
    …
private:
    int *p{nullptr};
};
```

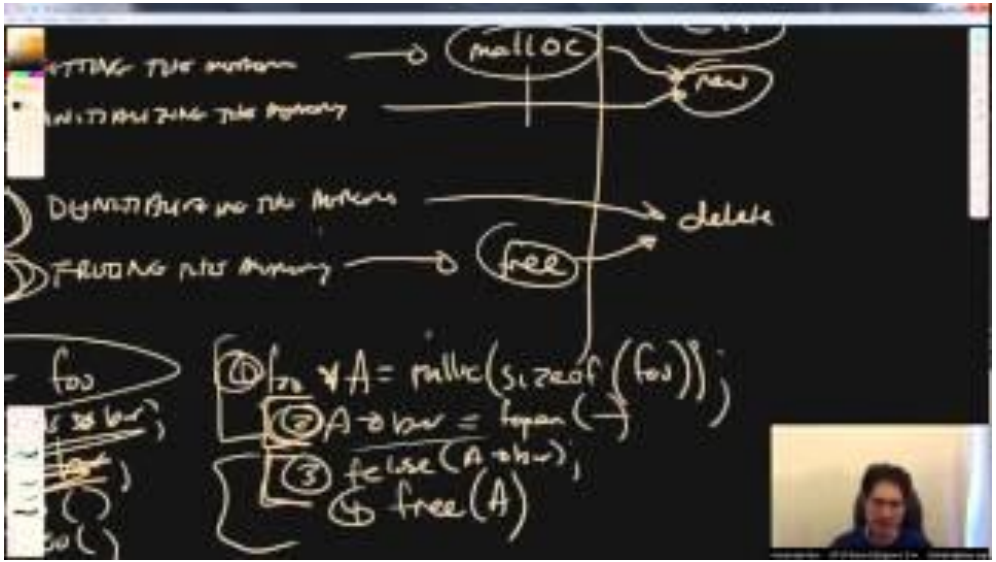
Полиморфизм под-типов

```
vector v = {1, 2, 3, 4};
…
list l = {5, 4, 3, 2, 1};
…
abstract_data_t *a = &v;
abstract_data_t *b = &l;
…
a->find(4);
b->find(4);
```

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/polymorphism

Виртуальные функции

https://youtu.be/zjkuXtiG1og



```
auto x = prod({1, 2, 3}, {4, 5, 6});

auto y = prod({1.0, 2.0}, {4.5, 5.0});

auto z = prod({"a", "b"}, {"c", "d"});
```

Перегрузка функций (ad-hoc)

```
int prod(int (&a)[10], int (&b)[10]) {
    int k = 0;
    for(size_t i = 0; i < 10; i++) {
        k += a[i] * b[i];
    }
    return k;
}

std::string prod(std::string (&a)[10], std::string(&b)[10]) {
    std::string k = "";
    for(size_t i = 0; i < 10; i++) {
        k += a[i] + b[i];
    }
    return k;
}
```

Параметризация в C++

```
template <typename T, size_t n>
auto prod(const T(&a)[n], const T(&b)[n]) {
    T k = 0;
    for(size_t i = 0; i < n; i++) {
        k += a[i] * b[i];
    }
    return k;
}
```

Параметризация в C++

```
template <typename T, size_t n>
auto prod(const T(&a)[n], const T(&b)[n]) {
    T k = T{};
    for(size_t i = 0; i < n; i++) {
        k += a[i] * b[i];
    }
    return k;
}
```

https://youtu.be/FshTrPe_Woc?t=128

Виды полиморфизма

- Универсальный (единая структура типов)
  - параметризация
  - включение
- Специальный (разная структура типов)
  - перегрузка
  - приведение

Runtime vs compile-time

- Написанный **код не указывает** явно на **типы** данных, которые используются в приложении, вместо этого логика программы использует основные **свойства ожидаемых типов**
- Детали **реализации** подбираются или **уточняются во время работы** приложения (dynamic dispatch)
- На это можно смотреть как на **один общий тип**, который **конфигурируется при обращении** к объекту, чтобы предоставлять **разное поведение**

https://www.youtube.com/watch?v=uM72qP5Wh18

Что такое OOP?

- инкапсуляция (абстракция состояния и избавление от манипуляции состоянием в коде, **public** и **private**)
- ~~наследование~~
- ~~полиморфизм~~
- делегация? (**forwarding**, **move** semantics)
- динамическое связывание (**виртуальные функции**)
- абстракция (скрытая реализация, открытый **интерфейс**)
- подмена типов? (subtyping)

Что такое OOP?

- инкапсуляция (абстракция состояния и избавление от манипуляции состоянием в коде, **public** и **private**)
- ~~наследование~~
- ~~полиморфизм~~
- делегация? (**forwarding**, **move** semantics)
- динамическое связывание (**виртуальные функции**)
- абстракция (скрытая реализация, открытый **интерфейс**)
- ~~подмена типов~~
- обобщенные типы данных (**generics**)