Объектно-ориентированное программирование

Object-oriented programming

Работа над ошибками

Intermission

Цитаты великих людей

“В контексте ООП, состояние объекта может быть рассмотрено, как информация, характеризующая текущее состояние этого объекта.”

“Например, можно создать объект "Автомобиль", который имеет цвет белый, марку Toyota, модель Mk 2, турбина Garrett 35, столб манящий и скорость 300 км/ч.”

Аааааааааааааааааааааааааавтомобиль!

leon.brinzan@iis.utm.md

Инкапсуляция, наследование, полиморфизм

“Класс в ООП – это …”

“Очередь сообщений – это …”

“Объект в программировании – это …”

“Сокрытие в программировании – это …”

Цитаты великих людей

“Используйте код с осторожностью. Подробнее…

В целом, выбранные варианты являются правильными и обоснованными.”

“Используя эти два куска кода, мы создаем структуру, в которой можно хранить числа и инициализировать их сразу при создании объекта.

2 / 2 Отправить сообщение ChatGPT may prod”

Ну вы поняли

Инкапсуляция и сокрытие данных

```
class rational {
public:
    rational() = delete;
    rational(int x, int y) {
        assert(y != 0);
        if(y < 0) {
            p = -x;
            q = -y;
        }
        else {
            p = x;
            q = y;
        }
        if(x != 0) reduce(*this);
    }
```

```
    void reduce(rational &r);
    void set_p(int x) {
        p = x;
    }
    void set_q(int x) {
        assert(x != 0);
        q = x;
    }
    friend bool operator<(
        const rational &lhs,
        const rational &rhs) const;
private:
    int p;
    int q;
};
```

Сравнение рациональных чисел

Для **b**, **d** не равных 0:

a / b < c / d ⇔ a * d < c * b

Тогда:

```
assert(rational(2, 3) < rational(-3, -4));      // ?
assert(rational(1, 2) <= rational(2, 4));       // ?
```

При чем тут алгебра?

“My math background made me realize that **each object could have several algebras associated with it**, and there could be **families** of these, and that these would be very very useful. The term "*polymorphism*" was imposed much later (I think by Peter Wegner) and **it isn't quite valid**, since it really comes from the nomenclature of functions, and *I wanted* quite a bit *more than functions*. I made up a term "genericity" for dealing with **generic behaviors in a quasi-algebraic form**.”

A. Kay

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

Принципы сравнения объектов в C++20

<compare>

1. std::**strong_ordering**
2. std::**partial_ordering**
3. std::**weak_ordering**
4. **operator** ⇔ (concept **three_way_comparable**)

Частично и линейно упорядоченное множество

ЧУМ (partial order):

Для всех **a, b, c** в **P:**

1. **a <= a**
2. **if a <= b and b <= a then a == b**
3. **if a <= b and b <= c then a <= c**

ЛУМ (total order):

Для всех **a, b, c** в ЧУМ **T:**

1. **a <= a**
2. **if a <= b and b <= a then a == b**
3. **if a <= b and b <= c then a <= c**
4. **a <= b or b <= a**

https://en.wikipedia.org/wiki/Total_order

Примеры

1. **Сравнение рациональных чисел (strong order)**
2. **Сравнение чисел с плавающей запятой (partial order)**
3. **Сравнение точек в декартовой системе координат (weak order)**

https://en.wikipedia.org/wiki/Weak_ordering

Так что там с **swap**?

```
class X {
    int *p{nullptr};
public:
    X() = default;
    X(int *q) : p(q) {}
    void swap(X &right) noexcept {
        auto tmp = p;
        p = right.p;
        right.p = tmp;
    }
    friend void swap(
        X &left, X &right) noexcept
    {
        left.swap(right);
    }
};
```

```
#include <algorithm>

using std::swap;

int main()
{
    X a = X(new int);
    X b;
    swap(a, b);
}
```

Сколько в этом коде определений функции **swap**?

Зависит от вызова конкретной функции

```
class X {
    void swap(X &right) noexcept;
    friend void swap(X &left, X &right) noexcept;
};

// libstdc++/move.h
namespace std {
    template<typename _Tp> inline void swap(_Tp& __a, _Tp& __b);
};
```

std::swap

```
#define _GLIBCXX_MOVE(__val) std::move(__val)

template<typename _Tp>
inline void
swap(_Tp& __a, _Tp& __b) {
        // concept requirements
        __glibcxx_function_requires(_SGIAssignableConcept<_Tp>)
        _Tp __tmp = _GLIBCXX_MOVE(__a);
        __a = _GLIBCXX_MOVE(__b);
        __b = _GLIBCXX_MOVE(__tmp);
}
```

std::move

```
template<typename _Tp>
inline typename std::remove_reference<_Tp>::type&&
move(_Tp&& __t) {
    return static_cast<typename
        std::remove_reference<_Tp>::type&&>(__t);
}
```

std::remove_reference

```
// libstdc++/type_traits.h
/// remove_reference
template<typename _Tp>
struct remove_reference
{ typedef _Tp   type; };

template<typename _Tp>
struct remove_reference<_Tp&>
{ typedef _Tp   type; };

template<typename _Tp>
struct remove_reference<_Tp&&>
{ typedef _Tp   type; };
```

swap vs. ::swap, https://godbolt.org/z/4aMf1zf48

```
swap(X&, X&):
  push rbp
  mov rbp, rsp
  sub rsp, 16
  mov QWORD PTR [rbp-8], rdi
  mov QWORD PTR [rbp-16], rsi
  mov rdx, QWORD PTR [rbp-16]
  mov rax, QWORD PTR [rbp-8]
  mov rsi, rdx
  mov rdi, rax
  call X::swap(X&)
```

```
X::swap(X&):
  push rbp
  mov rbp, rsp
  mov QWORD PTR [rbp-24], rdi
  mov QWORD PTR [rbp-32], rsi
  mov rax, QWORD PTR [rbp-24]
  mov rax, QWORD PTR [rax]
  mov QWORD PTR [rbp-8], rax
  mov rax, QWORD PTR [rbp-32]
  mov rdx, QWORD PTR [rax]
  mov rax, QWORD PTR [rbp-24]
  mov QWORD PTR [rax], rdx
  mov rax, QWORD PTR [rbp-32]
  mov rdx, QWORD PTR [rbp-8]
  mov QWORD PTR [rax], rdx
```

swap vs. ::swap

```
  lea rdx, [rbp-16]
  lea rax, [rbp-8]
  mov rsi, rdx
  mov rdi, rax
  call std::enable_if<
       std::__and_<
            std::__not_<
            std::__is_tuple_like<X> >,
       std::is_move_constructible<X>,
       std::is_move_assignable<X>
  >::value,
  void>::type std::swap<X>(X&, X&)

... std::move<X&>(X&):
  mov rbp, rsp
  mov QWORD PTR [rbp-8], rdi
  mov rax, QWORD PTR [rbp-8]
```

```
... std::swap<X>(X&, X&):
  mov rbp, rsp
  sub rsp, 32
  mov QWORD PTR [rbp-24], rdi
  mov QWORD PTR [rbp-32], rsi
  mov rax, QWORD PTR [rbp-24]
  mov rdi, rax
  call std::remove_reference<X&>::type&&
       std::move<X&>(X&)
  // 4x mov
  call ... std::move<X&>(X&)
  mov rdx, QWORD PTR [rbp-24]
  // 4x mov
  call ... std::move<X&>(X&)
  mov rdx, QWORD PTR [rbp-32]
  mov rax, QWORD PTR [rax]
  mov QWORD PTR [rdx], rax
```

Много это или мало?

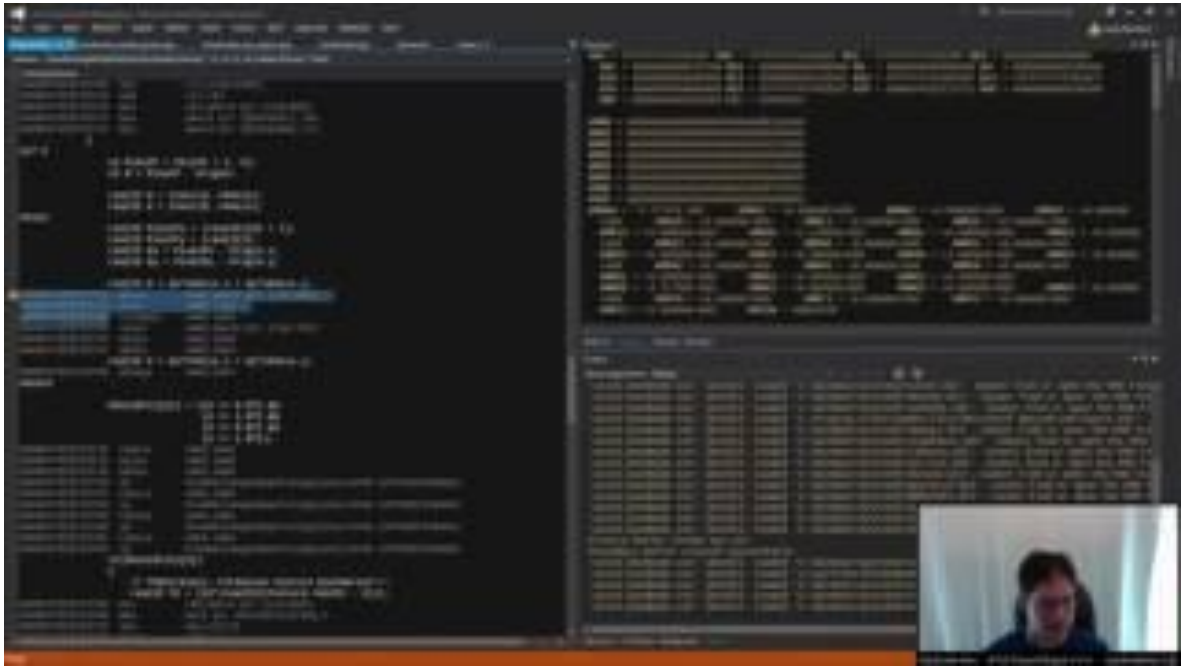
- Асимптотический анализ (complexity)
- Логическая организация кэша (cache locality)
- Динамический анализ кода (profiling)

https://en.wikipedia.org/wiki/Locality_of_reference

https://habr.com/ru/articles/179647/

“Zero-cost” abstraction

https://youtu.be/B2BFbs0DJzw



leon.brinzan@iis.utm.md