Объектно-ориентированное программирование

Object-oriented programming

III. Наследование

Inheritance

"I strongly felt then, as I still do, that there is no one right way of writing every program, and a language designer has no business trying to *force* programmers to use a particular **style**. The language designer does, on the other hand, have an obligation to encourage and **support a variety of styles and practices** that have proven effective and to provide language features and tools to help programmers avoid the well-known traps and pitfalls."

B. Stroustrup

"A History of C++: 1979-1991"

Влияние Simula на C++

- Классы ведут себя как сопрограммы (легко писать симуляции многопоточности)
- Классы позволяют мыслить о сущностях в программах напрямую
- Конструкторы, оператор new
- Статическая система типов данных (компилятор выявляет ошибки программиста еще до запуска программы)
- Программы легко организовываются в иерархию подпрограмм
- Сам язык оставлял желать лучшего (linking time, связывание классов занимало очень много времени, большие программы писать было тяжело; run-time type checking; garbage collection etc.)

https://dl.acm.org/doi/pdf/10.1145/234286.1057836

Выводы

"C++ was designed to provide Simula's **facilities for program organization** together with C's **efficiency** and **flexibility** for systems programming."

B. Stroustrup

- Организационная структура программы как в Simula (иерархии классов, многопоточность, статическая система типов)
- Высокая производительность (как при сборке программ, так и в работе)
- Портативность ("железо", операционные системы)

Абстрактные типы данных против классов

```
struct date { int day, month, year; };
struct date today;
extern void set_date();
extern void next_date();
extern void print_date();
extern void next_today();
```

```
class date {
    int day, month, year;
    friend void set_date(date*, int, int, int),
                next_date(date*),
                print_date(date*), next_today();
};
```

Основные инструменты C++

1. Виртуальные функции.
2. Перегрузка функций и операторов.
3. Ссылочные переменные.
4. Константные переменные - **const**.
5. Ручное управление динамической памятью.
6. Строгая система статической проверки типов данных.

leon.brinzan@iis.utm.md

Виртуальные функции*

```
enum kind { circle, triangle, square };

class shape {
    point center;
    color col;
    kind k; // необходимо дополнительное поле для конкретизации вида фигуры
public:
    // реализация открытых методов здесь
    void draw() {
        switch(k) {
        case circle: // логика, которая рисует круг
            break;
        case triangle: // логика, которая рисует треугольник
            break;
        case square: // логика, которая рисует квадрат
        }
    }
};
```

Виртуальные функции

```
class shape {
    point center;
    color col;
    // дополнительное поле не нужно
public:
    virtual void draw();
    // реализация открытых методов здесь
};

void draw_all(shape** v, int size) {
    for(int i = 0; i < size; ++i) v[i].draw();
}

class circle : public shape {
    int radius;
public:
    void draw () { /* логика, которая рисует круг */ }
};
```

Особенности наследования

- Implementation inheritance vs. interface inheritance
  - private vs. public
- Множественное наследование
  - Комбинирование классов в один таким образом, чтобы дочерний класс описывал **объекты**, которые могут себя вести как **любой из своих базовых классов**
- Абстрактные базовые классы
  - Позволяют изменять реализацию без компилирования всей иерархии
  - Позволяют явно выделить интерфейс в отдельный класс
- Inheritance vs. containment
  - Явное включение базового класса в большинстве случаев эквивалентно

Наследование против композиции

```
class stack {
    int *first;
    int *last;
    int *total;
public:
    // здесь работа с ресурсами

    void push_back();
    void pop_back();
};
```

```
class queue : public stack {
public:
    // здесь работа с ресурсами

    void push_back() { stack::push_back(); }
    void pop_front();
};
```

```
class deque : public queue {
public:
    // здесь работа с ресурсами

    void push_back() { stack::push_back(); }
    void pop_back() { stack::pop_back(); }
    void pop_front() { queue::pop_front(); }
    void push_front();
};
```

Наследование против композиции

```
class deque {
    int *first;
    int *last;
    int *total;
public:
    // здесь работа с
ресурсами

    void push_front();
    void push_back();
    void pop_front();
    void pop_back();
};
```

```
class stack {
    deque data;
public:
    void push_back() { data.push_back(); }
    void pop_back() { data.pop_back(); }
};
```

```
class queue {
    deque data;
public:
    void push_front() { data.push_front(); }
    void pop_front() { data.pop_front(); }
};
```

"*Object-oriented* programming is **programming using inheritance**. *Data abstraction* is **programming using user-defined types**. With few exceptions, object-oriented programming can and ought to be a *superset of data abstraction*."

B. Stroustrup

Associations of Simula Users Conference, 1986