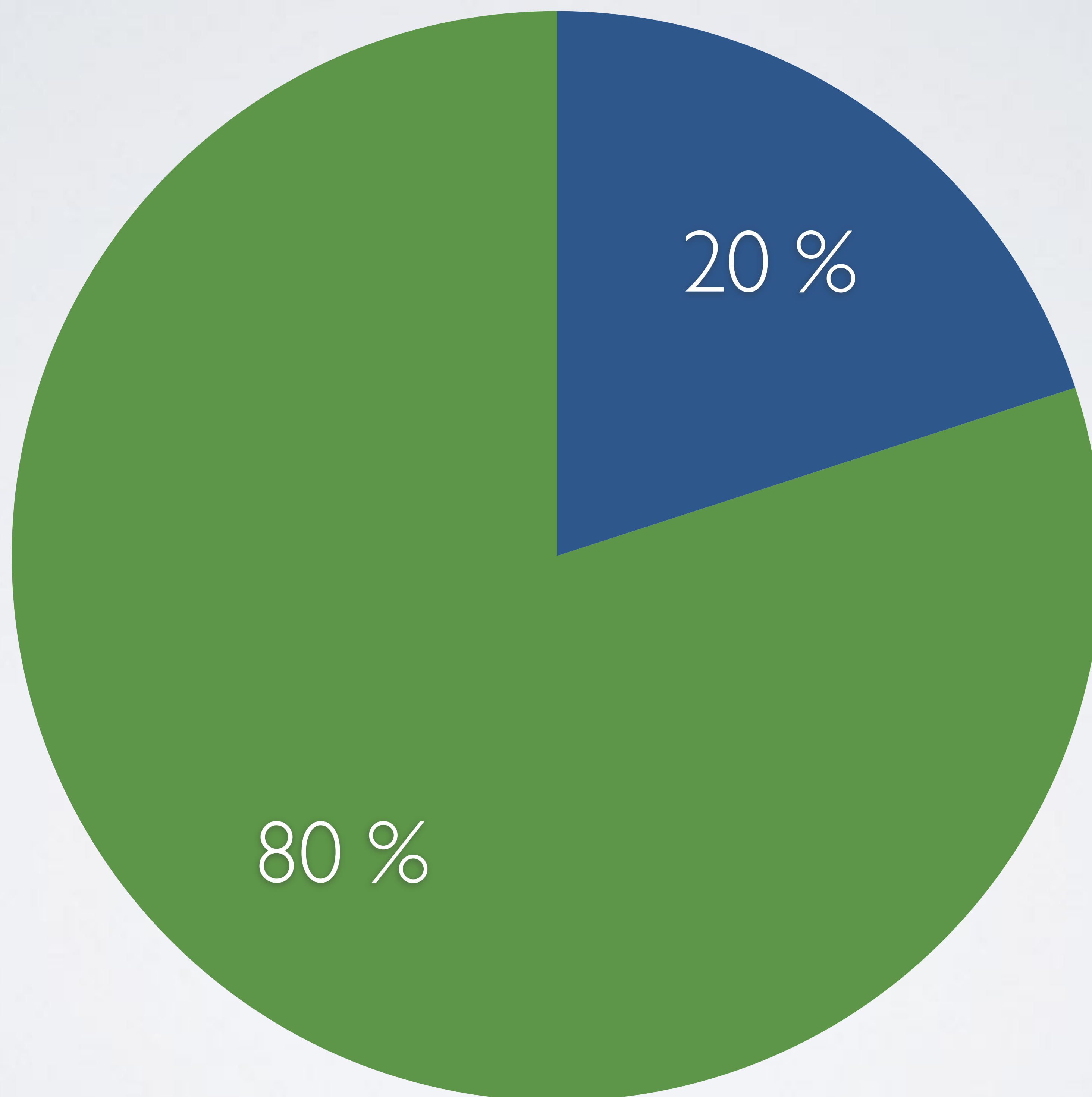
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция № 1 / **04**
26.02.2018 г.

Объектно-ориентированное Программирование



ПРЕОБРАЗОВАНИЕ ТИПОВ

```
int x = 100;
unsigned y, z;

y = (unsigned)x;     // старый способ из C
z = unsigned(x);     // новый способ из C++

printf("%u %u\n", y, z);  // 100 100
```

CONST

```
// Замена для #define
const int SQUARE_3 = 9;
const int SQUARE_9 = 81;
const double PI = 3.14159265358979323846264338;
```

1. Способ задать константы

```
int square(const int &n) {
    // Изменить n здесь невозможно
    return n * n;
}

void square2(int &n) {
    n *= n;
}

void f(int i) {
    int j    = square(i);
    int nine = square(3);

    int k = i;
    square2(k);   // OK
    square2(3);   // Ошибка!
}
```

2. Модификатор для ссылок и указателей

```
const int *p1;

int *const p2;
```

int const *p1	int *const p2

`p1++;       // OK`
`*p1;        // OK`
`*p1 = 10;   // ERROR`
	`p2++;       // ERROR`
	`*p2;        // OK`
	`*p2 = 10;   // OK`

```
class Foo {
public:
    Foo() : foo(0) {}

    int getFoo() const    { return foo; }
    void setFoo(int _foo) { foo = _foo; }
private:
    int foo;
};

//----------------------------------------

Foo f;
Foo &fr = f;

fr.setFoo(100);
fr.getFoo();                // 100

const Foo &frc = f;
frc.getFoo();               // 100
frc.setFoo(200);            // Ошибка!
```

3. const-члены класса

ПЕРЕГРУЗКА

Возможность иметь функции с разными телами и списками аргументов, но с одним именем

```
int sqrt(int n);
double sqrt(double d);
char *sqrt(char *);

void f() {
    int i      = sqrt(36);
    double d1 = sqrt(49.0);
    double d2 = sqrt(double(i*i));
    char *s    = sqrt("36");
}
```

Перегрузка функций

Прототип	GNU C++	Microsoft Visual C++
`void h(int);`	`_Z1hi`	`?h@@YAXH@Z`
`void h(int, char);`	`_Z1hic`	`?h@@YAXHD@Z`
`void h(void);`	`_Z1hv`	`?h@@YAXXZ`

«Манглинг» (mangling)

```
char *format_number(double d, int precision = 2);

format_number(10.0, 3);      // d: 10.0, precision: 3
format_number(10.0, 2);      // d: 10.0, precision: 2
format_number(10.0);         // d: 10.0, precision: 2
```

```
void f(int a, int b = 2, int c = 3);  // OK
void g(int a = 1, int b = 2, int c);  // ERROR!
void h(int a, int b = 3, int c);      // ERROR!
```

Параметры функций по умолчанию

```
class Rational {
public:
    Rational(int n);
    Rational(int p, int q);
    Rational(double d);
    Rational(const Rational &r);

    // ....
private:
    int p, q;
};
```

Перегрузка конструктора

ПРОСТРАНСТВА ИМЁН

Способ создать отдельные области видимости для классов, констант, функций…

Модуль Awesome

```
class Something {
public:
    Something(const char *name);
    // ...
};
```

Модуль Joe

```
class Something {
public:
    Something();
    // ...
};
```

Конфликт имён

```
class AwesomeSomething {
public:
    AwesomeSomething(const char *name);
    // ...
};

typedef AwesomeSomething *AwesomeSomethingPtr;

const int AwesomeNumber = 42;

enum AwesomeTypes {
    AWESOME_FOO,
    AWESOME_BAR,
    /* ... */
};

void AwesomeGlobalFunction(AwesomeSomething *);
```

Так себе решение

```cpp
// awesome.h
namespace Awesome {
    class Something {
    public:
        Something(const char *name);
        // ...
    };

    typedef Something *SomethingPtr;

    const int Number = 42;

    enum Types { FOO, BAR, /* ... */ };

    void GlobalFunction(Something *);
}
```

Настоящее решение — создать пространство имён

```
// awesome.cxx

#include "awesome.h"

Awesome::Something::Something(const char *name) {
    int n = Number;   // здесь префикс не нужен!

    // ...
}

void Awesome::GlobalFunction(Awesome::Something *ps) {
    /* ... */
}
```

Для адресации используется префикс Awesome::

```cpp
// client.cxx

#include "awesome.h"


void f(int q) {
    Awesome::Something x;
    int n = Awesome::Number;

    if (q == Awesome::FOO) {
        ...
    }

    Awesome::GlobalFunction(&x);
}
```

```cpp
// client.cxx

#include "awesome.h"

using namespace Awesome;

void f(int q) {
    Something x;
    int n = Number;

    if (q == FOO) {
        ...
    }

    GlobalFunction(&x);
}
```

Клиентский код

```c
// something.c
void public_interface_function() {
    // ...
    internal_function2();
}

void one_more_public_interface_function() {
    internal_function1();
    // ...
}

static void internal_function1() {
    // ...
}

static void internal_function2() {
    // ...
}
```

Сокрытие деталей реализации, C-style

```cpp
// something.cxx
namespace {
    void internal_function1() {
        // ....
    }

    void internal_function2() {
        // ....
    }
}

void public_interface_function() {
    // ....
    internal_function2();
}

void one_more_public_interface_function() {
    internal_function1();
    // ....
}
```

C++ style. Безымянное пространство имён

```
namespace His_lib {
    class String { /* ... */ };
    class Vector { /* ... */ };
}

namespace Her_lib {
    class String { /* ... */ };
    class Vector { /* ... */ };
}

namespace My_lib {
    using namespace His_lib;
    using namespace Her_lib;

    using His_lib::String;    // разрешение возможных конфликтов
    using His_lib::Vector;    // разрешение возможных конфликтов

    class List { /* ... */ };
}
```

Отбор и селекция пространств имён

ШАБЛОНЫ

Способ управляемой генерации классов по заданным правилам

МОТИВАЦИЯ

- Вспомним АТД «Очередь».
- Двойная абстракция:
  - …от деталей реализации (способа хранения элементов в памяти).
  - **…от типа хранимых данных.**

```
class IntegerQueue { /* ... */ };

class DoubleQueue  { /* ... */ };

class WTFElseQueue { /* ... */ };
```

Шаблон класса

```
template <typename T>
class Queue {
public:
    Queue();

    void enqueue(const T &el);
    bool empty() const;
    T dequeue();
    const T &peek() const;
private:
    T *buf, *head;
    size_t buf_size;
};

class Person {
    /* ...... */
};

Queue<double> doubleQ;
Queue<int>    intQ;
Queue<Person> personQ;
```

ПЕРЕОПРЕДЕЛЕНИЕ ОПЕРАТОРОВ

```
struct Vector2D {
    double x, y;
    // ....
};

// можно писать так:
Vector2D x, y;
Vector2D z = x.add(y);
z = z.mul(x);
z = z.sub(x.div(y));
```

```
// а хочется писать так:
z = x + y;
z *= x;
z -= x / y;
```

```
// Магия C++:
z = x + y;
z *= x;
z -= x / y;
```

```
// Превращается в:
z = operator+(x, y);
z.operator*=(x);
z.operator-=(operator/(x, y));

// Остаётся только определить
// эти функции и методы…
```

```cpp
class Vector2D {
public:
    // ...
    Vector2D &operator*=(const Vector2D &other);
    Vector2D &operator*=(double x);
    Vector2D &operator-=(const Vector2D &other);
};

Vector2D operator+(const Vector2D &a, const Vector2D &b);
Vector2D operator/(const Vector2D &a, const Vector2D &b);
```

КОНЕЦ ЧЕТВЁРТОЙ ЛЕКЦИИ

```
Lection::~Lection() { }
```