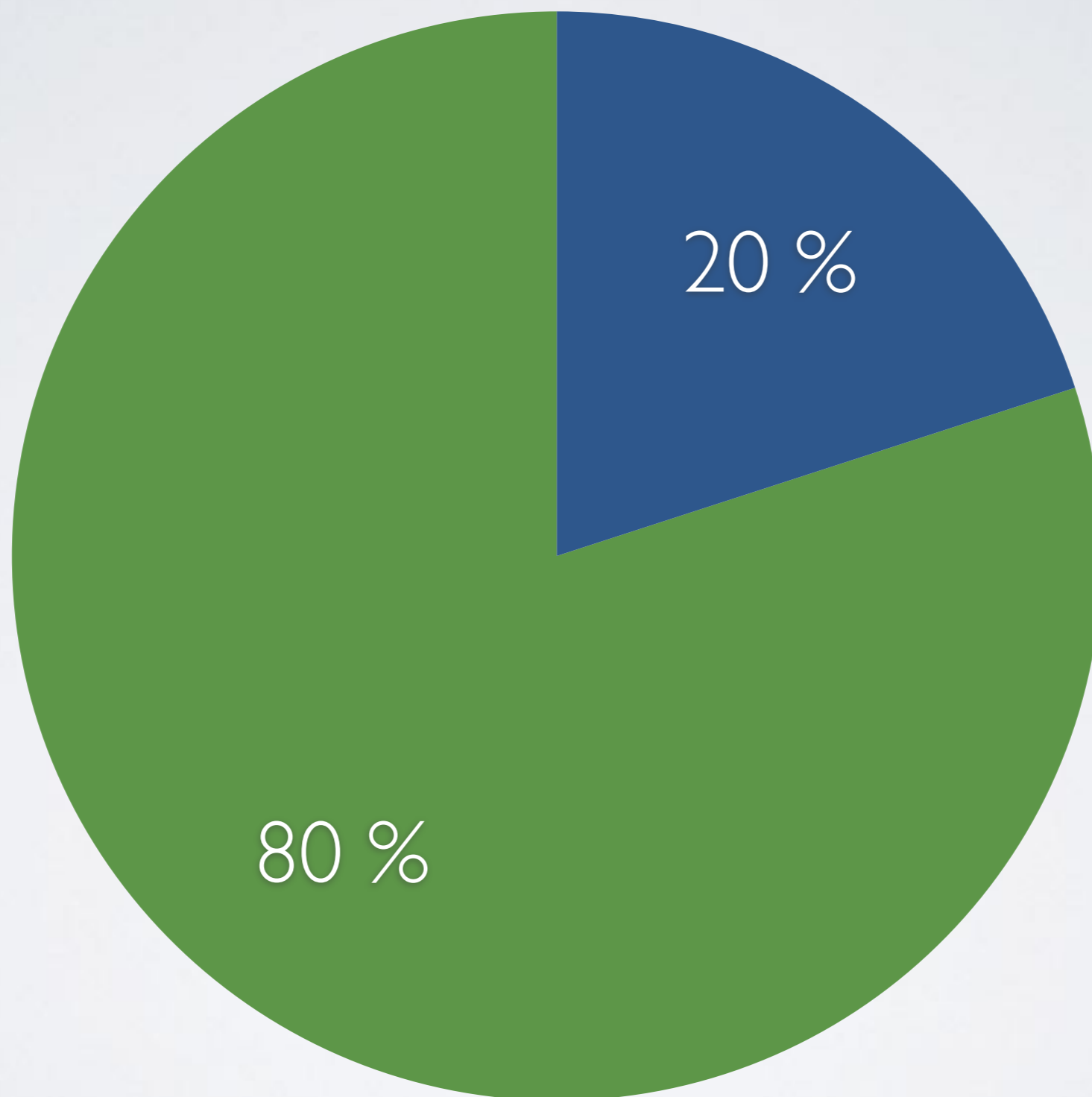


# ОБЪЕКТНО- ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ



Лекция № 1 / 04  
06.03.2017 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

```
// неплохая замена для #define  
const int SQUARE_3 = 9;  
const int SQUARE_9 = 81;  
const double PI = 3.141592653589793238462643383;
```

**I. 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);    // ОК  
    square2(3);    // Ошибка!  
}
```

## 2. Модификатор для ссылок и указателей

```
const int *p1;
```

```
int *const p2;
```

int const *p1	int *const p2
<pre>p1++; // OK</pre>	<pre>p2++; // ERROR</pre>
<pre>*p1; // OK</pre>	<pre>*p2; // OK</pre>
<pre>*p1 = 10; // ERROR</pre>	<pre>*p2 = 10; // OK</pre>

```
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++
<code>void h(int);</code>	<code>_Z1hi</code>	<code>?h@@YAXH@Z</code>
<code>void h(int, char);</code>	<code>_Z1hic</code>	<code>?h@@YAXHD@Z</code>
<code>void h(void);</code>	<code>_Z1hv</code>	<code>?h@@YAXXZ</code>

**«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**

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

    typedef Something *SomethingPtr;

    const int Number = 42;

    enum Types { F00, 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::`**

```
// client.cxx

#include "awesome.h"

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

    if (q == Awesome::F00) {
        ...
    }

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

```
// client.cxx

#include "awesome.h"

using namespace Awesome;

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

    if (q == F00) {
        ...
    }

    GlobalFunction(&x);
}
```

**КЛИЕНТСКИЙ КОД**

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

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

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

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

**Соккрытие деталей реализации, C-style**

```
// 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



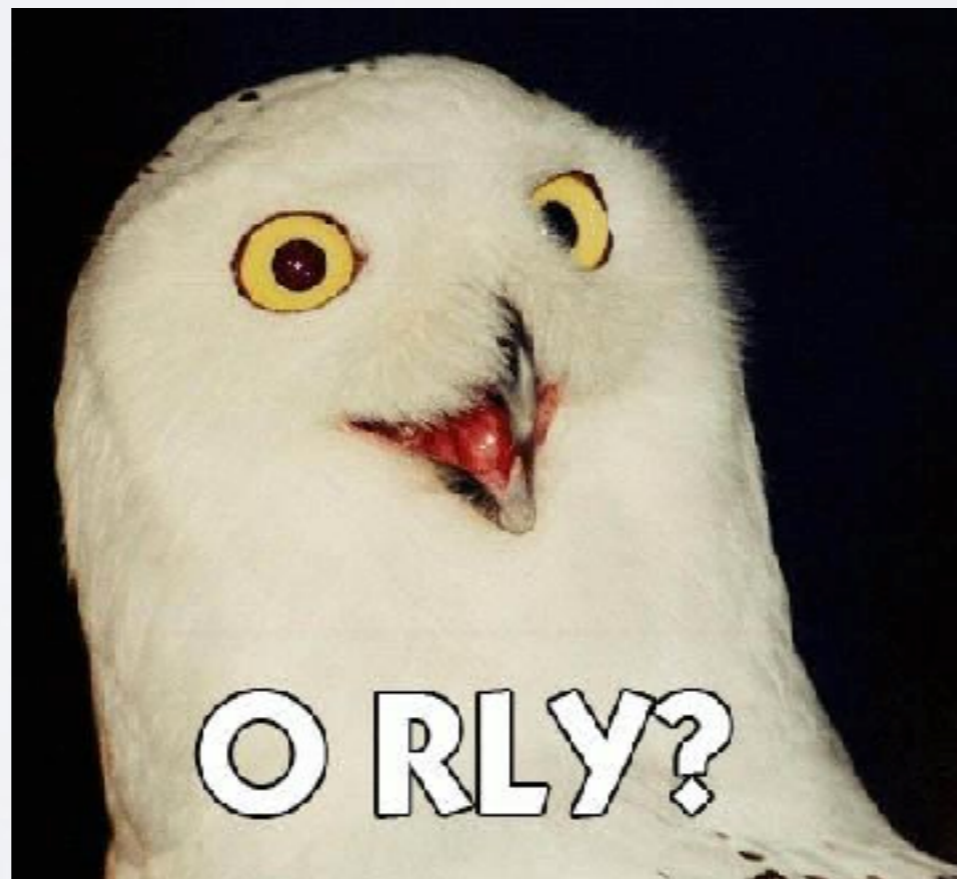
# МОТИВАЦИЯ

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

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

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

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



```
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;
```

## Шаблон класса

# 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...

```
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

```
Lecture::~~Lecture() { }
```