Конкретные классы

Блиц-опрос

1. Зачем нужны классы?
   (Какой критерий выбора между классом и структурой?)
2. Зачем нужен конструктор?
3. Зачем нужна инкапсуляция?

Покритикуйте этот код

```
struct IntVector {
  int* begin_;
  int* end_;
  int* capacity_;
};

size_t vector_size(const IntVector* v) {
  return ???;
}
```

Покритикуйте этот код

```
struct IntVector {
  int* begin_;
  int* end_;
  int* capacity_;
};

size_t vector_size(const IntVector* v) {
  return v->end_ - v->begin_;
}
```

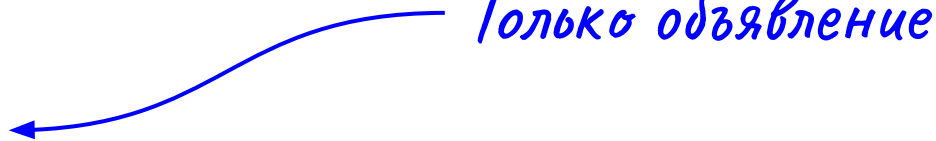
Иллюстрация проблемы

Можно ли защититься от написания такого кода?
Как это возможно в языке C?

```
IntVector* iv = vector_new(/*size=*/42);
iv->end_ = iv->begin_ - 1;
```

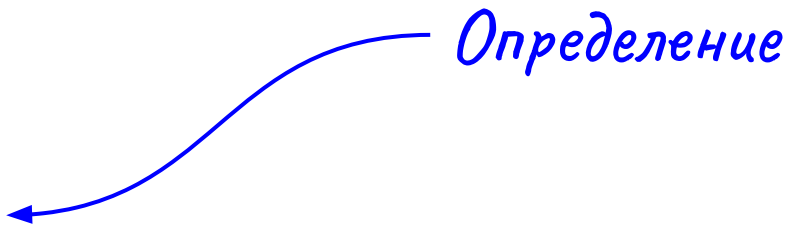
Инкапсуляция в языке C

```
// IntVector.h

struct IntVector;

IntVector* vector_new(size_t size);
void vector_push_back(IntVector* v);
size_t vector_size(const IntVector* v);
```



Инкапсуляция в языке C

```
// IntVector.c
#include “IntVector.h”

struct IntVector { ... };

size_t vector_size(const IntVector* v) {
  return v->end_ - v->begin_;
}
```



Инкапсуляция в языке C

```
// IntVector.h

struct IntVector;

IntVector* vector_new(size_t size);
void vector_push_back(IntVector* v);
size_t vector_size(const IntVector* v);
```

Покритикуйте этот подход

Инкапсуляция в языке C

```
// IntVector.c

struct IntVector { ... };

IntVector* vector_new(size_t size) {
  IntVector* iv = malloc(...);
  ...
  return iv;
}
```

Вынужденная аллокация

Инкапсуляция в C++

```
class IntVector {
 public:
  void push_back(int value);
  std::size_t size() const;

 private:
  int* begin_;
  int* end_;
  int* capacity_;
};
```

Инкапсуляция в C++

Что мешает написать такой код?

```
IntVector v(42);
std::memset(&v, 0, sizeof(v));
```


https://godbolt.org/z/fa3qTa9PT

Инкапсуляция в C++

Что мешает написать такой код?

```
IntVector v(42);
std::memset(&v, 0, sizeof(v));
```

Мешает совесть

https://godbolt.org/z/fa3qTa9PT

Инкапсуляция в C++

- В C++ линейная модель памяти,
  поэтому модификаторы доступа не могут защитить данные.
- Сокрытие защищает **имена**.

```
error: 'int* IntVector::begin_' is private
       within this context
```

Пример конструктора

```
class IntVector {
 public:
  IntVector(std::size_t size) {
    begin_ = new int[size]();
    end_ = begin_ + size;
    capacity_ = end_;
  }
  ...
};
```

Написано не очень хорошо, доработаем позже

Еще раз

Класс

Конструктор

Инкапсуляция

Инвариант

Инвариант класса — это множество условий, сохраняющих свою истинность на протяжении всего времени жизни экземпляра класса.

Инвариант определяет внутренне непротиворечивое (согласованное) состояние объекта.

Примеры инвариантов

- vector
  - begin ≤ end ≤ capacity
- string
  - begin ≤ end ≤ capacity
  - c_str() — null-terminated string
- Rational (простая дробь, например 3/4)
  - ???

Примеры инвариантов

- vector
  - begin ≤ end ≤ capacity
- string
  - begin ≤ end ≤ capacity
  - c_str() — null-terminated string
- Rational (простая дробь, например 3/4)
  - знаменатель != 0
  - Дробь несократима

Объединяем механизмы

Класс создает контекст для инварианта.

Конструктор устанавливает инвариант.

Инкапсуляция помогает поддерживать инвариант.

Рекомендации

C.ctor: Constructors

C.40: Define a constructor if a class has an invariant

C.41: A constructor should create a fully initialized object

Класс Point

```
struct Point {
  double x_;
  double y_;
};
```

Какой инвариант у этого типа?

Класс Point

```
struct Point {
  double x_;
  double y_;
};
```

Инварианта может не быть.

И тогда класс не нужен, достаточно структуры.

Константный интерфейс

```
class Point {
 public:
  Point(double x, double y);

  double x() const;
  double y() const;

 private:
  double x_, y_;
};
```

Поля объекта нельзя менять независимо

https://godbolt.org/z/z1T7sGK8K

В чем разница?

```
class Point {
 public:
  Point(double x, double y);

  double x() const;
  double y() const;

 private:
  double x_, y_;
};
```

```
struct ROPoint {
  const double x_ = 0;
  const double y_ = 0;
};
```


https://godbolt.org/z/rWEhcW8Wq

Вспомним о синонимах типов

```
using DocumentId = std::size_t;
using TermPosition = std::size_t;

void f(DocumentId doc_id, TermPosition tp);

DocumentId doc_id = ...;
TermPosition tp = ...;

f(doc_id, tp); // OK
f(tp, doc_id); // OK
```

Идиома Strong Typedef

```
struct X {
  explicit X(std::size_t value)
    : value_(value) {}

  std::size_t value_;
};
```

Вместо X – имя вашего типа

Причины создания классов

- Моделирование объектов реального мира
- Моделирование абстрактных объектов
- Сокрытие деталей реализации
- Упрощение передачи параметров в методы
- **Упаковка родственных операций**
- и др.

Но:

- *Книга написана в контексте Java*
- *Содержит Java-специфичные идиомы*

«Совершенный код», 6.4

Java-specific пример

```
class Collections {
  public static
  int binarySearch(List<...> list, T key)

  public static
  void shuffle(List<...> list, Random rnd)

  public static
  <T> void copy(List<...> dest, List<...> src)
}
```


java.util.Collections

Где определить метод find?

```
class Vector {
 public:
  T* find(T value) const;
};

T find(Iterator begin, Iterator end);
```

1

2

Подход STL

- InputIt find(InputIt first, InputIt last, const T& value)

Внешняя функция подходит для:

– Массивов

– std::vector, std::list

– std::string

Все просто, так?

Подход STL

- InputIt find(InputIt first, InputIt last, const T& value)

Да, но:

- std::map::find
- std::set::find
- std::unordered_map::find
- std::unordered_set::find

Подход STL

- InputIt find(InputIt first, InputIt last, const T& value)

Да, но:

- std::map::find
- std::set::find
- std::unordered_map::find
- std::unordered_set::find

Ладно, но:

- std::string::find

Пример «как не надо»

```
class QuadraticEquationSolver {
 public:
  void set_a(double a); // + set_b, set_c
  void solve();

  int root_count() const;
  double x1() const;
  double x2() const;
 ...
};
```


https://godbolt.org/z/fEYeEfzTn

Пример «как не надо»

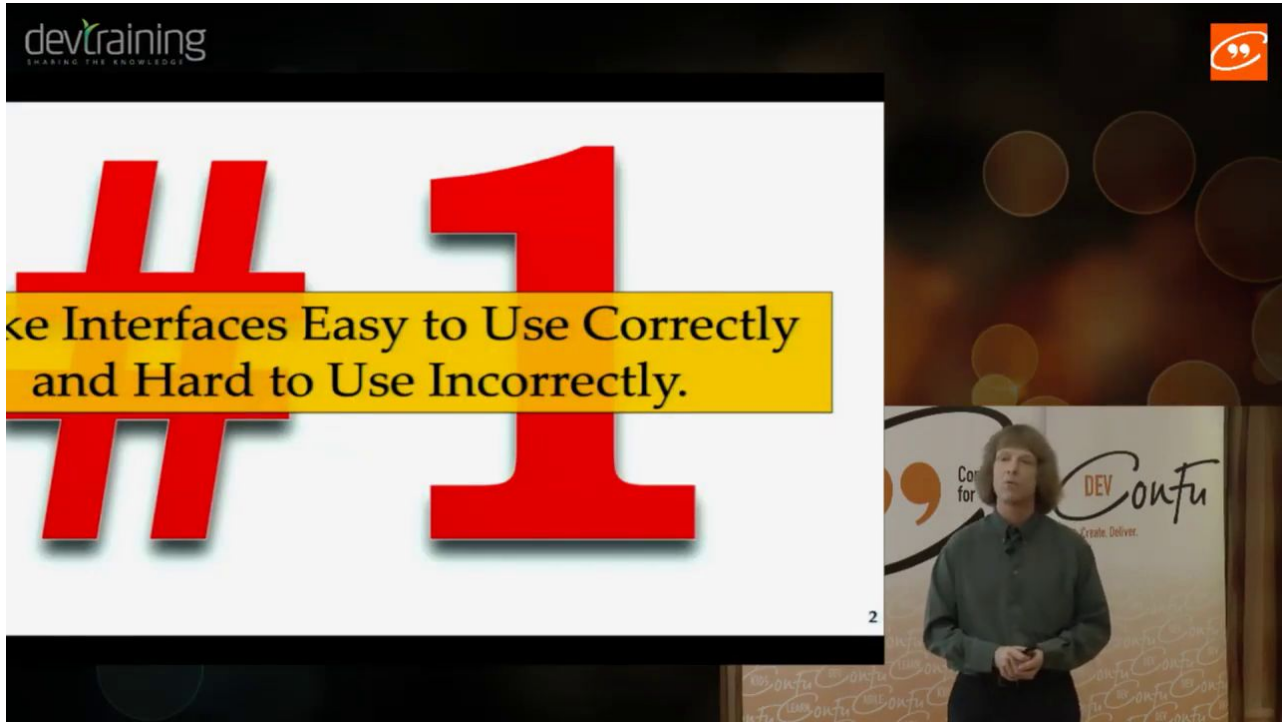
```
QuadraticEquationSolver solver;

solver.set_a(3);
solver.set_b(-14);
solver.set_c(-5);

solver.solve();

fmt::print("x1 = {} x2 = {}\n",
           solver.x1(), solver.x2());
```

Все это время объект в неконсистентном состоянии

Архитектурные проблемы такого солвера

1. У класса нет четкого инварианта.
2. Класс предполагает определенный порядок вызова методов, но не форсирует его.
3. Согласованность внутреннего состояния легко нарушить (вызов любого set-метода после solve).
4. Поля класса выполняют функции глобальных переменных.

The Most Important Design Guideline



https://www.bilibili.com/video/BV15g4y1878e/

Конструктор

Constructor is a **special** non-static member function of a class that is used to initialize objects of its class type.

Special. Very special.

constructor

Объявление функции

```
void f(int x);
```

1. Тип возвращаемого значения
2. Имя
3. Список параметров

Тип возвращаемого значения

```
void g() {}
void f() { return g(); } // Ok

class Widget {
 public:
  Widget() {
    return;       // Ok
    return g();   // Error
  }
};
```

Имя конструктора

```
class Widget {
 public:
  Widget() { ... }


  constructor() { ... }
};
```

Конструктор использует имя своего класса

Почему не выделили ключевое слово?

Указатель на функцию

```
void ff();

struct Widget {
  Widget() {}
  static void sf();
  void mf();
};
```

```
auto ffp = ff;
auto sfp = Widget::sf;
auto mfp = &Widget::mf;

ffp();
sfp();

Widget w;
(w.*mfp)();

auto cp = &Widget::Widget;
```


https://godbolt.org/z/YxT9ebbha

Converting Constructor

```
class Widget {
 public:
  /*explicit*/ Widget(int) {}
};

int main() {
  Widget w1(1);   // Ok, direct init
  Widget w2 = 1;  // Ok, copy init
}
```

Converting Constructor

```
class Widget {
 public:
  explicit Widget(int) {}
};

int main() {
  Widget w1(1);   // Ok, direct init
  Widget w2 = 1;  // Error, copy init
}
```


https://godbolt.org/z/ox7PT39P3

Converting Constructor

```
void f(Widget) { ... }


// Ok, если конструктор explicit(false)
// Error, если конструктор explicit(true)
f(1);
```

Когда explicit(false) удивляет

```
class Vector {
 public:
  Vector(int size);
};

void f(Vector v);

f(42);
```

Когда explicit(false) удобен

```
class Rational {
 public:
  Rational(int num = 0, int denom = 1) { ... }
};

void f(Rational r);

f(42);
```

Рекомендация

C.46: By default, declare single-argument constructors explicit

Exception If you really want an implicit conversion from the constructor argument type to the class type, don’t use explicit

CppCoreGuidelines: C.46

Что есть в пустом классе?

```
struct Widget {









};
```

Что есть в пустом классе?

```
struct Widget {
  Widget() = default;

  ~Widget() = default;
  Widget(const Widget&) = default;
  Widget(Widget&&) = default;
  Widget& operator=(const Widget&) = default;
  Widget& operator=(Widget&& w) = default;
};
```

Что есть в пустом классе?

```
clang++ -std=c++20 -Xclang -ast-dump -fsyntax-only

CXXRecordDecl struct Widget definition
 |-DefaultConstructor
 |-CopyConstructor
 |-MoveConstructor
 |-CopyAssignment
 |-MoveAssignment
 `-Destructor
```


https://godbolt.org/z/bYW1TzqY8

Порядок конструирования класса

1. Сначала конструкторы полей
2. Затем собственный конструктор

Live:

https://godbolt.org/z/9qPWGG1je

Инициализация констант и ссылок

```
struct Widget {
  Widget(int x): x_(), y_(x_), z_(y_) {}

  const int x_;
  int y_;
  int& z_;
};
```

Default init

```
struct Point {
  /*
  Point(double x = 0, double y = 0) :
    x_(x), y_(y) {}
  */

  double x_ = 0;
  double y_ = 0;
};
```

Delegating constructors

```
Rect(double top, double right,
     double bottom, double left)
  : top_(top), right_(right),
    bottom_(bottom), left_(left) { }


Rect(Point top_left, Point bottom_right)
    : Rect(top_left.y_, bottom_right.x_,
           bottom_right.y_, top_left.x_) { }
```


https://godbolt.org/z/vqPn8K1Wd