

ОБЪЕКТНО- ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ



Лекция № 1 / 10
17.04.2017 г.

ОТНОШЕНИЯ МЕЖДУ КЛАССАМИ

- Наследование
- Композиция
- Агрегация
- Ассоциация

Вспоминаем!

```
class Boss {  
    Slave slave;  
    std::unique_ptr<AnotherSlave> slave2;  
  
public:  
    Boss() : slave(),  
            slave2(new AnotherSlave)  
            {}  
};
```

Композиция

```
class Order {  
    // ...  
};
```

```
class User {  
    std::vector<Order *> orders;  
public:  
    // ...  
};
```

Агрегация

```
class Auditor {
public:
    // ...
    bool auditRead(const string &key);
    bool auditWrite(const string &key);
};
```

```
class AuditedKVStore {
    AuditedKVStore(Auditor &_auditor,
        map<string, string> *_storage)
        : auditor(_auditor),
          storage(_storage)
    {}
```

```
    string get(const string &key);
    // ...
```

```
private:
    Auditor &auditor;
    map<string, string> *storage;
};
```

Ассоциация

ГРАФИЧЕСКАЯ НОТАЦИЯ

SquareEquation

a, b, c: double

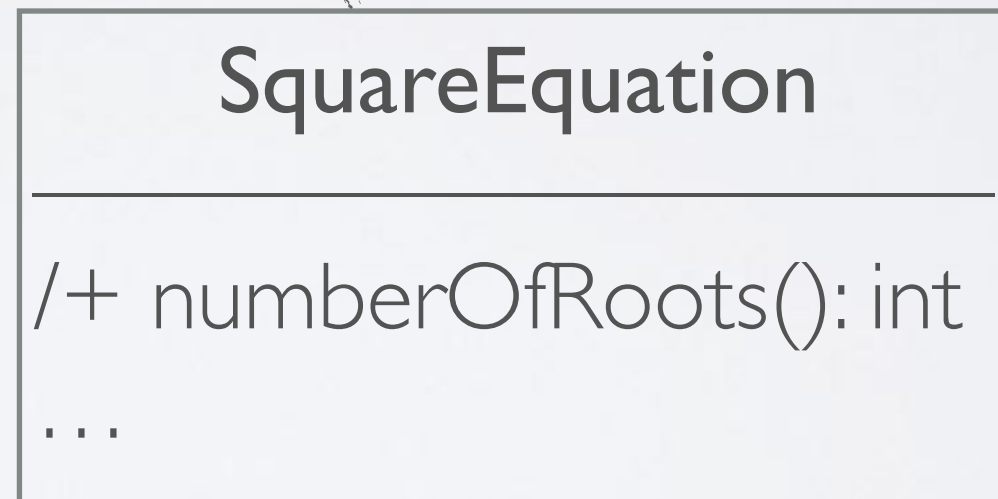
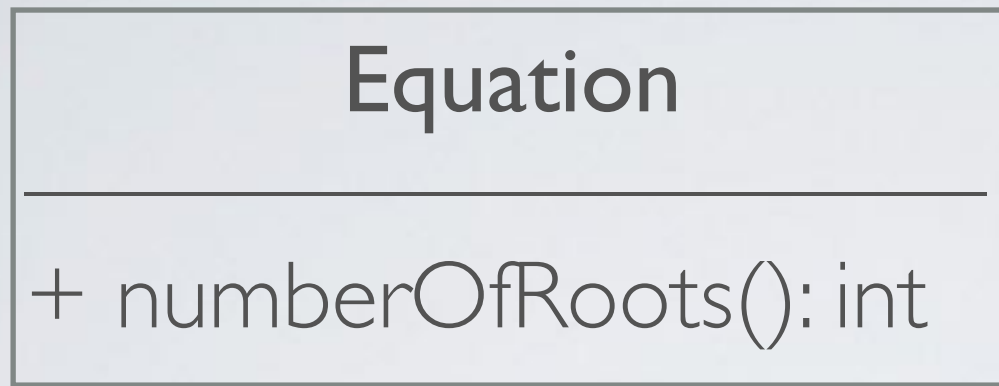
+ numberOfRoots(): int

+ root1(): double

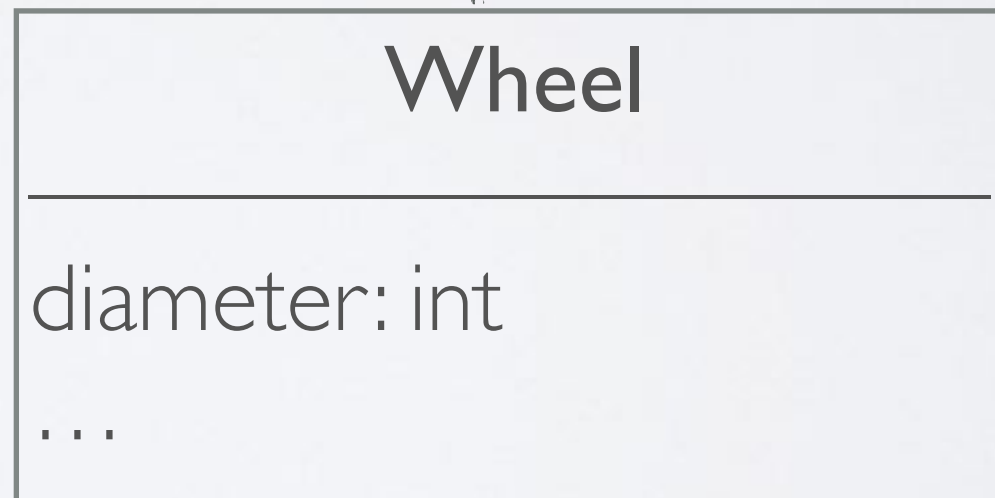
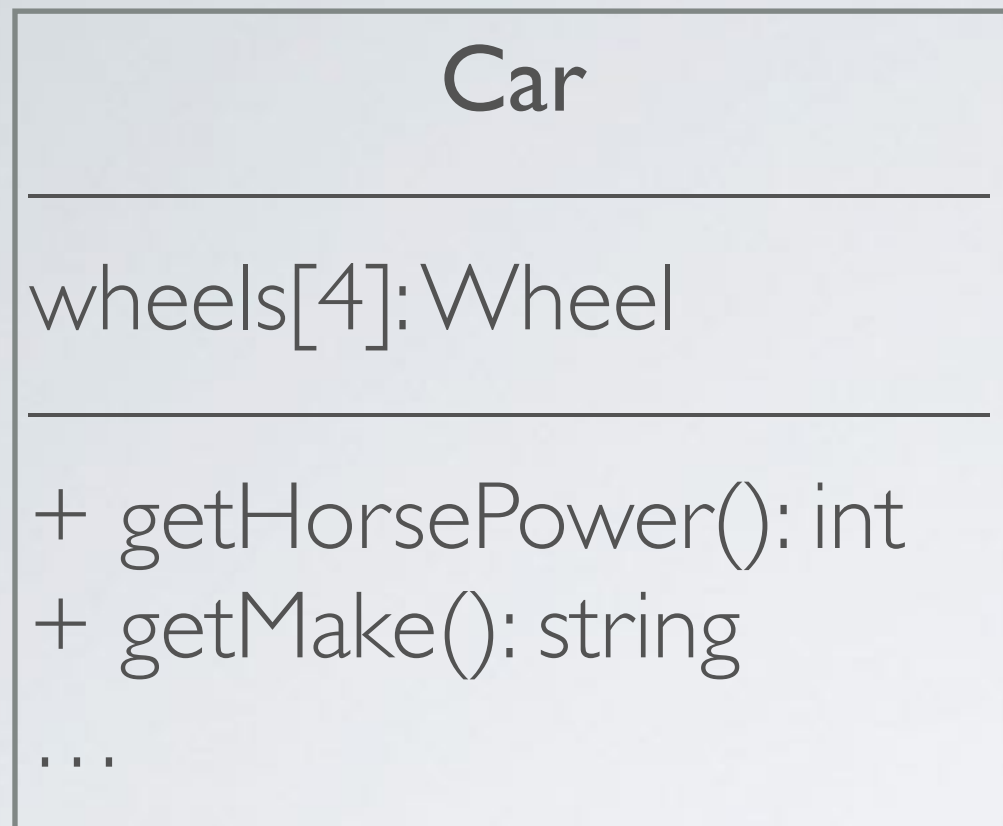
+ root2(): double

– discriminant(): double

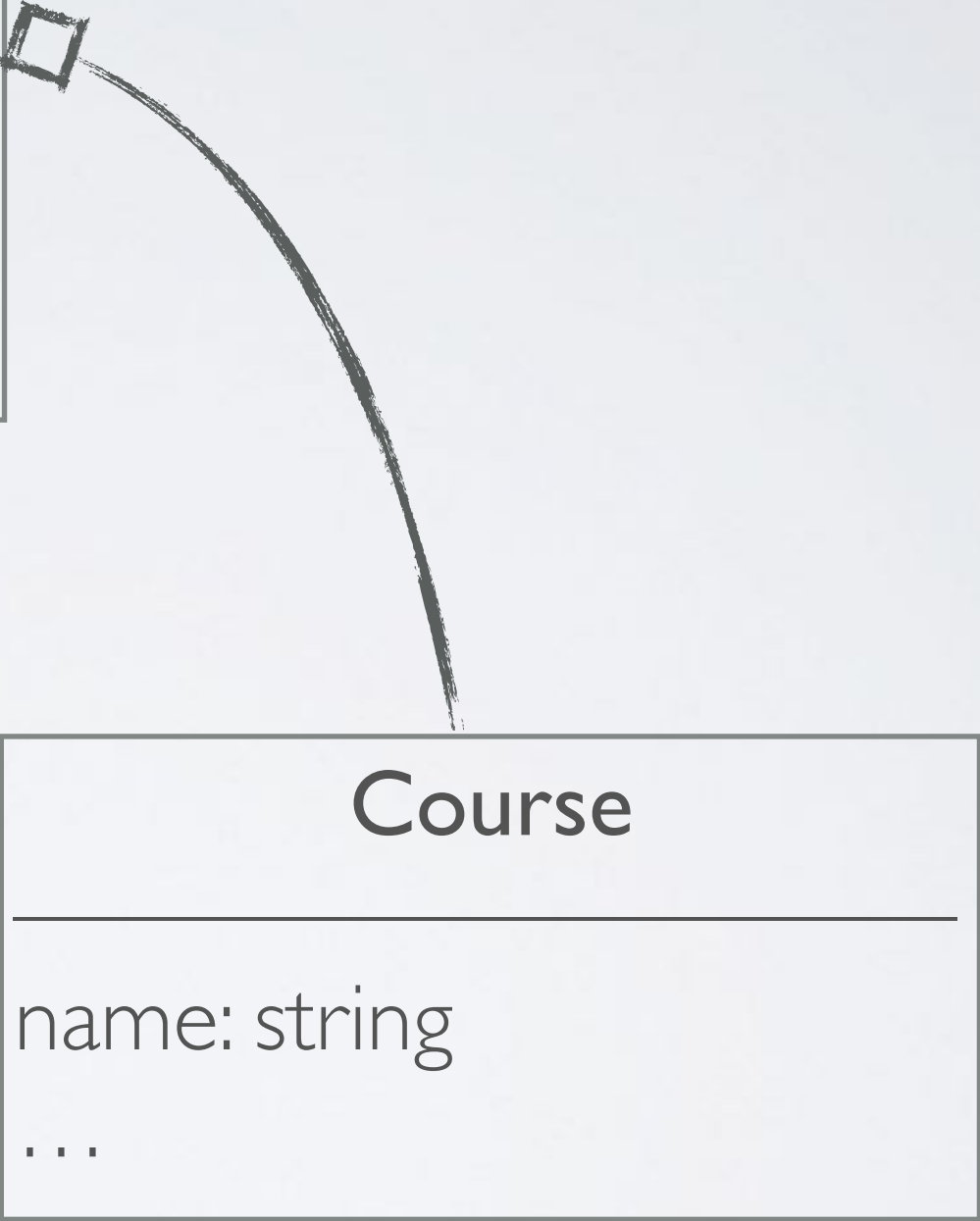
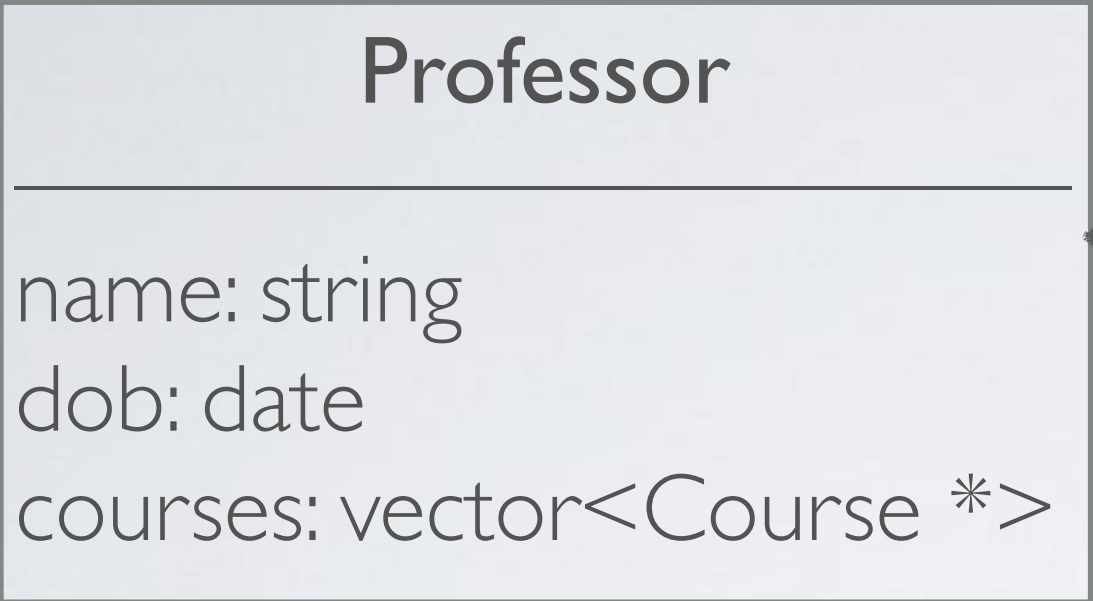
protectedMethod(): int



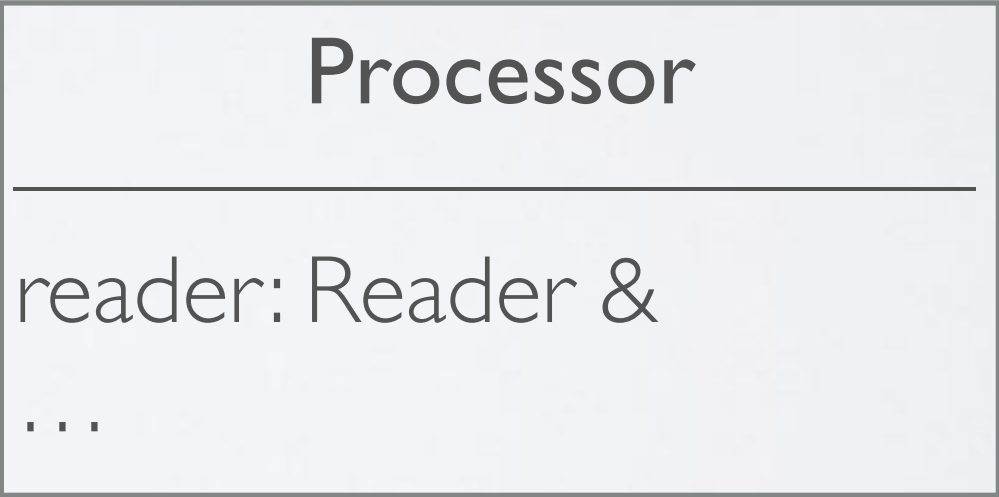
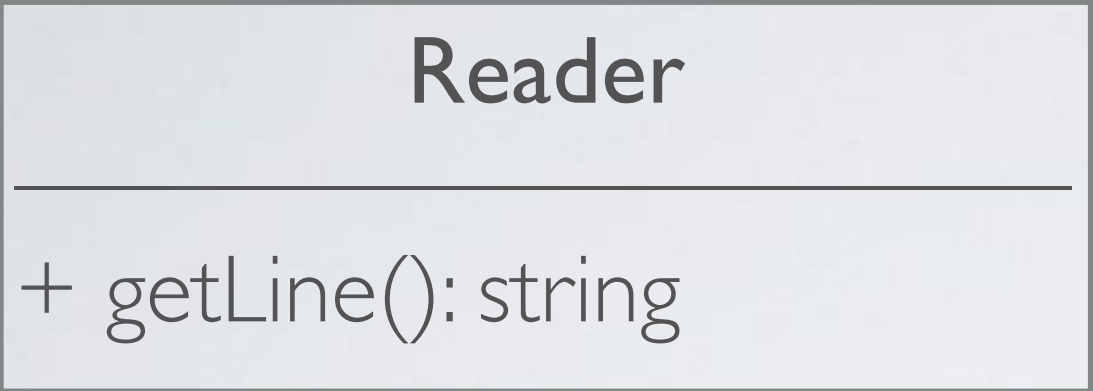
Наследование



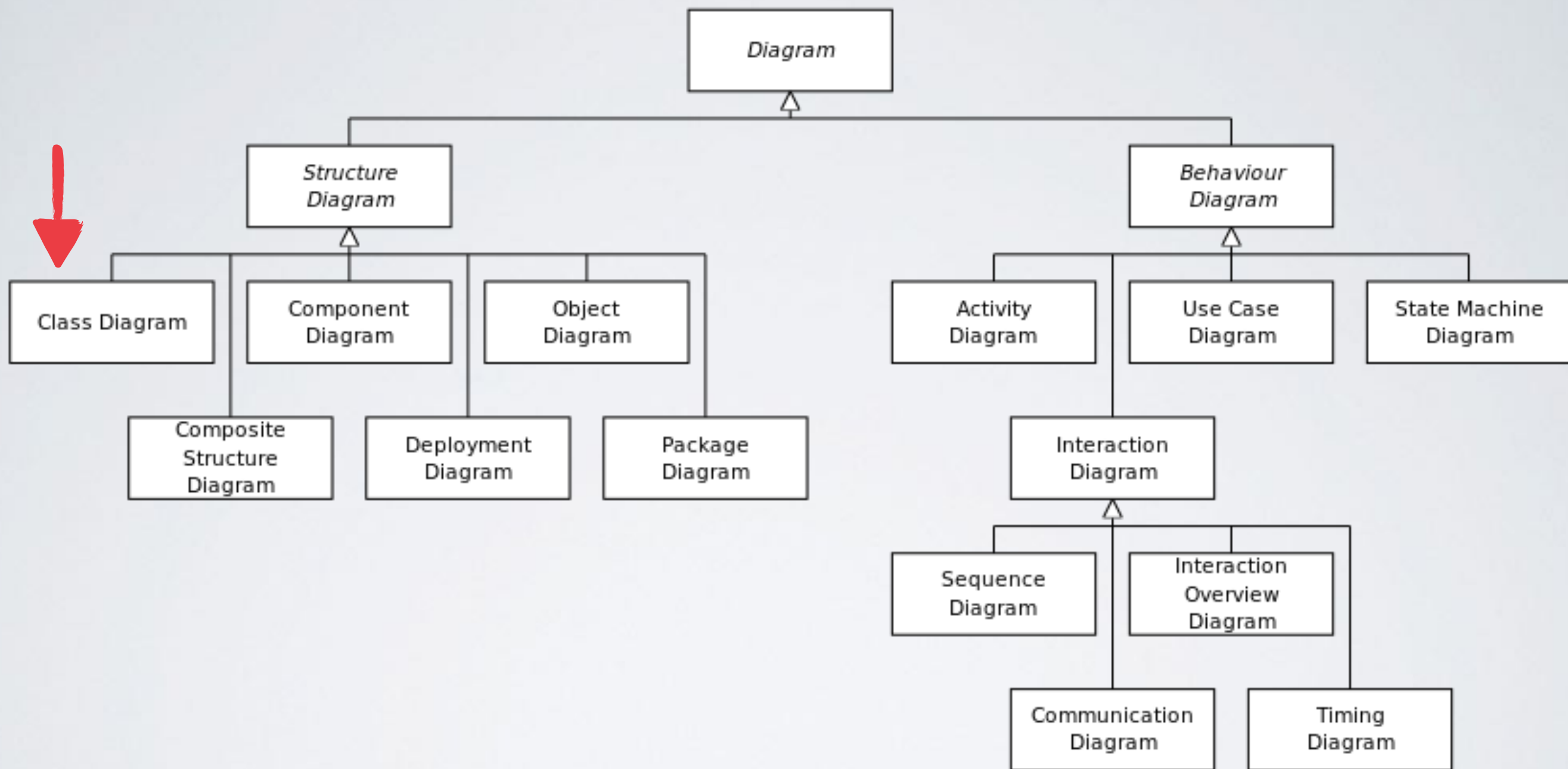
Композиция



Агрегация



Ассоциация



Все диаграммы UML

From: Начальник <boss@acme.com>

To: Кодер Вася <vasya@acme.com>

Date: Mon, 10 Nov 2014 07:59:11 +0600

Subject: СРОЧНО!!!!!!!!!!!!!! |

Вася,

нужно написать программу, форматирующую текстовые файлы по заданной ширине и с заданным выравниванием: по левому, правому краю и по центру. Абзацы разделены пустой строкой.

Как только что-то будет, присылай.

—

Начальник АСМЕ

TextFormatter

width: int

algorithm: enum { Left, Right, Center }

+ TextFormatter(width : int,
 algorithm: enum)

+ format(in_fn, out_fn: string)

God Object

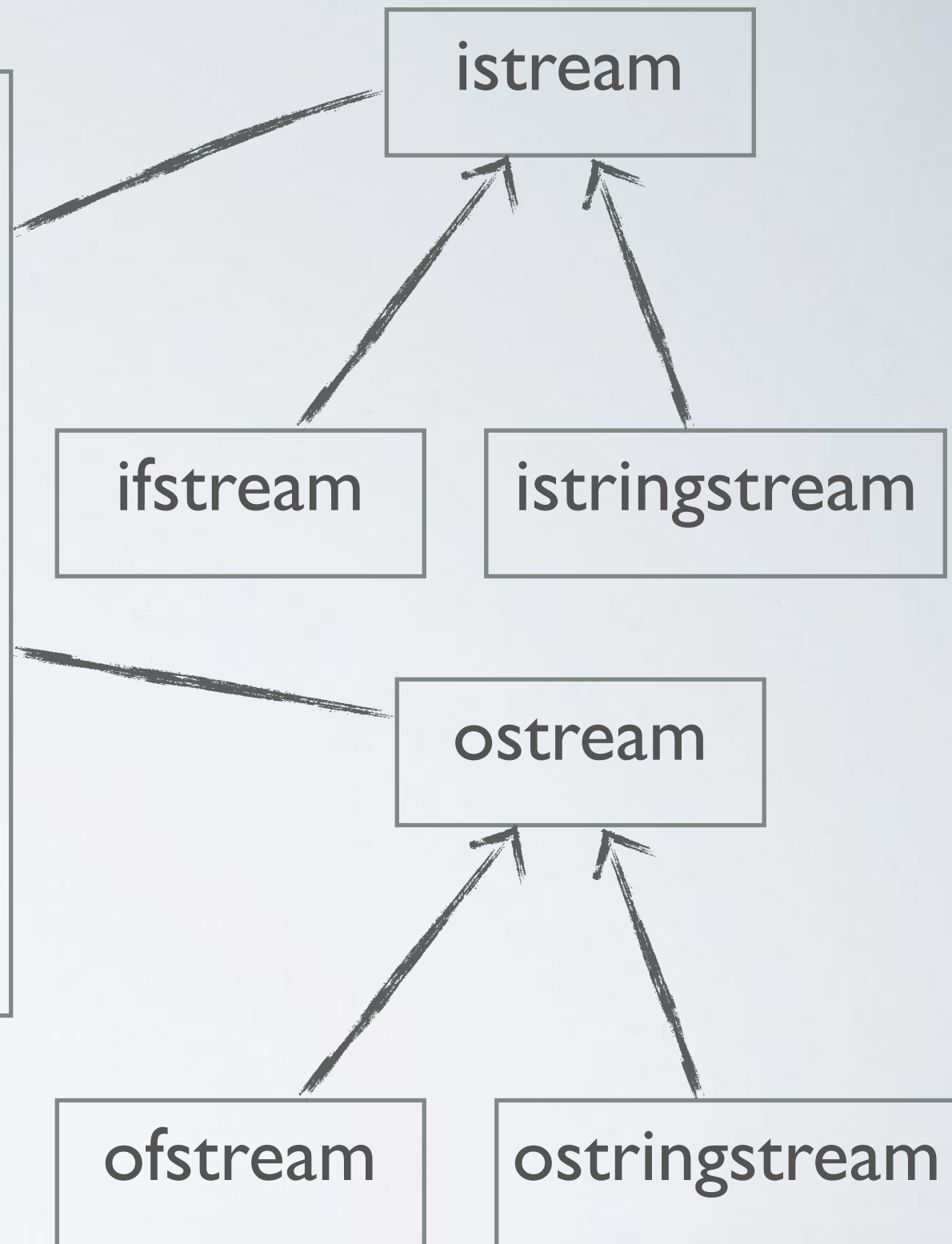
- Открытие файла на чтение.
- Открытие файла на запись.
- Построчное чтение файла.
- Форматирование (в зависимости от параметров).
- Построчная запись в файл.

Список обязанностей TextFormatter

TextFormatter

width: int
algorithm: enum { Left, Right, Center }
in: istream &
out: ostream &

+ TextFormatter(width : int,
 algorithm: enum)
+ format(in: istream &, out: ostream &)



**Используем абстракции ввода/вывода:
istream и ostream**

1. Считываем строку. Разбиваем на слова.
2. Если строка непустая, то добавляем слова в массив ГОТОВЫХ СЛОВ.
3. Если строка пустая или конец файла, то в цикле:
 1. Из массива готовых слов выбираем первые N слов так, чтобы они по ширине помещались в строку с учётом пробелов, а N было максимальным:
 $(\text{суммарная длина слов} + \text{количество слов} - 1) \leq \text{ширина}$.
 2. Форматируем выбранные N слов нужным способом, выводим, удаляем из массива.
 3. Цикл повторяется до тех пор, пока массив готовых слов не опустеет.
4. Вывод пустой строки и переход к п. 1, если ещё не конец файла.

Примерный алгоритм форматирования текста

ParagraphFormatter

width: int

words: vector<string>

+ addLine(line: string)

+ getFormattedLine(): string

formatLine(): string

LeftJustifyFormatter

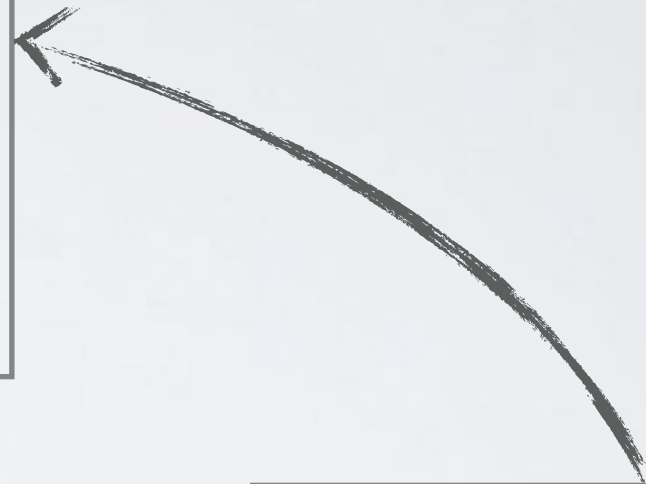
/# formatLine(): string

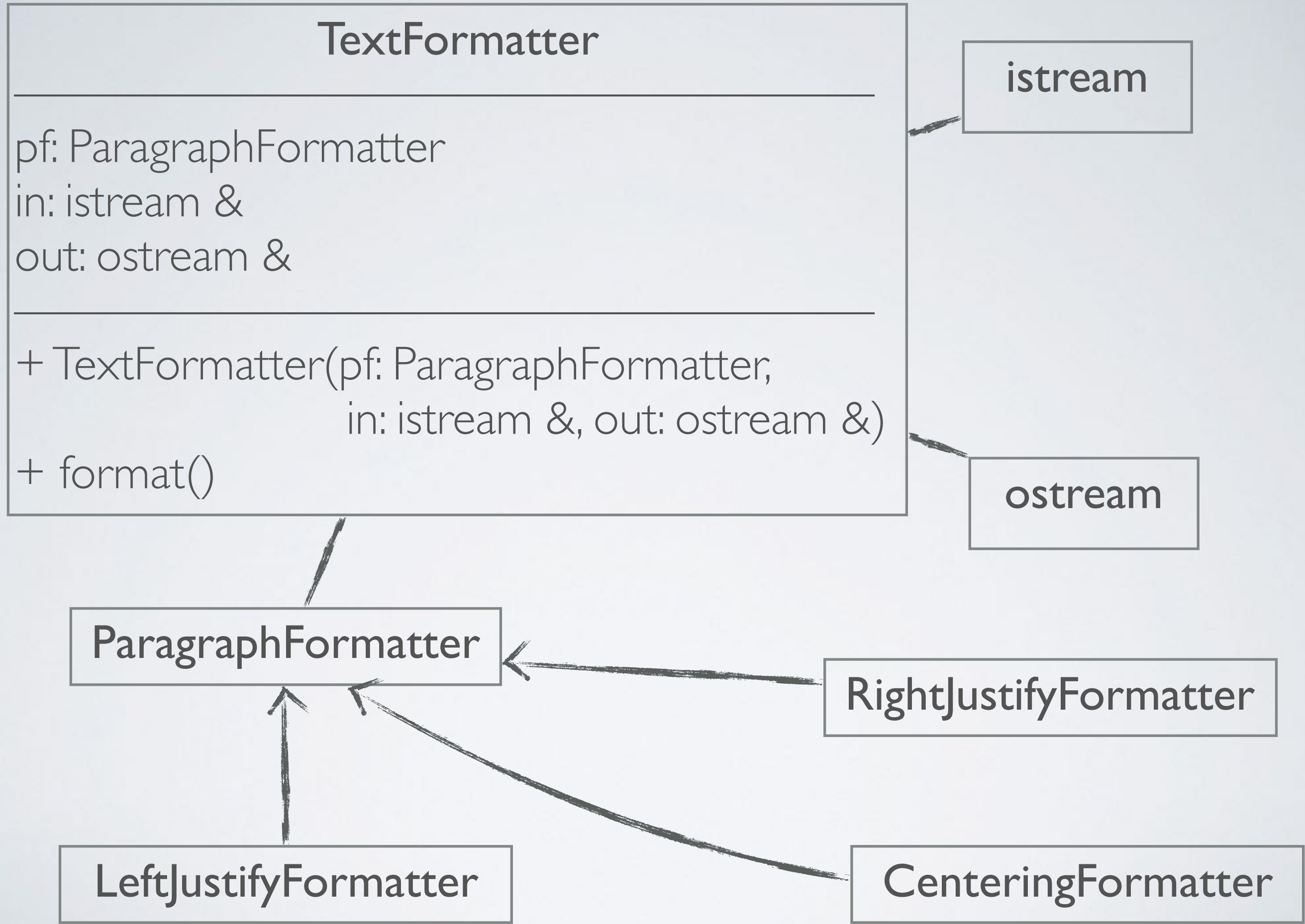
RightJustifyFormatter

/# formatLine(): string

CenteringFormatter

/# formatLine(): string





```

using namespace std;
static void drainFormatter(ParagraphFormatter *pf, ostream &os) {
    for (;;) {
        string line = pf->getFormattedLine();

        if (line.empty())
            break;

        os << line << endl;
    }
}

void TextFormatter::format() {
    string line;

    while (getline(in, line)) {
        if (line.empty()) {
            drainFormatter(pf, out);
            out << endl;
        } else
            pf->addLine(line);
    }
    drainFormatter(pf, out);
}

```

Примерный код TextFormatter::format()

ПРИНЦИП ПОДСТАНОВКИ ЛИСКОУ

- Liskov Substitution Principle (LSP).
- «Подтипы должны быть заменяемы их исходными типами».

- «Пусть $q(x)$ является свойством, верным относительно объектов x некоторого типа T . Тогда $q(y)$ также должно быть верным для объектов y типа S , где S является подтипом T ».
- «Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом».
- «Подкласс не должен требовать от вызывающего кода больше, чем базовый класс, и не должен предоставлять вызывающему коду меньше, чем базовый класс».

SQUARE И RECTANGLE

- Если класс **Rectangle** имеет интерфейс, предполагающий раздельность изменений ширины и высоты (**setWidth**, **setHeight**), то **Square** нельзя наследовать от **Rectangle**.
- Можно сделать, чтобы у класса **Square** и **setWidth**, и **setHeight** устанавливали как высоту, так и ширину, поддерживая «квадратность», но...

```
void testArea(Rectangle &r)
{
    r.setWidth(5);
    r.setHeight(4);

    assert(r.area() == 20);
}
```

Сломается, если передать **Square**

ЛИНИЯ И ОТРЕЗОК

Стоит ли наследовать класс **LineSegment** от класса **Line**?

(если **Line** моделирует бесконечную линию)

ПРИНЦИП ИНВЕРСИИ ЗАВИСИМОСТЕЙ

- **Dependency Inversion Principle (DIP).**
- Модули высокого уровня не должны зависеть от модулей низкого уровня. Оба типа модулей должны зависеть от абстракций.
- Абстракции не должны зависеть от подробностей, наоборот, подробности должны зависеть от абстракций.

Функция высокого уровня

```
void copy() {  
    int ch;
```

```
    while ((ch = getchar()) != EOF)  
        putchar(ch);
```

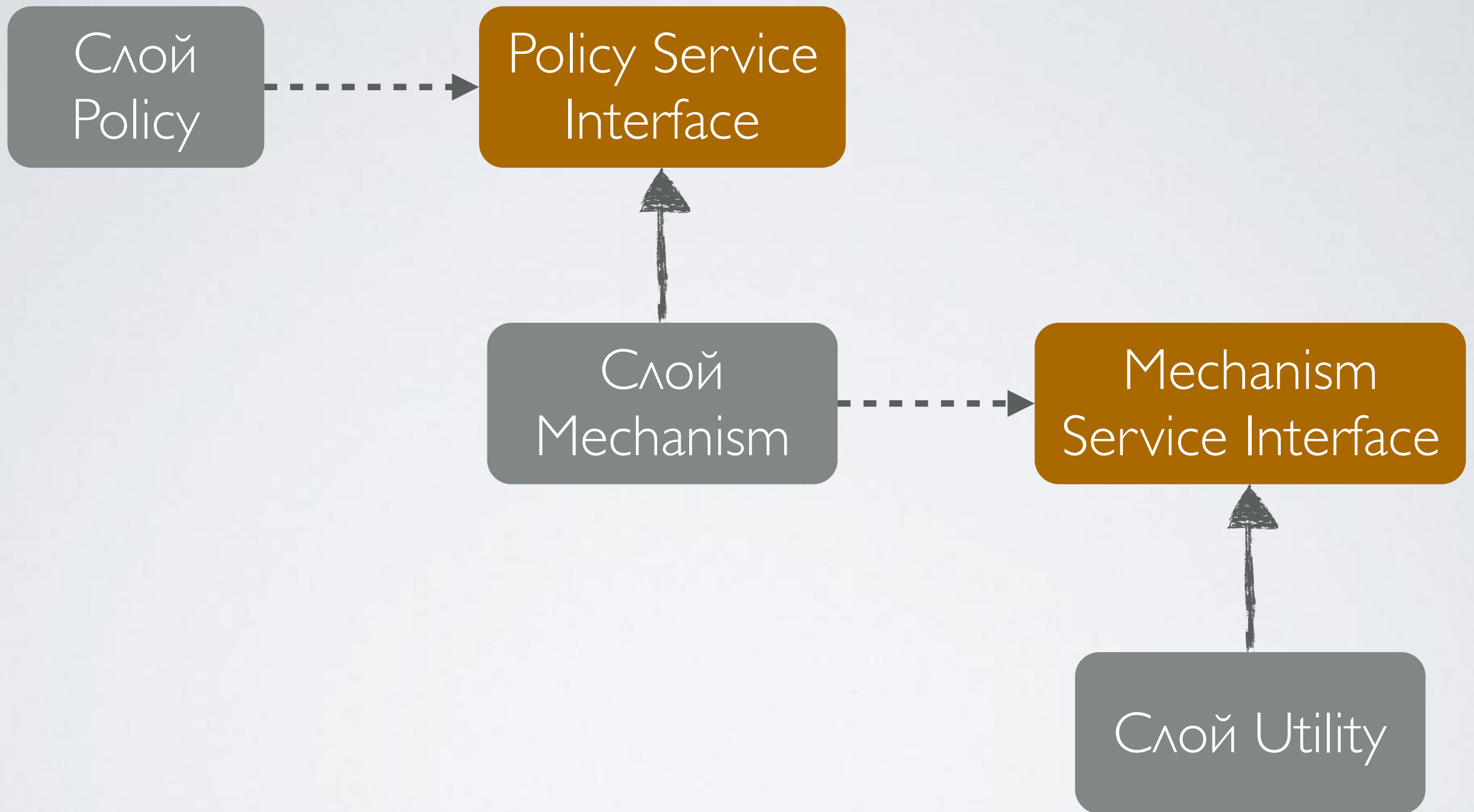
```
}
```

Функции низкого уровня

«СЛОИСТАЯ СТРУКТУРА»



Policy слишком сильно зависит от Utility :(

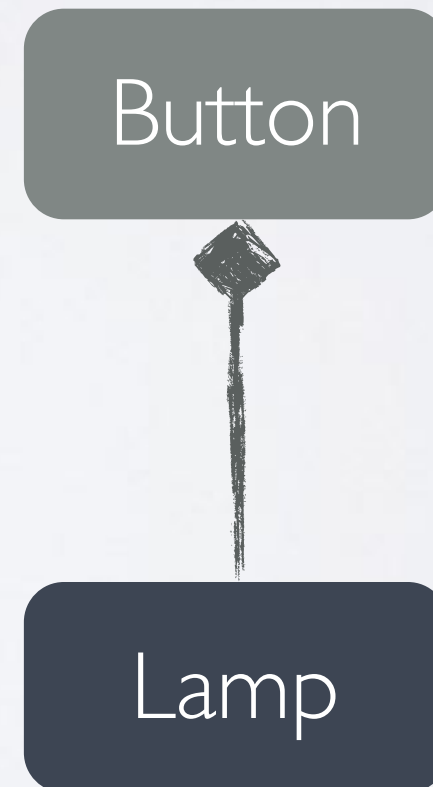


Инверсия зависимостей

Сильная зависимость.
Невозможно использовать Button
для управления чем-то еще

```
class Lamp {  
public:  
    void on();  
    void off();  
};
```

```
class ToggleButton {  
    Lamp lamp;  
    bool is_on;  
public:  
    ToggleButton() : is_on(false) {}  
  
    void toggle() {  
        is_on = !is_on;  
  
        if (is_on)  
            lamp.on();  
        else  
            lamp.off();  
    }  
};
```



```
class Switchable {
public:
    virtual void on() {}
    virtual void off() {}
};

class Lamp: public Switchable { /* ... */ };

class ToggleButton {
    Switchable *object;
    bool is_on;
public:
    ToggleButton(Switchable *o)
        : object(o), is_on(false) {}

    void toggle() {
        is_on = !is_on;

        if (is_on)
            object->on();
        else
            object->off();
    }
};
```



```
#define THERMOMETER 0x86
#define FURNACE 0x87
#define ENGAGE 1
#define DISENGAGE 0

void regulate(double minTemp, double maxTemp) {
    for (;;) {
        while (in(THERMOMETER) > minTemp)
            wait(1);

        out(FURNACE, ENGAGE);

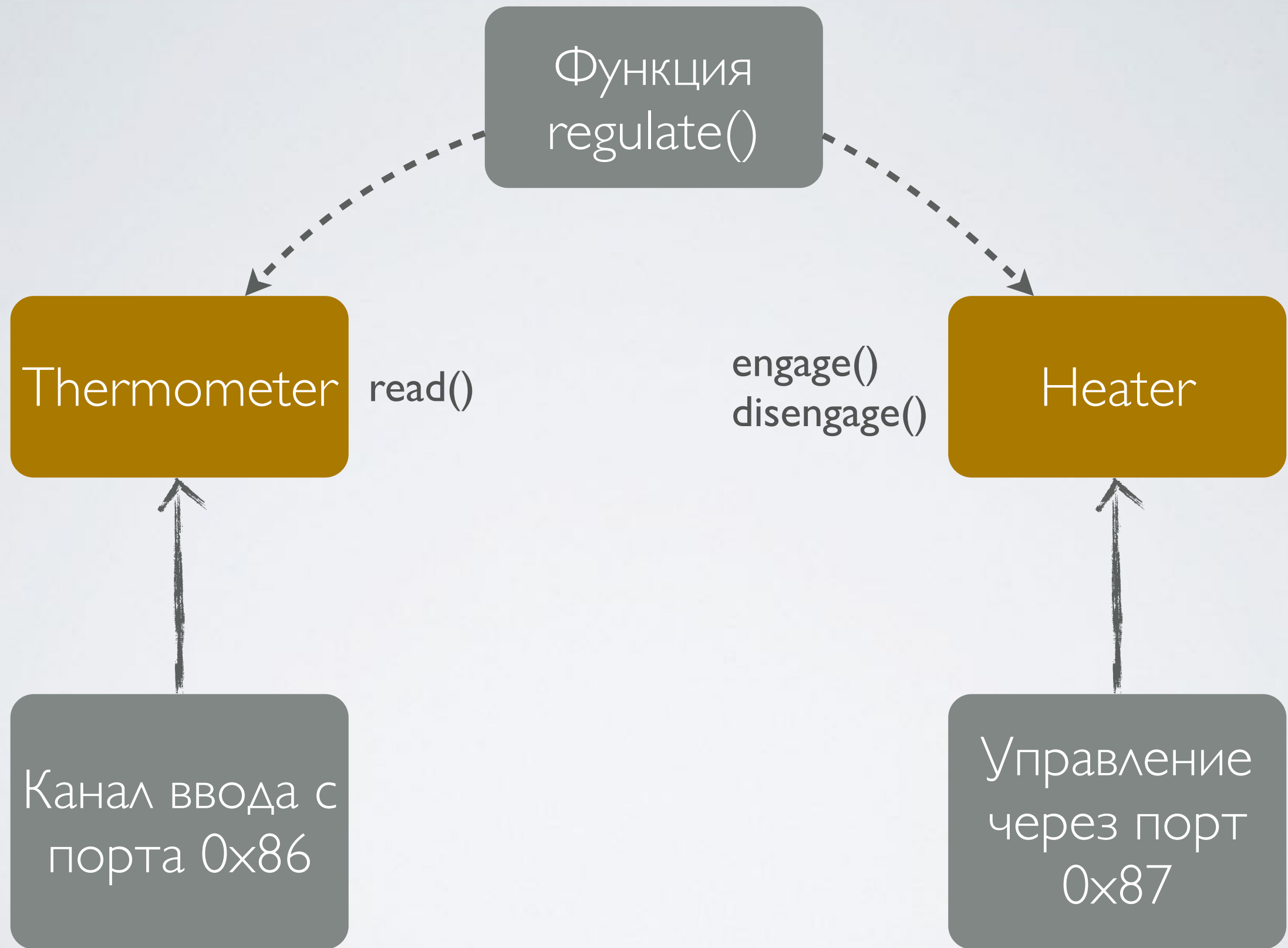
        while (in(THERMOMETER) < maxTemp)
            wait(1);

        out(FURNACE, DISENGAGE);
    }
}
```

«Поддержание температуры»

```
void regulate(Thermometer &t, Heater &h,  
             double minTemp, double maxTemp) {  
  
    for (;;) {  
        while (t.read() > minTemp)  
            wait(1);  
  
        h.engage();  
  
        while (t.read() < maxTemp)  
            wait(1);  
  
        h.disengage();  
    }  
}
```

**Применение принципа инверсии
зависимостей**



КОНЕЦ ДЕСЯТОЙ ЛЕКЦИИ

