

Архитектура в Agile:

~~компоненты и модули~~

слабая связность

Бибичев Андрей

март 2011



@bibigine

Андрей Бибичев

- E-mail: bibigone@gmail.com
- Twitter: [@bibigine](https://twitter.com/bibigine)
- Profile: <http://www.google.com/profiles/biBIGone>
- Slideshare: <http://www.slideshare.net/bibigine>



0. Проблематика

...

Alistair Cockburn:

*«starting with a walking skeleton,
then evolving it iteratively»*



Mary and Tom Poppendieck:

«divisible system architecture»



Robert Martin (Uncle Bob):

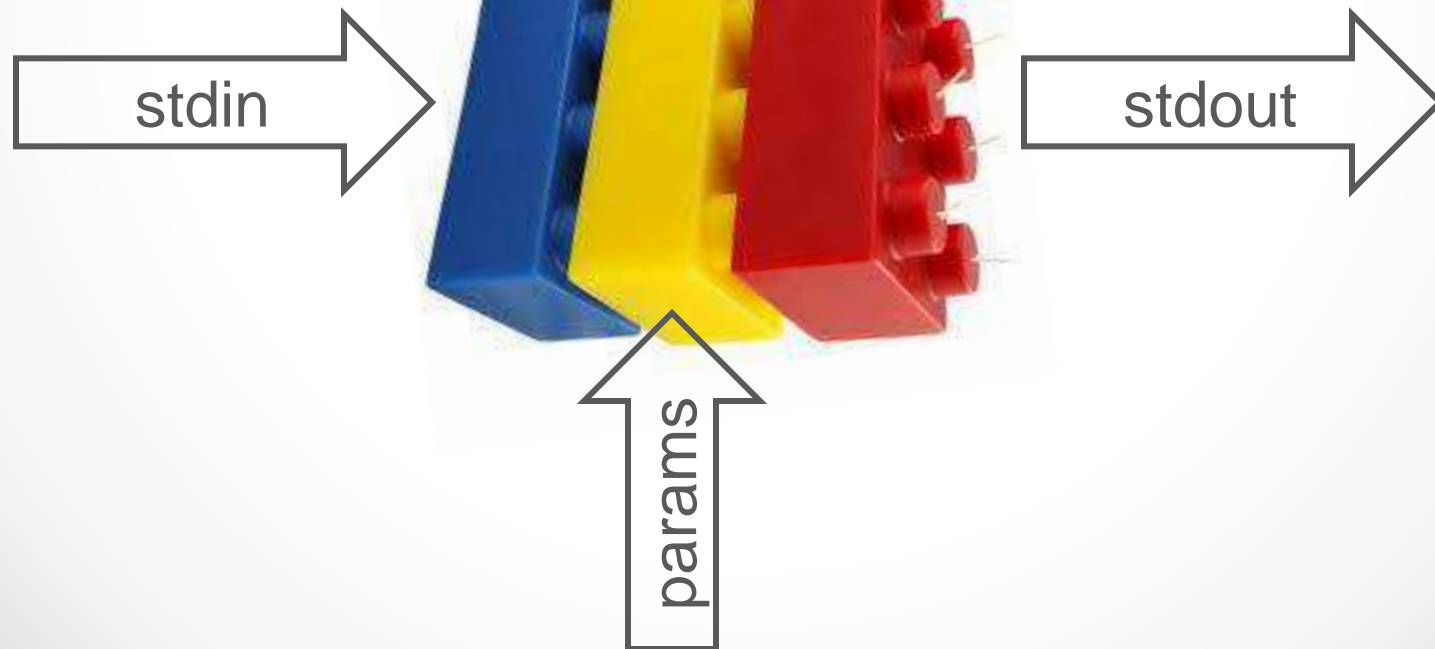
*«No doubt that BDUF is harmful.
Size matters! 'B' is bad, but 'L' is good.
Indeed, LDUF is absolutely essential.»*



Давняя мечта:



```
tr 'A-Z' 'a-z' <fnord.txt  
| tr -cs 'a-z' '\n' | sort | uniq  
| comm -23 - /usr/share/dict/words
```



ЭВОЛЮЦИЯ

1. goto
2. подпрограммы
3. динамические библиотеки
4. классы (ООП)
5. компоненты (COM/CORBA/...)
6. ...

Но что-то не легчает №1...

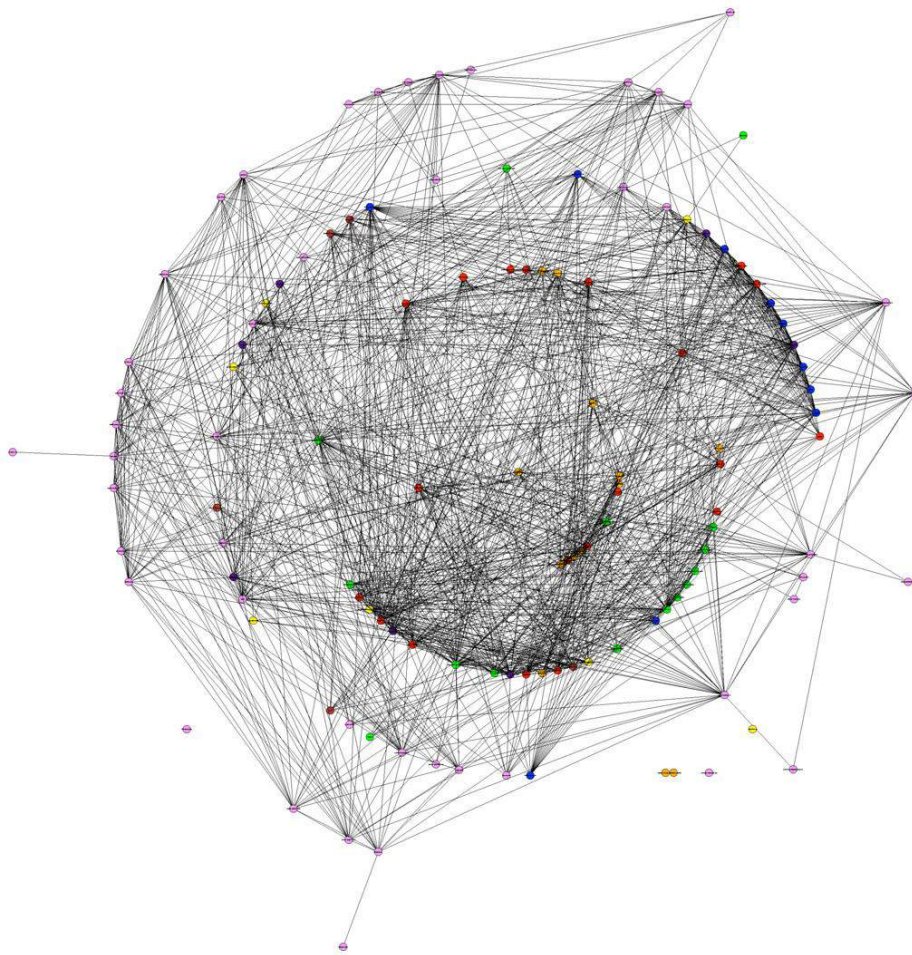


Корнелис Киммель « Снежный ком»



Сильная связность

≈ КАЖДЫЙ С КАЖДЫМ $\Rightarrow O(N^2)$

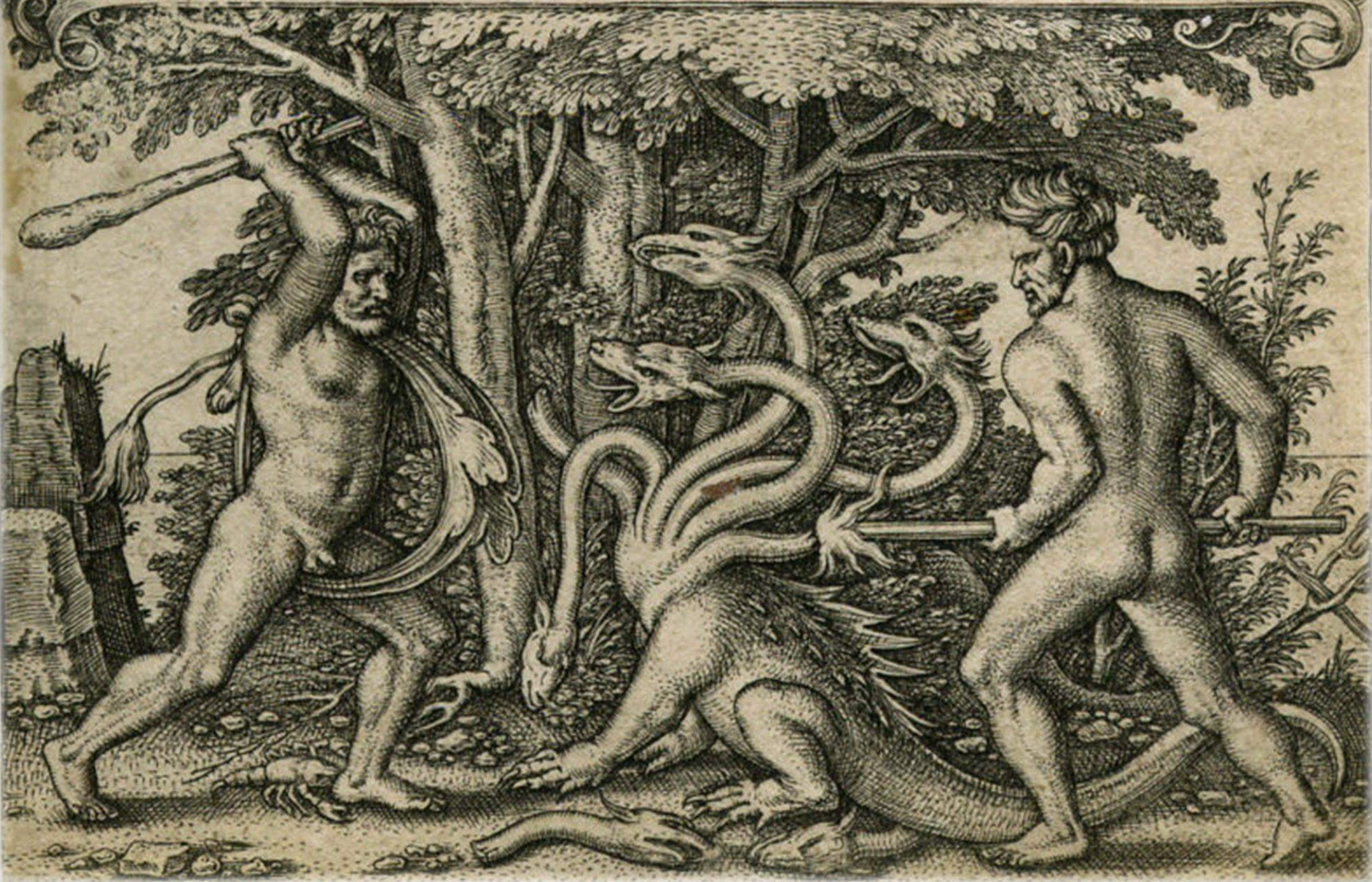


Но что-то не легчает №2...



Код-гидра:
один баг исправишь - несколько появляются

HERCVLES VNA CVM IO LAO HYDRAM OCCIDIT. JS ISB 48 D



Side-эффекты

Могучий глобальный контекст

+

Мутабельность всего и вся

+

Избыточная область видимости

=

поправил здесь, отвалилось там

Но что-то не легчает №3...



GOD-classes



WinForms.Net

Класс **Control**:

100+ СВОЙСТВ ($\approx 80\%$ public)

400+ МЕТОДОВ ($\approx 30\%$ public)

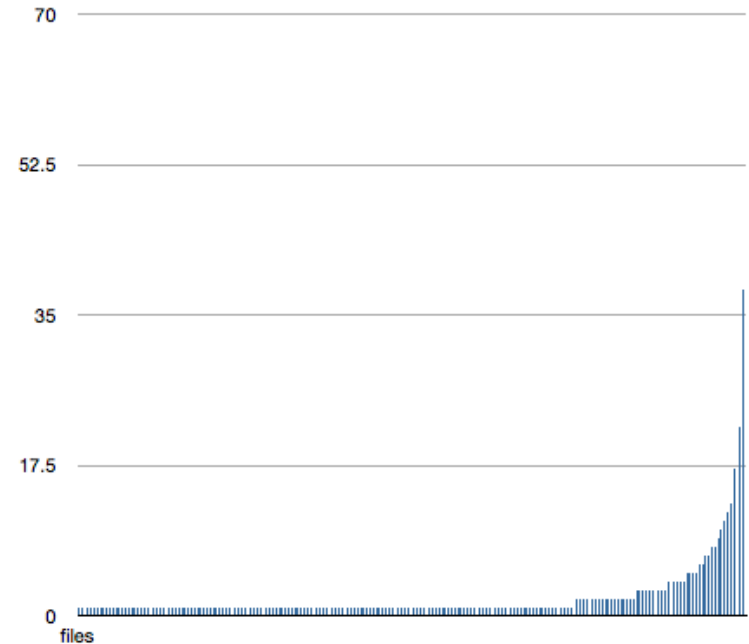
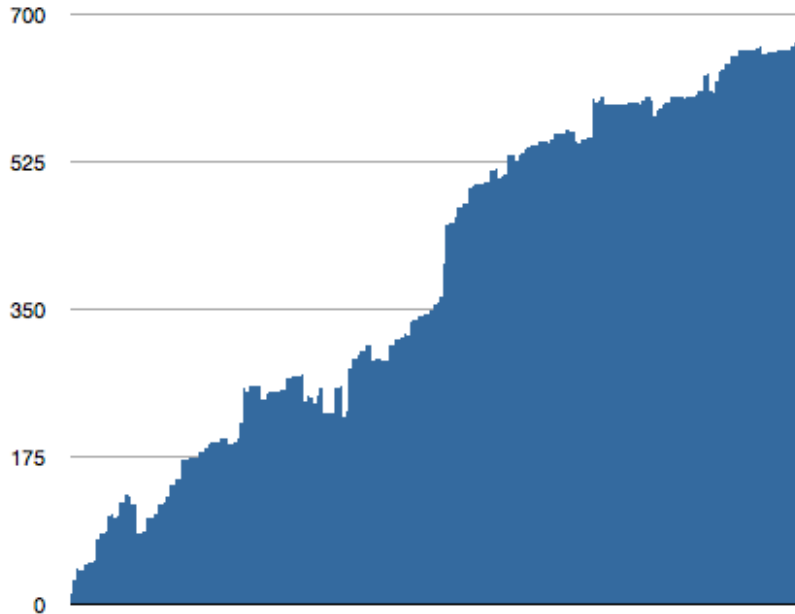
50+ СОБЫТИЙ (все public)

Разработчик и GOD-класс



А вокруг пустынно и уныло...

История сложности одного файла и распределение числа check-in по файлам



http://michaelfeathers.typepad.com/michael_feathers_blog/2011/03/data-rich-development.html

Но что-то не легчает №4...



«Рулонные функции»

«ПРОМЫШЛЕННЫЙ ИГРОКОД»

<http://code.google.com/p/lugaru/source/browse/Source/GameTick.cpp?r=cmake#1673>

```
1673 void Game::Tick()
1674 {
1675     static int i,k,j,l,m;
1676     static XYZ facing,flatfacing,absflatfacing;
1677     static XYZ rotatetarget;
1678     static bool oldkey;
1679     static float oldtargetrotation;
1680     static int target, numgood;
1681     static XYZ tempcoords1,tempcoords2;
1682     static XYZ test;
1683     static XYZ test2;
1684     static XYZ lowpoint,lowpointtarget,lowpoint2,lowpointtarget2,lowpoint3,
1685     static int whichhit;
1686     static bool donesomething;
1687     static bool oldjumpkeydown;
```

• • •

```
10985     }
10986 }
10987 if(IsKeyDown(theKeyMap, MAC_F1_KEY)&&!freezetogglekeydown){
10988     Screenshot();
10989     freezetogglekeydown=1;
10990 }
10991 }
10992 }
```

10 992 – 1673 + 1 = 9 320 строк

LUGARU HD

ENTER

OPTIONS

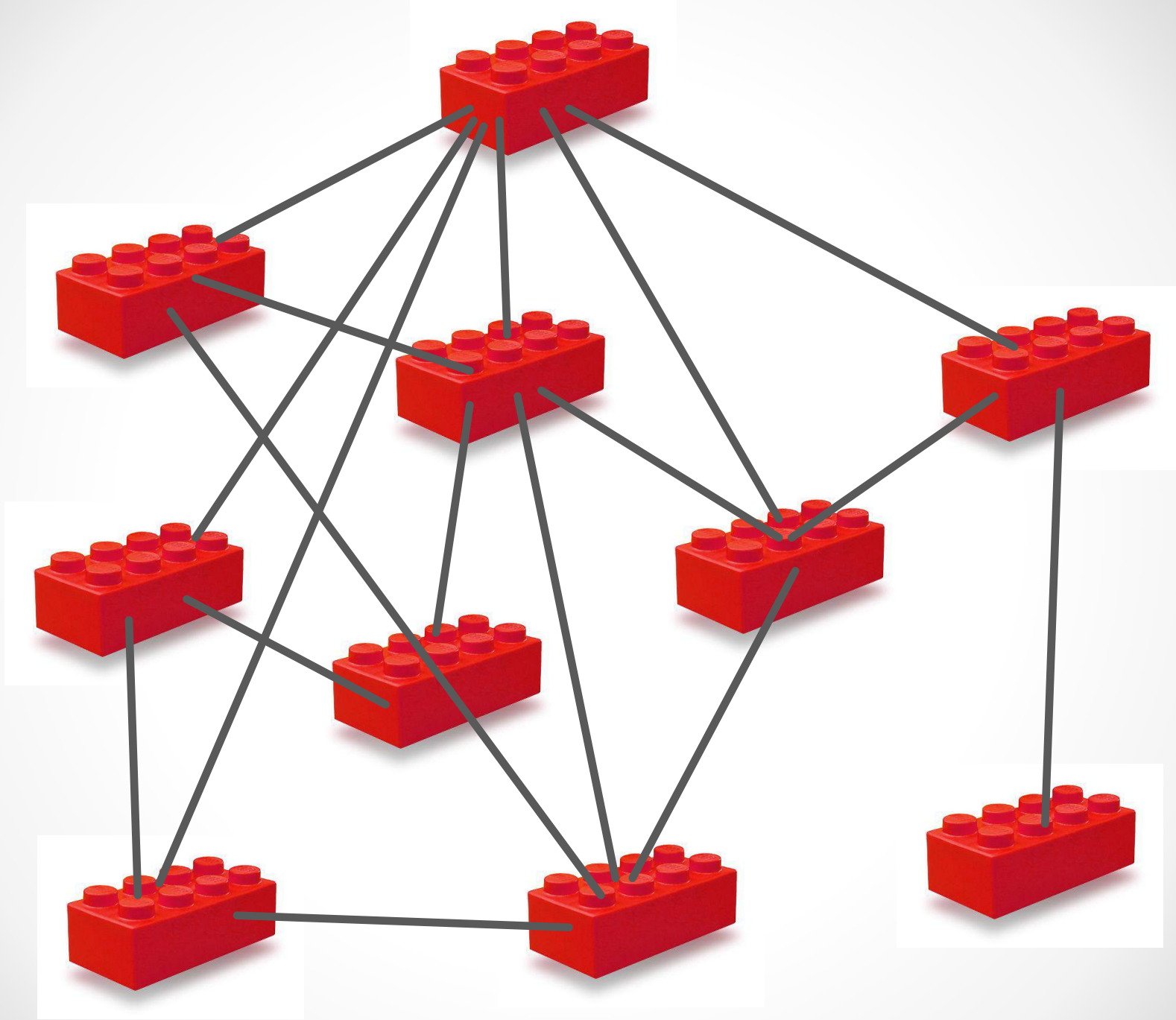
QUIT

SCORE: 152



Типичные болезни:

- Сильная связность
- Неуловимые и непознаваемые side-эффекты
- Вездесущий и всеобъемлющий глобальный КОНТЕКСТ (привет любителям singleton-ов!)
- GOD-классы
- Переусложненный и запутанный workflow
- Дублирование логики
- ...



Мечтали о:



А выходит так:



1. Классы и их взаимодействие

...

1. Классы и их взаимодействие ...

1.0. Ремарка

Для процедурных языков

- Класс => Структура + функции для неё

```
struct a {
    int v;
    double d;
};

double f(struct a* p, double k) {
    return p->v * p->d / k;
}
```

- Интерфейс => Структура указателей на функции (вспомним как работают вызовы к DLL!)

```
struct i {
    f1_t f1;
    f2_t f2;
};

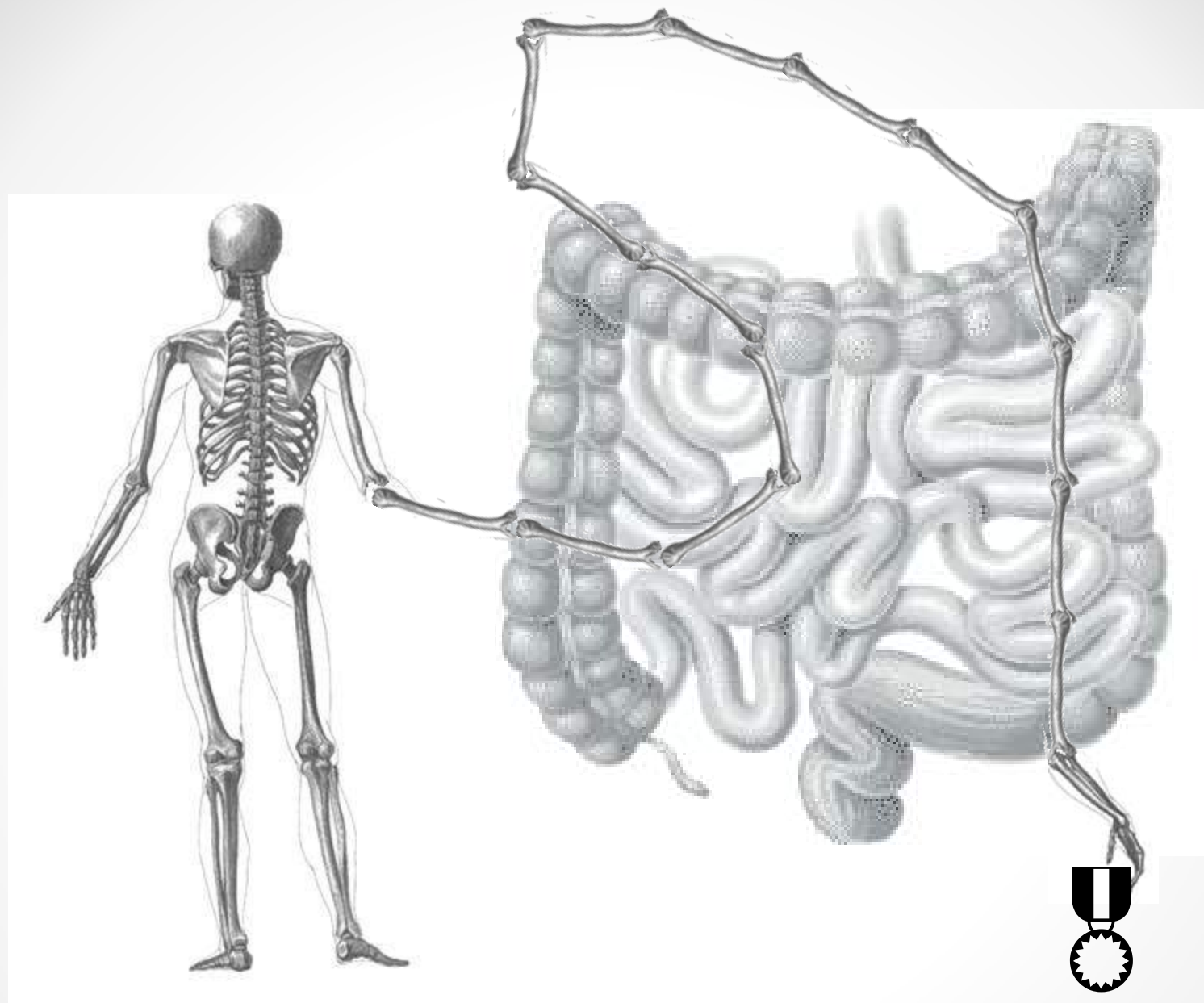
typedef void (*f1_t)(int s);
typedef double (*f2_t)();
```

- Наследование => Композиция

```
struct b {
    struct a super;
};
```

1. Классы и их взаимодействие ...

1.1. Современное ООП

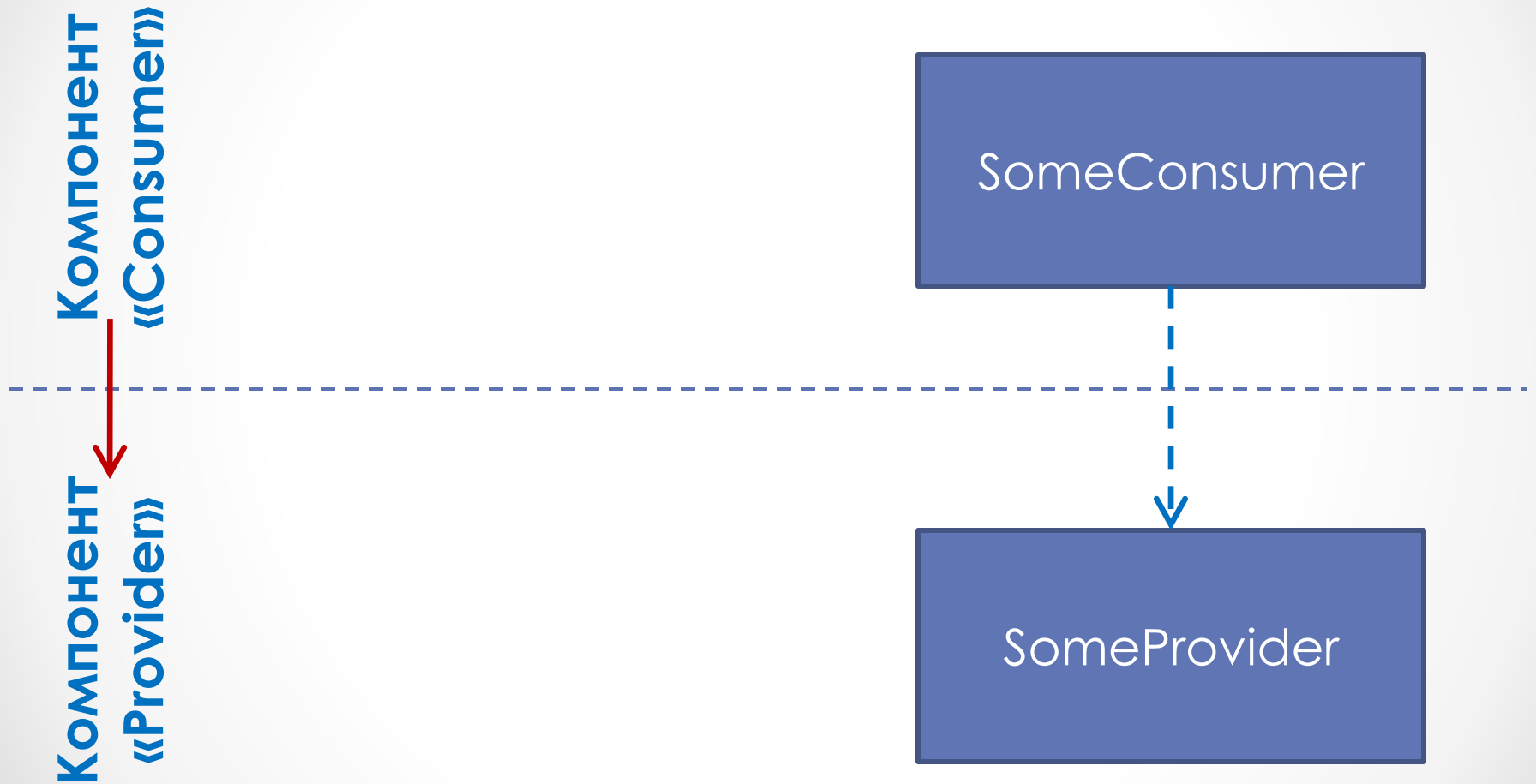


Длинные «руки» объекта:
если ему чего нужно, то он сам за этим тянется

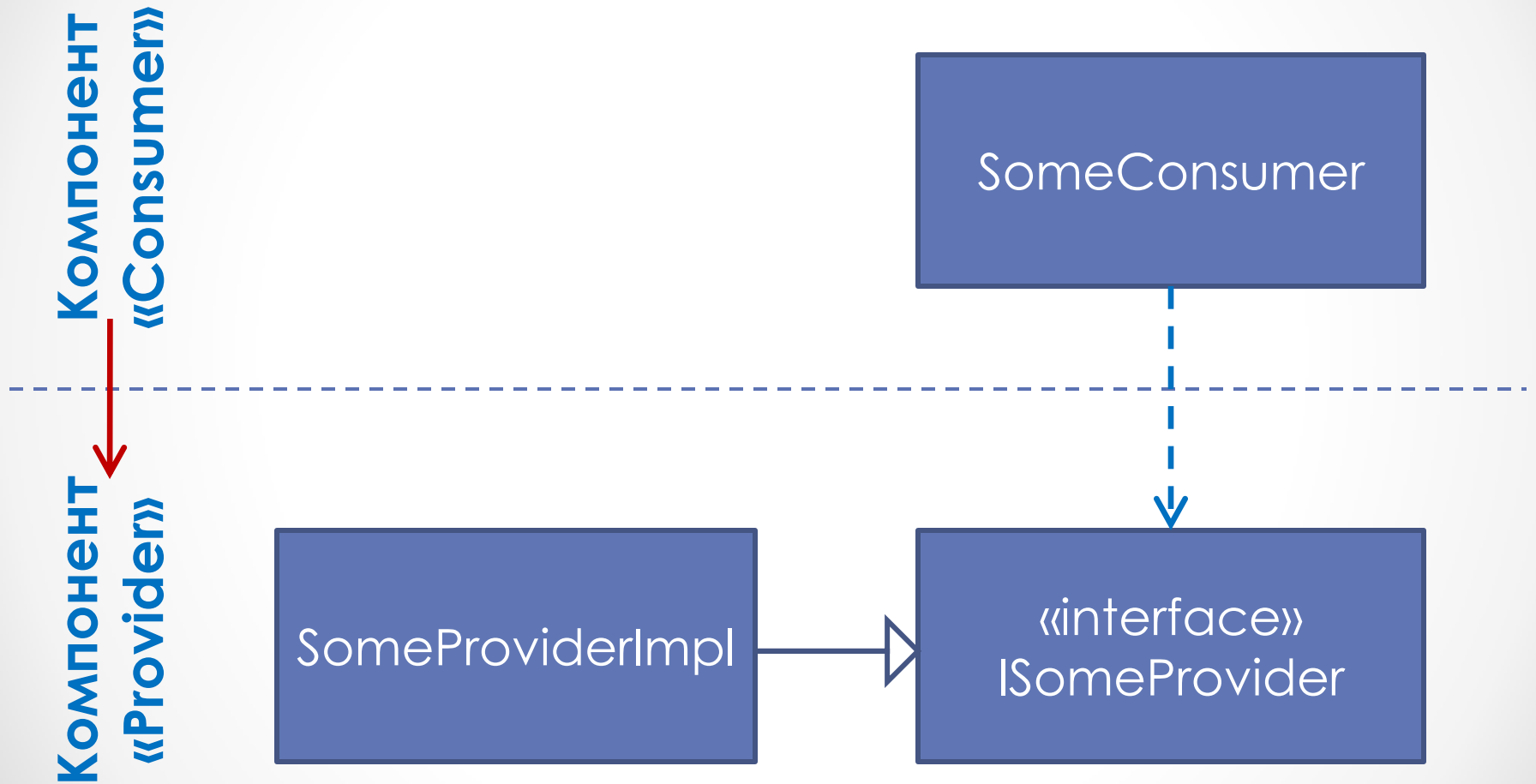
Укорачиваем руки:

- Inversion-of-Control (IoC)
- Law of Demeter
- Tell, Don't Ask

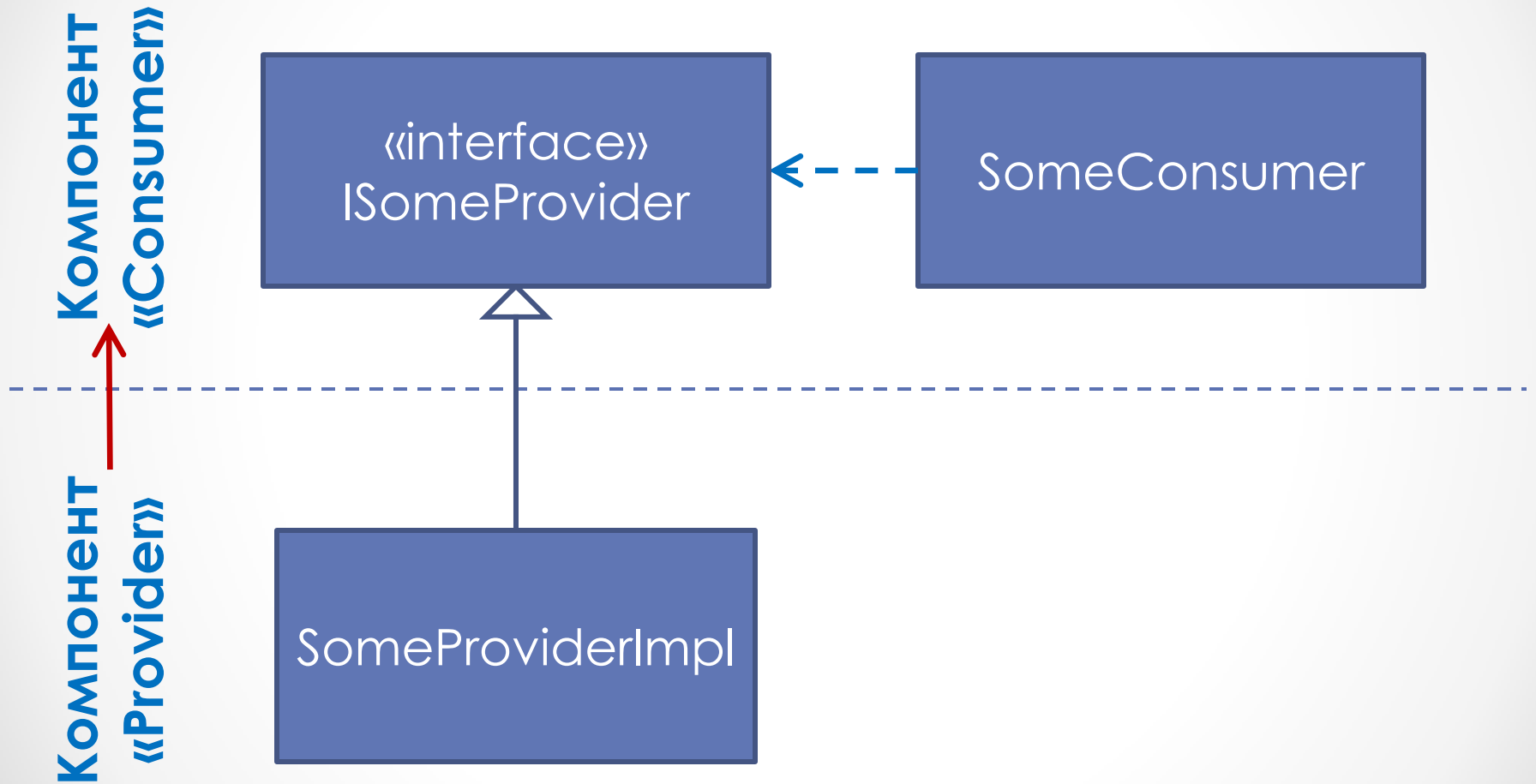
Inversion of Control (IoC)

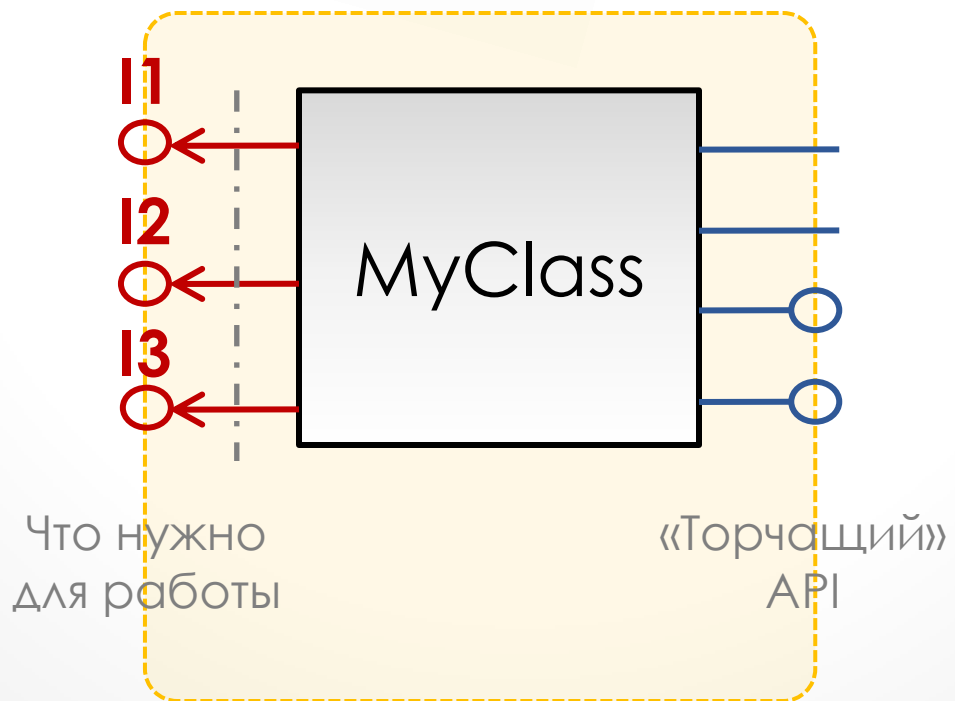
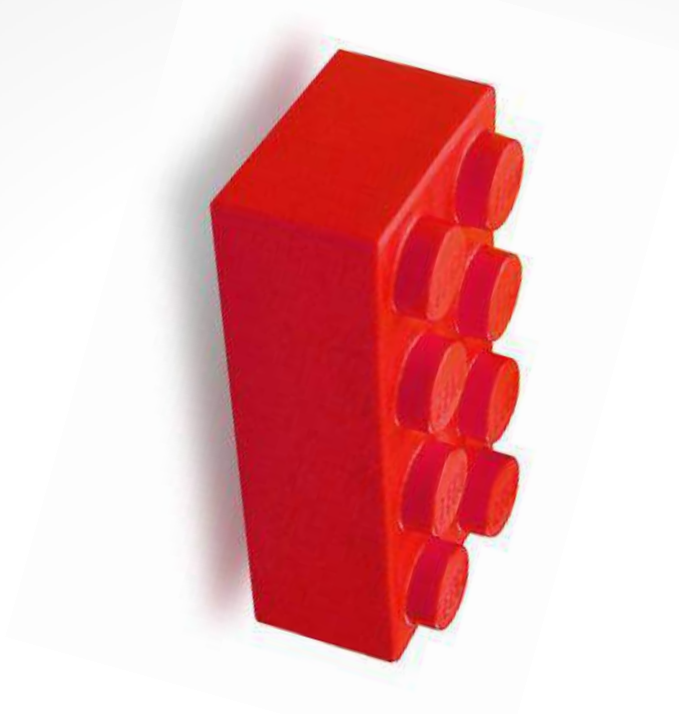


Inversion of Control (IoC)



Inversion of Control (IoC)





Как передать реализации интерфейсов объекту класса?

- Dependency Injection **через конструктор**

```
MyClass {  
    MyClass(I1 o1, I2 o2) { ... }  
}
```



- DI **через свойство/set-метод**

```
MyClass {  
    setI1(I1 o1) { ... }  
}
```



- **Service Locator**

```
o1 = MegaContext.resolve<I1>();
```



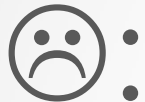
Law of Demeter

Any method of an object should only call methods belonging to:

- *itself.*
- *any parameters that were passed in to the method.*
- *any objects it created.*
- *any composite objects.*



Следствия



- Global context (включая Service Locator)
`MegaContext.Singleton.get<IService>().do()`
- ЦЕПОЧКИ ВЫЗОВОВ
`field.getSmth().getSmthElse().Prop.Count`



- True singleton (e.g. Null Object)

А работа с фабрикой?

```
void doSmtth(ISomeFactory someFactory) {  
    ISome some = someFactory.createSome();  
    some.act();
```

```
// someFactory.createSome().act();
```

- Кажалось бы, работа с фабрикой нарушает принцип Диметры
- Относитесь к фабрике как к переопределению оператора **new**. Тогда всё становится нормально: «*any objects it created*»

Tell, Don't Ask

Procedural code gets information then makes decisions. Object-oriented code tells objects to do things.

— Alec Sharp

```
void myFunc(Obj obj) {
    if (!obj.init()) throw new ...;
    String id = obj.getId();
    log.write("starting transaction with " + id);
    if (obj.supportTransaction()) {
        if (!obj.startTransaction()) throw new ...;
        Photo photo = obj.GetPhoto();
        if (!analyzePhoto(photo)) {
            obj.rollback();
            obj.kill();
        } else {
            obj.act();
        }
    }
}
```

me: Привет!

obj: Хай!

/ me: Ага, ответила, значит продолжаем разговор */*

me: Как тебя зовут?

obj: Саша

me: Саша – красивое имя!

/ me: Блин, мужик или баба?! */*

me: А полное имя Александр или Александра?

obj: Александра

/ me: Ура! Баба! */*

me: Здорово! Давай куда-нить сходим?

obj: Давай. Только куда?

me: В клуб. Только как я тебя узнаю? Пришли фотку

obj: Держи

/ me: Фу */*

me: == disconnected ==

Минусы «чата»:

- Сложный интерфейс
- Тесная связность => С высокой вероятностью придется согласованно править в двух местах
- Попахивает нарушением инкапсуляции

Инкапсулировать в другом месте

```
interface IObj {  
    void performInTransaction(Callback action);  
}
```

Где риаализовывать:

- Либо в классе, с которым мы общаемся
- Либо сделать специализированный wrapper

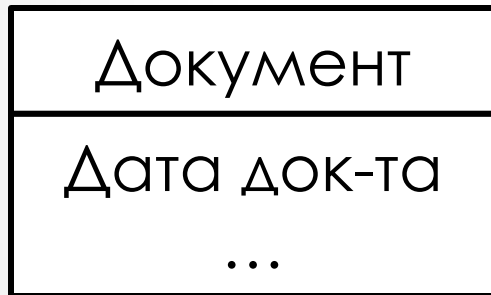
Декларативный стиль в императивных языках

- Иди на кухню
- Возьми чашку
- Налей туда заварки
- Добавь кипятку
- Две ложки сахара
- Поставь на блюдце
- Принеси мне



- Мне бы чаю

Дуализм: интерфейс vs. данные



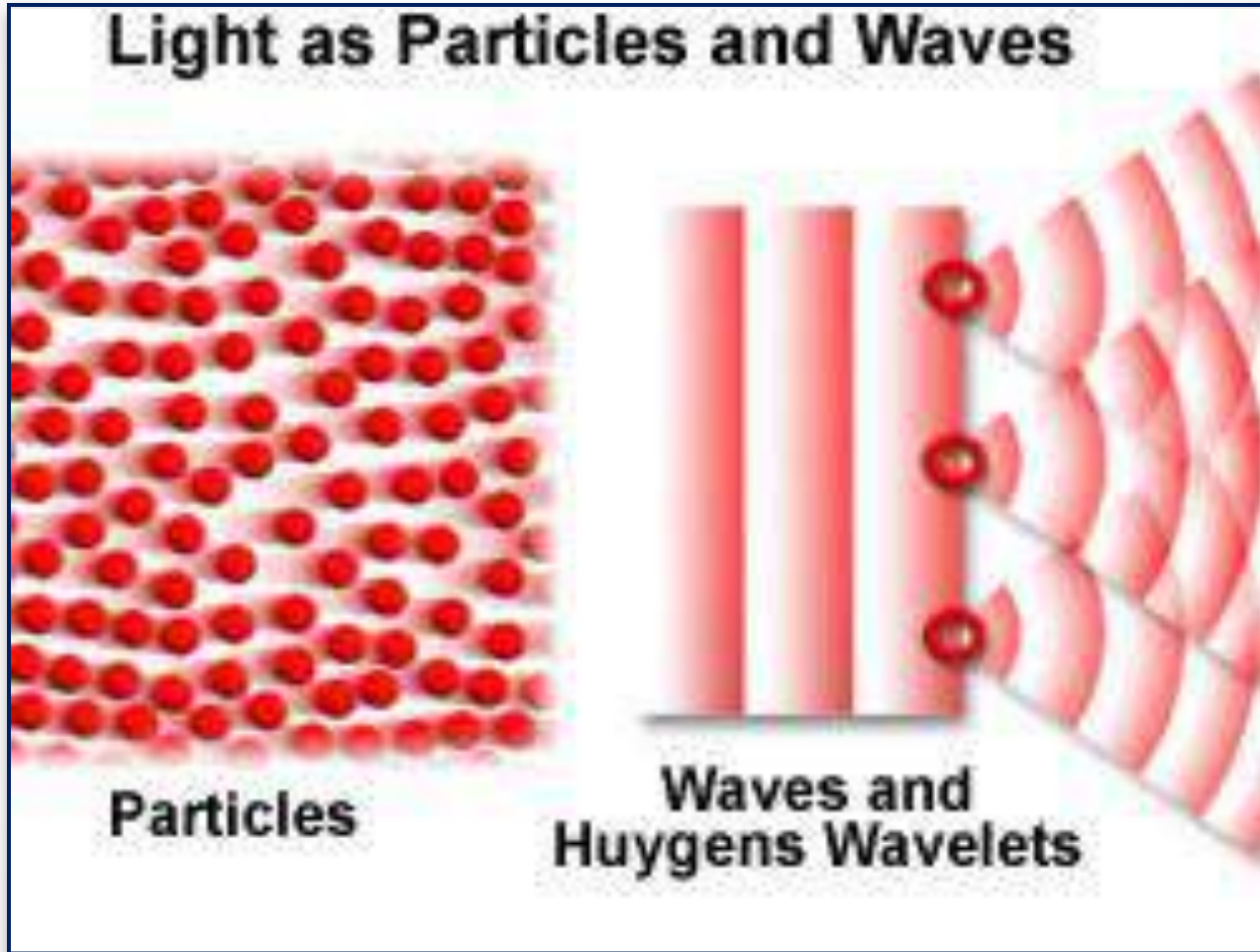
```
class MyDoc {  
    MyDoc(ICurrentDateProvider cdp) {  
        docDate = cdp.get();  
        ...  
    }  
}
```

```
class MyDoc {  
    MyDoc(Date date) {  
        docDate = date;  
        ...  
    }  
}
```

Требования к такому «транспортному» классу:

- Как и интерфейс должен «жить и мыслиться» вместе с тем классом, который его использует
- Минималистичный (только те данные, которые нужны)
- **Immutable**
- Должны легко конструироваться в любом состоянии (для тестирования – a la Mock в случае интерфейсов)
- Сериализуемые (для транспорта)

Это как корпускулярно- волновой дуализм



Иногда удобнее так, иногда сяк.
Полезно владеть и тем и тем.

Лучше сильно не увлекаться:

- Anemic Model

<http://martinfowler.com/bliki/AnemicDomainModel.html>

1. Классы и их взаимодействие ...

1.2. MVP и Entity-Repository

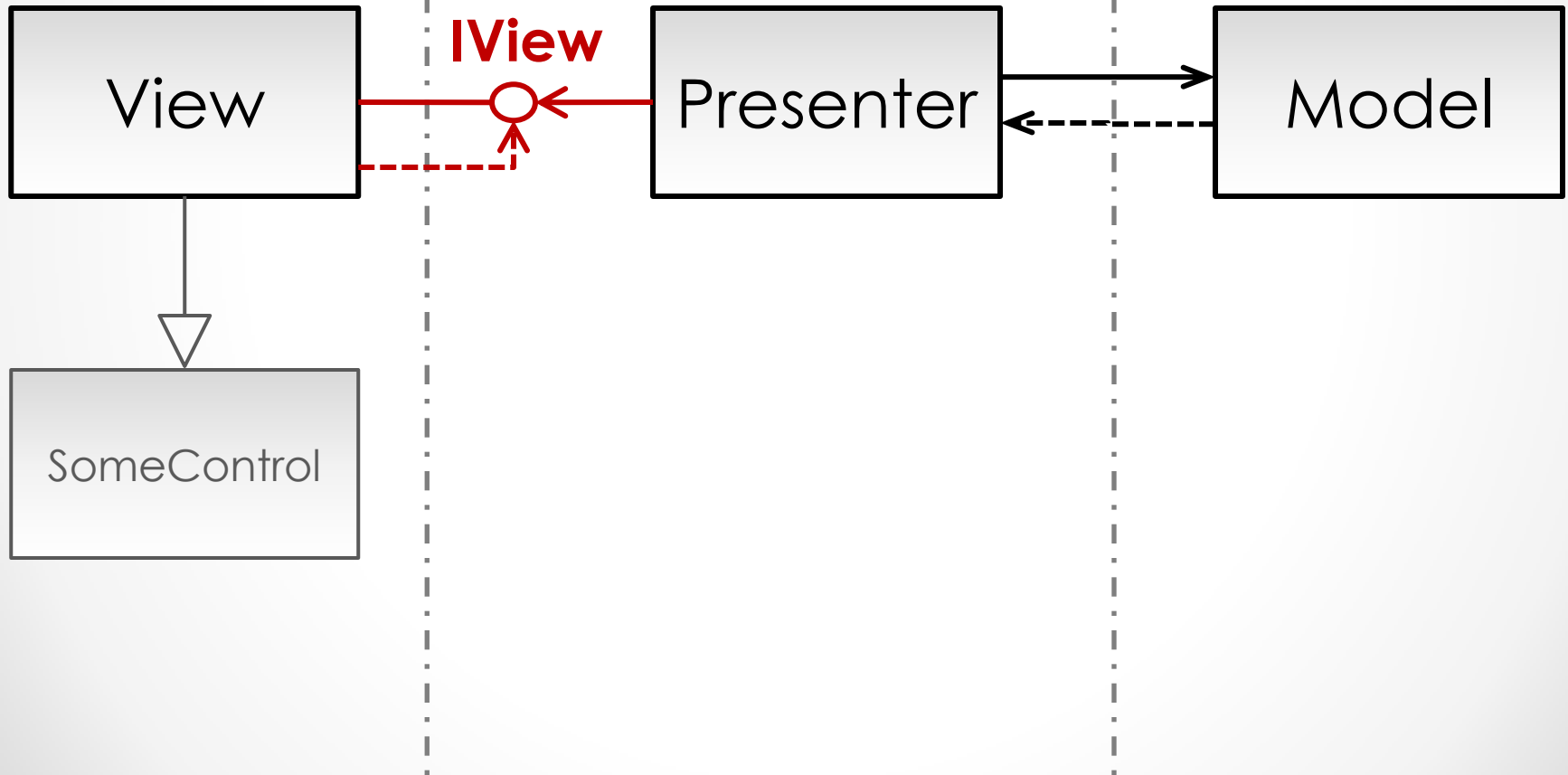
Классика жанра

- Model-View-Presenter (MVP)
- Entity-Repository

Presentation

Appl. Logic

Domain



10:00 **Андрей Совцов**
10:15 Все в ваших руках: быть готовым к изменениям в

10:30 **Дмитрий Никонов**
10:45 Планирование релизов в методологиях быстрой
11:00 разработки (Agile)
11:15
11:30 Казалось бы структура релизов в командах

11:45 Кофе-брейк

12:00 **Станислав Калканов**
12:15 В чем счастье заказчика? Готовые фичи вместо
12:30 гант чарта!

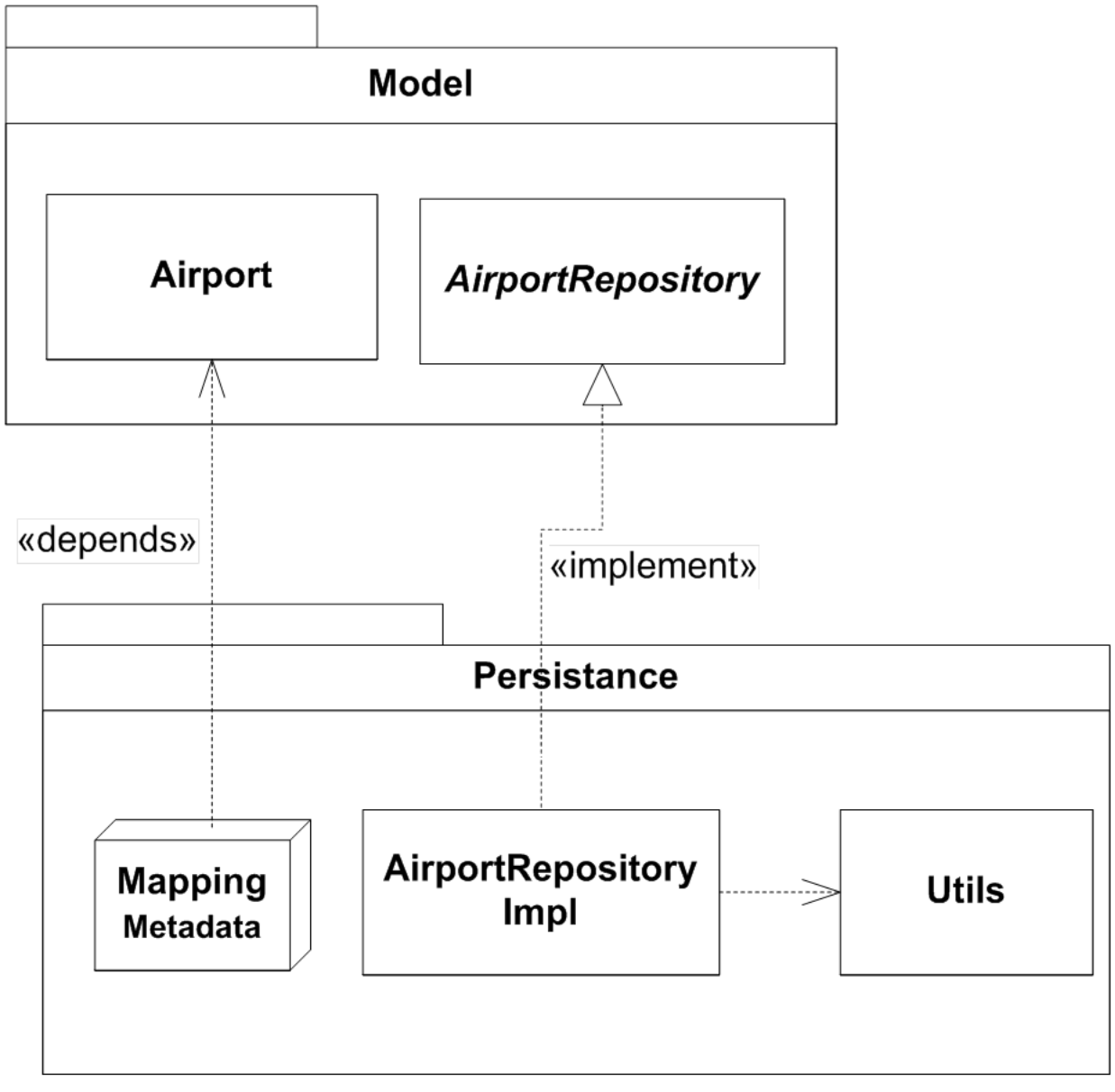
12:45 **Сергей Евтушенко**
13:00 Agile in Investment Banking
13:15 От ресурса к продукту: применение agile-подхода

Андрей Бибичев
Архитектура в Agile: переосмысляя идею
модульности и компонентности

Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы

Антон Бевзюк
Архитектура для автоматизированного
тестирования UI

Николай Гребнев
Domain Driven Design в условиях разработки
распределенных приложений



10:00	Андрей Совцов	Андрей Бибичев
10:15	<u>Все в ваших руках: быть готовым к изменениям в</u>	<u>Архитектура в Agile: переосмысляя идею</u>
10:30	Дмитрий Никонов	<u>модульности и компонентности</u>
10:45	<u>Планирование релизов в методологиях быстрой</u>	Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы
11:00	<u>разработки (Agile)</u>	
11:15	Казалось бы структура релизов в командах	
11:30	Кофе-брейк	
11:45		
12:00	Станислав Калканов	Антон Бевзюк
12:15	<u>В чем счастье заказчика? Готовые фичи вместо</u>	<u>Архитектура для автоматизированного</u>
12:30	<u>гант чарта!</u>	<u>тестирование UI</u>
12:45	Сергей Евтушенко	Николай Гребнев
13:00	<u>Agile in Investment Banking</u>	<u>Domain Driven Design в условиях разработки</u>
13:15	<u>От ресурса к результату: применение agile-подхода</u>	<u>распределенных приложений</u>

Как декомпозировать сложную форму?

The screenshot shows a web application titled "PACKAGE MANAGEMENT SYSTEM". The interface includes a menu bar (File, Edit, Package, Settings, Help), a toolbar (Reload, Update All, Install/Update, Remove), and a repository dropdown set to "12345.lorem ipsum dolores.234". A left sidebar lists categories, and a main table displays a list of packages with columns for Name, Installed Version, Latest Version, Rating, and Description. A detailed view for the "ADEPT-COMMON" package is shown below the table. A right-hand panel titled "Page Details" provides metadata and user scenario information. Annotations (1-4) highlight specific UI elements, and a callout (1.1) explains a column header change.

PACKAGE MANAGEMENT SYSTEM

File Edit Package Settings Help

Repository: 12345.lorem ipsum dolores.234

Reload Update All Install/Update Remove

All Categories

All

Amateur Radio (universe)

Base System

Base System (restricted)

Base System (universe)

Communication

Communication (multiverse)

Communication (universe)

Cross Platform

Cross Platform (multiverse)

Cross Platform (universe)

Lorem Ipsum

Scilicet Lapide Fractis

Nomine Quae Fractis

Lapide Eum

Pervenissent

Multiplices Antiochia

Levius et Disiungitur

Mandatum Pater

Constaret Exillium

Apollinares

Mevius Rum

Lorem Ipsum

Show: All Packages Search:

Name	Installed Version	Latest Version	Rating	Description
adduser	3.100		****	lorem ipsum do package...
adept	2.900	2.1.2lorem26.1	***	lorem ipsum do package...
b-package	2.1.1lorem26.1		*	lorem ipsum do package...
c-package	2.1.1lorem26.1		****	package manage...
d-package	2.1.1lorem26.1	2.1.2lorem26.1	**	lorem ipsum do package...
e-package	2.1.2lorem26.1		**	lorem ipsum do package...
f-package	2.1.2lorem26.1		****	lorem ipsum do package...
g-package	012.1-2	013.1-2	****	lorem ipsum do package...
h-package	012.1-2	013.1-2	*	lorem ipsum do package...

ADEPT-COMMON
package manager for KDE -- common files

General Info Installed Files Dependencies

Description: Package management suite for KDE. Sed quid est quod in hac causa maxime homines admirentur et reprehendant meum consilium, cum ego idem antea multa decreverim

Total number of packages in (-) repository 5113, 1052 installed, 0 broken. 0 selected.

Page Details

File name: 01_main_view
Project: Package Manager
Release: PH1
Created: 01.02.08
Modified: 02.22.08
Version: 6.3
Designer: Jenya Gestrin
Page Type: main page

User Scenario

Scenario 1: User opens Package Management Application.

Page Components

For interaction rules /behaviors for each component refer to the appropriate component spec.

1 Top menu. See [Top Menu Spec](#).

2 Repository view dropdown component: allows switching between repositories.

3 'Type in' Search: table results show as user types.

4 Packages table - main work area. See [02_selection](#) page for the details.

1.1 Column is now called 'Latest Version' (used to be Latest Available)

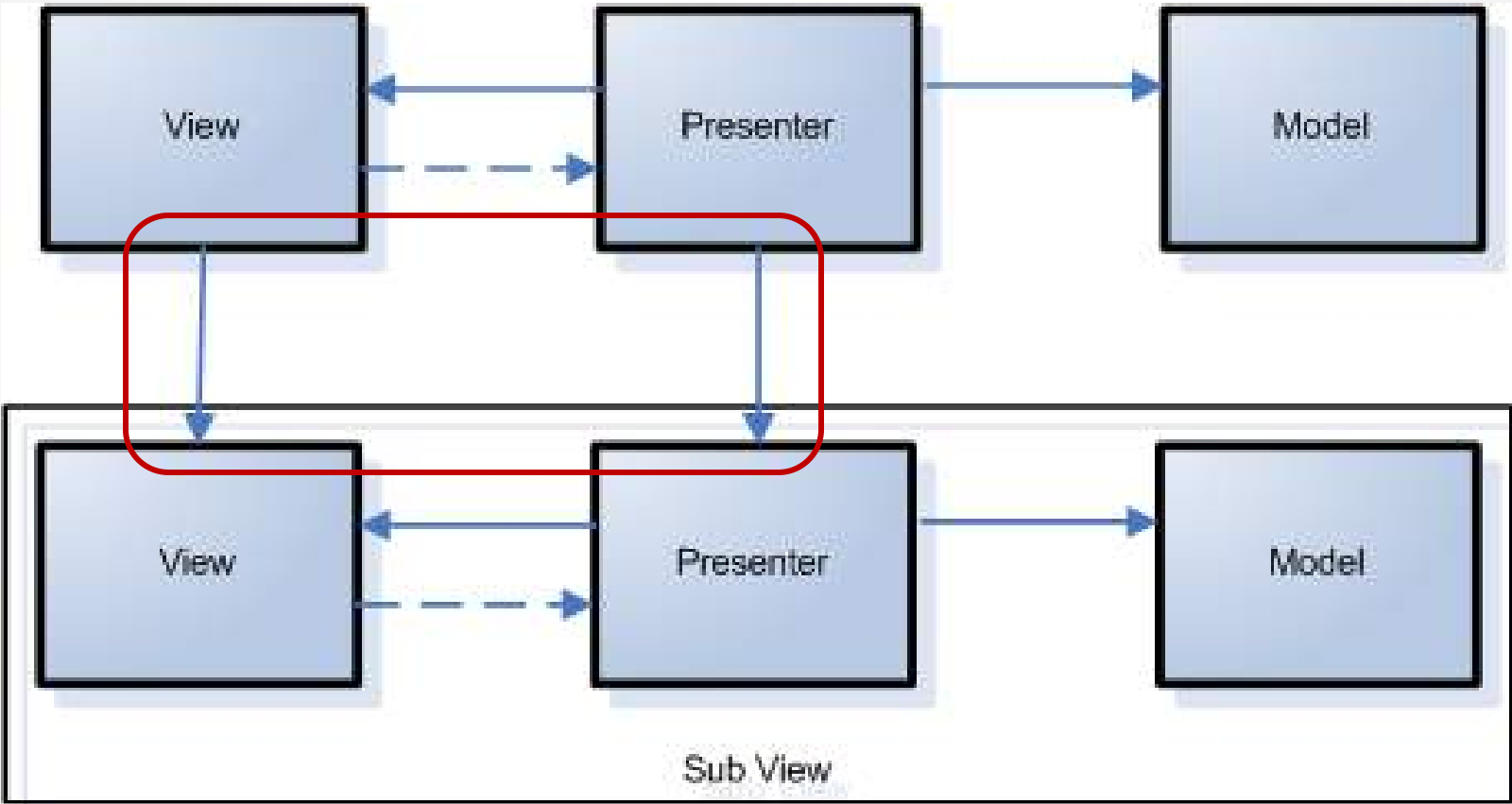
Alternate Scenarios

Page Revision Details

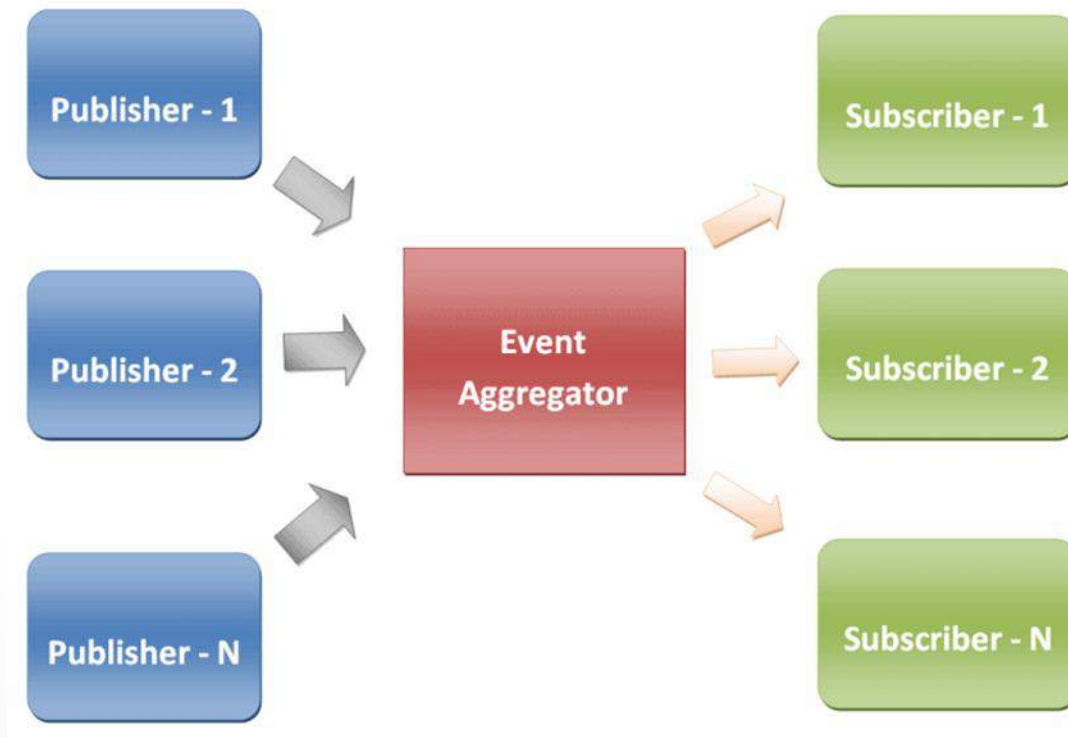
From the implementation point of view it will be time consuming task to implement initially proposed non standard

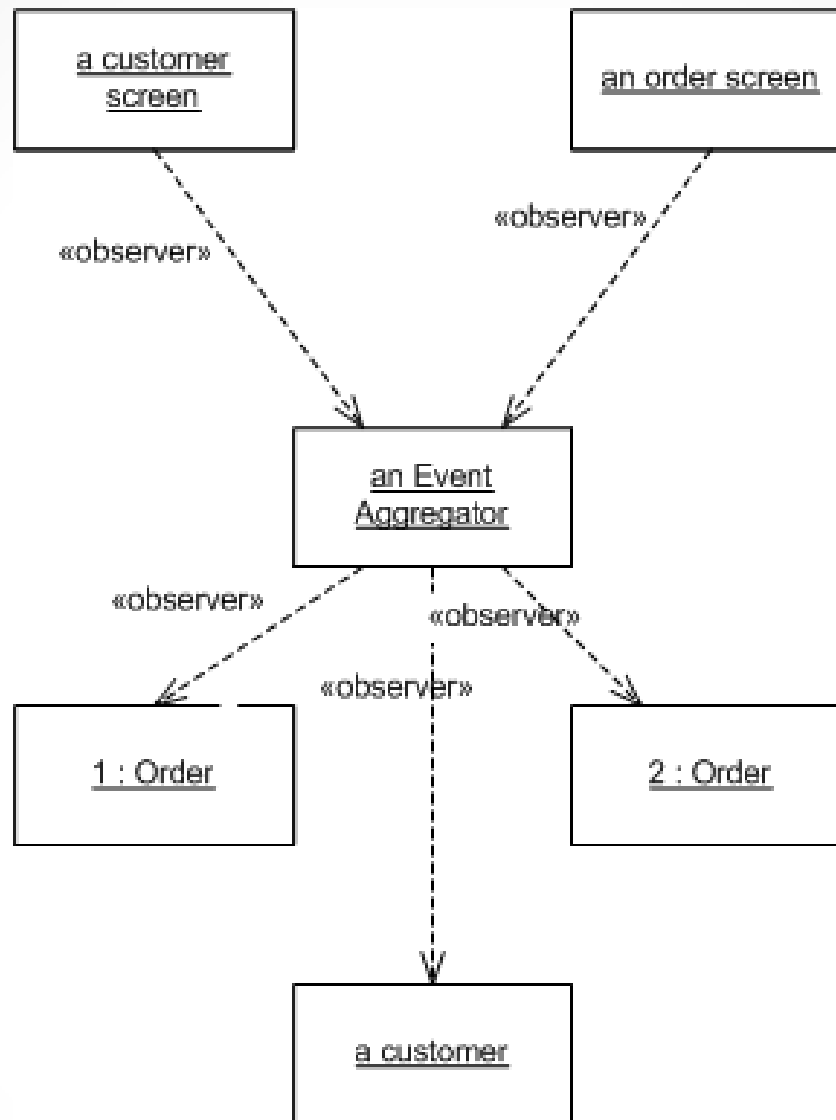
[click here to proceed to the next step of Install / Update use case.](#)

[back to inventory](#)



Event Aggregator aka Message Bus





См. <http://martinfowler.com/eaaDev/EventAggregator.html>

10:00 **Андрей Совцов**
10:15 Все в ваших руках: быть готовым к изменениям в

10:30 **Дмитрий Никонов**
10:45 Планирование релизов в методологиях быстрой
11:00 разработки (Agile)
11:15
11:30 Казалось бы структура релизов в командах

11:45 Кофе-брейк

12:00 **Станислав Калканов**
12:15 В чем счастье заказчика? Готовые фичи вместо
12:30 гант чарта!

12:45 **Сергей Евтушенко**
13:00 Agile in Investment Banking
13:15 От ресурса к продукту: применение agile-подхода

Андрей Бибичев
Архитектура в Agile: переосмысляя идею
модульности и компонентности

Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы

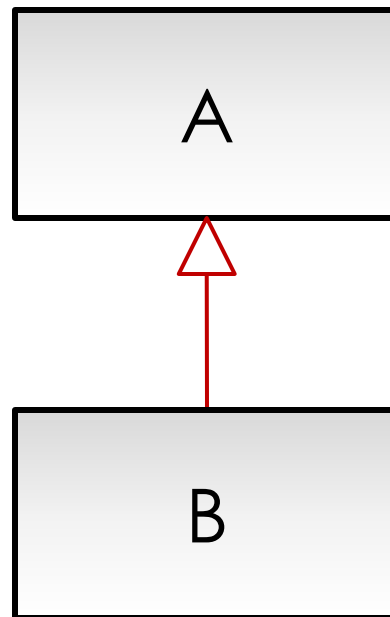
Антон Бевзюк
Архитектура для автоматизированного
тестирования UI

Николай Гребнев
Domain Driven Design в условиях разработки
распределенных приложений

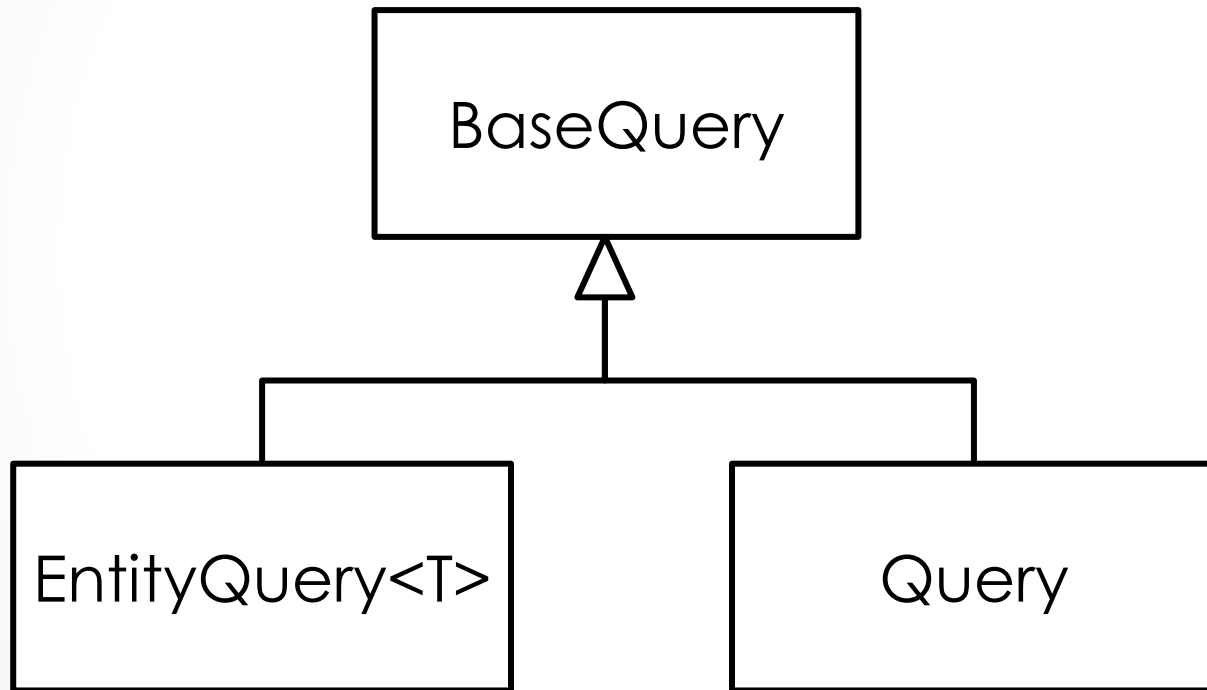
1. Классы и их взаимодействие ...

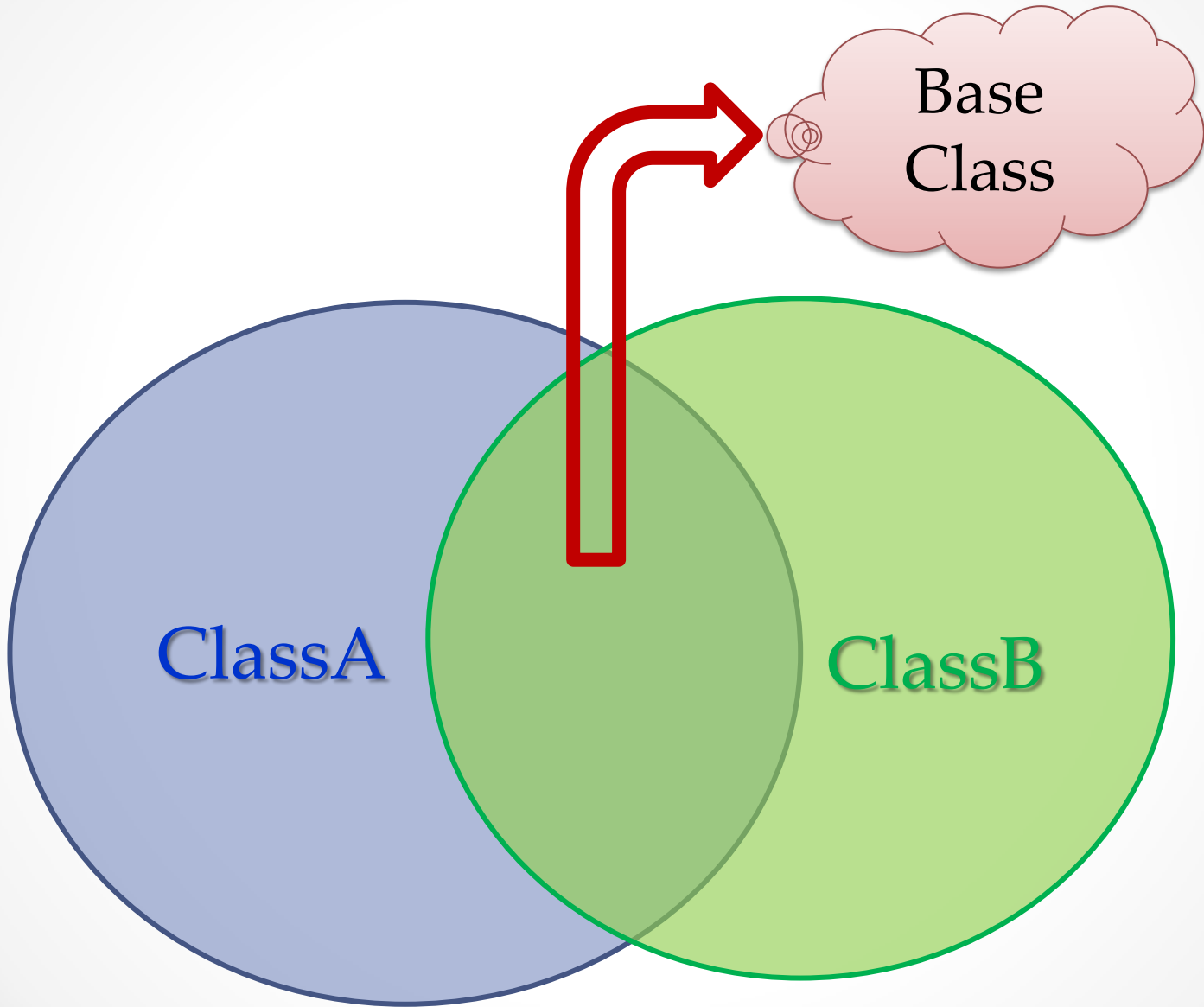
1.3. Про наследование

Наследование –
тоже вид зависимости,
причем опасный



1. Общие для нескольких классов helper-методы и данные к ним





```
class Mission {
    protected string Id { get; set; }

    protected DateTime WhatTimeIsItNow() {
        // ...
    }
}
```

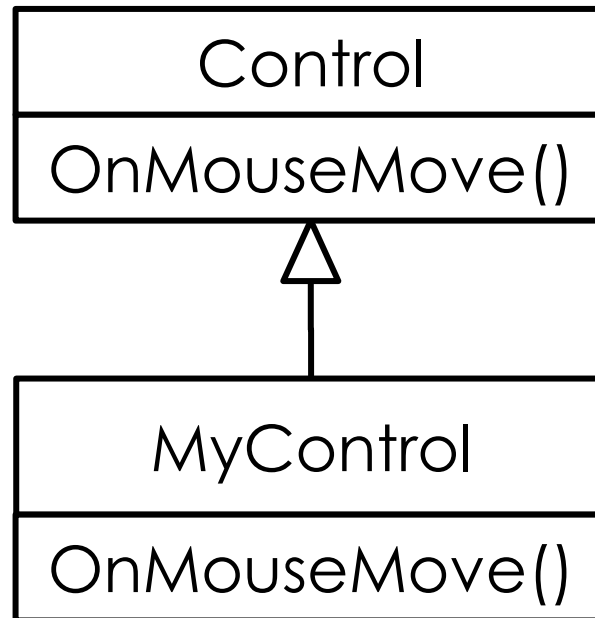
```
class FlyOnTheMoon : Mission {
    EngineCommand DetermineNextAction() {
        // ...
        var time = WhatTimeIsItNow();
        // ...
    }
}
```

```
class SleepInTheBed : Mission {
    bool ShouldIWakeup() {
        // ...
        var time = WhatTimeIsItNow();
        // ...
    }
}
```

Характерные признаки

- Как правило, наследников мало
- Не предполагается написание «своих» наследников
 - базовый класс сурово заточен под нужды своих конкретных наследников и не предполагает других наследников
- Сам базовый класс нигде в API не фигурирует
 - его использование лишено смысла!
- При правках в наследниках очень часто приходится подкручивать в базовом классе

2. Расширение функциональности за счет перекрытия виртуальных protected-методов



Умный и
мощный класс
(«на все случаи
жизни»)

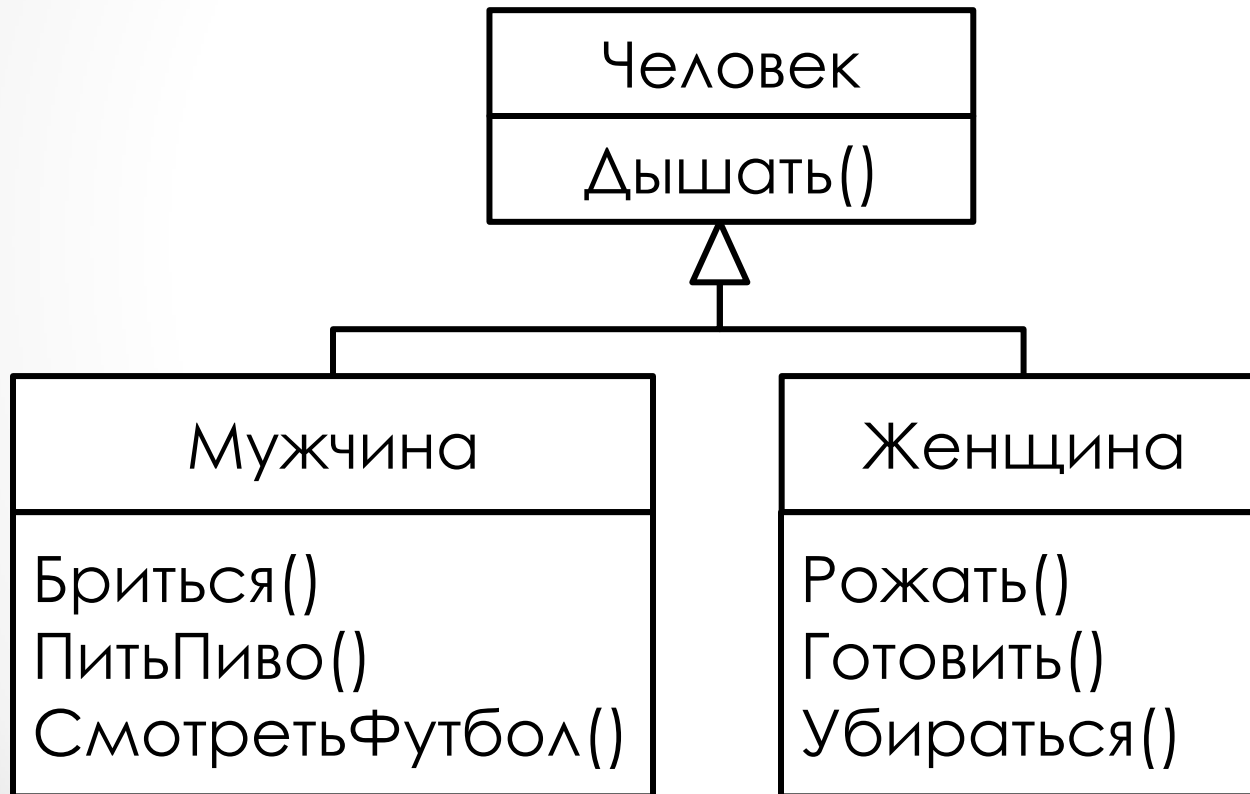


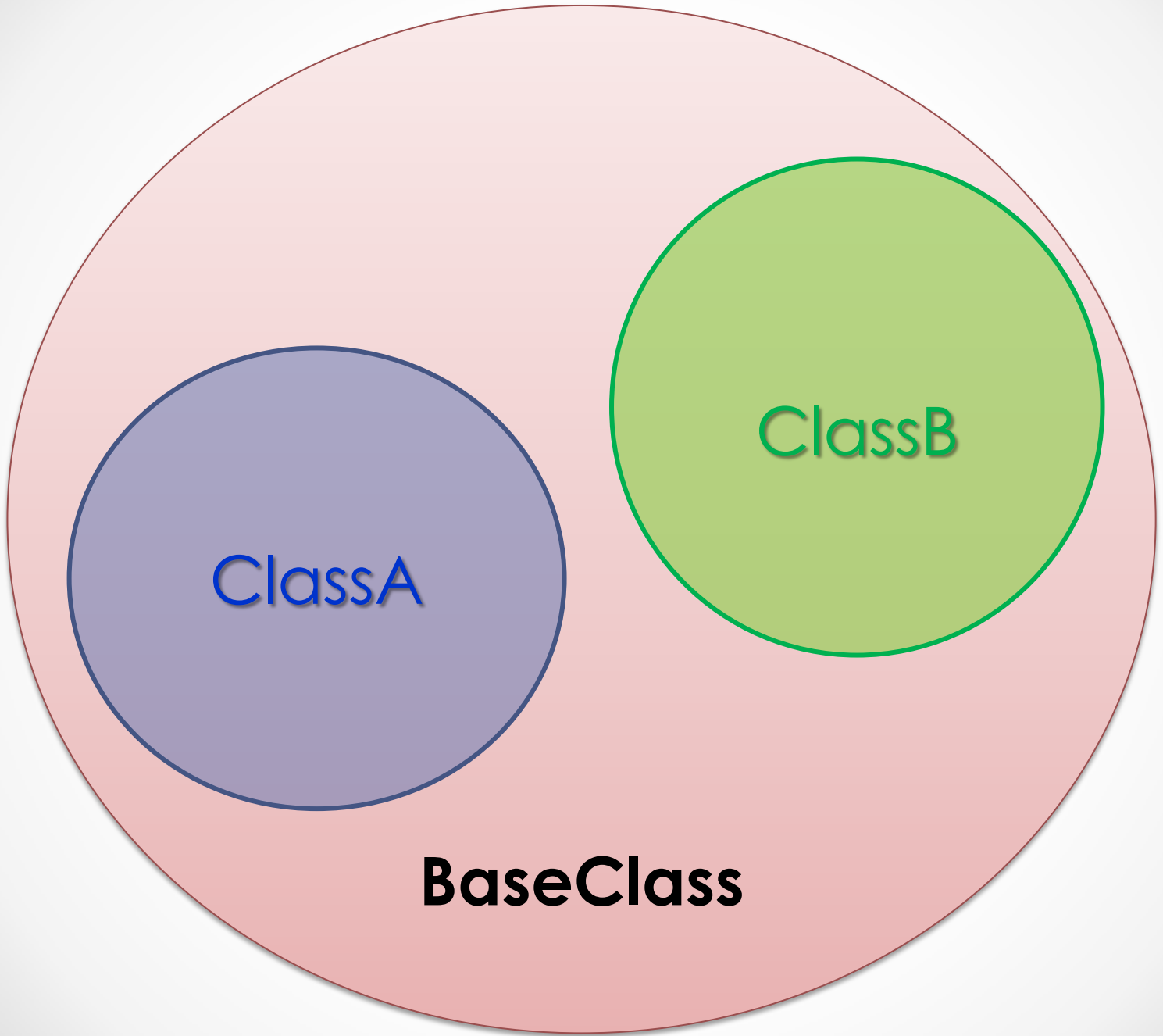
Его
исполь-
зование

Характерные признаки

- Виртуальные методы, которые активно перекрываются в наследниках
- Сам базовый класс инкапсулирует какую-то общую «идею» (обобщенный алгоритм, обобщенный компонент), которая лишена смысла без «правильного» перекрытия виртуальных методов
- Часто сочетается с (1)
- Массивными override-ами можно **чуть менее чем полностью** изменить поведение базового класса

3. Отражение отношения «является» (is)

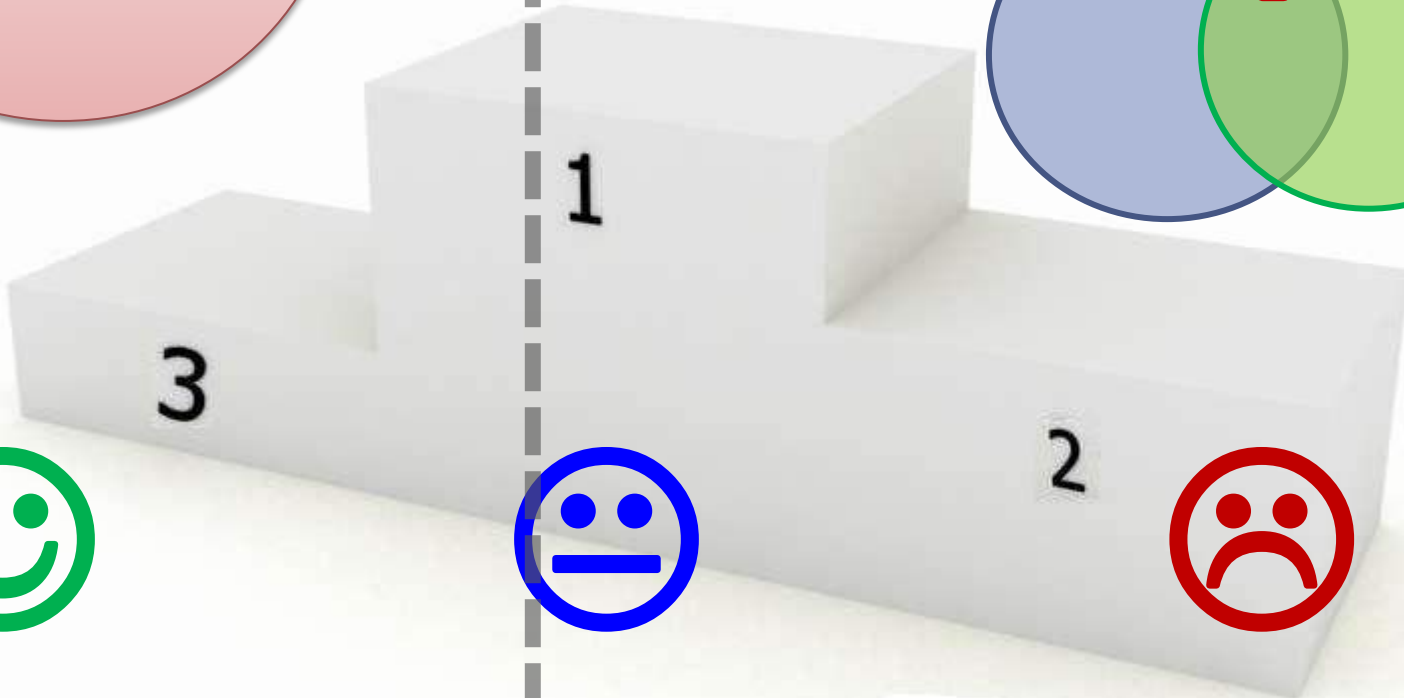
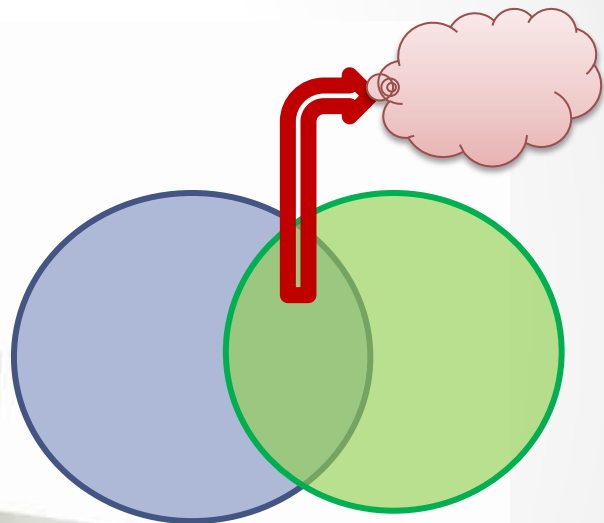
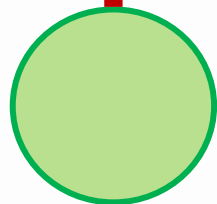
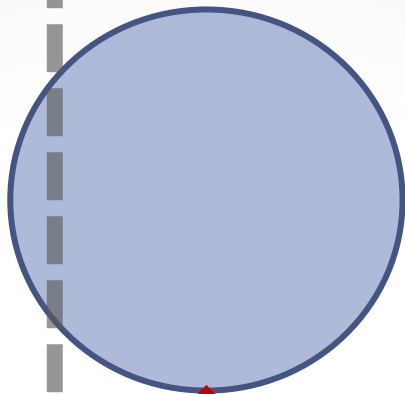
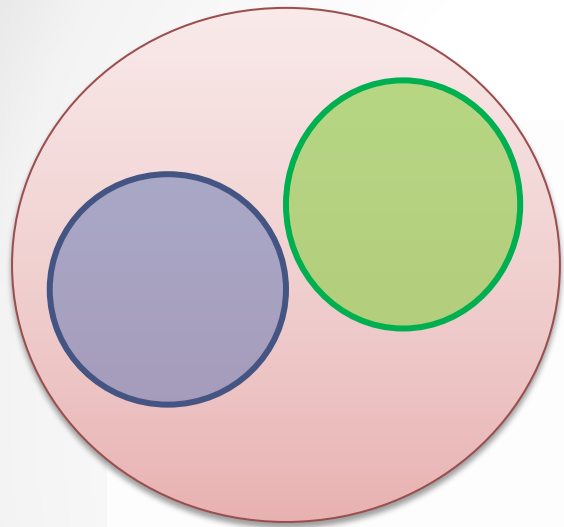




ClassA

ClassB

BaseClass



The Liskov Substitution Principle (LSP)

If for each object \mathbf{o}_1 of type \mathbf{S} there is an object \mathbf{o}_2 of type \mathbf{T} such that for all programs \mathbf{P} defined in terms of \mathbf{T} , the behavior of \mathbf{P} is unchanged when \mathbf{o}_1 is substituted for \mathbf{o}_2 then \mathbf{S} is a subtype of \mathbf{T} .

1988



GIVEN

S

\mathbf{o}_1

T

\mathbf{o}_2



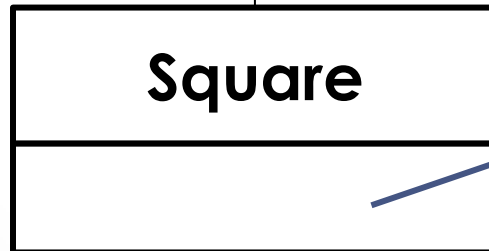
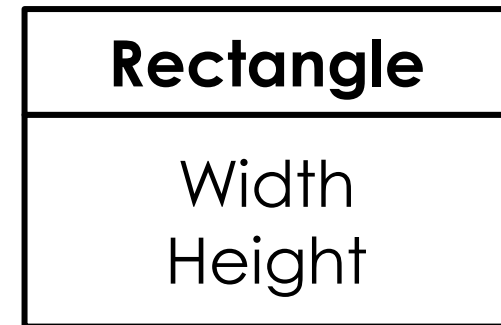
WHEN

$\forall \mathbf{o}_1 \exists \mathbf{o}_2 : \forall P \text{ResultOf}[P(\mathbf{o}_1)] \equiv \text{ResultOf}[P(\mathbf{o}_2)]$

THEN



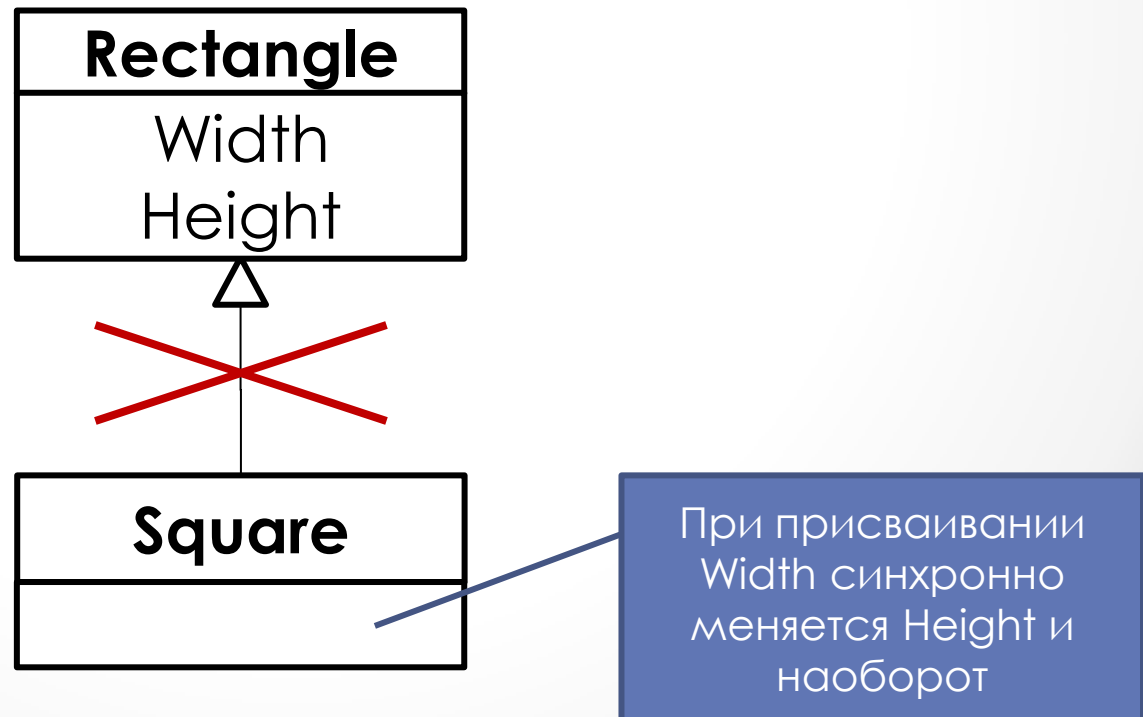
Это проясняет отношение «является» (is)



При присваивании
Width синхронно
меняется Height и
наоборот

Design-by-Contract

- Контракт базового класса не должен нарушаться наследниками

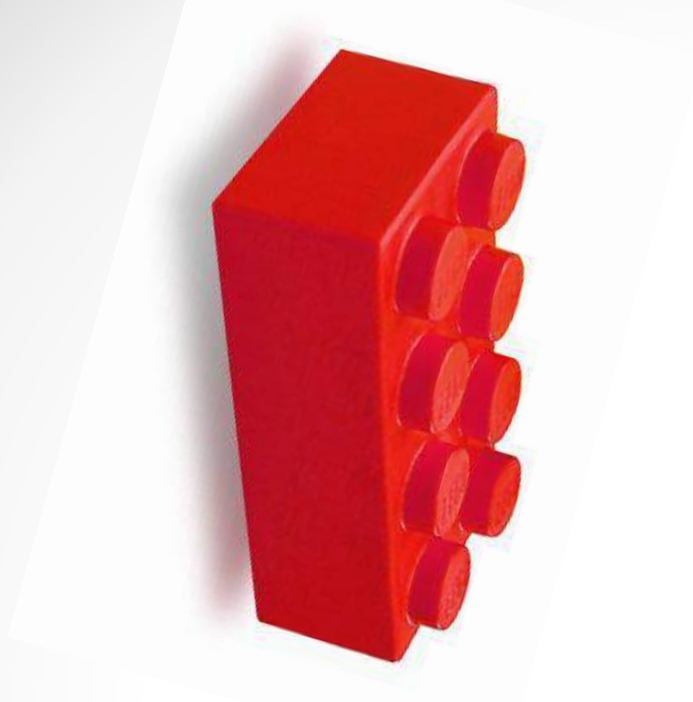


Как быть?

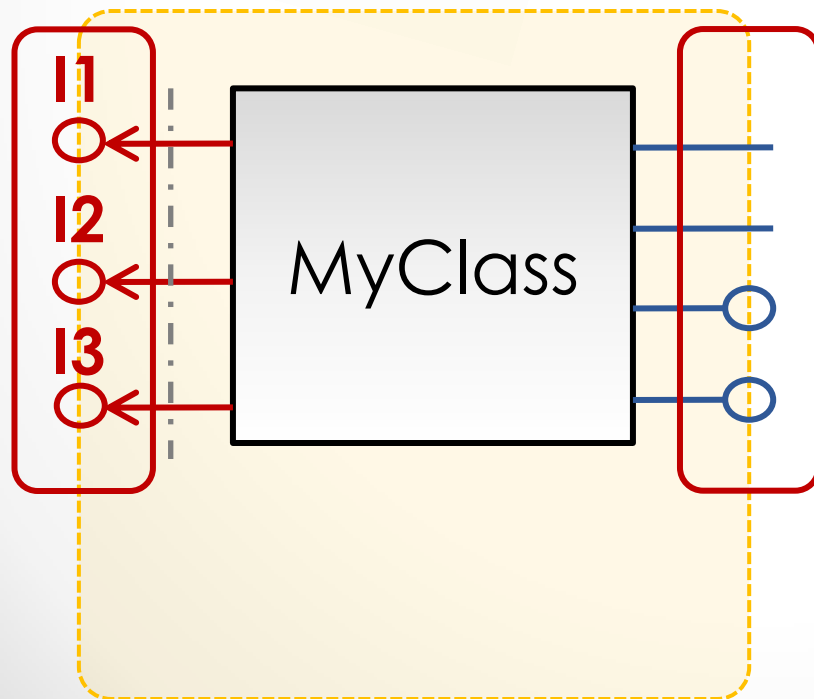
- **Interface** over Inheritance
- **Composition** over Inheritance
- **Delegation** over Inheritance

1. Классы и их взаимодействие ...

1.4. Design by Contract (DbC)



Просто
сигнатуры
методов не
хватает, чтобы
описать API
(как
требуемое,
так и
предоставляе
мое)



Три кита DbC:

- Пред-условия

Пример: `w >= 0`

- Пост-условия

Пример: `Width == w`

`Height == oldValue(Height)`

- Инварианты

Пример: `Angle == 90`

Rectangle
Width
Height
Angle
<code>setWidth(w)</code>

Инструментарий

- **Java:** cofoja (by google)

```
@Requires({
    "h >= 0",
    "h <= 23"
})
@Ensures("getHour() == h")
void setHour(int h);
```

- **C#:** CodeContracts (by MS research)

```
[ContractClassFor(typeof(IMy))]
sealed class ContractForMy : IMy {
    void setHour(int h) {
        Contract.Requires(h >= 0);
        Contract.Requires(h <= 23);
        Contract.Ensures(Hour == h);
    }
}
```

Полезен не статический анализ (он у всех хромает),
а то, что разработчик **думает о контракте**
и явно **декларирует его**

1. Классы и их взаимодействие ...

1.5. Что еще

Избавление от if-ов

- State
- Strategy
- Null-Object
- Specification

2. Модуль

...

За всё приходится платить:



Лазанья-код (ака пахлава-код)

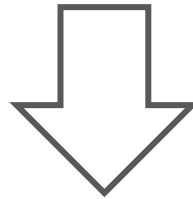
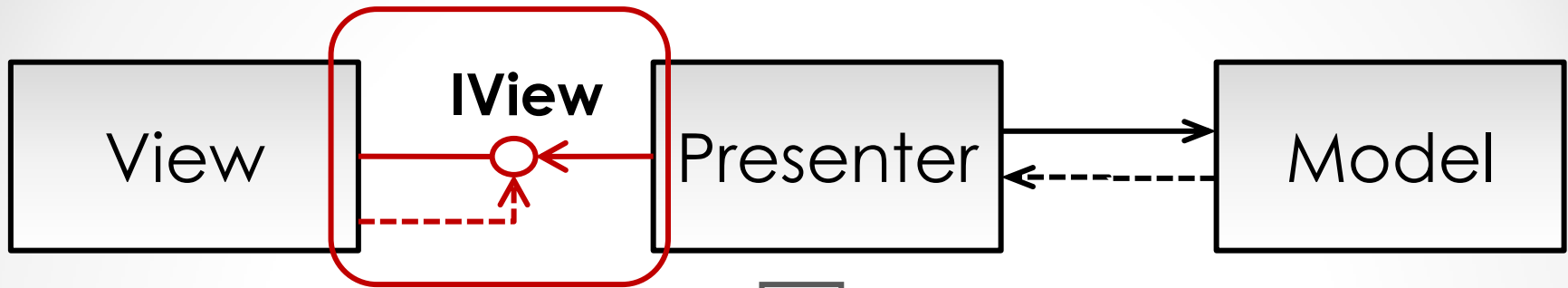
Добавление атрибута сущности:

1. СУБД: поле в таблице
2. Репозиторий (как минимум метаданные Hibernate)
3. [in-memory репозиторий]
4. Сущность
5. Data Transfer Object (DTO)
6. Presenter (N штук)
7. View (N штук)
8. А еще unit-тесты 😊

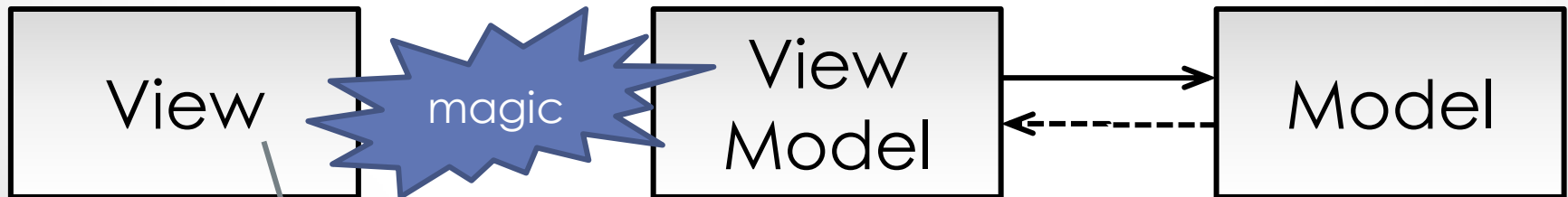
Как выкручиваться?

- Функциональное программирование
 - Задачи трансформации и обработки данных
- Аспектно-ориентированное программирование
 - Журналирование, проверка прав, валидация ...
- Декларативное программирование, DSL
 - Всё остальное рутинное 😊

Пример 1:



Model-View-ViewModel



Декларативное описание привязки свойств и событий контролов к свойствам и командам ViewModel

10:00 **Андрей Совцов**
10:15 Все в ваших руках: быть готовым к изменениям в

10:30 **Дмитрий Никонов**
10:45 Планирование релизов в методологиях быстрой
11:00 разработки (Agile)
11:15
11:30 Казалось бы структура релизов в командах

11:45 Кофе-брейк

12:00 **Станислав Калканов**
12:15 В чем счастье заказчика? Готовые фичи вместо
12:30 гант чарта!

12:45 **Сергей Евтушенко**
13:00 Agile in Investment Banking
13:15 От ресурса к продукту: применение agile-подхода

Андрей Бибичев
Архитектура в Agile: переосмысляя идею
модульности и компонентности

Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы

Антон Бевзюк
Архитектура для автоматизированного
тестирования UI

Николай Гребнев
Domain Driven Design в условиях разработки
распределенных приложений

Пример 2:

- Типовой атрибут:
 - Метаданные, описывающие тип значения
 - Метаданные, описывающие хранение
 - Метаданные, описывающие способ редактирования на UI
 - [Метаданные, описывающие права доступа]
 - [Метаданные, описывающие кеширование]
- По ним «генерируется»:
 - Обновление структуры БД
 - Кусок кода сущности
 - Реализация в классе репозитория
 - Реализация GUI -контрола

Способы генерации:

- Runtime
- Post-compile time (e.g. code-rewrite)
- Additional code generation

Мета-программирование:

- Нельзя сильно увлекаться – появляется своя слабо познаваемая эклектика и “зюко-хрючный” синтаксис
- Тяжело обеспечить адекватным инструментарием для удобной работы с метаданными
- Должна быть возможность всегда **просто** вставить императивный код (Ant vs. Scons)

DSL: JetBrains MPS

The screenshot shows the JetBrains MPS IDE interface. The title bar reads "collections - jetbrains.mps.baseLanguage.collections.samples.sequence_queries\Main_where_select_operations - J...". The menu bar includes File, Edit, View, Go To, Generate, Build, Tools, Version Control, Window, and Help. The toolbar contains various icons for file operations and development tools. On the left, the Project tool window shows a tree view of the project structure, including folders like "internalCollections" and "collections_sample", and files like "Main_where_select_operations". The main editor window displays the following code:

```
<<constructor>>
<<methods>>
/*package*/ static void main(string[] args) {
    sequence<Integer> nums = new sequence<int>({ =>
        for (int i = 0; i < 10; i++) {
            yield i;
        }
    });
    System.out.println("from sequence of 10 numbers,");
    System.out.println("where num is even,");
    System.out.println("produce string");
    int count = 0;
    sequence<string> strings = nums.where({~n => return n % 2 == 0;}).select({~n =>
        count = count + 1;
        return "num:" + n;
    });
    System.out.println("count (before):" + count);
    foreach s in strings {
        System.out.println(s);
    }
    System.out.println("count (after):" + count);
}

<<static inner classifiers>>
```

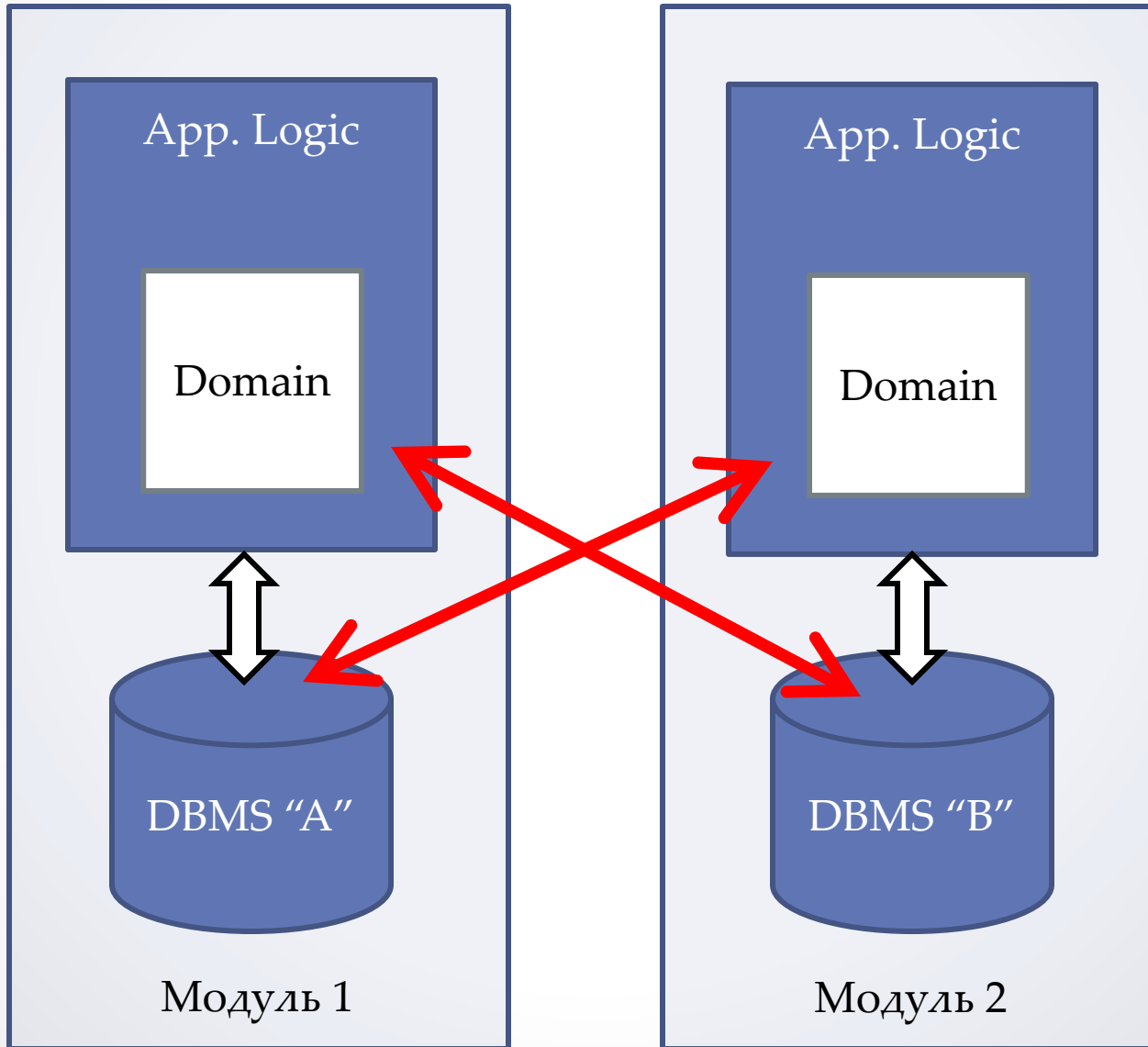
The code uses a DSL for sequence operations, including `sequence<Integer>`, `yield`, `where`, `select`, and `foreach`. The IDE status bar at the bottom shows "2: Changes", "0: MPS Messages", "5: Inspector", and "232M of 441M".

Пока больше распространены:

- Java: Groovy
- .Net: Boo

3. Взаимодействие модулей ...

Проще всего так:



Ущербность такого подхода
ПОНЯЛИ ДАВНО
И ИСПОЛЬЗУЮТ **сервисы**

и всё это загажено
маркетинговым булшито́м ☹️

Приемы проектирования аналогичны:

- **Контракт** на API (интерфейс) + Data Contract
- Подсистемы-**адаптеры**
- **Шина**

Серьезное «НО»:

Abstraction Penalty



Примеры борьбы:

- Накладные расходы на транспорт
 - SOAP => REST
- Распределенные транзакции
 - Double-check commit => Идемпотентные операции

упрощение абстракций!

Идемпотентные операции:

- Повторный вызов ничего не ломает и ничего не дублирует
 - Просто выполняет действие, если до этого оно почему-то не было завершено
 - Если же операция уже была выполнена, то ничего не делая, возвращает результат её выполнения
- Очень легко реализуется
 - Желаящим расскажу отдельно 😊

4. ИТОГ

...

Зачем всё так сложно?!

- Основная ценность – **работающий код!**
- Если вы практикуете unit-тестинг, то следующая ценность – **тестопригодный код**
- Чтобы не накапливать технический долг и не становиться заложником кода – **понятный и сопровождаемый код**

код, который легко сопровождать и развивать,
писать сложнее и дольше, чем просто
работающий код

макет / мини-тула

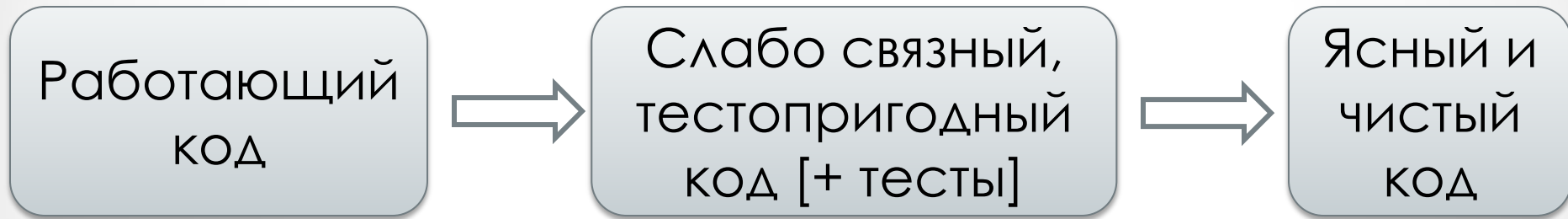


что-то большое

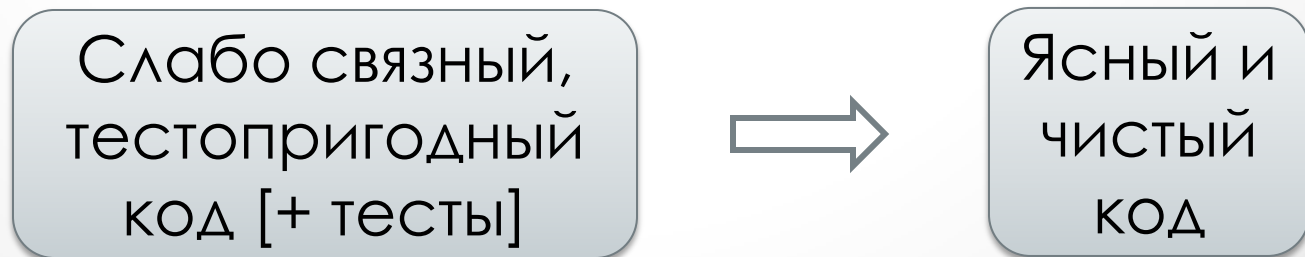


Если вы согласны потратить доп.усилия:

1. Research / Алгоритмика / Высокие тех. риски:



2. Уже есть опыт (по аналогии) / Если вы очень крутой:



Agile как Буддизм

- S.O.L.I.D. принципы
- закон Деметры + Tell, don't ask
- DbC
- Composition/Delegation over Inheritance
- функциональное программирование
- мета-программирование и DSL
- REST
- индемпотентные сервисы
- ...

Это не заповеди

Это направления возможных улучшений

Спасибо за внимание!

Вопросы?

bibigone@gmail.com

[@bibigine](#)

<http://www.google.com/profiles/biBIGone>

<http://www.slideshare.net/bibigine>