Ag;)eDays'11

Архитектура в Agile: ~~компоненты и модули~~ слабая связность

Бибичев Андрей

март 2011

@bibigine

Андрей Бибичев

- E-mail: bibigone@gmail.com
- Twitter: @bibigine
- Profile: http://www.google.com/profiles/biBIGone
- Slideshare: http://www.slideshare.net/bibigine

iPi Soft

ScrumTrek

0. Проблематика

Alistair Cockburn:

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

Mary and Tom Poppendieck:

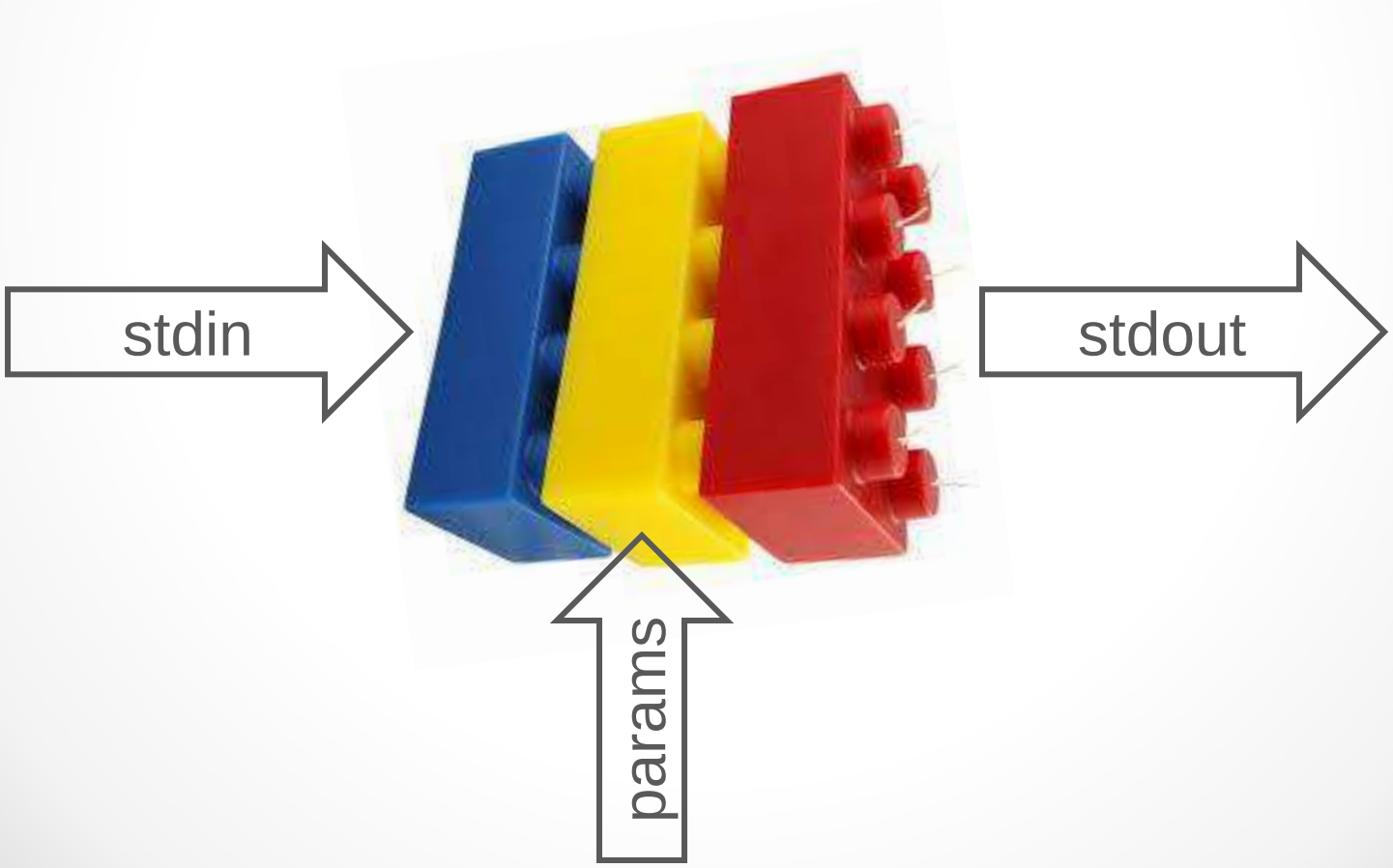
«divisible system architecture»

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

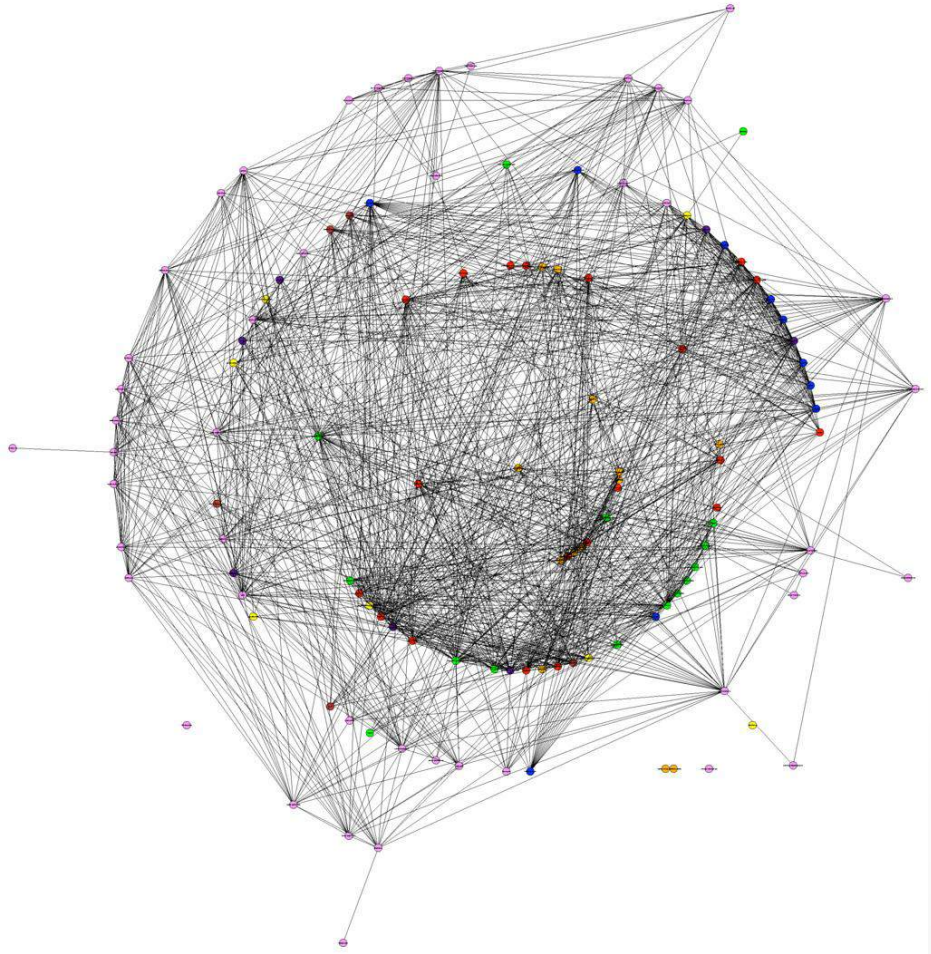
1. goto
2. подпрограммы
3. динамические библиотеки
4. классы (ООП)
5. компоненты (COM/CORBA/…)
6. …

Но что-то не легчает №1…

Корнелис Киммель « Снежный ком»

Сильная связность

≈ каждый с каждым => $O(N^2)$



Но что-то не легчает №2…

Код-гидра:
один баг исправишь - несколько появляются


HERCVLES VNÀ CVM IOLAO HYDRAM OCCIDIT.

Side-эффекты

Могучий глобальный контекст

+

Мутабельность всего и вся

+

Избыточная область видимости

=

поправил здесь, отвалилось там

Но что-то не легчает №3…

WinForms.Net

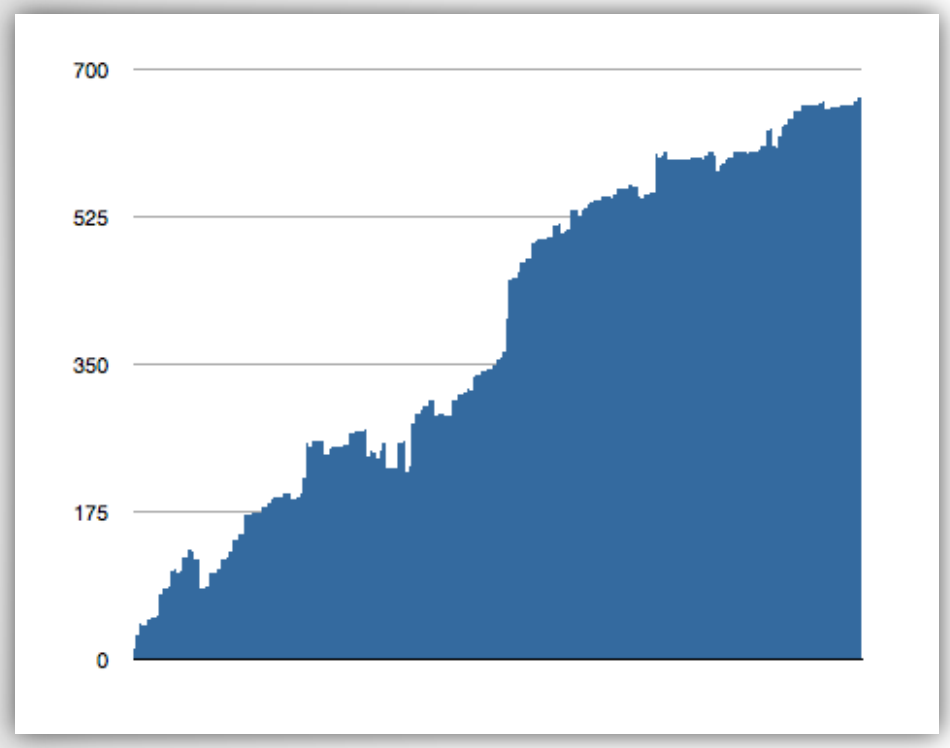
Класс **Control**:

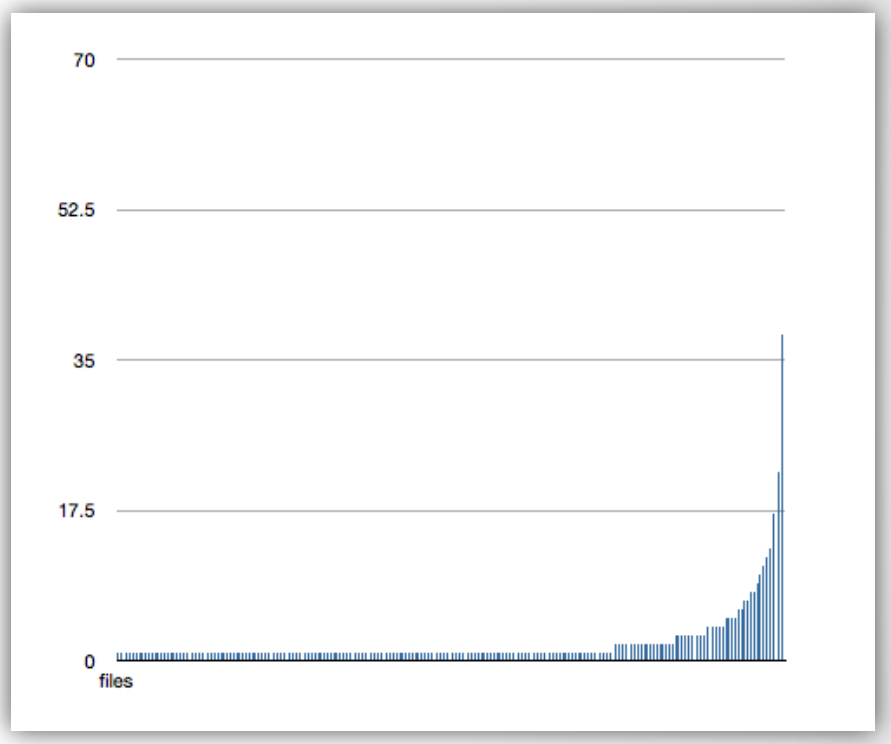
100+ свойств (≈80% public)
400+ методов (≈30% public)
50+ событий (все public)

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

10 992 – 1673 + 1 = **9 320** строк

LUGARU HD

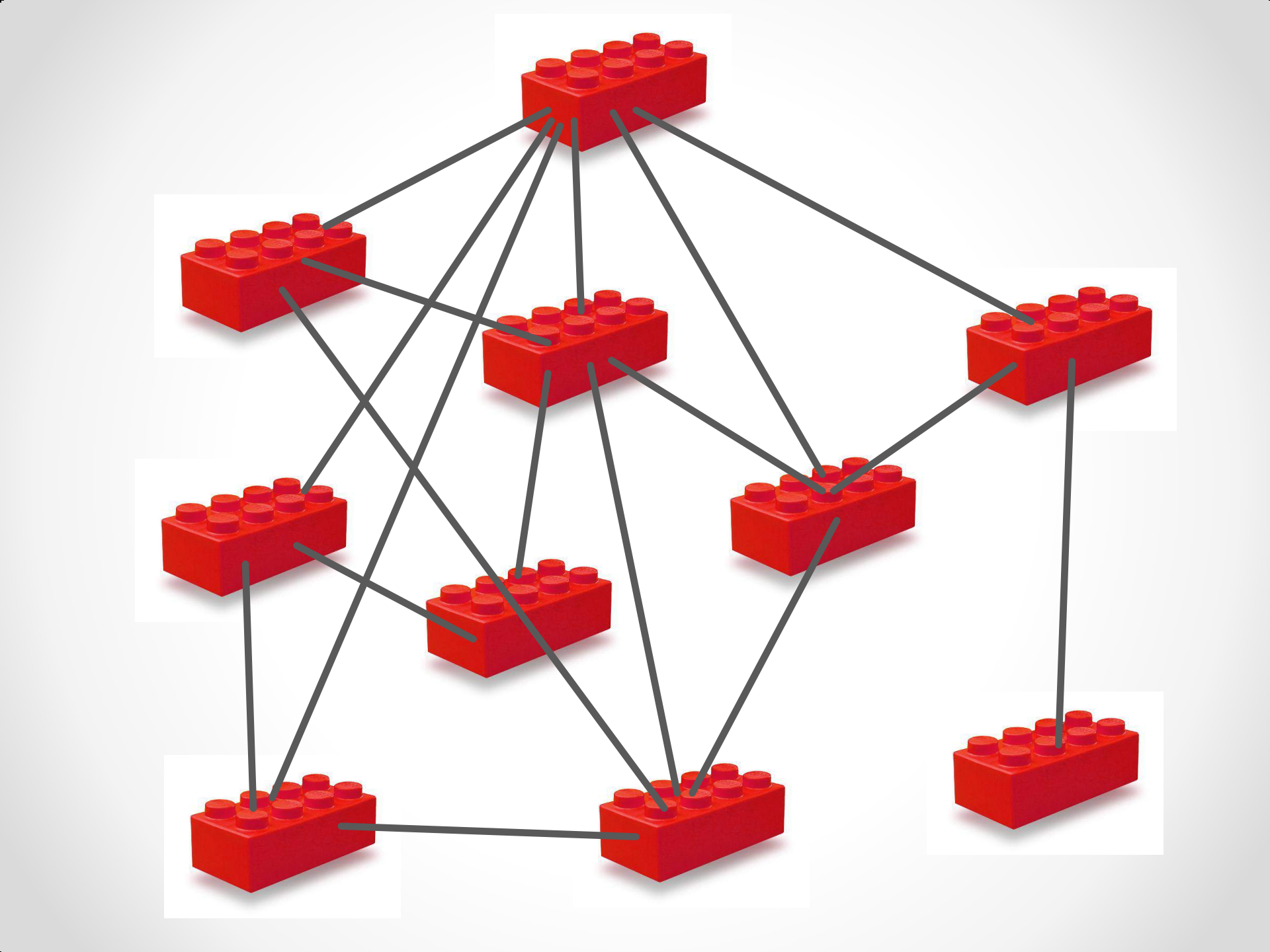
ENTER

OPTIONS

QUIT

Типичные болезни:

- Сильная связность
- Неуловимые и непознаваемые side-эффекты
- Вездесущий и всеобъемлющий глобальный контекст (привет любителям singletone-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;
};
```

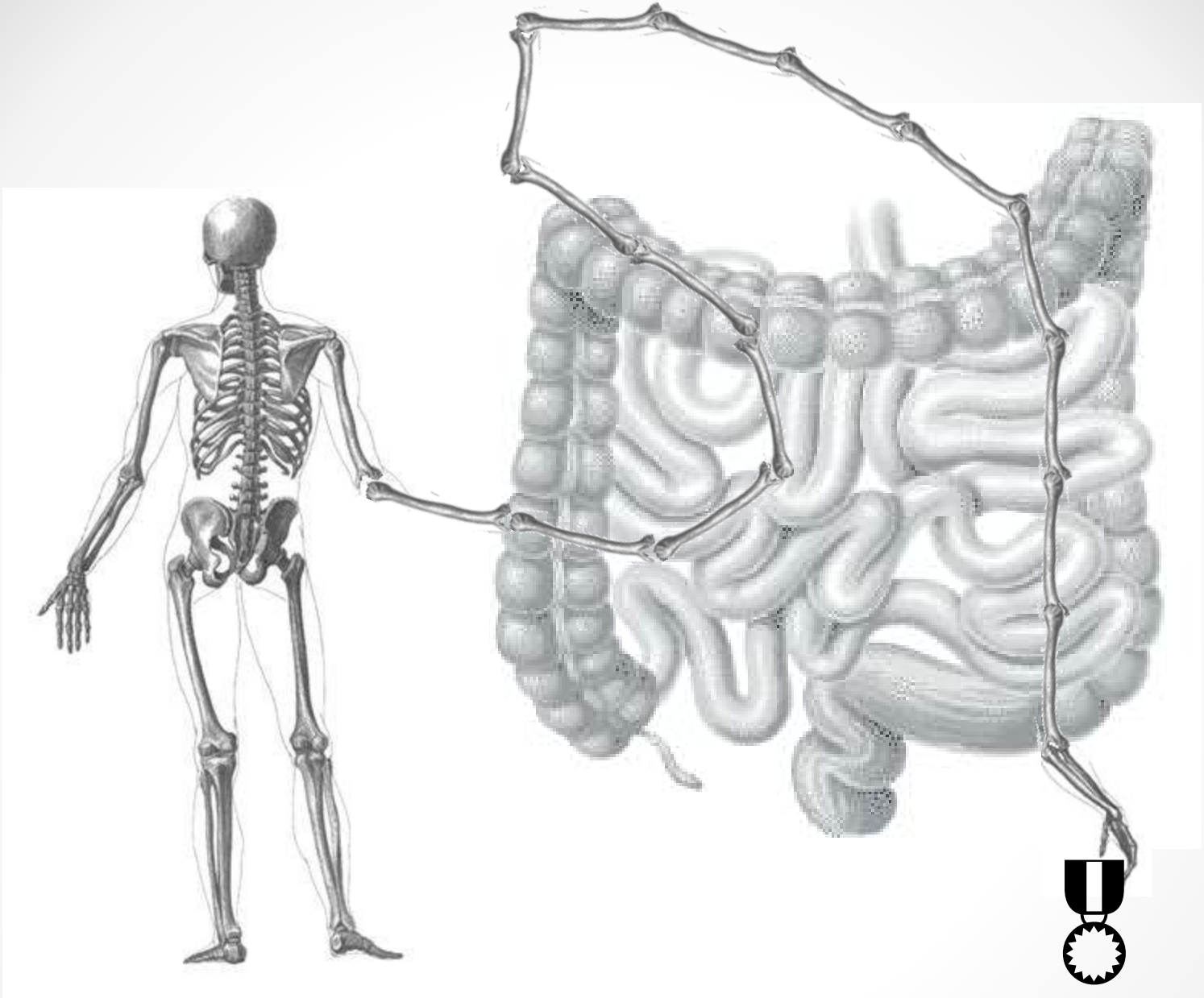
```
typedef void (*f1_t)(int s);
typedef double (*f2_t)();
```

- Наследование => Композиция

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

1. Классы и их взаимодействие

1.1. Современное ООП



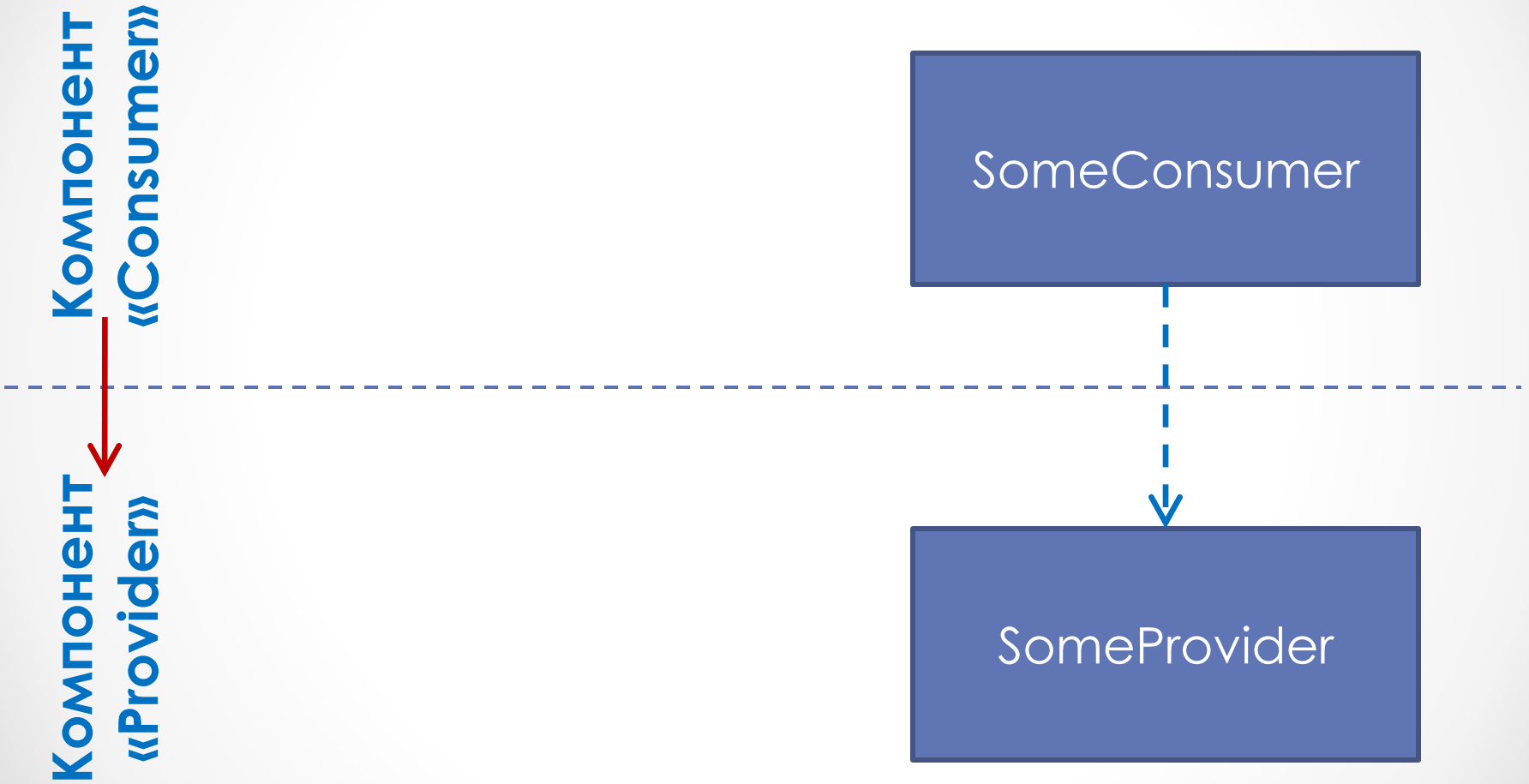
Длинные «руки» объекта:

если ему чего нужно, то он сам за этим тянется

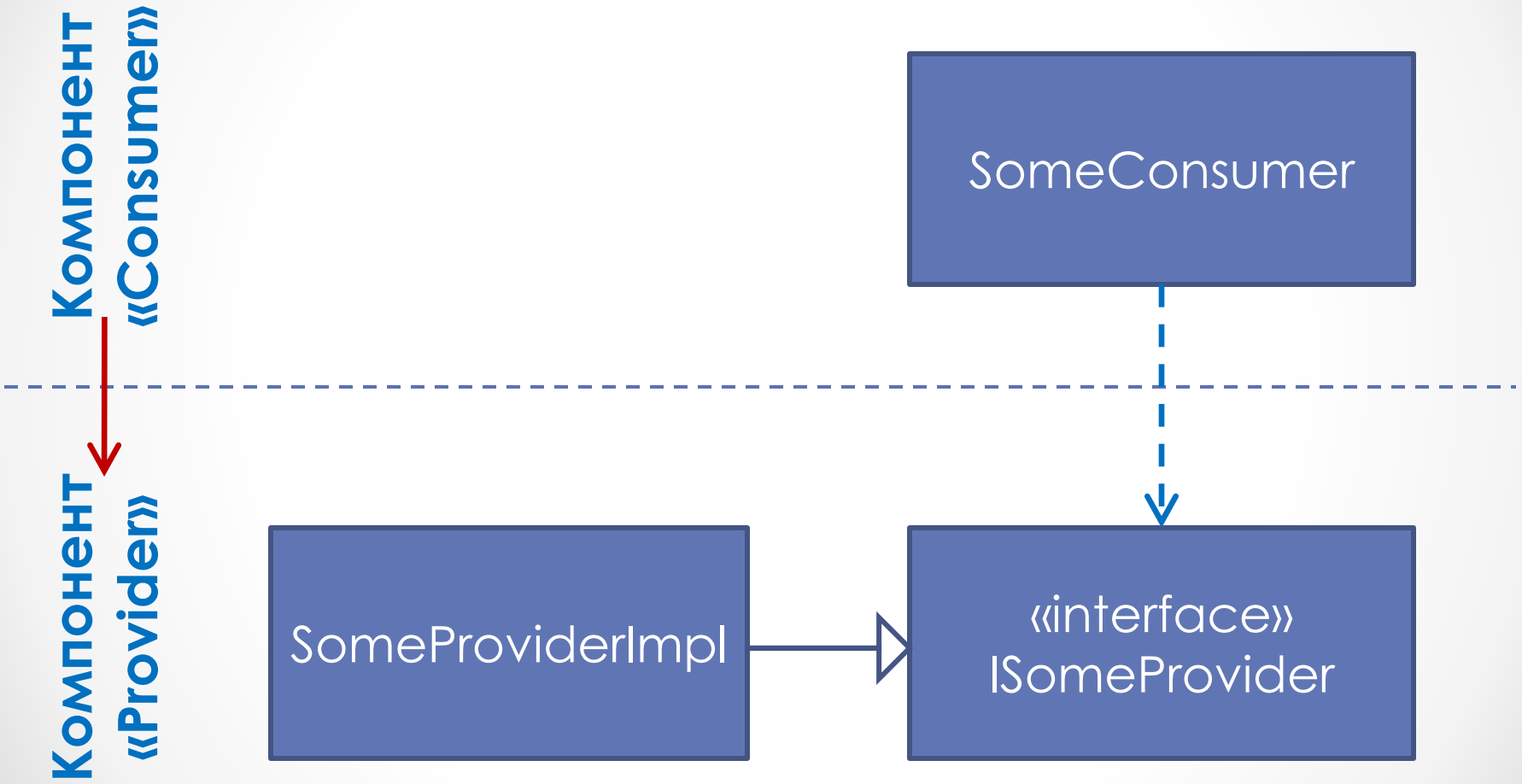
Укорачиваем руки:

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

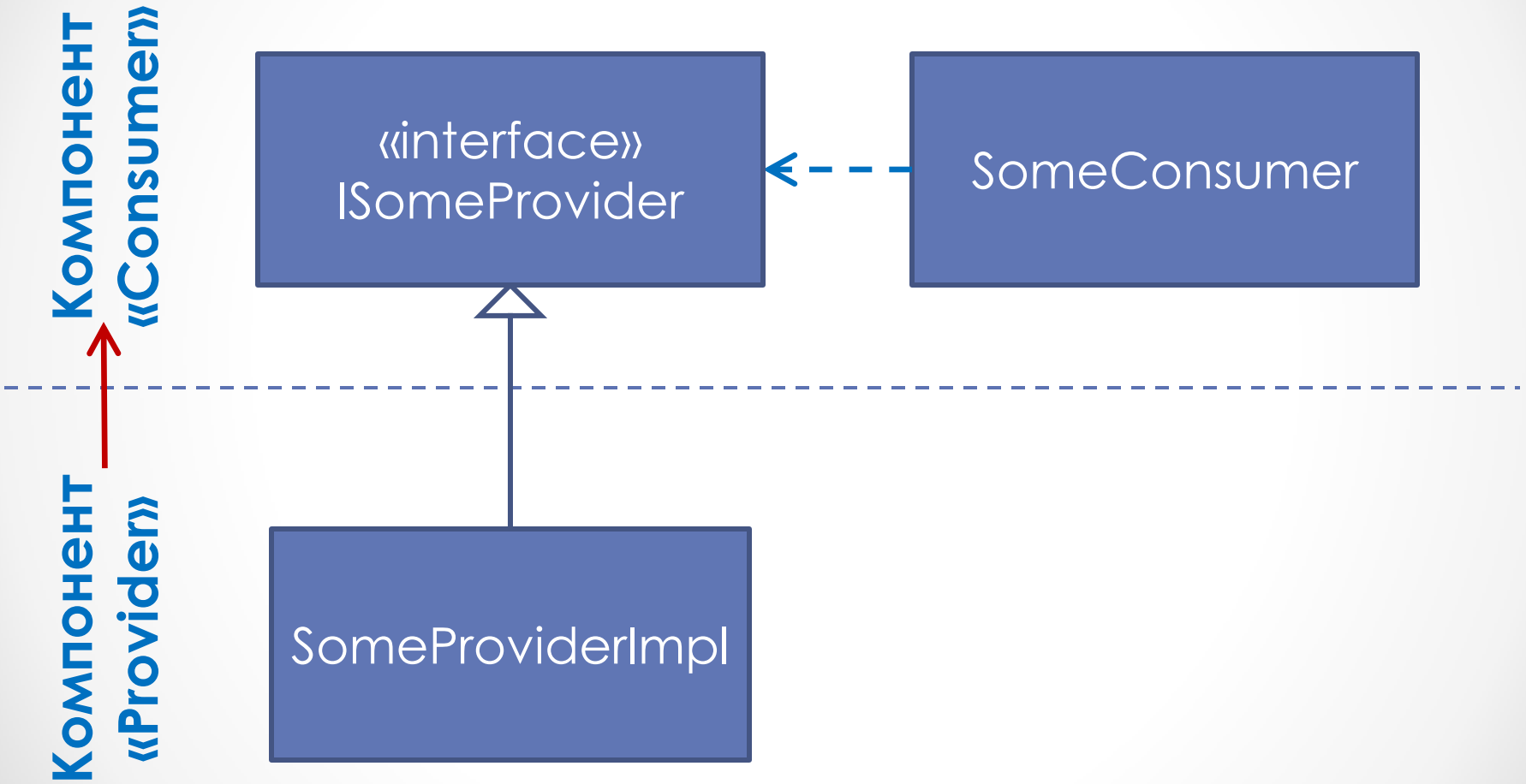
Inversion of Control (IoC)

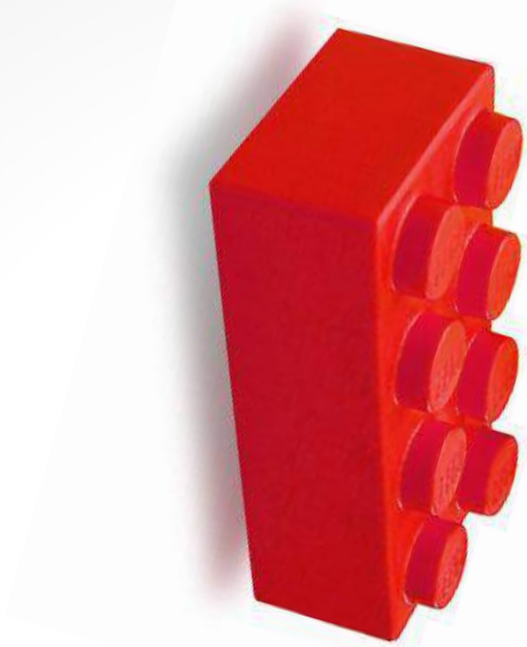


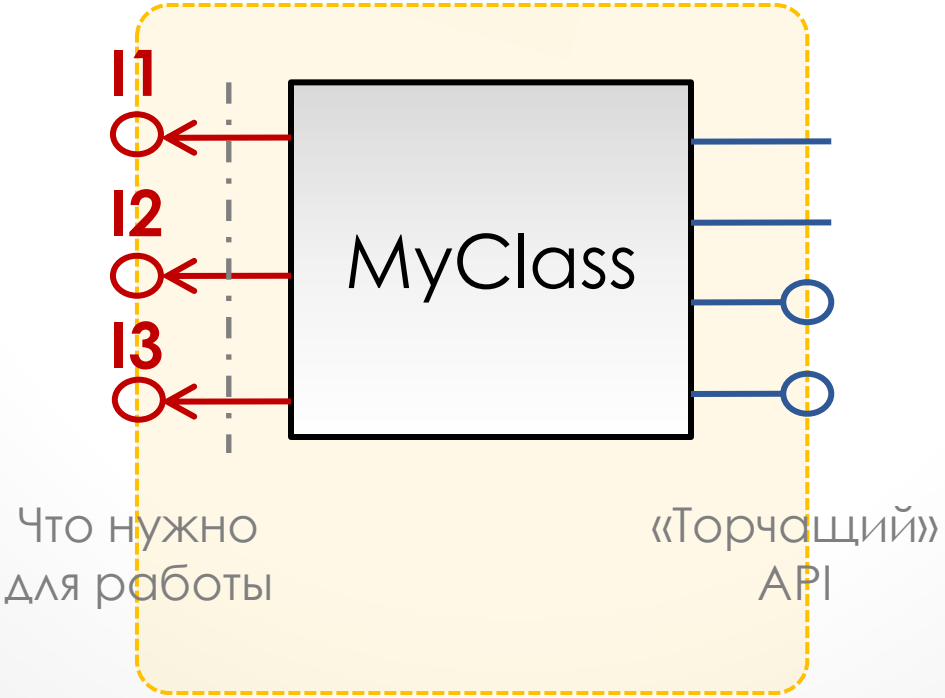
Inversion of Control (IoC)



Inversion of Control (IoC)






I1
I2
I3
MyClass
Что нужно
для работы
«Торчащий»
API

Как передать имплементации 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.Singletone.get<IService>().do()
```

- Цепочки вызовов

```
field.getSmth().getSmthElse().Prop.Count
```

☺:

- True singletone (e.g. Null Objet)

А работа с фабрикой?

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

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

- Казалось 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

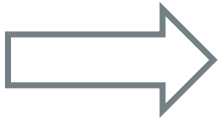
```
interface IObj {
    void performInTransaction(Callback action);
}
```

Где риализовывать:

- 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

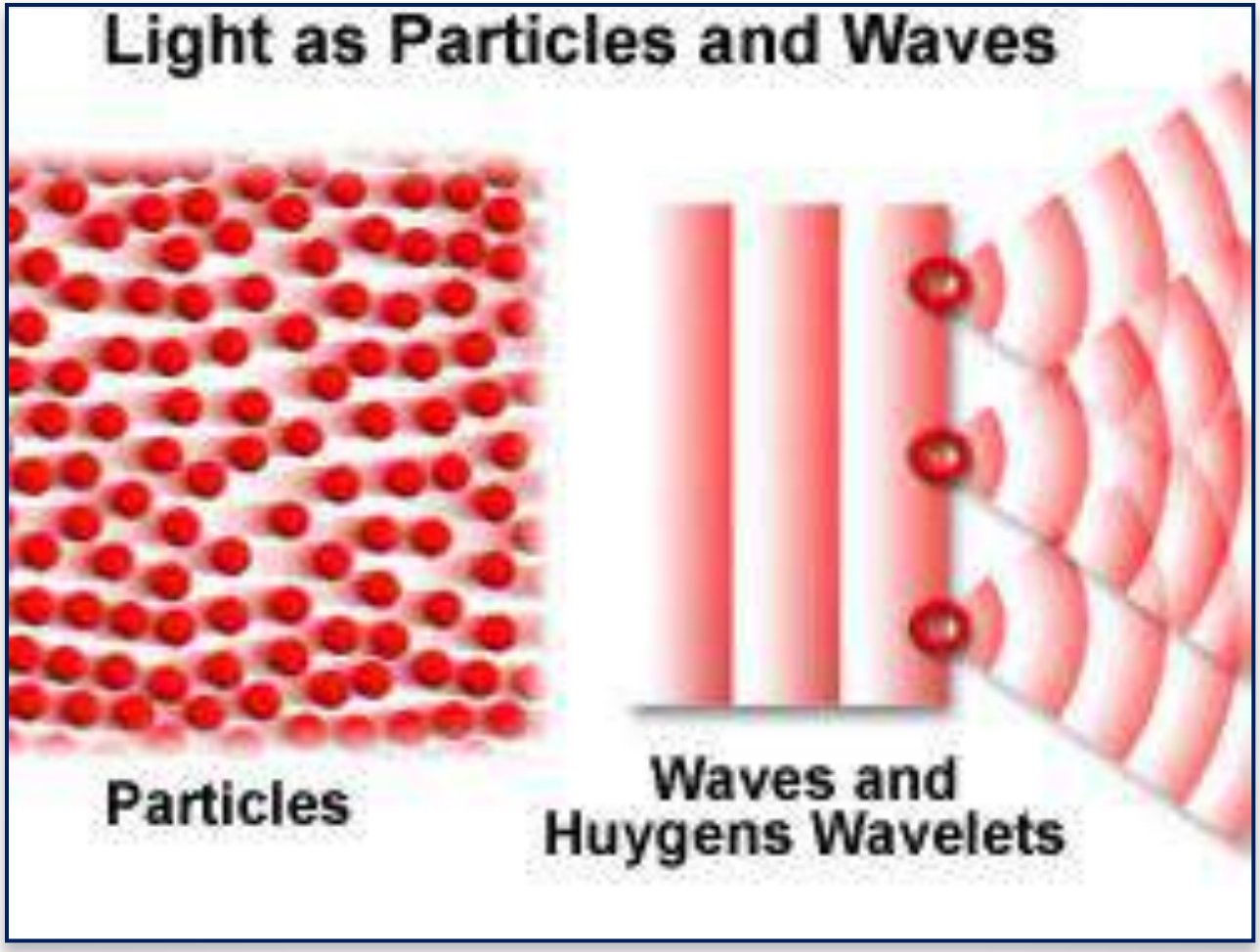

```
class MyDoc {
    MyDoc(ICurrentDateProvider cdp) {
        docDate = cdp.get();
        …
    }
```

```
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 карпускулярно-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

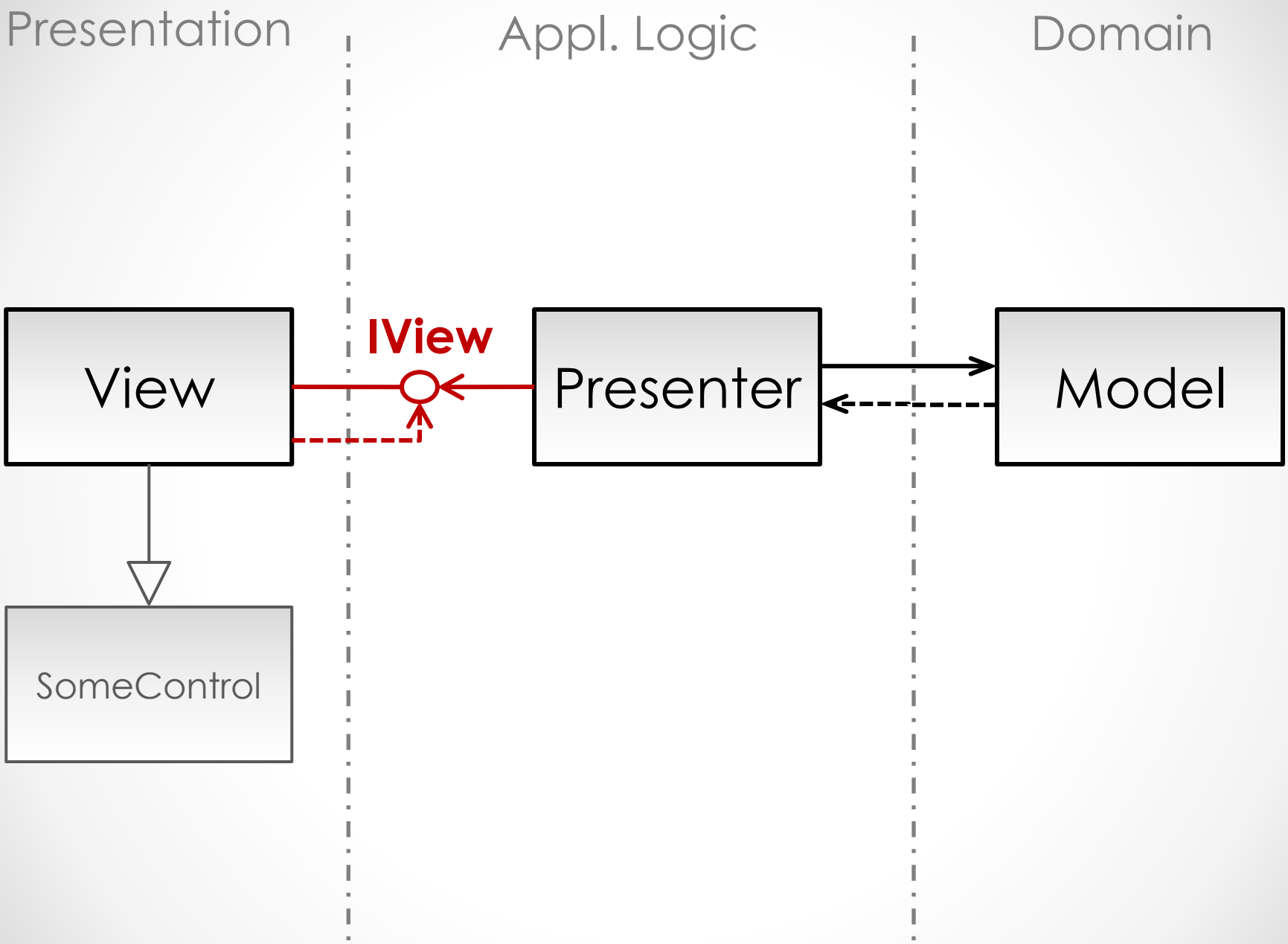
1. Классы и их взаимодействие

...

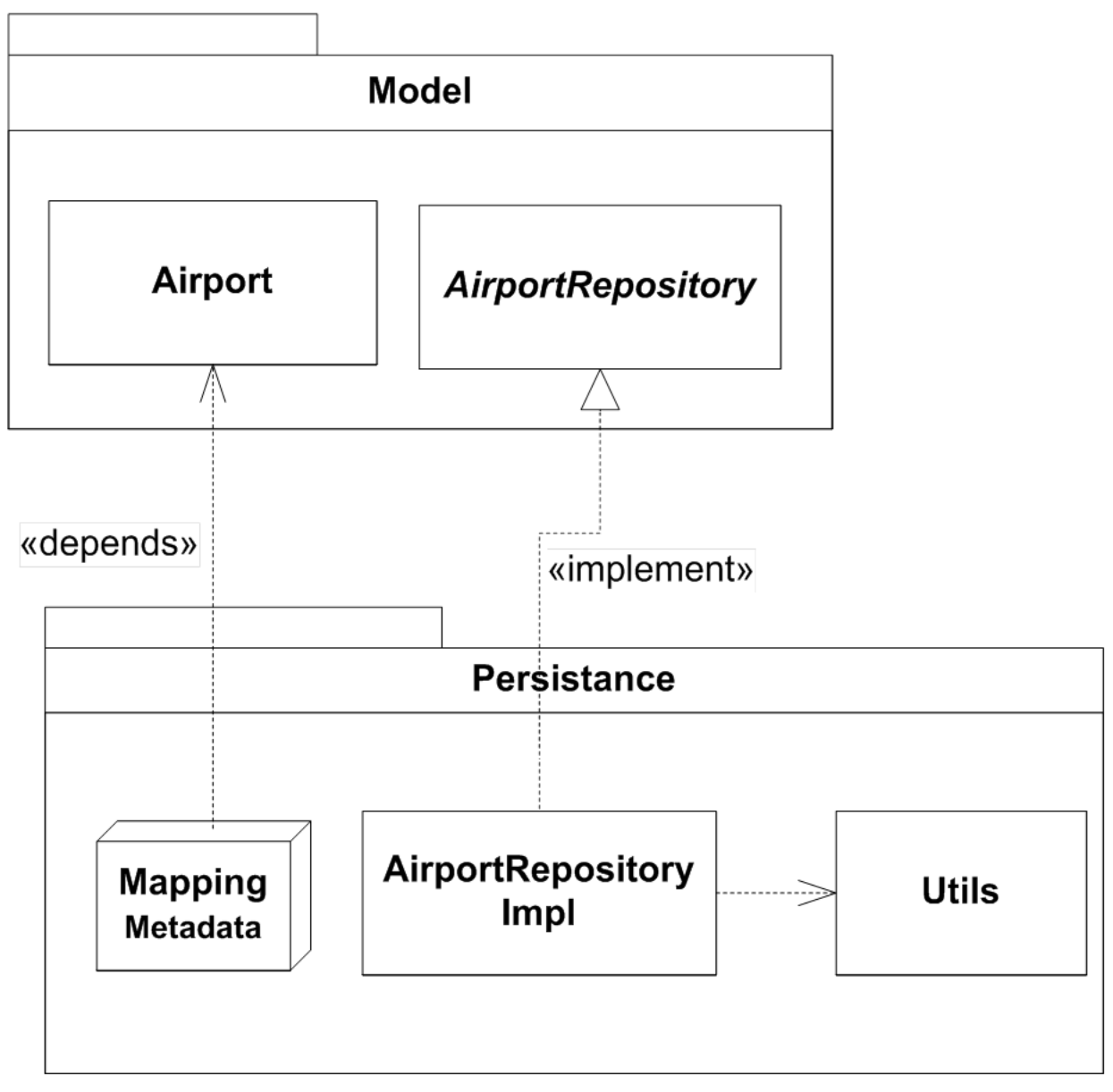
1.2. MVP и Entity-Repository

Классика жанра

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

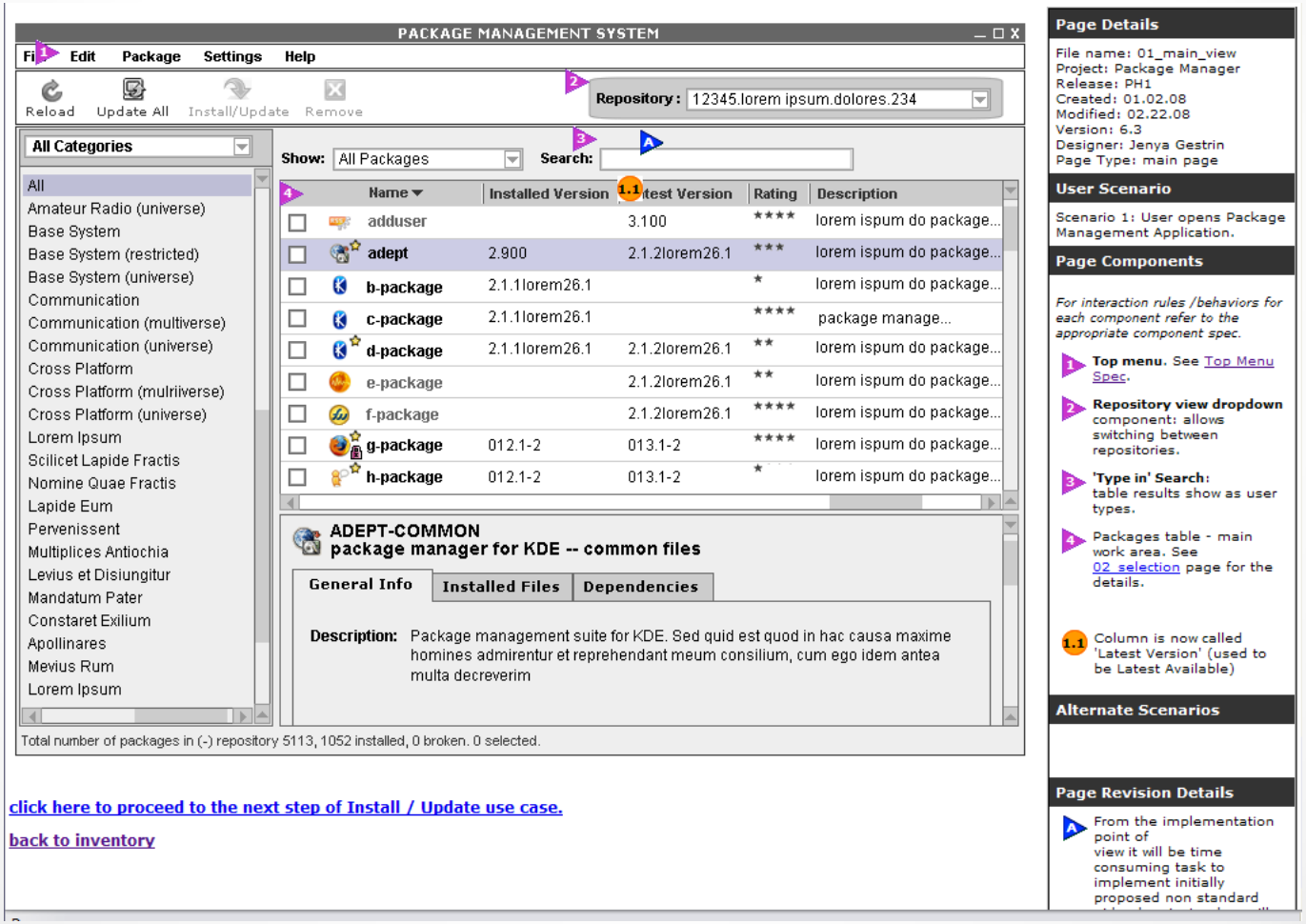

Presentation
Appl. Logic
Domain
View
IView
Presenter
Model
SomeControl

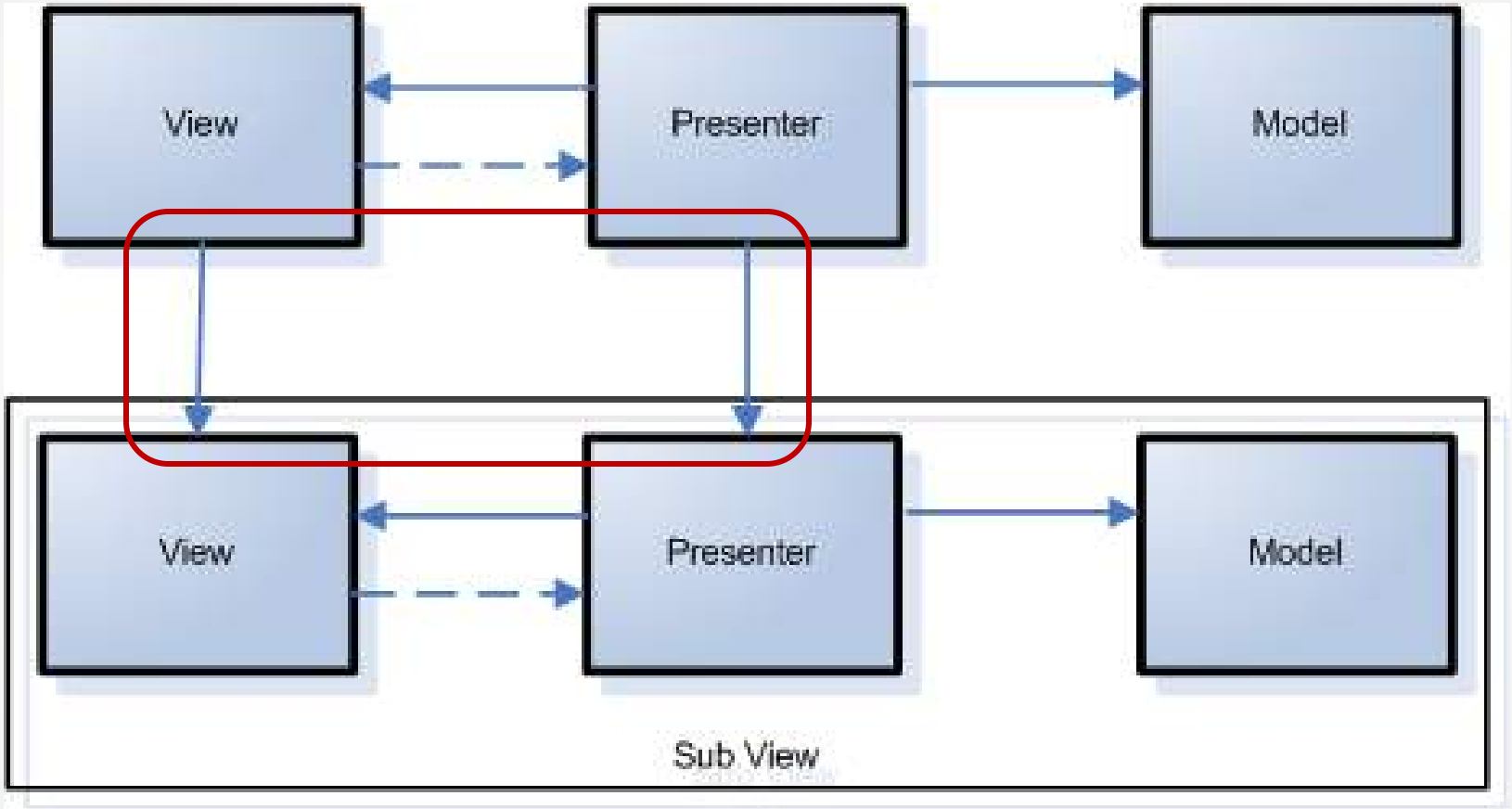
10:00		
10:15	**Андрей Совцов** Все в ваших руках: быть готовым к изменениям в	**Андрей Бибичев** Архитектура в Agile: переосмысляя идею модульности и компонентности Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы
10:30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


Model
Airport
AirportRepository
«depends»
«implement»
Persistance
Mapping
Metadata
AirportRepository
Impl
Utils

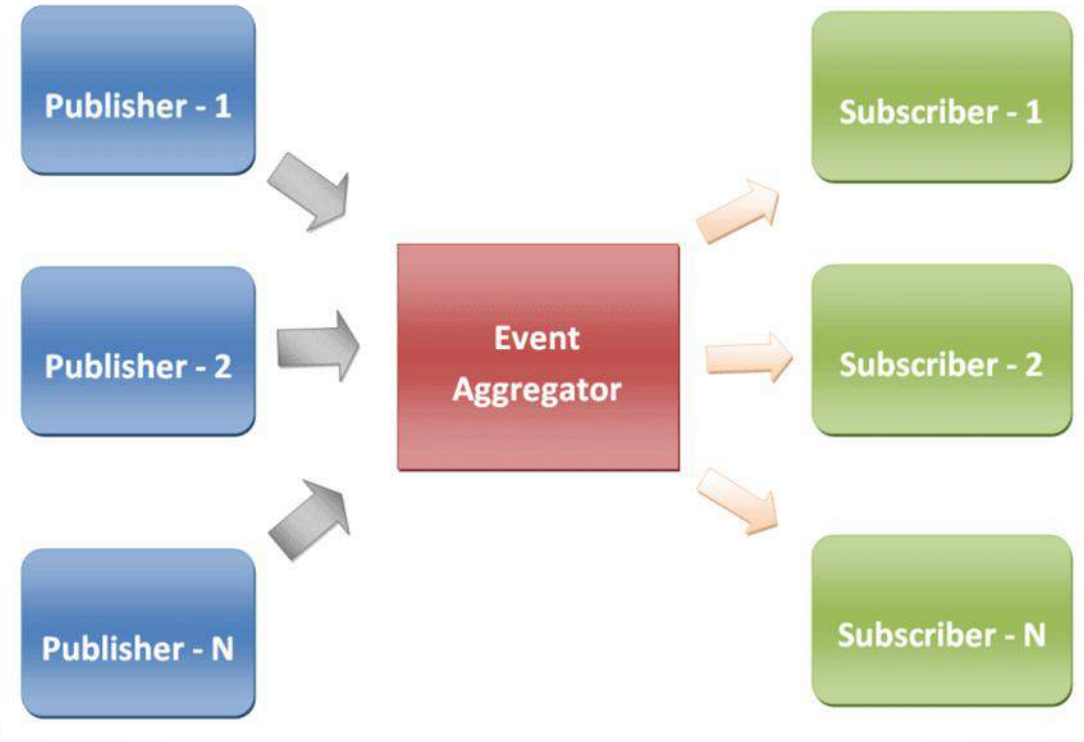
10:00		
10:15	**Андрей Совцов** Все в ваших руках: быть готовым к изменениям в	**Андрей Бибичев** Архитектура в Agile: переосмысляя идею модульности и компонентности Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы
10:30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

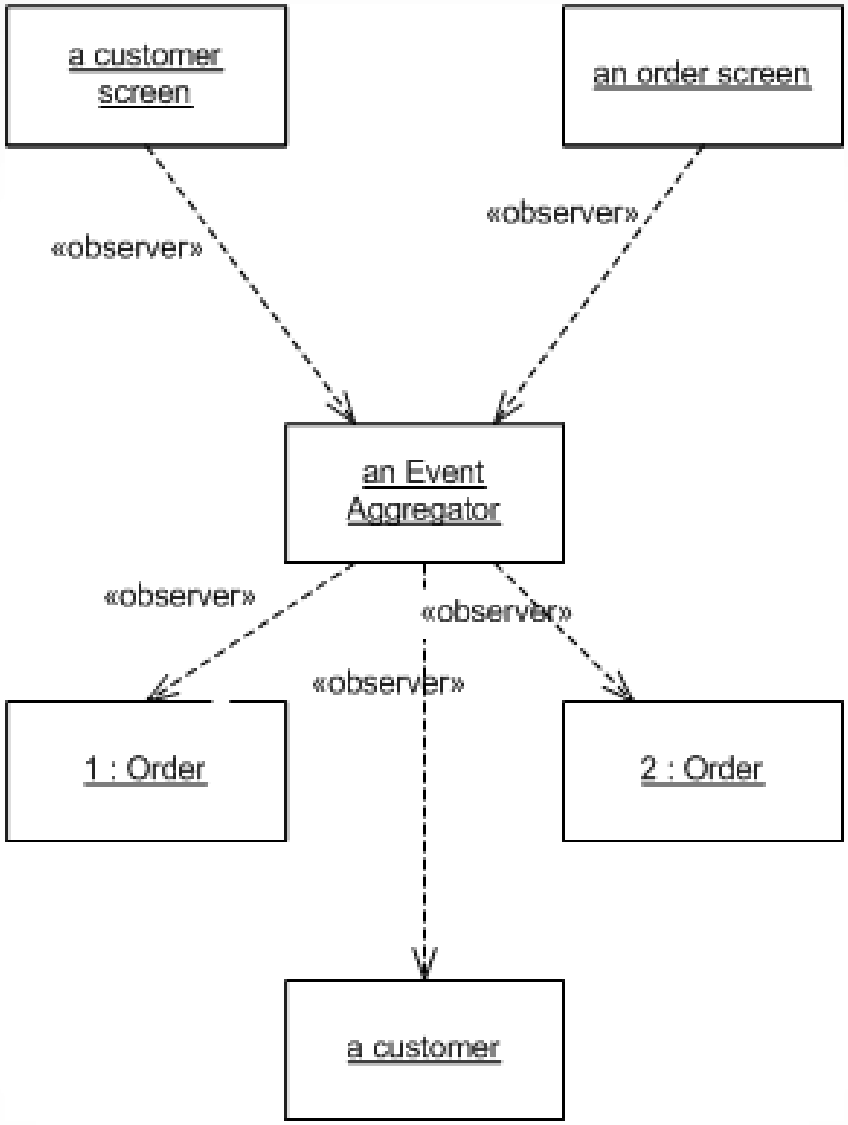
Как декомпозировать сложную форму?




View
Presenter
Model
View
Presenter
Model
Sub View

Event Aggregator
aka Message Bus





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

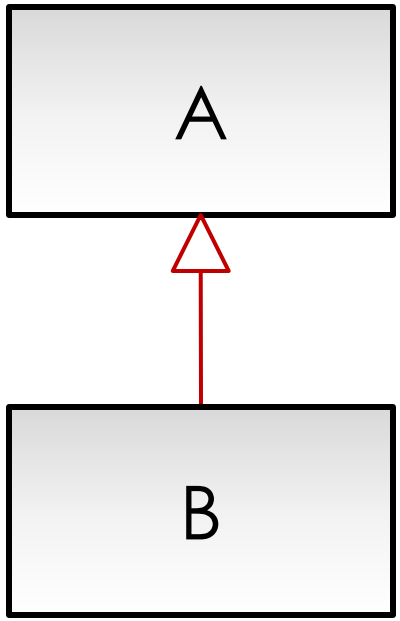
10:00		
10:15	**Андрей Совцов** Все в ваших руках: быть готовым к изменениям в	**Андрей Бибичев** Архитектура в Agile: переосмысляя идею модульности и компонентности Знакомо ли вам: - Как нам выполнять работу итерациями, если у нас реализация одной фичи занимает месяц-два? - Декомпозируйте! - Да мы
10:30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

1. Классы и их взаимодействие

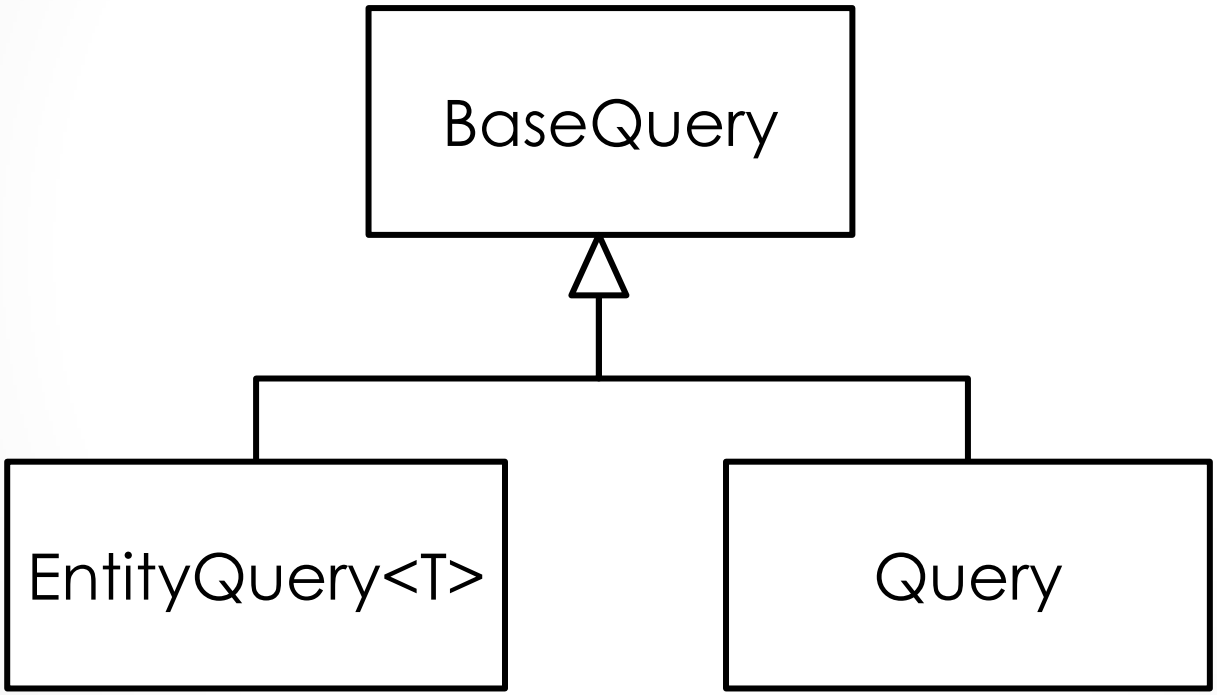
...

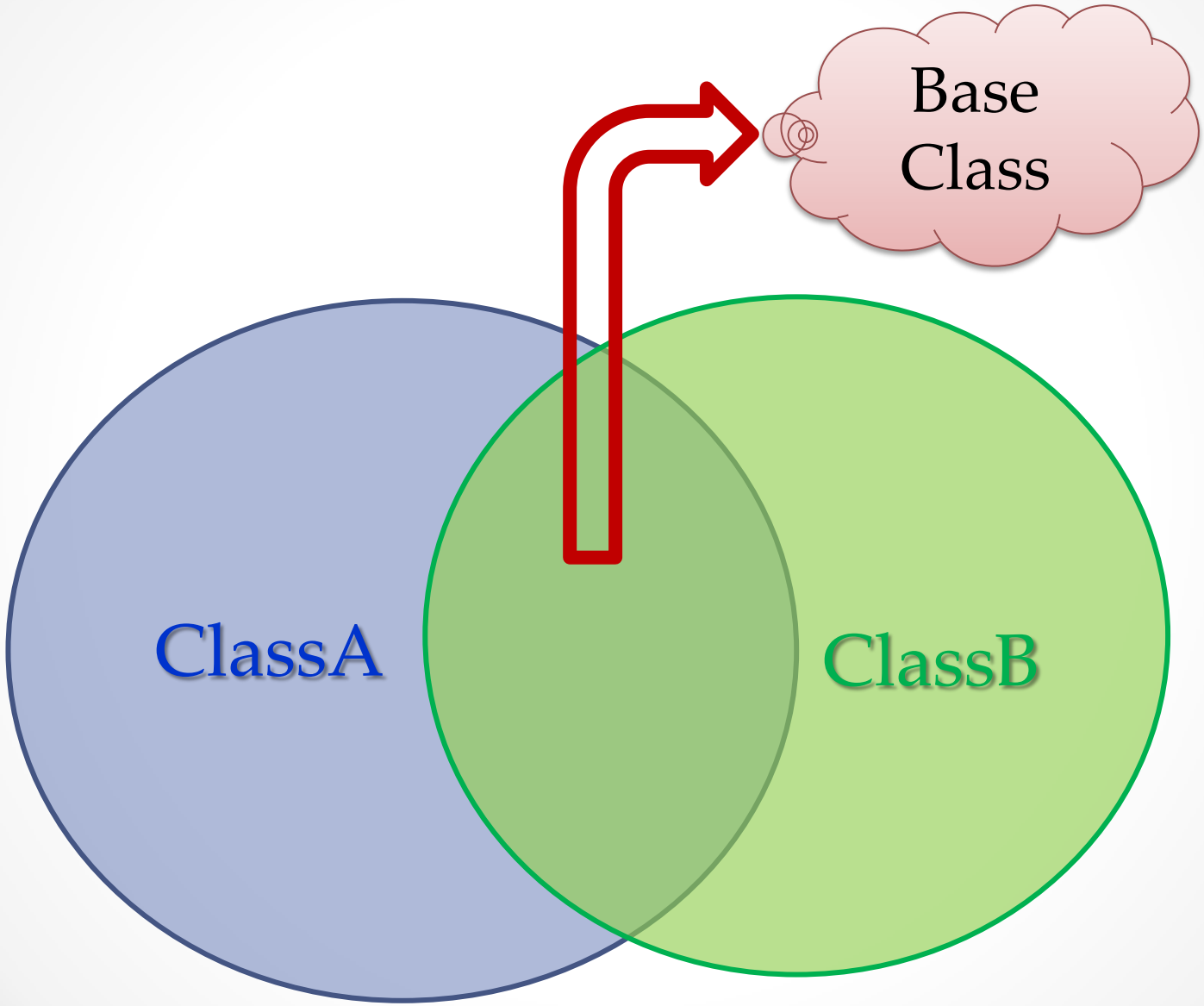
1.3. Про наследование

Наследование – тоже вид зависимости, причем опасный



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



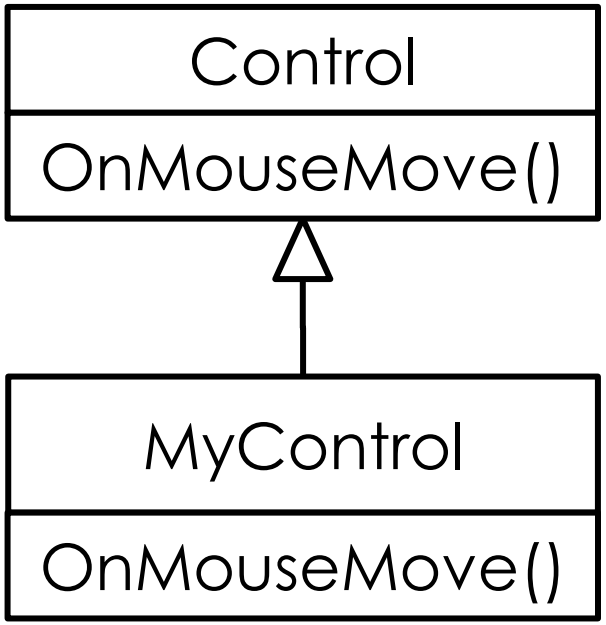

Base
Class
ClassA
ClassB

```
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

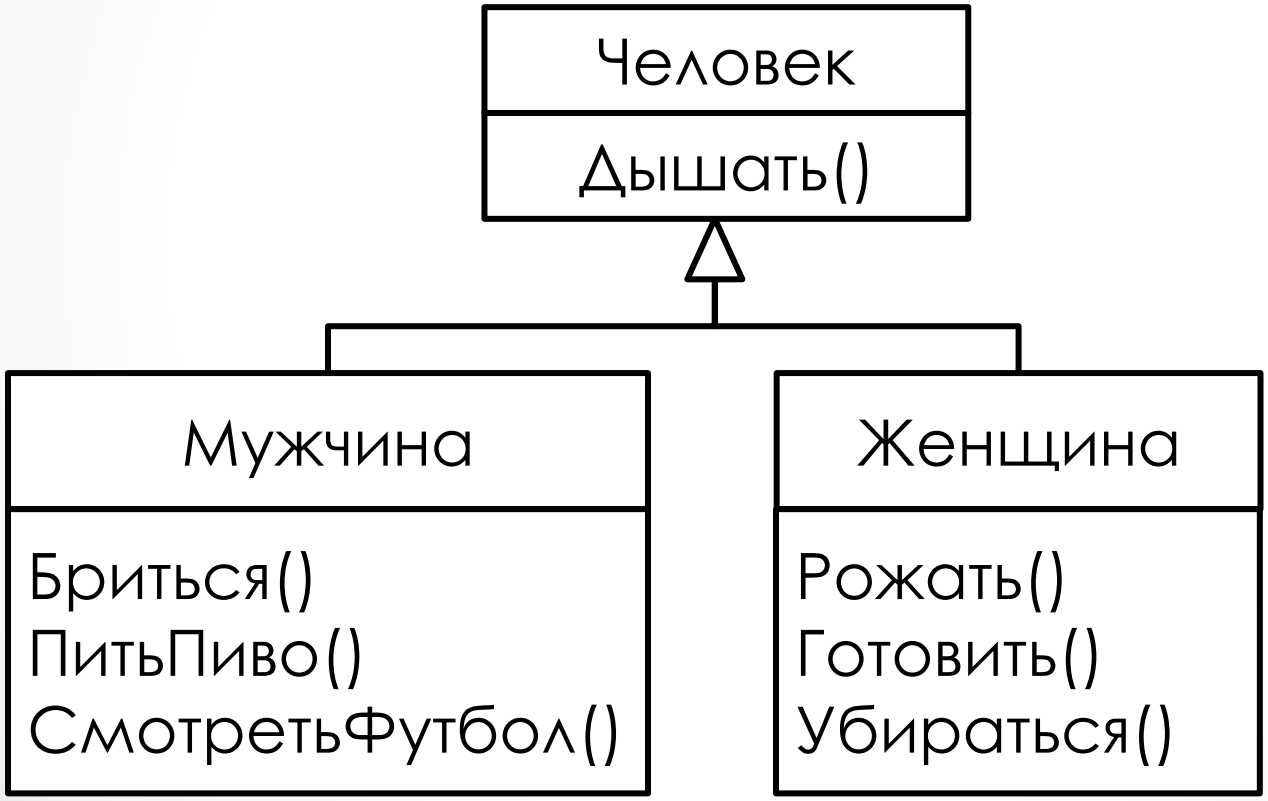


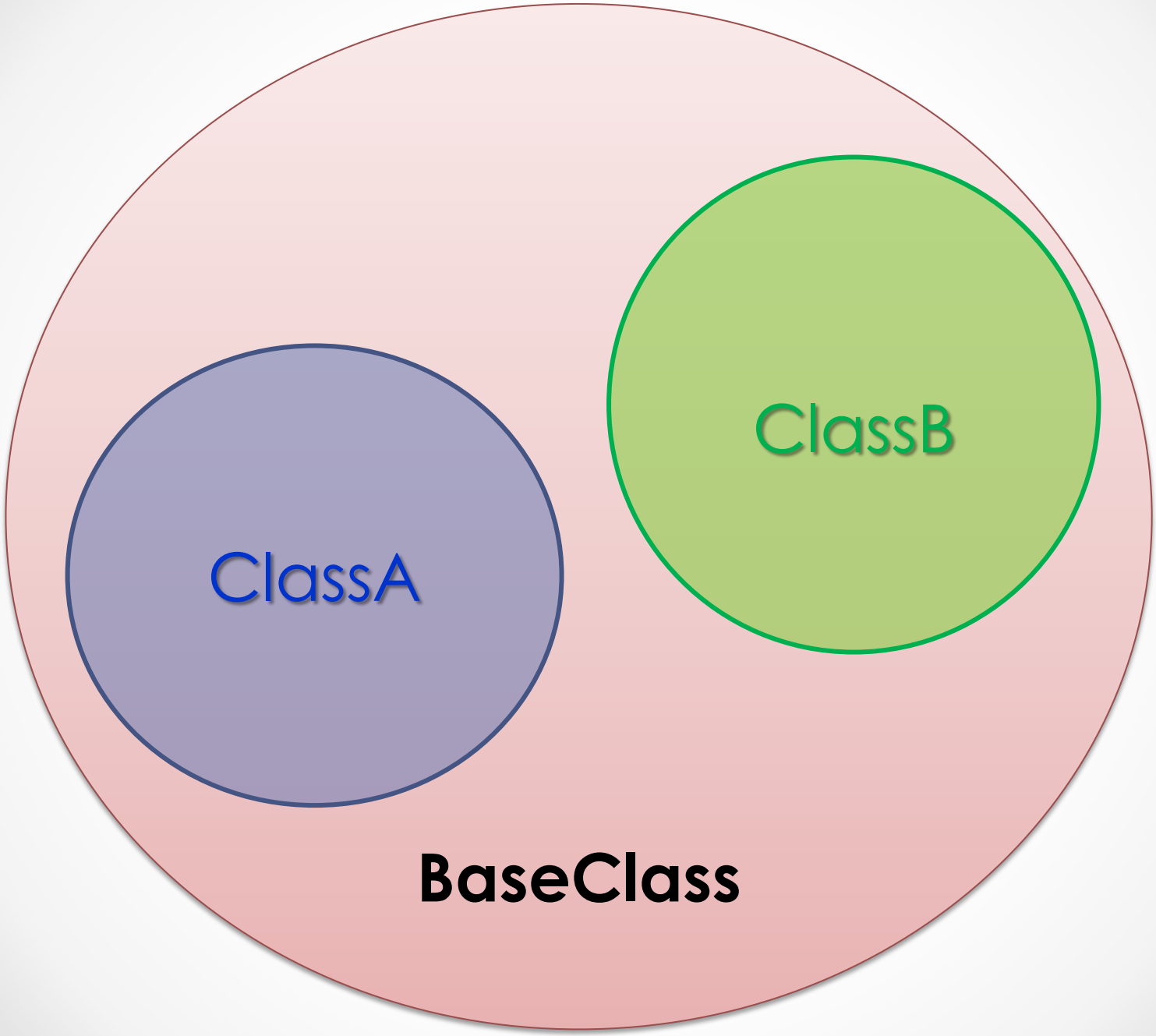

Умный и мощный класс («на все случаи жизни»)
Его исполь-зование

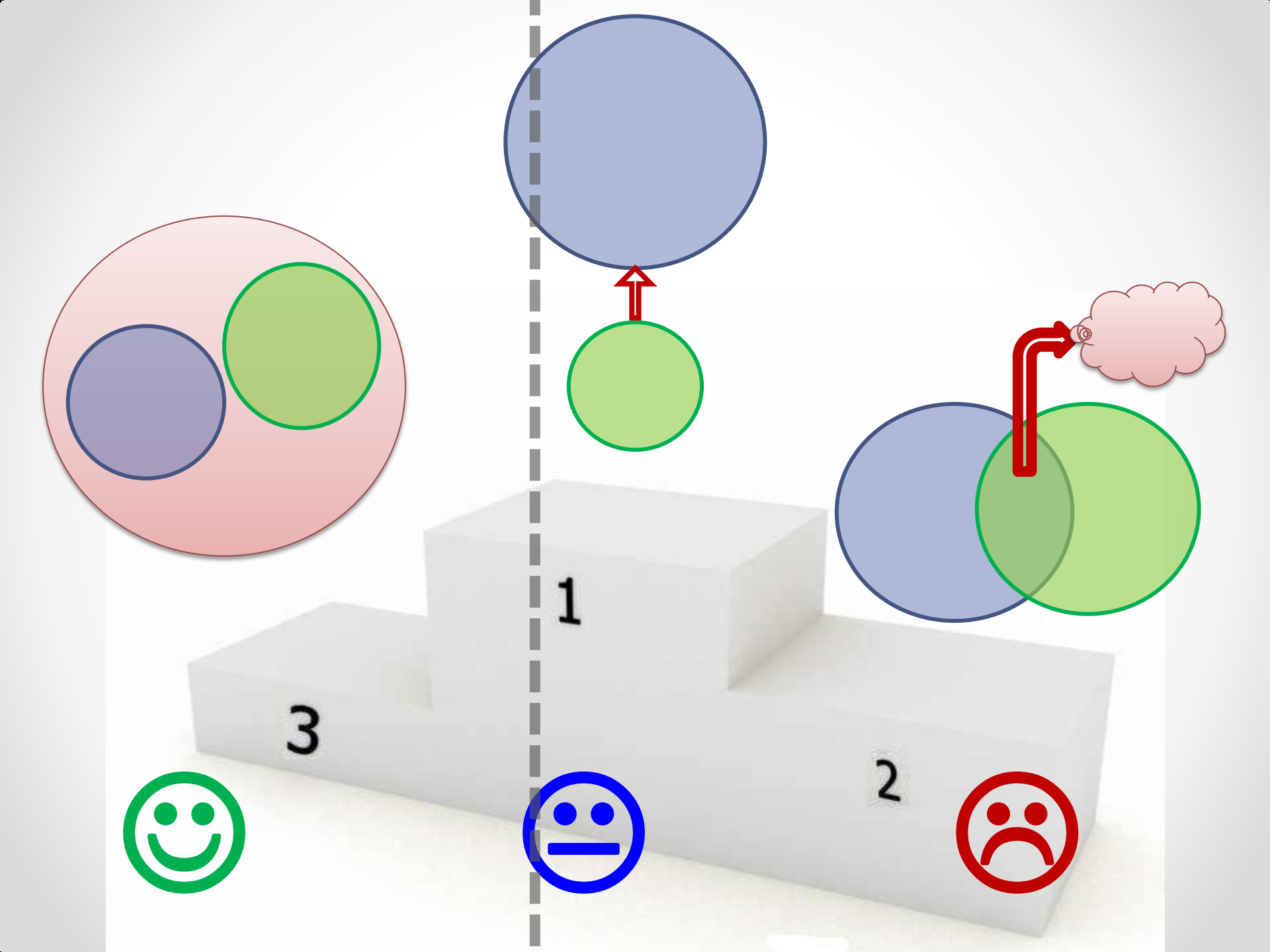
Характерные признаки

- 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

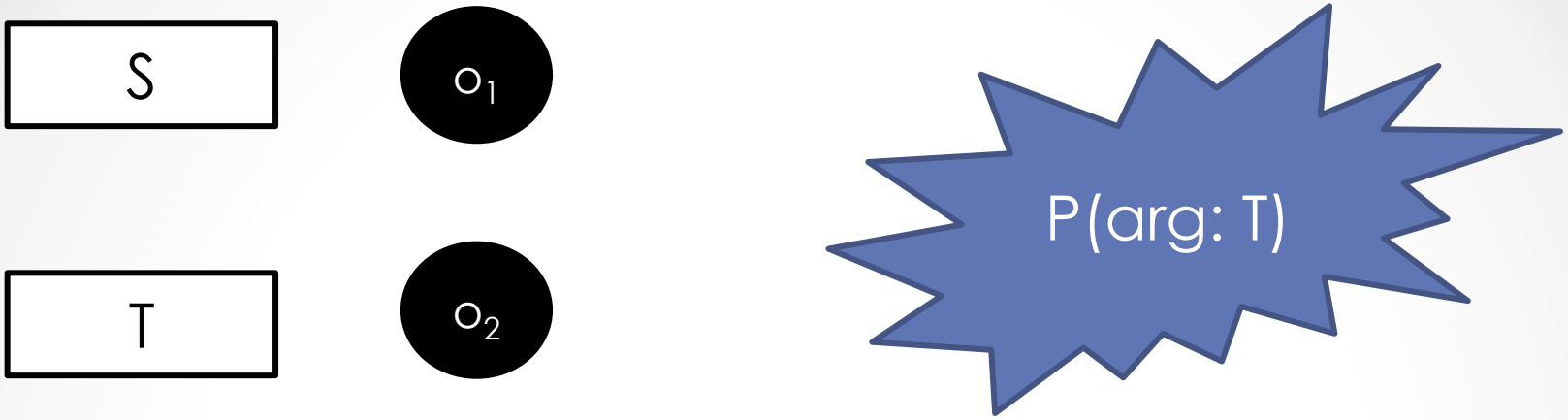

1
3
2

The Liskov Substitution Principle (LSP)

If for each object o_1 *of type* ***S*** *there is an object* o_2 *of type* ***T*** *such that for all programs* ***P*** *defined in terms of* ***T****, the behavior of* ***P*** *is unchanged when* o_1 *is substituted for* o_2 *then* ***S*** *is a subtype of* ***T****.*

1988

GIVEN

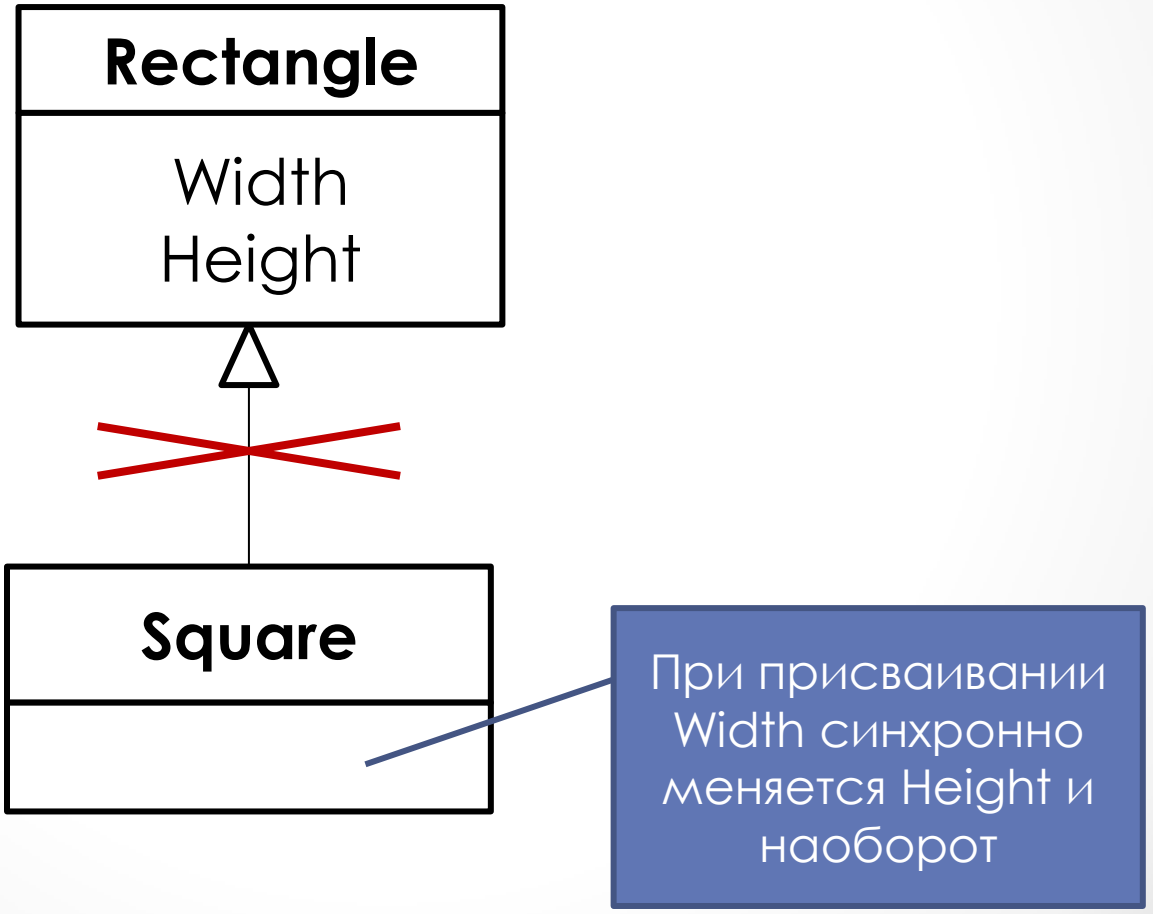


WHEN

$$\forall \boldsymbol{o_1} \quad \exists \boldsymbol{o_2} : \forall \boldsymbol{P} \text{ ResultOf}[\boldsymbol{P}(\boldsymbol{o}_1)] \equiv \text{ResultOf}[\boldsymbol{P}(\boldsymbol{o}_2)]$$

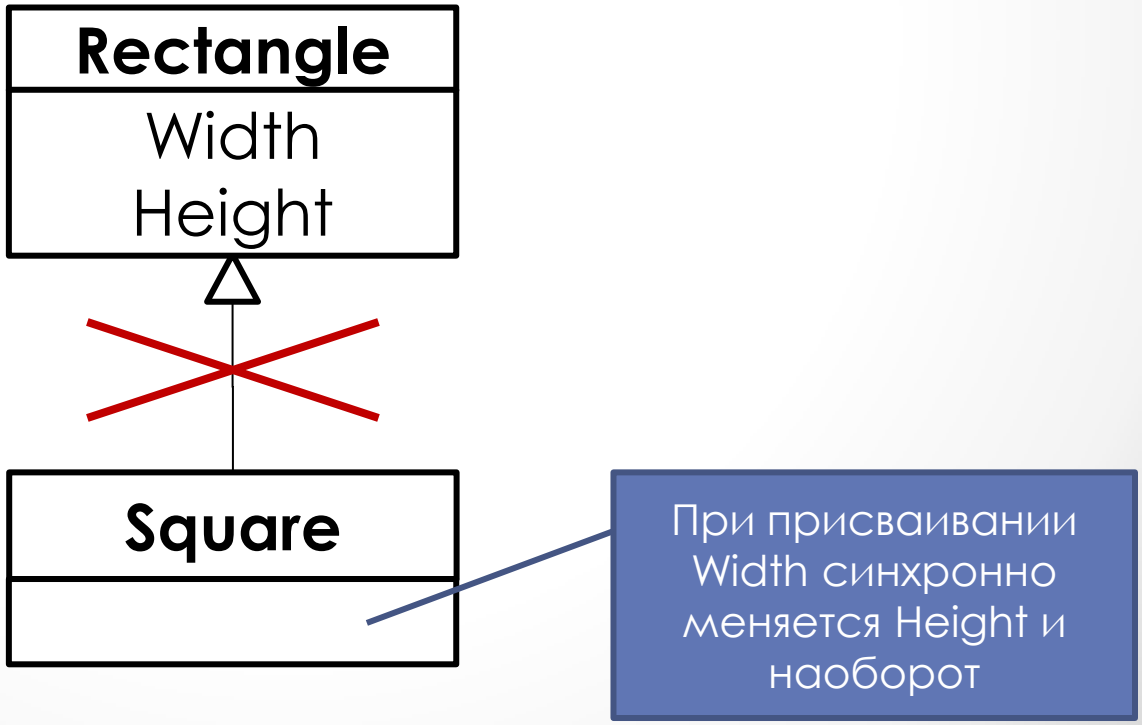
THEN

Это проясняет отношение «является» (is)



Design-by-Contract

- Контракт базового класса не должен нарушаться наследниками



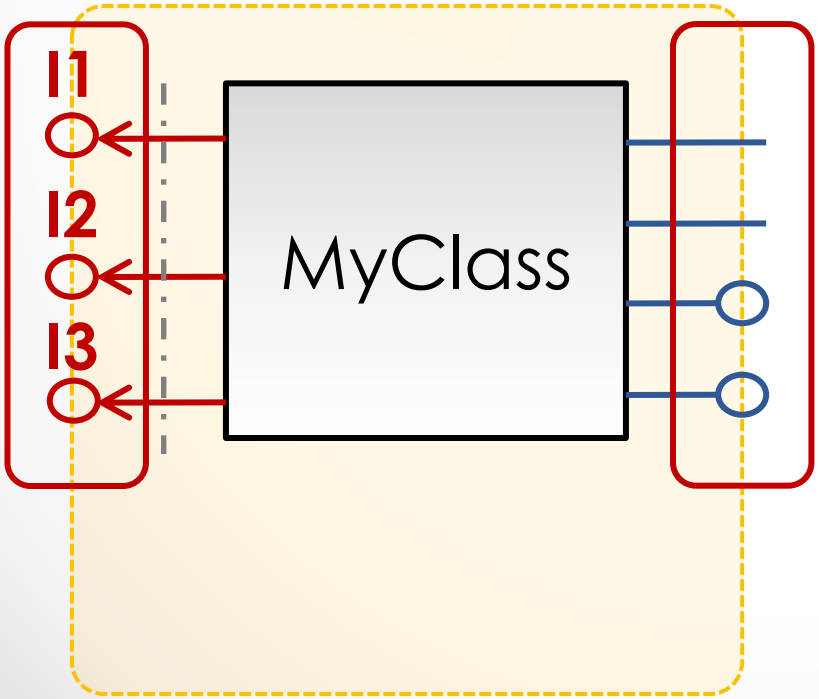
Как быть?

- 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 предоставляемое)

Три кита DbC:

- Пред-условия

  Пример: `w >= 0`

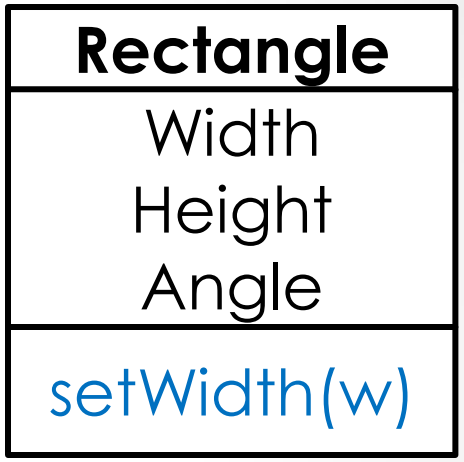
- Пост-условия

  Пример: `Width == w`

  `Height == OldValue(Height)`

- Инварианты

  Пример: `Angle == 90`



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 приходиться платить:

Лазанья-код (aka пахлава-код)

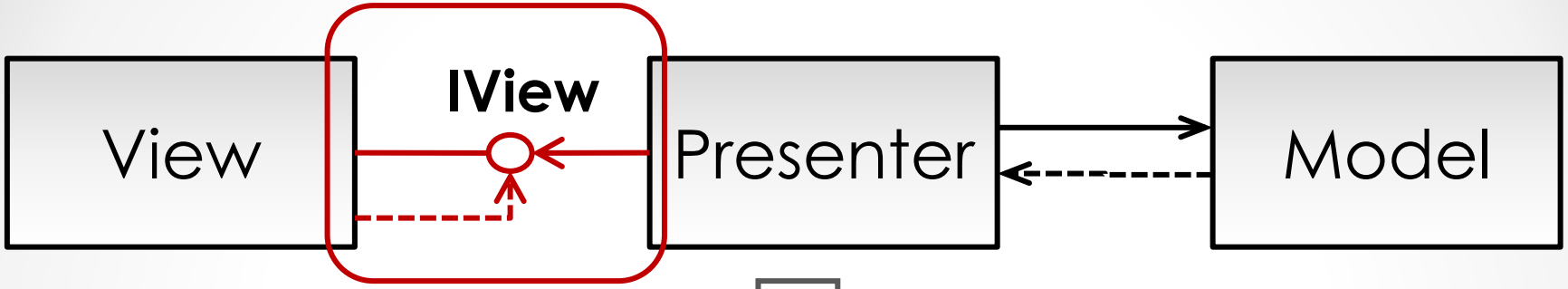
Добавление атрибута сущности:

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 ☺

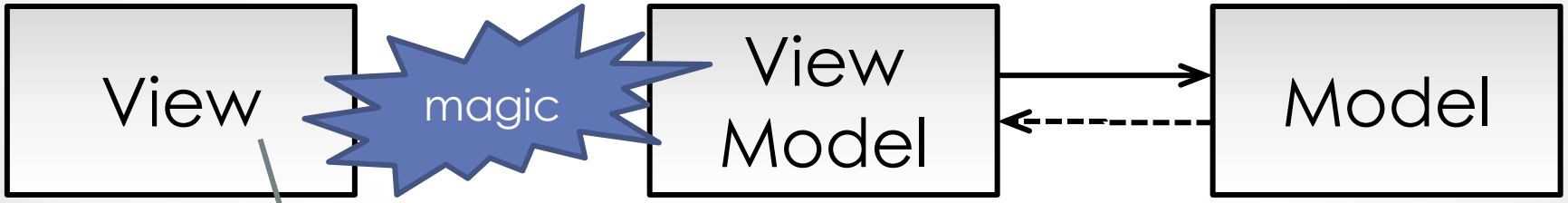
Как выкручиваться?

- 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		
10:15	**Андрей Совцов** Все в ваших руках: быть готовым к изменениям в	**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
10:30		
10:45	**Дмитрий Никонов** Планирование релизов в методологиях быстрой разработки (Agile) Казалось бы структура рилизов в командах	
11:00		
11:15		
11:30	Кофе-брейк	
11:45		
12:00	**Станислав Калканов** В чем счастье заказчика? Готовые фичи вместо гант чарта!	**Антон Бевзюк** Архитектура для автоматизированного тестирование UI
12:15		
12:30		
12:45	**Сергей Евтушенко** Agile in Investment Banking	**Николай Гребнев** Domain Driven Design в условиях разработки распределенных приложений
13:00		

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

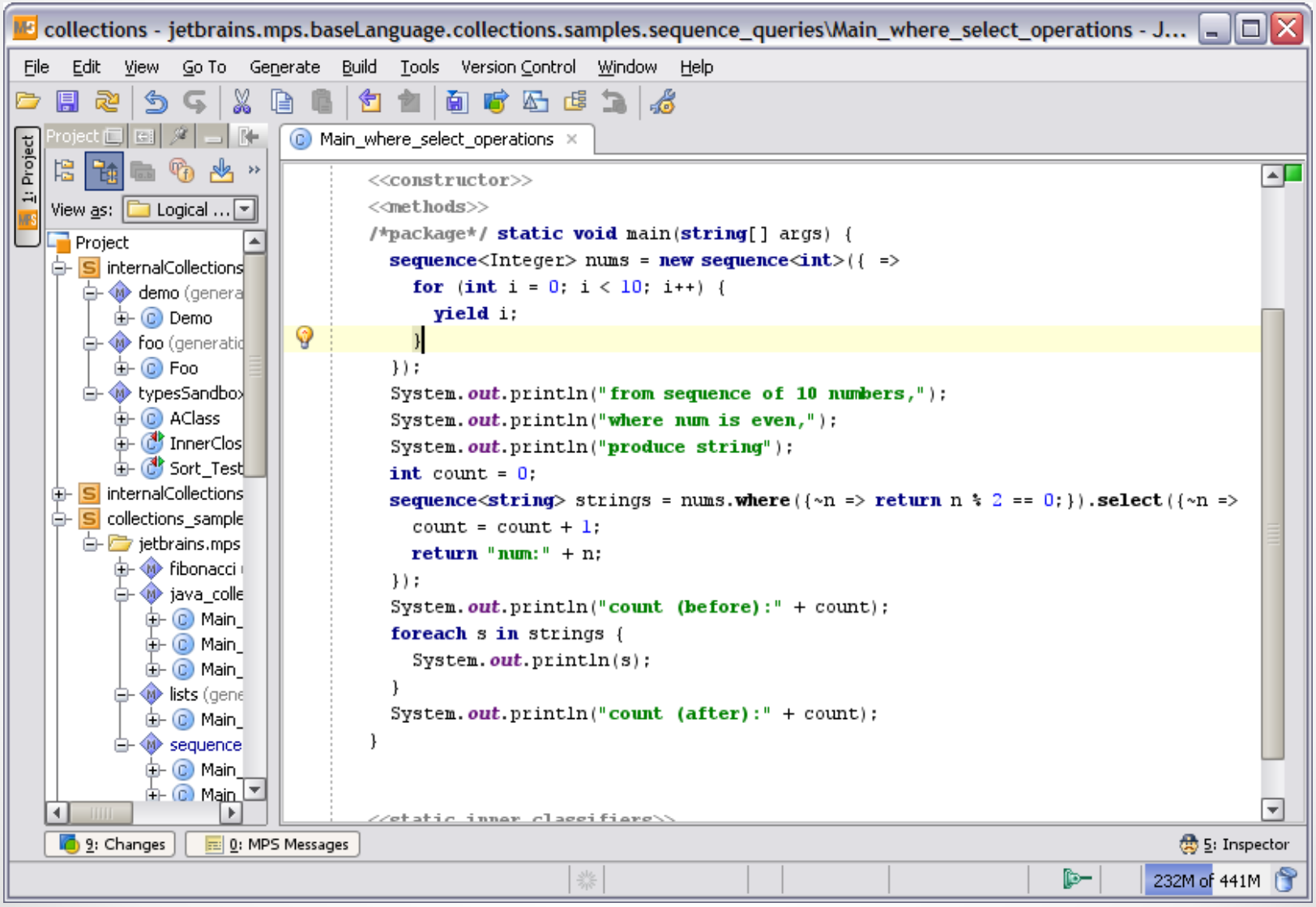
Способы генерации:

- 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

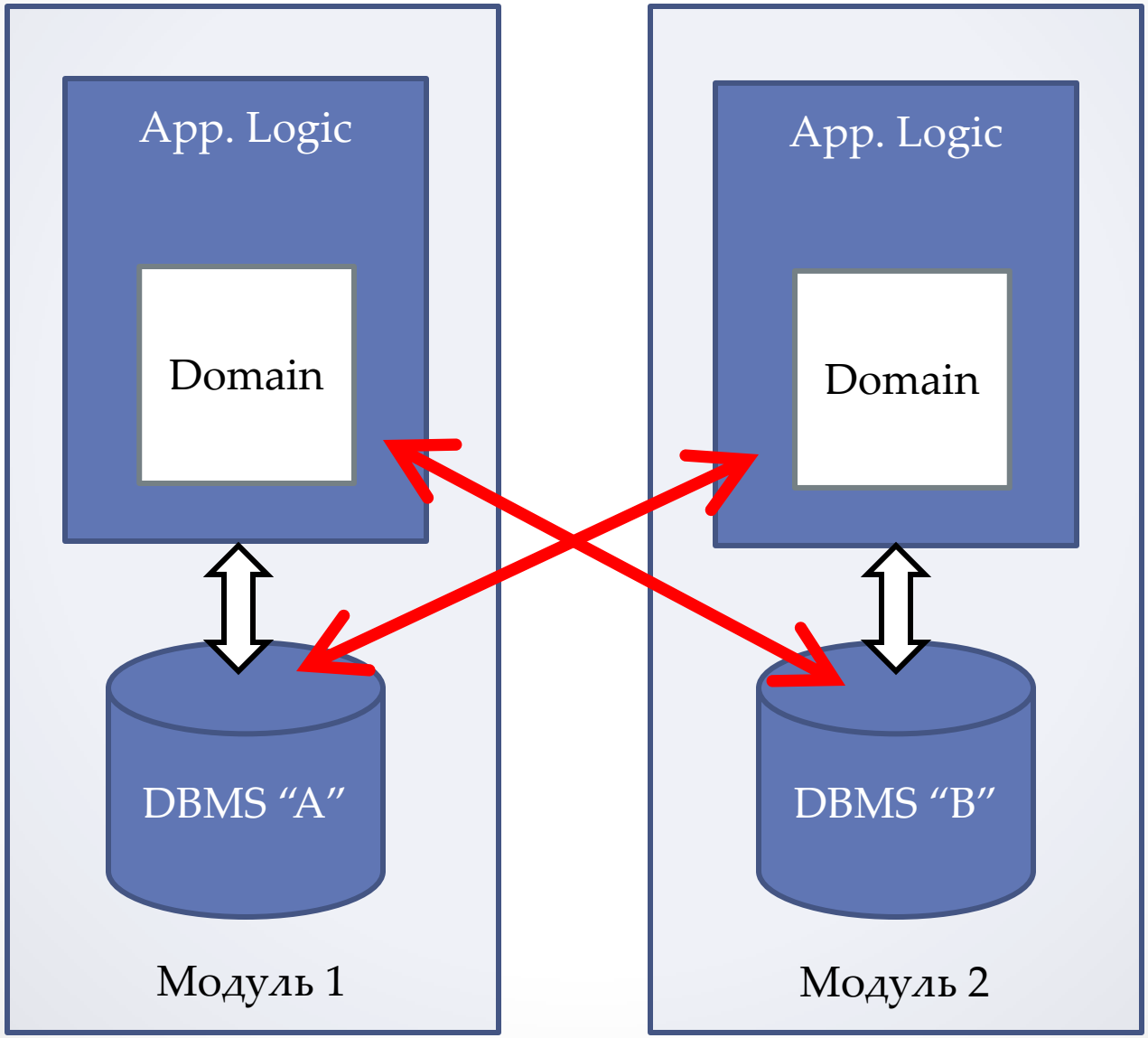


Пока больше распространены:

- Java: Groovy
- .Net: Boo

3. Взаимодействие модулей

Проще всего так:



Ущербность такого подхода поняли давно и используют **сервисы**

и всё это загажено маркетинговым булшитом ☹

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
  - Желающим 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**

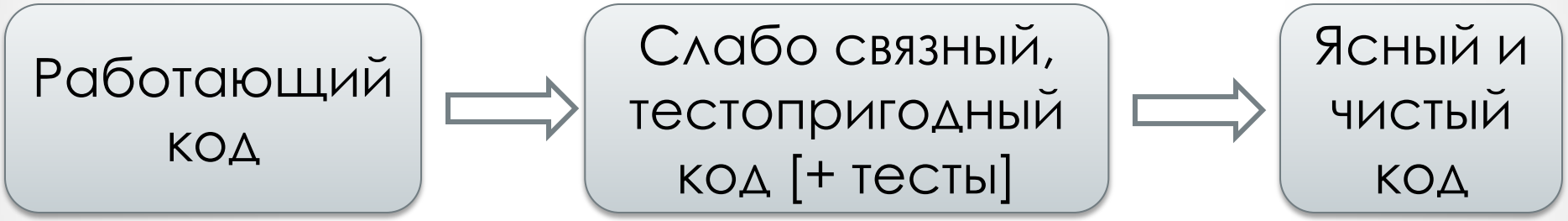
код, который легко сопровождать и развивать, писать сложнее и дольше, чем просто работающий код

макет / мини-тула

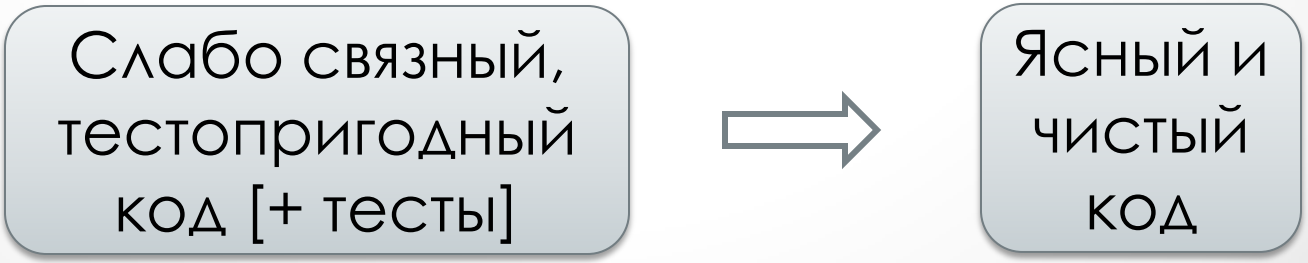
что-то большое

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