



Онлайн-образование

Проверить, идет ли запись!



Меня хорошо видно && слышно?

Ставьте +, если все хорошо

Напишите в чат, если есть проблемы

Архитектура кода



Дмитрий Кириллов

Технический директор

1С-Старт

@esteps_kirillov

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем в Slack #php-2022-01 или #general



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара

01. Чистая архитектура



02. Принципы SOLID



—



—

Цели вебинара | После занятия вы сможете

1

Проектировать приложения

2

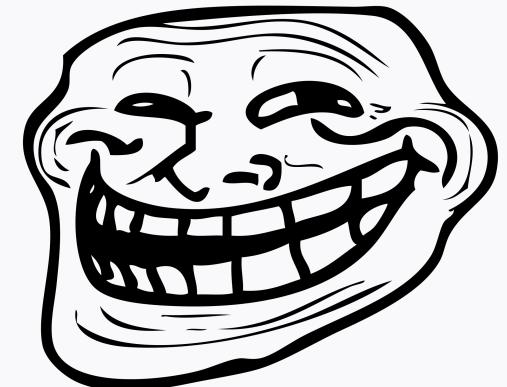
Использовать многослойную
архитектуру

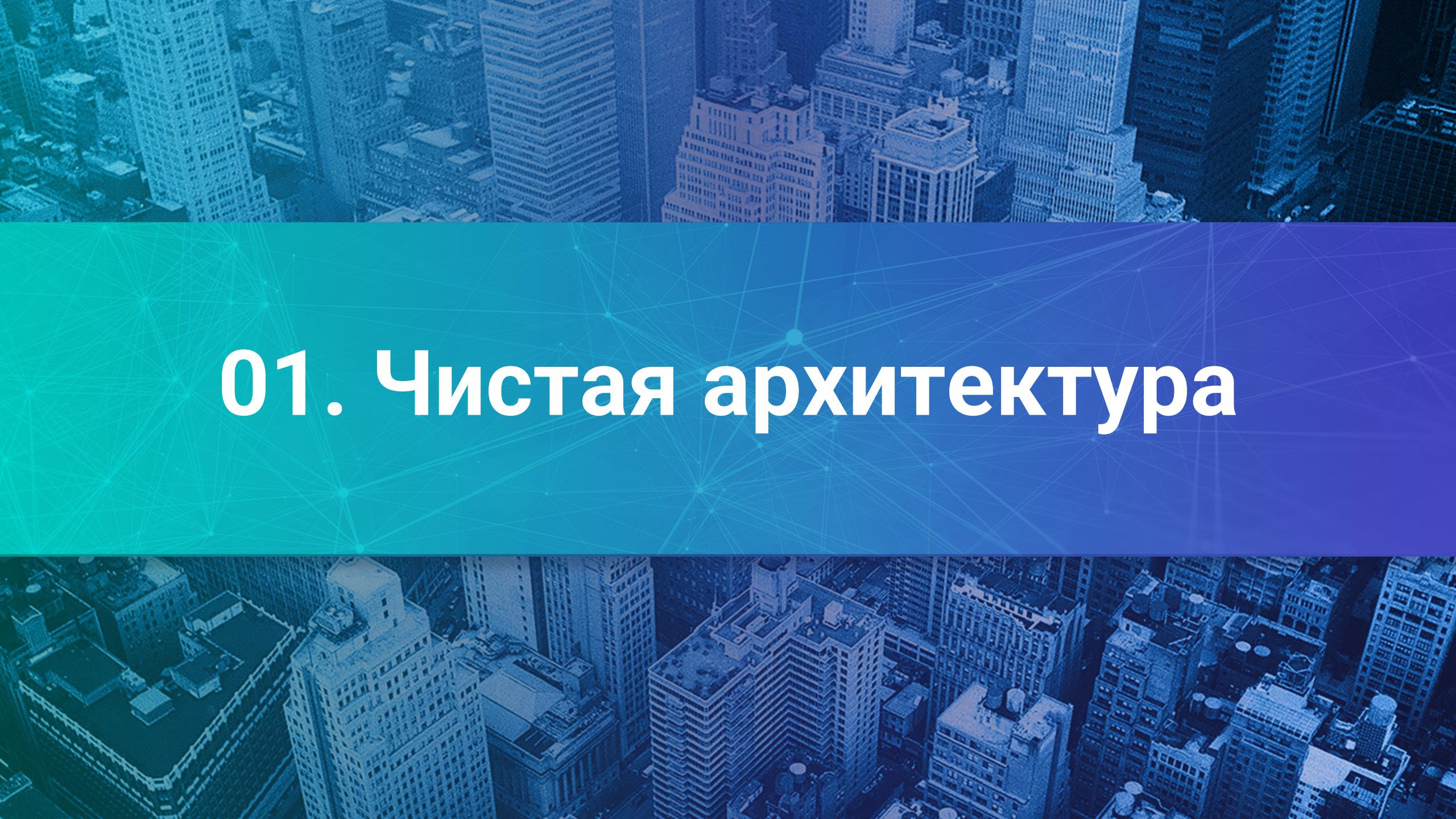
3

Опираться на принципы SOLID
при разработке кода

Мнение «иксперда»

*Архитектура ни на что не влияет!
У меня в проекте всего один файл
на 40,000 строк,
зато я уже купил себе Maserati)*





01. Чистая архитектура

Классификация компонентов

Описание	Класс
Принимает и обрабатывает HTTP-запросы	Controller
Описывает ключевые объекты приложения	Model / Entity
Возвращает сущности/коллекции из БД	Repository
Обрабатывает запросы из консоли (CLI)	Command
Выполняет операции бизнес-логики	Service

Устойчивость к изменениям

- программный продукт со временем будет изменяться
- эти изменения неизбежны
- поэтому хороший код **изначально** проектируется готовым к изменениям

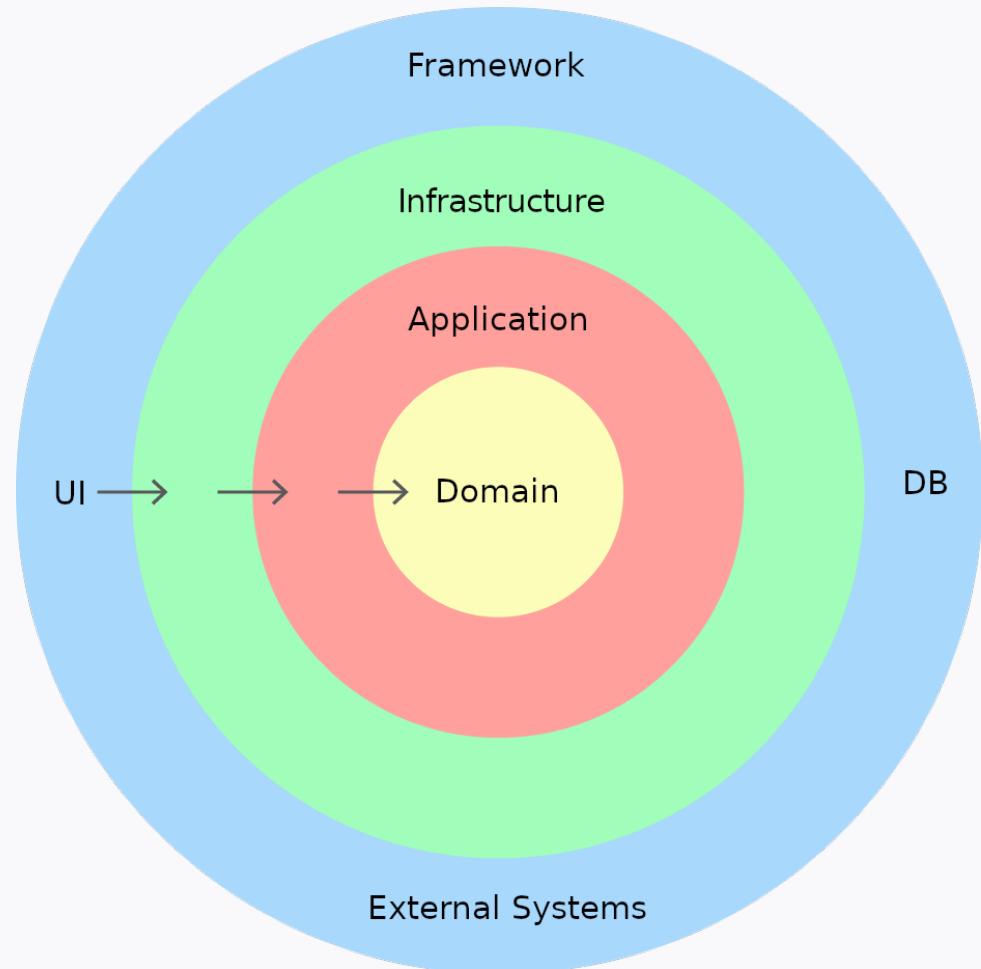
Изменчивость компонентов

Компонент	Правим/добавляем
Controller	Часто
Model / Entity	Редко
Repository	Часто
Command	Часто (для CLI-приложений)
Service	Средне

Чистая архитектура

Чистая архитектура

- приложение разделяется на уровни
- уровень **инфраструктуры** работает с внешним миром (Web, БД...)
- уровень **приложения** умеет решать пользовательские сценарии
- уровень **домена** описывает предметную область и умеет решать сценарии бизнеса



Уровень инфраструктуры (Infrastructure)

- отвечает на вопрос
"Как приложение взаимодействует с внешним миром"
- здесь хранится вспомогательный код
для организации ввода-вывода
- одни классы получают внешние запросы и проектируют их
внутрь приложения (HTTP, CLI...)
- другие классы получают данные из приложения
и отправляют их во внешний мир (API, БД...)

Уровень приложения (Application)

- отвечает на вопрос "Как именно работает наше приложение"
- здесь хранятся инструкции, которые выполняются в ответ на какие-то события или действия
- этому уровню безразлично, откуда именно пришло событие (HTTP, CLI, очередь...)

Пример:

- на уровень приложения поступил массив "id товара -> кол-во"
- нам нужно посчитать цену, сформировать заказ и отправить SMS

Уровень домена (Domain)

- отвечает на вопрос "За что отвечает наше приложение"
- здесь хранятся все сущности и бизнес-правила
- классы этого уровня обычно проектируют так, чтобы их можно было обсуждать с бизнесом на **едином языке**
- этому уровню безразлично, как работает приложение

Пример:

- на уровень домена поступил список товаров для заказа
- нам нужно посчитать скидку по определённым правилам

Чистая архитектура

Изменчивость компонентов

Компонент	Правим/добавляем	Слой
Controller	Часто	Infrastructure
Model / Entity	Редко	Domain
Repository	Часто	Infrastructure
Command	Часто (для CLI-приложений)	Infrastructure
Service	Средне	Application

Чистая архитектура

Изменчивость компонентов

Компонент	Правим/добавляем	Слой
Controller	Часто	Infrastructure
Model / Entity	Редко	Domain
Repository	Часто	Infrastructure
Command	Часто (для CLI-приложений)	Infrastructure
Service	Средне	Application

Правило зависимости

- **снаружи** — низкоуровневый код (механизмы)
- **внутри** — код более высокого уровня (политики)
- направление зависимости — **только вовнутрь**

«Проблемы индейцев вождя не волнуют»



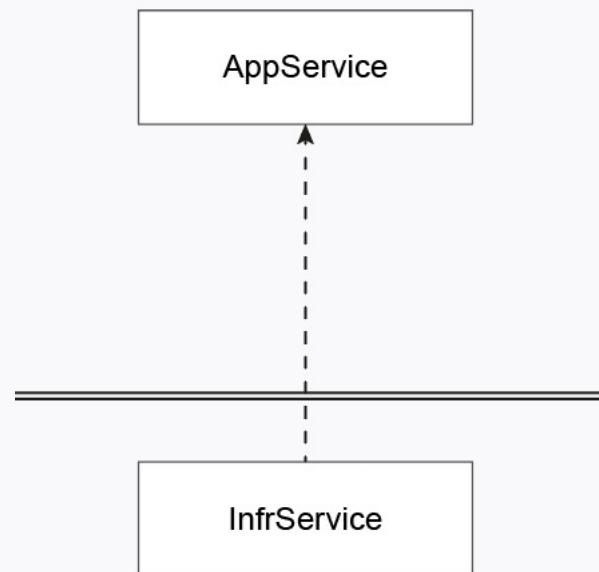
Чистая архитектура

Почему это важно

- код нижнего уровня меняется чаще, чем код верхнего уровня
- зависимость должна быть направлена в сторону устойчивости

Расшифровка UML:

- прерывистая линия с обычной (чёрной) стрелкой — зависимость
- класс, на который указывает стрелка, **ничего не знает** о наличии зависимости



Чистая архитектура

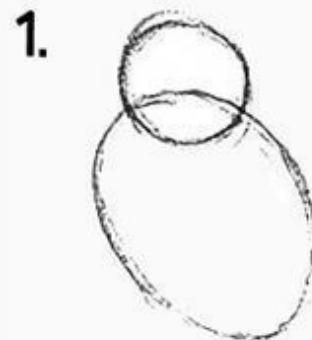
Как это реализуется в коде

Три уровня выносятся в каталоги:

- Infrastructure
- Application
- Domain

после чего дописывается
остальной код

КАК НАРИСОВАТЬ СОВУ



1.



2.

РИСУЕМ КРУЖОЧКИ

РИСУЕМ ОСТАТОК СОВЫ

Практика: разработка приложения

- 1** Мне нужно разработать приложение для отправки кредитных заявок в банк
- 2** Ваша задача: помочь мне в разработке архитектуры и кода
- 3** Критерии оценки: приложение должно соответствовать чистой архитектуре



Тайминг: 15–20 минут

Проблема: значения полей

- поля с примитивными типами могут содержать любую информацию
- эта информация может быть невалидной
- кроме того, полю по ошибке можно присвоить значение совершенно другого поля

Решение: объект-значение (Value Object)

- содержит неизменяемое (immutable) значение
- это не сущность (нет уникального идентификатора)
- но это именно объект конкретного **пользовательского типа**
(т.е. сложно ошибиться при установке значений)
- зачастую содержит охранные методы
для обеспечения корректности значения
- есть поддержка во всех современных ORM

Проблема: зависимости класса

- одни классы могут вызывать методы других классов
- мы говорим, что классы зависят от других классов
- объекты других классов нужно откуда-то взять,
и часто их создают прямо в коде

```
$outerClass = new OuterClass();  
$outerClass->doSomething();
```

- из-за этого возникает жёсткая связь с другим классом,
которая мешает разработке

Решение: внедрение зависимостей

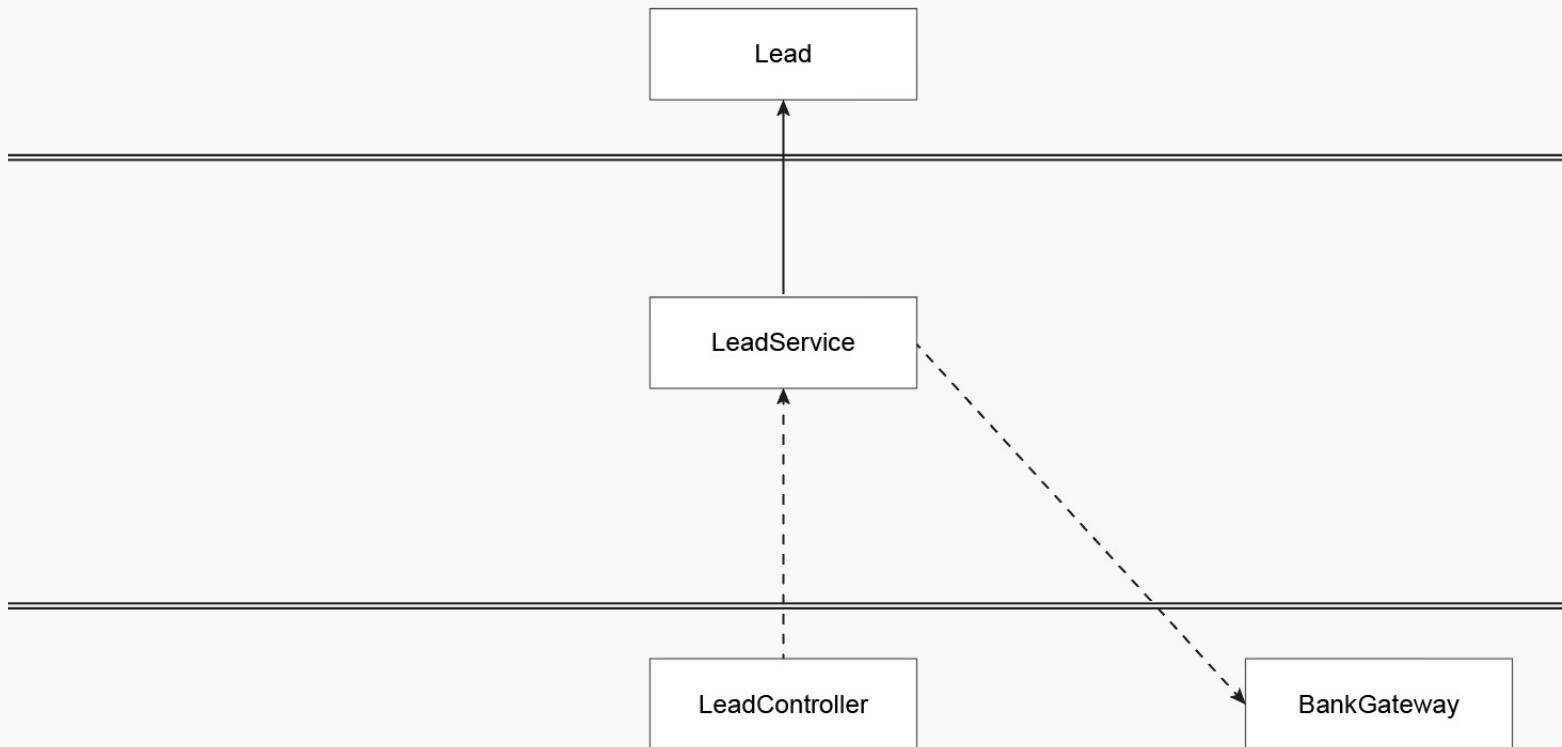
- мы больше не создаём объекты других классов
- вместо этого мы передаём их снаружи в конструктор

```
private ServiceInterface $service;  
  
public function __construct(ServiceInterface $someService) {  
    $this->service = $someService;  
}
```

- все современные фреймворки выполняют такое внедрение **автоматически**, при необходимости создавая нужные объекты

Чистая архитектура

Диаграмма классов: пилотная версия



Чистая архитектура

Правило зависимости

Слои высокого уровня **не могут** задействовать низкоуровневые компоненты:

- код (классы, методы, переменные...)
- структуры данных (сущности, DTO...)

Логичный вопрос:

- а как тогда обращаться к БД или к внешним API?

Чистая архитектура

Начальник МФЦ



Менеджер



Клиент МФЦ



Архив документов



Проблема: передача данных между слоями

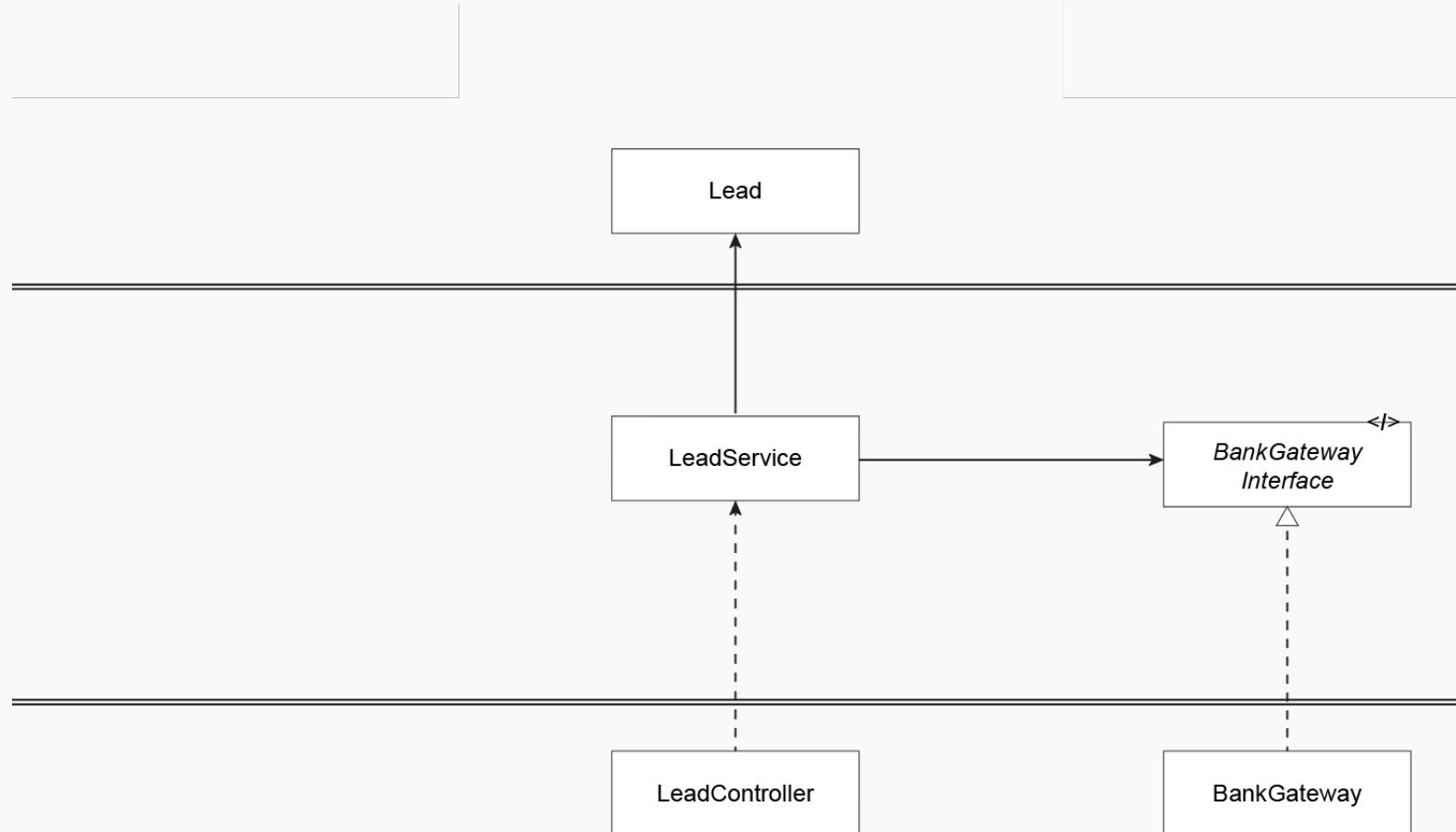
- нам часто нужно передавать данные между слоями
- этот процесс может быть двунаправленным
- использовать для этого сущности невыгодно,
т.к. в разных случаях нужен разный объем информации
- создавать для этого объекты-значения бессмысленно,
т.к. валидировать данные в момент передачи не требуется

Решение: DTO (Domain Transfer Object)

- это абсолютно "тупые" объекты
- их задача — быть "курьером" между разными слоями
- их код иногда сводится к набору публичных полей
- для чистой архитектуры важно, чтобы код с классами DTO лежал в том слое, который находится выше

Чистая архитектура

Диаграмма классов: выделение интерфейса



Чистая архитектура

Вопрос на засыпку:

Какой известный принцип мы только что избрали?

02. Принципы SOLID

Принцип инверсии зависимостей

Модули верхнего уровня не должны импортировать компоненты из модулей нижнего уровня

Оба типа модулей должны зависеть от абстракций

Абстракции не должны зависеть от деталей.

Детали должны зависеть от абстракций

Принцип инверсии зависимостей

- слои постоянно взаимодействуют друг с другом
- взаимодействие может идти снизу вверх (Web)
или сверху вниз (БД)
- в любом случае оно должно происходить
на условиях (и в терминах) **верхнего** слоя

Принцип инверсии зависимостей

Рецепт счастья:

- решаем, какие действия нужны коду верхнего слоя
- создаём интерфейс в этом же слое
- пишем класс, производный от интерфейса,
на нижнем слое

Принцип инверсии зависимостей

Что нам это даёт:

- мы избегаем зависимости от компонентов, которые часто меняются
- код становится проще тестировать
- имена методов соответствуют предметной области вышестоящего слоя
- на верхние слои не «протекают» ненужные им концепции

Новая задача от бизнеса

- нам нужны 2 вида лидов: на кредит и на страхование
- нам нужно научиться искать лид в БД по его ID
- помимо обычных полей, каждый вид лида должен уметь возвращать доп.информацию

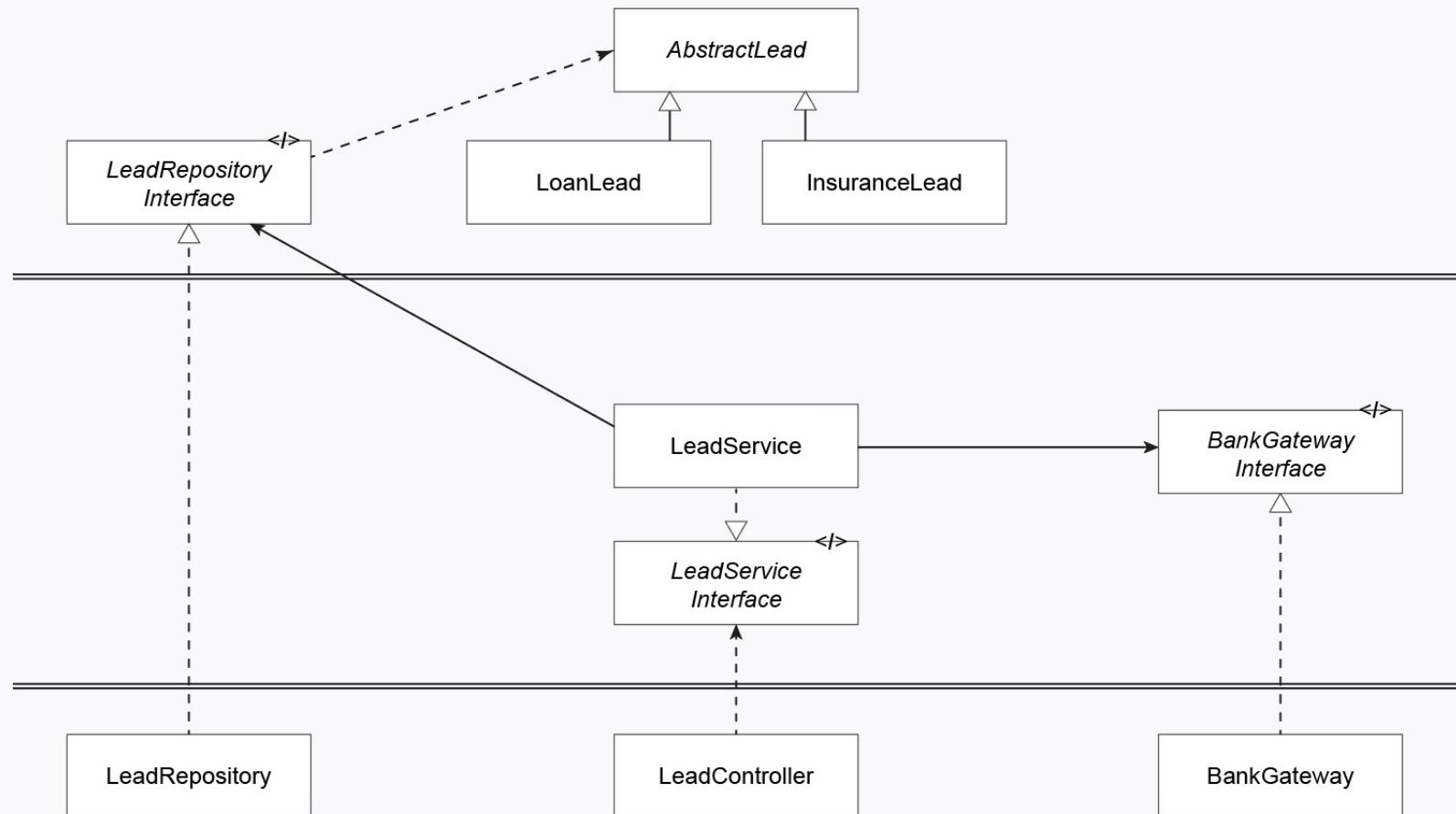
Примеры:

- Заявка на кредит, клиент Иванов
- Заявка на страхование, телефон 9051234567

Что нам потребуется?

- новые классы лидов
- репозиторий лидеров
- новый метод в сервисе
- новый метод в контроллере

Диаграмма классов: репозиторий



Устойчивые классы

- от них зависят другие классы
- обычно это код высокого уровня
- неустойчивые классы проще изменять;
устойчивые — сложнее
- зависимости должны быть направлены
в сторону устойчивости

А как тогда вносить правки в устойчивые классы?

В нашем арсенале есть:

- абстракции
- шаблоны проектирования

Вариант 1. Создаём абстрактный класс

Особенности:

- мы готовим жёсткий каркас для конкретных классов
- шаблон проектирования — Template Method
- выбирать этот вариант нужно только в том случае, если классы связаны отношением наследования

Вариант 2. Создаём интерфейс

Особенности:

- это более гибкое, но и более сложное решение
- шаблоны проектирования – Strategy, Visitor...
- потребуется разработать дополнительные классы, которые будут внедряться в производный класс

Принцип открытости/закрытости

Программные сущности должны быть открыты для расширения, но закрыты для изменения

Принцип открытости/закрытости

- позволяет писать программы, которые потом легко расширять
- мы не вносим при этом никаких правок в старый код
- расширение — только за счёт нового кода

Принцип открытости/закрытости

Рецепт счастья:

- заранее прикидываем, какие методы класса могут изменяться в будущем
- объявляем эти методы абстрактными, используя абстрактный класс или интерфейс
- реализуем эти методы в производном классе
- в клиентском коде везде упоминаем только абстрактный класс или интерфейс

Принцип открытости/закрытости

Что нам это даёт:

- **код закрыт для изменений**: все правки мы вносим в производных классах
- **код открыт для расширения**: мы добавляем новый функционал, создавая или изменяя производные классы
- мы можем в любой момент подменить одну реализацию абстрактного класса / интерфейса на другую

Новая задача от бизнеса

— нам нужно реализовать поиск лида из командной строки

Пример:

```
# php bin/console app:lead:find 12
```

Test

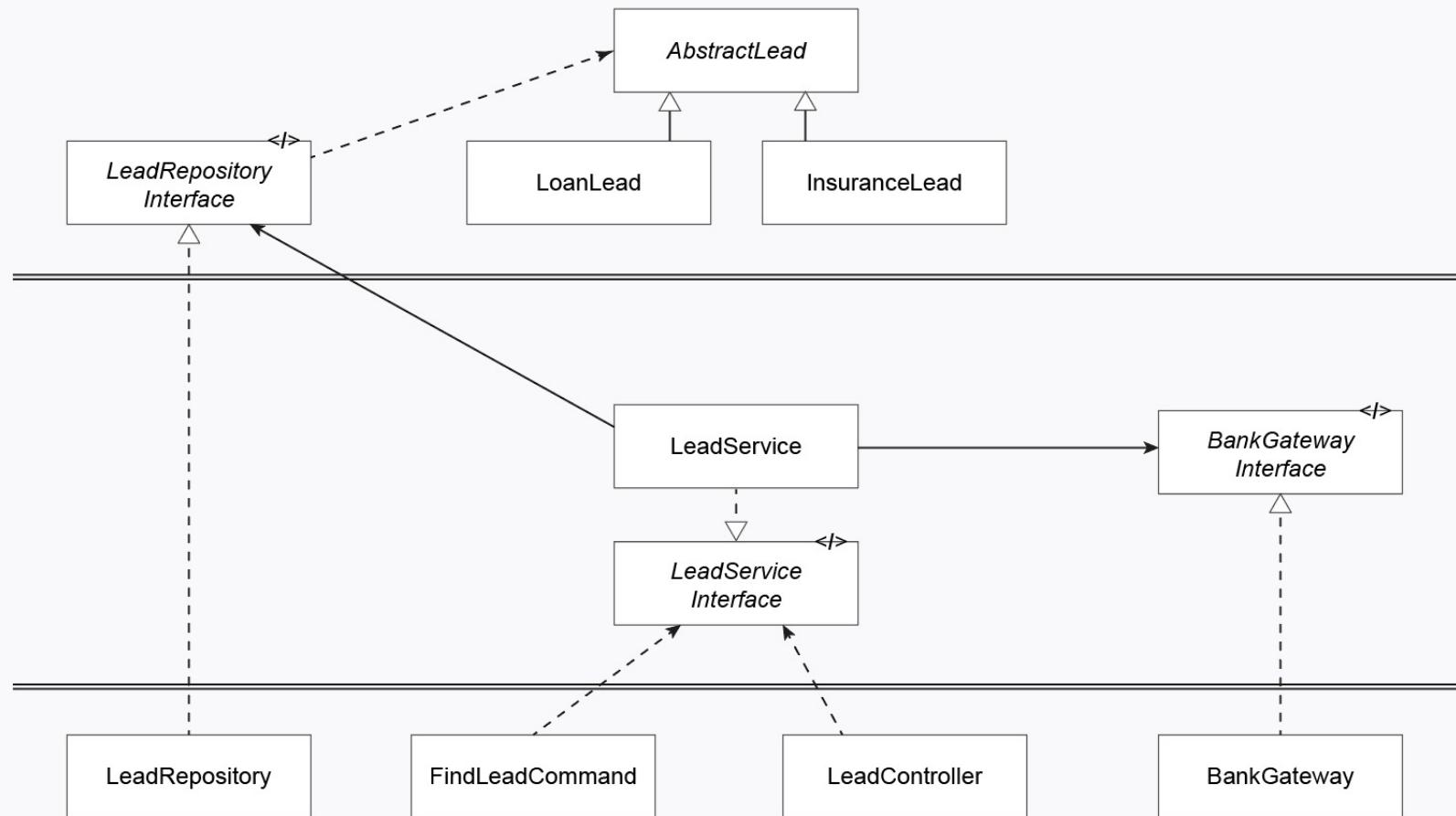
9051234567

Заявка на кредит, клиент Test

Что нам потребуется?

- просто разработать команду)
- мы переиспользуем существующие сервисы и DTO

Диаграмма классов: команда CLI



Вопросы для обсуждения:

- у LeadServiceInterface есть несколько клиентов
- но одному из этих клиентов нужны **только** те методы, которые связаны с поиском лидов
- получается, что клиент "вхолостую" получает доступ к ненужным методам
- как можно решить эту проблему?

Вариант 1. Выносим общий интерфейс

Проблемы:

- конкретным клиентам этого интерфейса не нужны все его методы
- приходится реализовывать «пустые» методы без операций

Вариант 2. Разбиваем интерфейс на части

Проблемы:

- конкретным клиентам этого интерфейса не нужны все его методы
- приходится реализовывать «пустые» методы без операций

Принцип разделения интерфейсов

Программные сущности не должны зависеть от методов, которые они не используют

Принцип разделения интерфейсов

- интерфейс говорит о том, что умеет делать класс
- иногда эти умения нужны не всем клиентам класса
- несколько маленьких интерфейсов лучше, чем один большой:
правки интерфейса у одного клиента не затронут других

Принцип разделения интерфейсов

Рецепт счастья:

- определяем, какие методы нужны конкретному клиенту класса
- выносим их в отдельный интерфейс
- класс в итоге будет реализовывать
несколько интерфейсов

Принцип разделения интерфейсов

Что нам это даёт:

- каждый клиент использует только те методы, которые ему нужны
- большой класс можно будет со временем разделить на два независимых класса, и это **никак** не затронет код верхнего уровня

Вопросы для обсуждения:

- наш сервис зависит от двух разных интерфейсов
- одновременно он использует только один из них
(публичные методы никак не связаны друг с другом)
- получается, что сервис отвечает
за несколько разных задач
- как можно решить эту проблему?

Логичное решение:

- сделать несколько классов, каждый из которых будет отвечать за что-то одно

Принцип единственной ответственности

*Программная сущность должна иметь
одну и только одну причину для изменения*

*Программная сущность должна отвечать
за одного и только за одного актора*

Принцип единственной ответственности

- это самый субъективный принцип
- он менялся несколько раз
- многие разработчики понимают его буквально и создают классы по принципу «1 класс – 1 метод – 1 строка кода»

Если в классе нужно скомбинировать методы:

- можно делегировать их другим классам (композиция)
- можно импортировать их с помощью трейтов
- шаблон проектирования – Facade

Принцип единственной ответственности

Рецепт счастья:

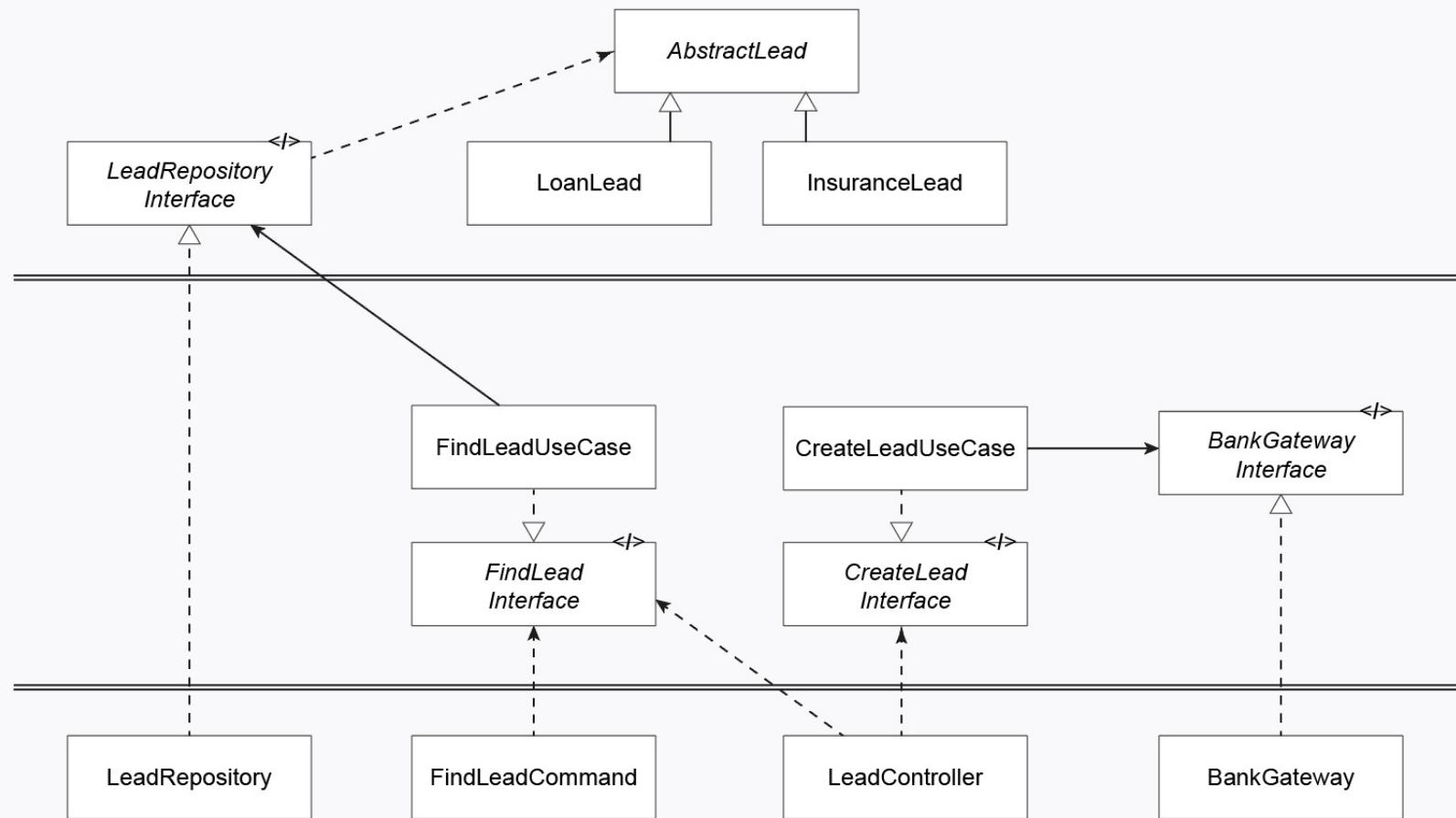
- смотрим, какие компоненты класса (свойства / методы) тесно связаны между собой
- связь может быть как технической (внутри класса), так и логической (один внешний клиент)
- если таких групп несколько — выделяем их в классы
- при необходимости разделяем интерфейсы

Принцип единственной ответственности

Что нам это даёт:

- разработчики не мешают друг другу, внося независимые правки
- становится проще найти нужный класс в структуре проекта
- можно подменить реализацию конкретного интерфейса, не внося масштабные правки в код

Диаграмма классов: разделение классов



Новая задача от бизнеса

- при выводе описания LoanLead
нужно посчитать отношение всех символов в имени
к цифровым символам

Пример:

```
# php bin/console app:lead:find 12
```

```
Test  
9051234567
```

Заявка на кредит, клиент Test2, средняя длина 0.2

Что нам потребуется?

- просто внести правки в класс LoanLead)

А насколько безопасно подменять реализации?

Что может пойти не так:

- мы ожидаем, что производные классы ничего не поломают в нашем приложении
- большинство языков программирования не в состоянии автоматически это проверить
- в итоге ошибка в проектировании может привести к падению всего приложения

Вариант 1. Проверяем конкретный тип

Проблемы:

- вместо абстракции мы вынуждены импортировать конкретный производный класс
- это автоматически нарушает принцип открытости/закрытости

Вариант 2. Дорабатываем код конкретного типа

Проблемы:

- вместо абстракции мы вынуждены импортировать конкретный производный класс
- это автоматически нарушает принцип открытости/закрытости

Принцип подстановки Барбары Лисков

Если существует тип данных Supertype и его подтип Subtype, то поведение программы, реализованной в терминах Supertype, не должно меняться, если Supertype заменить на Subtype

Принцип подстановки Барбары Лисков

Рецепт счастья:

- берём методы подтипа (Subtype) и проверяем типы аргументов + возвращаемого значения*
- внимательно смотрим: может ли подтип что-то поломать из-за слабо продуманного кода?
- в клиентском коде везде упоминаем только супертип (Supertype)

* в большинстве случаев это автоматически проверят IDE и PHP

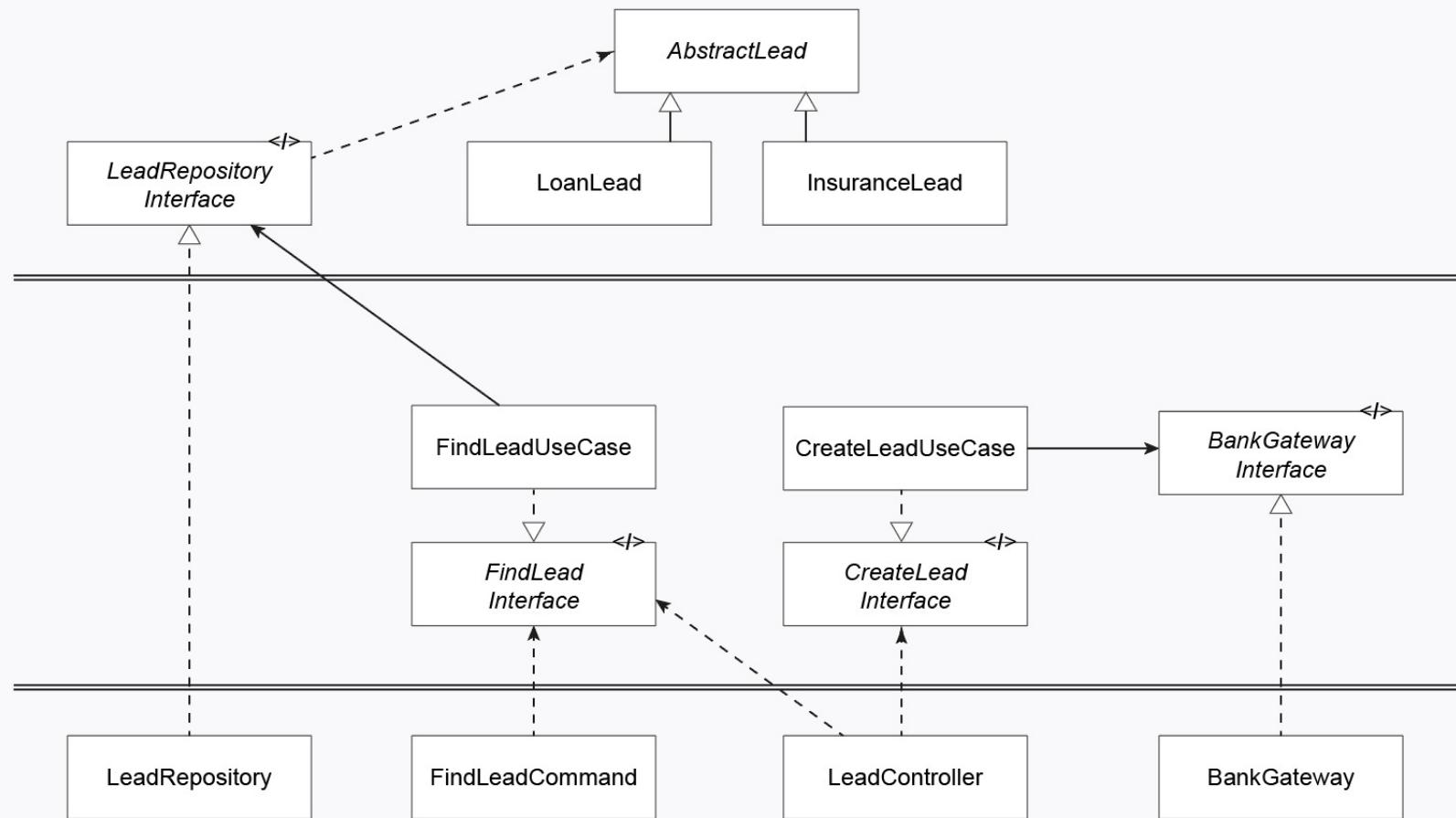
Принцип подстановки Барбары Лисков

Что нам это даёт:

- возможность расширять наш код с учётом особенностей подтипов в ООП*
- соответствие другим принципам, в первую очередь — принципу открытости/закрытости

* см. «Поведенческий полиморфизм подтипов»

Диаграмма классов: итоговый продукт



Итоги

- принципы SOLID упрощают внесение изменений в код
- ключевой принцип — **принцип инверсии зависимостей**
- остальные принципы опираются на него
- но ещё более важный принцип в основе SOLID (и ООП) —
это полиморфизм

Цели вебинара | Проверка достижения целей

1

Проектировать приложения

2

Использовать многослойную
архитектуру

3

Опираться на принципы SOLID
при разработке кода

Заключение

Список литературы

- Р. Мартин "Гибкая разработка программ на Java и C++"
- Р. Мартин "Чистая архитектура"

Домашнее задание

- 1 Выберите один из своих проектов
- 2 Проанализируйте его архитектуру,
в т.ч. на соответствие принципам SOLID
- 3 Предложите свои варианты исправления
(можно в текстовой форме)



Срок: желательно сдать до 22.12

Следующий вебинар

Тема: Design patterns. Часть 1



Четверг, 7 апреля, в 20:00



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию
в ЛК – можно
изучать



Обязательный
материал обозначен
красной лентой



Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие вебинары



Дмитрий Кириллов

Технический директор

1С-Старт

@esteps_kirillov