Паттерны корпоративных приложений

Patterns of Enterprise Application Architecture

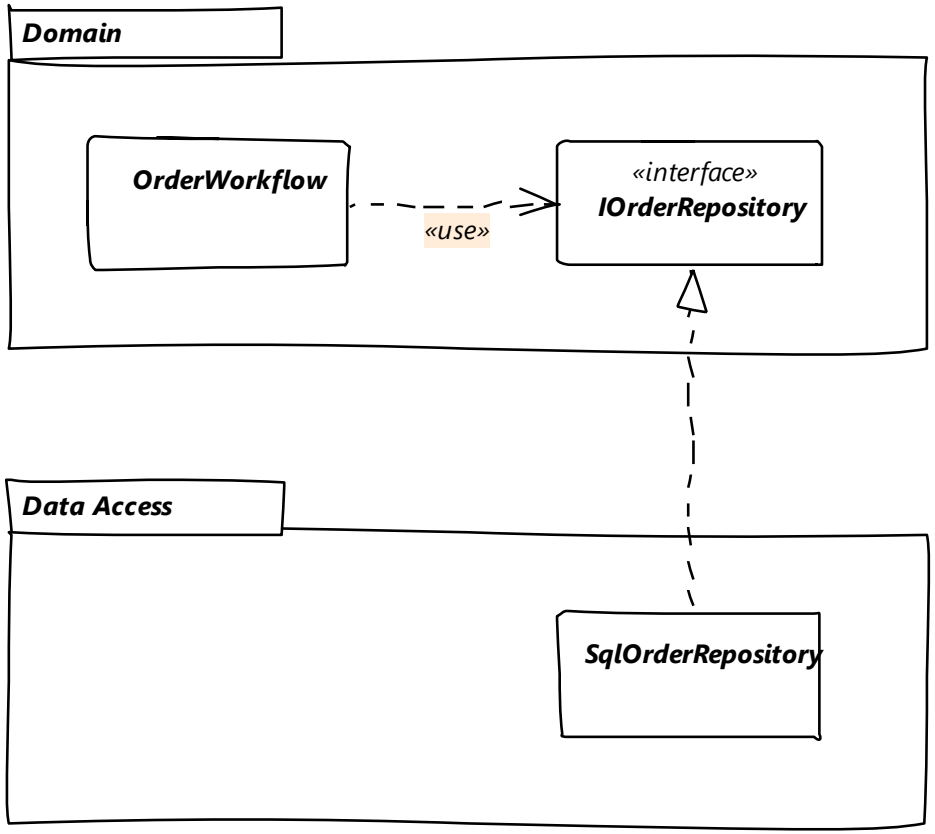
Separated Interface

Проблема:

При разработке системы можно добиться улучшение её архитектуры, уменьшая связанность между её частями. Это можно сделать, распределив классы по отдельным пакетам и контролировать зависимости этими пакетами.
Однако могут появиться транзитивные зависимости.

Решение:

Выделение какого-либо интерфейса к объекту в отдельный от объекта пакет.



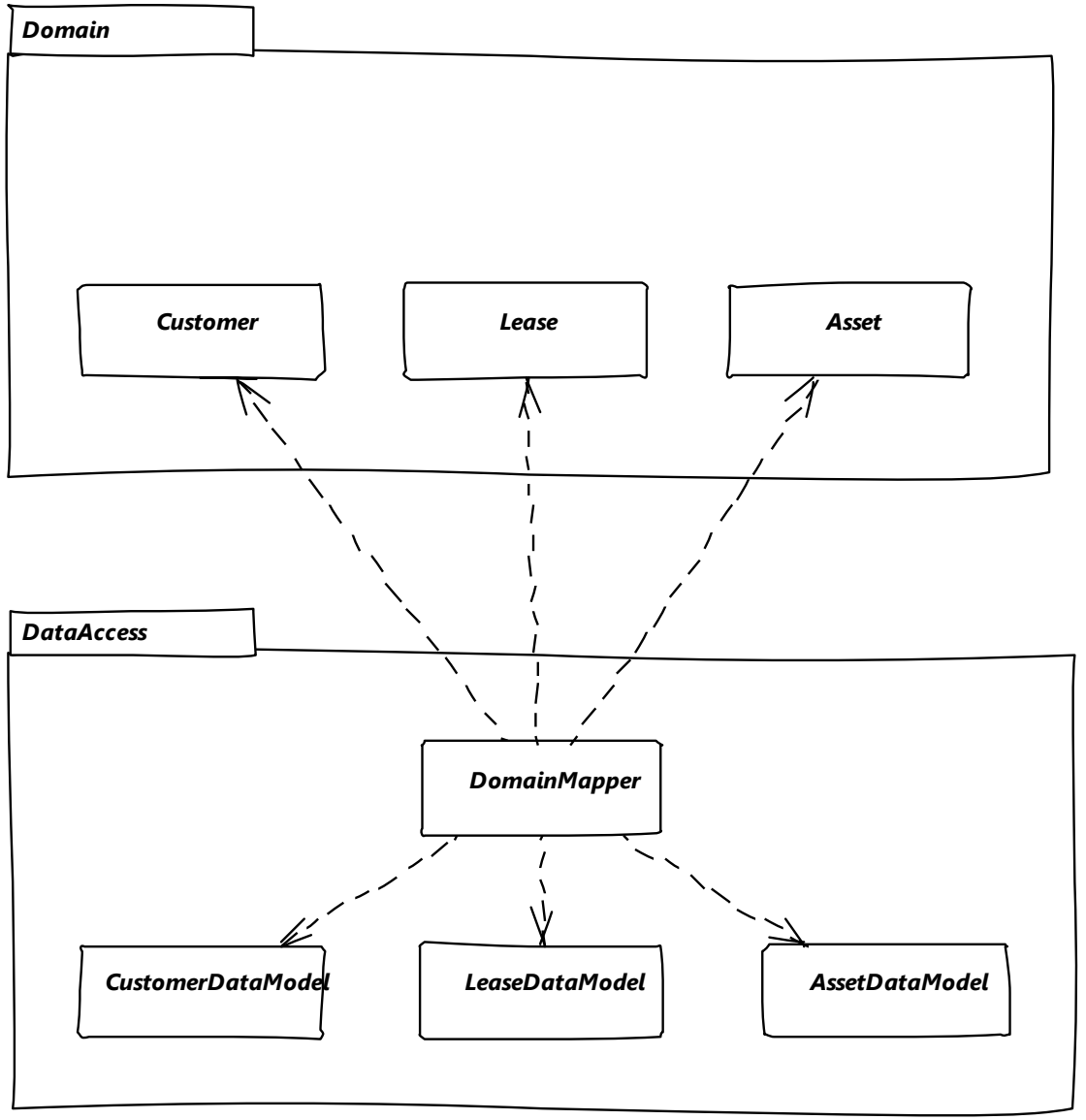
Mapper

Проблема:

Иногда нужно установить сообщение между двумя подсистемами, которые, между тем должны оставаться в неведении друг о друге. Это может быть обусловлено невозможностью изменения этих объектов, или просто нежеланием создавать зависимости между ними или между ними и изолирующей частью.

Решение:

Объект, который управляет сообщением между независимыми друг от друга объектами.



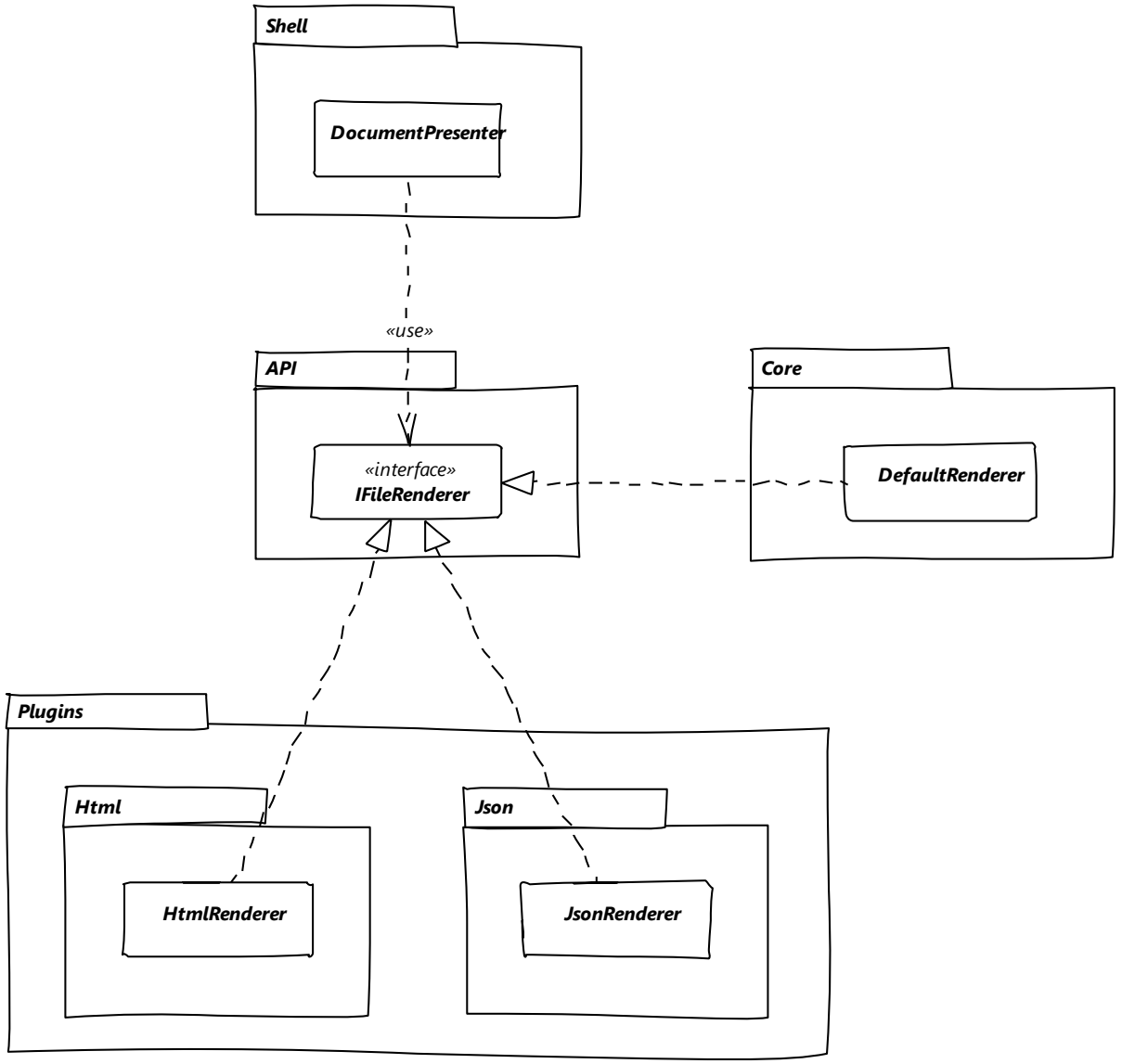
Plugin

Проблема:

Система должна расширяться гибко и без постоянных перекомпиляций.

Решение:

Соединяет классы во время конфигурации, а не компиляции.



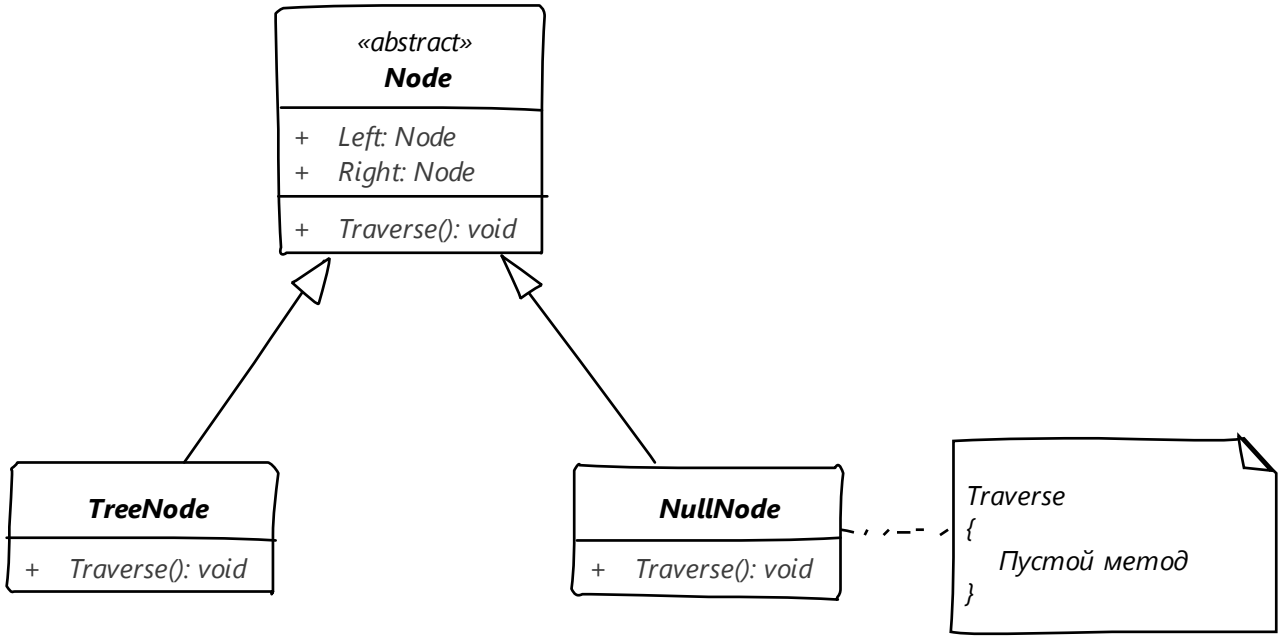
Special Case

Проблема:

Null-значения в ООП - неуклюжая вещь, так как она нарушает полиморфизм и переусложняет логику.

Решение:

Подкласс, содержащий особую логику для отдельных ситуаций.

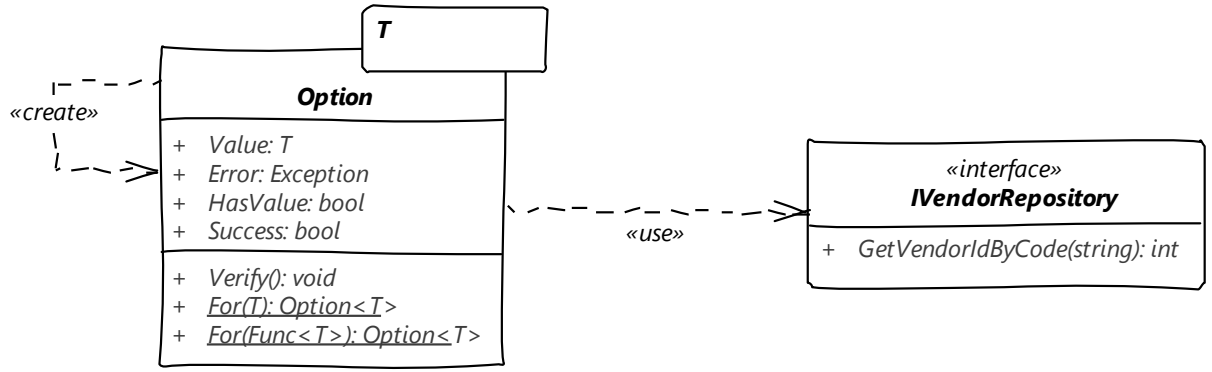


Option

Проблема:

Более универсальный случай паттерна Special Case (Null object).

Из внешнего мира приходят или не приходят данные, или данные неполные, или вообще неправильные. Логика непредсказуемо усложняется множеством проверок и ветвлений.



Решение:

Инкапсурировать результат выполнения метода в объект Result, содержащий собственно результат метода и вспомогательную информацию: флаг ошибки и исключение, флаг наличия значения.

```
// Вместо прямого вызова
var vendorId  = repository.GetVendorIdByCode(code)

// выполняется обертывание вызова в монаду
var vendorId = Option.For(() => repository.GetVendorIdByCode(code));

if (vendorId.Success)
{
    UsageOfVendor(vendorId);
}

// А при попытке прочитать неустановленное значение
// выбрасывается исключение


// Можно императивно проверить целостность:
vendorId.Verify();
```

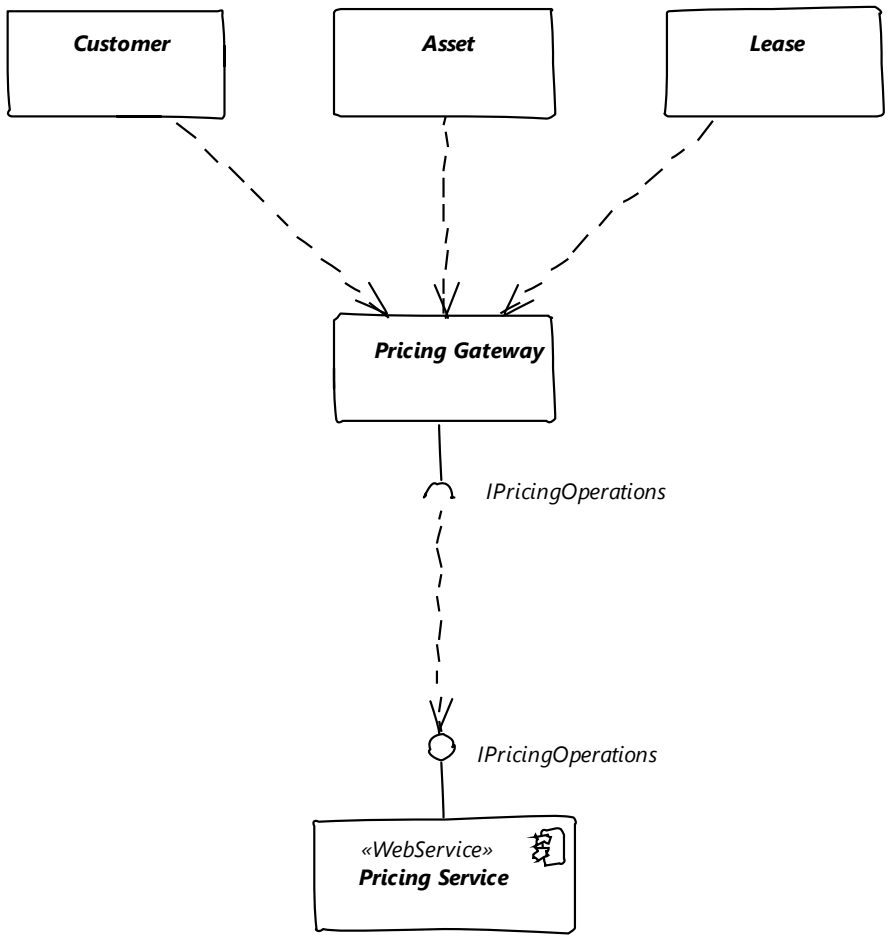
Gateway

Проблема:

ПО редко функционирует в изоляции от внешнего мира, для доступа к ресурсам используется специальные API. Однако, API изначально являются сложными, потому что принимают во внимание структуру ресурса. Это делает ПО более сложным.

Решение:

Объект, который инкапсулирует доступ к внешней системе и ресурсу.



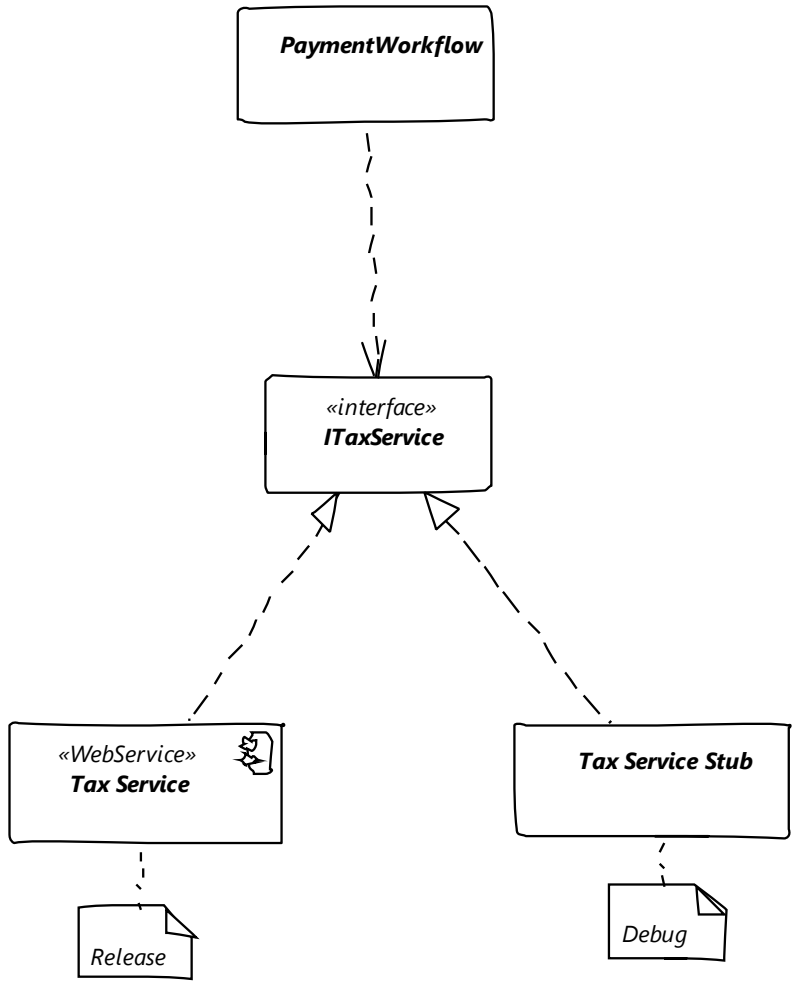
Service Stub

Проблема:

Корпоративные системы часто зависят от внешних сервисов, таких как расчёт кредитного рейтинга, налоговые ставки. При разработке такие зависимости становятся головной болью из-за своей непредсказуемости.

Решение:

Замена сервиса на заглушку упростит разработку и позволит писать модульные тесты.



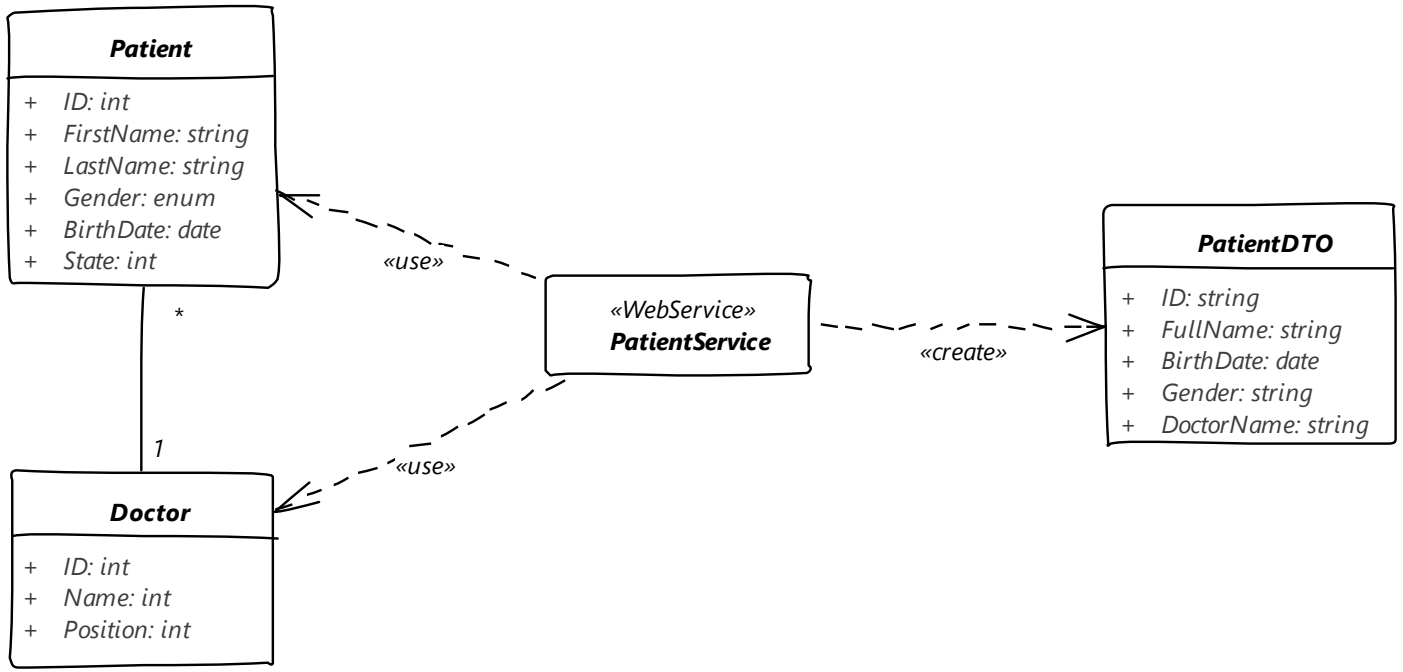
Data Transfer Object

Проблема:

Классы модели предметной области избыточны для передачи наружу.

Решение:

Синтетические классы для передачи данных между слоями системы.



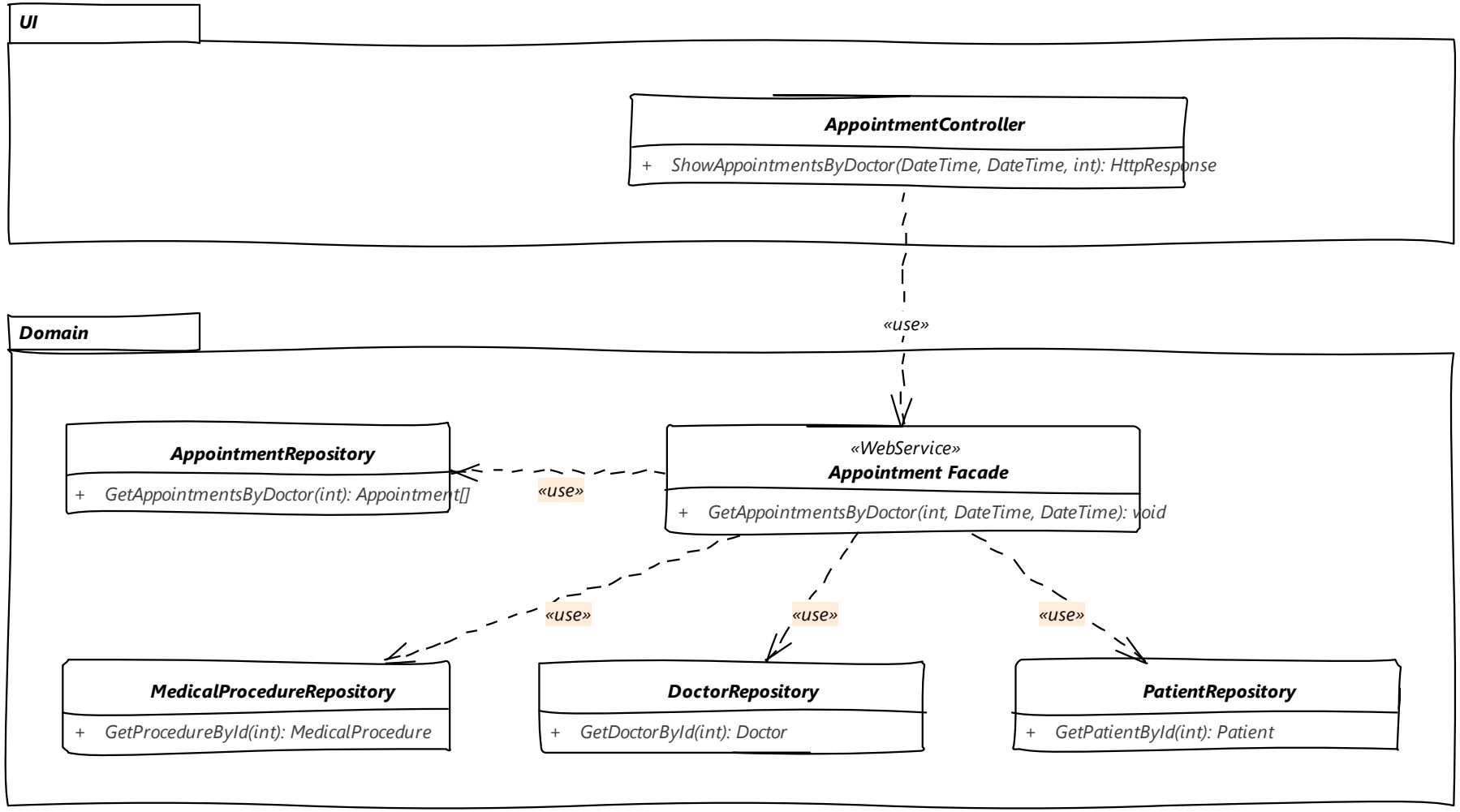
Remote Facade

Проблема:

В одном адресном пространстве “мелкогранулярные” взаимодействия работаю хорошо, но всё меняется, когда происходит взаимодействие между процессами, т.к. удалённые вызовы значительно более затратны.

Решение:

Общий объединяющий интерфейс для набора методов объекта для улучшения эффективности сетевого взаимодействия. Remote Façade транслирует общие запросы в набор небольших запросов к подчиненным объектам.



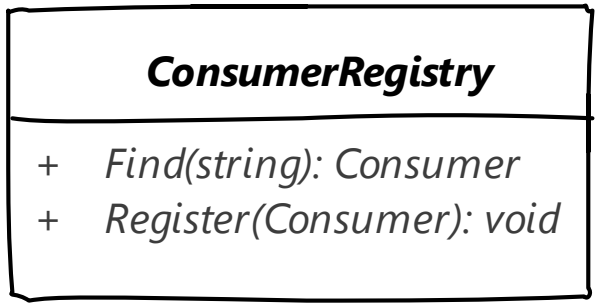
Registry

Проблема:

Когда нужно найти какой-либо объект, обычно начинают с другого объекта, связанного с целевым. Например, если нужно найти все счета для покупателя, начинают, как раз с покупателя и используют его метод получения счетов. Тем не менее, в некоторых случаях нет подходящего объекта, с которого начать. Например, известен ID покупателя, но нет ссылки на него.

Решение:

Хорошо известный объект, который используется другими объектами для получения общих объектов.



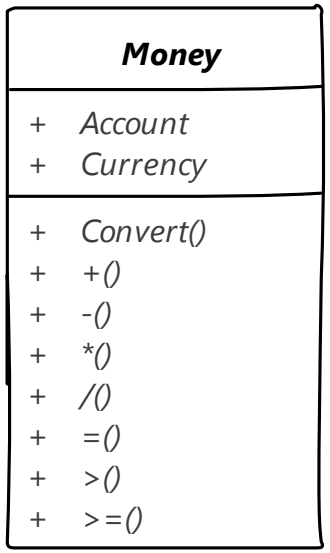
Money

Проблема:

Огромное количество компьютеров в мире обрабатывают данные о деньгах. Финансовые операции трудно реализуются в современных языках программирования.

Решение:

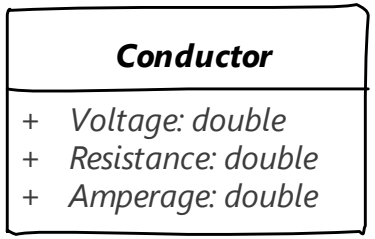
Создаем
класс Money (Деньги),
чтобы работать с
денежными
величинами и избегать
общих ошибок.

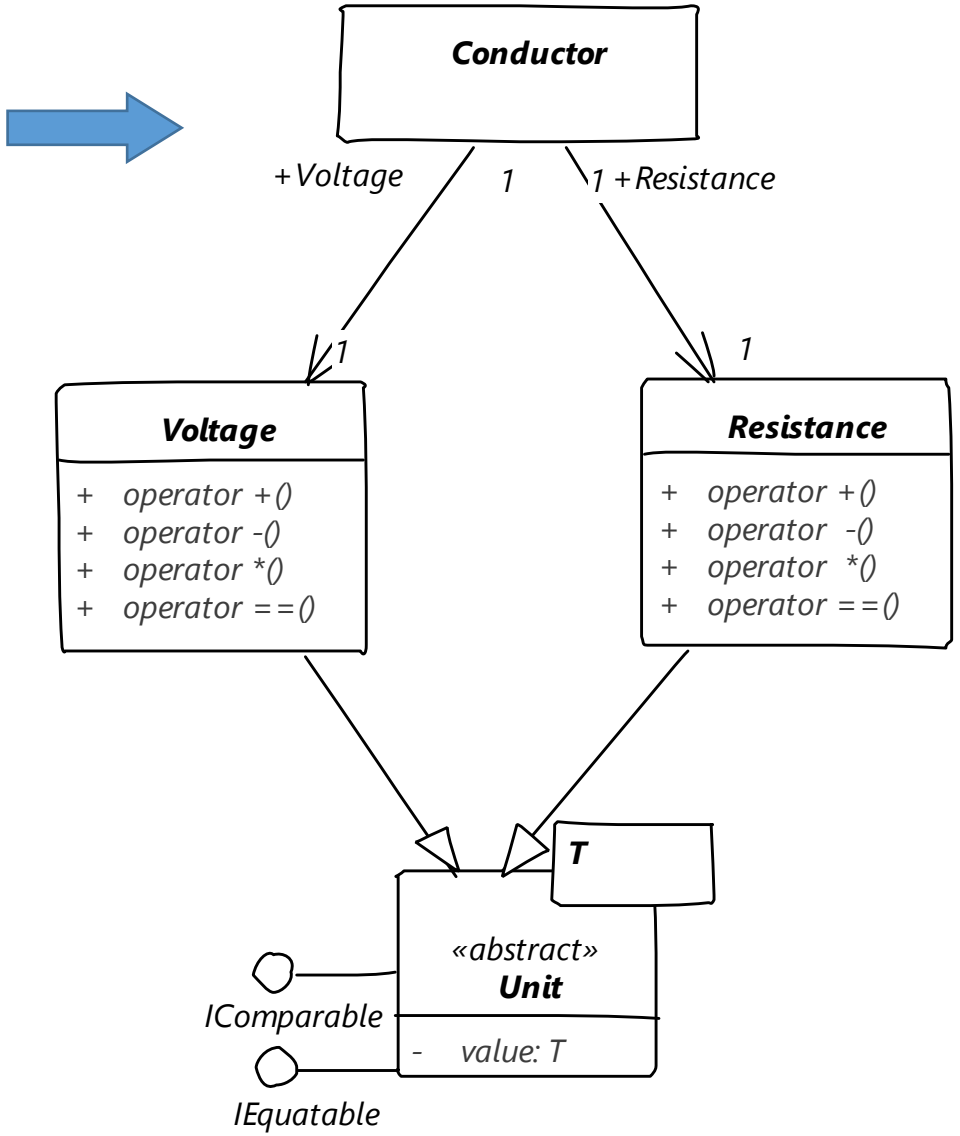


Microtype

Проблема:

Выразительность базовых типов недостаточна для моделирования физических значений.
Такие значения похожи на числа, но имеют много дополнительных правил и ограничений.





Решение:

Моделируем физические значения специальными классами, ведущими себя как базовые типы, но обеспечивающими необходимые инварианты.

```
var v0 = 220.Volt();
var tolerance = 12.0.Volt();
var v = v0 + tolerance ;
```

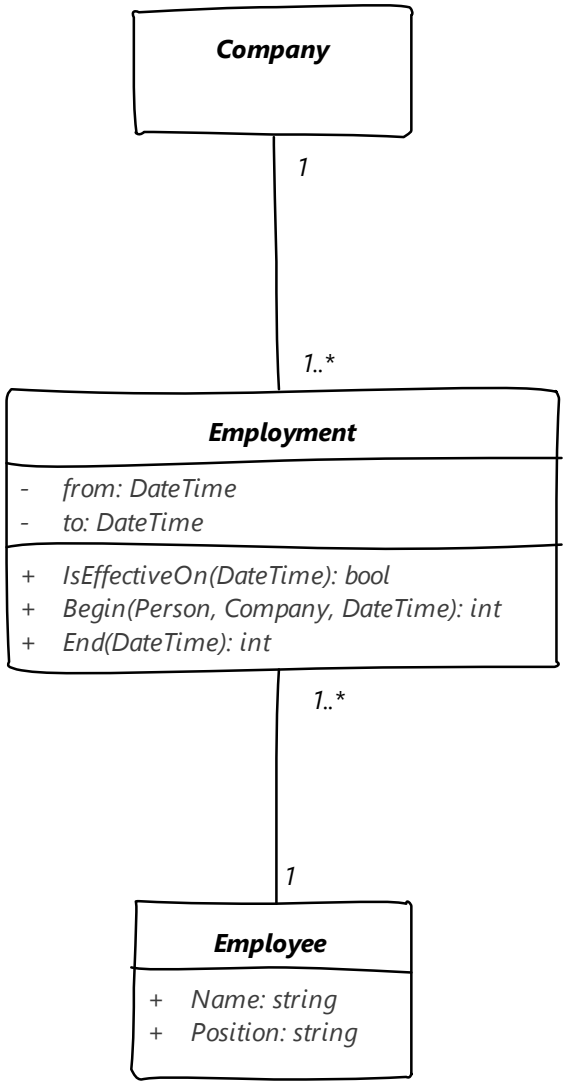
Effectivity

Проблема:

Многие факты верны только в течение определенного периода времени. Таким образом, очевидный способ описать эти факты - обозначить их периодом времени. Во многих случаях этот период представляет собой пару дат.

Решение:

Введение в модель диапазона действия. Диапазон имеет смысл предоставить в виде интерфейса, который лучше подходит для класса.
Метод создания может принять дату начала периода действия, а затем использовать диапазон с открытым концом, чтобы указать, что нет даты окончания: это соответствует случаям, когда что-то создано для определенной даты и действует до дальнейшего уведомления. Когда это дополнительное уведомление приходит, можно использовать метод закрытия периода.



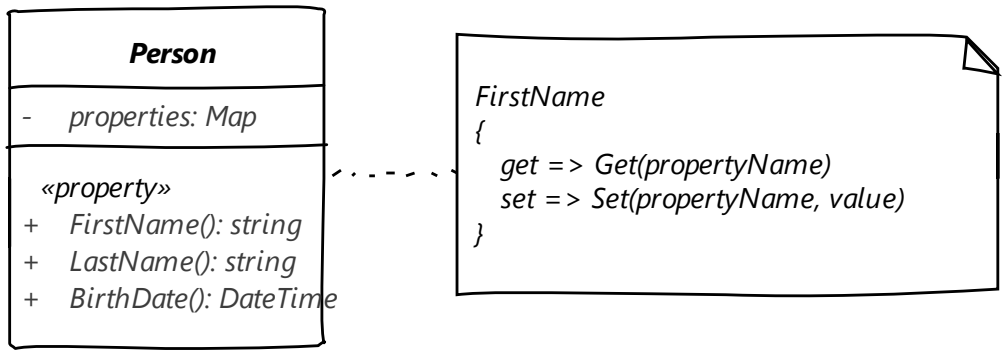
Temporal Property

Проблема:

Неудобно писать много аксессоров к полям, требуется дополнительная логика в аксессорах.

Решение:

Хранение виртуальных полей в словаре, к которому обращаются аксессоры.



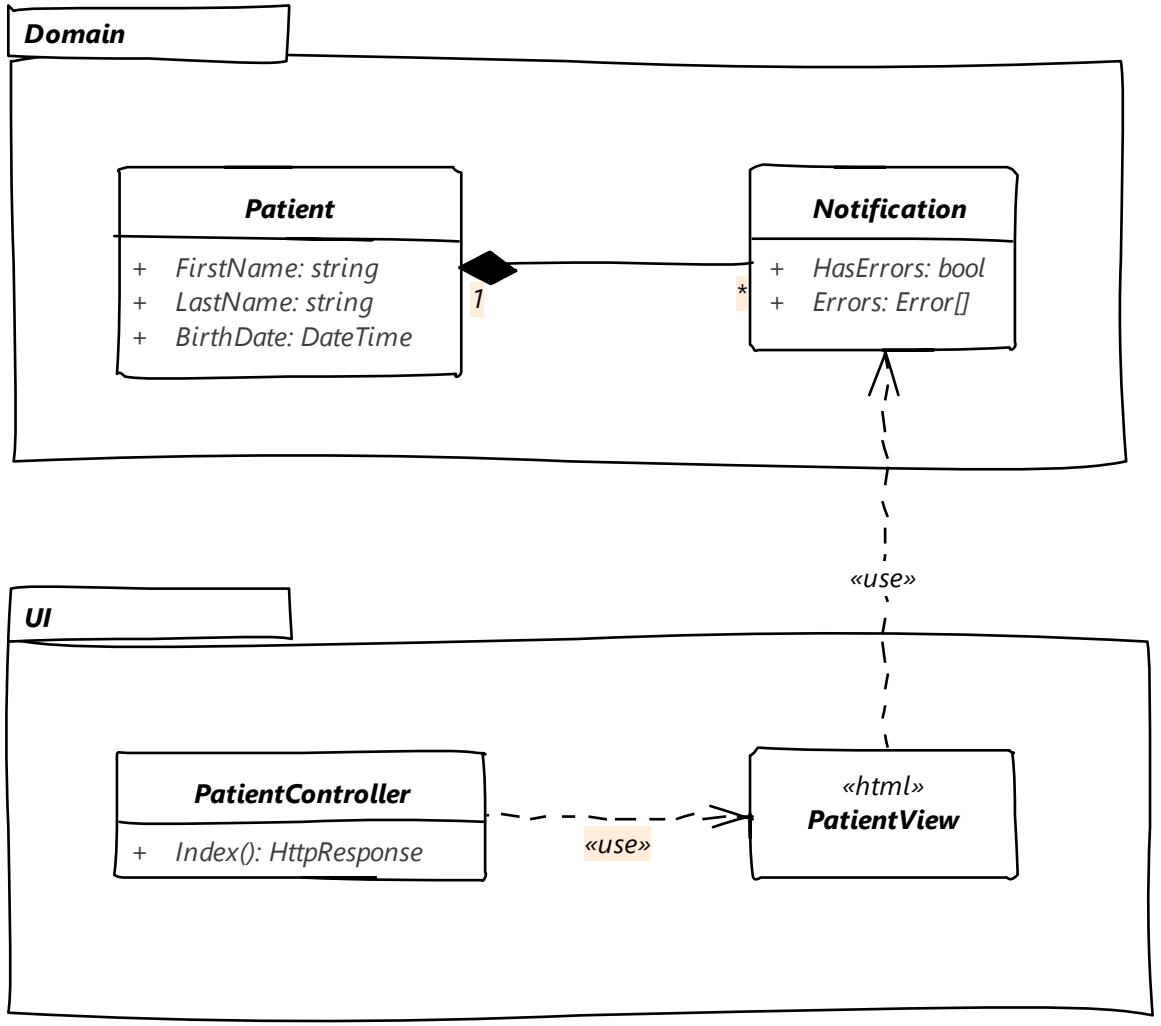
Notification

Проблема:

Ошибки уровня бизнес-логики необходимо удобным и безопасным образом отображать в пользовательском интерфейсе.

Решение:

Объект, который собирает информацию об ошибках и другую подобную информацию на уровне бизнес-логики и передает ее в представление.



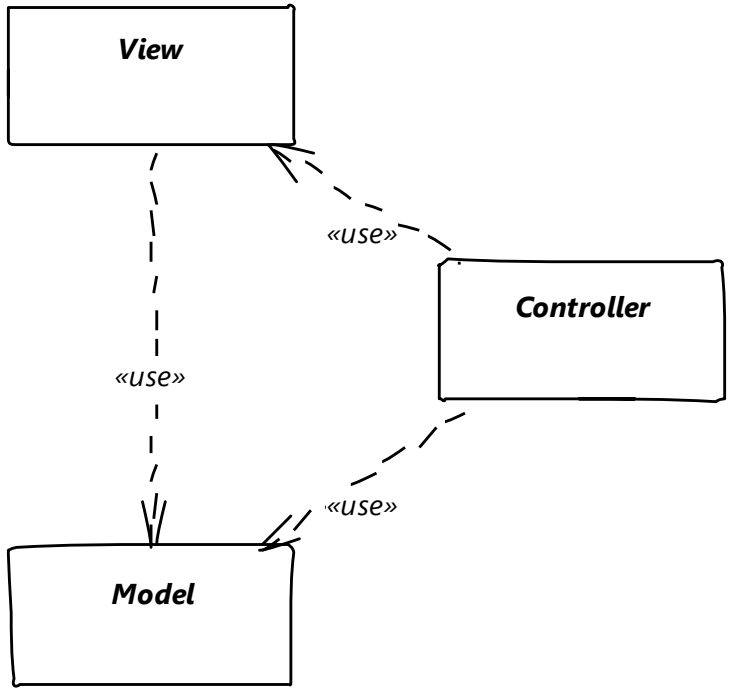
MVC

Проблема:

Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Контроллер управляет логикой представления и переходами, представление читает модель.



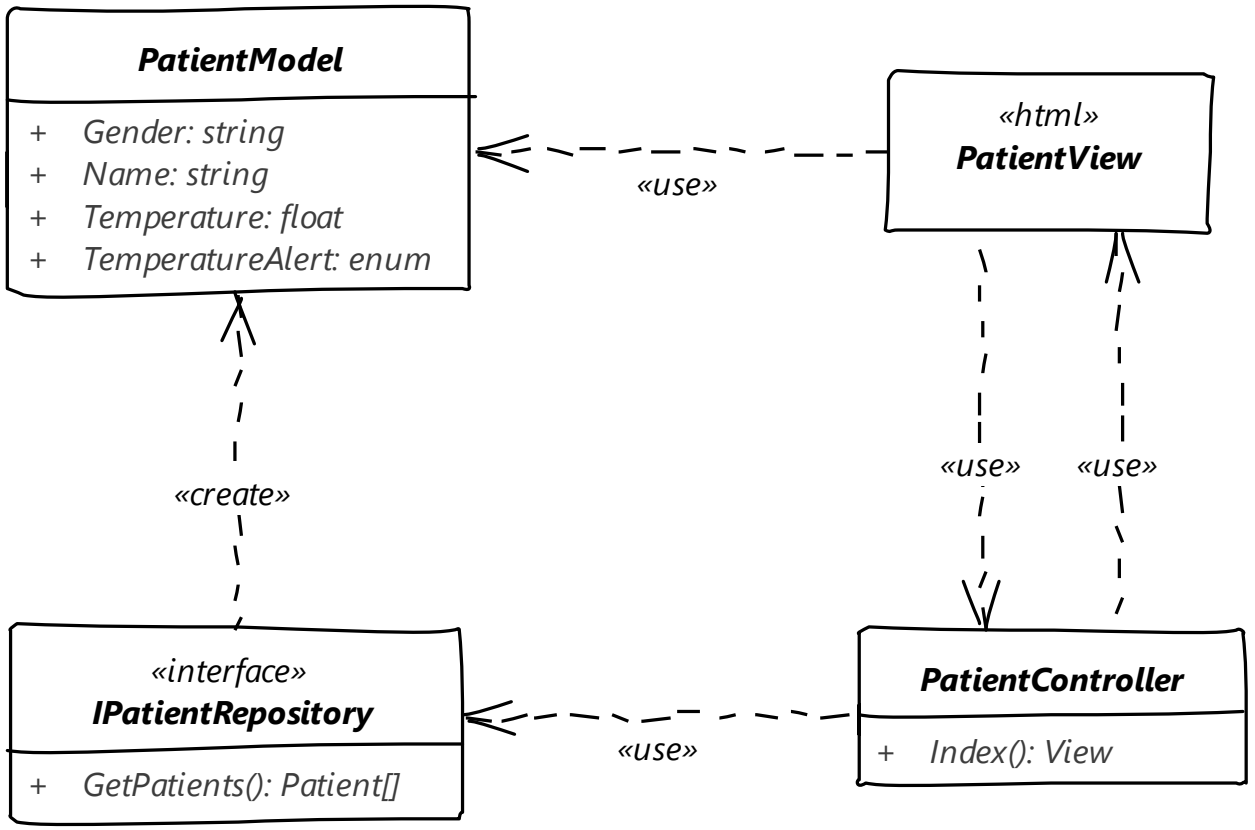
MVC

Проблема:

Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Контроллер управляет логикой представления и переходами, представление читает модель.



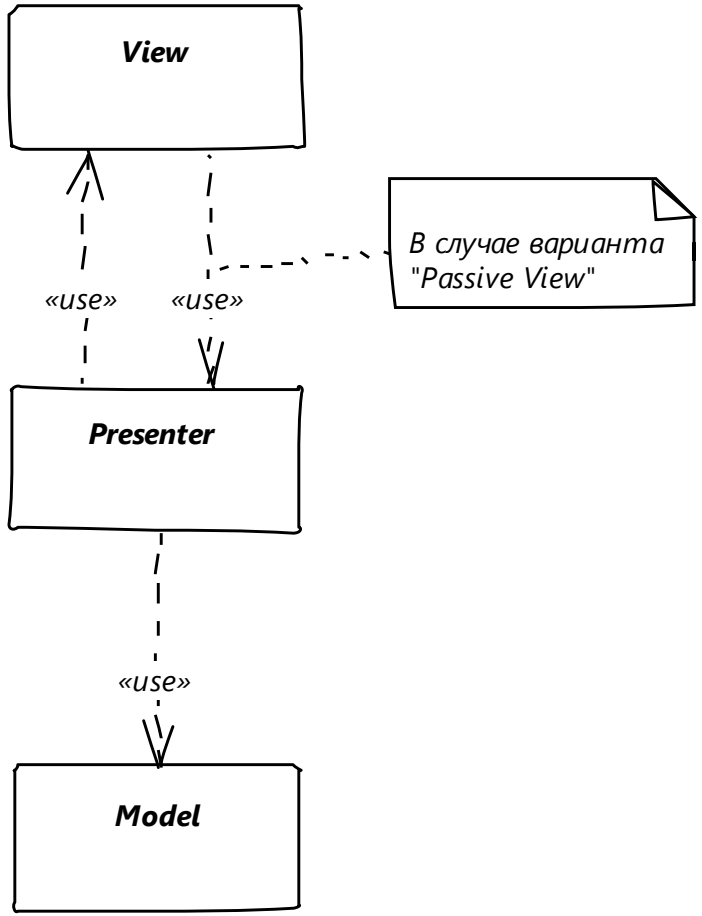
MVP

Проблема:

Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Презентер содержит логику управления представлением через абстрактный интерфейс.



MVP

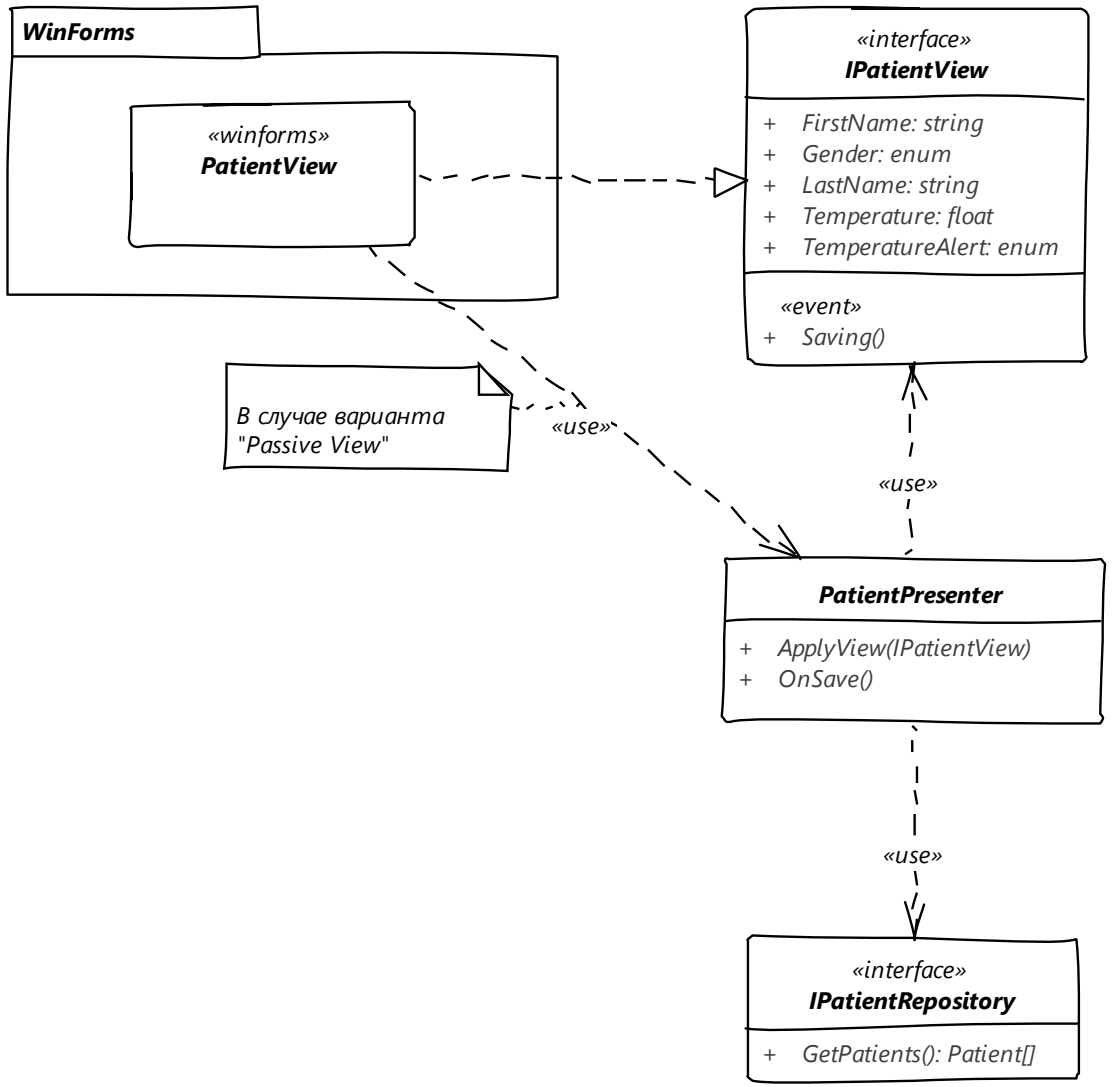
Проблема:

Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Презентер содержит логику управления представлением через абстрактный интерфейс.

Бывает разновидностей Passive View и Supervising Controller.



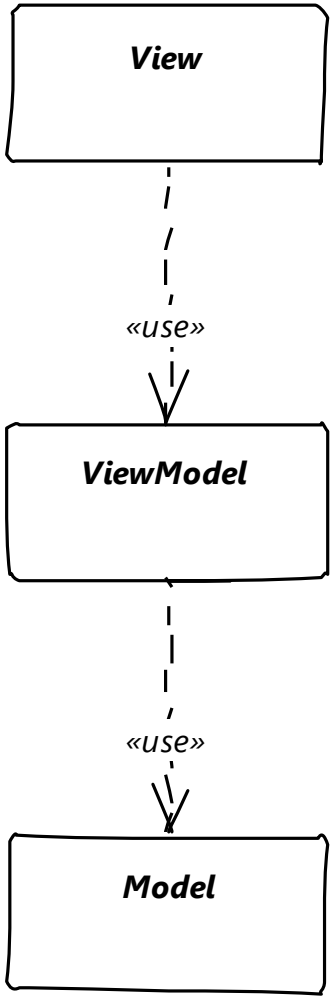
MVVM

Проблема:

Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Расширить паттерн MVP, используя мощные средства биндинга.



MVVM

Проблема:

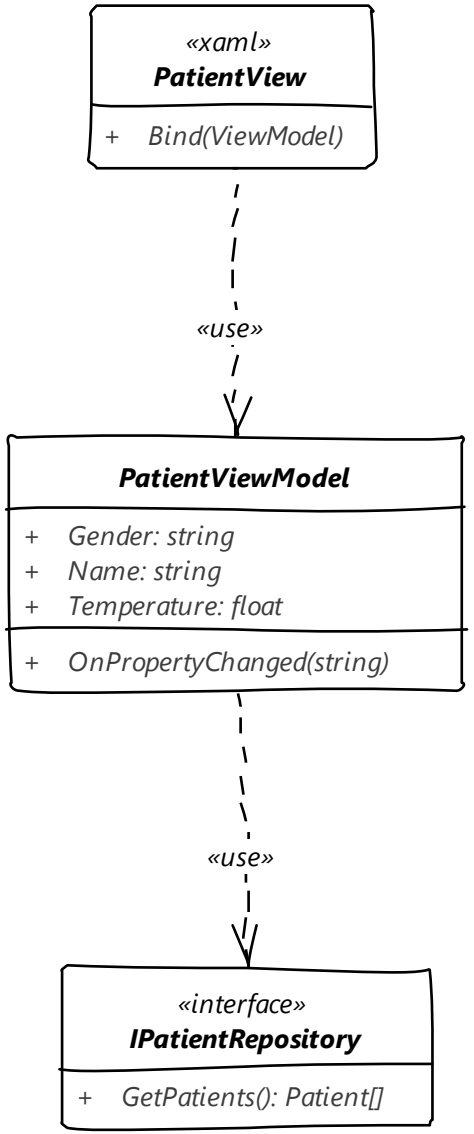
Сложность и высокая связность кода пользовательского интерфейса.

Решение:

Расширить паттерн MVP, используя мощные средства биндинга.

Презентационная модель выступает в роли посредника-прокси для данных.

Закон Парето ☹



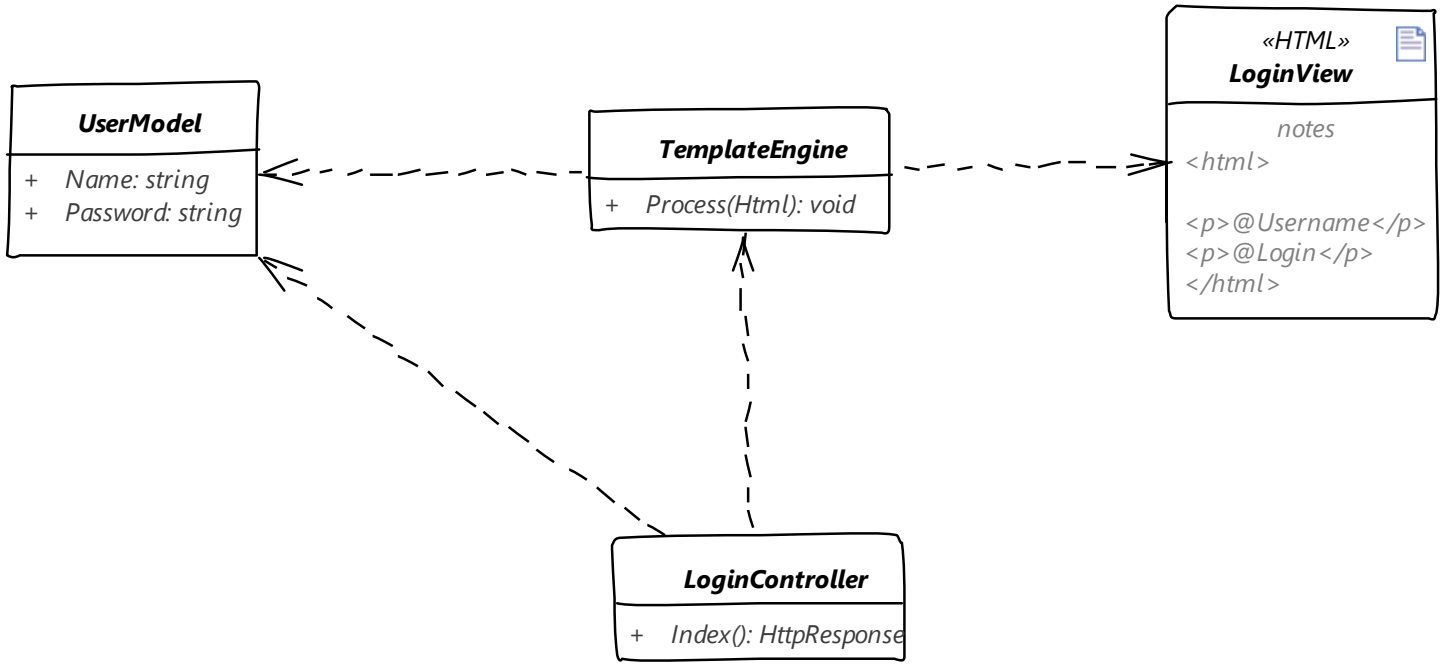
Template View

Проблема:

Создание приложений, генерирующих HTML, весьма трудоемко.

Решение:

Заполняет HTML-шаблон информацией при помощи маркеров, указанных в шаблоне.



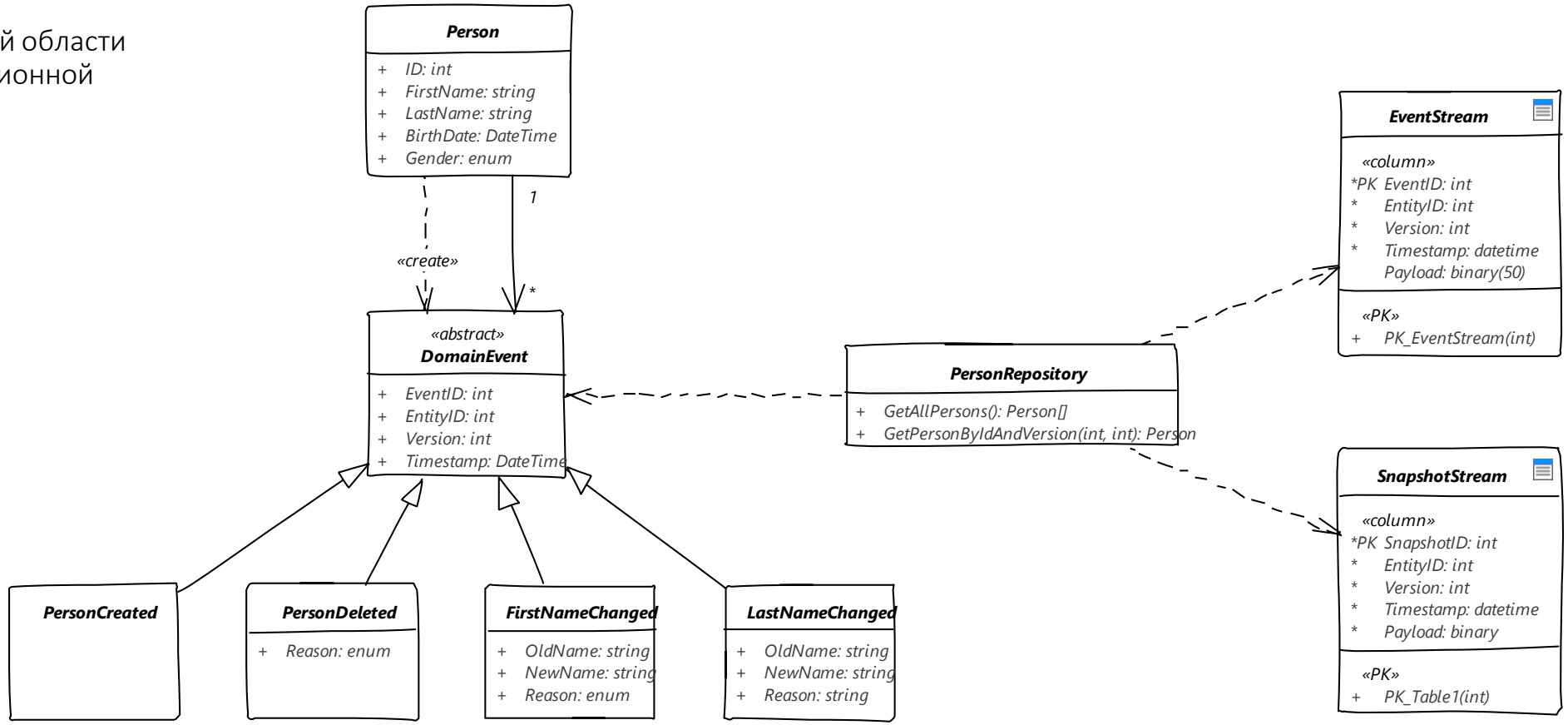
Event Sourcing

Проблема:

Некоторые модели предметной области плохо представляются в реляционной модели хранения.

Решение:

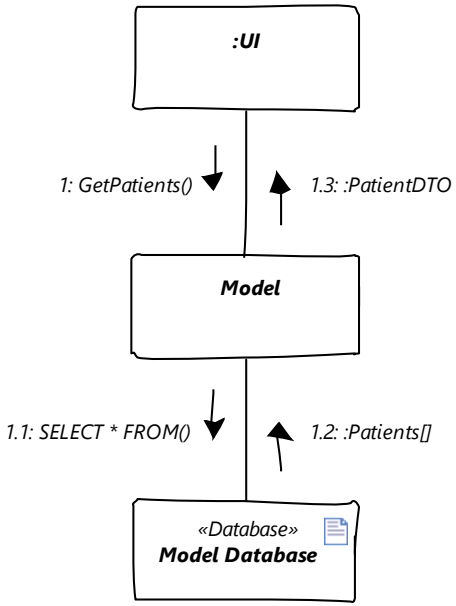
Хранить не слепки объектов в таблицах БД, а поток событий об изменениях объекта.



CQRS

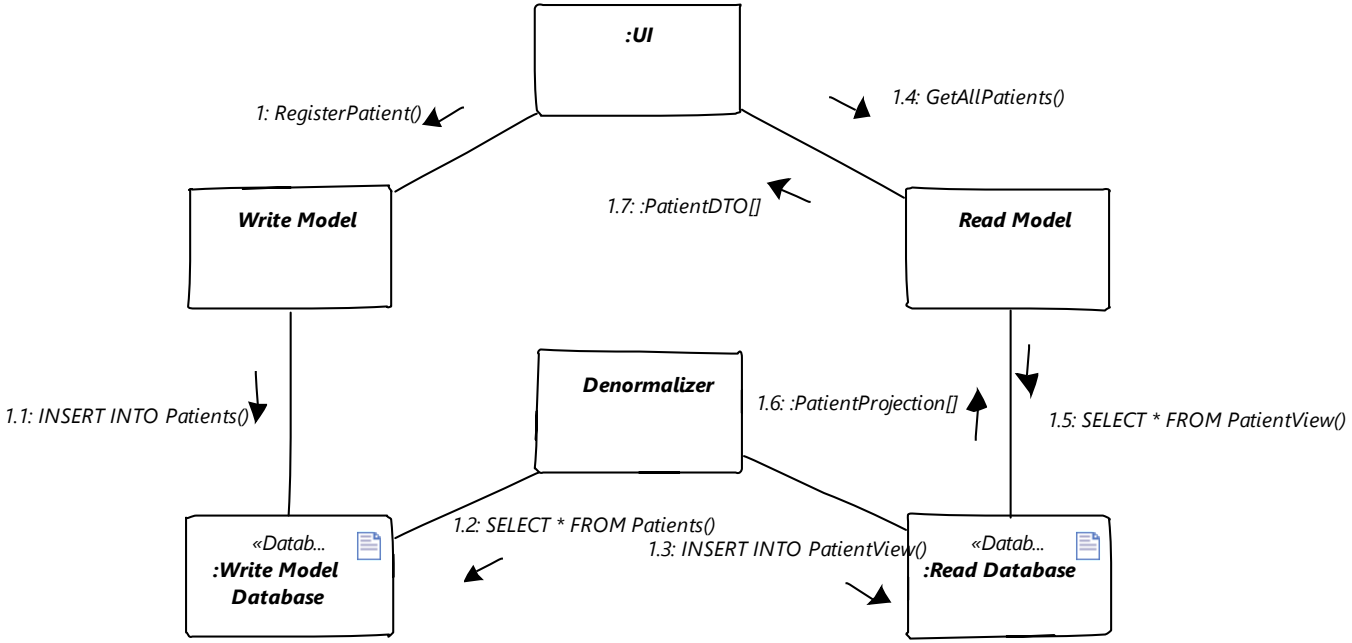
Проблема:

Проблемы горизонтального масштабирования транзакционной синхронной логики.



Решение:

Явное отделение команд от запросов и компромисс между актуальностью и моментальной целостностью данных,
и эластичностью.



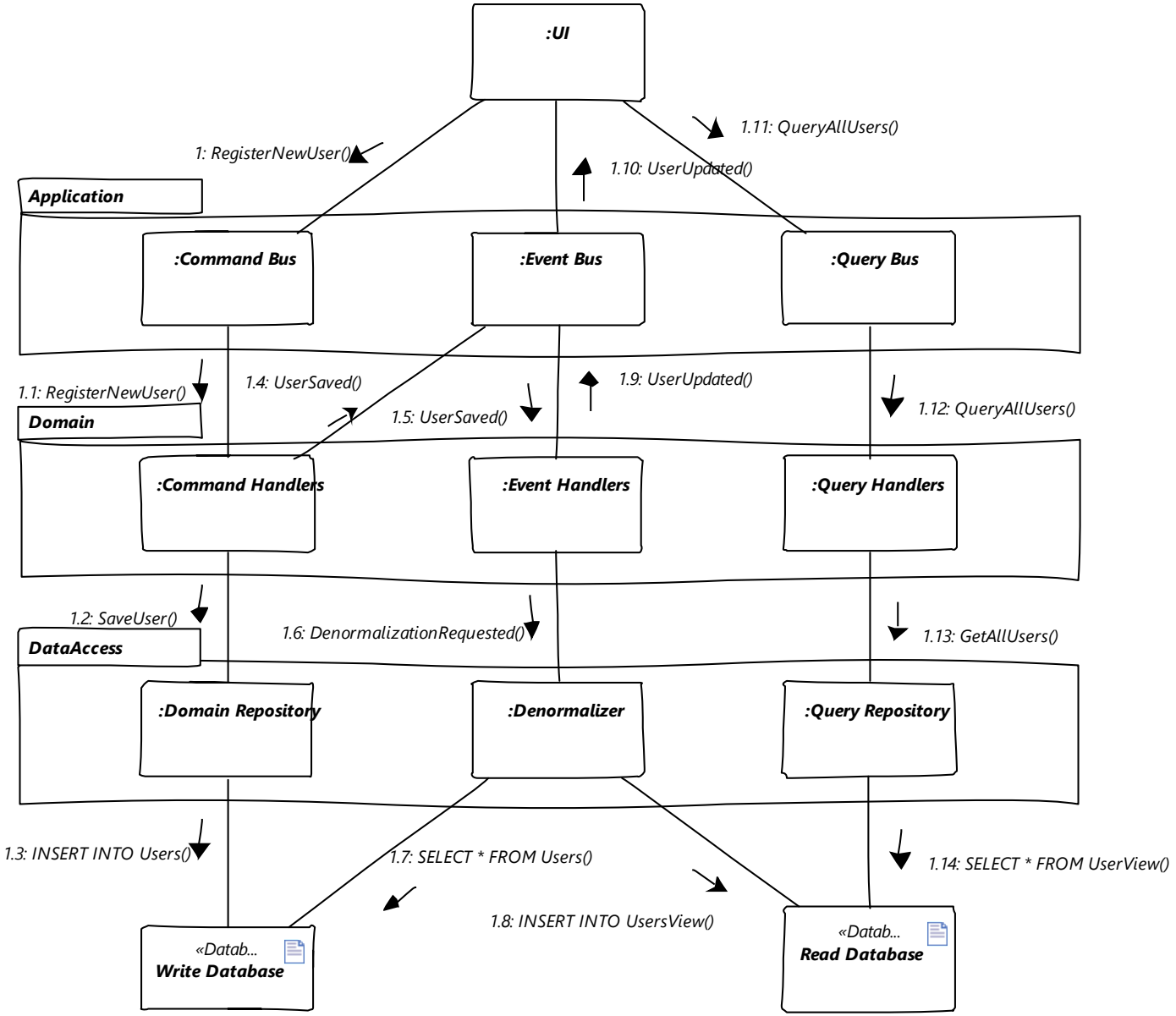
CQRS

Проблема:

Проблемы горизонтального масштабирования транзакционной синхронной логики.

Решение:

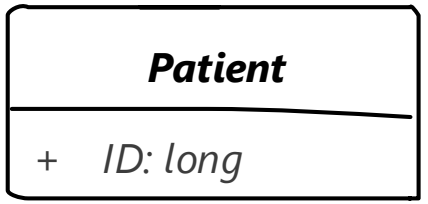
Явное отделение команд от запросов и компромисс между актуальностью и моментальной целостностью данных,
и эластичностью.



Identity Field

Проблема:

Реляционные базы данных отличают одну запись от другой при помощи первичного ключа, но объекты в памяти не нуждаются в таком ключе, так как сравниваются по ссылке. Для записи нужна привязка объектов к БД.



Решение:

Хранит первичный ключ из БД в объекте, чтобы обеспечивать соответствие между объектом и строкой в БД.

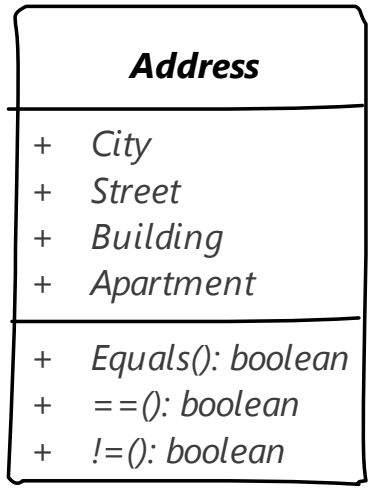
Value Object

Проблема:

Идентичность некоторых объектов определяется их содержанием.

Решение:

Маленький объект для хранения величин таких как деньги или диапазон дат, равенство которых не основано на идентичности.



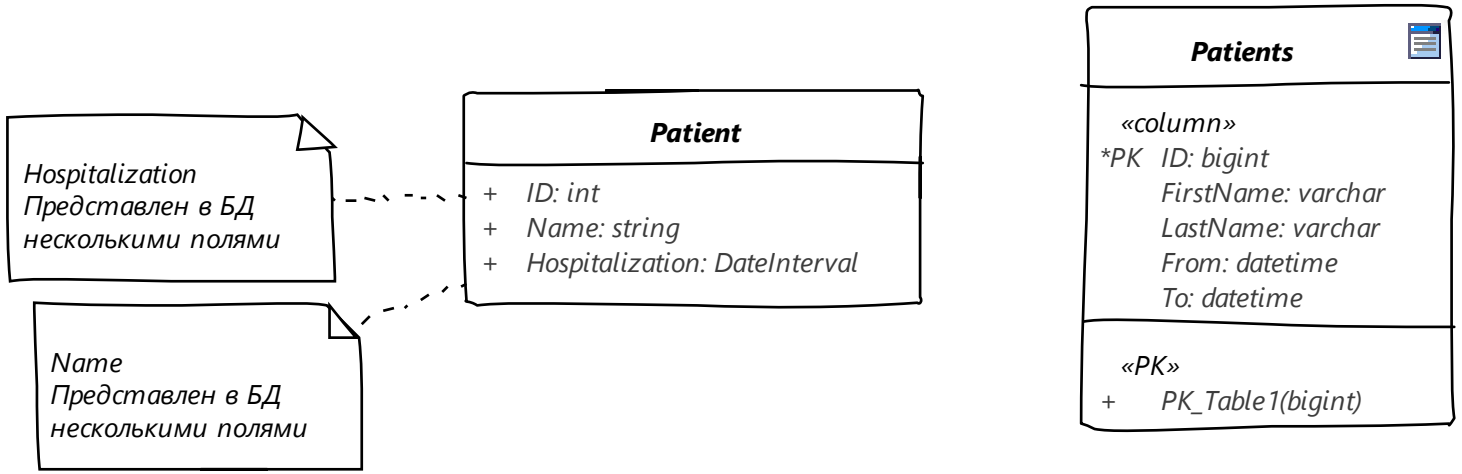
Embedded Value

Проблема:

Множество небольших объектов играют важную роль в объектно-ориентированной системе, но не подходят для хранения в отдельной таблице.

Решение:

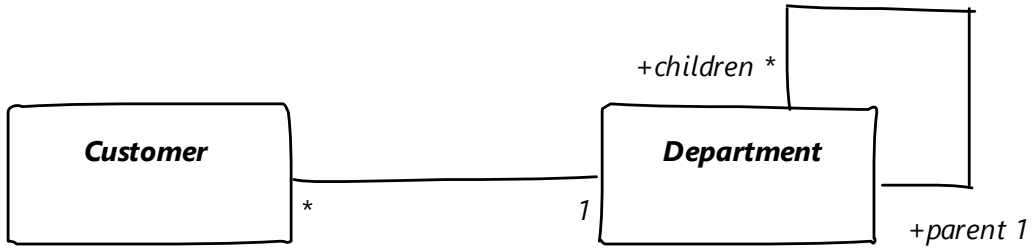
Записывает объект в несколько полей таблицы другого объекта.



Serialized BLOB

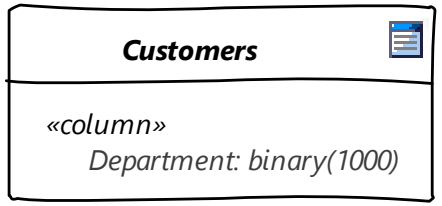
Проблема:

Объектная модель зачастую содержит сложные графы взаимосвязей мелких объектов, большая часть информации в таких структурах содержится не в самих объектах, а в связях между ними.
Хранение этой структуры в БД - непростая задача.



Решение:

Хранение графа связей объектов в БД посредством сериализации их в один binary large object (BLOB).



Active Record

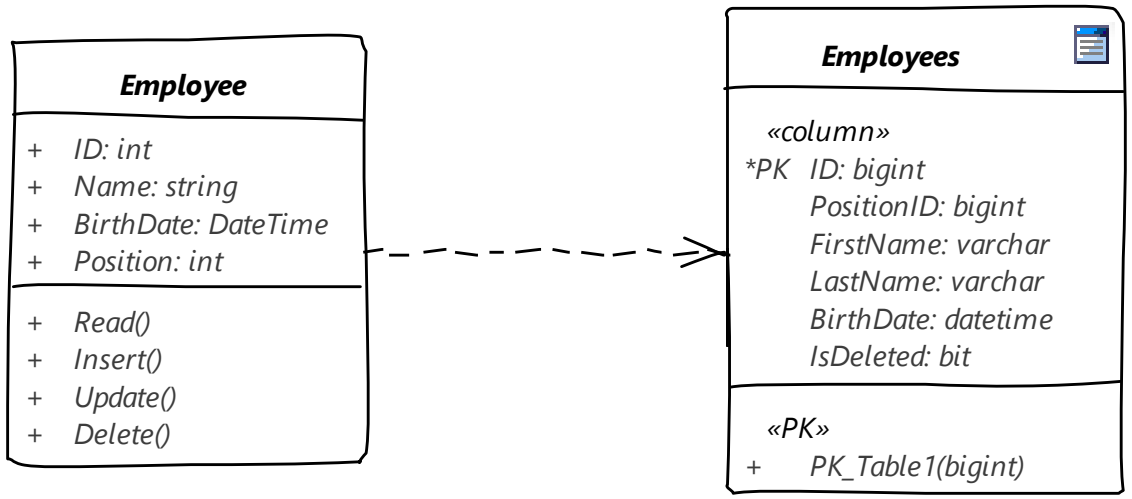
Проблема:

Модель данных проста и все объекты легко представляются записью в таблице БД.

Решение:

Один объект управляет и данными, и поведением.

Объект является "обёрткой" одной строки из БД, включает в себя доступ к БД и логику обращения с данными.



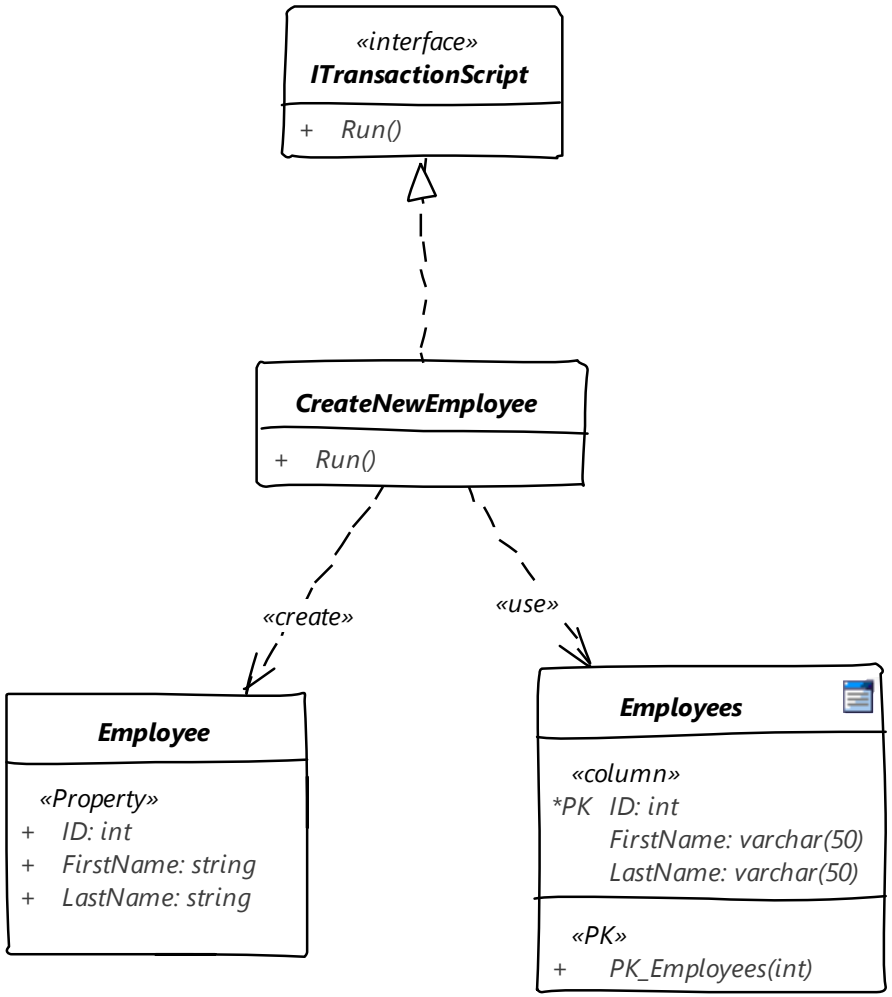
Transaction Script

Проблема:

Большинство бизнес-приложений можно представить в виде набора транзакций. Каждое взаимодействие пользователя и системы содержит определённый атомарный набор действий.

Решение:

Организует бизнес-логику в процедуры, которые управляют каждая своим запросом, работая с БД напрямую или через тонкую обёртку.



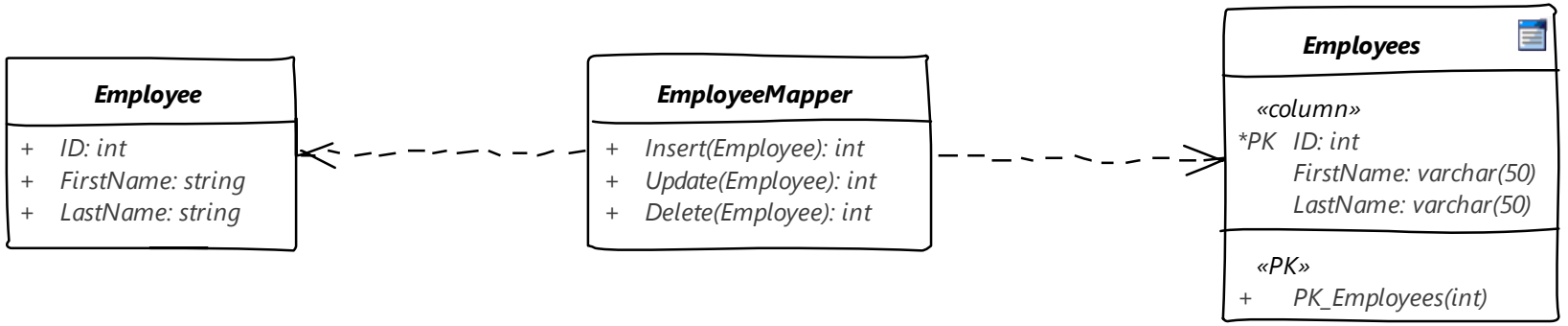
Data Mapper

Проблема:

Объекты предметной области должны быть по возможности избавлены от знания о БД, в которой хранятся.

Решение:

Программная прослойка, разделяющая объект и БД.
Его обязанность — пересылать данные между ними и изолировать их друг от друга.



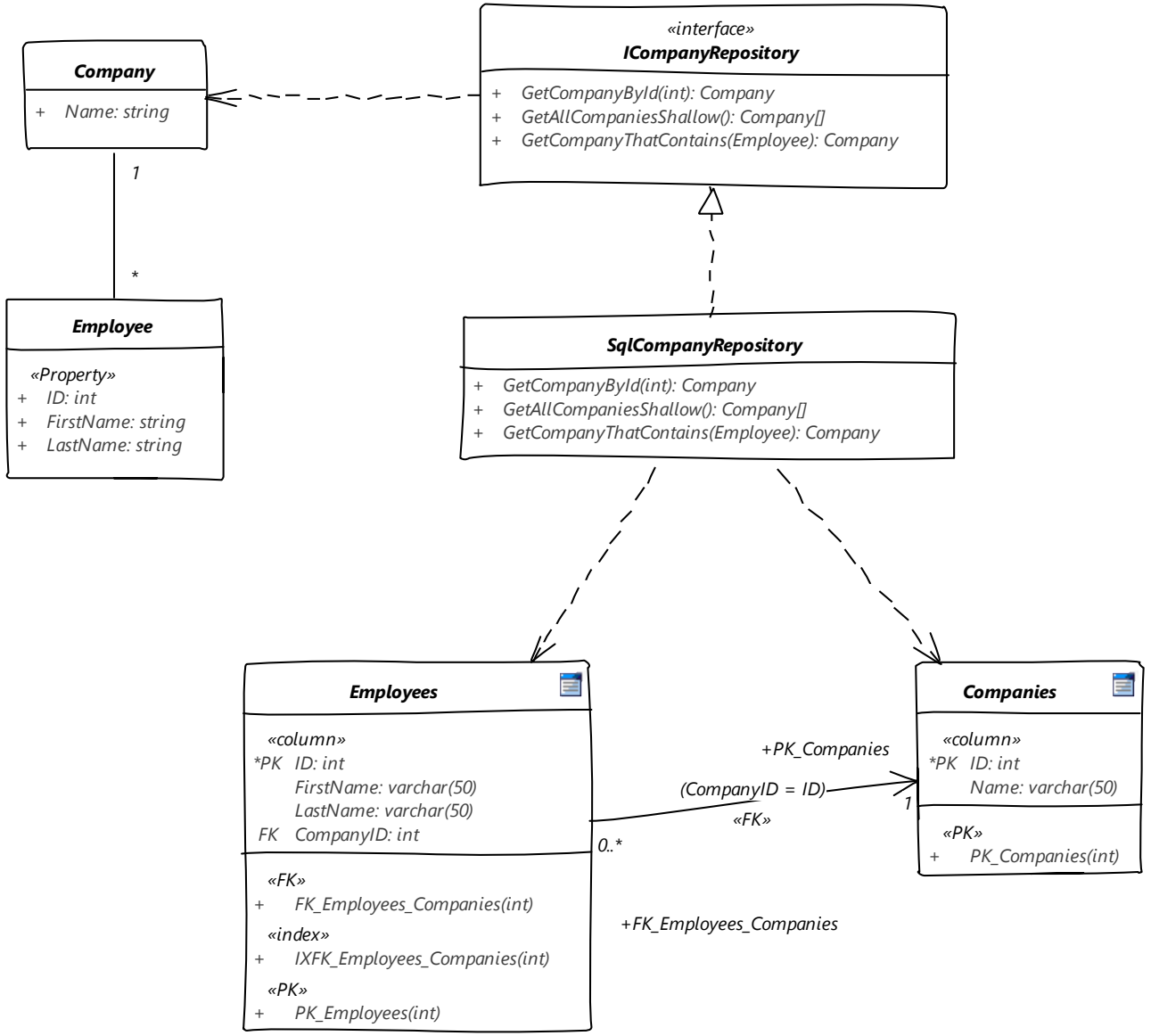
Repository

Проблема:

Удобно иметь объектно-ориентированный интерфейс над БД, позволяющий обращаться к ней как к простой коллекции доменных объектов.

Решение:

Посредник между уровнями бизнес-логики и хранения данных посредством интерфейса, инкапсулирующего специфические запросы к БД.



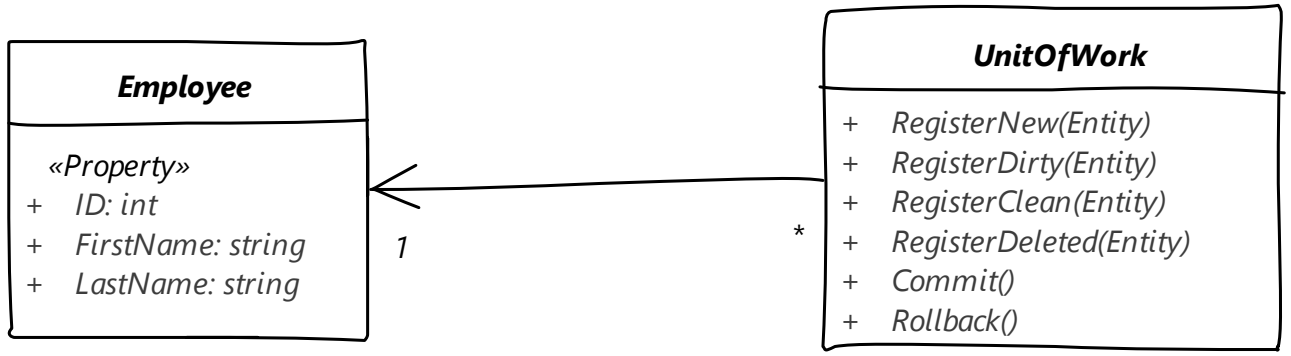
Unit of Work

Проблема:

Когда необходимо писать и читать из БД, важно следить за тем, что вы изменили и если не изменили - не записывать данные в БД. Также необходимо вставлять данные о новых объектах и удалять данные о старых. Можно записывать в БД каждое изменение объекта, но это приведёт к большому количеству мелких запросов к БД.

Решение:

Единица работы обслуживает набор объектов, изменяемых в бизнес-транзакции и управляет записью изменений и разрешением проблем конкуренции данных.



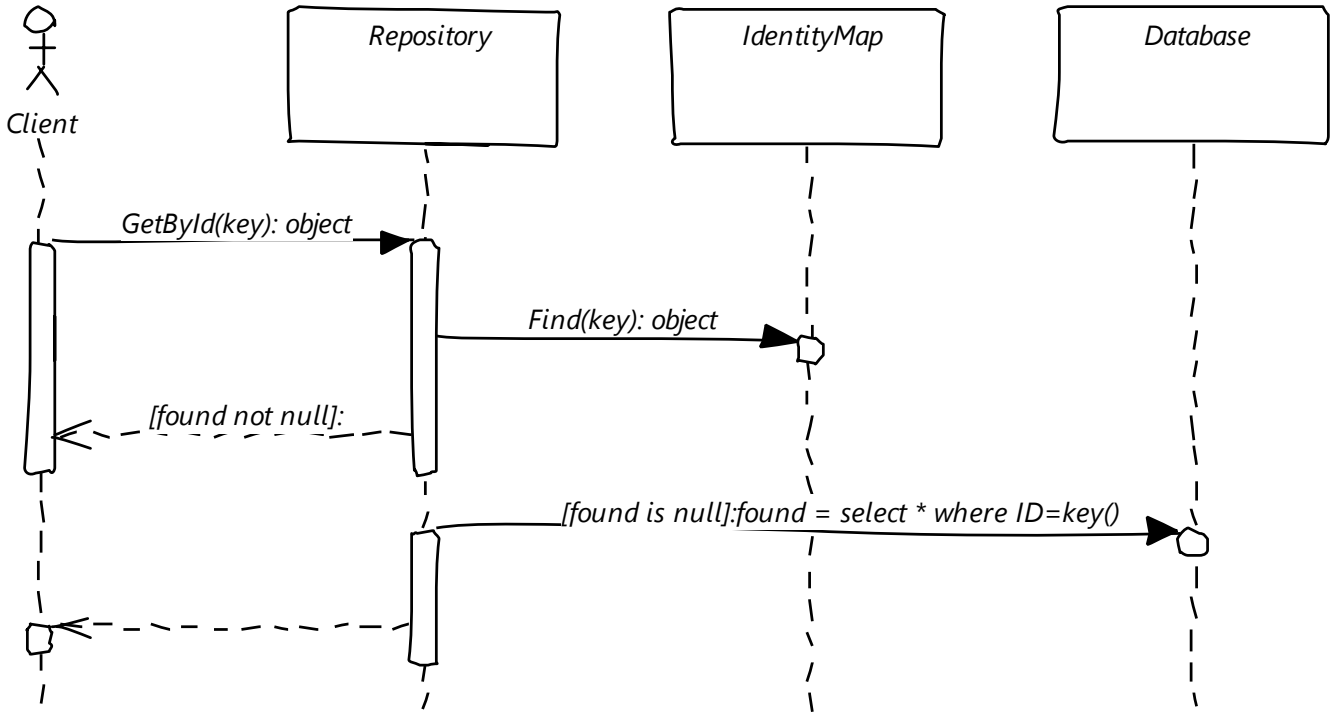
Identity Map

Проблема:

Когда дважды загружается одна и та же информация, увеличиваются затраты на передачу данных. Таким образом, отказ от загрузки одних и тех же данных дважды не только обеспечивает корректность информации, но и ускоряет работу приложения.

Решение:

Обеспечивает однократную загрузку объекта, сохраняя данные об объекте в карте соответствия. При обращении к объектам, ищет их в карте соответствия.



Query Object

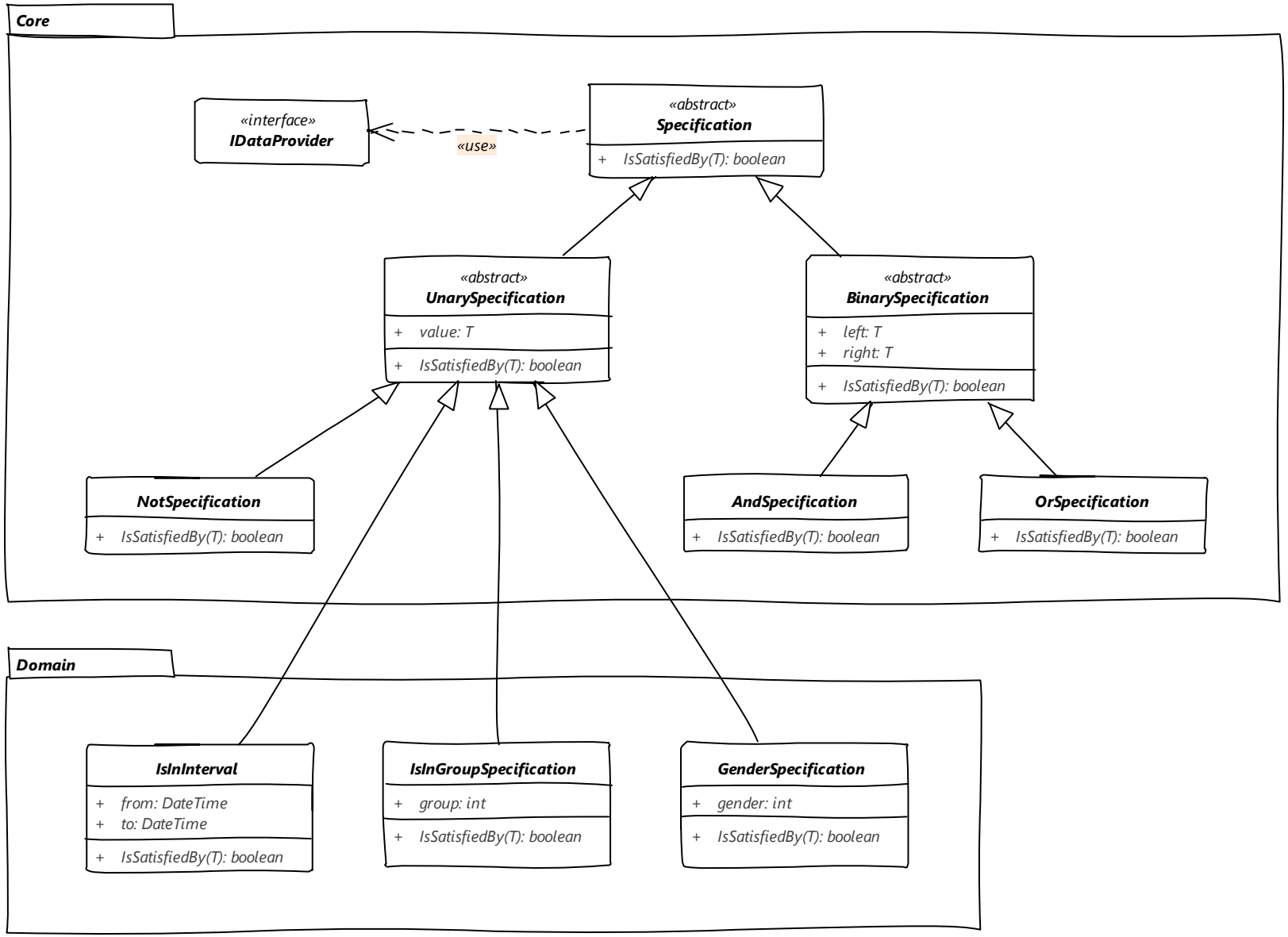
Проблема:

Использование языка SQL и других платформозависимых языков внутри слоя бизнес-логики делает архитектуру чрезвычайно хрупкой

Решение:

Объект, представляющий запрос к БД.

Структура объектов, которая может интерпретироваться в SQL-запрос. Таким образом создаётся независимость от структуры БД и конкретной реализации БД.



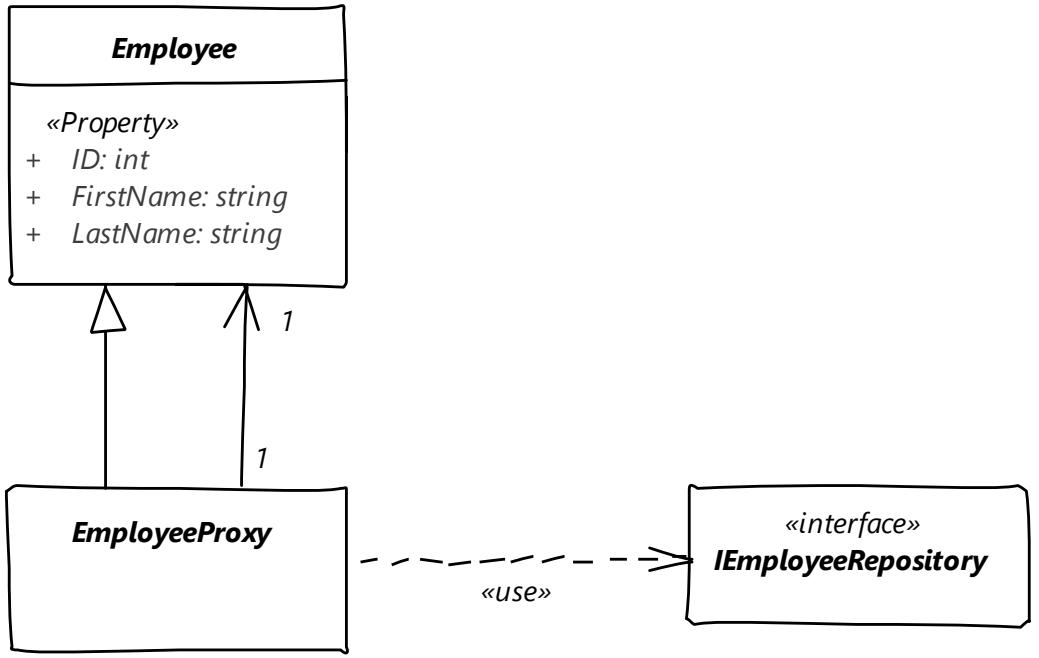
Lazy Load

Проблема:

Полная загрузка данных невыгодна с точки зрения расхода ресурсов.

Решение:

Объект, не содержит данных, но знает, где их взять. Ленивая загрузка подразумевает отказ от загрузки дополнительных данных, когда в этом нет необходимости. Вместо этого ставится маркер о том, что данные не загружены и их надо загрузить в случае, если они понадобятся.



Вопросы?