Объектно-ориентированный Анализ и Дизайн

Часть 3

Шаблоны проектирования
Архитектурные шаблоны
Rational Unified Process
Бизнес-анализ
Анти-шаблоны проектирования
Анти-шаблоны программирования

© V.Mukhortov, NSU, 2023

7. Шаблоны проектирования

Шаблон* проектирования / Design pattern – стандарное решение стандартной проблемы.

Понятие введено в книге
E.Gamma, R.Helm, R.Johnson and J.Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software

Авторов иногда совместно называют GoF (Gang of Four), а шаблоны проектирования – GoF-шаблонами.

* Наверное, было бы лучше использовать для слова pattern другой вариант перевода – *образец*, чтобы не путать с шаблонами (template) в C++, но так уж сложилось исторически.

Виды шаблонов

E. Gamma:

- Конструкционные шаблоны – абстрагируют создание объектов
    - Abstract Factory, Factory Method, Singleton

- Структурные шаблоны – решают проблемы композиции
    - Adapter, Bridge, Decorator, Proxy

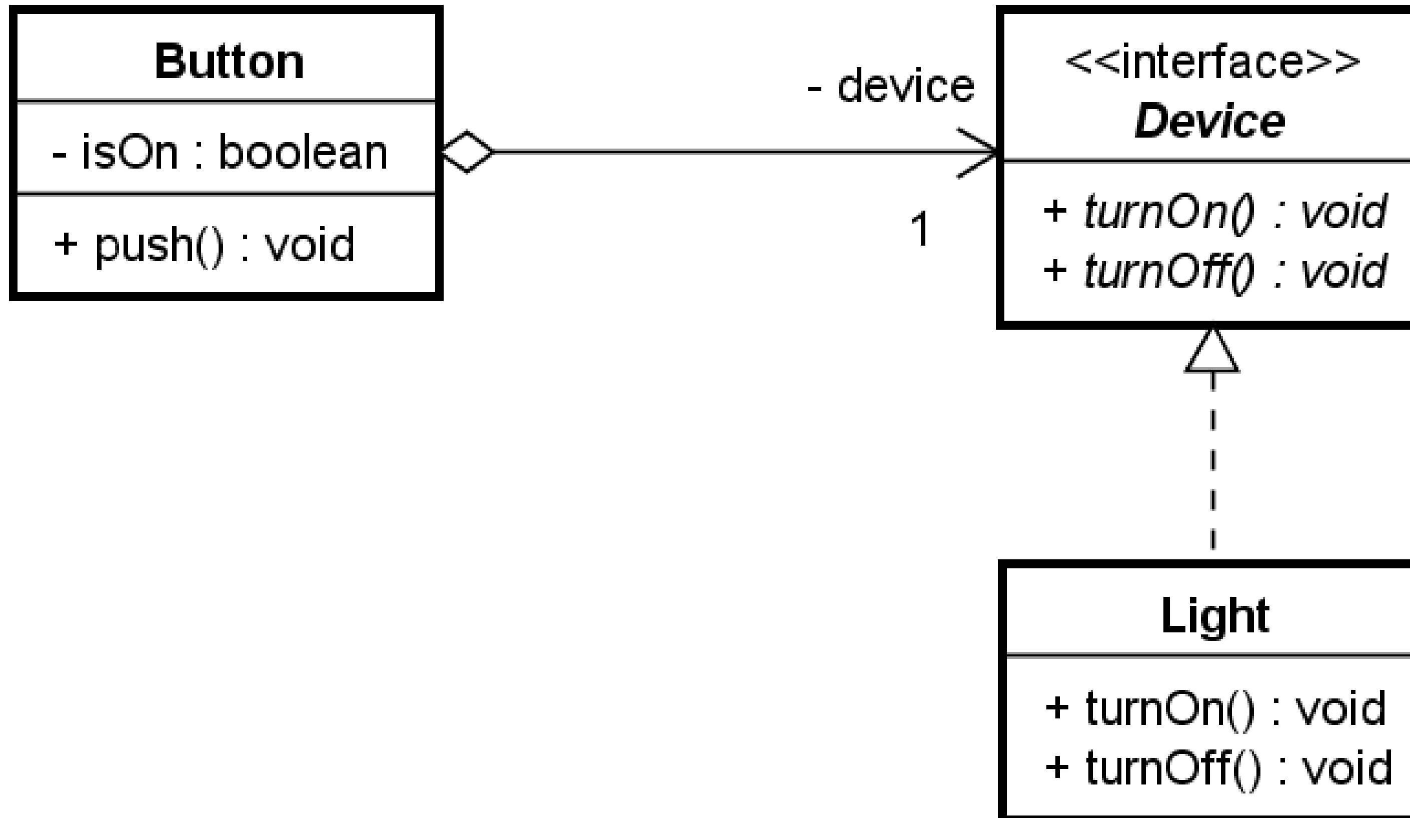
- Поведенческие шаблоны – алгоритмы и распределение ответственности между объектами
    - Command, Iterator, Observer, State, Strategy, Visitor

Abstract Server : проблема

Проблема:

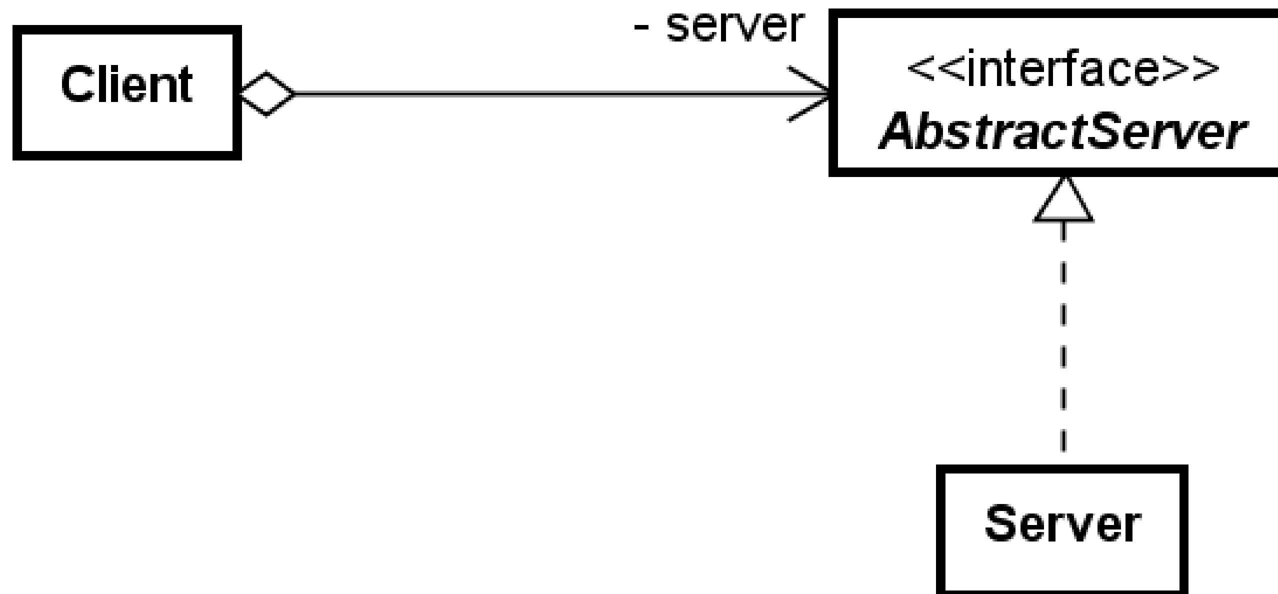
- Button нельзя использовать в контексте, не использующем Light

Abstract Server : решение



Решение: разорвать зависимость между Button and Light путем вставки интерфейса

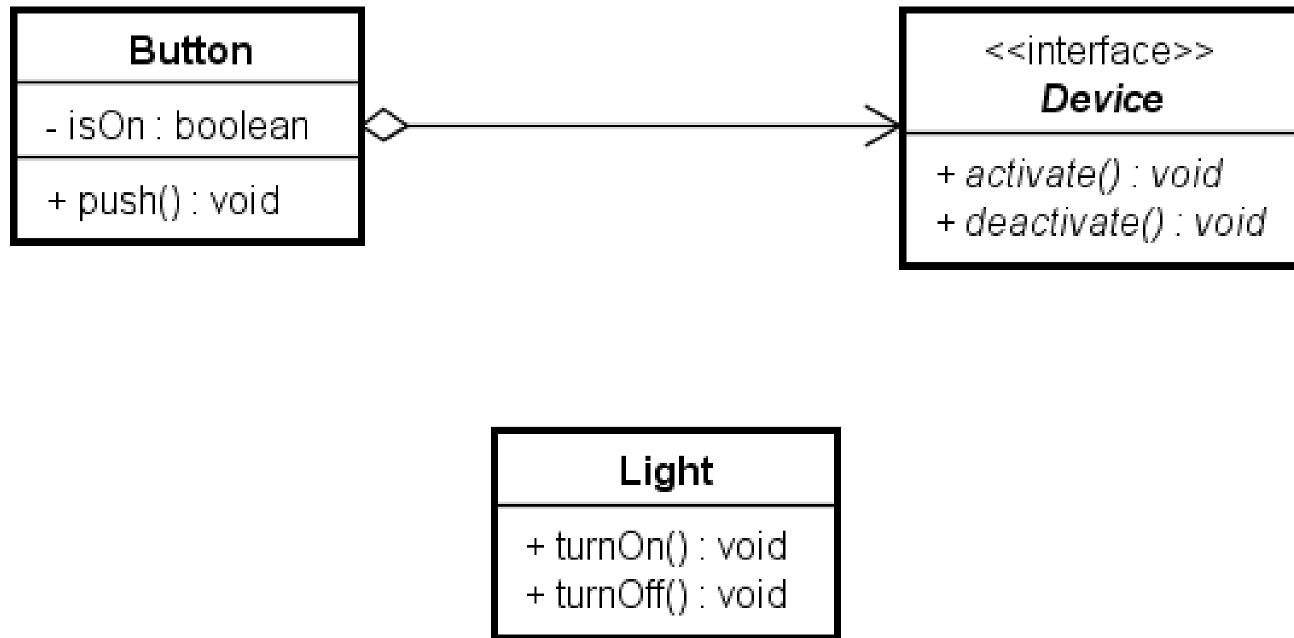
Abstract Server : шаблон



Плюсы:

- устраняет зависимость клиента от сервера
- сервера могут изменяться не влияя на клиентов
- устраняет нарушение DIP

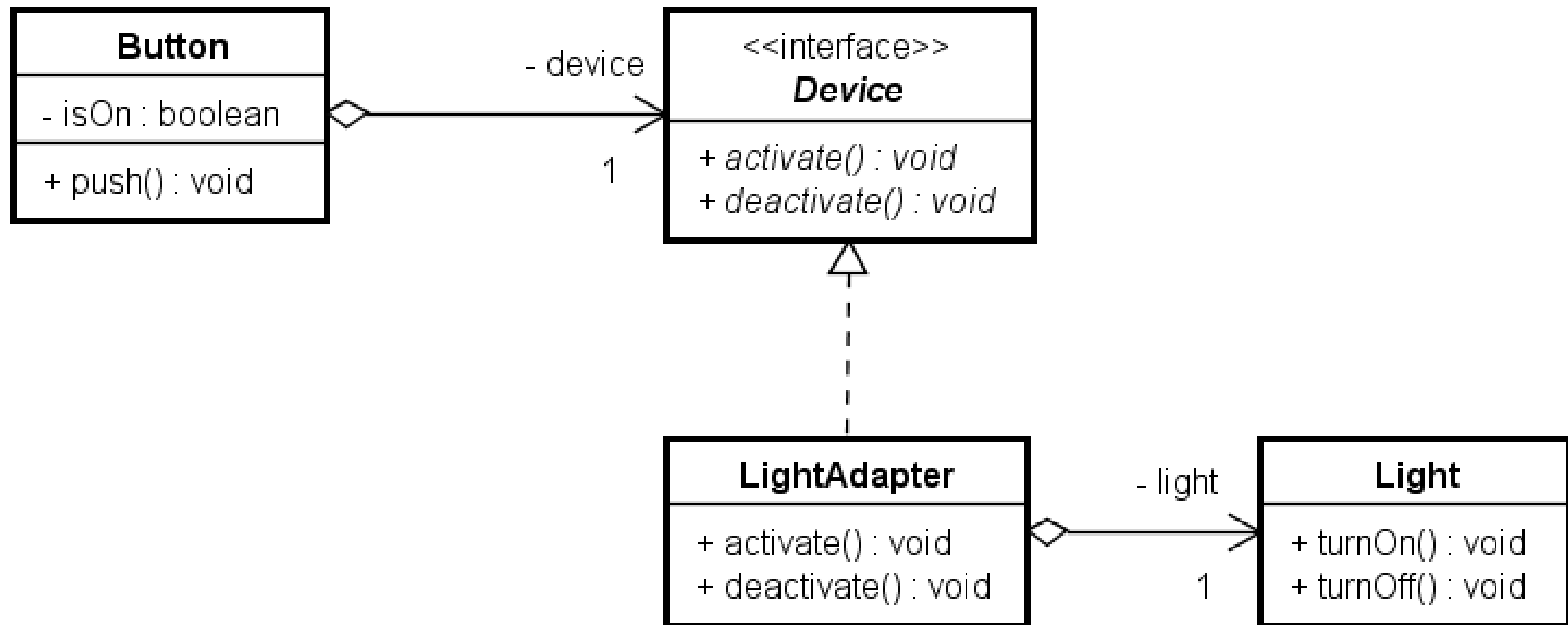
Adapter : проблема



Проблема:
- Light уже существует и не может реализовать интерфейс Device

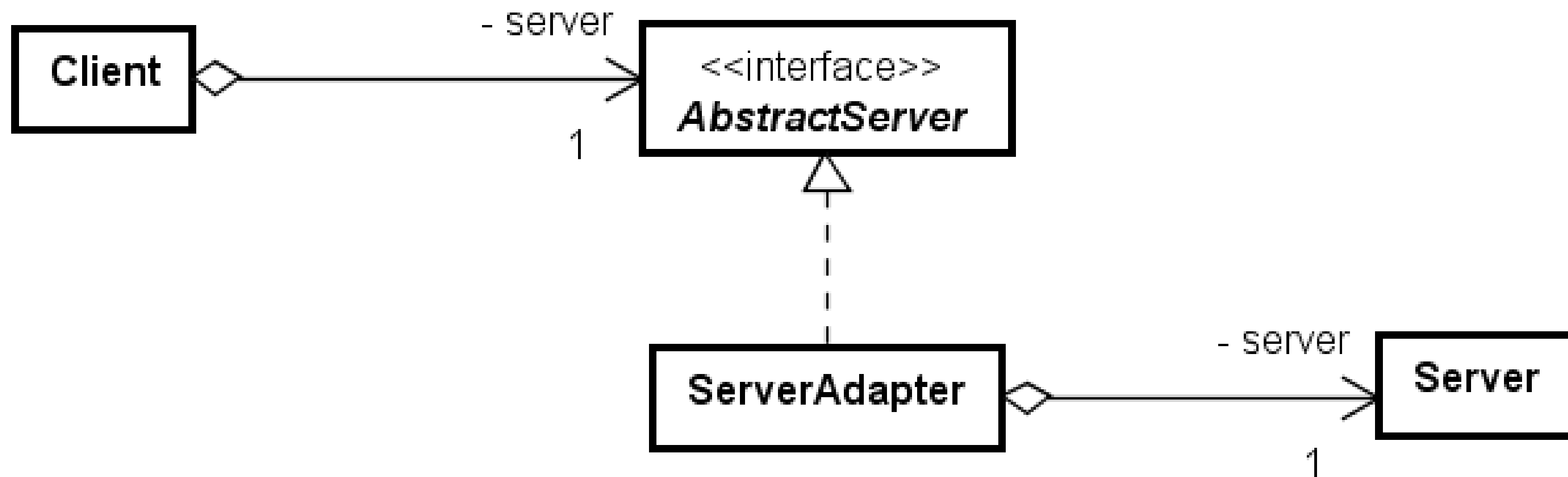
© V.Mukhortov, NSU, 2023

Adapter : решение



- Адаптер конвертирует один интерфейс в другой

Adapter : шаблон



Адаптирует Server к интерфейсу, необходимому Client

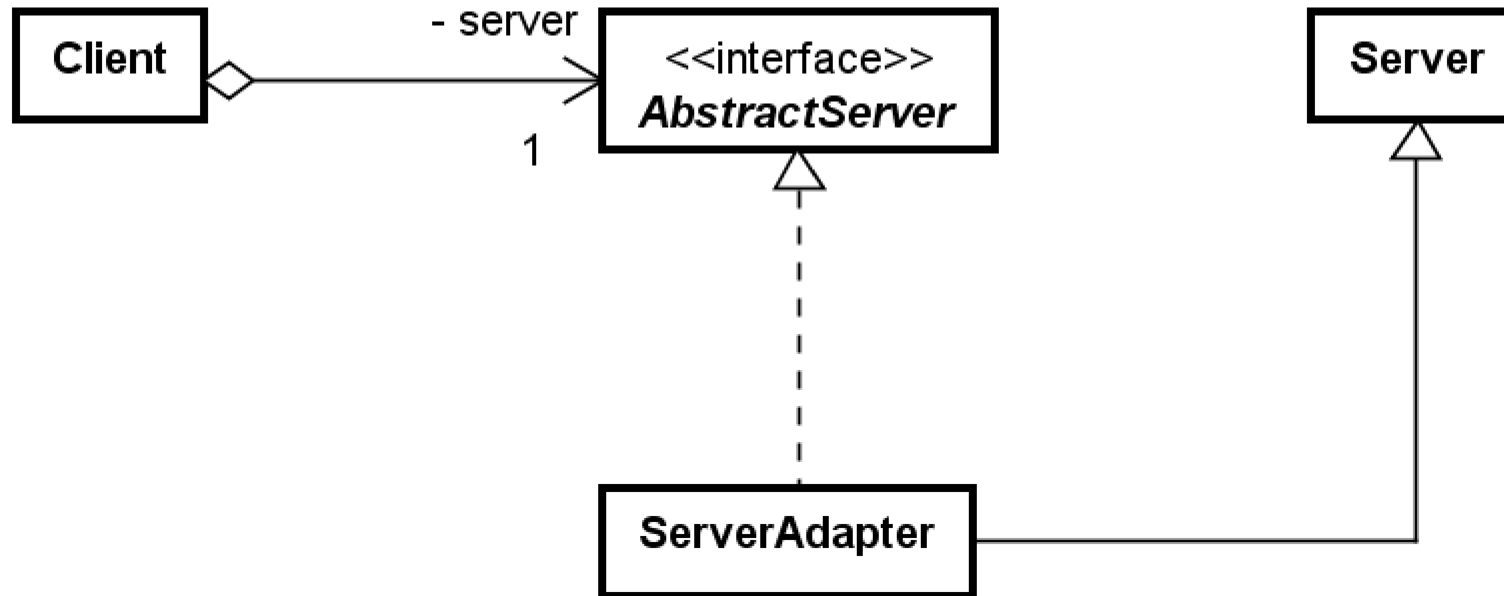
Также известен как: Wrapper

Плюсы:

+ разрыв зависимости между клиентом и сервером, когда сервер уже существует

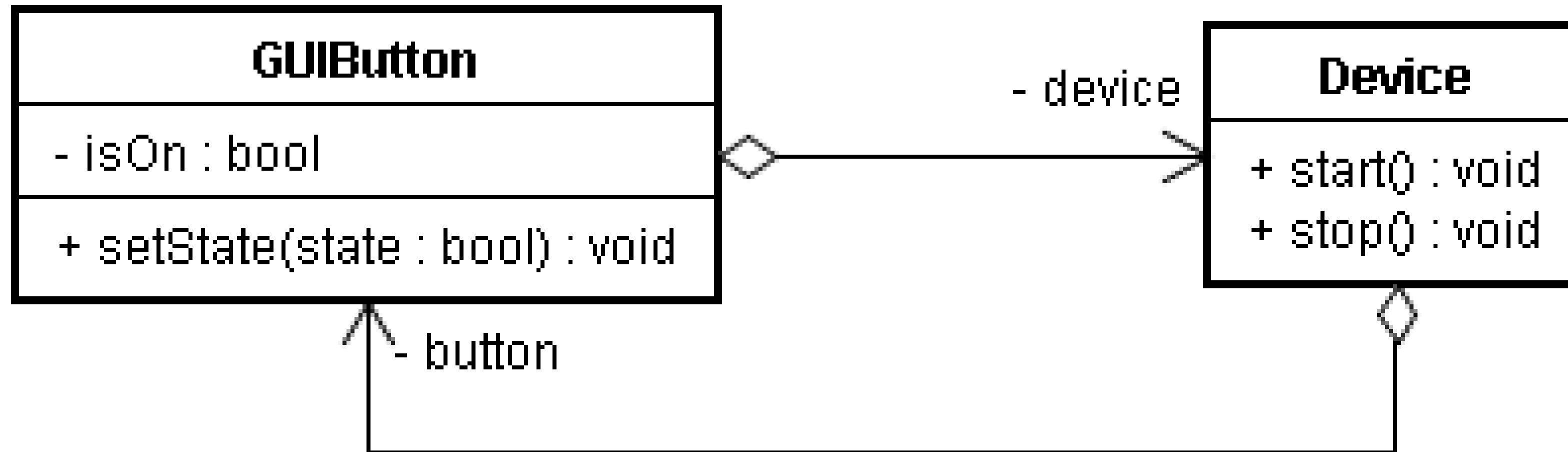
+ позволяет легко менять сервера, даже во время выполнения программы

Adapter : шаблон



Вариант, позволяющий перегрузить методы сервера

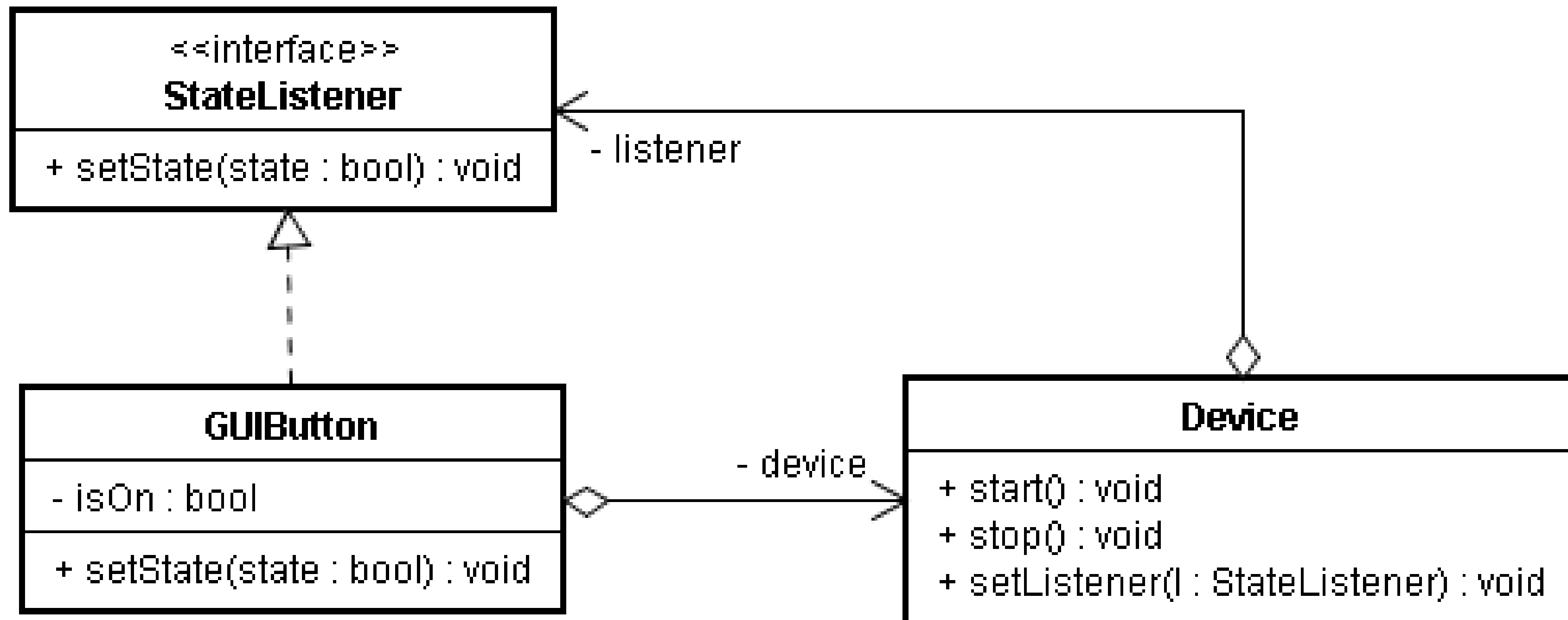
Abstract Client : проблема



Проблема:

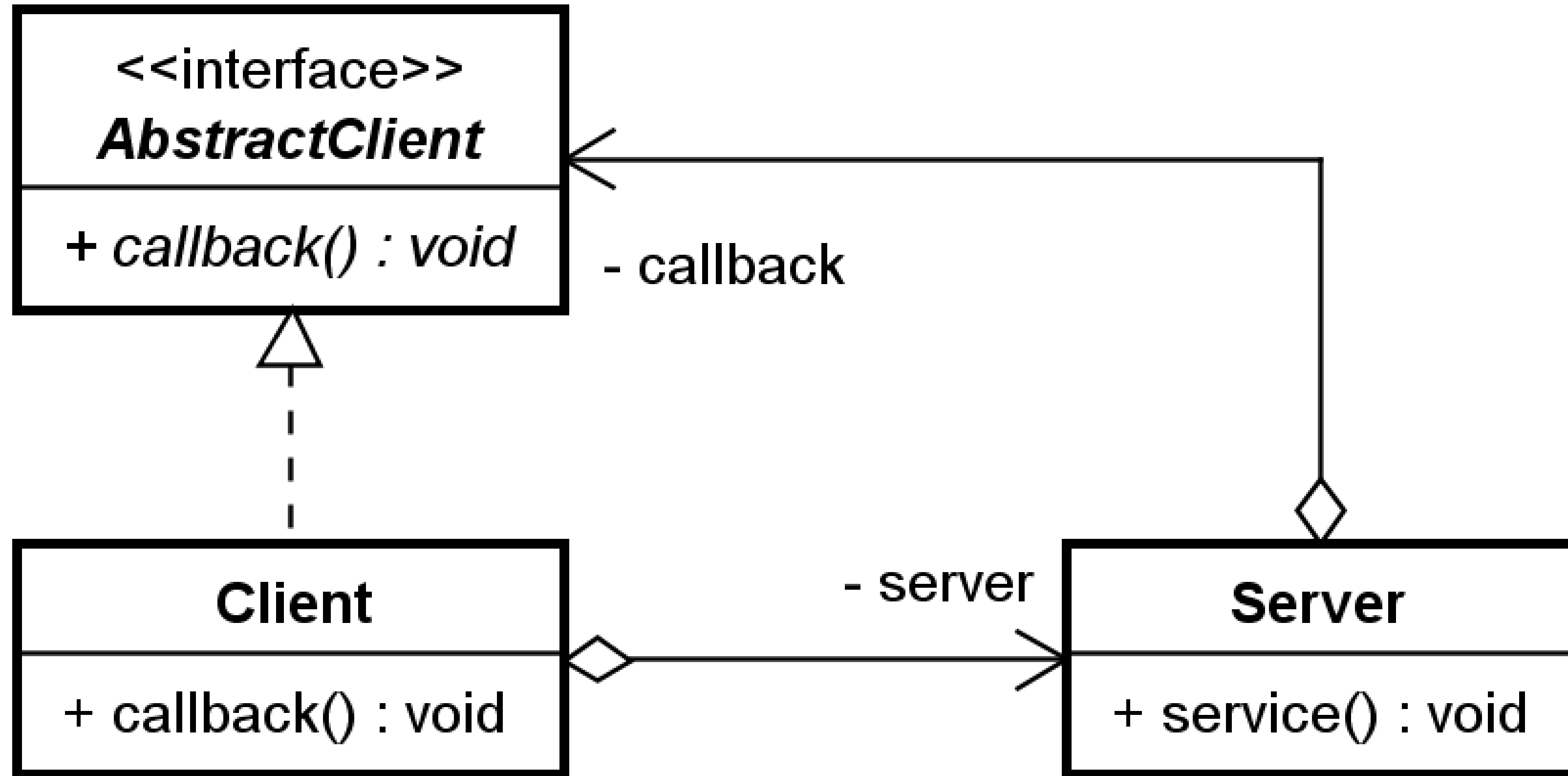
- Device нужно послать сообщение GUIButton, когда Device был включен/выключен извне, чтобы кнопка отобразила состояние
- GUIButton невозможно переиспользовать
- Callback метод делает Device также непереиспользуемым

Abstract Client : решение



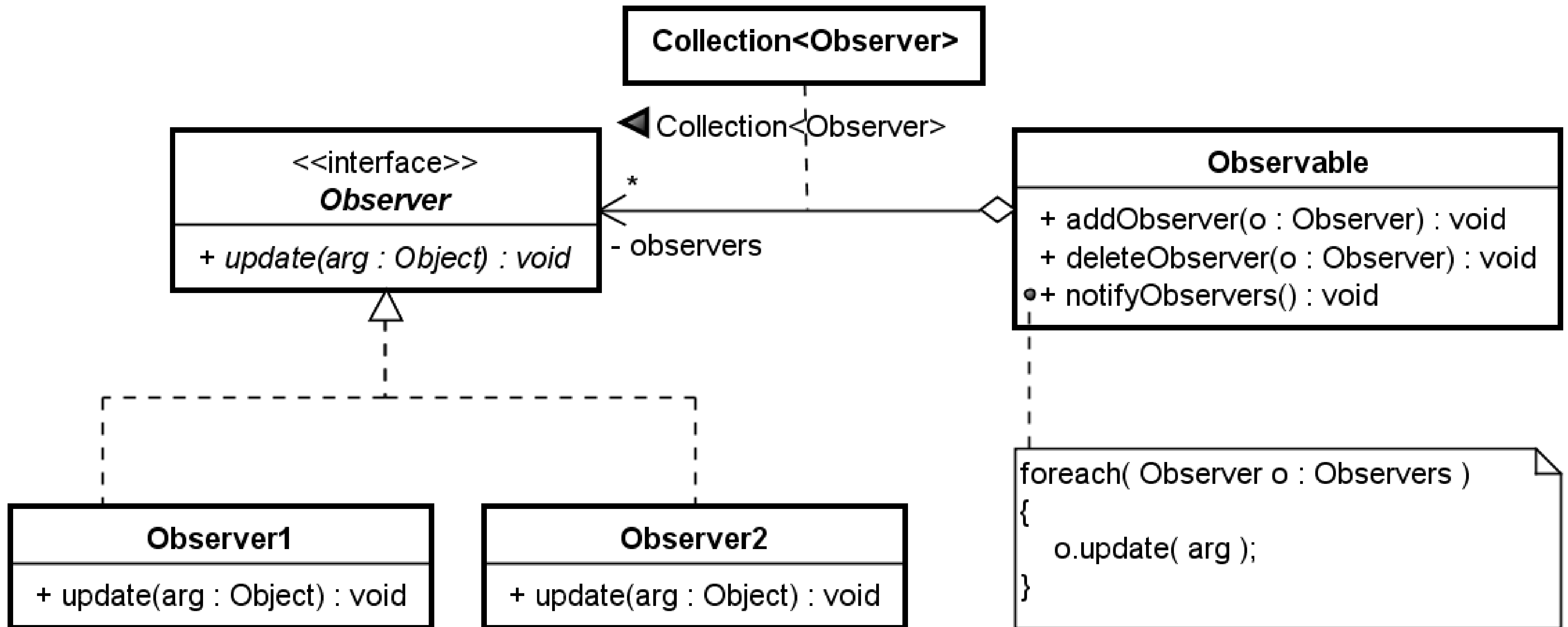
✓ Server (Device) предоставляет клиенту interface для сообщений

Abstract Client : шаблон



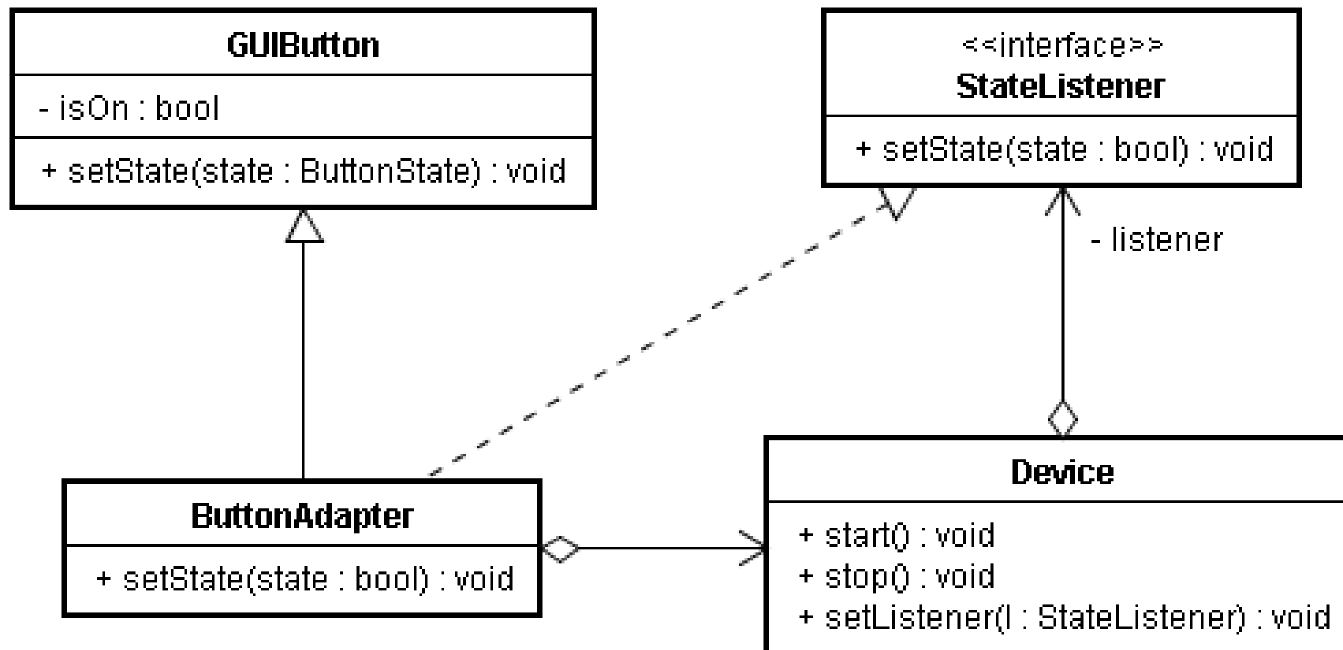
- **Q:** В каком пакете должен находиться AbstractClient ?
- **A:** Конечно, в пакете Server-а. Но почему?

Observer : шаблон



Похож на Abstract Client, немного более универсален

Adapted Client



Проблема:

GUIButton уже существует и не реализует StateListener

Решение:

использовать **Adapter** совместно с **Abstract Client**

Singleton

Проблема: Как много объектов класса Database (Logger, Profiler) требуется в программе?

- Чаще всего, только один
- Скорее всего, если кто-то создаст еще один такой объект – будут сюрпризы

Решение: Не давать возможности создавать такие объекты

- сделать конструктор приватным
- сделать сам класс ответственным за создание объектов своего типа, тем самым гарантировав, что никто не создаст больше одного объекта

Q: Какие еще есть способы решения этой проблемы?

Singleton

MyDatabase
- db : MyDatabase
- MyDatabase() + getInstance() : MyDatabase + findUser(name : String) : User

```
class MyDatabase {
  private MyDatabase() { }

  private static class SingletonHelper {
    private static final MyDatabase db = new MyDatabase();
  }

  public static MyDatabase getInstance() {
   return SingletonHelper.db;
  }

  public User findUser( String name ) { … }
}
```

Singleton используется в случаях, когда:

- в системе должен существовать только один объект класса
- этот объект должен быть доступен из любого места программы

User user = MyDatabase.getInstance().findUser("Alex");

© V.Mukhortov, NSU, 2023

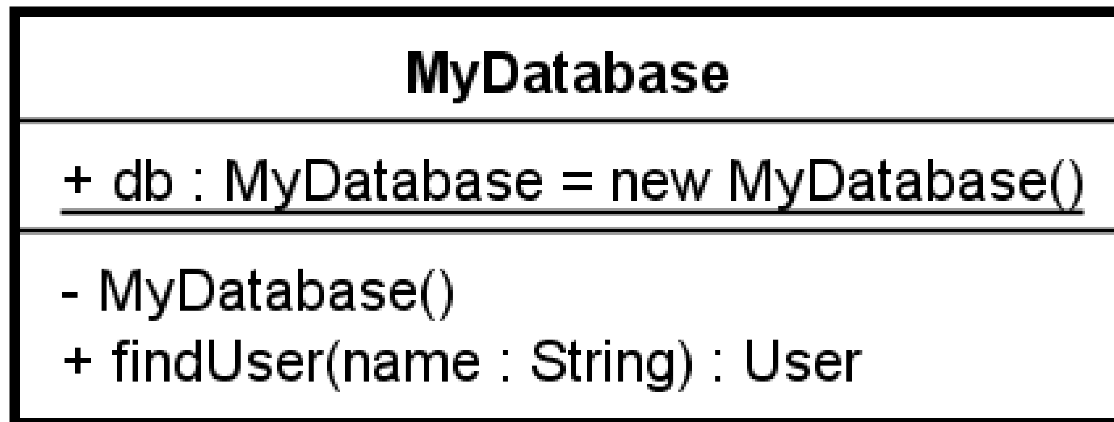
Singleton: C++

- https://www.fluentcpp.com/2018/03/06/issues-singletons-signals/
- https://habr.com/ru/post/455848/

Monostate

- ✓ Решает ту же проблему, что и Singleton
- ✓ Все поля - static
- ✓ Constructor и destructor - private

User user = MyDatabase.db.findUser("Alex");



Singleton vs. Monostate

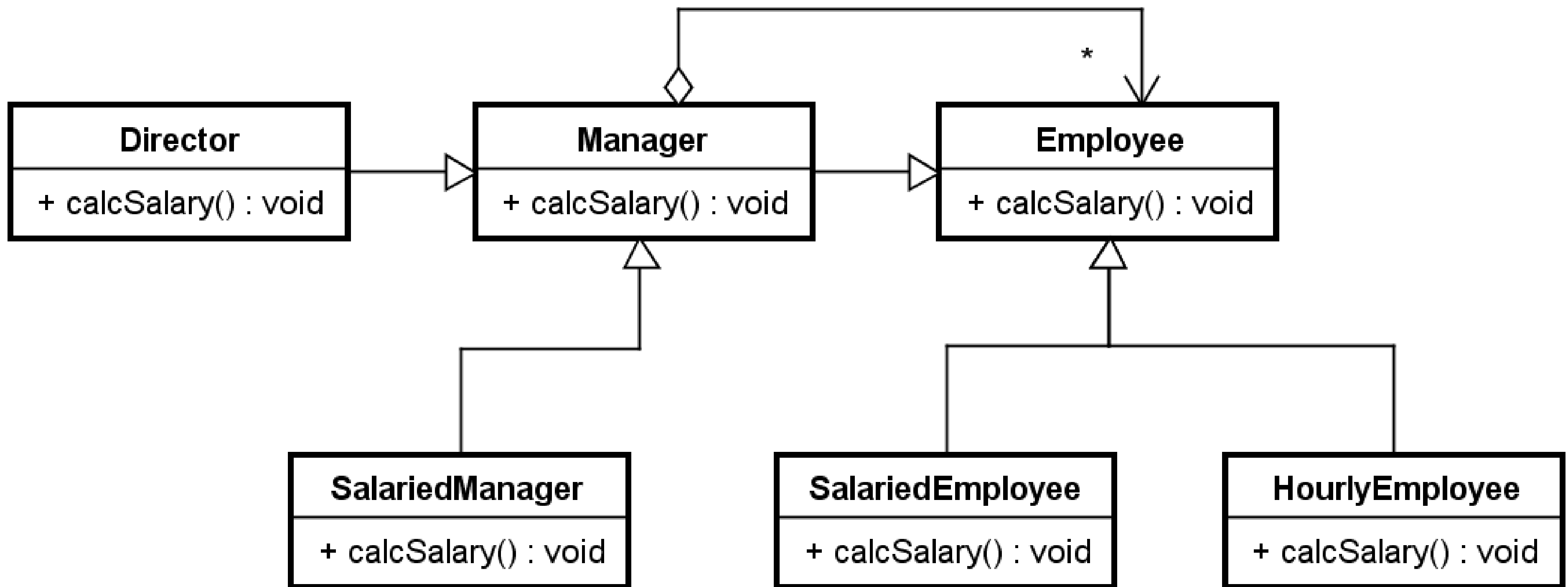
<u>Конструирование</u>:

- Singleton: отложеное конструирование (lazy construction)
  - Don’t pay unless you need it!
- Monostate - сконструирован всегда
- Singleton может иметь нетривиальный конструктор
- Monostate не может иметь сложного конструктора

<u>Деструкция</u>:

- Singleton - по запросу
- Monostate – по завершению программы

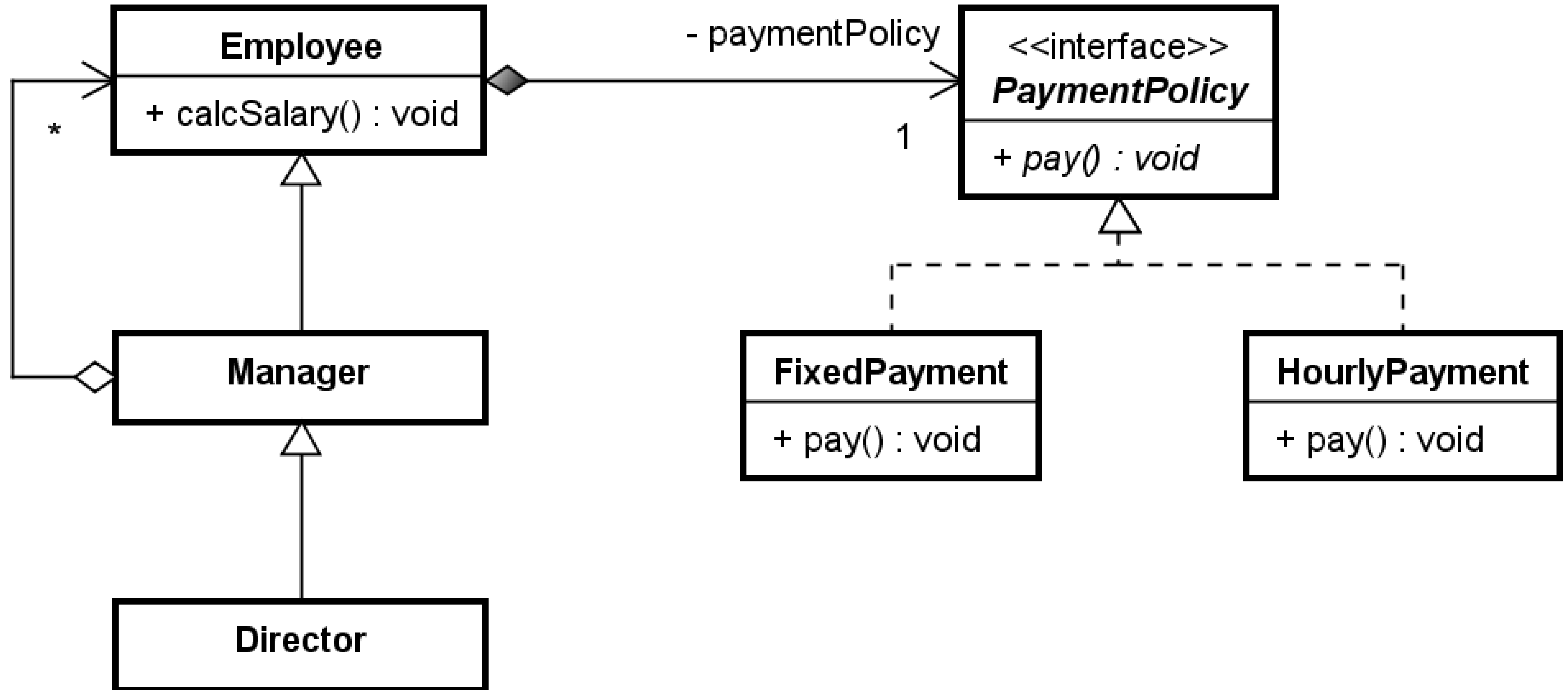
Strategy : Проблема



Проблема:

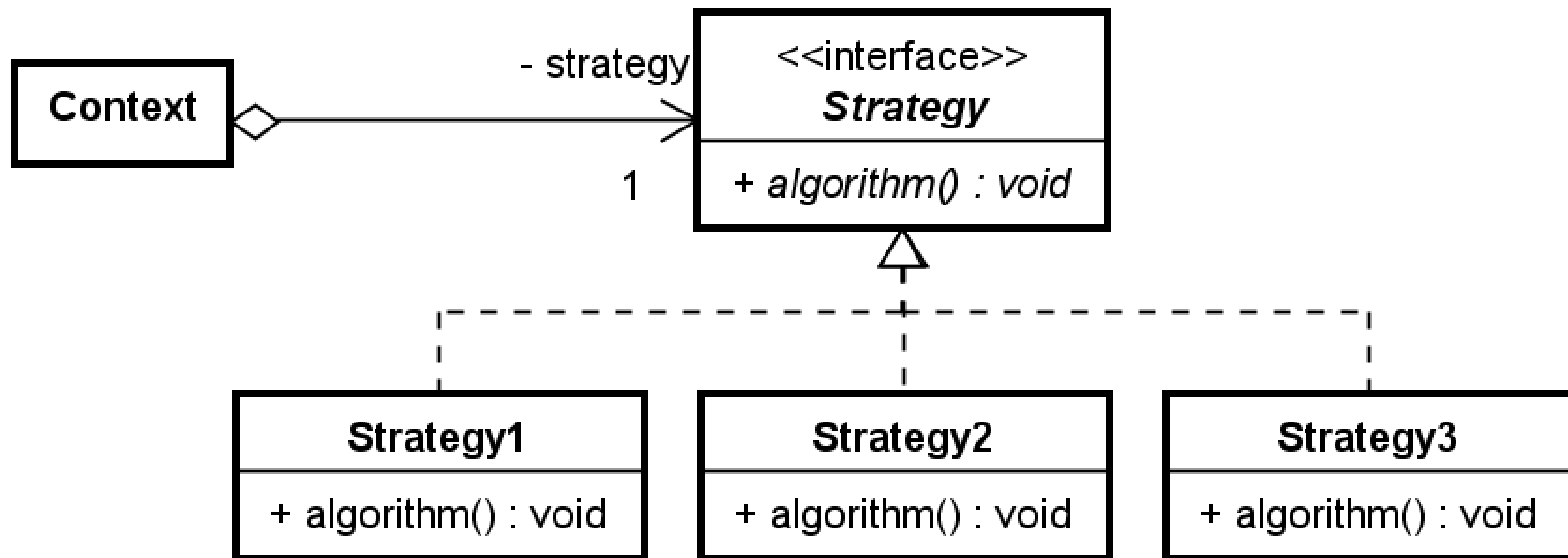
- различные сотрудники оплачиваются по разному
- как добавить hourly-paid manager ?

Strategy : Решение



- ✓ Алгоритм начисления зарплаты вынесен в PaymentPolicy

Strategy : шаблон

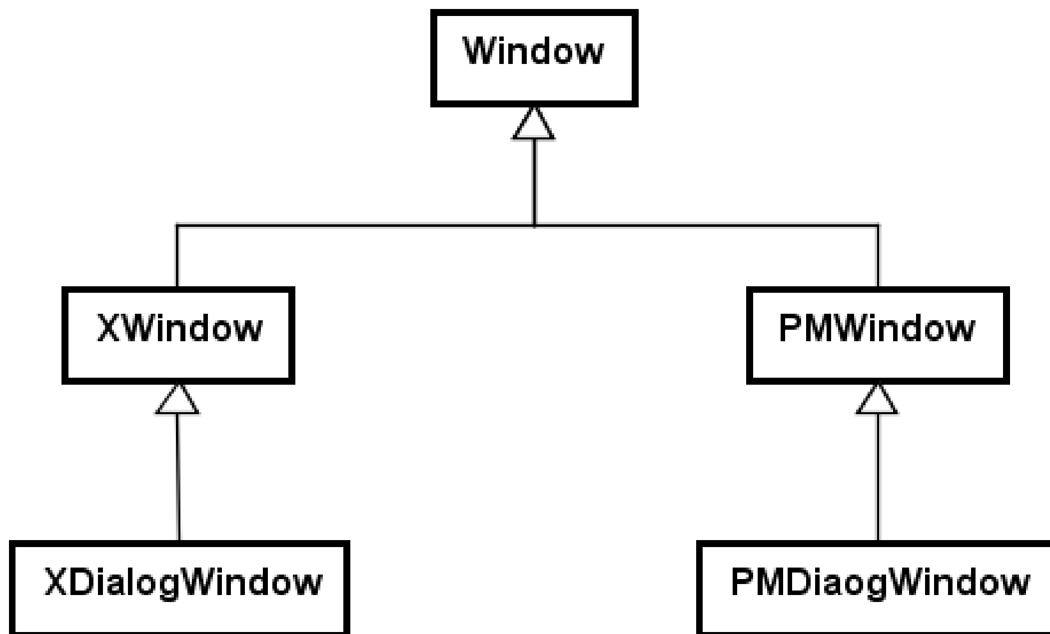


Также известен как: **Policy**

Плюсы:

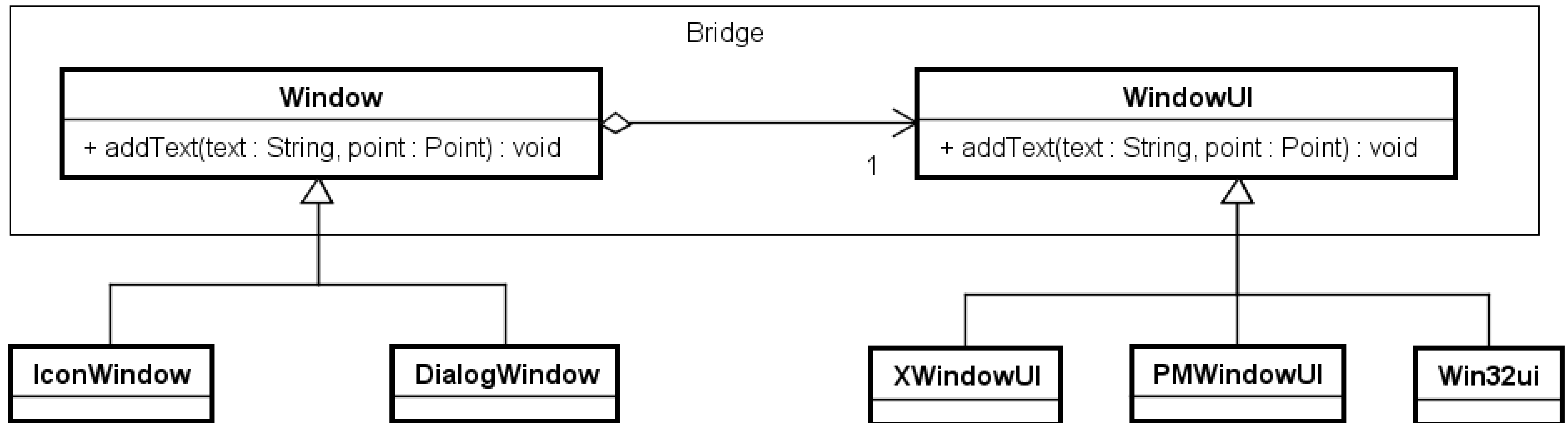
- стратегии можно менять во время исполнения
- Context закрыт от модификации стратегий (выполняется OCP)

Bridge: Проблема



Проблема:
- Код зависит от платформы
- Для поддержки новой платформы надо воспроизвести всю иерархию

Bridge: Решение



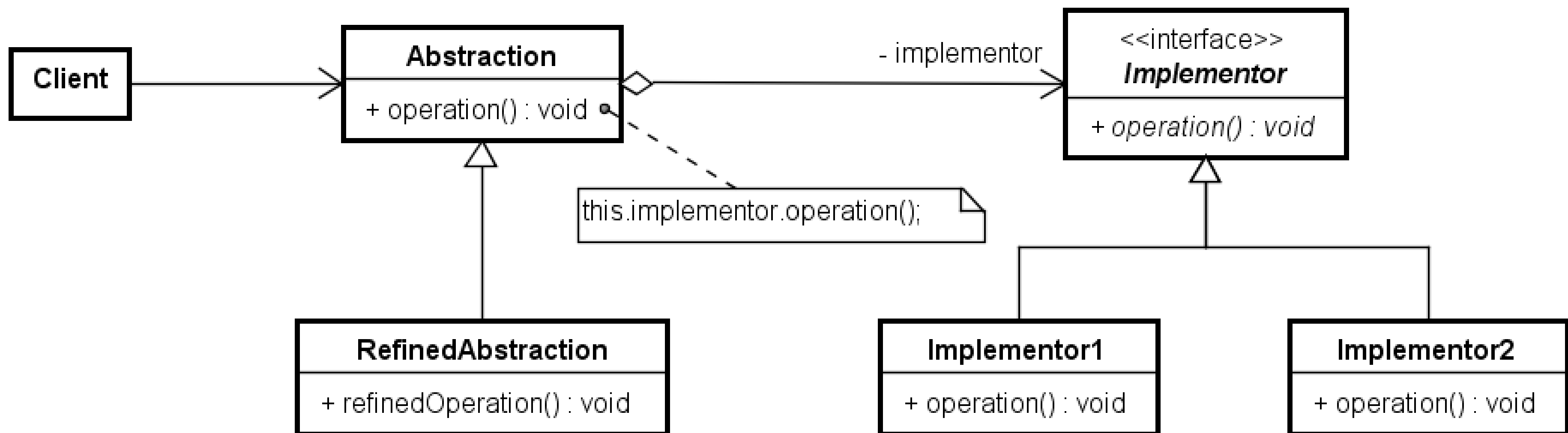
✓ Все операции подклассов Window реализованы в терминах операций из WindowUI

✓ Window и его подклассы платформно-независимы

Q: Как сделать код приложения независимым от подклассов WindowUI?

A: использовать фабрику

Bridge: шаблон

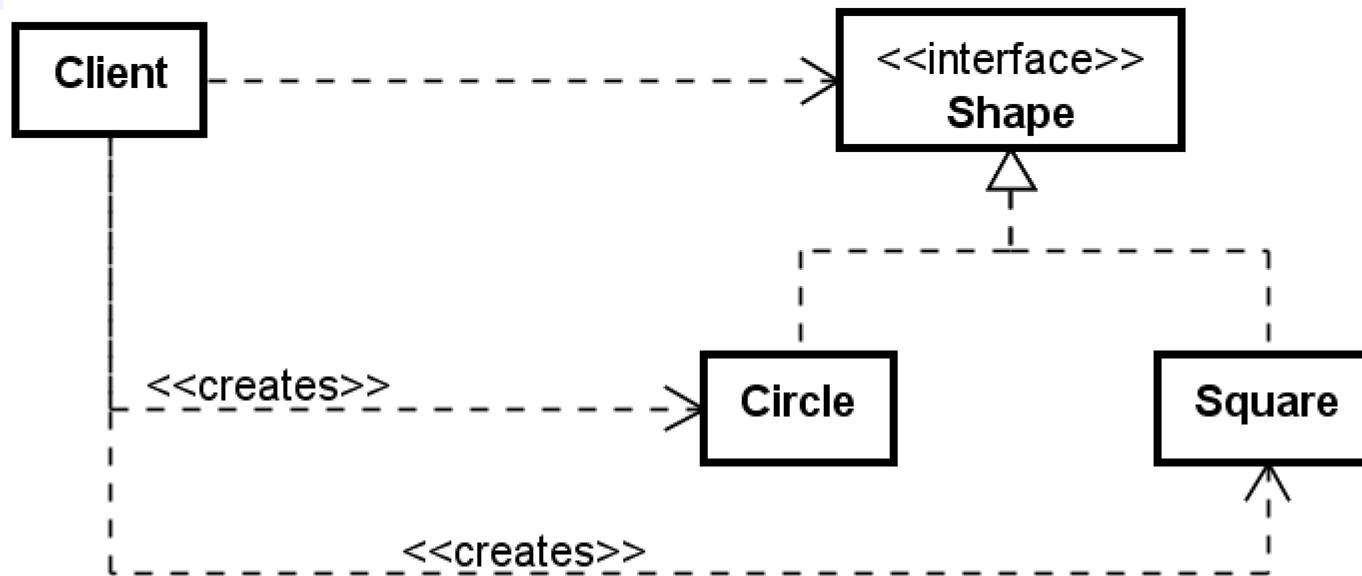


Также известен как: Handle/Body

Плюсы:

- Устраняет нарушение DIP & OCP

Abstract Factory

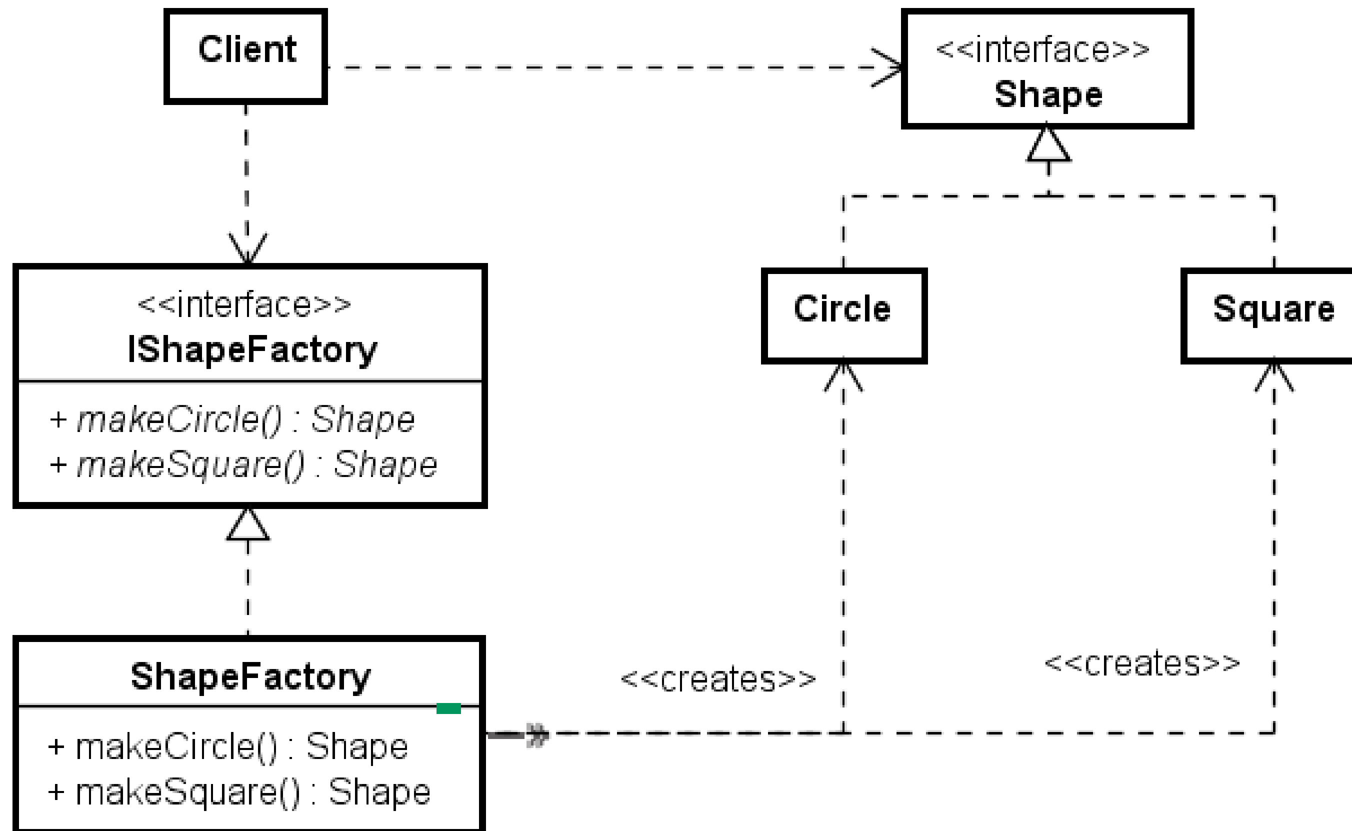


Проблема:

- создание объектов есть зависимость от производных типов
- но все прочие действия делаются через интерфейс
- что мы выиграли от использования интерфейса Shape?

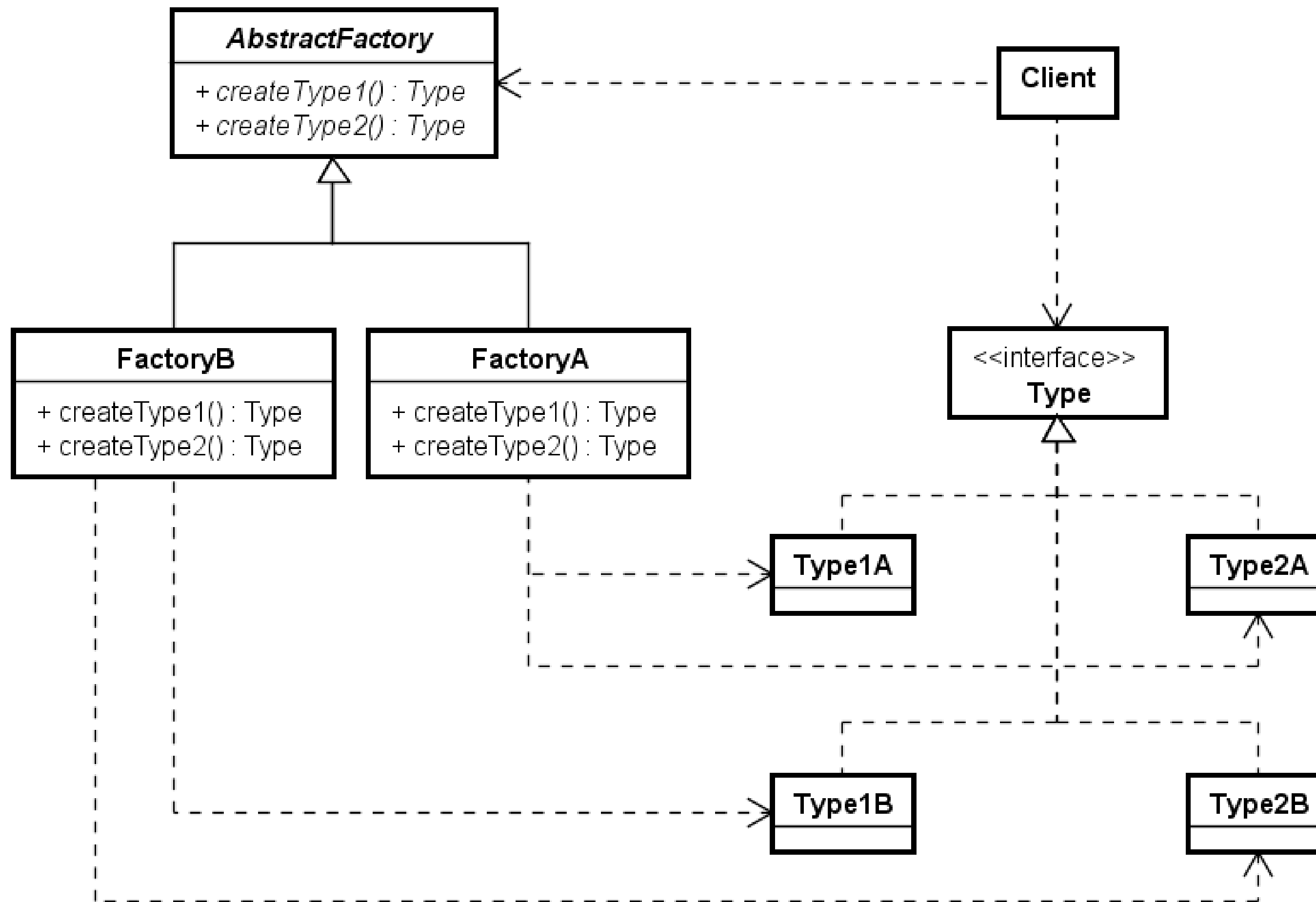
Ведь Client все равно будет очень нестабильным.

Abstract Factory : решение



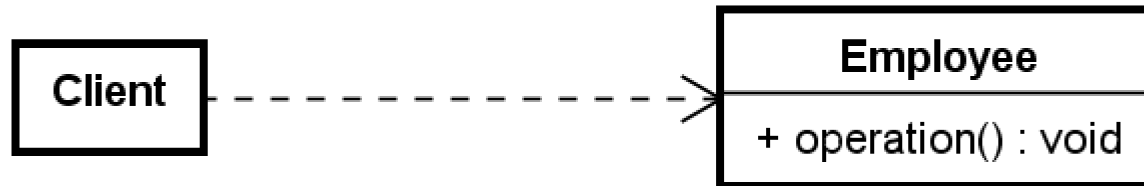
Создать класс, отвечающий за создание объектов.
Избежать создания объектов нельзя, но можно локализовать.

Abstract Factory : шаблон



Изолирует конкретные типы данных, упрощает внесение изменений

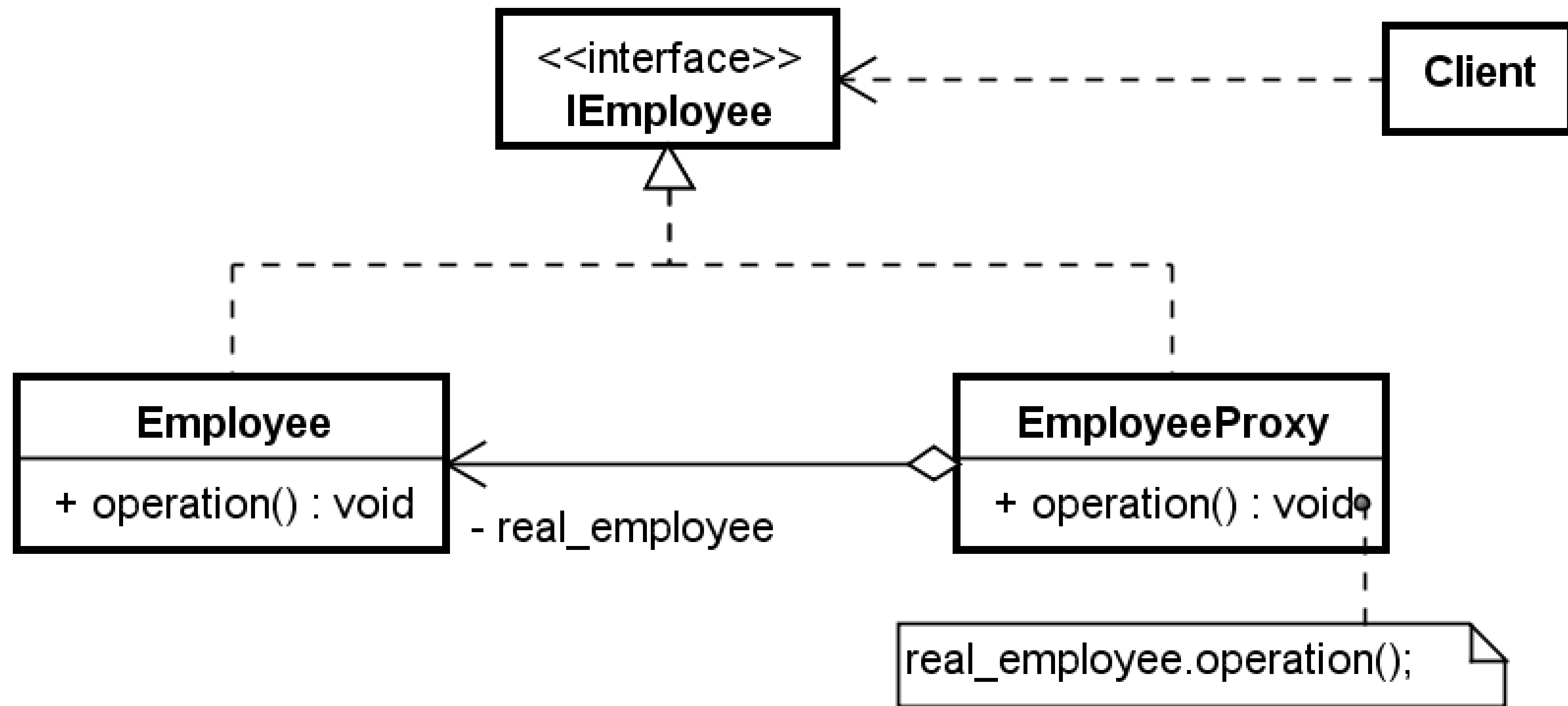
Proxy



Проблемы:

- Нужно контролировать доступ к объектам (проверять права)
- Нужно выполнять дополнительные действия при доступе (напр. логгировать факт доступа)
- Нужно обеспечить удаленный доступ к объекту

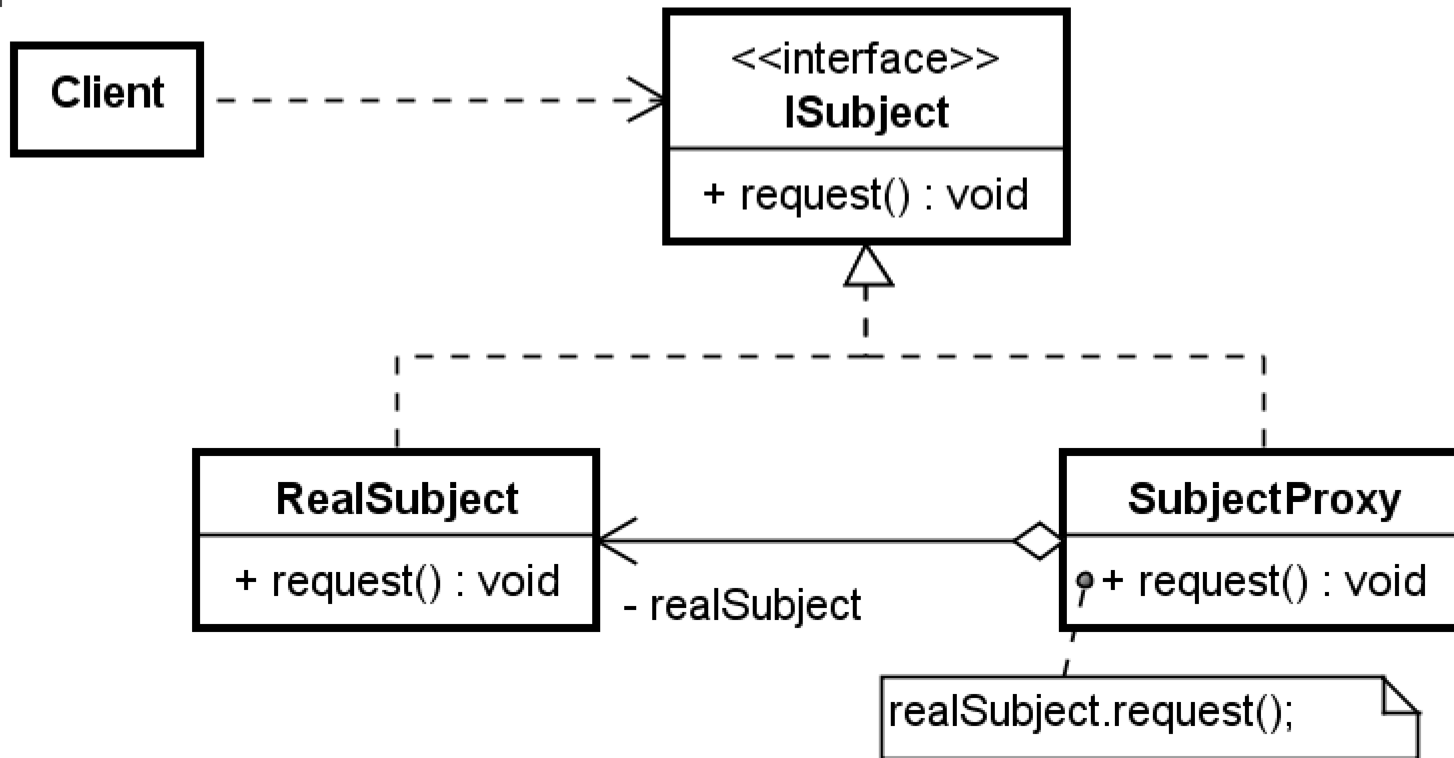
Решение



Решение:

- использовать суррогат, который контролирует доступ
- клиенты могут считать что работают с Employee, т.к proxy реализует тот же самый интерфейс

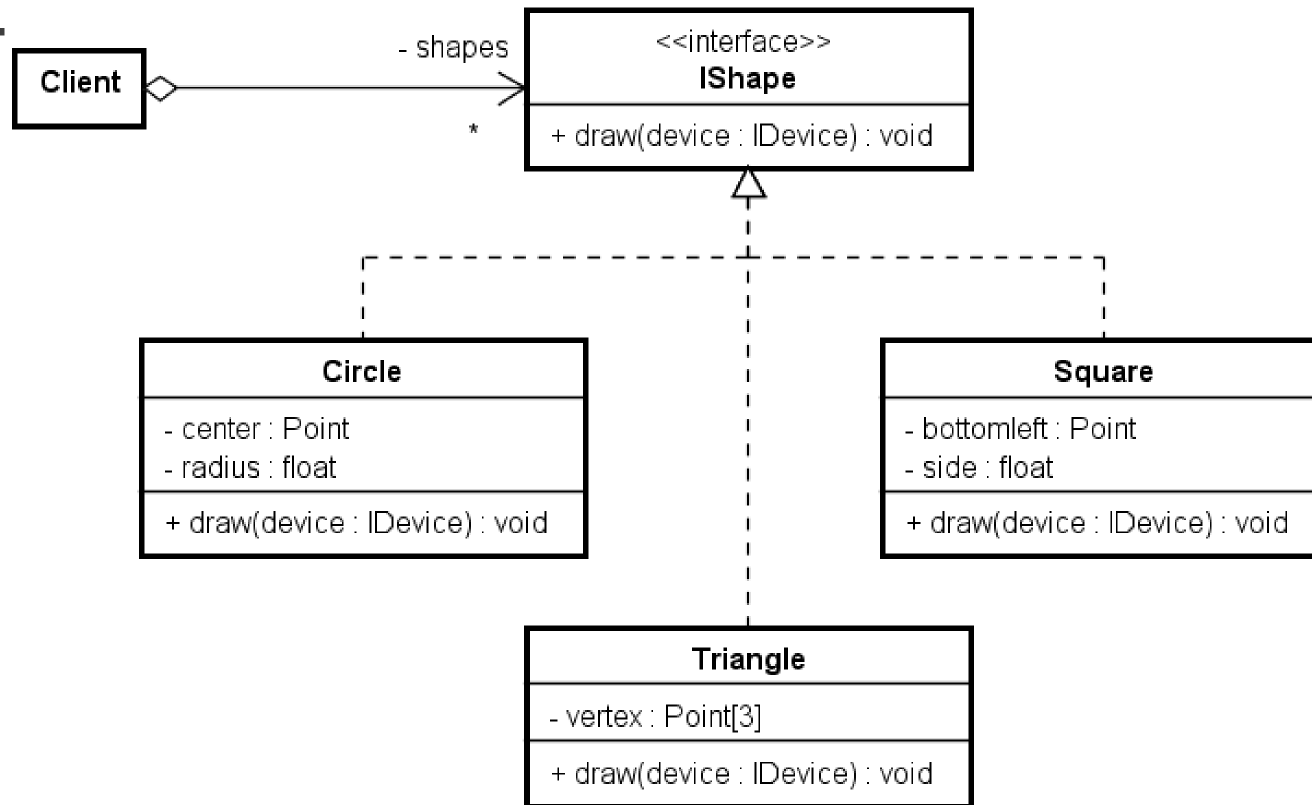
Proxy Pattern



Также известен как: Surrogate

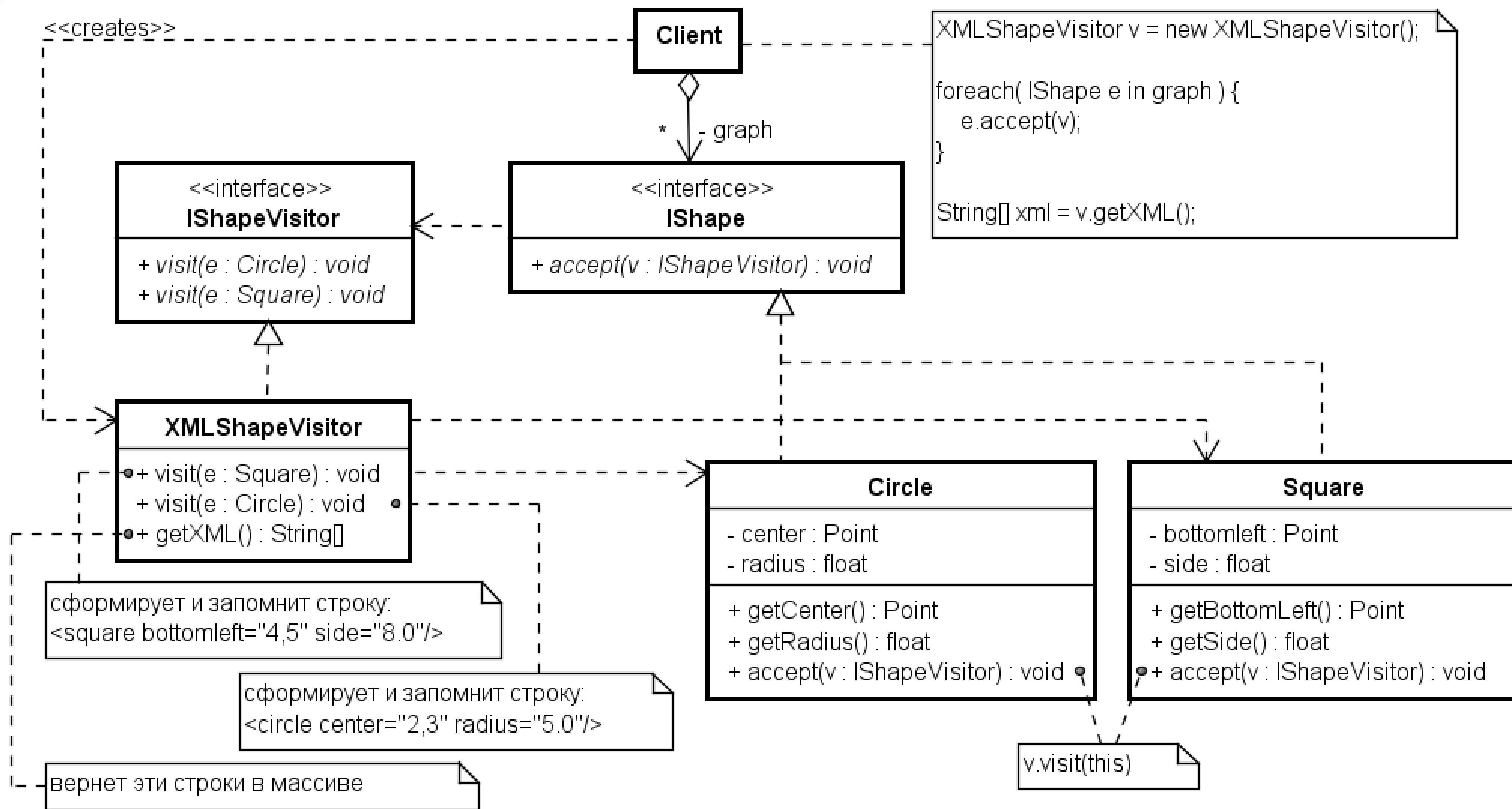
Применимость: remote proxy, virtual proxy (создание реальных объектов «на лету»), контроль доступа

Visitor

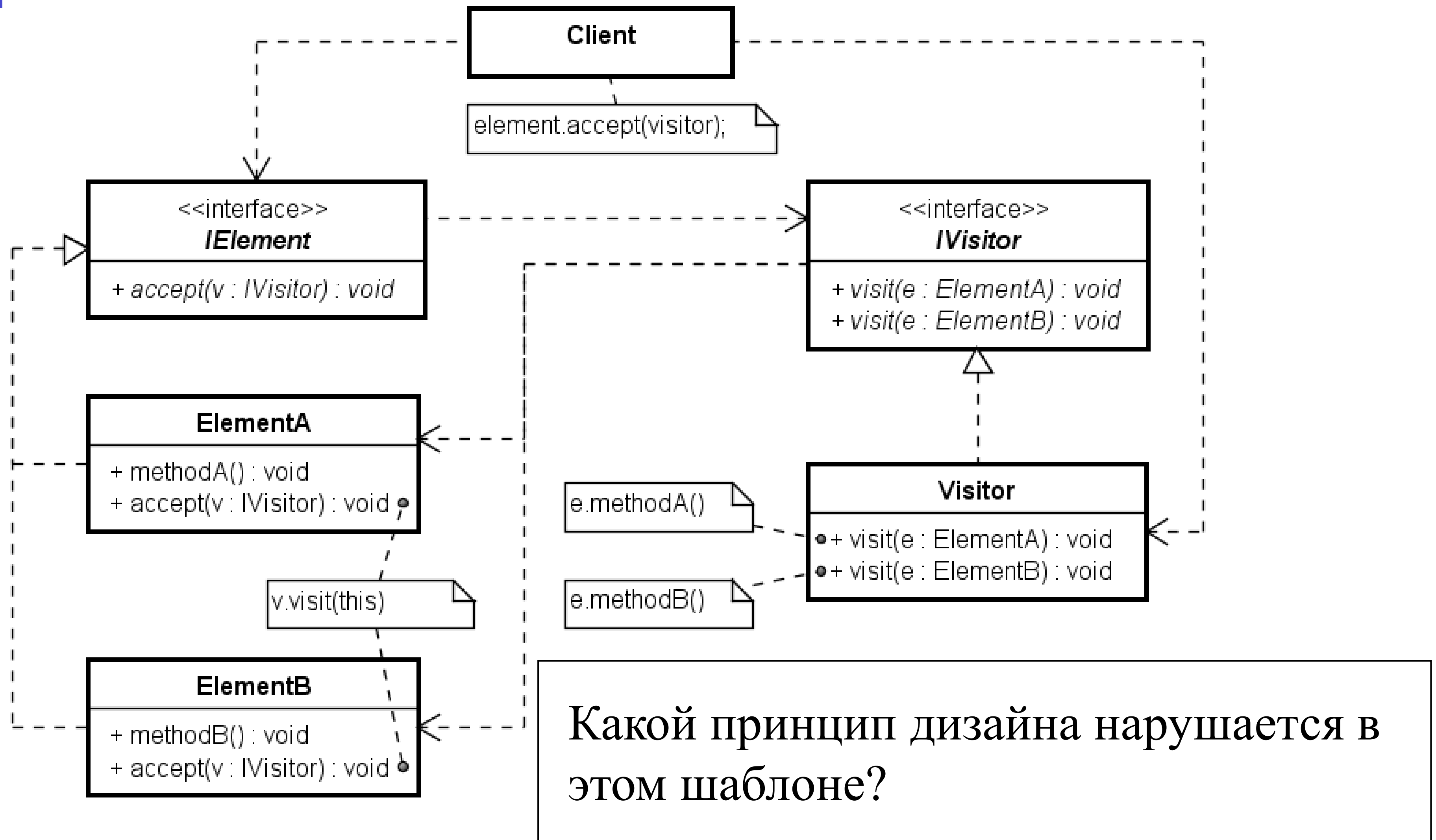


Проблема:
- нужно предоставить возможность реализовывать различные алгоритмы обработки коллекции объектов IShape, напр. сохранение в XML
- при этом, мы не хотим всякий раз добавлять метод в интерфейс

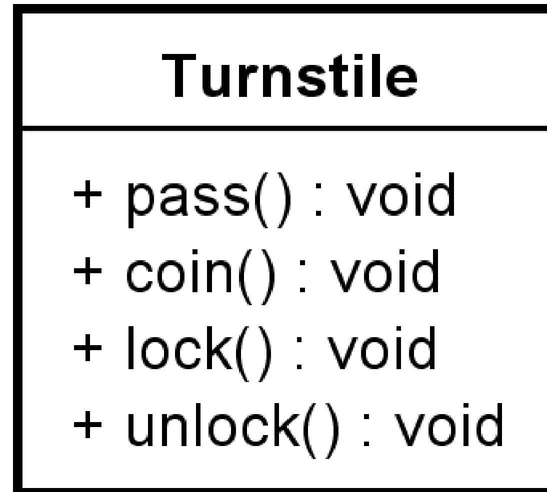
Visitor: решение



Visitor: pattern



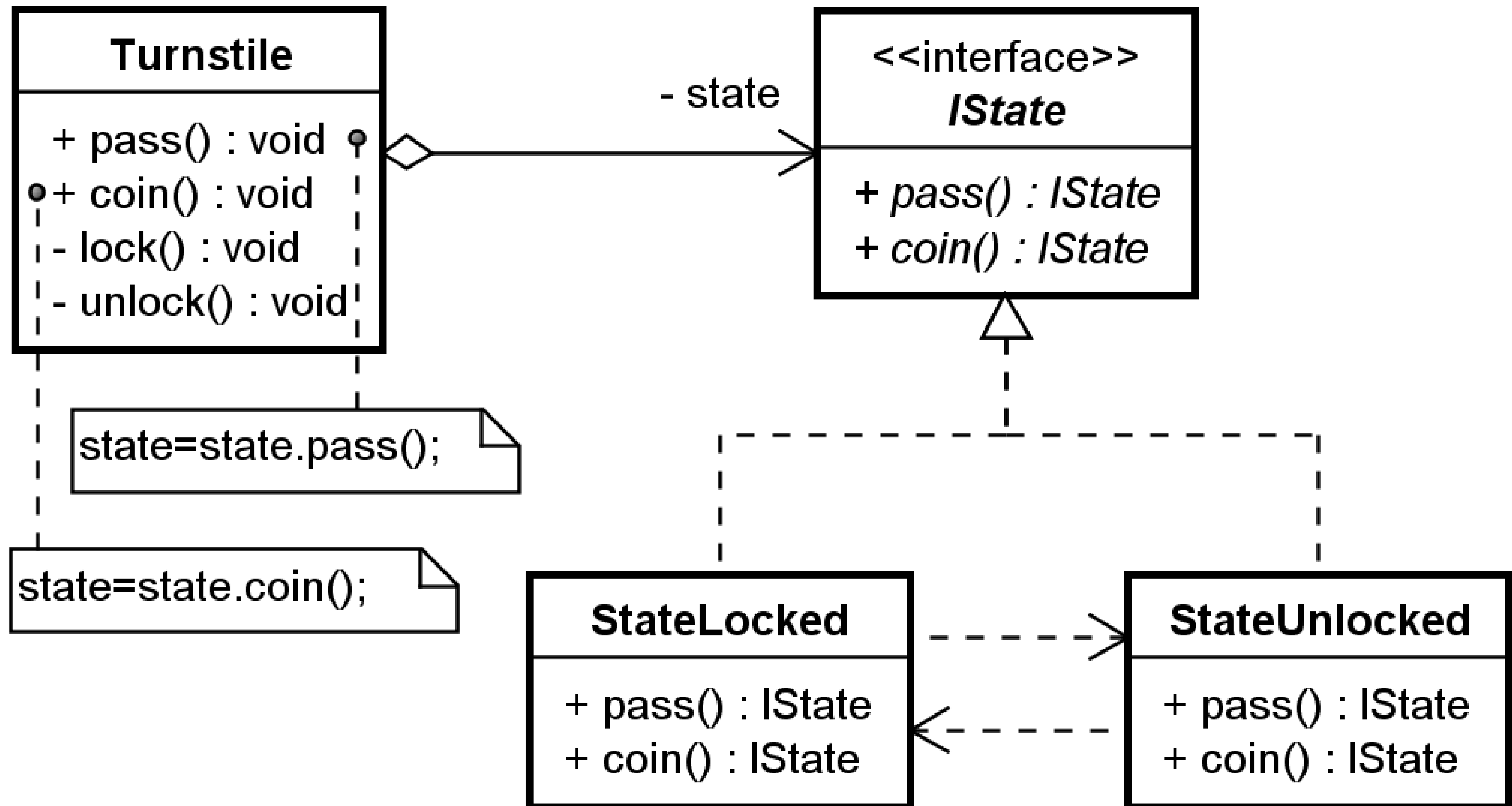
State machine



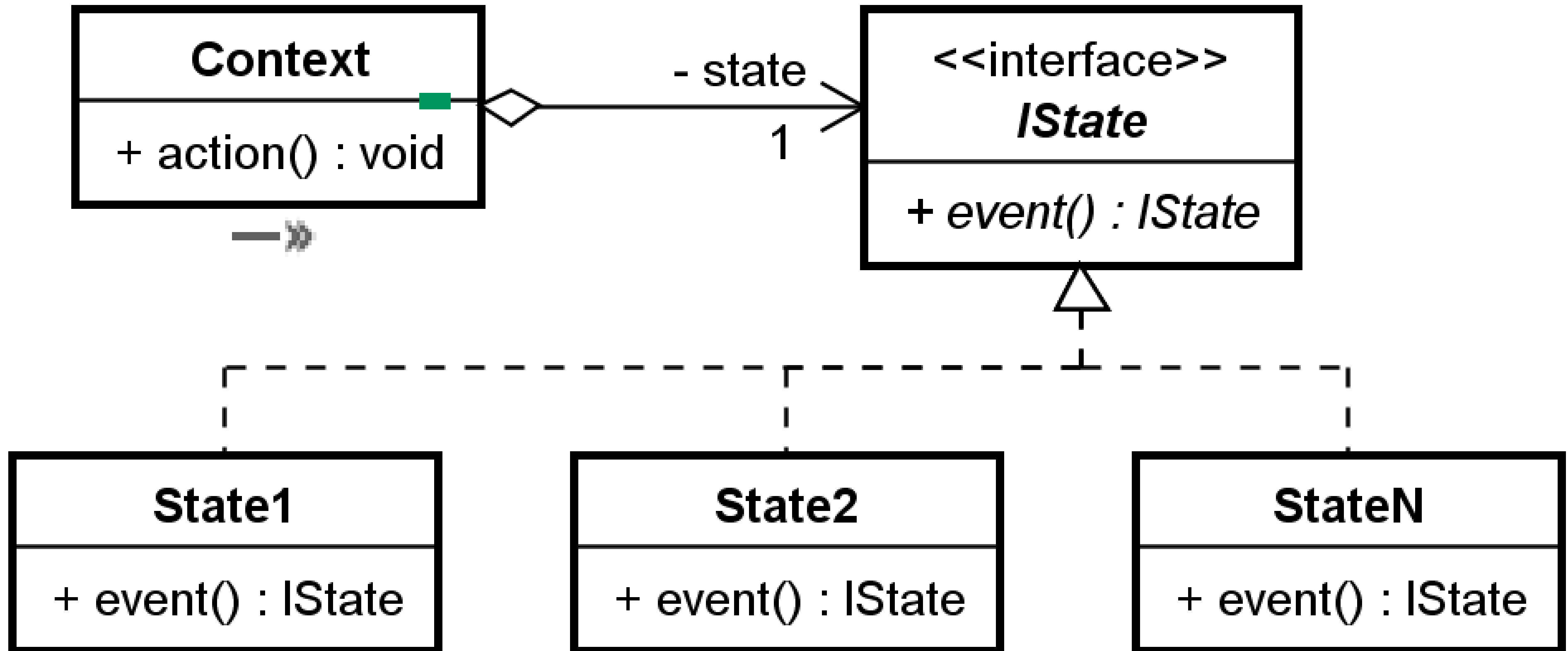
State	Event	Action	New state
Locked	Coin	Unlock	Unlocked
Locked	Pass	Alarm	Locked
Unlocked	Coin	Return	Unlocked
Unlocked	Pass	Lock	Locked

Как реализовать такой автомат? If-else в каждом методе? А если состояний больше, чем два?

State: решение



State: pattern

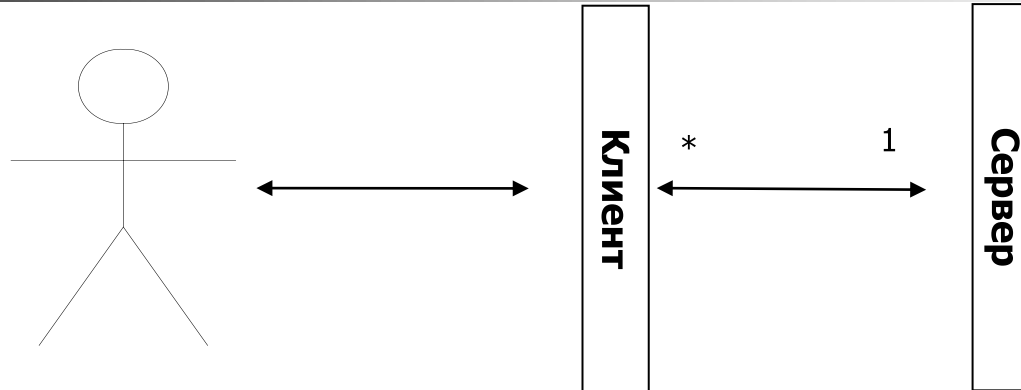


Похож на шаблон Стратегия, но в чем отличие?

8. Архитектурные шаблоны

- Client/server : Клиент/Сервер
- N-tier : Многоуровневая архитектура
- Peer-to-peer (P2P) : Одноранговая сеть
- Pipes and filters : Каналы и фильтры
- ACL security : Списки контроля доступа
- MVC (Model-View-Controller) : Модель-Представление-Управление

Архитектура клиент/сервер



Характерные черты:

- Единый сервер
- Отсутствие прямого взаимодействия между клиентами

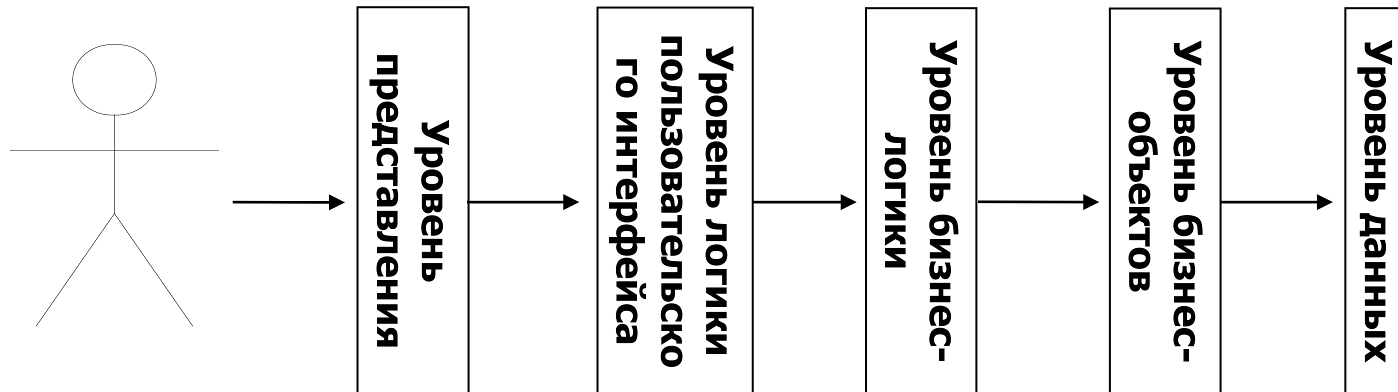
Преимущества:

- Централизованное обслуживание и защита данных

Недостатки:

- Сервер может оказаться тонким местом системы
- Требует администратора

Многоуровневая архитектура (N-tier)



Характерные черты:

- каждый слой предоставляет сервисы следующему и использует сервисы предыдущего
- взаимодействуют только соседние слои
- каждый слой реализует четко определенную часть функциональности

Преимущества:

- возможность независимой разработки
- сужение набора необходимых для создания каждого слоя знаний

Многоуровневая архитектура клиент-сервер

Разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов

Преимущества:

- Разравнивание нагрузки
- Более высокая защищенность данных

Частные случаи:

- Трехуровневая архитектура
  - Клиент
  - Сервер приложений
  - Сервер баз данных

MSA - Микросервисная архитектура

Разновидность архитектуры клиент-сервер, в которой функция обработки входящих запросов распределена между несколькими специализованными узлами

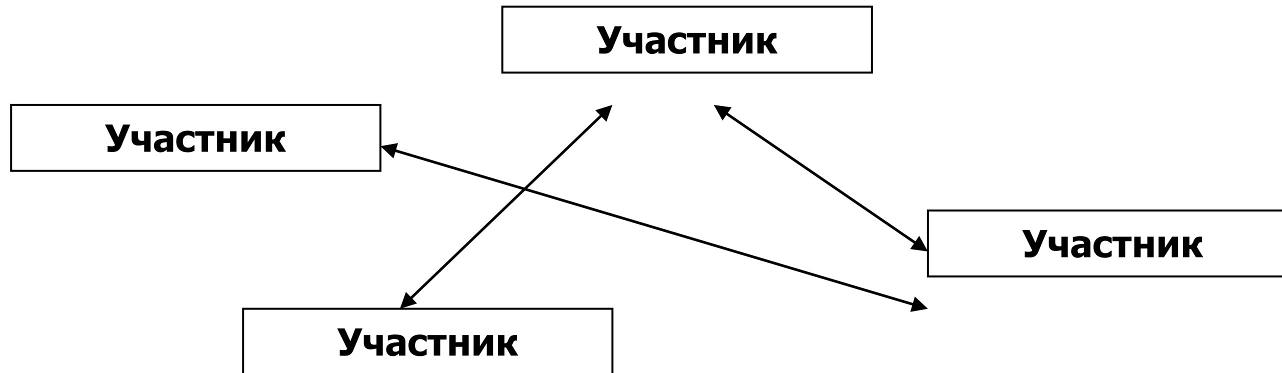
Преимущества:

- Высокая связность внутри сервисов и низкое зацепление между сервисами
- Возможность более гибкого разравнивания нагрузки
- Более высокая защищенность данных

Последствия:

- Усложняется обработка ошибок
- Бизнес логика, реализуемая с участием более одного сервиса, потребует управления распределенными транзакциями
- Поддержка немыслима без CI/CD

Одноранговая сеть



Характерные черты:

- отсутствие центрального сервера
- равные права участников

Преимущества:

- Надежность.
  - Отключение любого количества узлов не мешает работе других.

Недостатки:

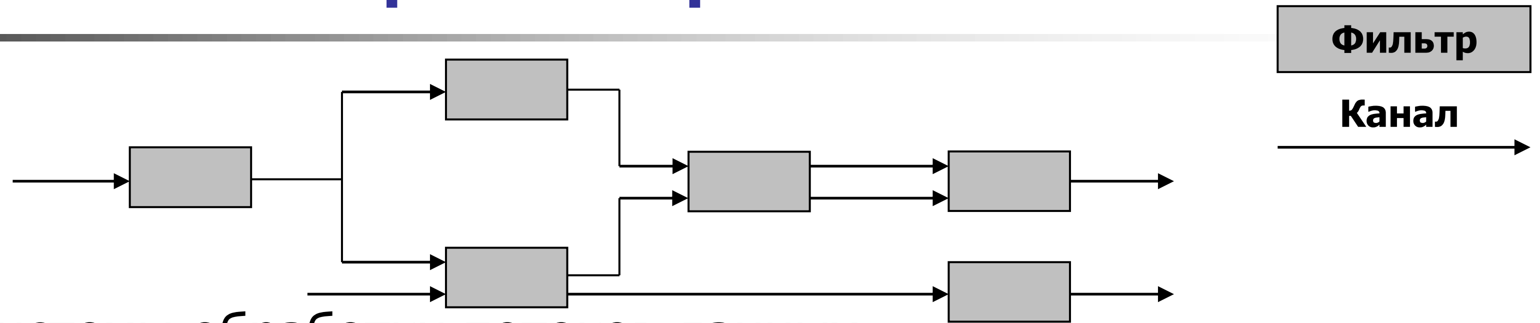
- Сложность распространения изменений (updates)

© V.Mukhortov, NSU, 2023

Применение

- Collaborative Computing
- Instant Messaging (совместно с Client-Server)
- P2P сети (UseNet news, SMTP, файлообменники)
- Simple file sharing (MS Windows network w/o domain)
- фантастика: сеть нонитов в StarGate SG-1
- уже не фантастика: рой беспилотников, решающих общую задачу
- Майнинг криптовалют

Каналы и фильтры



<u>Применение</u>: системы обработки потоков данных

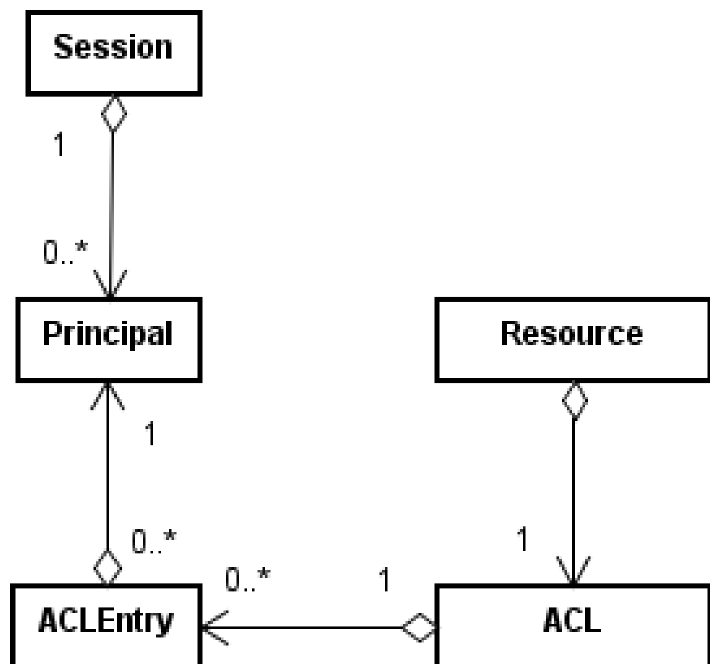
<u>Ситуации</u>:

- Система создается разными разработчиками (возможно, часть системы уже существует)
- Задача естественно разбивается на независимые этапы обработки
- Алгоритм может меняться во время работы системы, требуя отключения одних и подключения других фильтров

<u>Характерные черты</u>:

- Фильтры – модули, получающие на вход поток(и) данных и выдающие поток(и) данных
- Простота конфигурирования системы фильтров (соединение их с помощью каналов)
- Фильтры работают параллельно

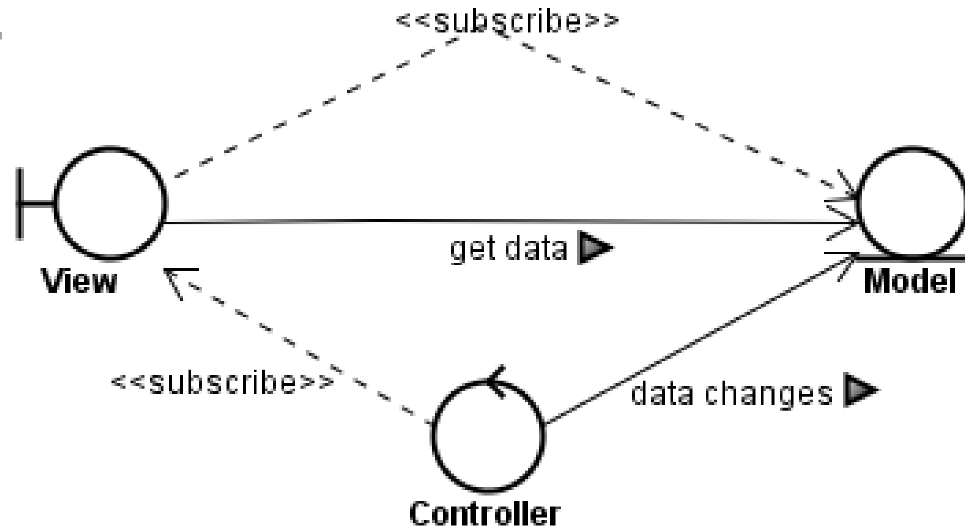
Списки контроля доступа (ACL)



- Session – сессия пользователя
- Principal – некоторый идентификатор пользователя, которому можно поставить в соответствие права доступа (например uid, gid, etc)
- Resource – ресурс, доступ к которому ограничен
- ACL – список контроля доступа к ресурсу
- ACLEntry – элемент списка контроля доступа, может содержать уровень доступа

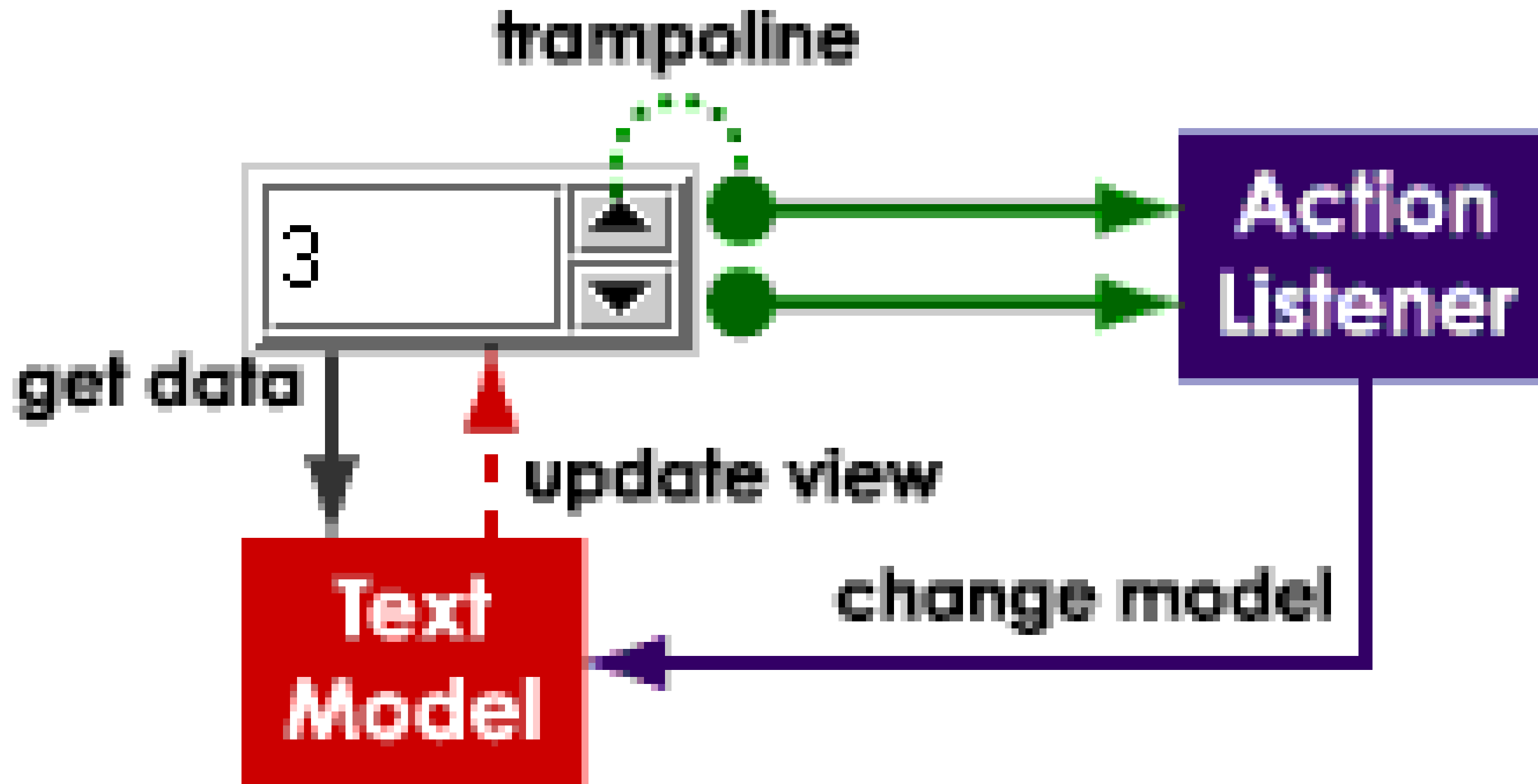
При каждом обращении к ресурсу производится проверка, есть ли у пользователя principal, которому разрешен доступ к ресурсу с требуемым уровнем доступа

Model-View-Controller



- Разделяет представление, данные и логику, и обработку событий UI
- Model: информация, бизнес-правила. Предоставляет данные и реагирует на команды контроллера
- View: элементы UI, получающие данные из модели и реагирующие на изменения модели
- Controller: обработка событий UI, часто через callback, передача изменений в модель
- Q1: где располагается бизнес-логика?
- Q2: где производится проверка введенных пользователем данных?

MVC



Подвиды MVC

- MVC
  - Тонкий контроллер, логика в модели
  - Толстый контроллер, модель только манипулирует данными
- MVP : Model – View – Presenter
  - Model – бизнес-логика + данные
  - View – отображение данных, создает Presenter и передает ему ссылку на свой интерфейс
  - Presenter – связывает Model и View, передает View данные из Model
- MVVM : Model – View – ViewModel (напр. Microsoft WPF)
  - View – графический интерфейс
  - Model – как в MVC
  - ViewModel – двойной адаптер: абстракция View для Model и наоборот

9. Rational Unified Process

Phases

Inception | Elaboration | Construction | Transition

Workflows

Business Modeling

Requirements

Analysis & Design

Implementation

Test

Deployment

Configuration & Change Mgmt

Project Management

Environment

Initial | Elab #1 | Elab #2 | Const #1 | Const #2 | Const #N | Tran #1 | Tran #2

Iterations

- Vision – представляет собой общее описание проекта и является базисом для уточнения требований к системе

- Содержит:
  - Цели проекта
  - Stakeholders & Users *(описание инициаторов проекта и конечных пользователей)*
  - Перспективы и возможности системы
  - Особенности
  - Ограничения

RUP: Business modeling

- Задачи:
  - Идентификация бизнес-процессов (business use-cases)
  - Идентификация бизнес-акторов и сущностей (business entity)
  - Улучшение (refine) бизнес-процессов

- Модели:
  - business use-case model
  - business object model

© V.Mukhortov, NSU, 2023

RUP: Требования (Requirements)

Задачи:

- сбор и анализ требований к системе
- классификация use-cases
- оценки затрат и рисков

Модели:

- Use-case model

SRS (Спецификация требований)

- SRS (Software Requirements Specification) - полностью определяет требования к системе, зависит от Vision

- Содержит:
  - Функциональные требования (что должна делать система, роли пользователей, фактически, описание use-cases)
  - Нефункциональные требования (производительность, ограничения по используемым технологиям и т.д.)

© V.Mukhortov, NSU, 2023

RUP: Анализ и проектирование

Задачи:

- Трансформировать требования собранные на предыдущем этапе в дизайн системы
- Проработать архитектуру системы
- Адаптировать дизайн к среде исполнения

Модели:

- Analysis model
- Design model

SAD (Архитектурный документ)

- SAD (Software Architecture Document) – содержит полное описание архитектуры системы

- Содержит:
  - Use-case view
  - Logical View (архитектурно важные части Design model)
  - Process View
  - Deployment View (диаграмма размещения системы)
  - Implementation View
  - Open issues (список известных проблем, например, с производительностью или масштабируемостью, возможные пути решения)
  - Quality issues (любые проблемы в качестве)

RUP: Реализация (Implementation)

Задачи:

- Структурирование системы
- Реализация компонент системы

Артефакты:

- SAD – приведение в соответствие с реализацией
- Implementation model – модель реализации системы в терминах компонент и процессов

© V.Mukhortov, NSU, 2023

RUP: Ключевые роли

- Project Manager
- Analyst
- Test Designer
- System Architect
- Designer
- Implementer
- Tester

Роль RUP

- Систематизация знаний в области процессов разработки ПО
- Высокоадаптивный процесс: используйте те практики, которые необходимы в конкретном проекте.
- Производные процессы: Open UP, Agile UP

Бизнес-анализ

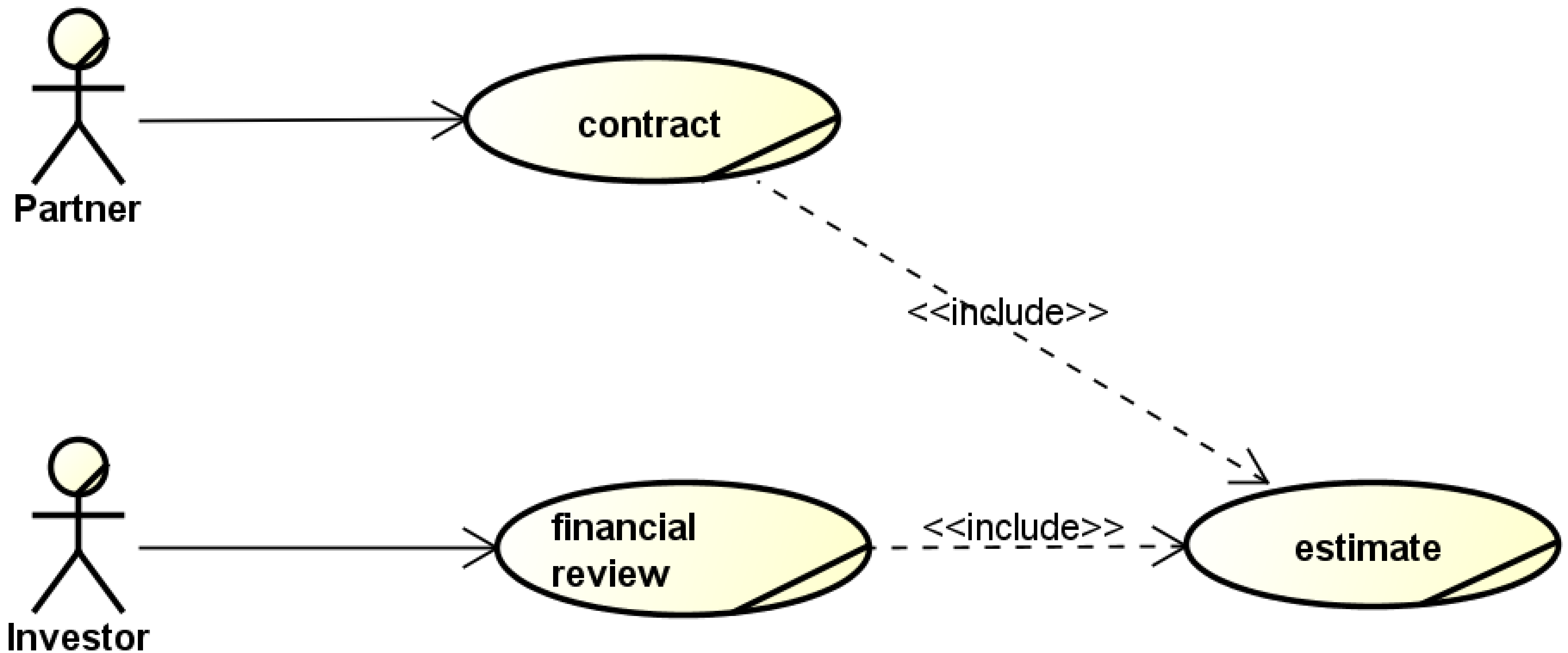
- Задачи:
  - Идентификация бизнес-процессов (use-cases)
  - Идентификация бизнес-актеров и сущностей(entity)
  - Улучшение (refine) бизнес-процессов

- Модели:
  - business use-case model
  - business object model

business use-case model

- Модель, описывающая бизнес процессы в терминах *business-actors* и *business use-cases*
- *Business actor – некто или нечто **вовне** бизнеса, взаимодействующее с ним*
  - UML: класс со стереотипом *<<business actor>>*
- Business use-case – *бизнес-процесс, представляющий ценность для business actor*
  - UML: use-case со стереотипом *<<business use-case>>*

Модель бизнес-процессов



© V.Mukhortov, NSU, 2023

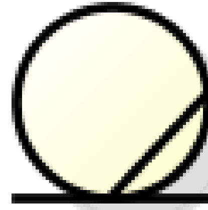
business object model

- Business object model - модель, описывающая реализацию business use-cases в терминах взаимодействующих объектов
  - UML: классы со стереотипами “business worker” и “business entity”

- Business use-case realization – *часть business object model, коллаборация, описывающая при помощи activity, sequence, и class диаграмм, как данный business use-case реализован в business-object model.*
  - UML: use-case со стереотипом “business use-case realization”, классы со стереотипом “business actor”

© V.Mukhortov, NSU, 2023

Бизнес-объекты



- Business-worker – *исполнитель бизнес-процесса*
  - *Rational: class со стереотипом <<business worker>>*
  - *ASTAH: <<control>> и <<business>>*
- Business-entity – *пассивная сущность, используемая в бизнесе*
  - *Rational: class со стереотипом <<business entity>>*
  - *ASTAH: <<entity>> и <<business>>*

activity diagram для бизнес-процесса “контракт”

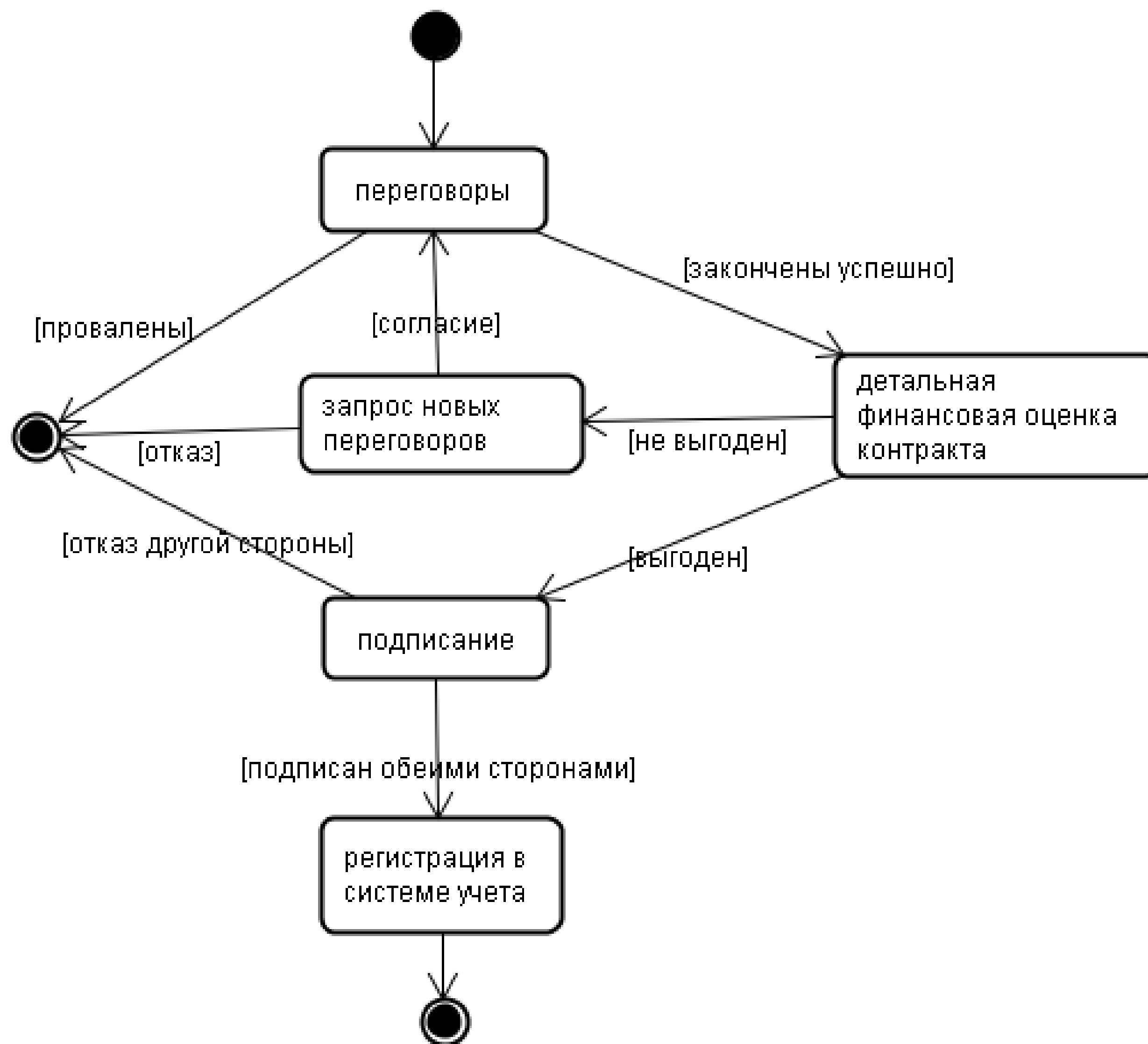


диаграмма классов для бизнес-процесса “контракт”

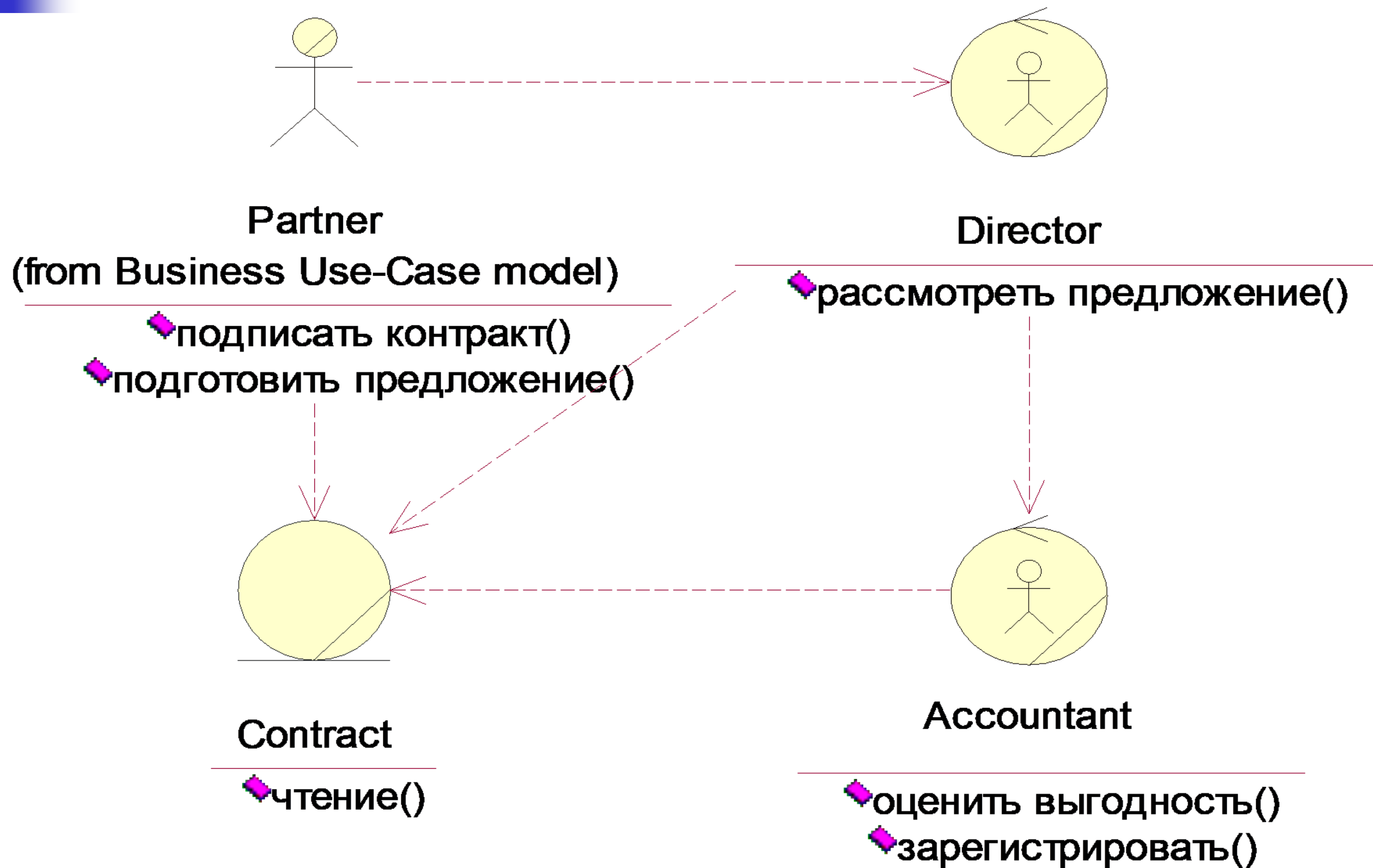
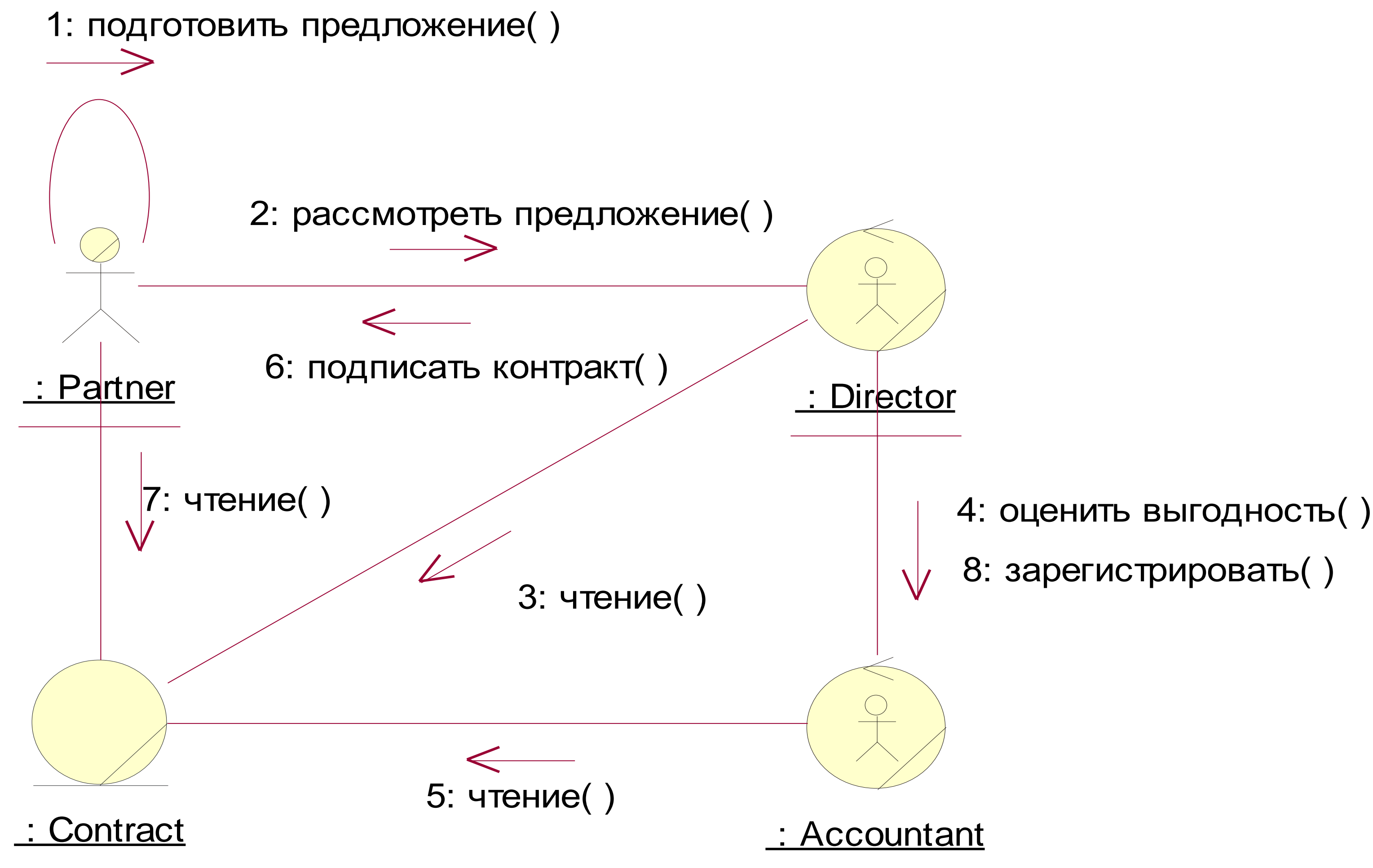


диаграмма коопераций для бизнес-процесса “контракт”



10. Анти-шаблоны

- ✓ Анти-шаблон – часто используемый, но заведомо неэффективный метод решения проблемы.

В области разработки ПО выделяют анти-шаблоны:

- Архитектурные
- ОО декомпозиции
- Программирования

Архитектурные анти-шаблоны

- Золотой молоток (или Серебряная пуля)
- Большой комок грязи
- Внутренняя платформа
- Database-as-IPC

Золотой молоток

Когда в руках молоток, все проблемы кажутся гвоздями.

Любая большая технология общего назначения может стать золотым молотком в руках разработчиков с узким технологическим кругозором.

Потенциальные «молотки»:

- E-commerce платформы, напр. Oracle ATG Web-commerce Solution
- Groupware платформы, напр. HCL Notes (ранее называлась IBM Notes, еще ранее Lotus Notes)
- Различные CMS (Wordpress, Joomla, Drupal и т.д)

Комок грязи / big ball of mud

- Система разрабатывается долго, часто при
  - высокой текучке кадров
  - и высоком давлении со стороны бизнеса
- Никто не заботится о рефакторинге
- Появление новых требований никогда не приводит к пересмотру архитектуры
- Документации или нет, или она давно неактуальна
- => Высокая стоимость поддержки

Система настолько обширна и гибка, что сама превращается в среду разработки систем.

- Emacs, MATLAB
- Mozilla, Opera, Netscape, Firefox
- Eclipse

Закон Куперберга-Завински: Каждая программа стремится вырасти настолько, чтобы начать читать почту.

Database as IPC

БД используется для организации межпроцессного взаимодействия (IPC - Inter Process Communication), превращаясь в хранилище всего и вся, включая:

- Конфигурацию приложения
- Временные объекты
- Файлы
- Очереди сообщений от одной части системы к другой

Последствия:

=> блокировки, сложность конкурентных изменений данных

=> тяжелая для понимания бизнес-логика приложения

Анти-шаблоны ООД

- Функциональная декомпозиция
    - C-style на классах
    - Классы на тысячи строк
- Полтергейст
    - объект, единственная задача которого – вызвать метод другого объекта и исчезнуть
- Блюдо спагетти / Object orgy
    - нарушение Law of Demeter
- God Object
    - класс, нарушающий SRP
- Sequential coupling
    - Класс, требующий вызова своих методов в строго определенном порядке

Анти-шаблоны программирования

- Hard code
- Soft code
- Copy-Paste
- Error hiding / Сокрытие ошибки
- Magic constant / Магические константы
- Создай себе проблему
- Implicit cast / Неявные преобразования
- Input kludge (or GIGO: garbage in - garbage out) / Ляпы ввода
- Cargo cult / Поклонение идолам

Hard code / Soft code

- **Hard code** – размещение в коде констант, строк, URL, паролей и т.п информации, которая может зависеть от окружения. Приводит к необходимости изменения кода и перекомпиляции при переносе ПО в другую среду исполнения.

- **Soft code** – обратная крайность: размещение бизнес-логики в конфигурационных файлах или базах данных.

© V.Mukhortov, NSU, 2023

Copy-Paste

Нет смысла копировать ошибку, если ее можно вызвать (как фунцию)

(очень старая шутка)

- Если мы что-то «накопипастили» и в этом коде есть проблема или требуется изменение логики – исправлять ее придется в нескольких местах.
- Нарушает OCP

Сокрытие ошибки

- Видели такое?

```
try {
    doSomething();
} catch( Exception ex) { }
```

- А вот это – на самом деле ничем не лучше:

```
try {
    Document doc = builder.parse(file.getAbsolutePath());
} catch( SAXException ex) {
    logger.error("Не удалось распарсить файл");
}
```

- Почему?

Магические константы

- Неименованые числовые константы

```
// перемешать колоду карт
for (int i = 0; i < 52; i++ ) {
    deck.swap( i, random.nextInt(52) );
}
```

- Строковые константы в коде

```
try {
    Document doc = builder.parse(file.getAbsolutePath());
} catch( SAXException ex) {
    logger.error("Не удалось распарсить файл");
}
```

- Этот enum используется как поле класса, сохраняемого в БД
- В БД enum сохраняется как целое число

```
public enum BatchType
{
   TRANSACTIONS,
   OBJECTS
}
```

- Почему так не стоит делать?

Создай себе проблему

- Чем этот код

```
     if( status.equals( “ok” ) )      doThis();
else if( status.equals( “error” ) ) doThat();
else doSomethingElse();
```

- хуже, чем следующий:

```
     if( “ok”.equals( status ) )       doThis();
else if( “error”.equals( status ) ) doThat();
else doSomethingElse();
```

© V.Mukhortov, NSU, 2023

Implicit cast

- Пользователь хочет получить выборку транзакций с 01/01/2021 до 10/12/2021
- В БД дата транзакции хранится в виде TIMESTAMP с точностью до миллисекунд
- Неправильная выборка:
  - date > ‘01/01/2021 00:00:00’ AND date <= ‘10/12/2021 23:59:59’
- Правильная:
  - date >= ‘01/01/2021’ AND date < ‘11/12/2021’
  - т.е from >= **date** < to + 1 day

Input kludge

- Отсутствие валидаторов данных, введенных пользователем
- Программа проходит тесты разработчика, но пользователь «роняет» ее с первой попытки
- Monkey test

Cargo cult

- Понятие cargo cult возникло после Второй мировой войны

11. Use Case 2.0

- User Stories vs. Use Cases
- Use Case 2.0