Лекция 6. Шаблоны проектирования

Классификация шаблонов проектирования

1. Шаблоны проектирования GRASP
2. Шаблоны проектирования GoF
3. Принципы SOLID

Проектирование на основе обязанностей GRASP

- Responsibility-driven design – это общий подход к проектированию программных объектов.
  - Программные объекты рассматриваются как люди, имеющие свои обязанности и сотрудничающие с другими людьми для их выполнения

Обязанности, относящиеся к действиям объекта

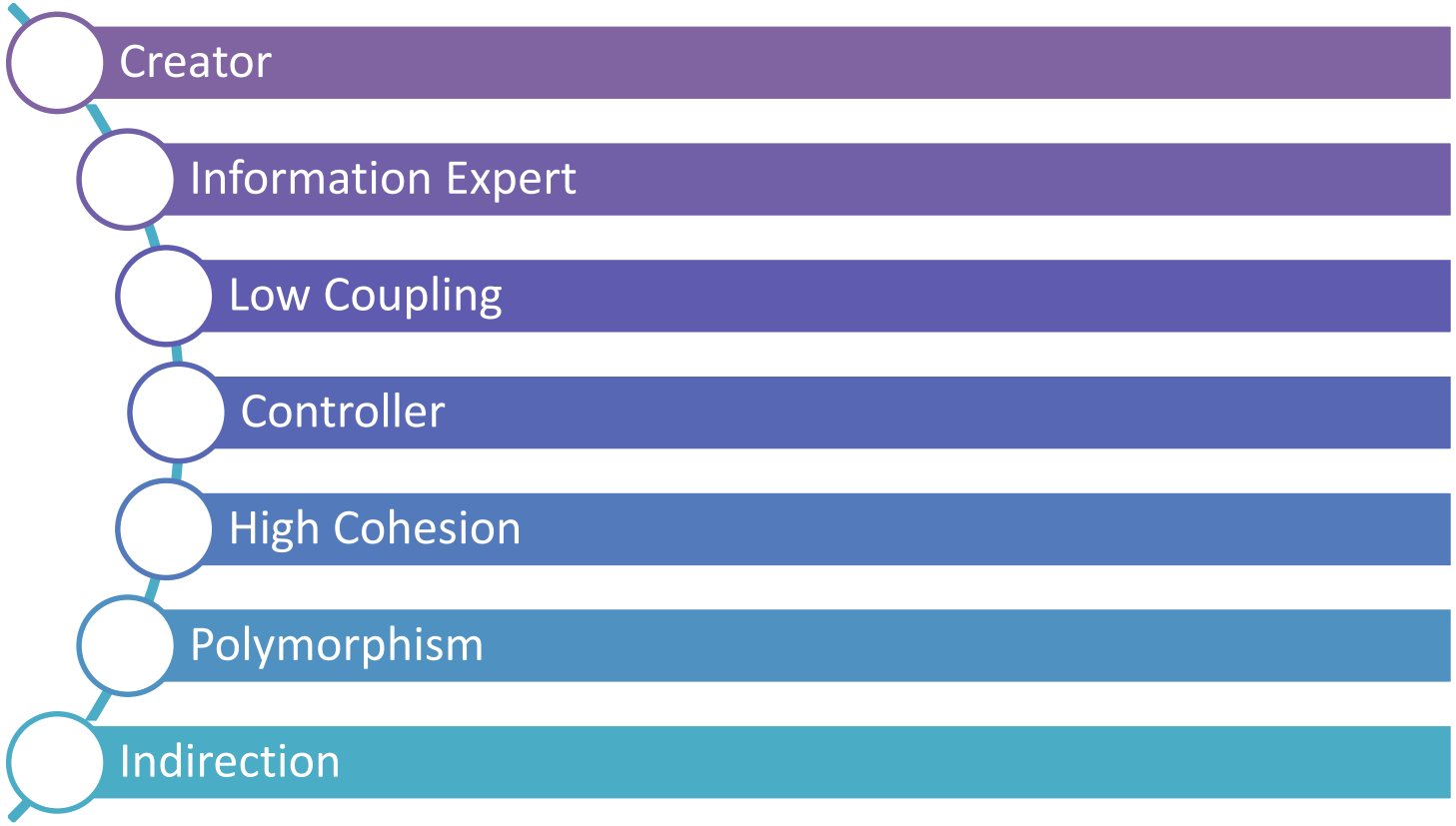
- Выполнение действий самим объектом
- Инициирование действий других объектов
- Управление действиями других объектов

Обязанности, относящиеся к знаниям объекта

- Наличие информации о закрытых инкапсулированных данных
- Наличие информации о связанных объектах
- Наличие информации о следствиях или вычисляемых величинах

Шаблон проектирования GRASP

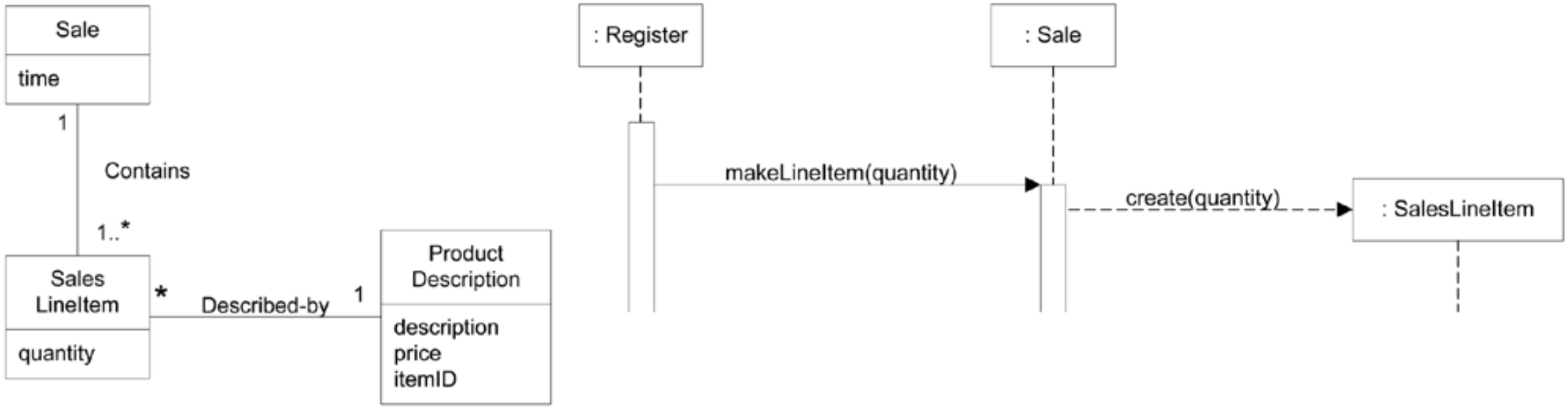
- Это именованная пара «проблема-решение», содержащая рекомендации для применения в различных конкретных ситуациях, которую можно использовать в различных контекстах



Шаблон проектирования GRASP. Creator

- **Проблема**: кто должен отвечать за создание нового экземпляра некоторого класса?
- **Решение**: назначить классу B обязанность создавать экземпляры класса A, если выполняется одно из условий:
  - Класс B **содержит** или **агрегирует** объекты A
  - Класс B **записывает** экземпляры объектов A
  - Класс B **активно использует** объекты A
  - Класс B **обладает данными инициализации**, которые будут передаваться объектам A при их создании
- Особенности: поддерживает принцип низкого связывания (Low Coupling)
- Родственные шаблоны:
  - Concrete Factory
  - Abstract Factory

Шаблон проектирования GRASP. Creator (пример)



Шаблон проектирования GRASP. Information Expert

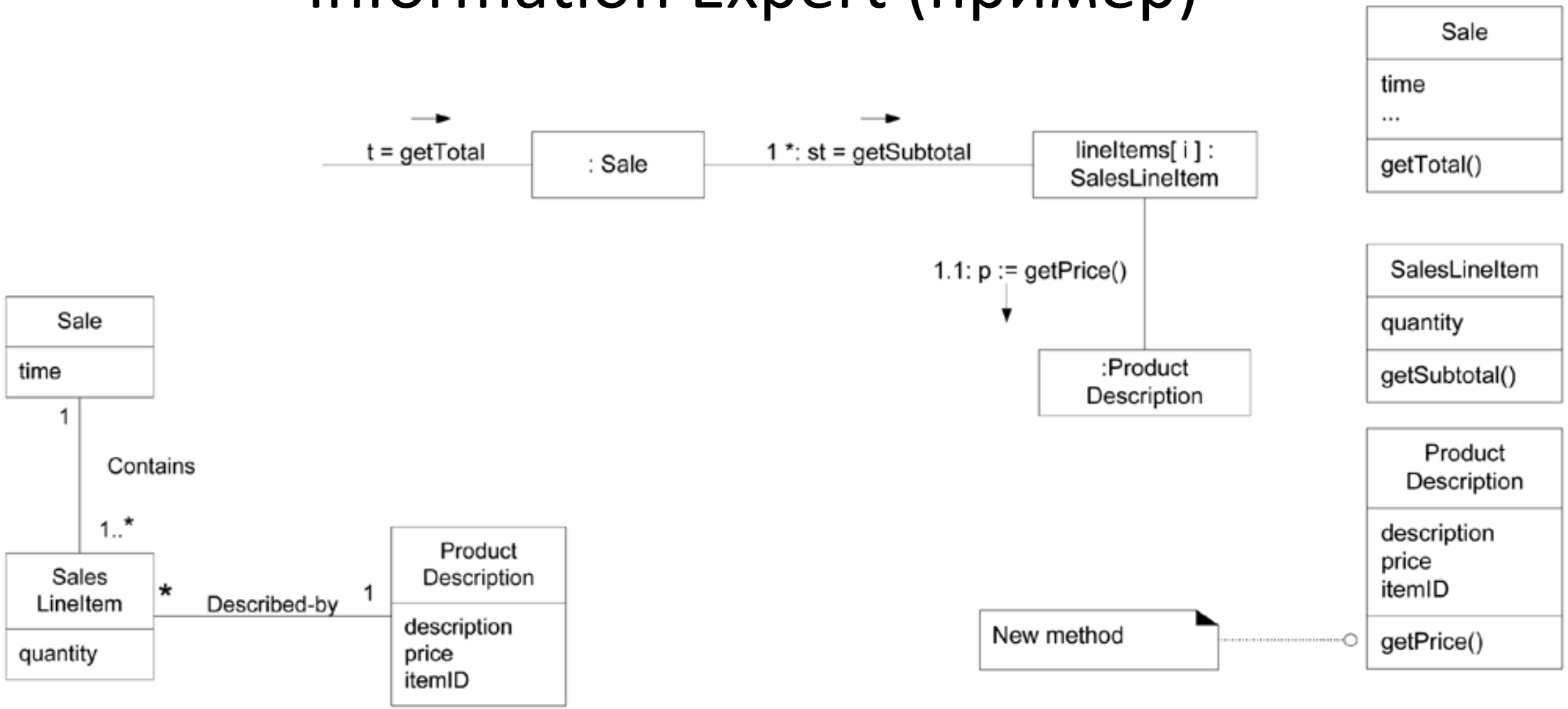
- **Проблема**: каков наиболее общий принцип распределения обязанностей между объектами при объектно-ориентированном проектировании?
- **Решение**: назначить обязанность информационному эксперту – классу, у которого имеется информация, требуемая для выполнения обязанностей
- Особенности:

✓ Поддерживает инкапсуляцию

✓ Поддерживает принцип низкого связывания (Low Coupling)

✓ Поддерживает принцип высокого зацепления (High Cohesion)

Шаблон проектирования GRASP. Information Expert (пример)



Шаблон проектирования GRASP. Low Coupling

- **Проблема**: как обеспечить зависимость, незначительное влияние изменений и повысить возможность повторного использования?
- **Решение**: распределить обязанности таким образом, чтобы степень связанности оставалась низкой
- Особенности:

✓ Тесно связан с шаблонами Expert и High Cohesion

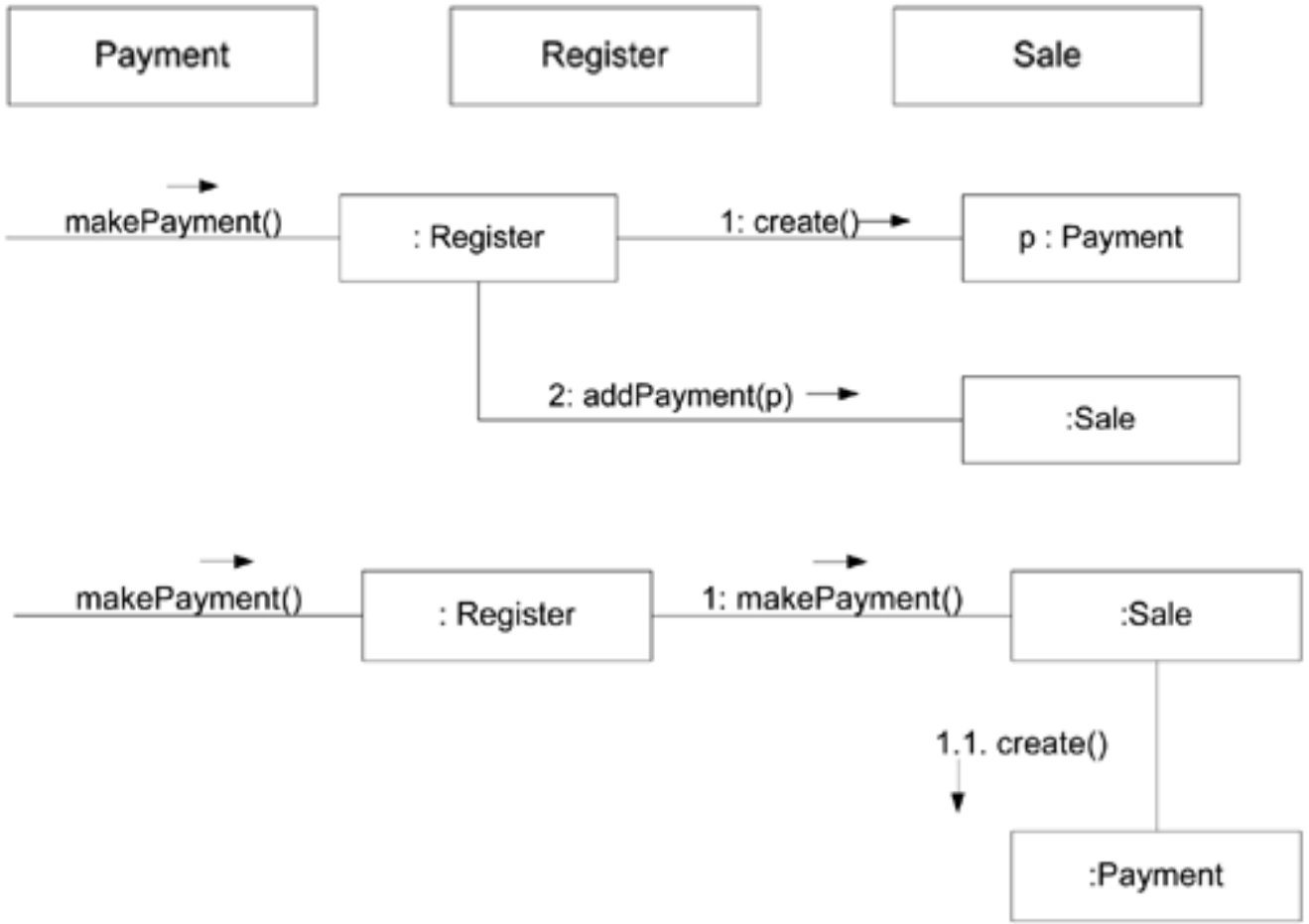
✓ Изменение компонентов мало сказывается на других объектах

✓ Принципы работы и функции компонентов можно понять, не изучая другие объекты

✓ Удобство повторного использования

✓ Связан с Protected Variations

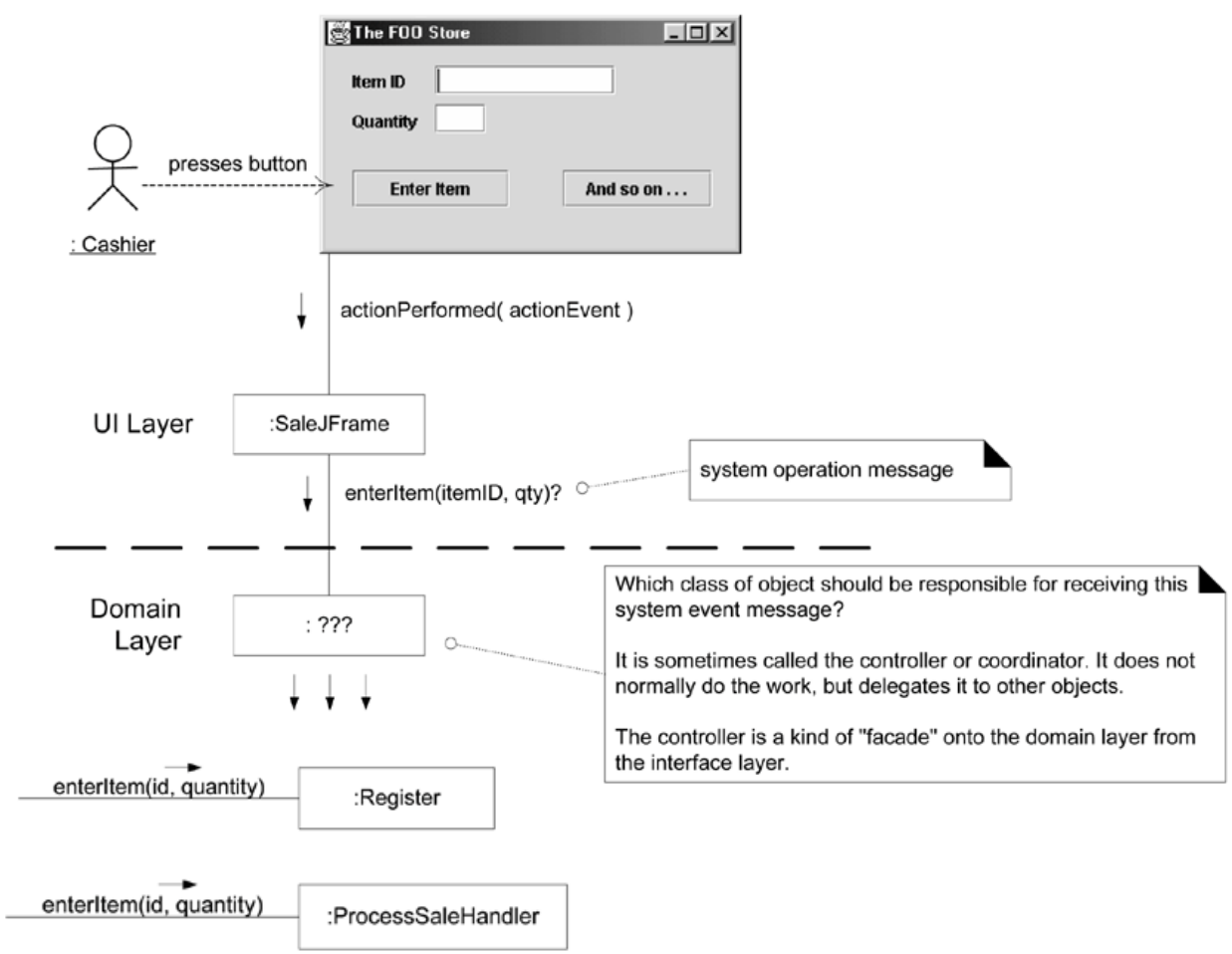
Шаблон проектирования GRASP. Low Coupling (пример)

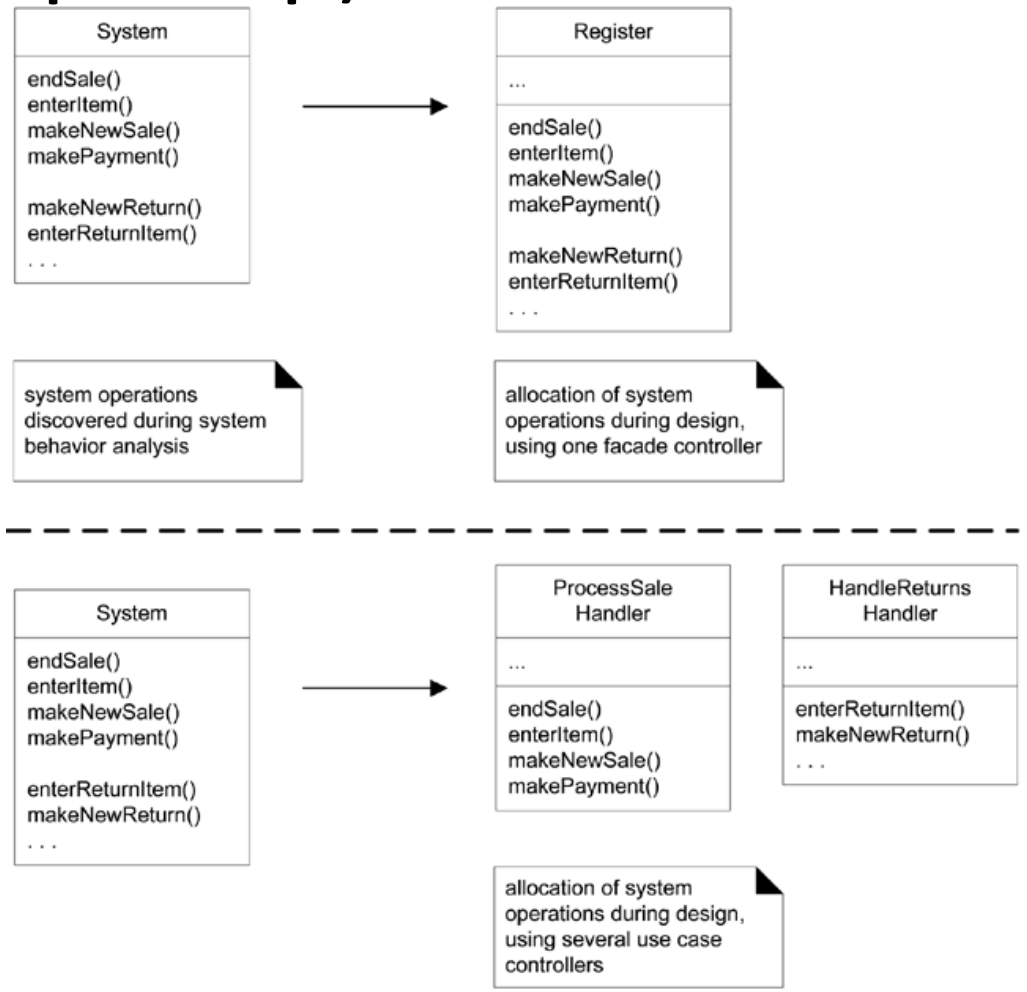


Шаблон проектирования GRASP. Controller

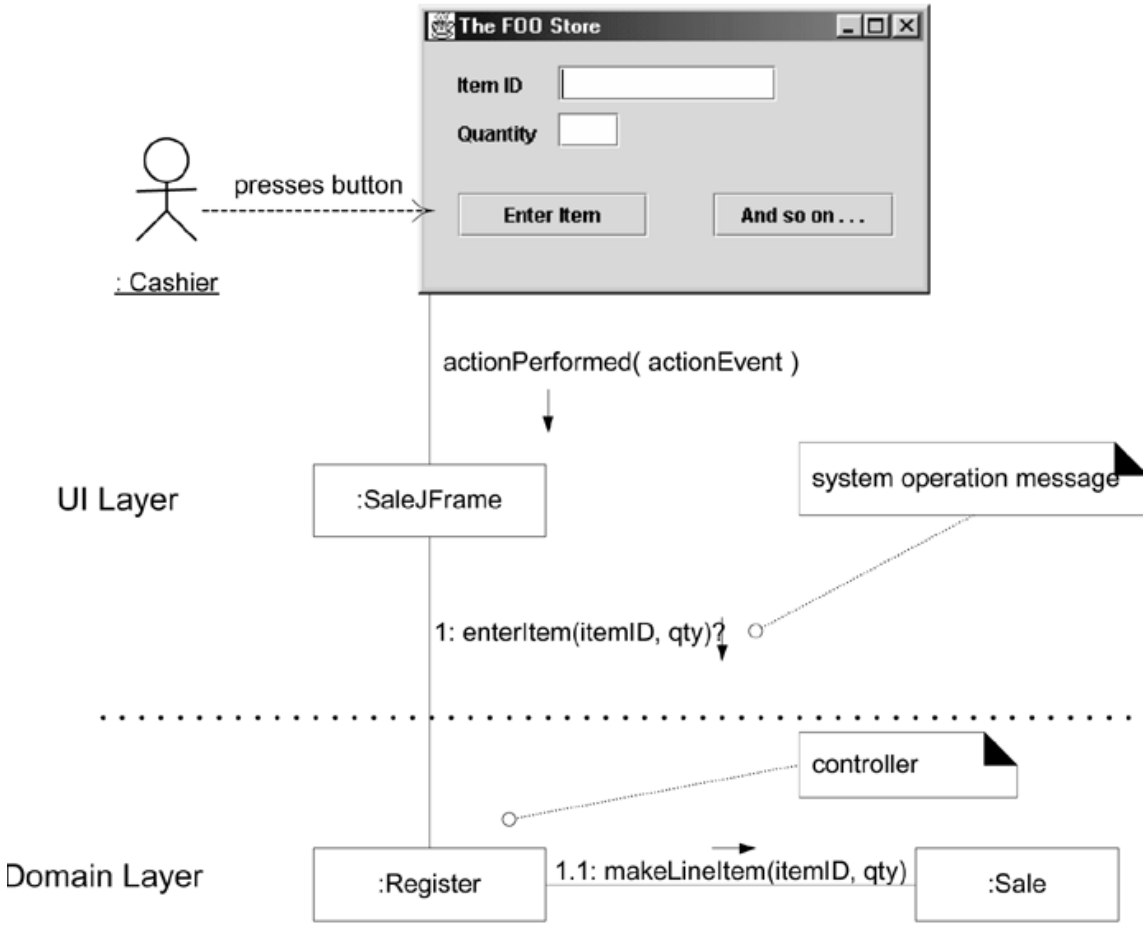
- **Проблема**: кто должен отвечать за получение и координацию выполнения системных операций, поступающих на уровне интерфейса пользователя?
- **Решение**: делегирование обязанностей по обработке системных сообщений классу, удовлетворяющему одному из следующих условий
  - Класс представляет всю систему в целом, корневой объект, устройство или подсистему
  - Класс представляет сценарий некоторого ВИ, в рамках которого выполняется обработка всех системных событий
- Особенности:
  - ✓ Улучшение условий повторного использования компонентов и подключения интерфейсов
  - ✓ Контроль состояний вариантов использования
- Похожие шаблоны:
  - Command
  - Facade
  - Layers
  - Pure Fabrication

Шаблон проектирования GRASP. Controller (пример)

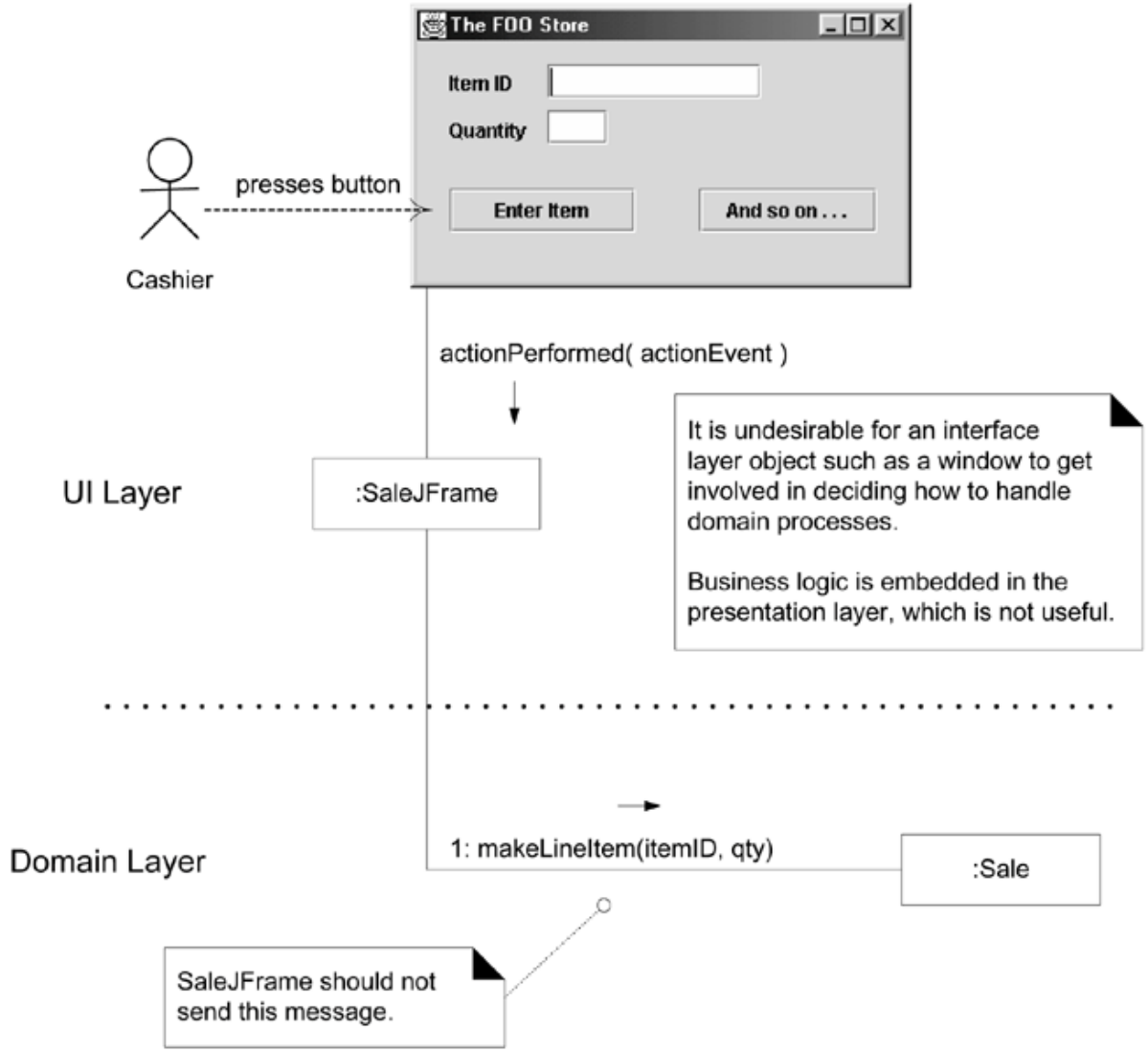




Хорошо



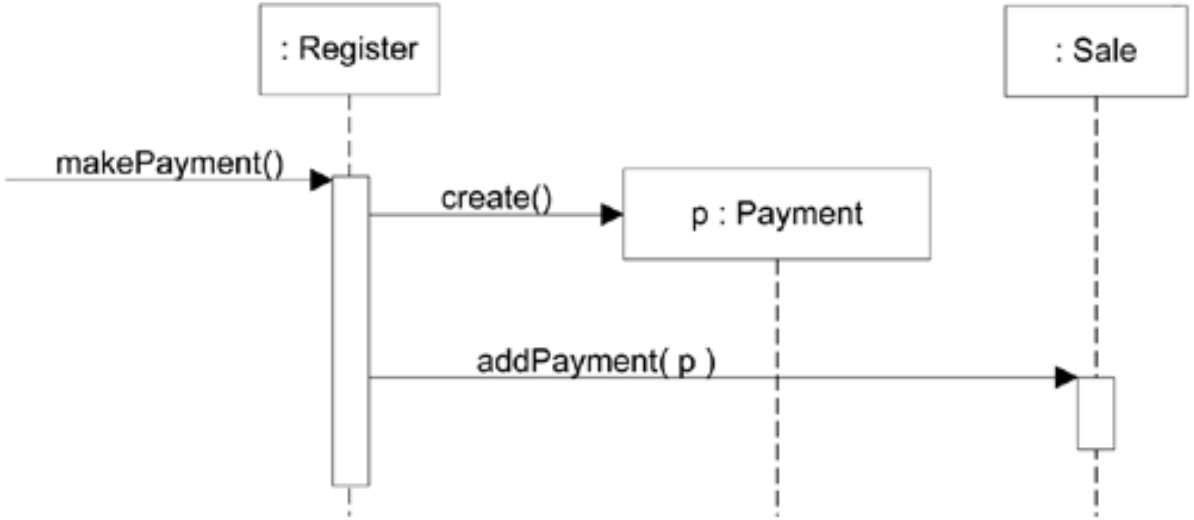
Плохо

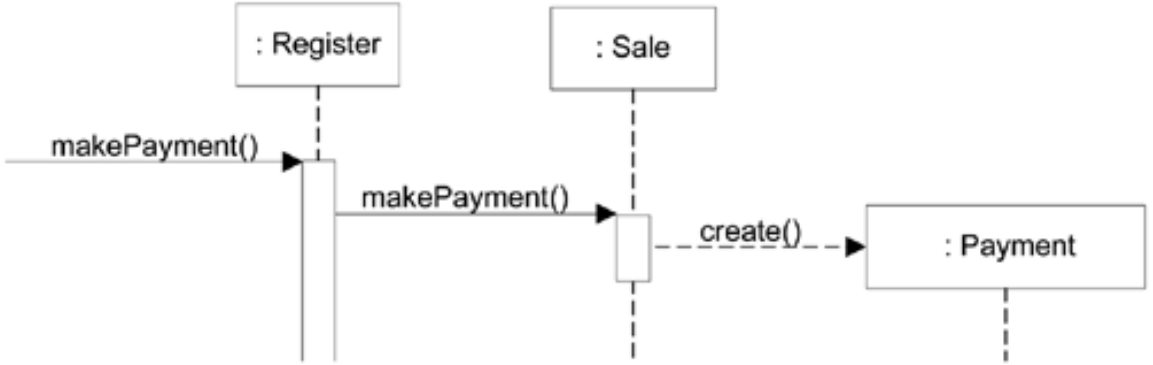


Шаблон проектирования GRASP. High Cohesion

- **Проблема**: как обеспечить сфокусированность обязанностей объекта, их управляемость и ясность, а заодно выполнение принципа Low Coupling?
- **Решение**: распределение обязанностей, поддерживающее высокую степень зацепления
- Особенности:
  - ✓ Повышается ясность и простота проектных решений
  - ✓ Упрощается поддержка и доработка
  - ✓ Зачастую обеспечивается слабое связывание
  - ✓ Улучшаются возможности повторного использования

Шаблон проектирования GRASP. High Cohesion (пример)

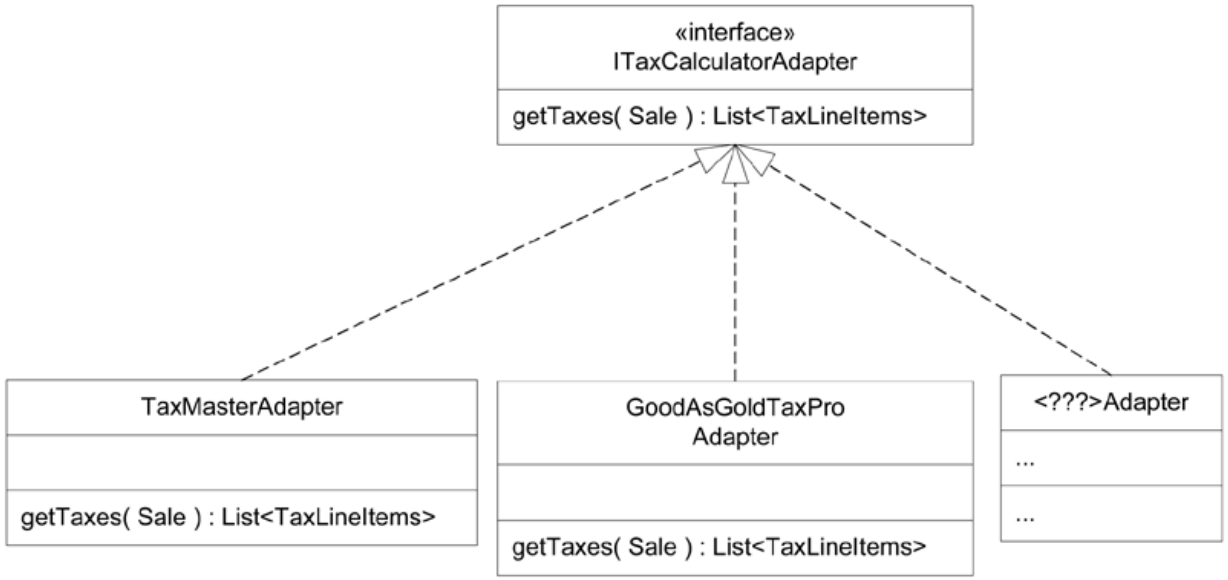


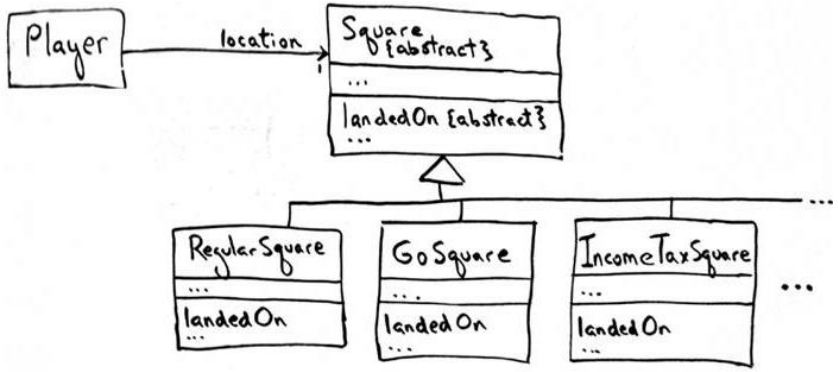


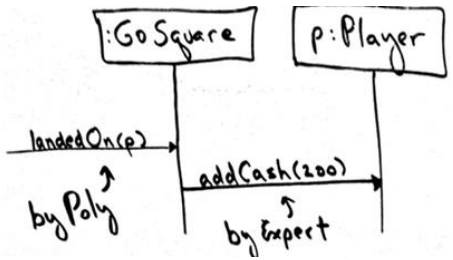
Шаблон проектирования GRASP. Polymorphism

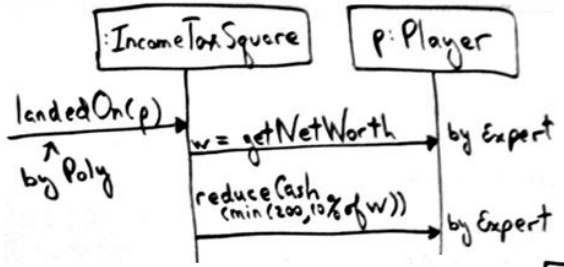
- **Проблема**: как обрабатывать альтернативные варианты поведения на основе типа? Как создавать подключаемые программные компоненты?
- **Решение**: если поведение объектов одного типа (класса) может измениться, обязанности распределяются для различных вариантов поведения с использованием полиморфных операций этого класса
- Особенности:
  - Позволяет легко расширять систему, добавляя новые вариации
  - Новые реализации можно вводить без модификации клиентской части приложения
- Связанные шаблоны:
  - Protected Variations
  - Adapter, Command, Composite, Proxy, State, Strategy …

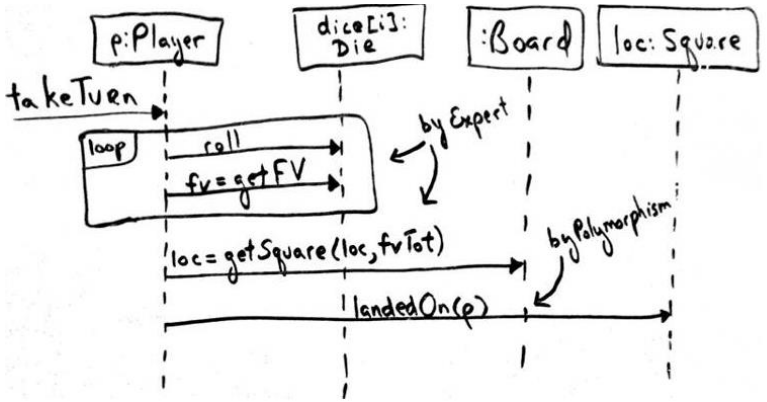
Шаблон проектирования GRASP. Polymorphism (примеры)

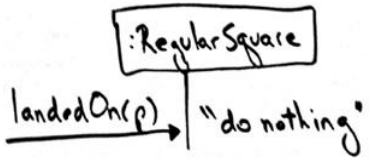


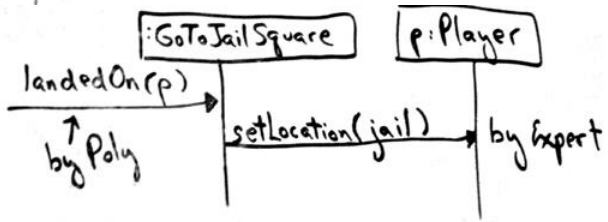








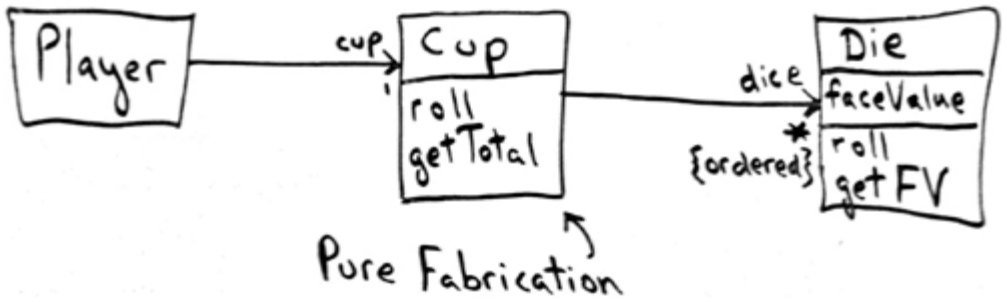


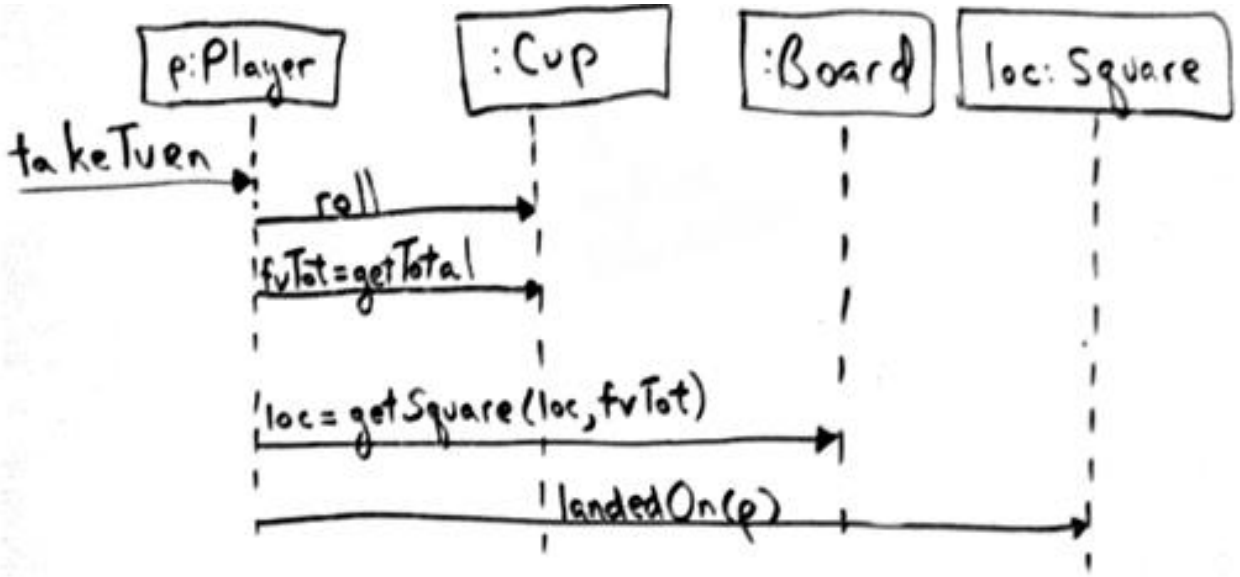


Шаблон проектирования GRASP. Pure Fabrication

- **Проблема**: какой класс должен обеспечивать реализацию шаблона High Cohesion и Low Coupling или других принципов проектирования, если шаблон Expert (например) не обеспечивает подходящего решения?
- **Решение**: присвоить группу обязанностей с высокой степенью зацепления искусственному классу, не представляющему конкретного понятия предметной области, т.е. синтезировать искусственную сущность для поддержки высокого зацепления, слабого связывания и повторного использования
- Особенности:
  - Реализуется принцип высокого зацепления
  - Повышается потенциал повторного использования
- Связанные шаблоны:
  - Low Coupling
  - High Cohesion
  - Adapter, Command, Strategy …

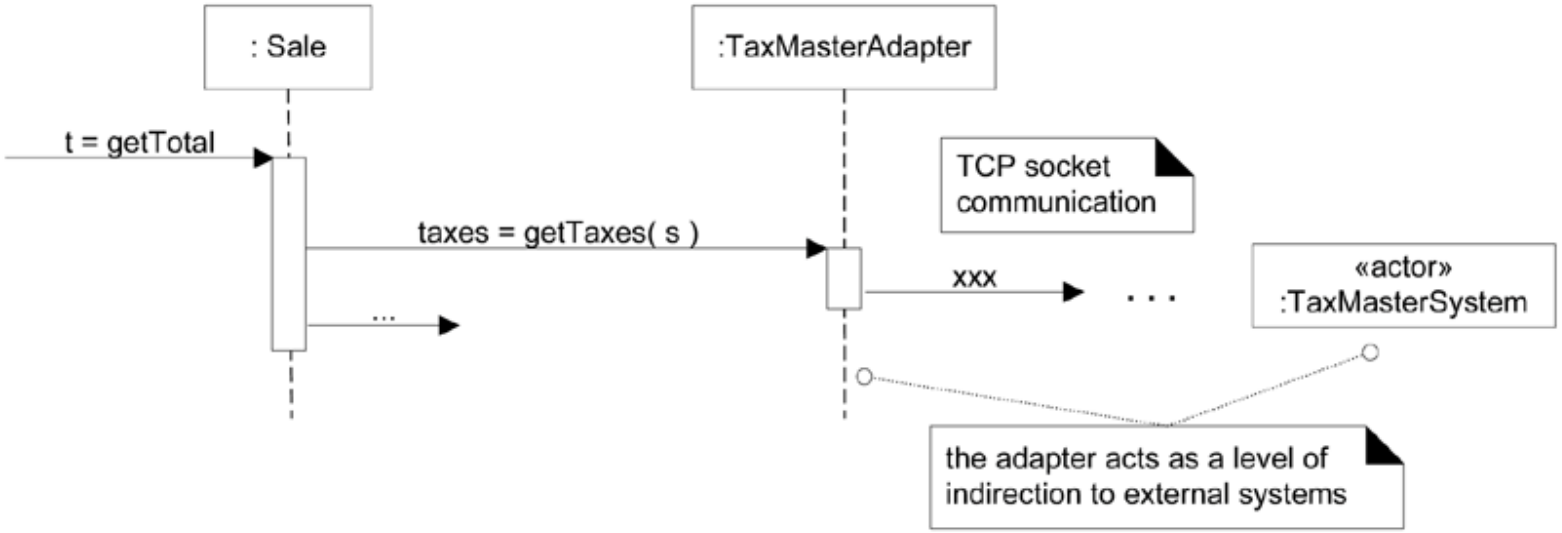
Шаблон проектирования GRASP. Pure Fabrication (примеры)





Шаблон проектирования GRASP. Indirection

- **Проблема**: как распределить обязанности, чтобы обеспечить отсутствие прямого связывания; снизить уровень связывания объектов и сохранить высокий потенциал повторного использования?
- **Решение**: присвоить обязанности промежуточному объекту для обеспечения связывания между другими компонентами или службами, которые не связаны между собой напрямую



Шаблон проектирования GRASP. Protected Variations

- **Проблема**: как спроектировать объекты, подсистемы и систему, чтобы изменение этих элементов не оказывало нежелательное влияние на другие элементы?
- **Решение**: идентифицировать точки возможных вариаций или неустойчивости; распределить обязанности таким образом, чтобы обеспечить устойчивый интерфейс

Шаблоны проектирования GoF

- Паттернами проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения.
- В данном случае предполагается, что есть некоторый набор общих формализованных проблем, которые довольно часто встречаются, и паттерны предоставляют ряд принципов для решения этих проблем.
- Концепцию паттернов впервые описал Кристофер Александер в книге «Язык шаблонов. Города. Здания. Строительство».
- Идею развили авторы Эрих Гамм, Ричард Хелм, Ральф Джонсон и Джон Влиссидес (Gang of Four) в книге 1995 года «Design Patterns: Elements of Reusable Object-Oriented Software», в которой применили концепцию типовых паттернов в программировании. В книгу вошли 23 паттерна, решающие различные проблемы объектно-ориентированного дизайна.

Применение шаблонов GoF

- Самое главная причина — паттерны упрощают проектирование и поддержку программ.
- **Проверенные решения.** Код более предсказуем, когда используются готовые решения, вместо повторного изобретения тех же методов и структур.
- **Стандартизация кода.** Меньше ошибок, так как используются типовые унифицированные решения, в которых давно найдены все скрытые проблемы.
- **Общий язык.** Используется только название паттерна, вместо того, чтобы час объяснять другим членам команды какой подход был придуман и какие классы для этого нужны.

Используемые обозначения шаблонов GoF

- Отношения между классами

агрегация (aggregation) — описывает связь «часть»–«целое», в котором «часть» может существовать отдельно от «целого». Ромб указывается со стороны «целого».

композиция (composition) — подвид агрегации, в которой «части» не могут существовать отдельно от «целого».

зависимость (dependency) — изменение в одной сущности (независимой) может влиять на состояние или поведение другой сущности (зависимой). Со стороны стрелки указывается независимая сущность.

обобщение (generalization) — отношение наследования или реализации интерфейса. Со стороны стрелки находится суперкласс или интерфейс.

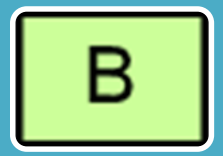
Используемые обозначения шаблонов GoF. Виды паттернов

порождающие (creational)

- это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов

структурные (structural)

- рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты



поведенческие (behavioral)

- они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение

Каталог шаблонов проектирования GoF

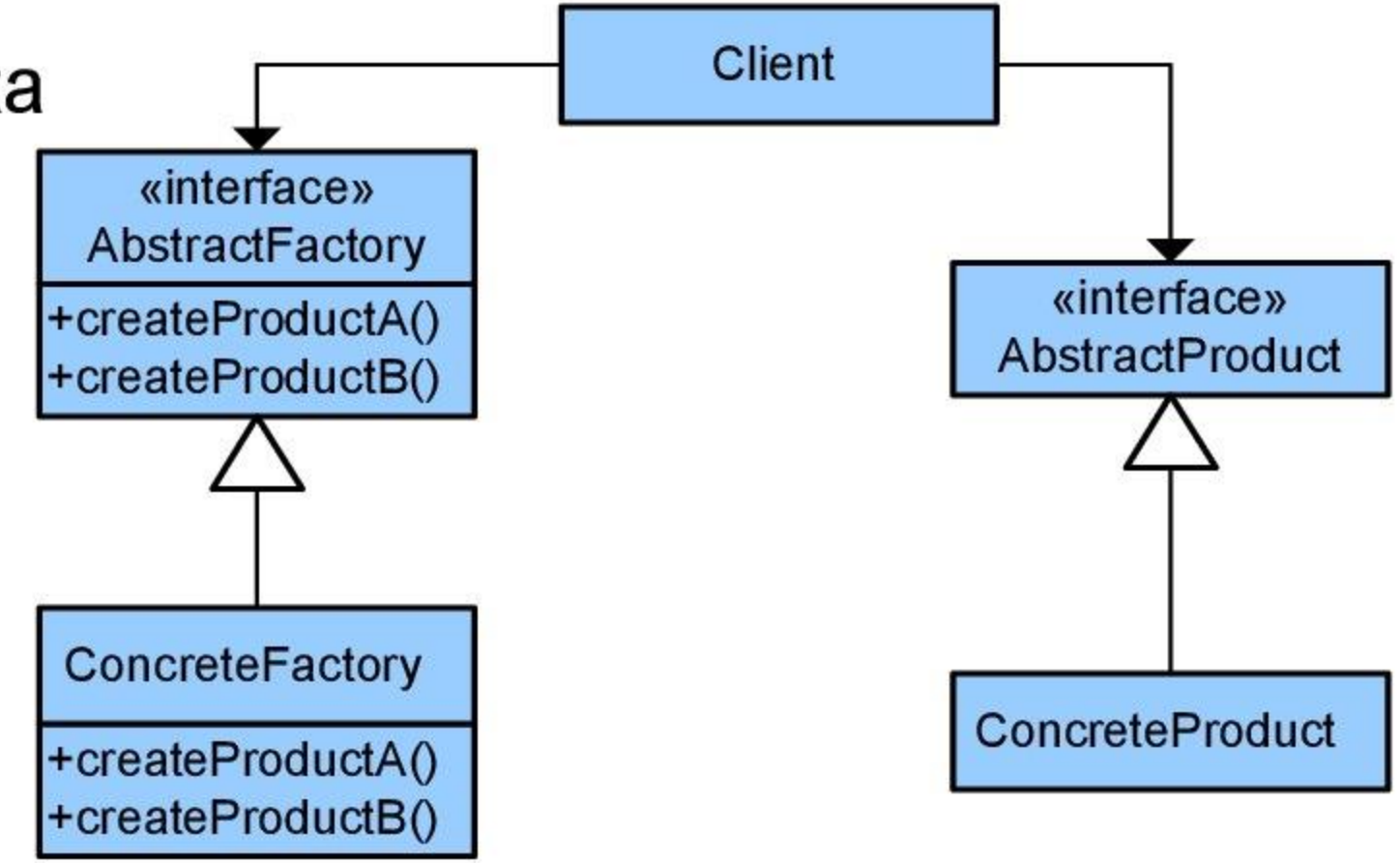
Каталог шаблонов проектирования GoF

Абстрактная фабрика
Abstract factory

Тип: Порождающий

Что это:

Предоставляет интерфейс для создания групп связанных или зависимых объектов, не указывая их конкретный класс.



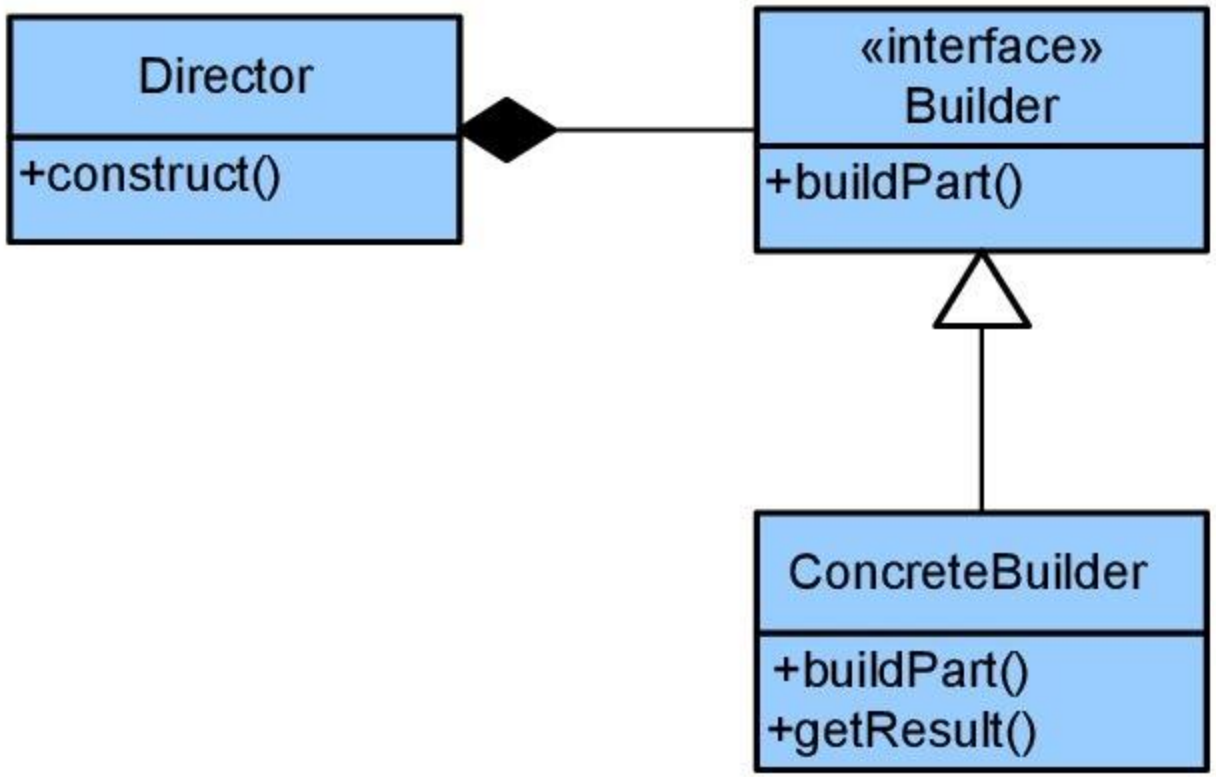
Каталог шаблонов проектирования GoF

Строитель
Builder

Тип: Порождающий

Что это:

Разделяет создание сложного объекта и инициализацию его состояния так, что одинаковый процесс построения может создать объекты с разным состоянием.



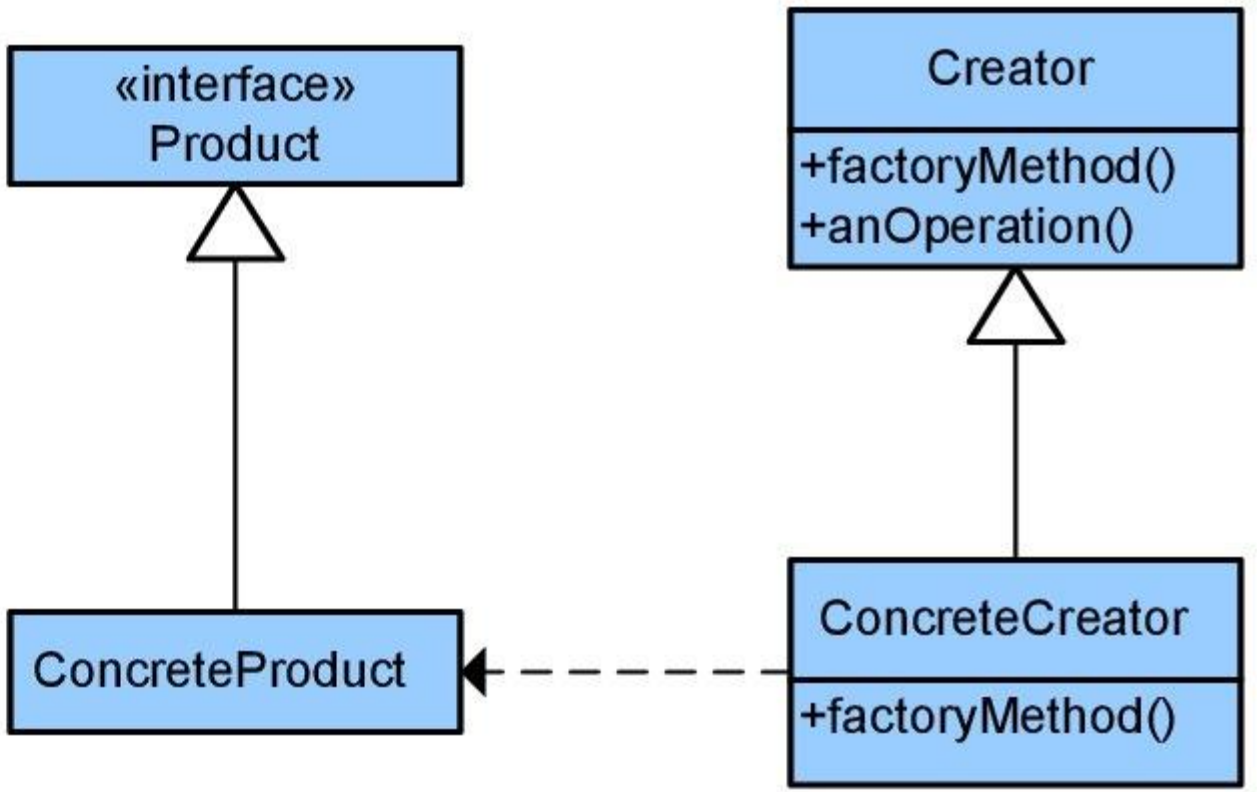
Каталог шаблонов проектирования GoF

Фабричный метод
Factory method

Тип: Порождающий

Что это:

Определяет интерфейс для создания объекта, но позволяет подклассам решать, какой класс инстанцировать. Позволяет делегировать создание объекта подклассам.



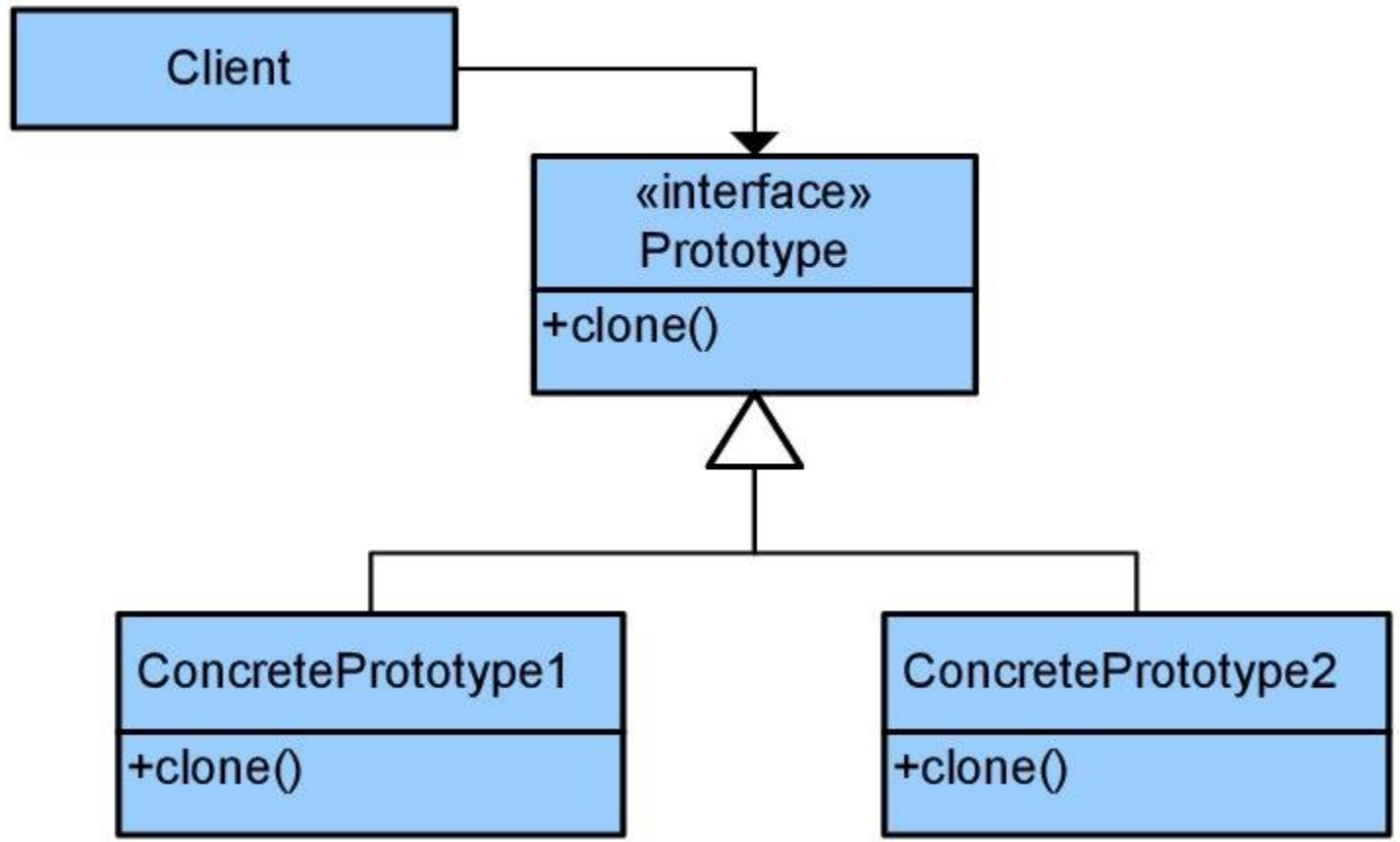
Каталог шаблонов проектирования GoF

Прототип
Prototype

Тип: Порождающий

Что это:

Определяет несколько видов объектов, чтобы при создании использовать объект-прототип и создаёт новые объекты, копируя прототип.



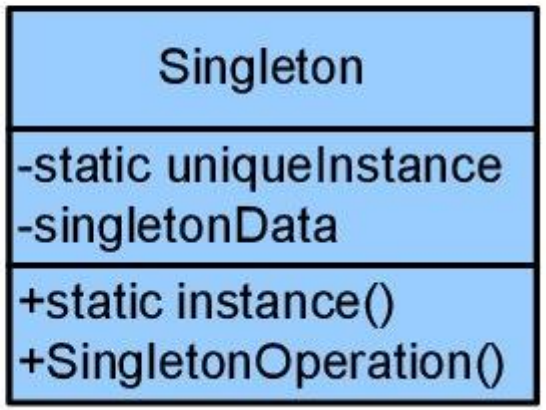
Каталог шаблонов проектирования GoF

Одиночка
Singleton

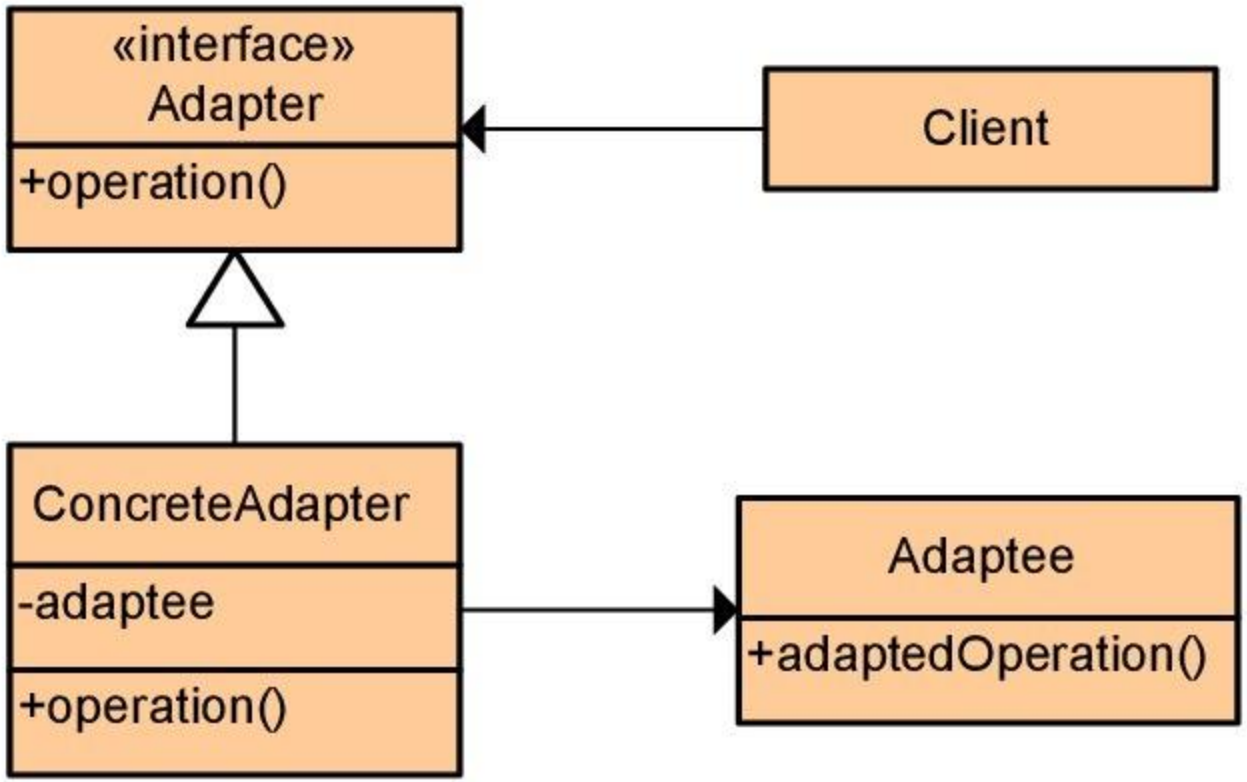
Тип: Порождающий

Что это:

Гарантирует, что класс имеет только один экземпляр и предоставляет глобальную точку доступа к нему.



Каталог шаблонов проектирования GoF



Адаптер
Adapter

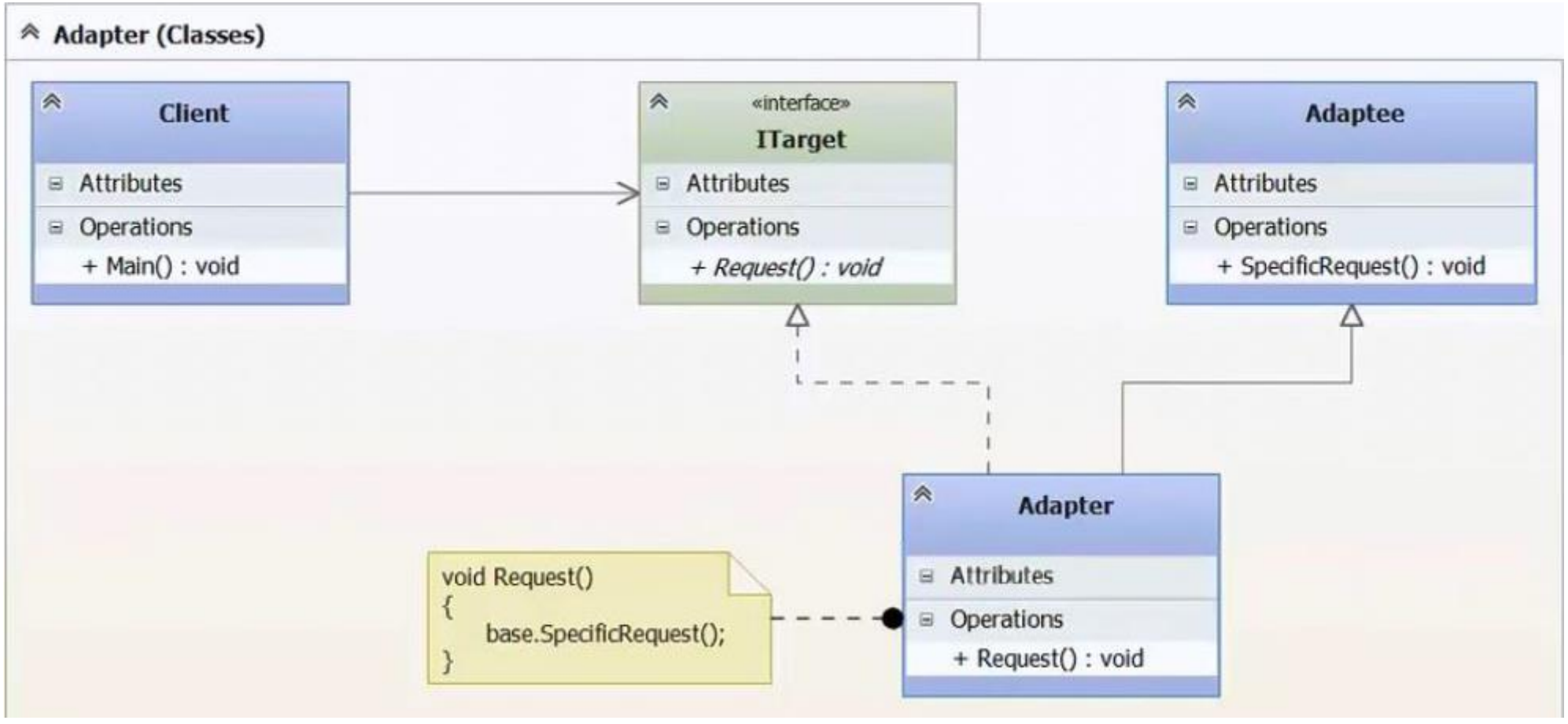
Тип: Структурный

Что это:

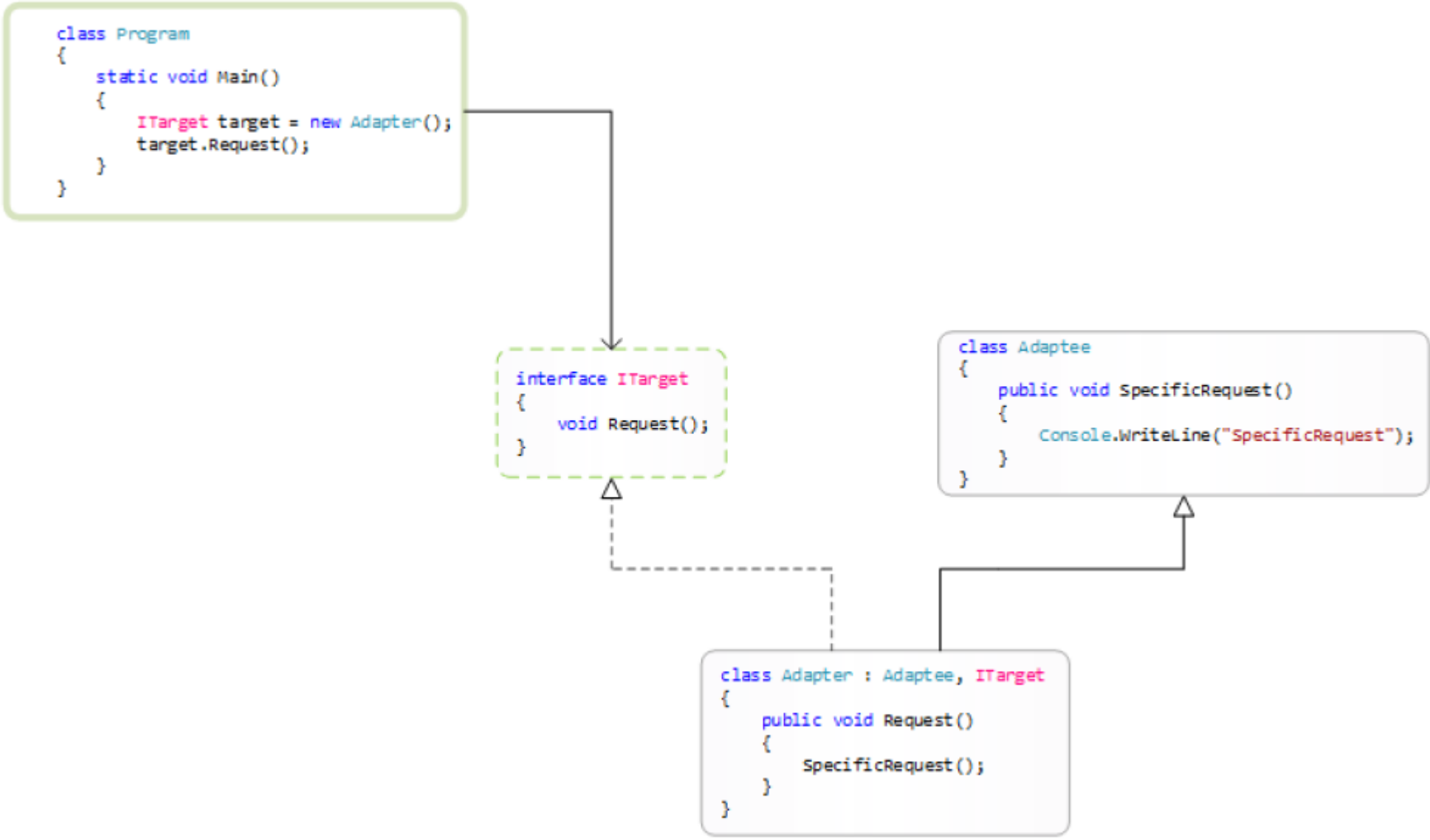
Конвертирует интерфейс класса в другой интерфейс, ожидаемый клиентом. Позволяет классам с разными интерфейсами работать вместе.

Адаптер. Структура паттерна на языке UML

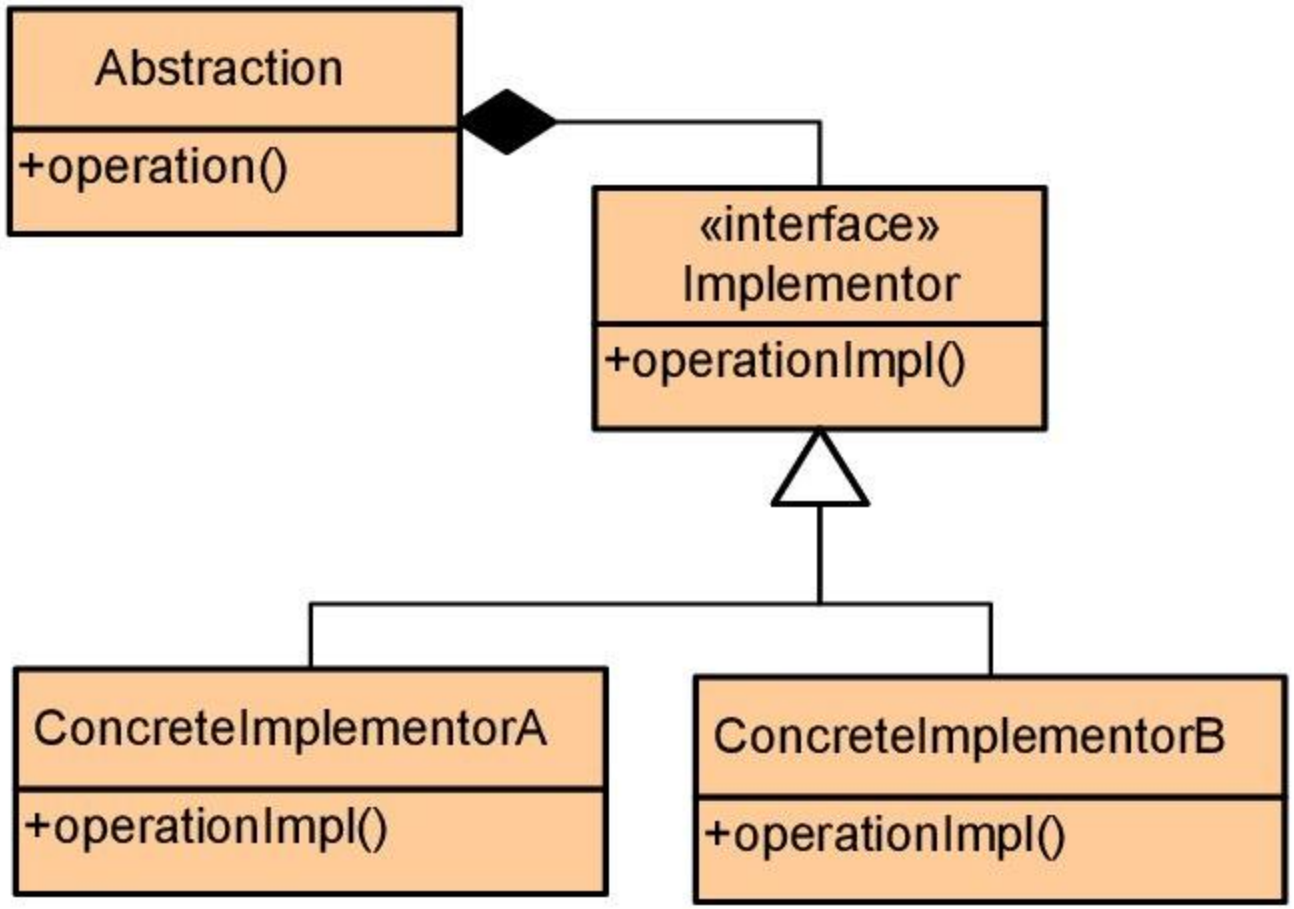
- Адаптирует интерфейс Adaptee к интерфейсу ITarget.
- Позволяет классу Adapter переопределить или заместить некоторые операции из базового класса Adaptee.
- Оставляет возможность создания только одного экземпляра класса Adapter.



Адаптер. Структура паттерна на языке C#



Каталог шаблонов проектирования GoF



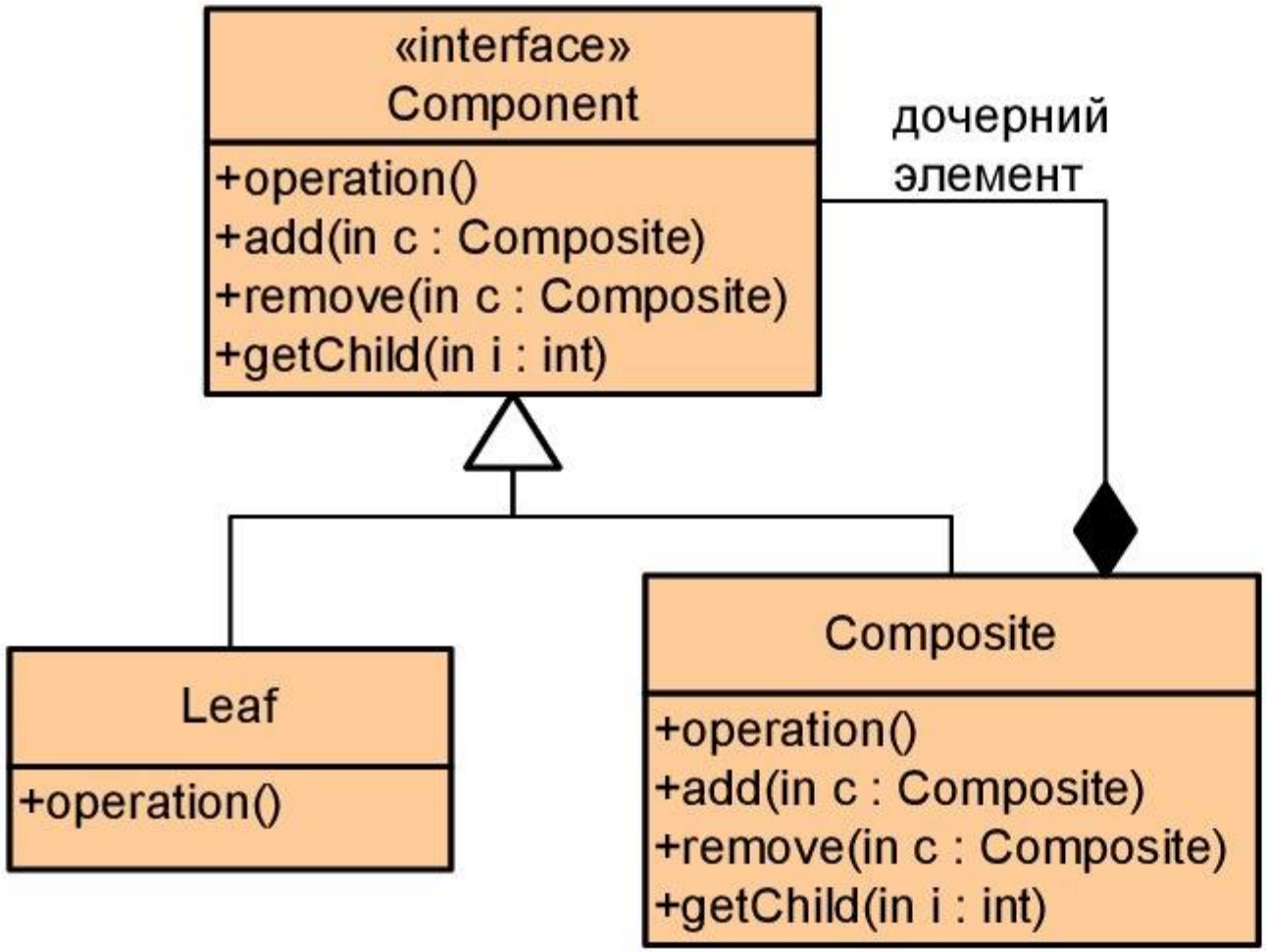
Мост
Bridge

Тип: Структурный

Что это:

Разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.

Каталог шаблонов проектирования GoF



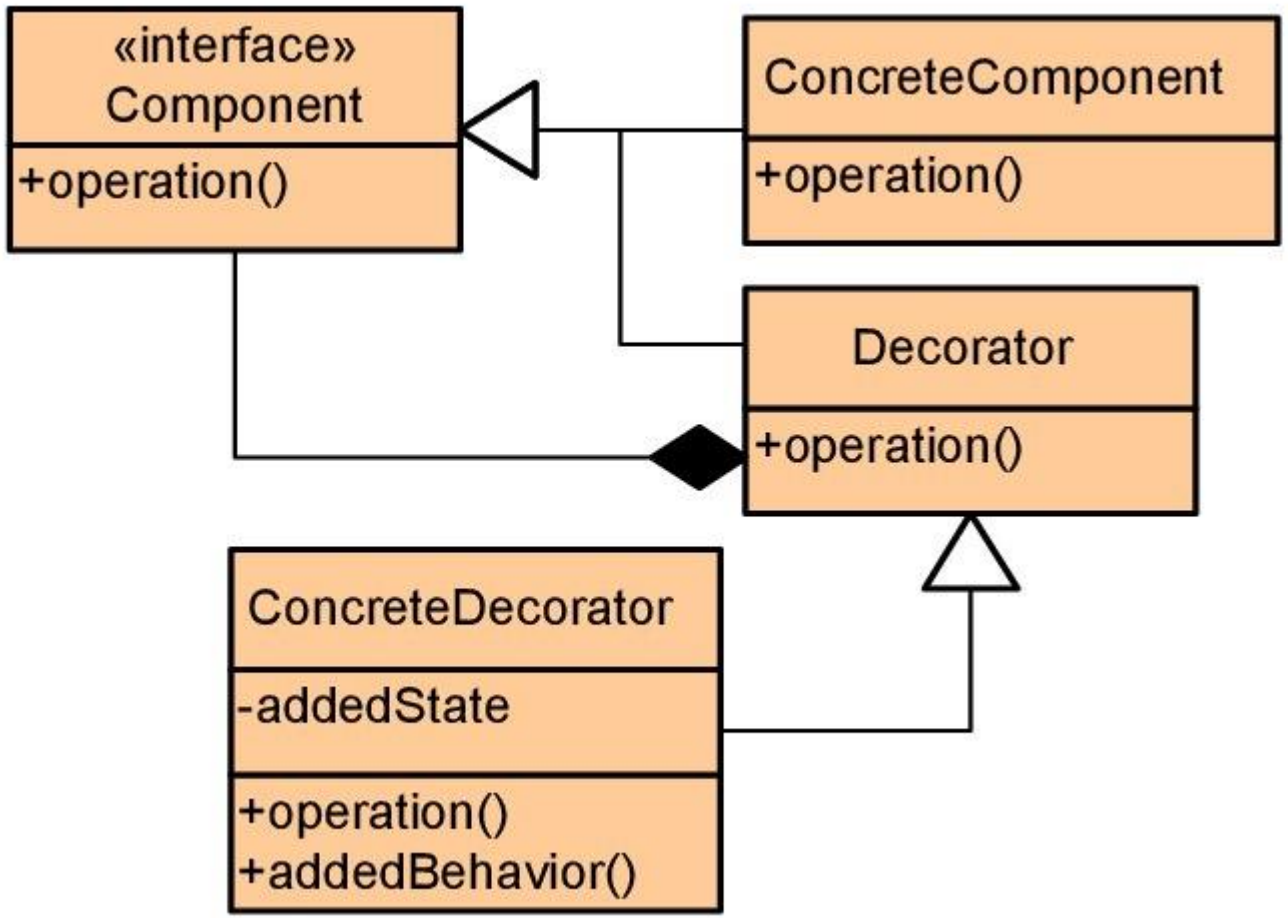
Компоновщик
Composite

Тип: Структурный

Что это:

Компонует объекты в древовидную структуру, представляя их в виде иерархии. Позволяет клиенту одинаково обращаться как к отдельному объекту, так и к целому поддереву.

Каталог шаблонов проектирования GoF



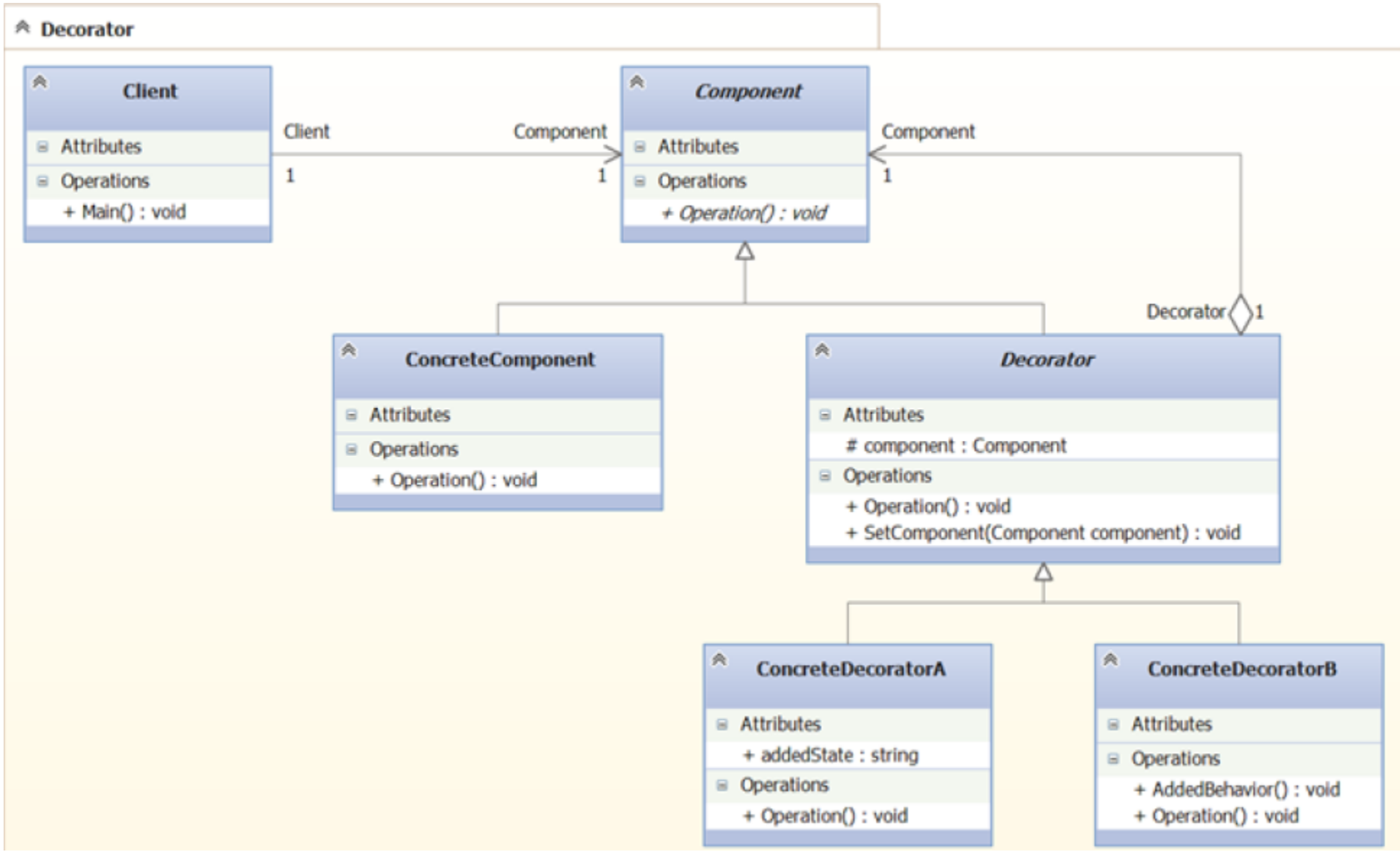
Декоратор
Decorator

Тип: Структурный

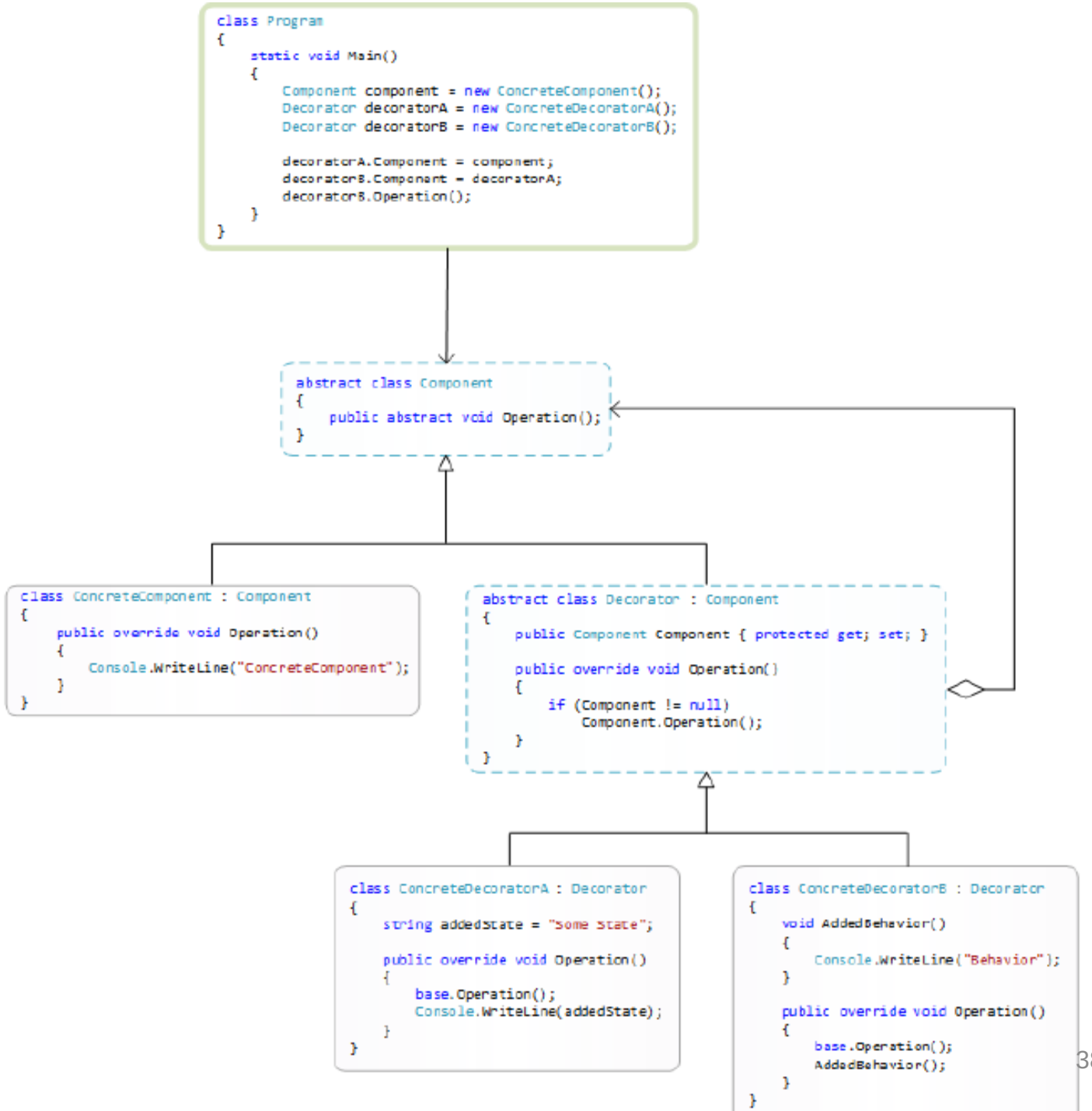
Что это:

Динамически предоставляет объекту дополнительные возможности. Представляет собой гибкую альтернативу наследованию для расширения функциональности.

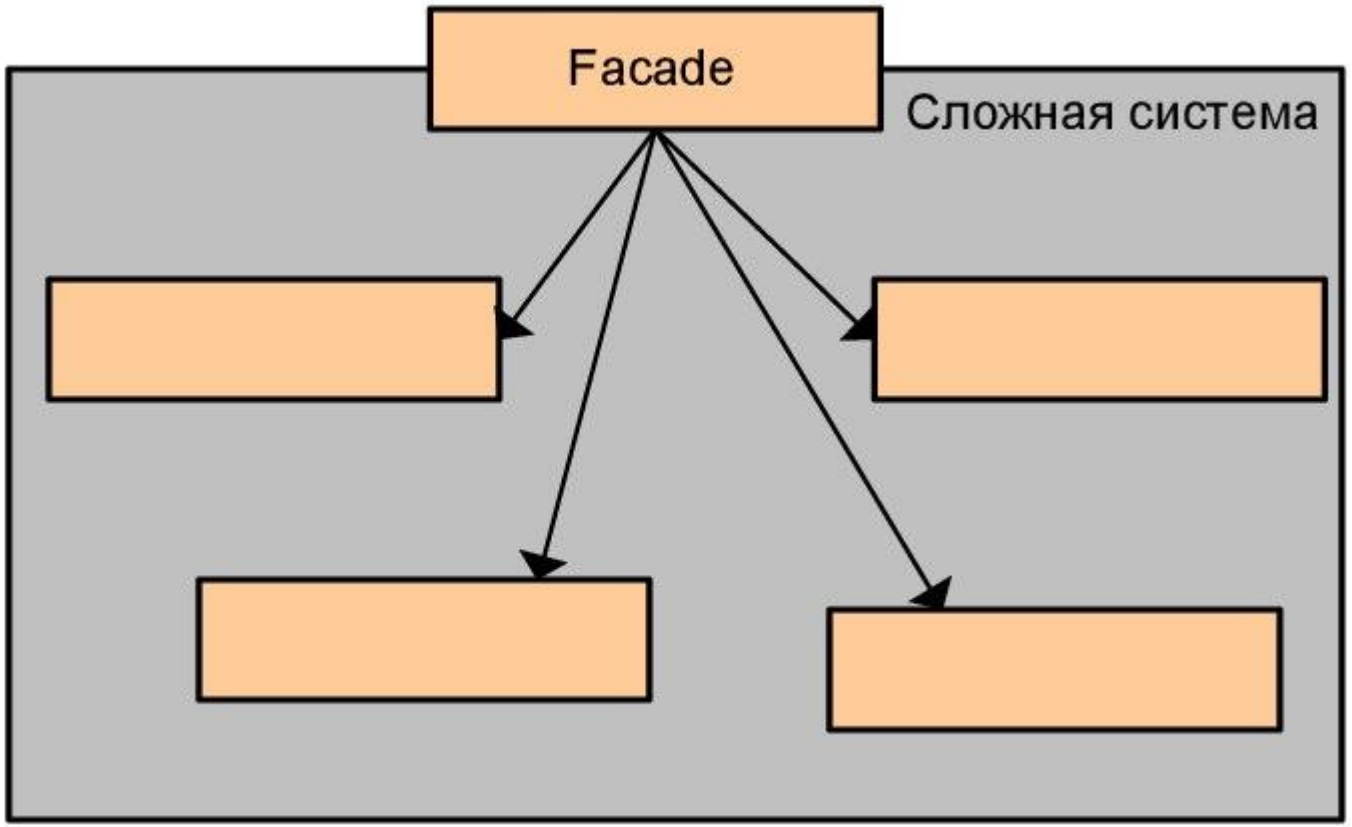
Декоратор. Структура паттерна на языке UML



Декоратор. Структура паттерна на языке C#



Каталог шаблонов проектирования GoF



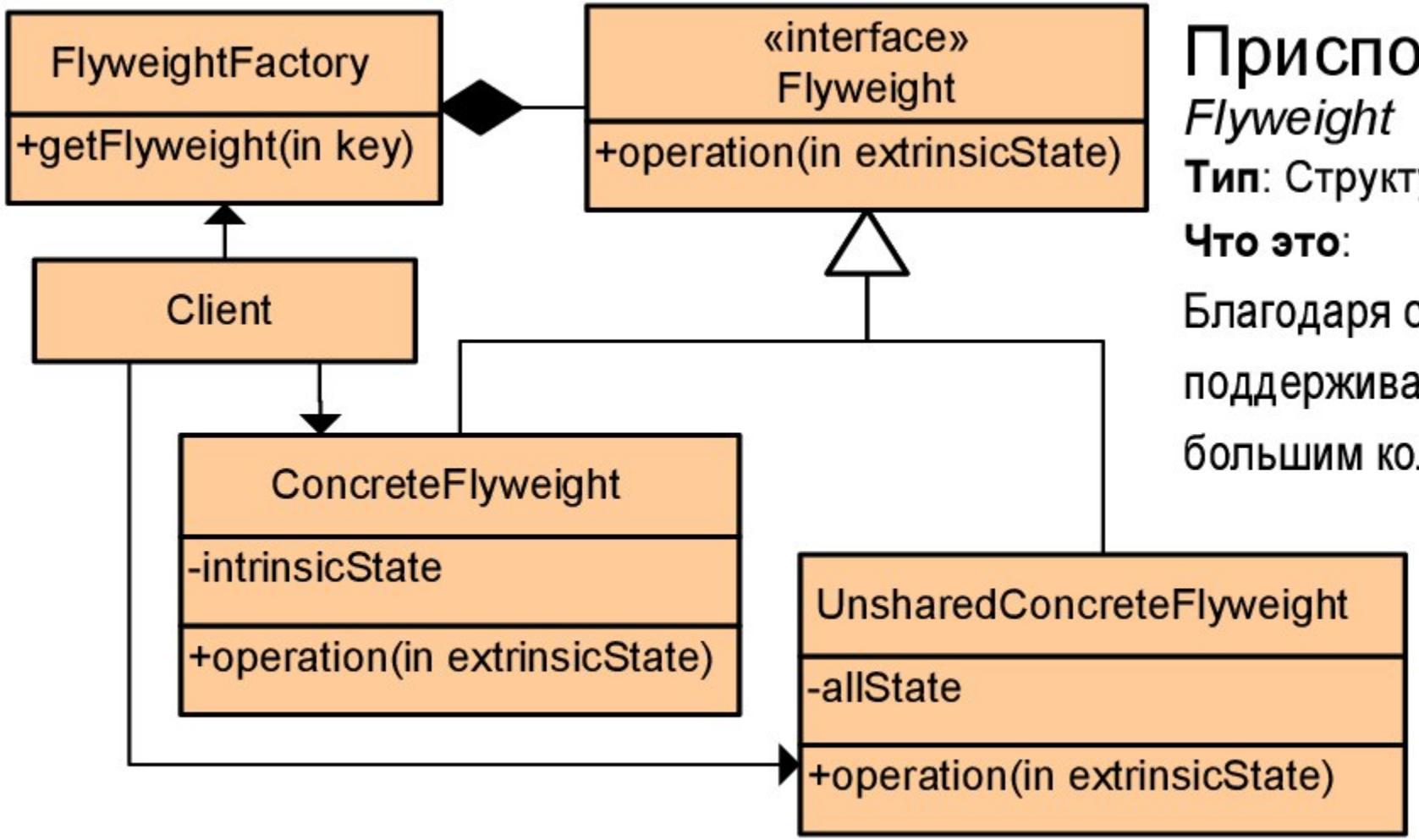
Фасад
Facade

Тип: Структурный

Что это:

Предоставляет единый интерфейс к группе интерфейсов подсистемы. Определяет высокоуровневый интерфейс, делая подсистему проще для использования.

Каталог шаблонов проектирования GoF



Приспособленец

Flyweight

Тип: Структурный

Что это:

Благодаря совместному использованию, поддерживает эффективную работу с большим количеством объектов.

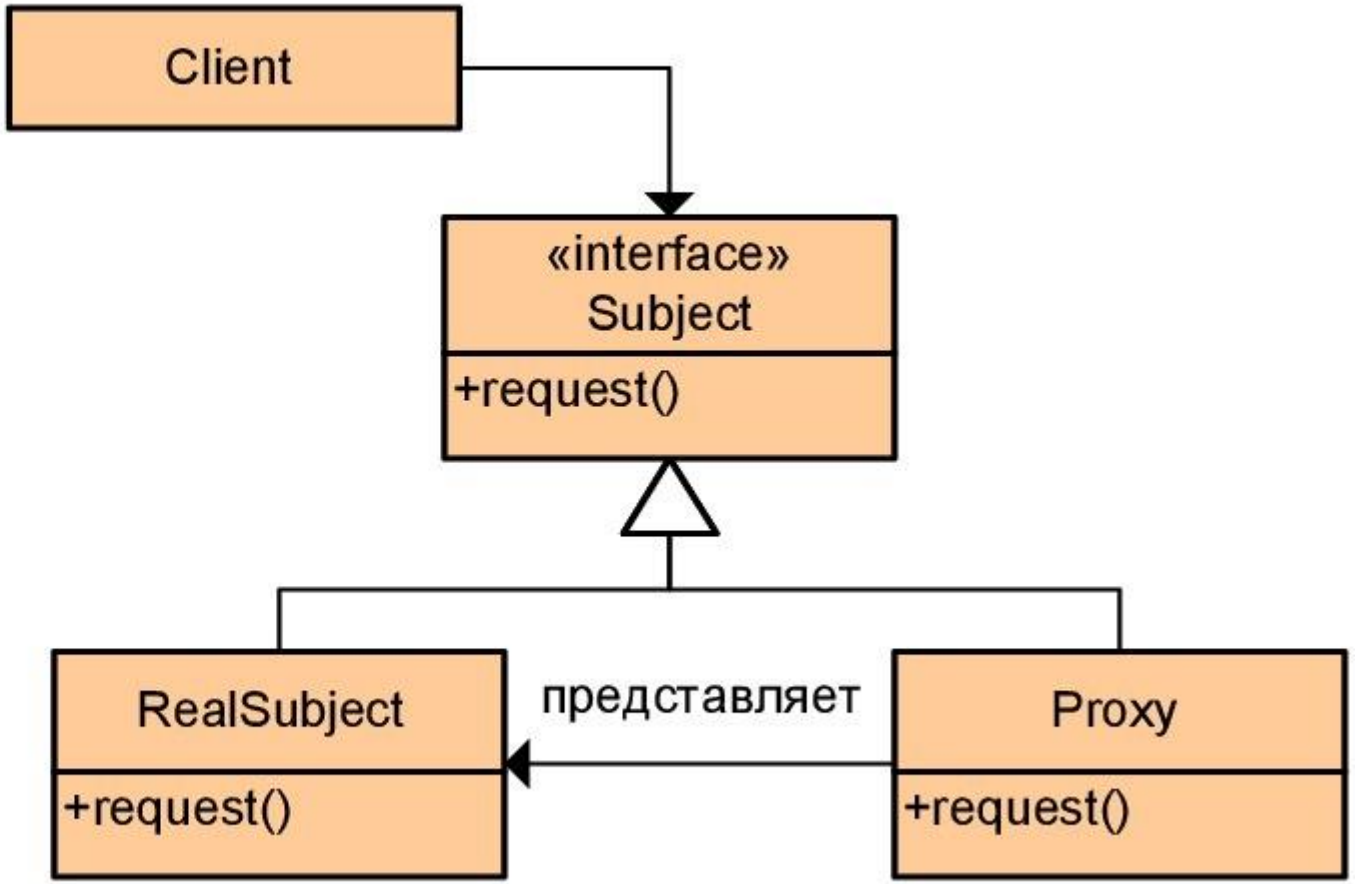
Каталог шаблонов проектирования GoF

Прокси
Proxy

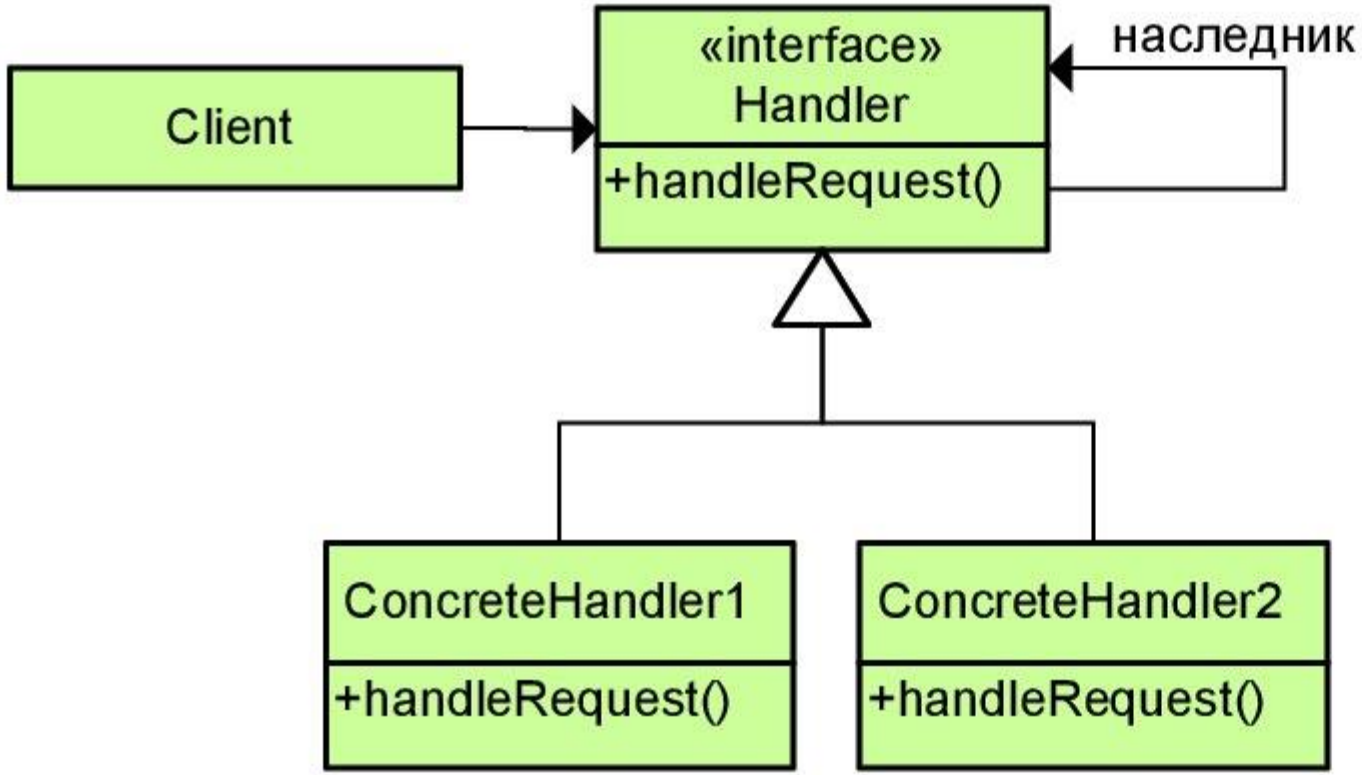
Тип: Структурный

Что это:

Предоставляет замену другого объекта для контроля доступа к нему.



Каталог шаблонов проектирования GoF



Цепочка обязанностей

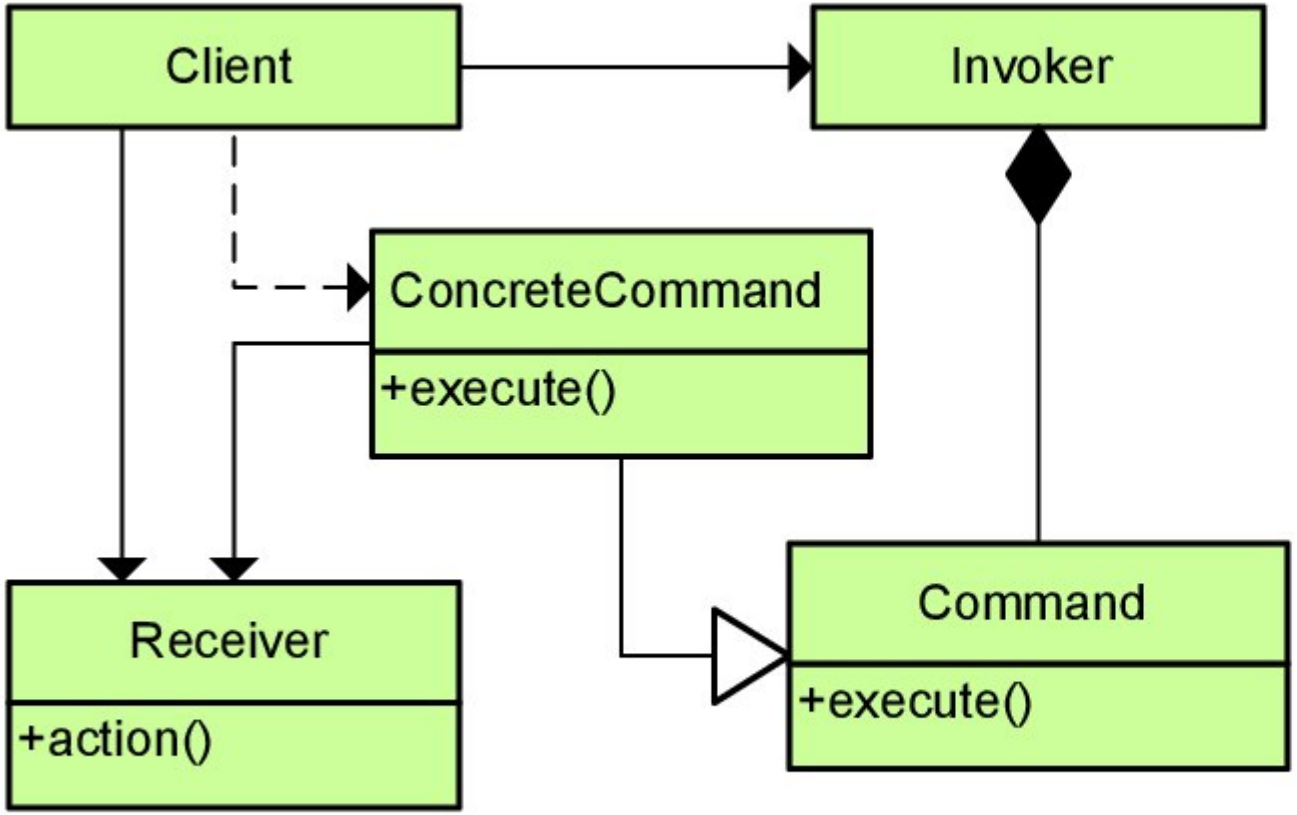
Chain of responsibility

Тип: Поведенческий

Что это:

Избегает связывания отправителя запроса с его получателем, давая возможность обработать запрос более чем одному объекту. Связывает объекты-получатели и передаёт запрос по цепочке пока объект не обработает его.

Каталог шаблонов проектирования GoF



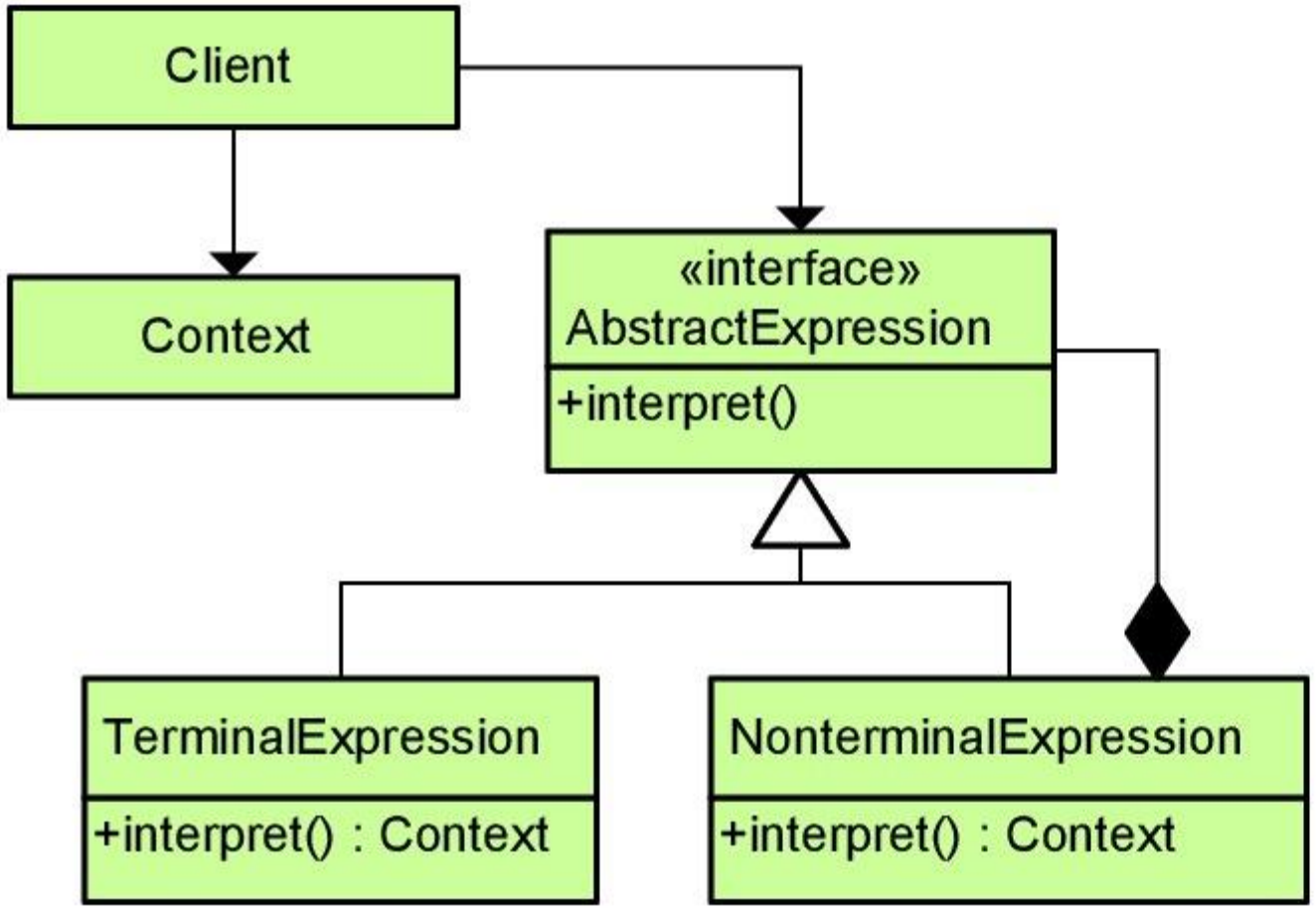
Команда
Command

Тип: Поведенческий

Что это:

Инкапсулирует запрос в виде объекта, позволяя передавать их клиентам в качестве параметров, ставить в очередь, логировать а также поддерживает отмену операций.

Каталог шаблонов проектирования GoF



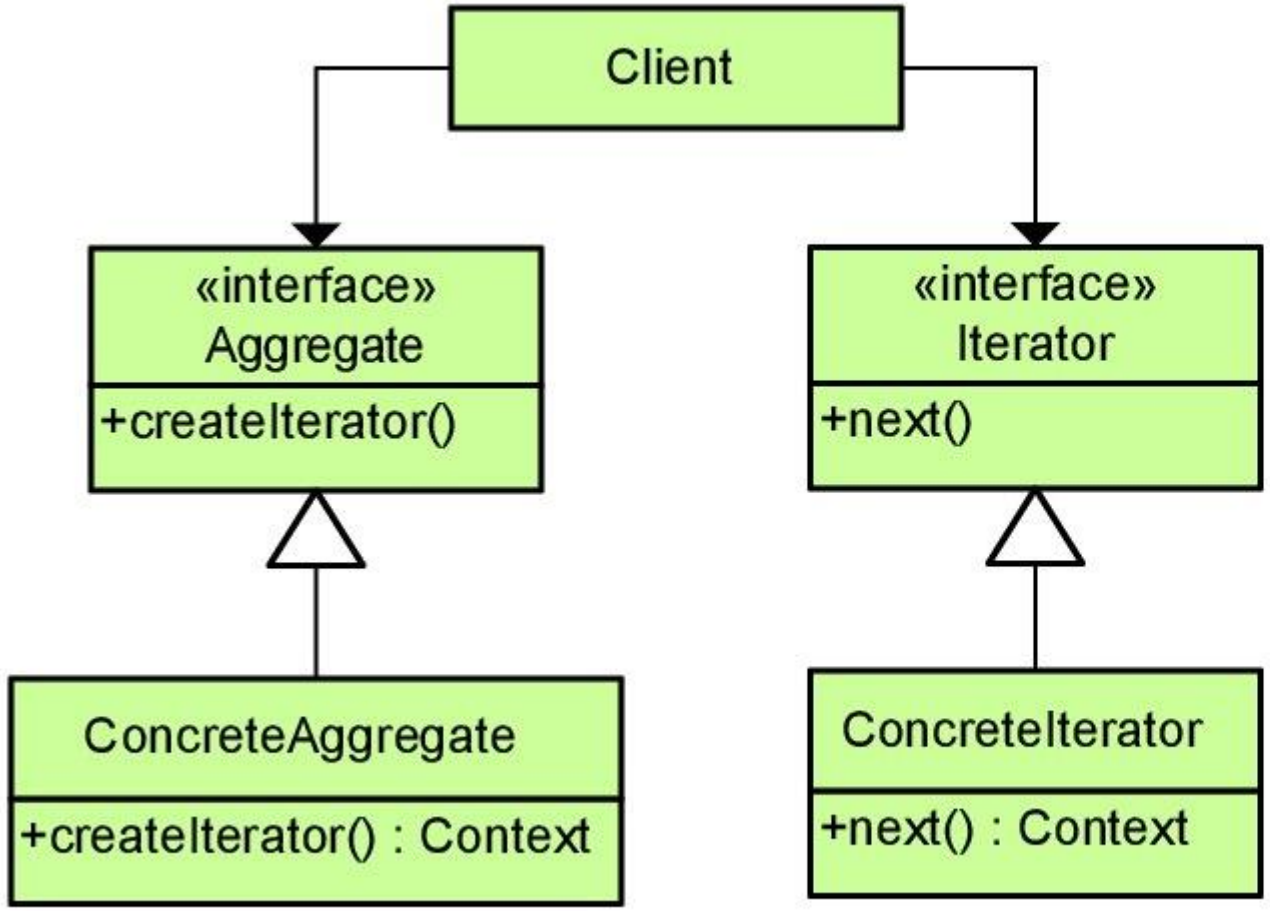
Интерпретатор
Interpreter

Тип: Поведенческий

Что это:

Получая формальный язык, определяет представление его грамматики и интерпретатор, использующий это представление для обработки выражений языка.

Каталог шаблонов проектирования GoF



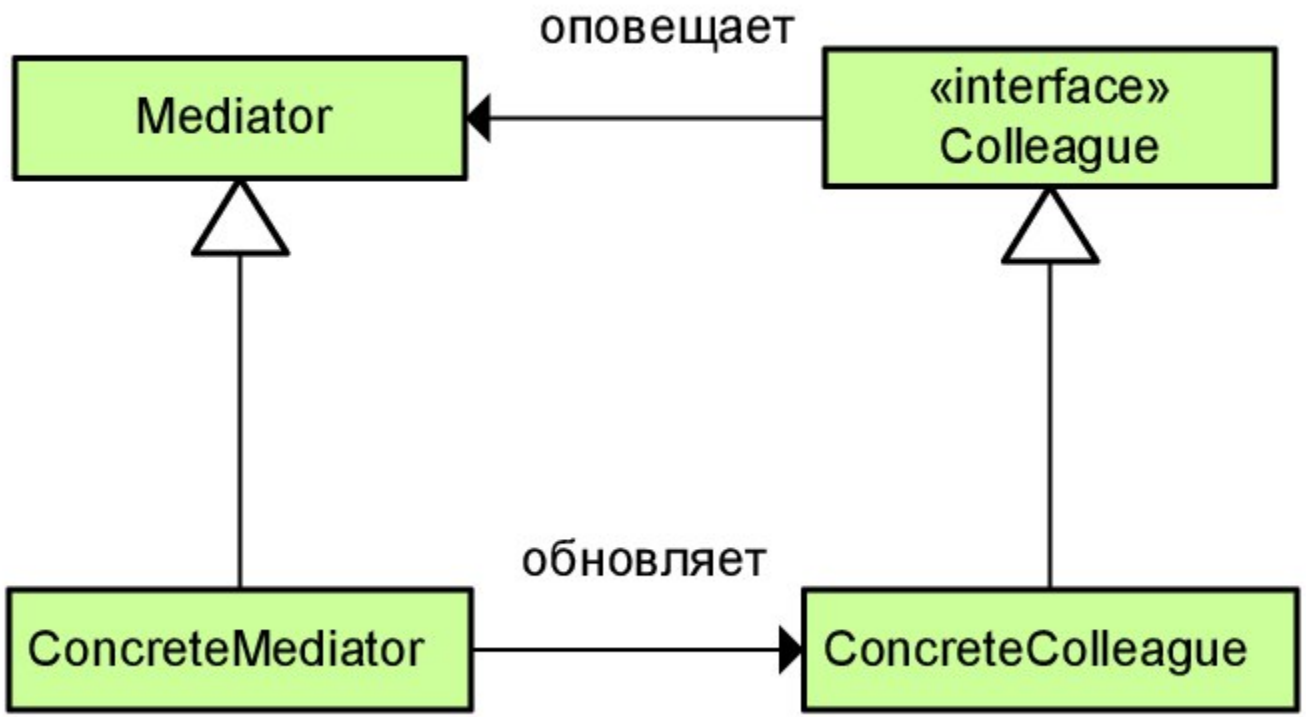
Итератор
Iterator

Тип: Поведенческий

Что это:

Предоставляет способ последовательного доступа к элементам множества, независимо от его внутреннего устройства.

Каталог шаблонов проектирования GoF



Посредник
Mediator

Тип: Поведенческий

Что это:

Определяет объект, инкапсулирующий способ взаимодействия объектов. Обеспечивает слабую связь, избавляя объекты от необходимости прямо ссылать друг на друга и даёт возможность независимо изменять их взаимодействие.

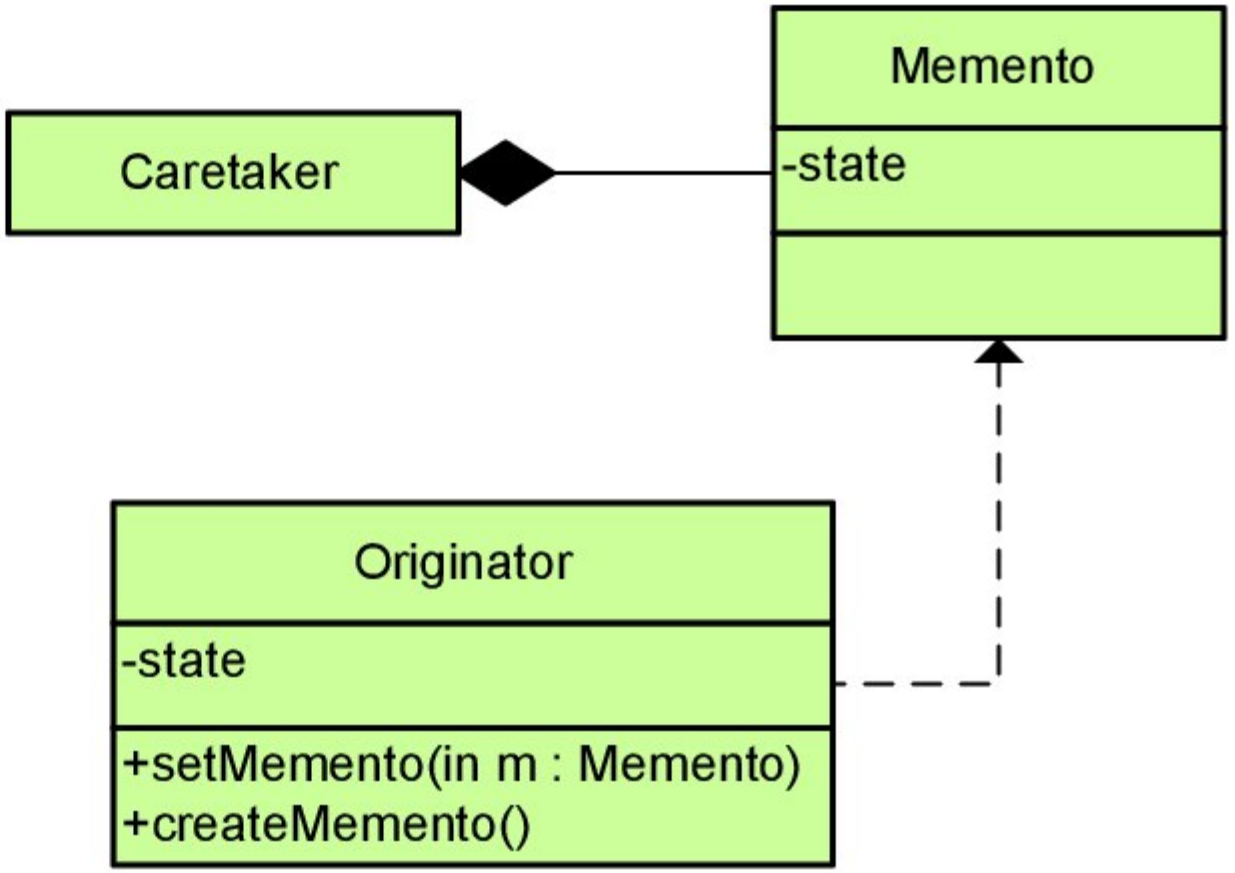
Каталог шаблонов проектирования GoF

Хранитель
Memento

Тип: Поведенческий

Что это:

Не нарушая инкапсуляцию, определяет и сохраняет внутреннее состояние объекта и позволяет позже восстановить объект в этом состоянии.



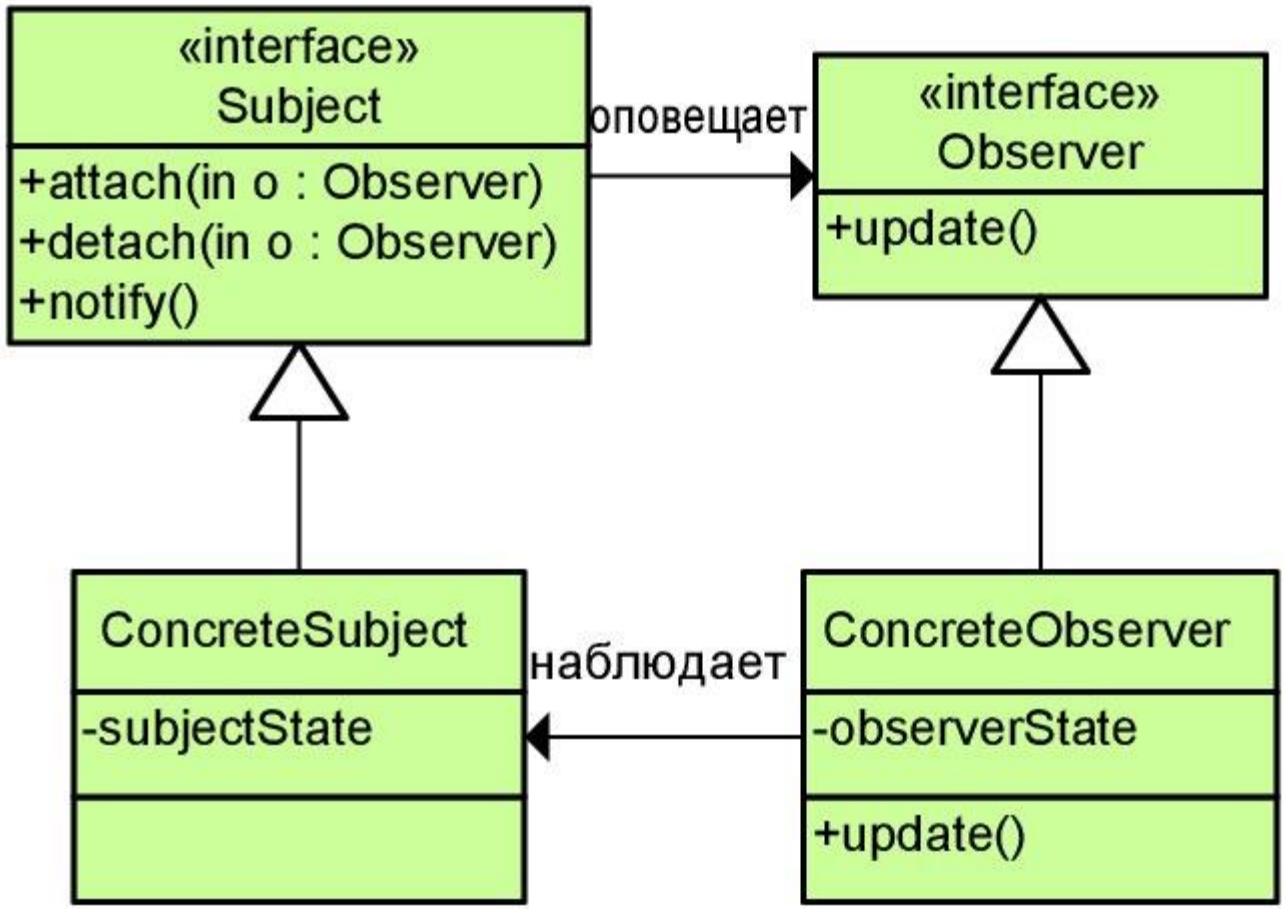
Каталог шаблонов проектирования GoF

Наблюдатель
Observer

Тип: Поведенческий

Что это:

Определяет зависимость “один ко многим” между объектами так, что когда один объект меняет своё состояние, все зависимые объекты оповещаются и обновляются автоматически.



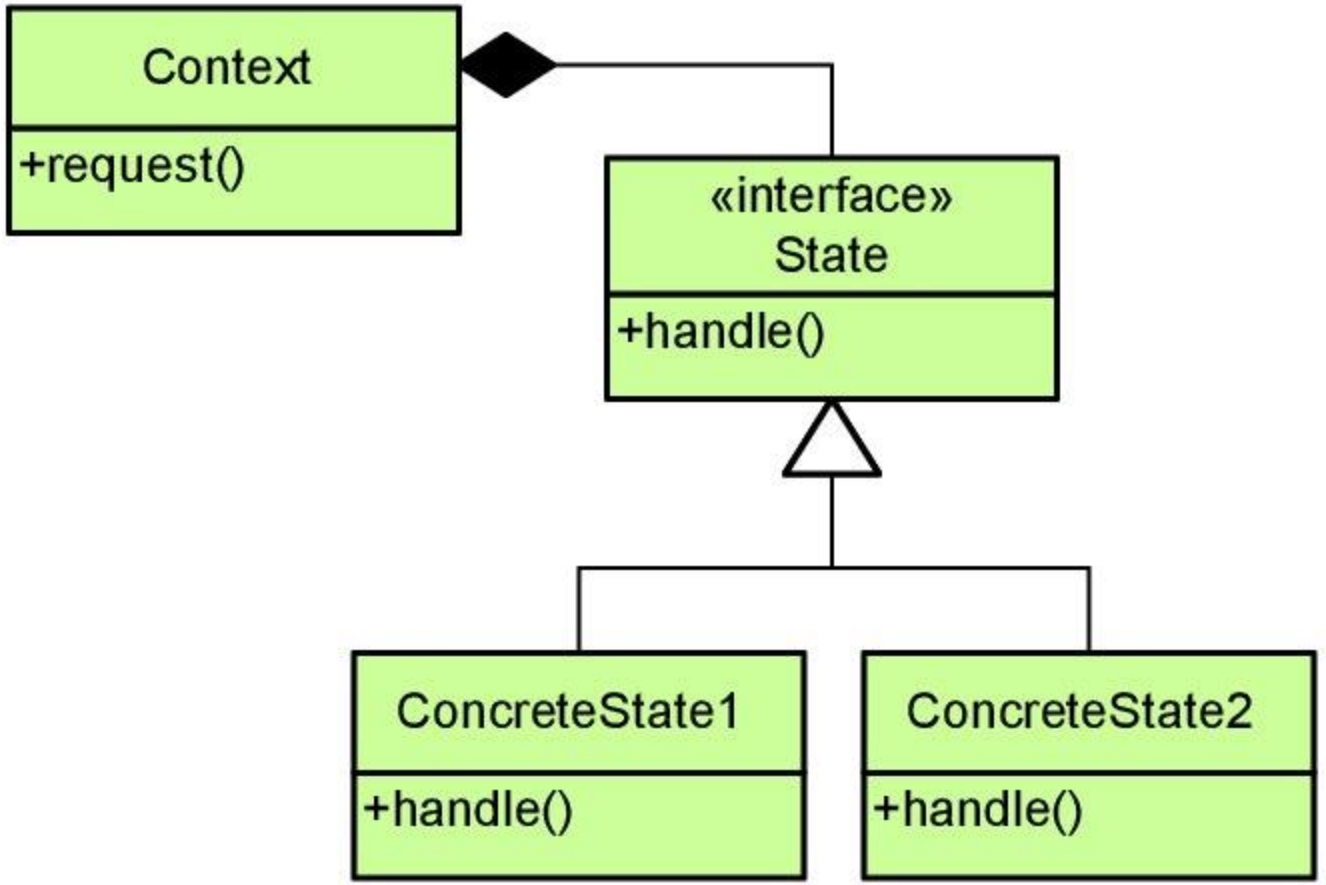
Каталог шаблонов проектирования GoF

Состояние
State

Тип: Поведенческий

Что это:

Позволяет объекту изменять своё поведение в зависимости от внутреннего состояния.



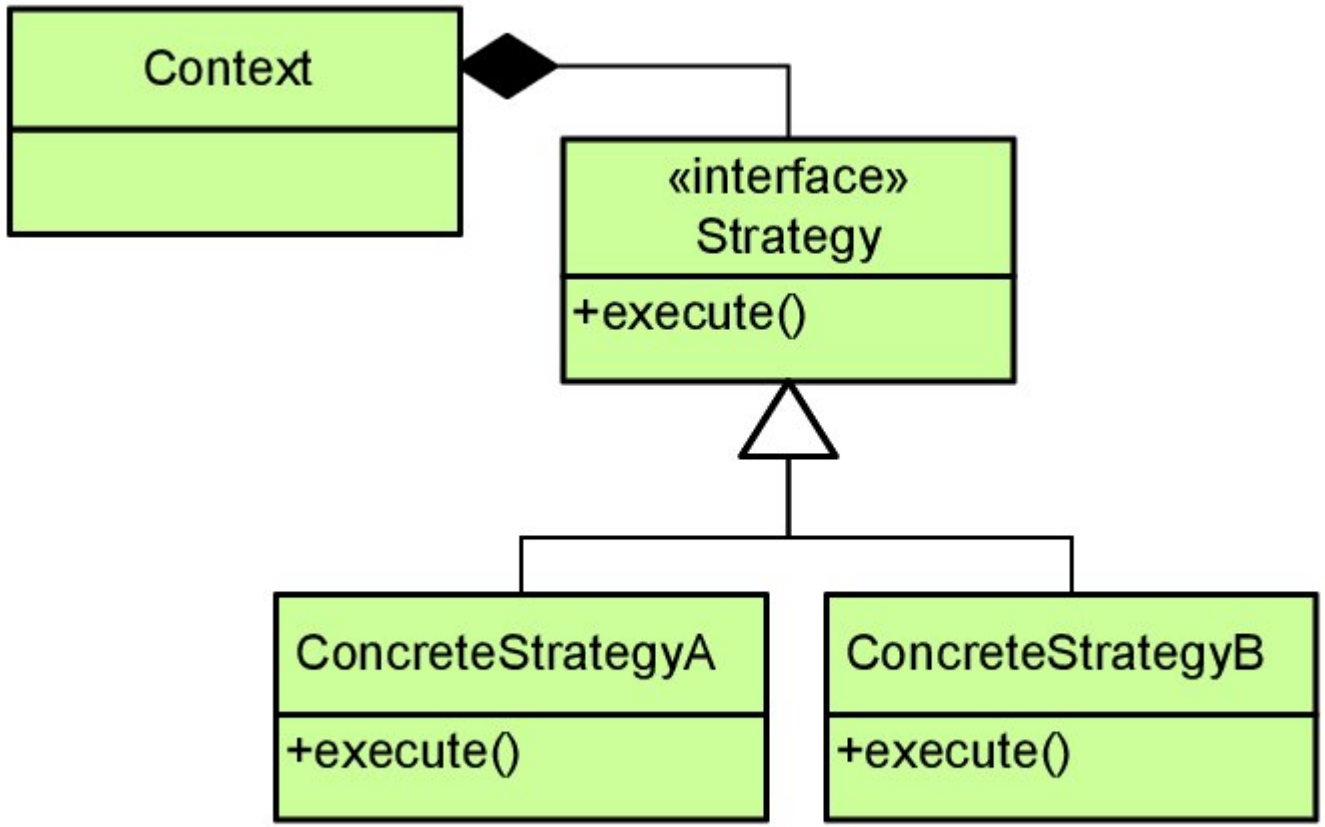
Каталог шаблонов проектирования GoF

Стратегия
Strategy

Тип: Поведенческий

Что это:

Определяет группу алгоритмов, инкапсулирует их и делает взаимозаменяемыми. Позволяет изменять алгоритм независимо от клиентов, его использующих.



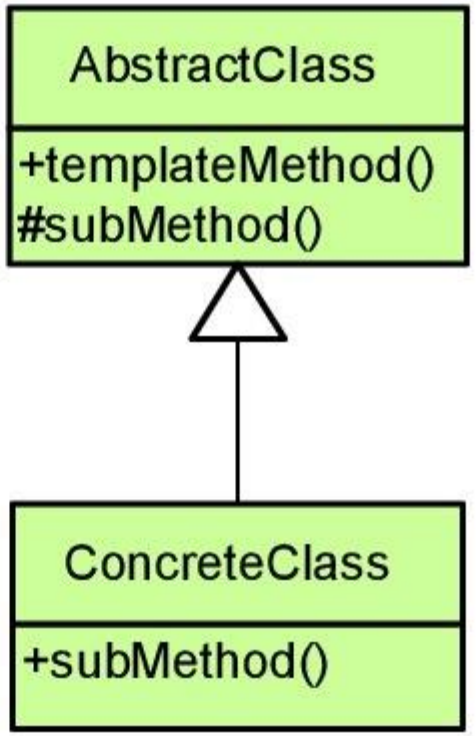
Каталог шаблонов проектирования GoF

Шаблонный метод
Template method

Тип: Поведенческий

Что это:

Определяет алгоритм, некоторые этапы которого делегируются подклассам. Позволяет подклассам переопределить эти этапы, не меняя структуру алгоритма.



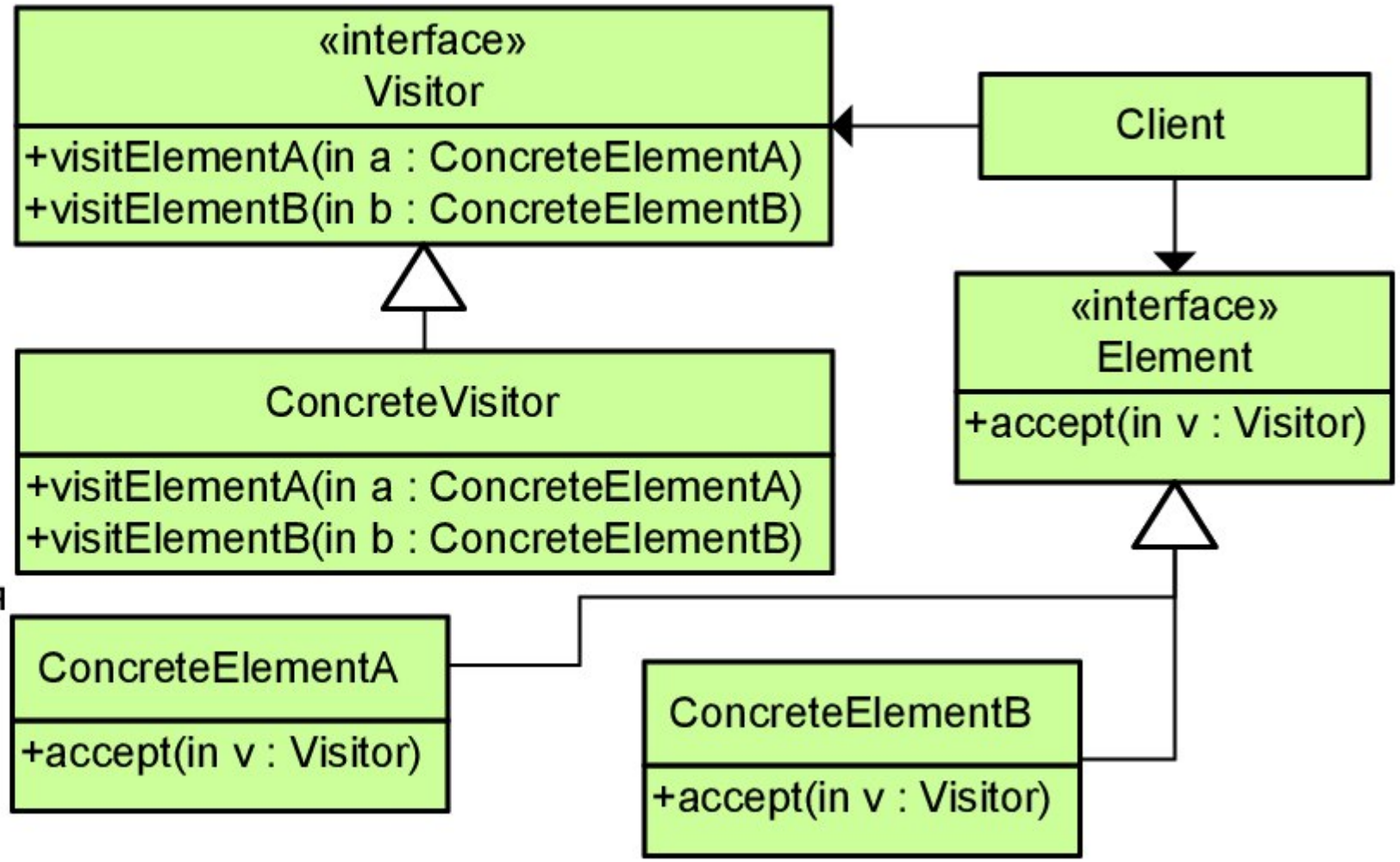
Каталог шаблонов проектирования GoF

Посетитель
Visitor

Тип: Поведенческий

Что это:

Представляет собой операцию, которая будет выполнена над объектами группы классов. Даёт возможность определить новую операцию без изменения кода классов, над которыми эта операция проводится.



SOLID принципы проектирования

Принцип единственной ответственности (Single responsibility)

- *«На каждый объект должна быть возложена одна единственная обязанность»*
- Для этого проверяем, сколько у нас есть причин для изменения класса — если больше одной, то следует разбить данный класс.

Принцип открытости/закрытости (Open-closed)

- *«Программные сущности должны быть открыты для расширения, но закрыты для модификации»*
- Для этого представляем наш класс как «чёрный ящик» и смотрим, можем ли в таком случае изменить его поведение.

Принцип подстановки Барбары Лисков (Liskov substitution)

- *«Объекты в программе могут быть заменены их наследниками без изменения свойств программы»*
- Для этого проверяем, не усилили ли мы предусловия и не ослабили ли постусловия. Если это произошло — то принцип не соблюдается

Принцип разделения интерфейса (Interface segregation)

- *«Много специализированных интерфейсов лучше, чем один универсальный»*
- Проверяем, насколько много интерфейс содержит методов и насколько разные функции накладываются на эти методы, и если необходимо — разбиваем интерфейсы.

Принцип инверсии зависимостей (Dependency Invertion)

- *«Зависимости должны строится относительно абстракций, а не деталей»*
- Проверяем, зависят ли классы от каких-то других классов(непосредственно инстанцируют объекты других классов и т.д.) и если эта зависимость имеет место, заменяем на зависимость от абстракции.

Дополнительные источники

Примеры и разбор паттернов проектирования

- https://bool.dev/blog/detail/grasp-printsipy
- https://bool.dev/blog/detail/gof-design-patterns
- https://refactoring.guru/ru/design-patterns/catalog
- https://metanit.com/sharp/patterns/