Паттерны проектирования

МКД 03.01 «Технология разработки программного обеспечения»

Определения

- **Шаблон проектирования** или **паттерн** (*design pattern*) в разработке ПО — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.
- **Шаблон** – не законченный образец, не код; это пример решения задачи, который можно использовать в различных ситуациях.
- **ОО шаблоны** показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Определения

- **Идиомы** – «низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования. Они не универсальные.
- **Архитектурные шаблоны** – «наивысший» уровень, охватывают архитектуру всей программной системы.
- **Алгоритмы** – шаблоны вычисления, так как решают вычислительные

- Паттерн – именованная конфигурация распределения ответственности по классам

История

- 1970-е гг. архитектор Кристофер Александр, набор шаблонов проектирования.
- 1987 г. Кент Бэк (Kent Beck) и Вард Каннингем (Ward Cunningham), шаблоны разработки ПО на языке Smalltalk.
- 1988 г. Эрих Гамма (Erich Gamma), докторская диссертация о применении шаблонов к разработке ПО.

- 1989-1991 гг. Джеймс Коплин (James Coplien), идиомы для программирования на C++, книга «Advanced C++ Idioms».
- 1991 г. «Банда четырёх» (*Gang of Four*, *GoF*): Эрих Гамма, Ричард Хелм (Richard Helm), Ральф Джонсон (Ralph Johnson), Джон Влиссидс (John Vlissides), книга «Design Patterns — Elements of Reusable Object-Oriented Software», описаны 23 шаблона проектирования.

Плюсы

- Снижение сложности разработки за счёт готовых абстракций для решения целого класса проблем.
- Облегчение коммуникации между разработчиками, позволяя ссылаться на известные шаблоны.
- Снижение количества ошибок за счет унификации деталей решений (модулей, элементов проекта, …).
- Возможность многократно использовать удачное решение (≈ как использование готовых библиотек кода).
- Возможность выбрать наиболее подходящий вариант проектирования из набора шаблонов.

Минусы

- Слепое следование шаблону может привести к усложнению программы.
- Использование шаблона без особых оснований.
- Шаблоны проектирования в ООП – идиоматическое воспроизведение *элементов* функциональных языков.
- Питер Норвиг: «16 из 23 шаблонов *GoF* в Lisp реализуются существенно проще, чем в C++, либо оказываются незаметны».
- Пол Грэхэм: идея шаблонов проектирования = анти-паттерн, сигнал, что система не обладает достаточным уровнем абстракции, и необходима её тщательная переработка.
- Шаблон = «*готовое решение, но не прямое обращение к библиотеке*» = отказ от повторного использования в пользу дублирования.

Типы шаблонов проектирования: Основные (Fundamental)

Название	Оригинал. название	Описание	Описан в Design Patterns
Шаблон делегирования	Delegation pattern	Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту	Н/Д
Шаблон функциональ-ного дизайна	Functional design	Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы	Н/Д
Неизменяемый интерфейс	Immutable interface	Создание неизменяемого объекта	Н/Д
Интерфейс	Interface	Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять	Н/Д
Интерфейс-маркер	Marker interface	В качестве атрибута применяется наличие или отсутствие реализации интерфейса-маркера.	Н/Д
Контейнер свойств	Property Container	Позволяет добавлять дополнительные свойства для класса в контейнер (внутри класса), вместо расширения класса новыми свойствами	Н/Д
Event Channel	Event Channel	Расширяет шаблон Publish/Subscribe, создавая централизованный канал для событий. Использует объект-представитель для подписки и объект-представитель для публикации события в канале. Представитель существует отдельно от реального издателя или подписчика. Подписчик может получать опубликованные события от более чем одного объекта, даже если он зарегистрирован только на одном канале	Н/Д

Типы шаблонов проектирования: Порождающие (Creational)

Абстрагируют процесс инстанцирования. Позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту

Название	Описание
Абстрактная фабрика (Abstract factory)	Класс, который представляет собой интерфейс для создания компонентов системы
Строитель (Builder)	Класс, который представляет собой интерфейс для создания сложного объекта
Фабричный метод (Factory method)	Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать
Отложенная инициали-зация (Lazy initialization)	Объект, инициализируемый во время первого обращения к нему
Пул одиночек (Multiton)	Гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним
Объектный пул (Object pool)	Класс, который представляет собой интерфейс для работы с набором инициализированных и готовых к использованию объектов
Прототип (Prototype)	Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор
Получение ресурса есть инициализация (Resour-ce acquisition is initializat)	Получение некоторого ресурса совмещается с инициализацией, а освобождение — с уничтожением объекта
Одиночка (Singleton)	Класс, который может иметь только один экземпляр

Типы шаблонов проектирования: Структурные (Structural)

Определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу

Название	Описание
Адаптер (Adapter / Wrapper)	Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс
Мост (Bridge)	Структура, позволяющая изменять интерфейс обращения и интерфейс реализации класса независимо Да Компоновщик Composite Объект, который объединяет в себе объекты, подобные ему самому
Декоратор или Обёртка (Wrapper/ Decorator)	Класс, расширяющий функциональность другого класса без использования наследования
Фасад (Facade)	Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое
Единая точка входа (Front Controller)	Обеспечивает унифицированный интерфейс для интерфейсов в подсистеме. Front Controller определяет высокоуровневый интерфейс, упрощающий использование подсистемы
Приспособленец (Flyweight)	Это объект, представляющий себя как уникальный экземпляр в разных местах программы, но по факту не являющийся таковым
Заместитель (Proxy)	Объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него

Типы шаблонов проектирования: Поведенческие (Behavioral)

определяют взаимодействие между объектами, увеличивая таким образом его гибкость

Название	Описание
Цепочка обязанностей (Chain of responsibility)	Предназначен для организации в системе уровней ответственности
Команда (Command, Action, Transaction)	Представляет действие. Объект команды заключает в себе само действие и его параметры
Интерпретатор (Interpreter)	Решает часто встречающуюся, но подверженную изменениям, задачу
Итератор (Iterator, Cursor)	Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации
Посредник (Mediator)	Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга
Хранитель (Memento, Token)	Позволяет не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях
Null Object	Предотвращает нулевые указатели, предоставляя объект «по умолчанию»

Типы шаблонов проектирования: Поведенческие (Behavioral)

Название	Описание
Наблюдатель (Observer, Publish-Subscribe,Listener)	Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии
Слуга (Servant)	Используется для обеспечения общей функциональности группе классов
Спецификация (Specification)	Служит для связывания бизнес-логики
Состояние (State, Objects for States)	Используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния
Стратегия (Strategy)	Предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости
Шаблонный метод (Template method)	Определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом
Посетитель (Visitor)	Описывает операцию, которая выполняется над объектами других классов. При изменении класса Visitor нет необходимости изменять обслуживаемые классы
Одноразовый посетитель (Single-serving visitor)	Оптимизирует реализацию шаблона посетитель, который инициализируется, единожды используется, и затем удаляется
Иерархический посетитель (Hierarchical visitor)	Предоставляет способ обхода всех вершин иерархической структуры данных (напр. древовидной)

Типы шаблонов проектирования: Частные

- **Шаблоны параллельного программирования -** используются для более эффективного написания многопоточных программ, и предоставляет готовые решения проблем синхронизации.
- **Active Object** Служит для отделения потока выполнения метода от потока, в котором он был вызван. Использует шаблоны асинхронный вызов методов и планировщик
- **Balking** Служит для выполнения действия над объектом только тогда, когда тот находится в корректном состоянии.
- **Обмен сообщениями (Messaging design pattern, MDP)** Позволяет компонентам и приложениям обмениваться информацией (сообщениями)

- **Шаблоны архитектуры системы**
  - Model-View-Controller (MVC)
  - Модель-представление-контроллер
  - Model-View-Presenter
  - Model-View-View Model
  - Presentation-Abstraction-Control

Типы шаблонов проектирования: Прочие

- **Carrier Rider Mapper** описывают предоставление доступа к хранимой информации
- **Аналитические шаблоны** описывают основной подход для составления требований для программного обеспечения (requirement analysis) до начала самого процесса программной разработки
- **Коммуникационные шаблоны** описывают процесс общения между отдельными участниками/сотрудниками организации
- **Организационные шаблоны** описывают организационную иерархию предприятия/фирмы
- **Анти-паттерны (Anti-Design-Patterns)** описывают, как не следует поступать при разработке программ, показывая характерные ошибки в дизайне и в реализации

- **Паттерны проектирования** — это проверенные и готовые к использованию решения часто возникающих в повседневном программировании задач.
- Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее.
- Шаблон проектирования, подходящий под задачу, реализуется в каждом конкретном случае.
- Шаблон не зависит от языка программирования. Хороший шаблон легко реализуется в большинстве, если не во всех языках, в зависимости от выразительных средств языка.
- Шаблон, будучи примененным неправильно или к неподходящей задаче, может принести немало проблем.
- Правильно примененный шаблон поможет решить задачу легко и просто.

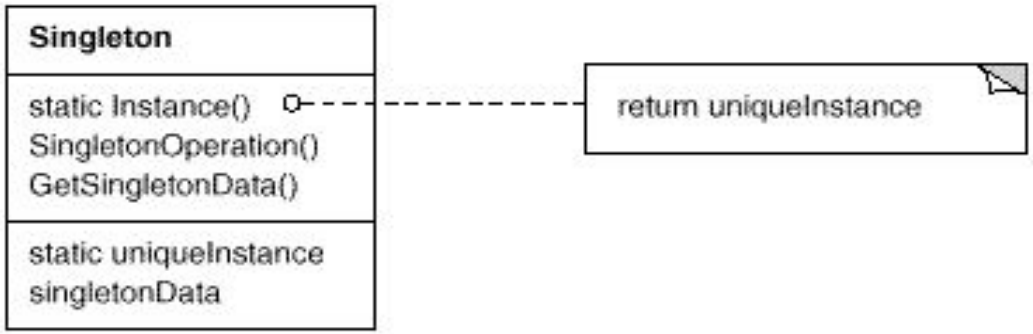
- Существует три типа шаблонов:
- структурные;
- порождающие;
- поведенческие.
- **Структурные** шаблоны определяют отношения между классами и объектами, позволяя им работать совместно.
- **Порождающие** шаблоны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом.
- **Поведенческие** шаблоны используются для того, чтобы упростить взаимодействие между сущностями.

Singleton

- Название и классификация
  Одиночка – паттерн, порождающий объекты
- Назначение
  Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа
- Применимость
  - Должен быть ровно один экземпляр некоторого класса, легко доступный всем клиентам
  - Единственный экземпляр должен расширяться путем порождения подклассов, и клиентам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода

Singleton: структура

- Структура

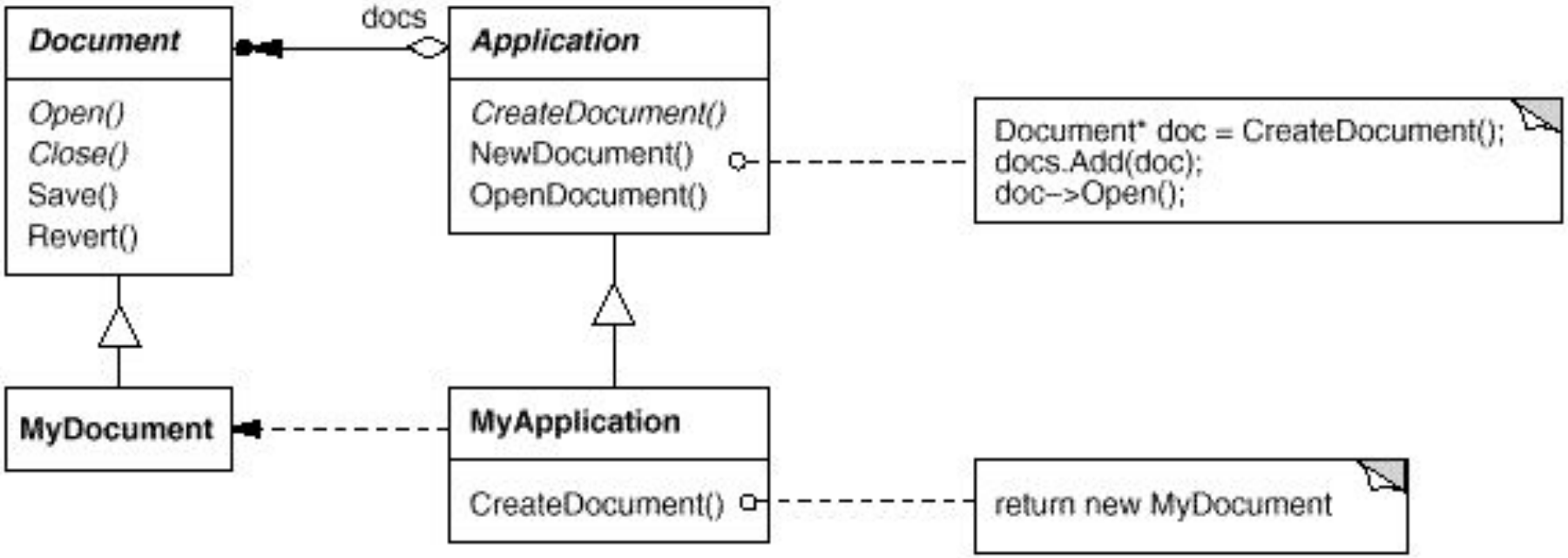


- Участники
  - Singleton – одиночка, Определяет операцию `Instance` , которая позволяет клиентам получать доступ к единственному экземпляру

Factory Method

- Название и классификация
  Фабричный метод – паттерн, порождающий объекты
- Назначение
  Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать инстанцирование в подклассы
- Известен также под именем
  Virtual Constructor

Factory Method
Мотивация



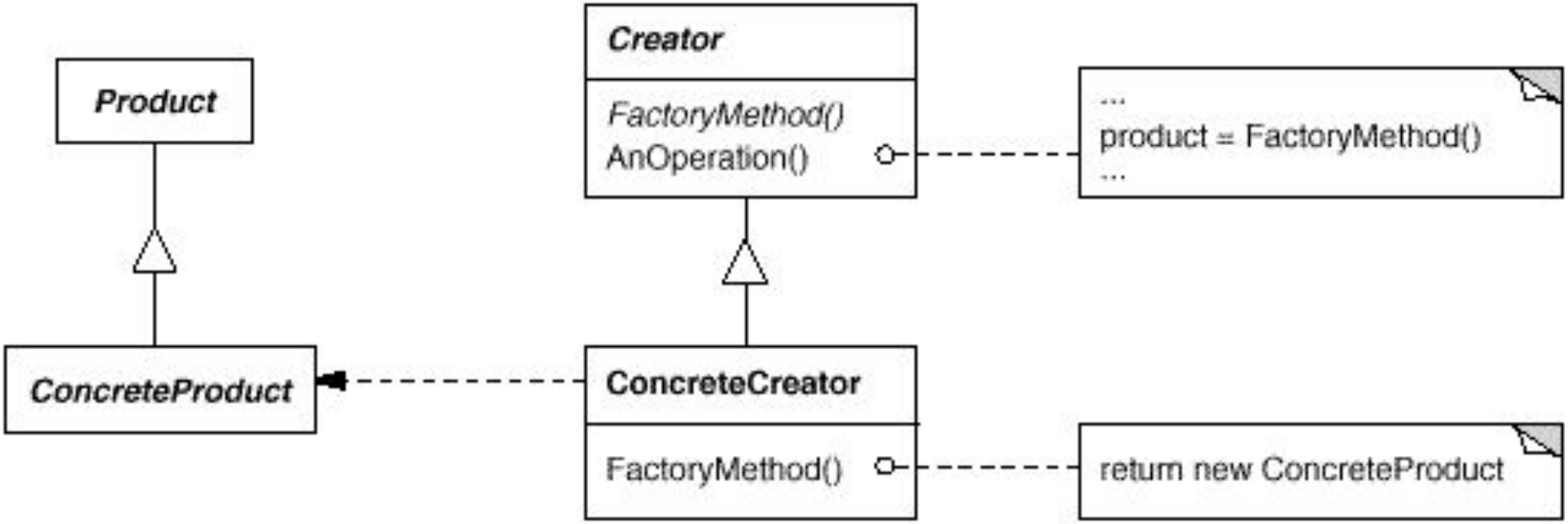
Factory Method
Применимость

Используйте паттерн фабричный метод, когда:

- классу заранее неизвестно, объекты каких классов ему нужно создавать
- класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и вы планируете локализовать знание о том, какой класс принимает эти обязанности на себя

Factory Method

Структура



Factory Method
Особенности

- Две основных разновидности:
  - класс Creator – абстрактный
  - Creator – конкретный класс, в котором по умолчанию есть реализация фабричного метода
- Параметризованные фабричные методы

Структурные паттерны

Adapter Адаптер	Изменение интерфейса
Bridge Мост	Разделение реализации объекта
Composite Компоновщик	Сложная структура и состав объекта
Decorator Декоратор	Изменение обязанностей объекта без порождения подкласса
Facade Фасад	Интерфейс к подсистеме
Flyweight Приспособленец	Снижение накладных расходов на хранение объектов
Proxy Заместитель	Способ доступа к объекту, смена его местоположения

Структурные паттерны

Adapter Адаптер	Изменение интерфейса
Bridge Мост	Разделение реализации объекта
Composite Компоновщик	Сложная структура и состав объекта
Decorator Декоратор	Изменение обязанностей объекта без порождения подкласса
Facade Фасад	Интерфейс к подсистеме
Flyweight Приспособленец	Снижение накладных расходов на хранение объектов
Proxy Заместитель	Способ доступа к объекту, смена его местоположения

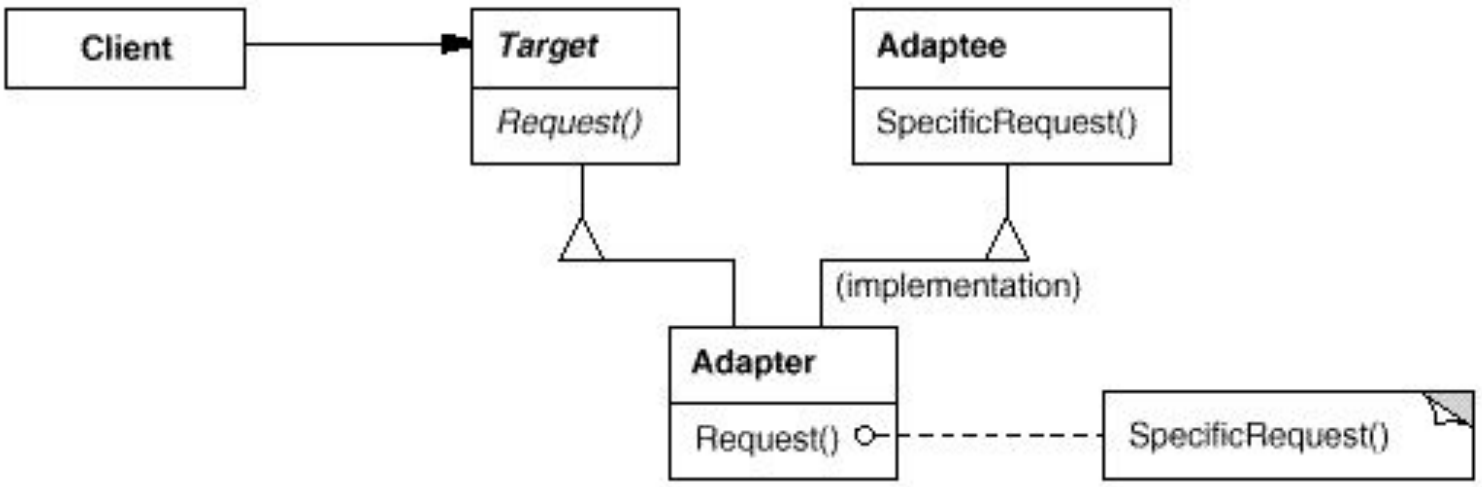
Adapter

- Название и классификация
  Адаптер – паттерн, структурирующий классы и объекты
- Назначение
  Преобразует интерфейс одного класса в интерфейс другого, который ожидают клиенты. Адаптер обеспечивает совместную работу классов с несовместимыми интерфейсами, которая без него была бы невозможна
- Известен также под именем
  Wrapper

Adapter
Применимость

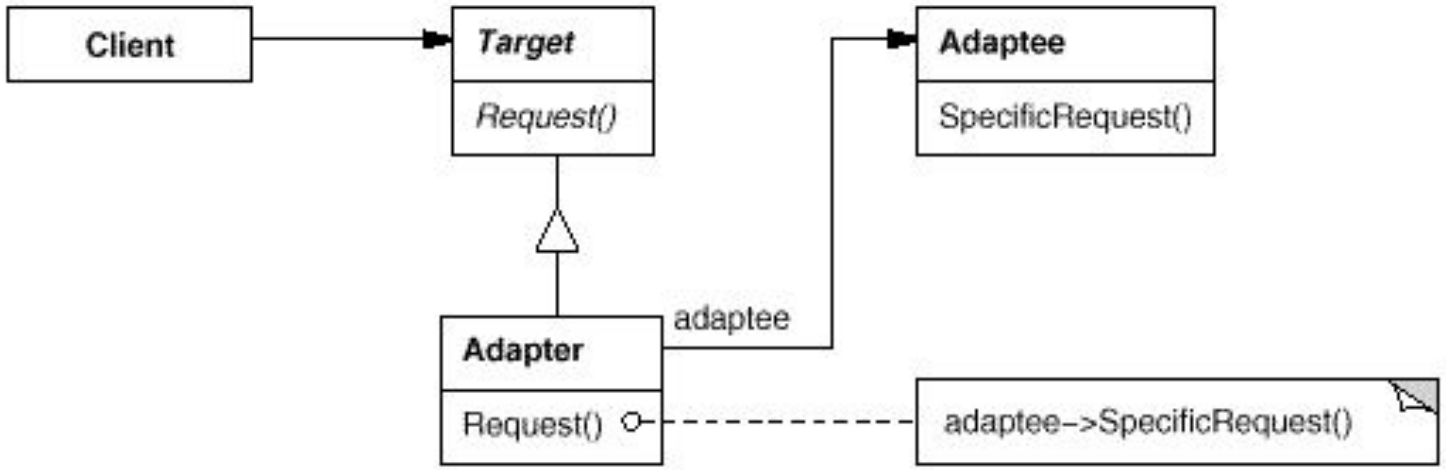
- Применяйте адаптер классов, когда:
  - хотите использовать существующий класс, но его интерфейс не соответствует вашим потребностям
  - собираетесь создать повторно используемый класс, который должен взаимодействовать с заранее неизвестными или не связанными с ним классами, имеющими несовместимые интерфейсы
- Применяйте адаптер объектов, когда
  - нужно использовать несколько существующих подклассов, но непрактично адаптировать их интерфейсы путем порождения новых подклассов от каждого. В этом случае адаптер объектов может приспосабливать интерфейс их общего родительского класса

Adapter (класса)



- Неприменим, если требуется адаптировать не только конкретный класс, но и его подклассы
- Возможно изменение в адаптере операций адаптируемого класса
- Вводится только один объект (непосредственно адаптера)

Adapter (объекта)



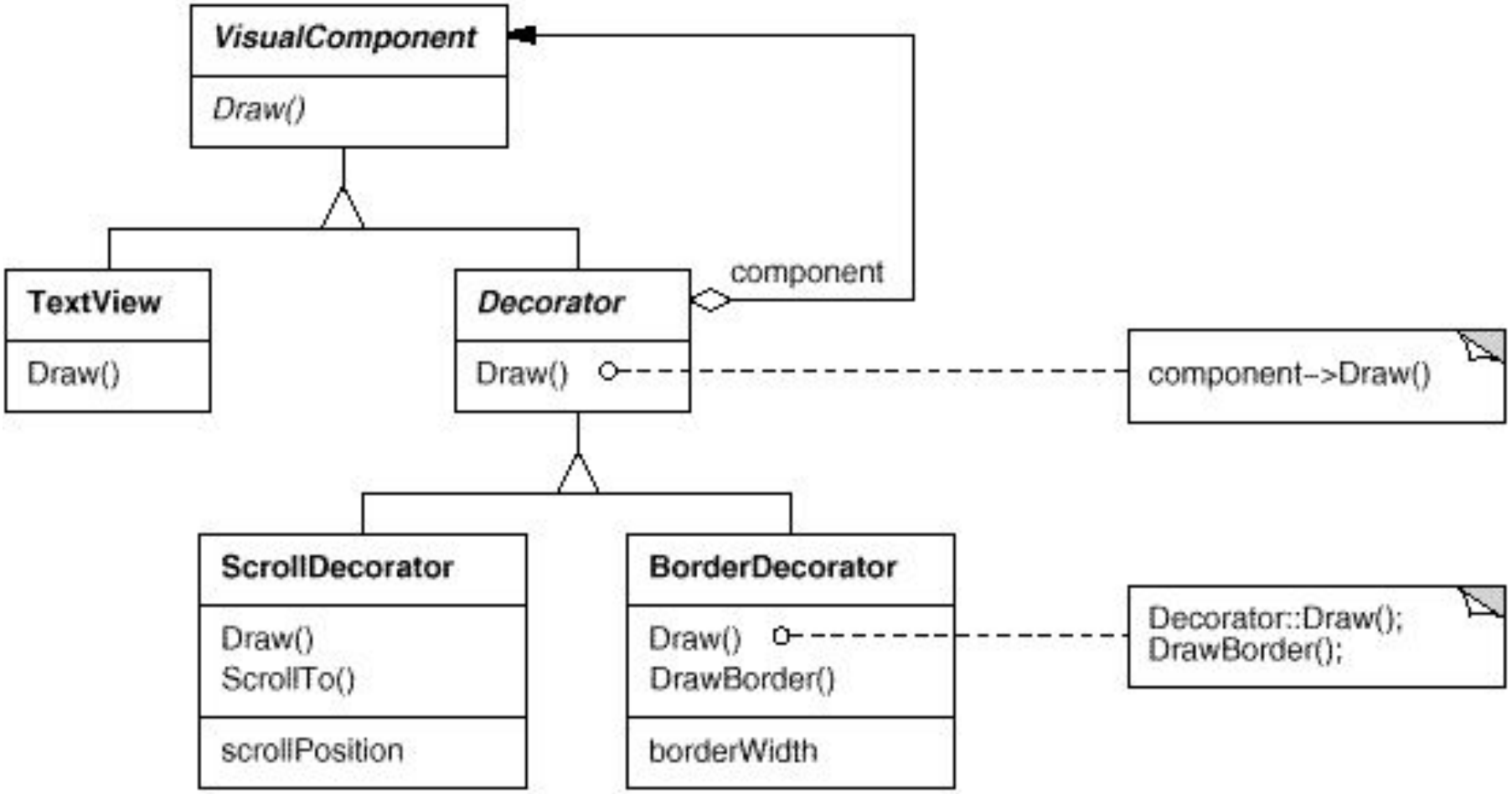
- Один адаптер может работать со множеством адаптируемых объектов, включая объекты подклассов
- Затруднено замещение операций адаптируемого класса

Decorator

- Название и классификация
  Декоратор – паттерн, структурирующий объекты
- Назначение
  Динамически добавляет объекту новые обязанности. Является гибкой альтернативой порождению подклассов с целью расширения функциональности
- Известен также под именем
  Wrapper

Decorator

Мотивация



Decorator

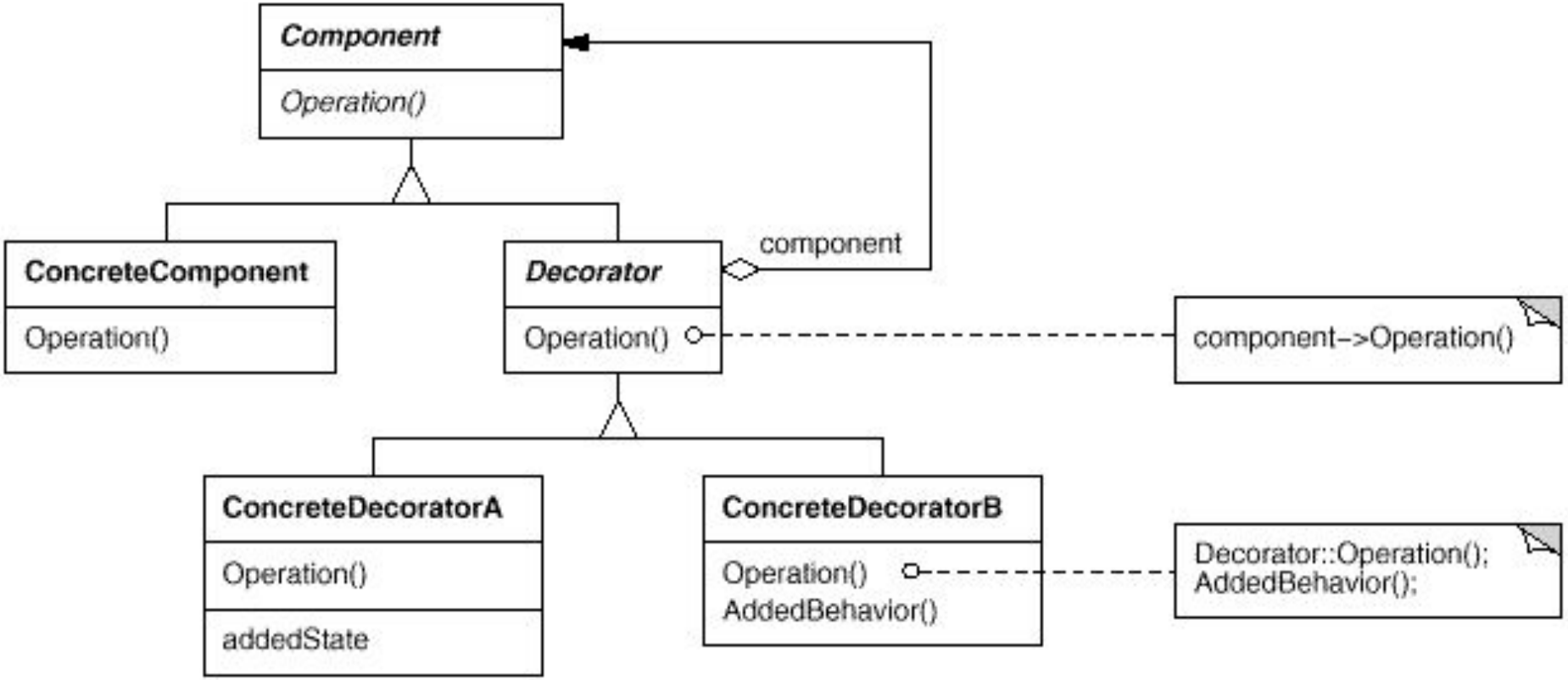
Применимость

Используйте паттерн декоратор:

- для динамического, прозрачного для клиентов добавления обязанностей объектам
- для реализации обязанностей, которые могут быть сняты с объекта
- когда расширение путем порождения подклассов по каким-то причинам неудобно или невозможно

Decorator

Структура



Decorator
Особенности

- Большая гибкость, чем у статического наследования
- Создание цепочек декораторов, в том числе из одних и тех же в одной цепочке
- Позволяет избежать перегруженных функциями классов на верхних уровнях иерархии
- Декоратор и его компонент не идентичны
- Множество мелких объектов

Decorator
Особенности

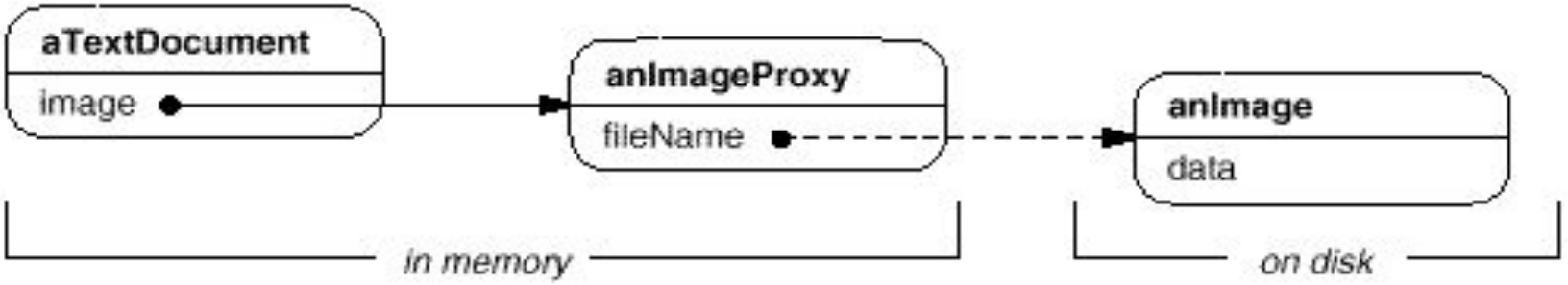
- Соответствие интерфейсов декоратора и декорируемого объекта
- Возможное отсутствие абстрактного класса декоратора
- Облегчение, по возможности, декорируемого класса
- Изменяется «облик», а не внутренне устройство объекта

Proxy

- Название и классификация
  Заместитель – паттерн, структурирующий объекты
- Назначение
  Является суррогатом
  другого объекта
  и контролирует доступ
  к нему
- Известен также под именем
  Surrogate

Proxy

- Мотивация



- Применимость

  Заместитель применим во всех случаях, когда возникает необходимость сослаться на объект более изощренно, чем это возможно, если использовать простую ссылку

Proxy
Применимость

- Удаленный заместитель
  предоставляет локального представителя вместо объекта, находящегося в другом адресном пространстве
- Виртуальный заместитель
  создает тяжелые объекты по требованию
- Защищающий заместитель
  контролирует доступ к исходному объекту

Proxy
Применимость

- Умная ссылка

  замена обычного указателя:
  - подсчет числа ссылок на реальный объект
  - загрузка объекта в память при первом обращении к нему
  - проверка и установка блокировки на реальный объект при обращении к нему, чтобы никакой другой объект не смог в это время изменить его

Proxy

Структура

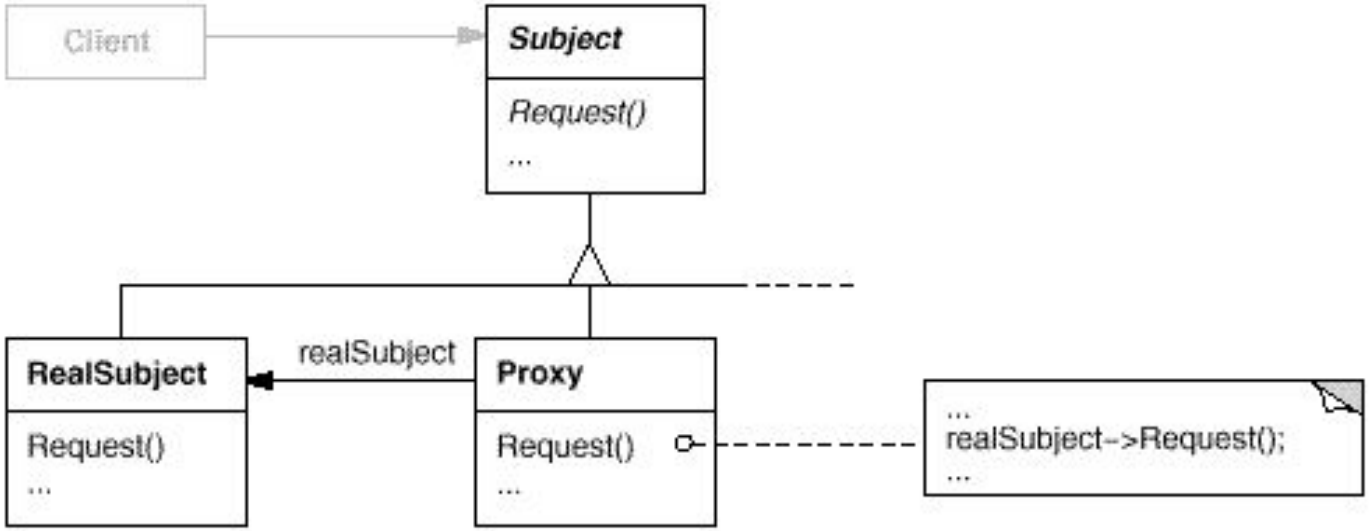
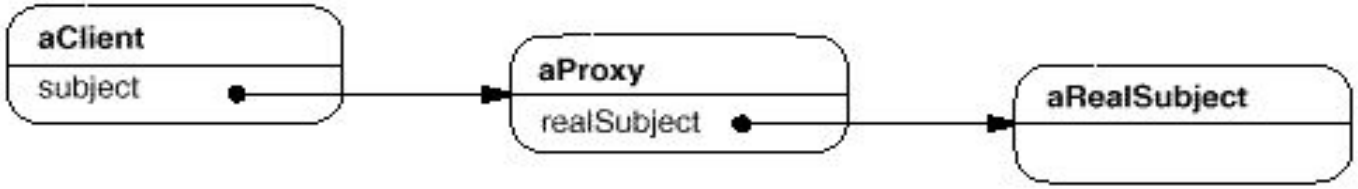


Диаграмма объектов



Паттерны поведения

Interpreter Интерпретатор	Грамматика и интерпретация языка
Iterator Итератор	Способ обхода элементов агрегата
Command Команда	Время и способ выполнения запроса за счет заключения запроса в объект
Observer Наблюдатель	Способ, которым зависимые объекты поддерживают себя в актуальном состоянии
Visitor Посетитель	Операции, которые можно применить к объекту (добавление операций к объектам)

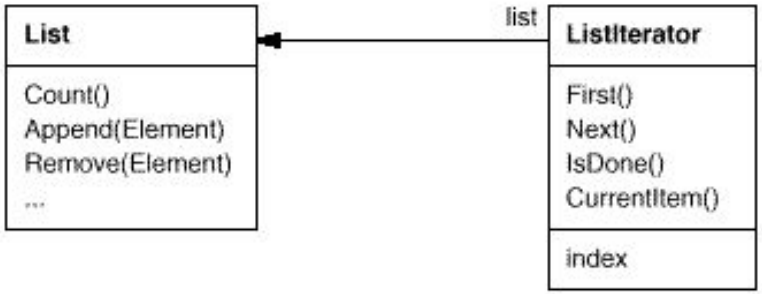
Паттерны поведения

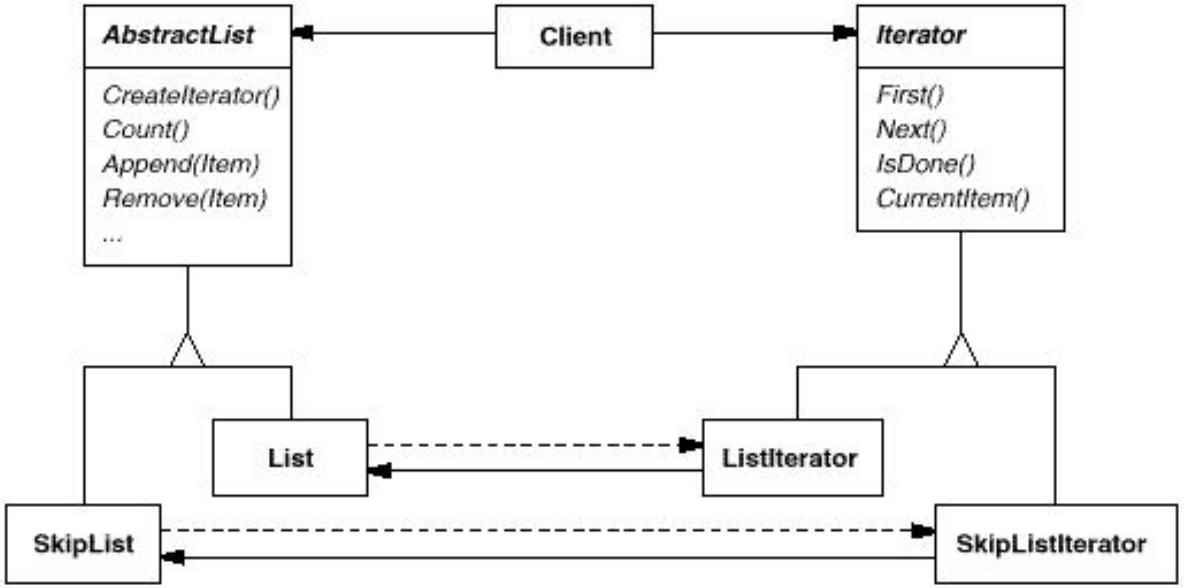
Mediator Посредник	Способ кооперации взаимодействующих объектов через промежуточный
State Состояние	Варьирование поведения объекта в зависимости от его состояния
Strategy Стратегия	Заключение алгоритма в объект, возможность замены алгоритмов
Memento Хранитель	Закрытая информация, хранящаяся вне объекта, и время ее сохранения
Chain of Responsibility Цепочка обязанностей	Набор объектов, выполняющих запрос
Template Method Шаблонный метод	Выделение в абстракцию шагов алгоритма

Iterator

- Название и классификация
  Итератор – паттерн поведения объектов
- Назначение
  Предоставляет способ последовательного доступа ко всем элементам составного объекта, не раскрывая его внутреннего представления
- Известен также под именем
  Cursor

Iterator
Мотивация





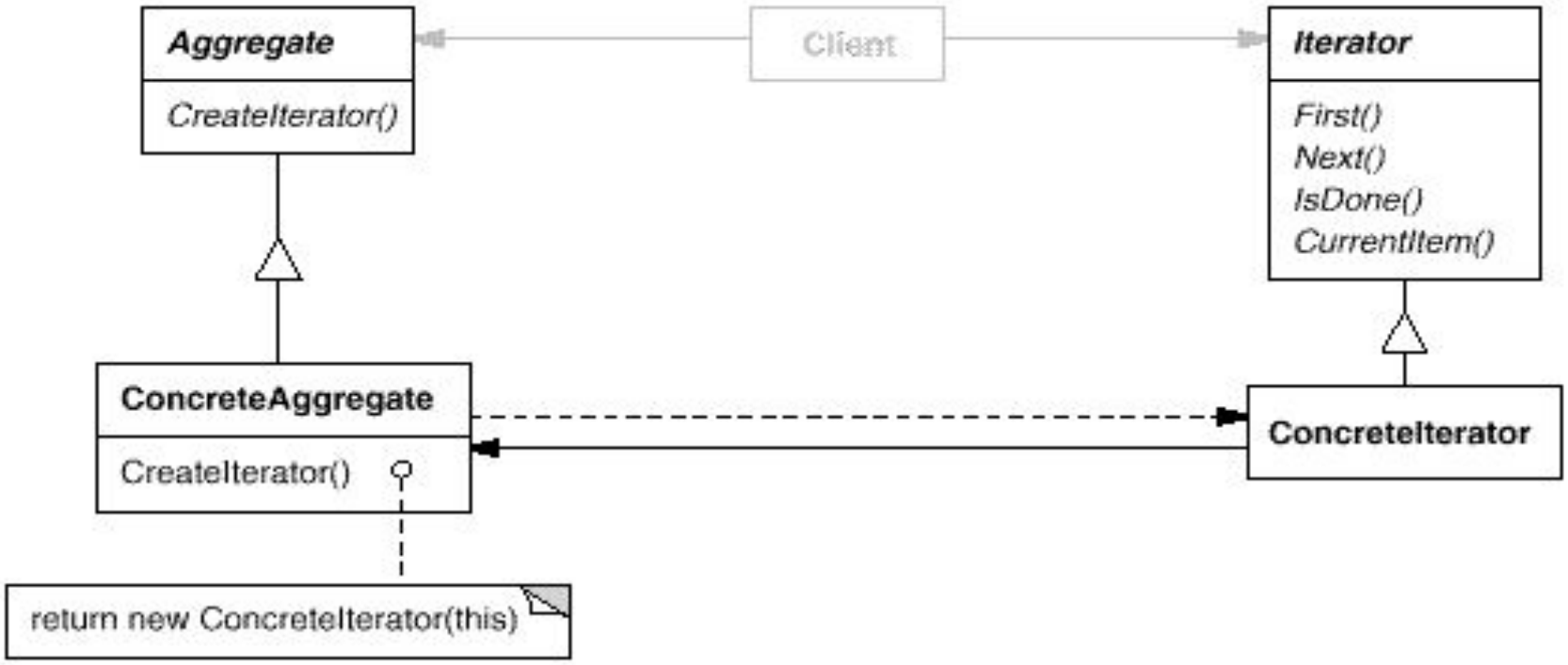
Iterator

Применимость

Используйте итератор:

- Для доступа к содержимому агрегированных объектов без раскрытия их внутреннего представления
- Для поддержки нескольких активных обходов одного и того же агрегированного объекта
- Для предоставления единообразного интерфейса с целью обхода различных агрегированных структур (для поддержки полиморфной итерации)

Iterator
Структура



Iterator

- Особенности
  - Поддерживает различные виды обхода агрегата
  - Итераторы упрощают интерфейс класса-агрегата
  - Одновременно для данного агрегата может быть активно несколько обходов
- Реализация
  - Какой участник управляет итерацией?
    - Внутренний
    - Внешний
  - Насколько итератор устойчив?
  - Дополнительные операции итератора

Observer

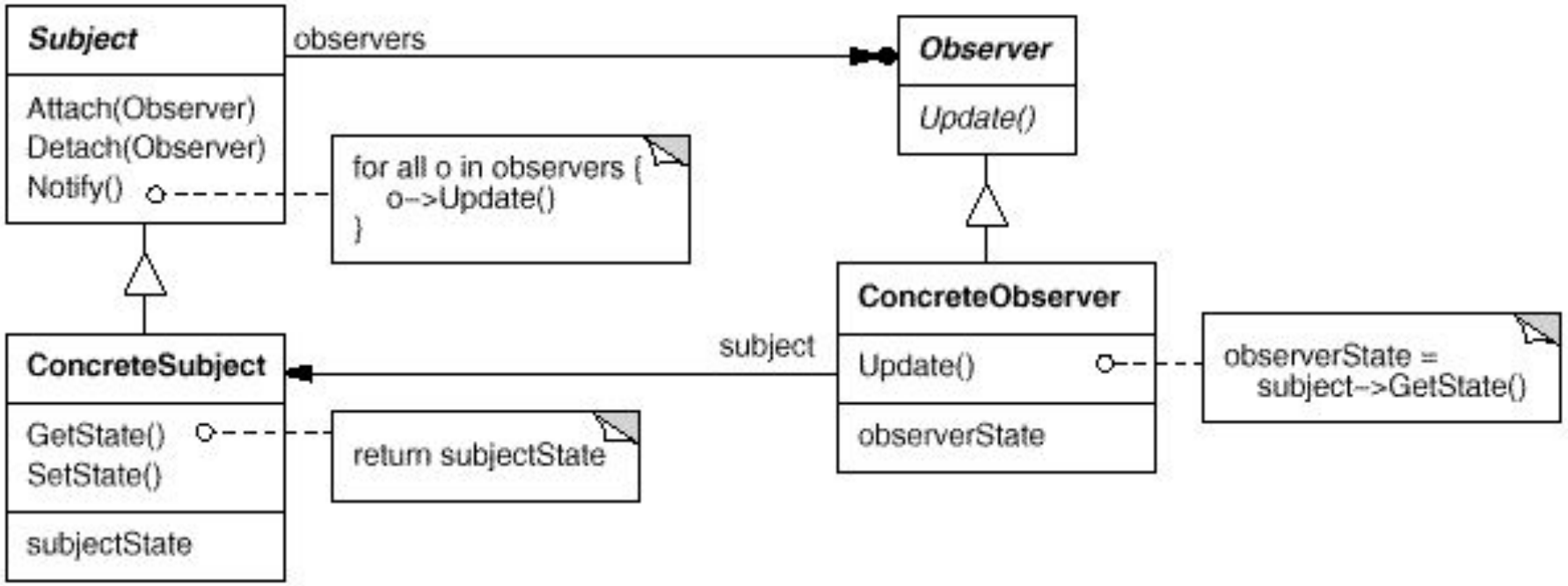
- Название и классификация
  Наблюдатель – паттерн поведения объектов
- Назначение
  Определяет зависимость типа “один ко многим” между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются
- Известен также под именем
  Dependents, Publish-Subscribe, Listener

Observer
Применимость

- Когда у абстракции есть два аспекта, один из которых зависит от другого. Инкапсуляции этих аспектов в разные объекты позволяют изменять и повторно использовать их независимо.
- Когда при модификации одного объекта требуется изменить другие и вы не знаете, сколько именно объектов нужно изменить.
- Когда один объект должен оповещать других, не делая предположений об уведомляемых объектах. Другими словами, вы не хотите, чтобы объекты были тесно связаны между собой.

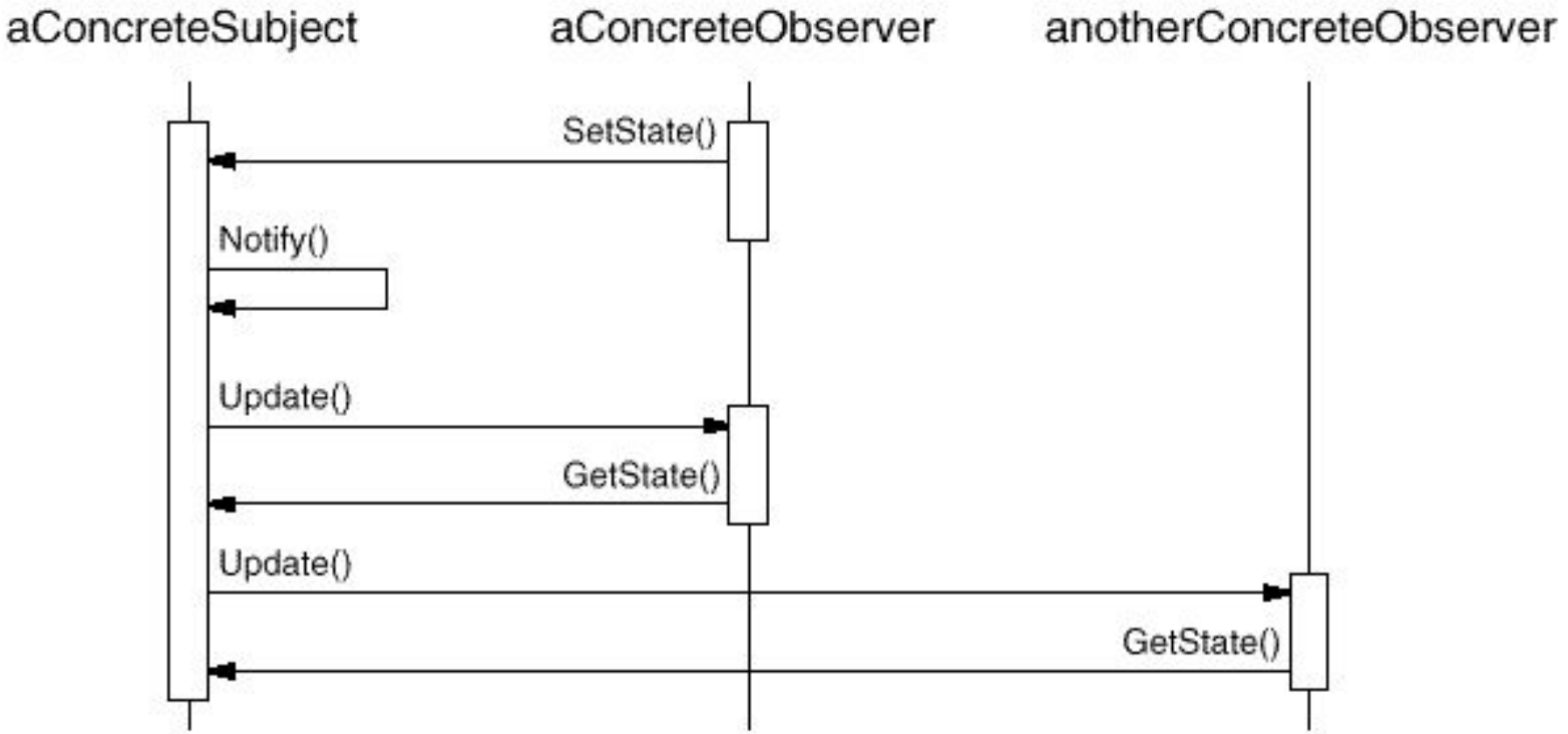
Observer

Структура



Observer

Отношения



Observer
Результаты

- Абстрактная связанность субъекта и наблюдателя
- Поддержка широковещательных коммуникаций
- Неожиданные обновления
- Простой протокол обновления не содержит никаких сведений о том, что именно изменилось в субъекте

Observer
Реализация

- Отображение субъектов на наблюдателей
- Наблюдение более чем за одним субъектом
- Инициатор обновления
- Модели вытягивания и проталкивания
- Явное специфицирование представляющих интерес модификаций