Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Объектно-ориентированное программирование

Введение в паттерны проектирования

Занятие 13

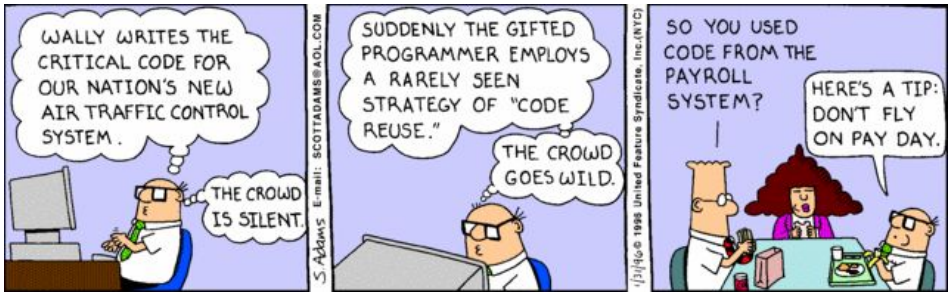
Самара
2015

План лекции

- Повторное использование кода
- Паттерны проектирования
- Порождающие паттерны
  - Singleton
  - Factory Method
- Структурные паттерны
  - Adapter
  - Decorator
  - Proxy
- Паттерны поведения
  - Iterator
  - Observer

Повторное использование кода

- Использование кода существующих приложений для создания новых приложений



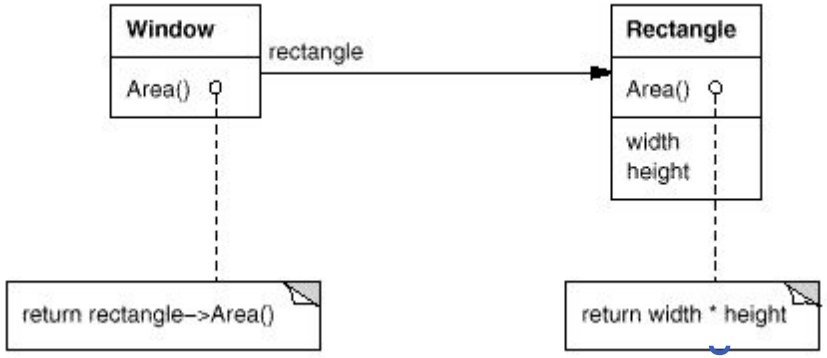
- Принципы создания хорошего повторно используемого кода
  - Модульность (modularity)
  - Слабая связность (low coupling)
  - Высокая сфокусированность (high cohesion)
  - Сокрытие информации (information hiding)
  - Разделение ответственности (separation of concerns)
- Примеры повторного использования
  - «Копипаста» (copy-and-paste)
  - Библиотеки (software libraries)
  - Паттерны проектирования (design patterns)
  - Фреймворки (software frameworks)

Механизмы повторного использования в ООП

- Наследование
  - Повторное использование родительского типа и/или реализации
  - Определяется статически
  - Нарушает инкапсуляцию
- Композиция
  - Повторное использование кода используемых классов
  - Определяется динамически
  - Решение представляет собой совокупность взаимодействующих объектов и может быть изменено путём замены объекта
- Шаблоны
  - Повторное использование реализации с частичным нарушением типа
  - Определяется статически
  - Типы и объекты требуют спецификации типов-параметров

Делегирование

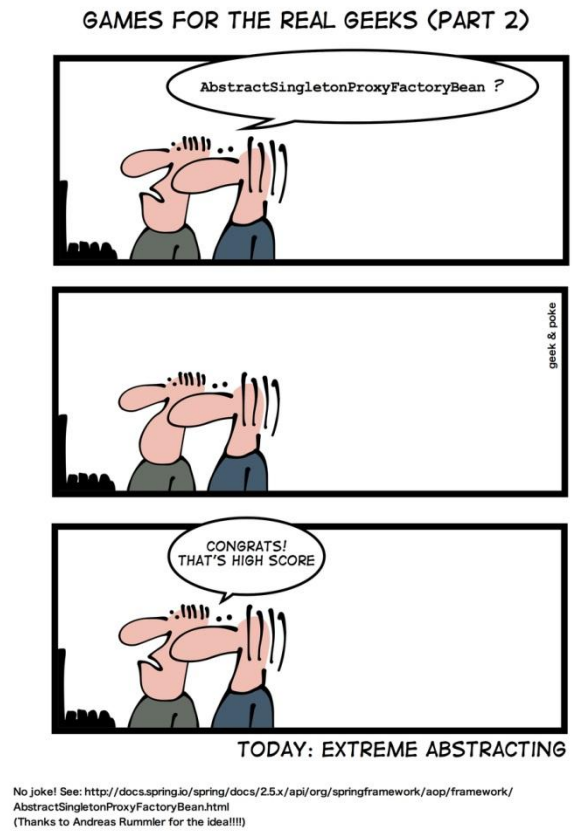
- Один объект в части своей реализации полагается на другой объект
  - Средство обеспечения композиции
  - Программа становится тяжелее для понимания



- При написании кода предпочитайте композицию наследованию класса

Рекомендации по написанию повторно-используемого кода

- Программируйте в соответствии с интерфейсом, а не реализацией
- Не объявляйте переменные как экземпляры конкретных классов. Вместо этого придерживайтесь интерфейса, определенного абстрактным типом
- Создавая в системе объекты конкретных классов, используйте порождающие паттерны проектирования

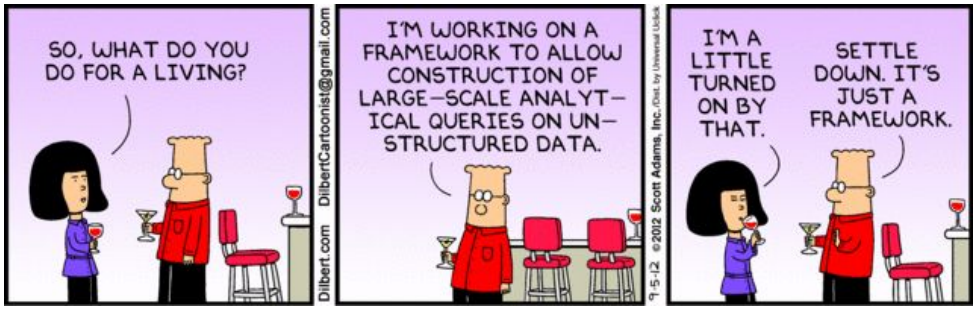


Причины перепроектирования

- При создании объекта явно указывается класс
- Зависимость от конкретных операций
- Зависимость от аппаратной и программной платформ
- Зависимость от представления или реализации объекта
- Зависимость от алгоритмов
- Сильная связанность
- Расширение функциональности за счет порождения подклассов

Фреймворки

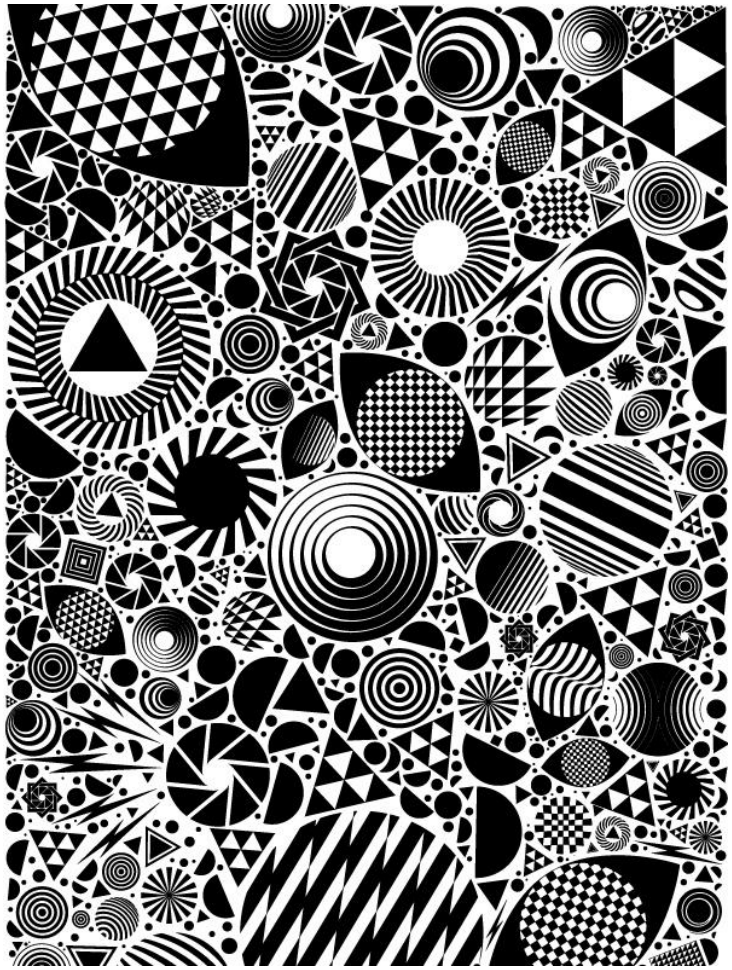
- **Фреймворк (Framework, каркас)** – это набор взаимодействующих классов, составляющих повторно используемое проектное решение для конкретного класса программ
- Диктует определенную структуру приложения или модуля
- Определяет общую структуру, ее разделение на классы и объекты, основные функции тех и других, методы взаимодействия потоков и классов, потоки управления



Паттерны проектирования

- **Паттерн проектирования (Design Pattern, образец проектирования, шаблон проектирования)** – описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте

- Паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого проектного решения



Паттерны в ООП

- Результат проектирования на уровне ООП – распределение ответственностей и активностей по классам
- Паттерн – именованная конфигурация распределения ответственности по классам

Фреймворки vs Паттерны

- Паттерны проектирования более абстрактны, чем фреймворки
- Как архитектурные элементы паттерны проектирования мельче, чем фреймворки
- Паттерны проектирования менее специализированны, чем фреймворки

Описания паттернов

- **GoF**
  Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software
- **POSA1**
  Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad (1996). Pattern-Oriented Software Architecture, Volume 1: A System of Patterns
- **POSA2**
  Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann (2000). Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects
- **PoEAA**
  Martin Fowler (2002). Patterns of Enterprise Application Architecture
- …

Порождающие паттерны

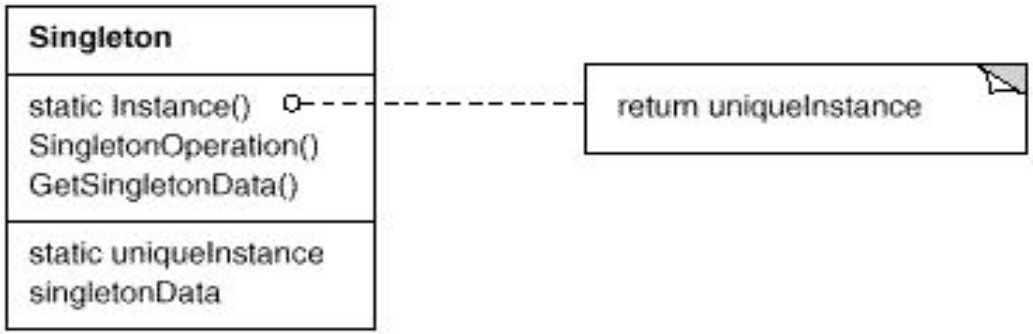
- Abstract Factory – Абстрактная фабрика
- Builder – Строитель
- Factory Method – Фабричный метод
- Prototype – Прототип
- Singleton – Одиночка

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

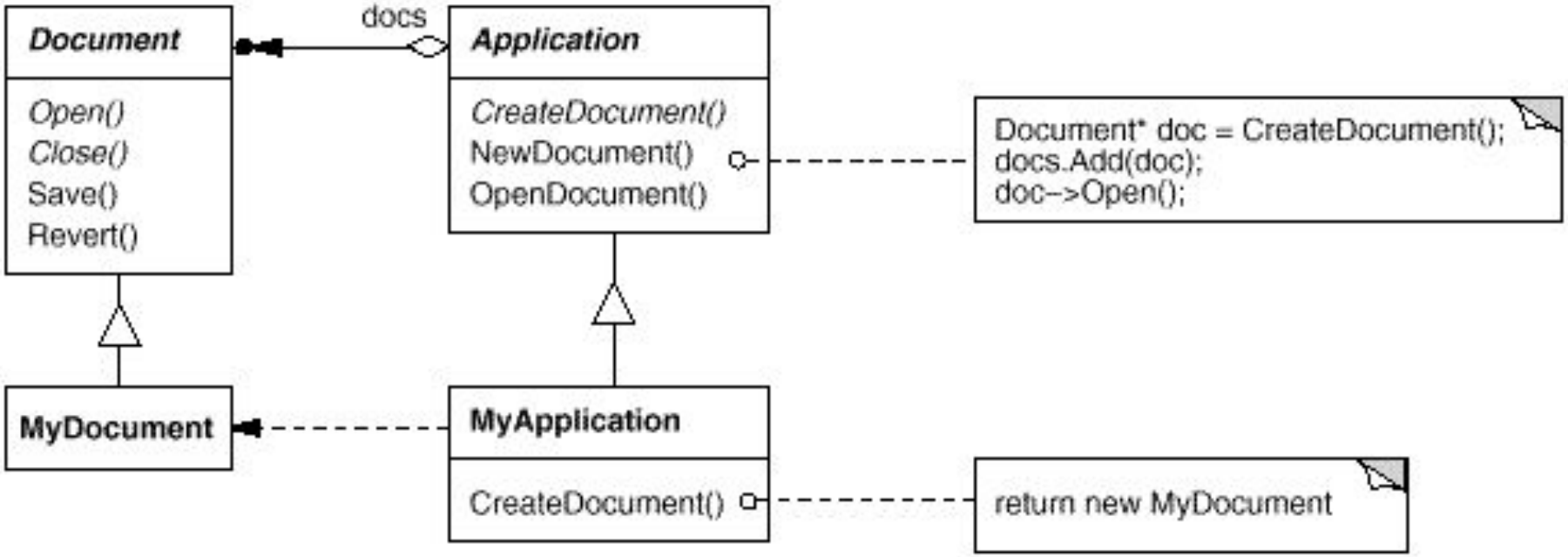


- Участники
  - Singleton – одиночка, Определяет операцию `Instance` , которая позволяет клиентам получать доступ к единственному экземпляру

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

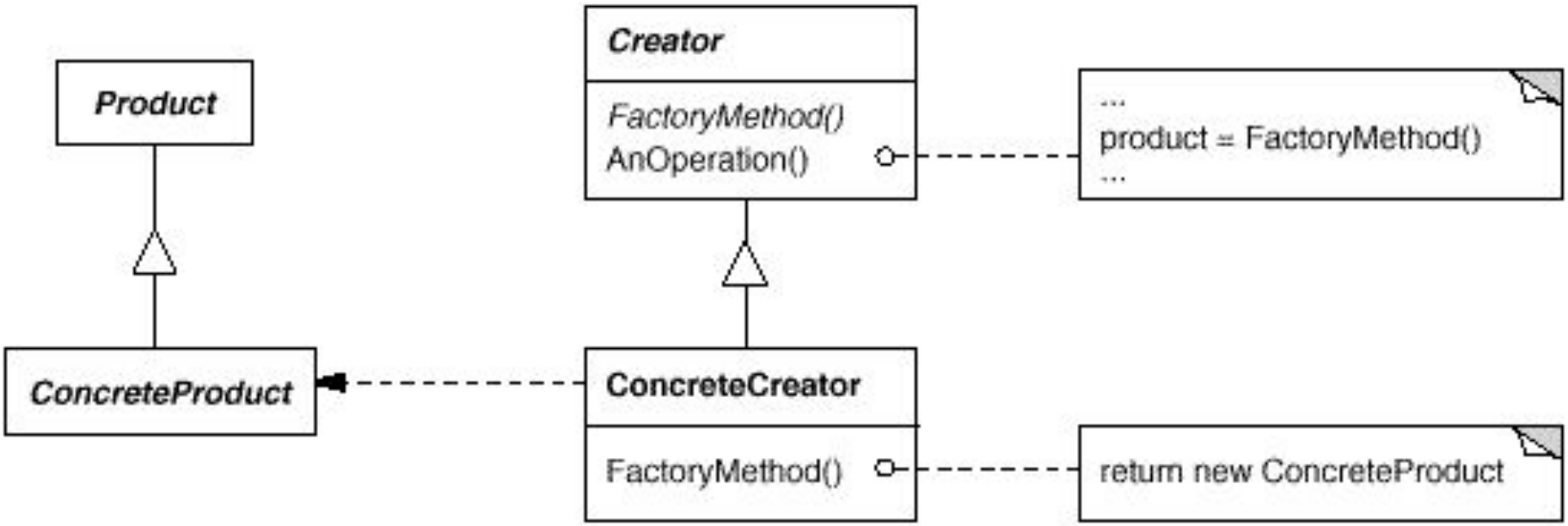


Factory Method
Применимость

Используйте паттерн фабричный метод, когда:

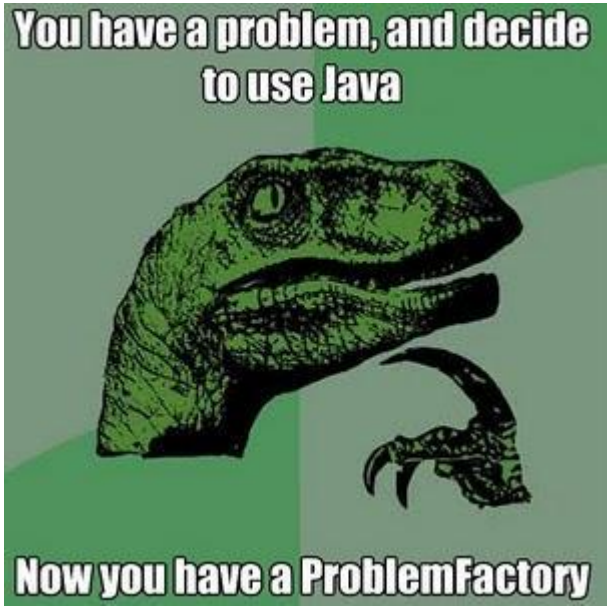
- классу заранее неизвестно, объекты каких классов ему нужно создавать
- класс спроектирован так, чтобы объекты, которые он создает, специфицировались подклассами
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и вы планируете локализовать знание о том, какой класс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

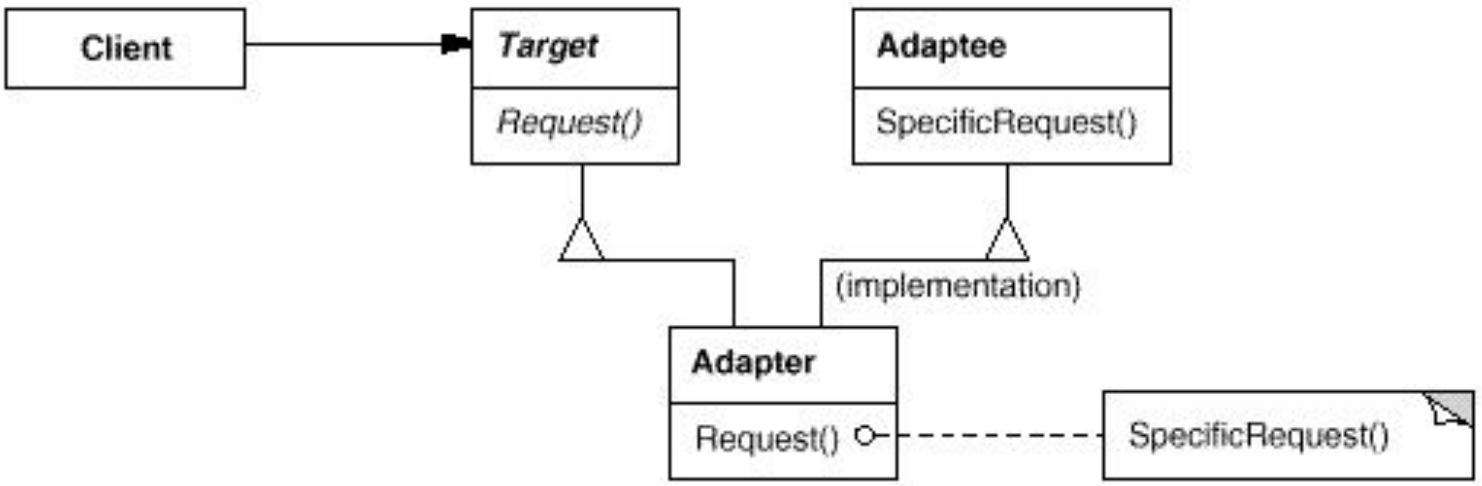
Adapter

- 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

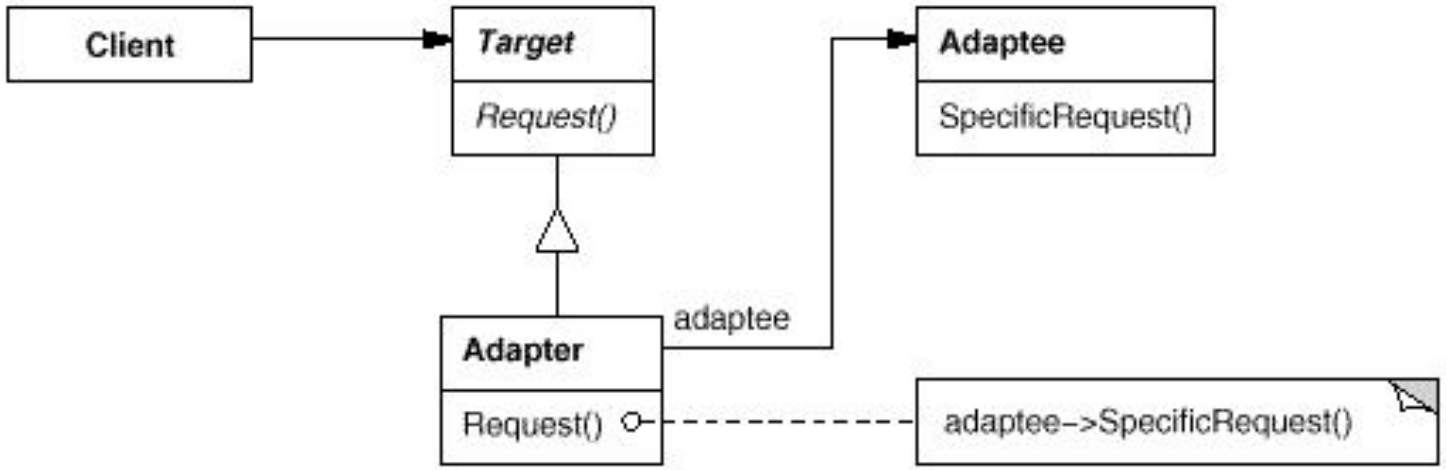
- Применяйте адаптер классов, когда:
  - хотите использовать существующий класс, но его интерфейс не соответствует вашим потребностям
  - собираетесь создать повторно используемый класс, который должен взаимодействовать с заранее неизвестными или не связанными с ним классами, имеющими несовместимые интерфейсы
- 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)



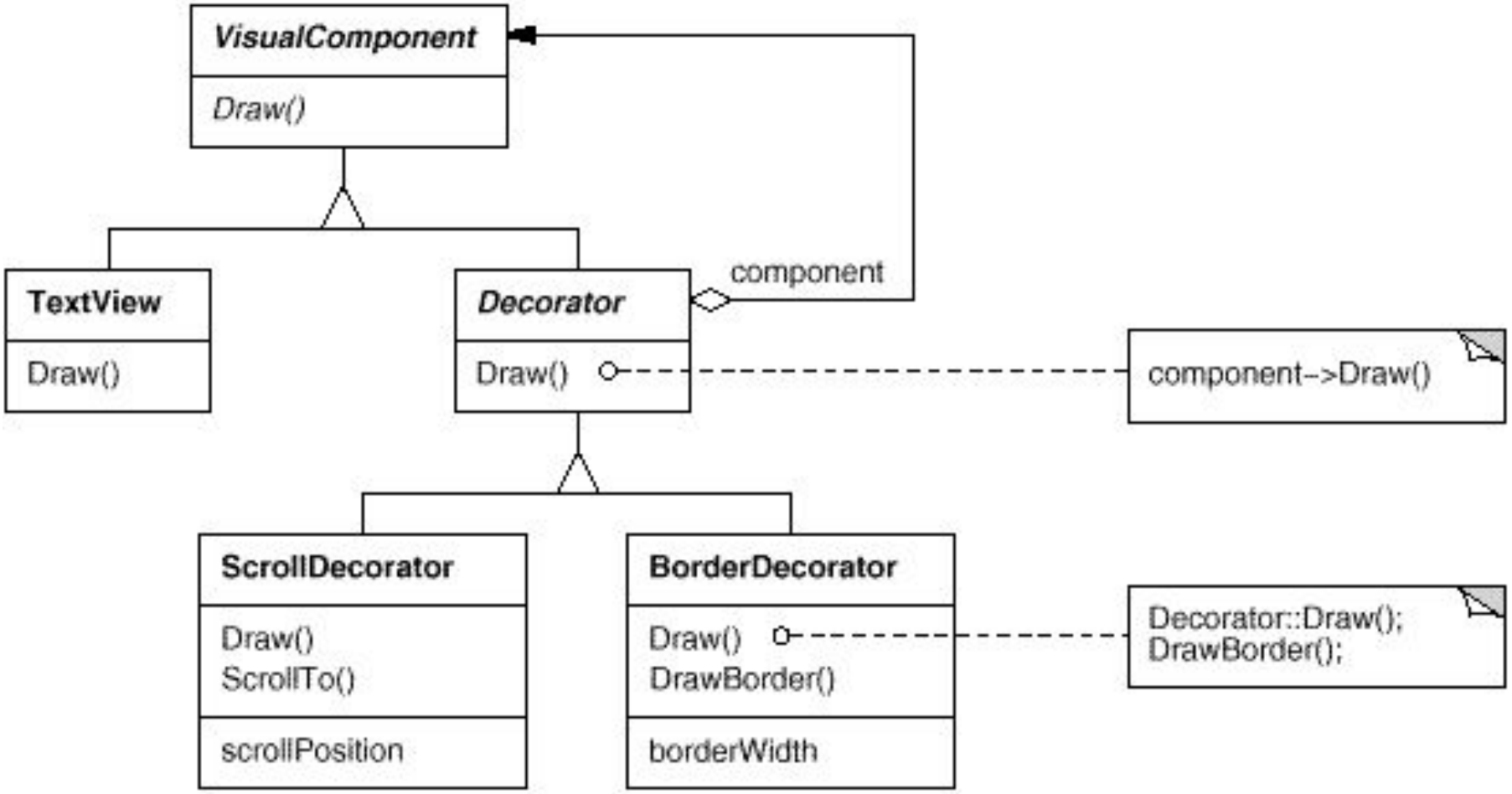
- Один адаптер может работать со множеством адаптируемых объектов, включая объекты подклассов
- Затруднено замещение операций адаптируемого класса

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



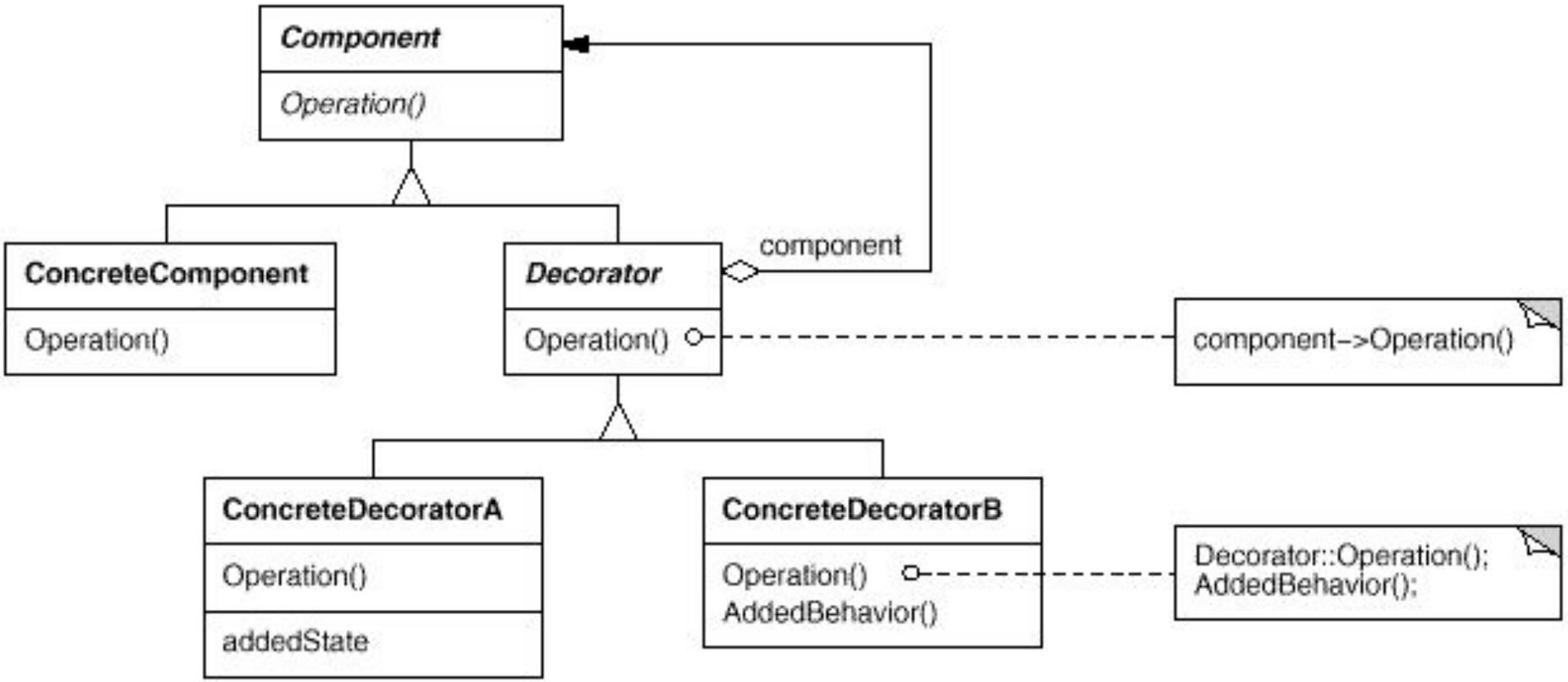
Decorator
Применимость

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

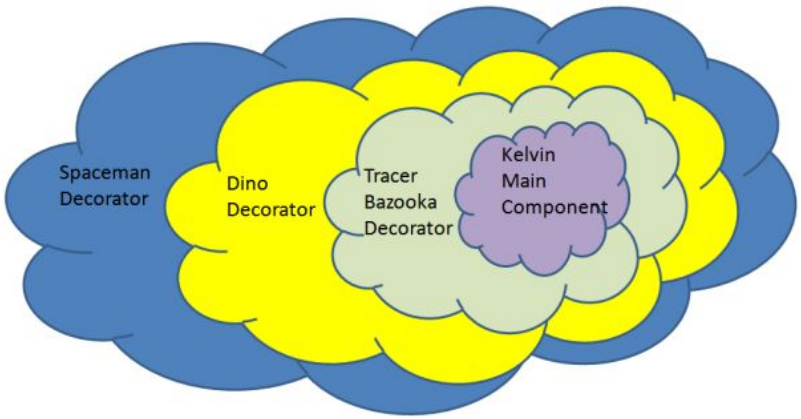


Decorator
Особенности

- 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

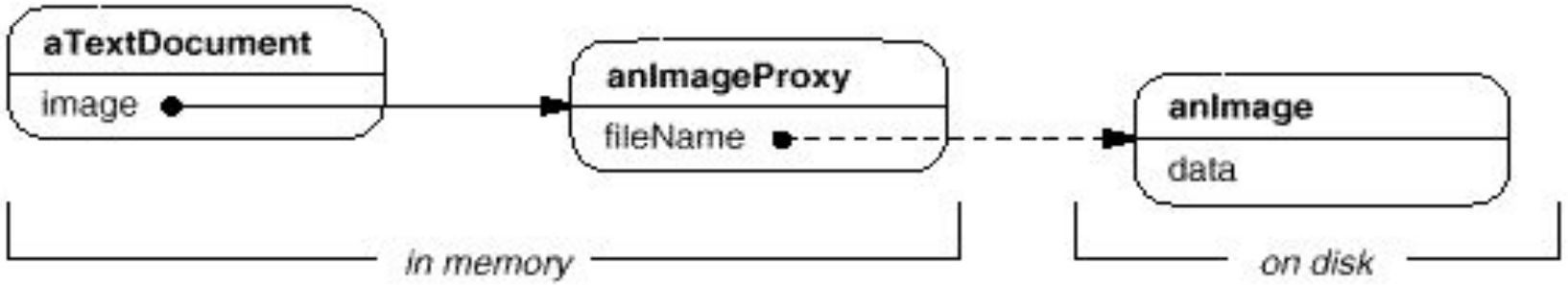


Proxy

- 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

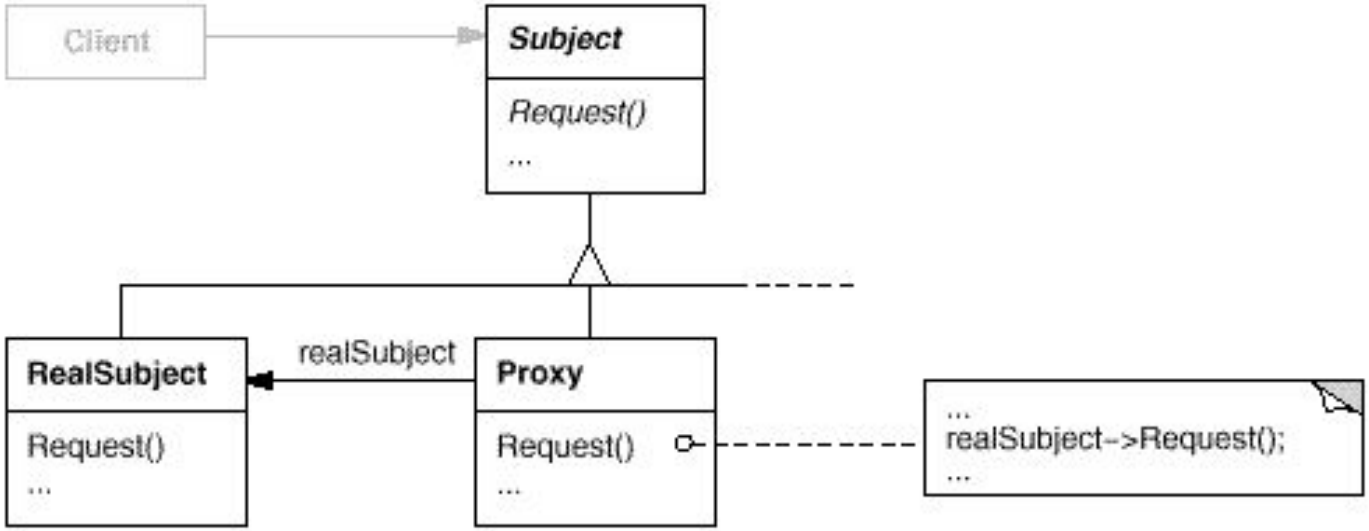
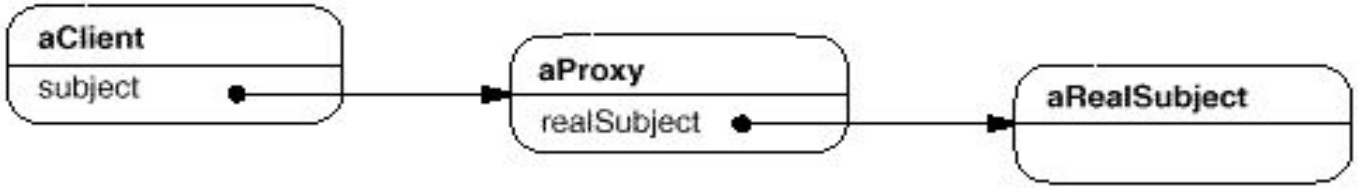


Диаграмма объектов



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)

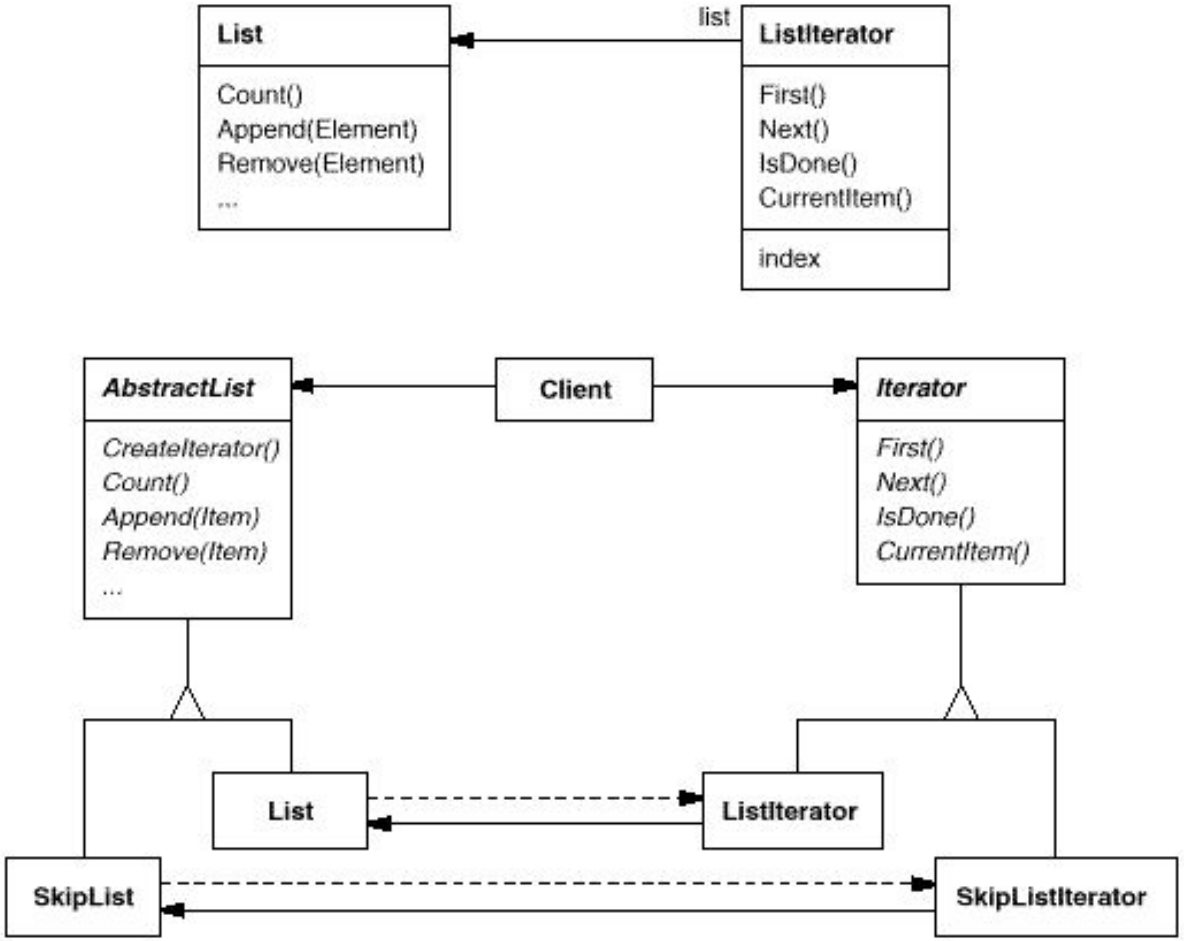
Паттерны поведения

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

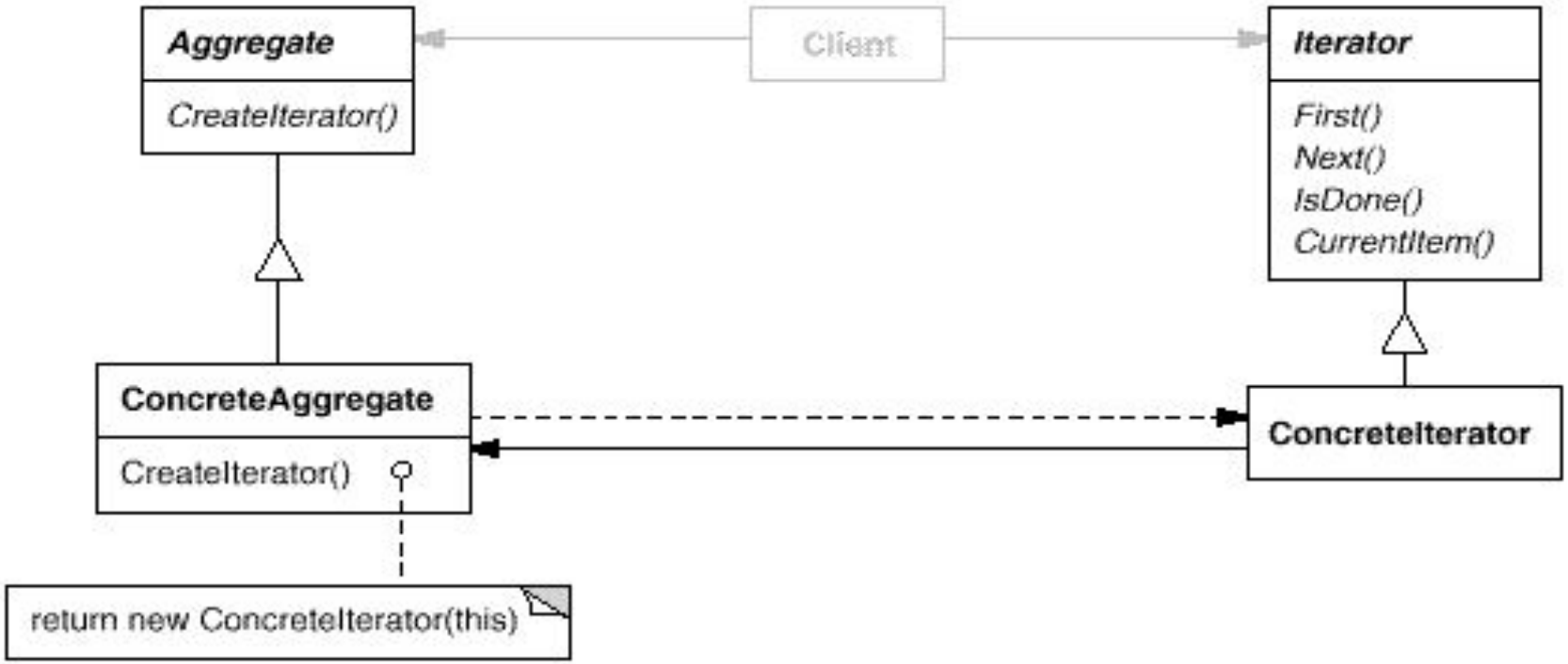


Iterator
Применимость

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

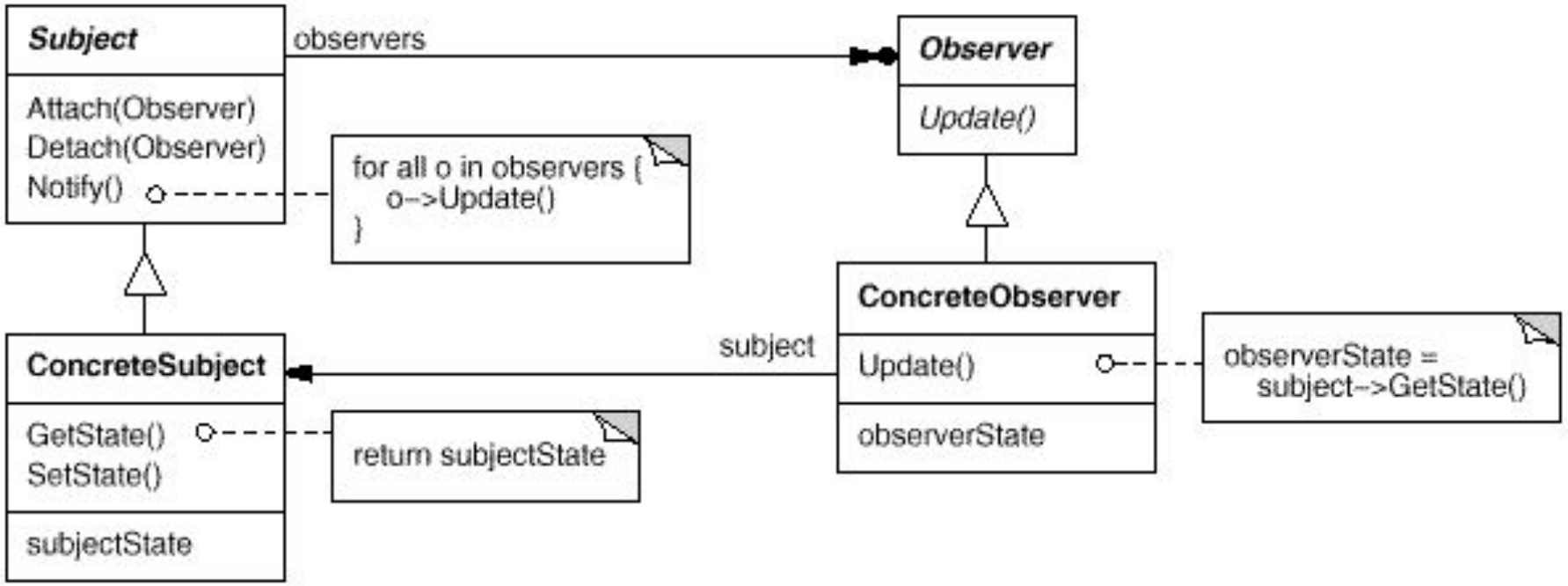
- Название и классификация
  Наблюдатель – паттерн поведения объектов
- Назначение
  Определяет зависимость типа “один ко многим” между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются
- Известен также под именем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

Observer

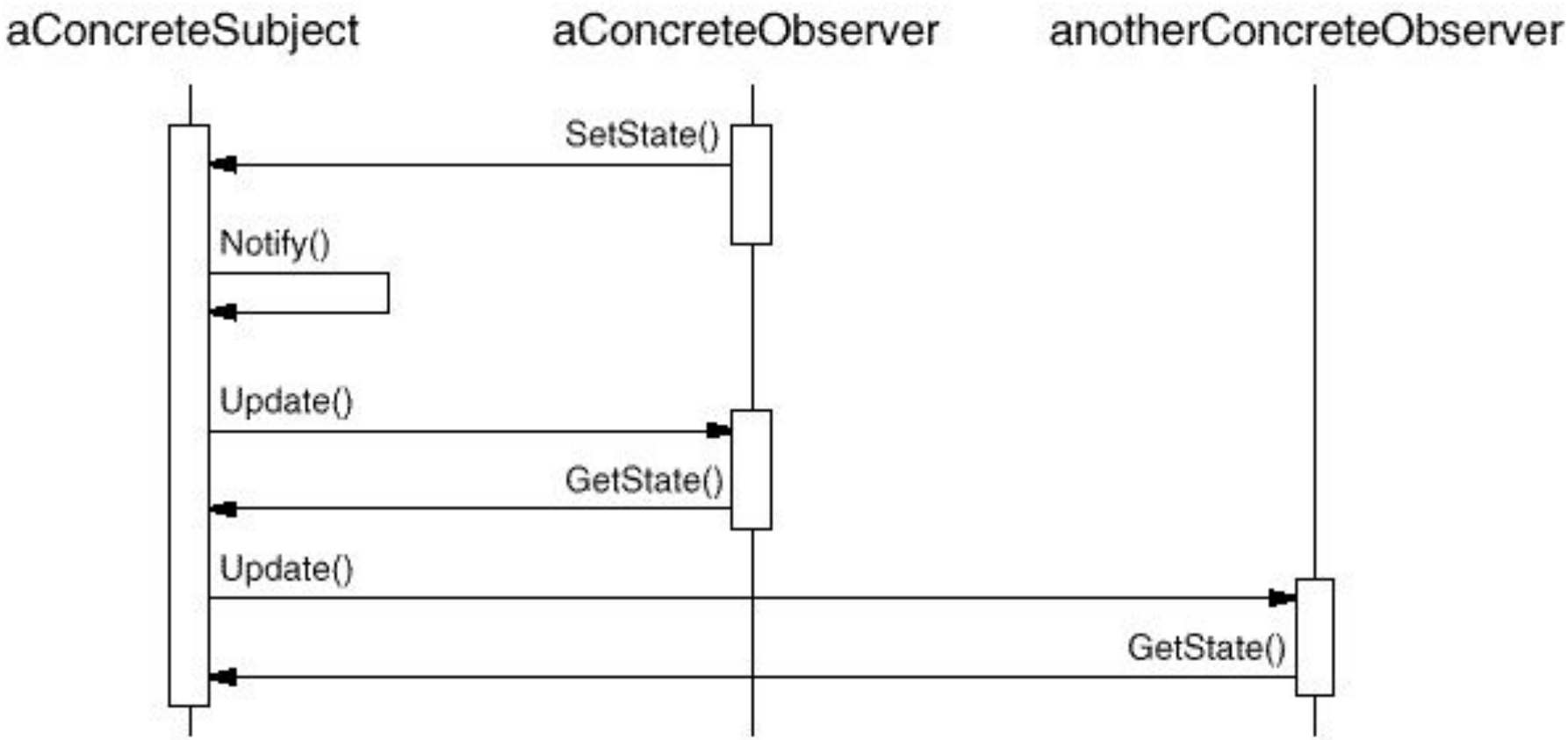
Применимость

- 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

Спасибо за внимание!

Дополнительные источники

- Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб. : Питер, 2007. – 366 с.
- Фримен, Э. Паттерны проектирования [Текст] / Эрик Фримен, Элизабет Робсон, Берт Бейтс, Кэти Сьерра. – СПб. : Питер, 2011. – 656 с.
- Обзор паттернов проектирования [Электронный ресурс]. – Режим доступа: http://citforum.ru/SE/project/pattern/index.shtml, дата доступа: 21.10.2011.
- Объектно-ориентированное проектирование, паттерны проектирования (шаблоны) [Электронный ресурс]. – Режим доступа: http://www.javenue.info/themes/ood/, дата доступа: 21.10.2011.
- Обзор паттернов проектирования [Электронный ресурс]. – Режим доступа: http://www.firststeps.ru/theory/patt/pattern1.html, дата доступа: 21.10.2011.