СТРУКТУРНЫЕ ПАТТЕРНЫ

- Паттерны проектирования

СТРУКТУРНЫЕ ПАТТЕРНЫ

ОПРЕДЕЛЕНИЕ

- Структурные паттерны рассматривают вопросы о компоновке системы на основе классов и объектов. При этом могут использоваться следующие механизмы:

- Наследование, когда базовый класс определяет интерфейс, а подклассы - реализацию. Структуры на основе наследования получаются статичными.

- Композиция, когда структуры строятся путем объединения объектов некоторых классов. Композиция позволяет получать структуры, которые можно изменять во время выполнения.

ADAPTER (АДАПТЕР)

НАЗНАЧЕНИЕ

Позволяет объектам с несовместимыми интерфейсами работать вместе

- Часто в новом программном проекте не удается повторно использовать уже существующий код. Например, имеющиеся классы могут обладать нужной функциональностью, но иметь при этом несовместимые интерфейсы. В таких случаях следует использовать паттерн Adapter (адаптер).

- Паттерн Adapter, представляющий собой программную обертку над существующими классами, преобразует их интерфейсы к виду, пригодному для последующего использования.

ADAPTER

ОПИСАНИЕ

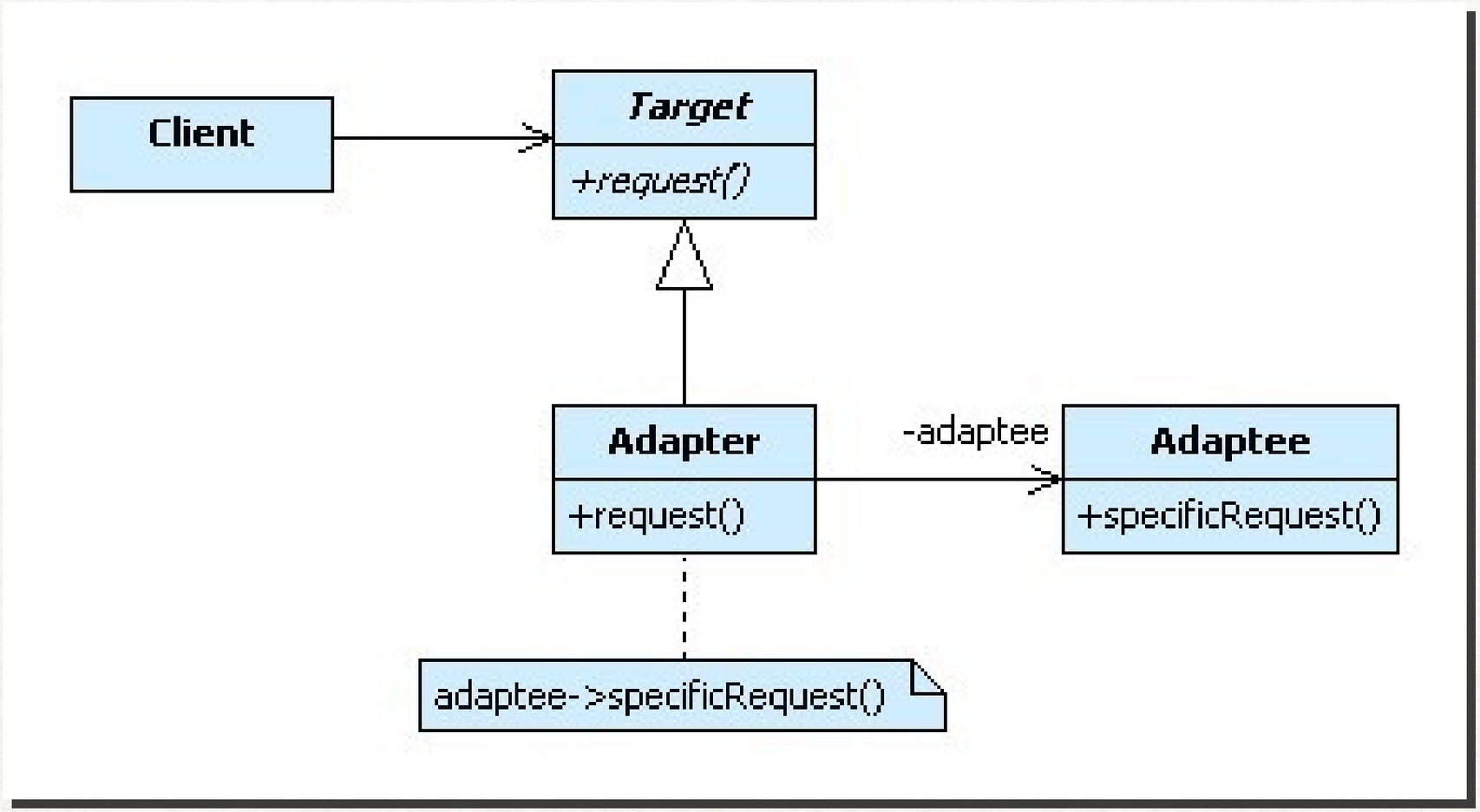
- Пусть класс, интерфейс которого нужно адаптировать к нужному виду, имеет имя Adaptee. Для решения задачи преобразования его интерфейса паттерн Adapter вводит следующую иерархию классов:

- Виртуальный базовый класс Target. Здесь объявляется пользовательский интерфейс подходящего вида. Только этот интерфейс доступен для пользователя.

- Производный класс Adapter, реализующий интерфейс Target. В этом классе также имеется указатель или ссылка на экземпляр Adaptee. Паттерн Adapter использует этот указатель для перенаправления клиентских вызовов в Adaptee. Так как интерфейсы Adaptee и Target несовместимы между собой, то эти вызовы обычно требуют преобразования.

ADAPTER

СТРУКТУРА



ADAPTER

РЕЗУЛЬТАТЫ

- Достоинства паттерна Adapter
  - Паттерн Adapter позволяет повторно использовать уже имеющийся код, адаптируя его несовместимый интерфейс к виду, пригодному для использования.
- Недостатки паттерна Adapter
  - Задача преобразования интерфейсов может оказаться непростой в случае, если клиентские вызовы и (или) передаваемые параметры не имеют функционального соответствия в адаптируемом объекте.

BRIDGE (МОСТ)

НАЗНАЧЕНИЕ

Разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга

- В системе могут существовать классы, отношения между которыми строятся в соответствии со следующей объектно-ориентированной иерархией: абстрактный базовый класс объявляет интерфейс, а конкретные подклассы реализуют его нужным образом. Такой подход является стандартным в ООП, однако, ему свойственны следующие недостатки:

- Система, построенная на основе наследования, является статичной. Реализация жестко привязана к интерфейсу. Изменить реализацию объекта некоторого типа в процессе выполнения программы уже невозможно.

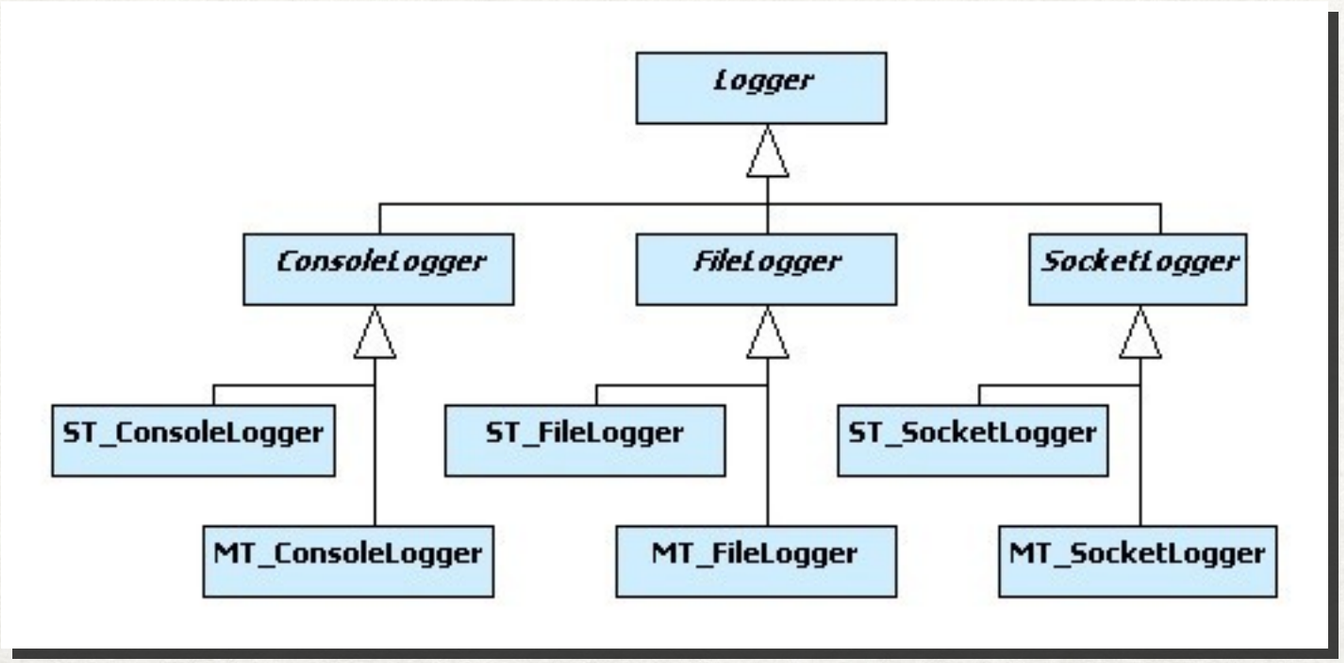
- Система становится трудно поддерживаемой, если число родственных производных классов становится большим.

BRIDGE

ПРИМЕР

- Логгер это система протоколирования сообщений, позволяющая фиксировать ошибки, отладочную и другую информацию в процессе выполнения программы. Разрабатываемый нами логгер может использоваться в одном из трех режимов: выводить сообщения на экран, в файл или отсылать их на удаленный компьютер. Кроме того, необходимо обеспечить возможность его применения в одно- и многопоточной средах.

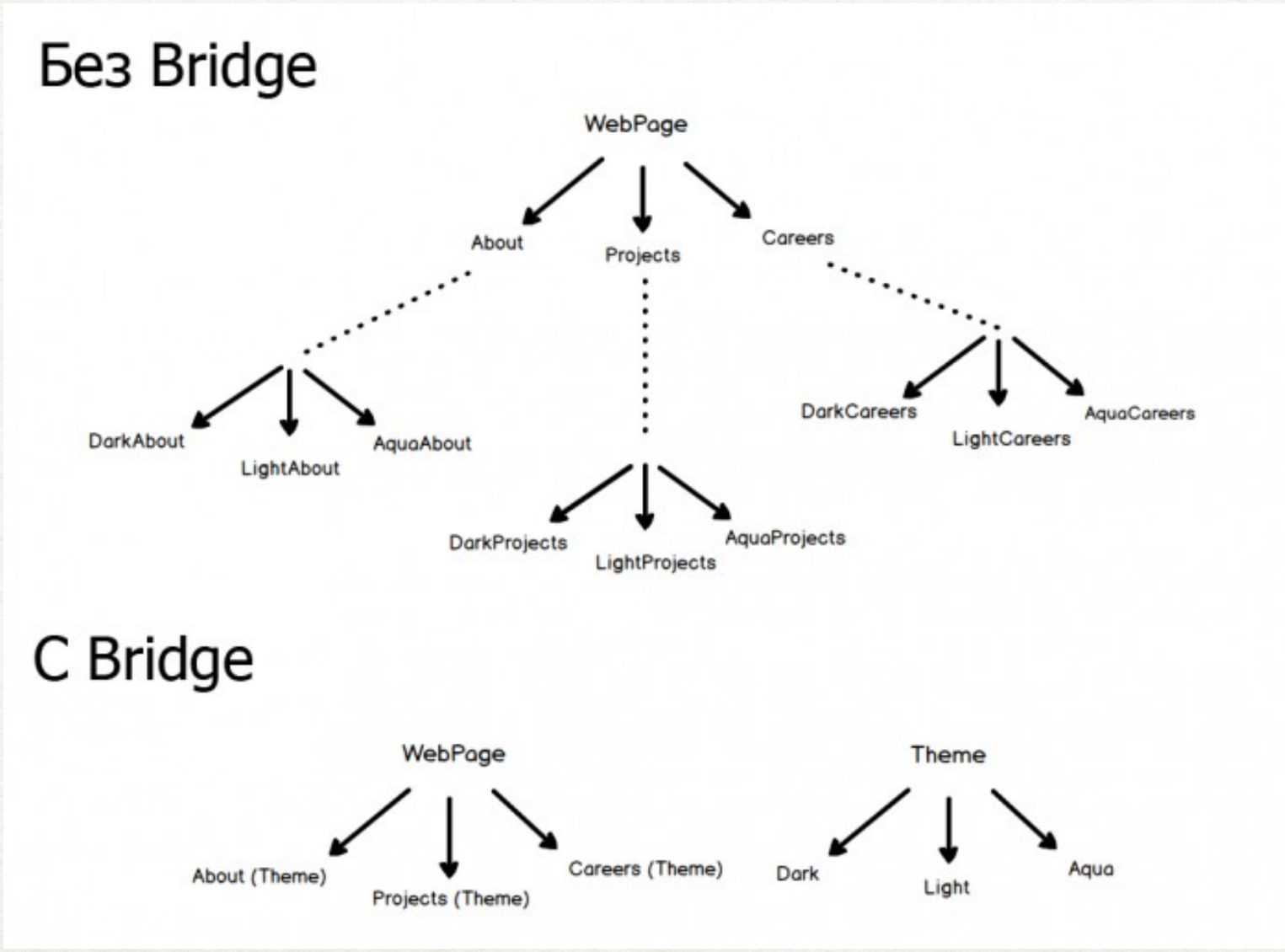
- Стандартный подход на основе полиморфизма использует следующую иерархию классов.



BRIDGE

ПРИМЕР

Есть сайт с разными страницами, и пользователь должен иметь возможность изменять тему. Что делать? Можно создать копии страниц для каждой из тем или же просто загрузить темы отдельно. Структурные паттерны Bridge позволят реализовать второй вариант.



BRIDGE

ОПИСАНИЕ

- Паттерн Bridge разделяет абстракцию и реализацию на две отдельные иерархии классов так, что их можно изменять независимо друг от друга.

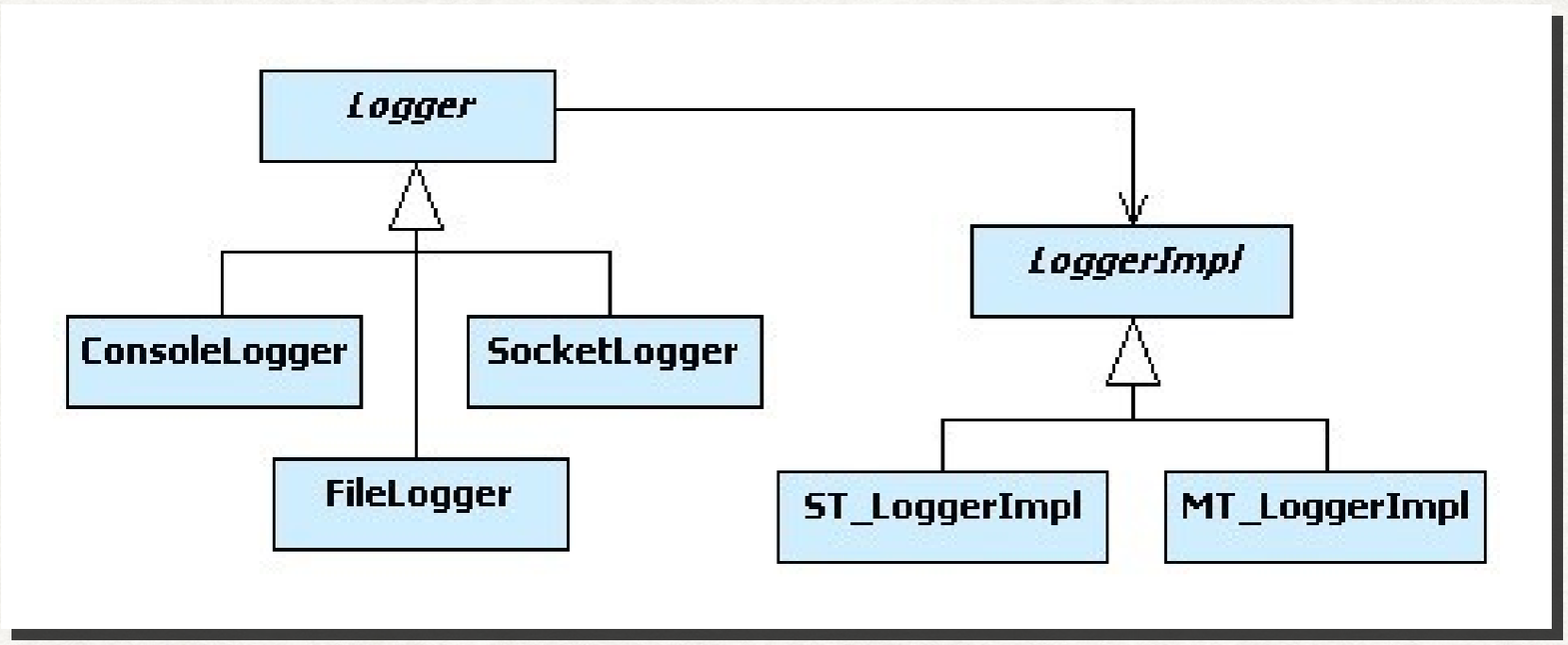
- Первая иерархия определяет интерфейс абстракции, доступный пользователю. Для случая проектируемого нами логгера абстрактный базовый класс Logger мог бы объявить интерфейс метода log() для вывода сообщений. Класс Logger также содержит указатель на реализацию pimpl, который инициализируется должным образом при создании логгера конкретного типа. Этот указатель используется для перенаправления пользовательских запросов в реализацию. Заметим, в общем случае подклассы ConsoleLogger, FileLogger и SocketLogger могут расширять интерфейс класса Logger.

- Все детали реализации, связанные с особенностями среды скрываются во второй иерархии. Базовый класс LoggerImpl объявляет интерфейс операций, предназначенных для отправки сообщений на экран, файл и удаленный компьютер, а подклассы ST_LoggerImpl и MT_LoggerImpl его реализуют для однопоточной и многопоточной среды соответственно. В общем случае, интерфейс LoggerImpl необязательно должен в точности соответствовать интерфейсу абстракции. Часто он выглядит как набор низкоуровневых примитивов.

BRIDGE

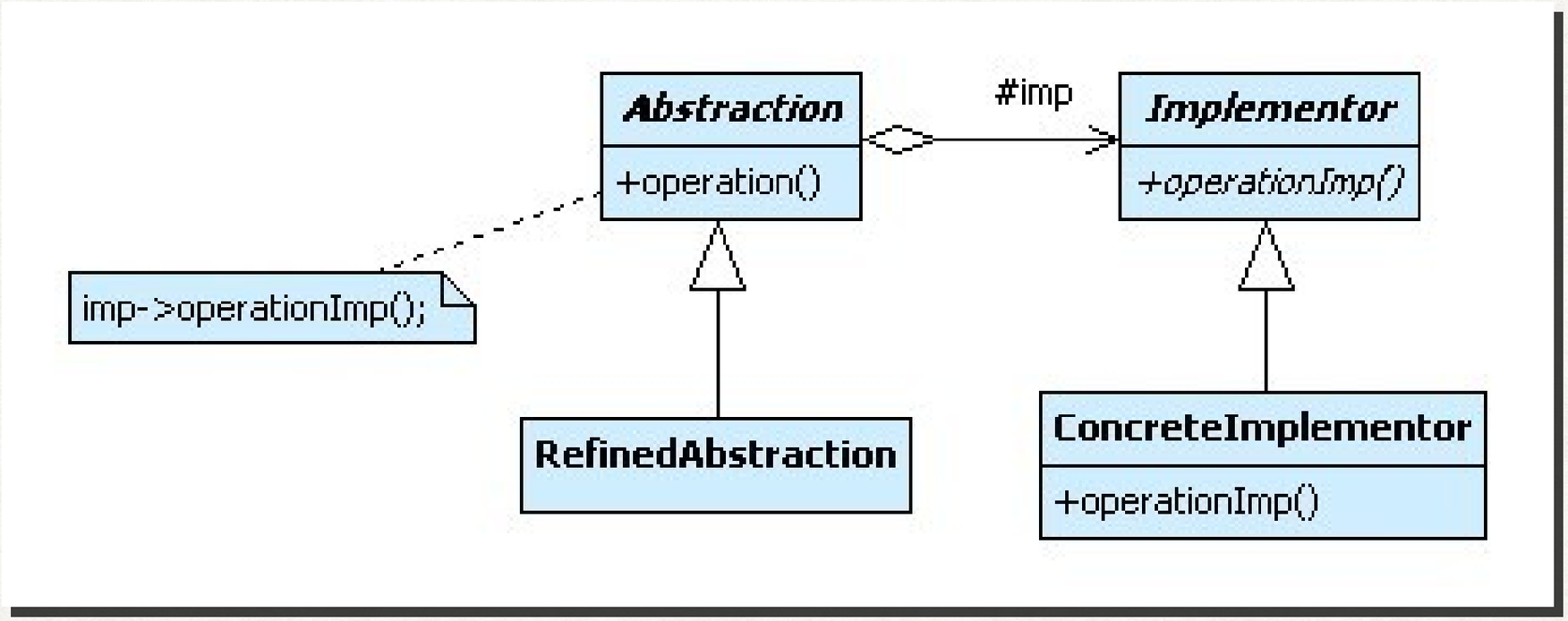
ОПИСАНИЕ

- Паттерн Bridge позволяет легко изменить реализацию во время выполнения программы. Для этого достаточно перенастроить указатель pimpl на объект-реализацию нужного типа. Применение паттерна Bridge также позволяет сократить общее число подклассов в системе, что делает ее более простой в поддержке.



BRIDGE

СТРУКТУРА



BRIDGE

РЕЗУЛЬТАТЫ

- Достоинства паттерна Bridge
- Проще расширять систему новыми типами за счет сокращения общего числа родственных подклассов.
- Возможность динамического изменения реализации в процессе выполнения программы.
- Паттерн Bridge полностью скрывает реализацию от клиента. В случае модификации реализации пользовательский код не требует перекомпиляции.

COMPOSITE (КОМПОНОВЩИК)

НАЗНАЧЕНИЕ

Позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект

- Используйте паттерн Composite если:

- Необходимо объединять группы схожих объектов и управлять ими.

- Объекты могут быть как примитивными (элементарными), так и составными (сложными). Составной объект может включать в себя коллекции других объектов, образуя сложные древовидные структуры. Пример: директория файловой системы состоит из элементов, каждый их которых также может быть директорией.

- Код клиента работает с примитивными и составными объектами единообразно.

COMPOSITE

ОПИСАНИЕ

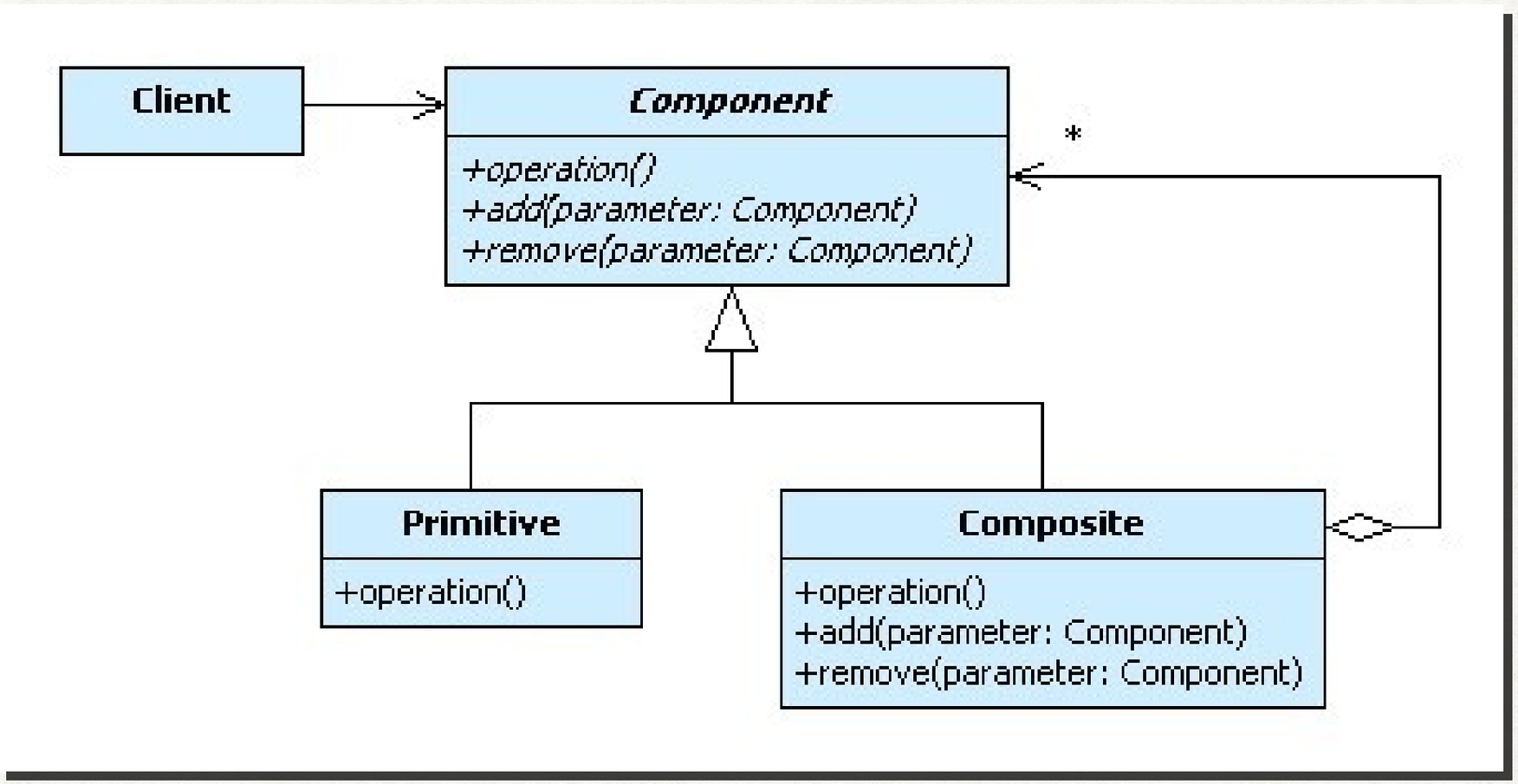
- Управление группами объектов может быть непростой задачей, особенно, если эти объекты содержат собственные объекты.

- Для военной стратегической игры ”Пунические войны ”, описывающей военное противостояние между Римом и Карфагеном, каждая боевая единица (всадник, лучник, пехотинец) имеет свою собственную разрушающую силу. Эти единицы могут объединяться в группы для образования более сложных военных подразделений, например, римские легионы, которые, в свою очередь, объединяясь, образуют целую армию. Как рассчитать боевую мощь таких иерархических соединений?

- Паттерн Composite предлагает следующее решение. Он вводит абстрактный базовый класс Component с поведением, общим для всех примитивных и составных объектов. Для случая стратегической игры - это метод getStrength() для подсчета разрушающей силы. Подклассы Primitive и Composite являются производными от класса Component. Составной объект Composite хранит компоненты-потомки абстрактного типа Component, каждый из которых может быть также Composite.

COMPOSITE

СТРУКТУРА



Для добавления или удаления объектов-потомков в составной объект Composite, класс Component определяет интерфейсы add() и remove()

COMPOSITE

РЕЗУЛЬТАТЫ

- Достоинства паттерна Composite
- В систему легко добавлять новые примитивные или составные объекты, так как паттерн Composite использует общий базовый класс Component.
- Код клиента имеет простую структуру – примитивные и составные объекты обрабатываются одинаковым образом.
- Паттерн Composite позволяет легко обойти все узлы древовидной структуры
- Недостатки паттерна Composite
- Неудобно осуществить запрет на добавление в составной объект Composite объектов определенных типов. Так, например, в состав римской армии не могут входить боевые слоны.

DECORATOR (ДЕКОРАТОР)

НАЗНАЧЕНИЕ

Позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки»

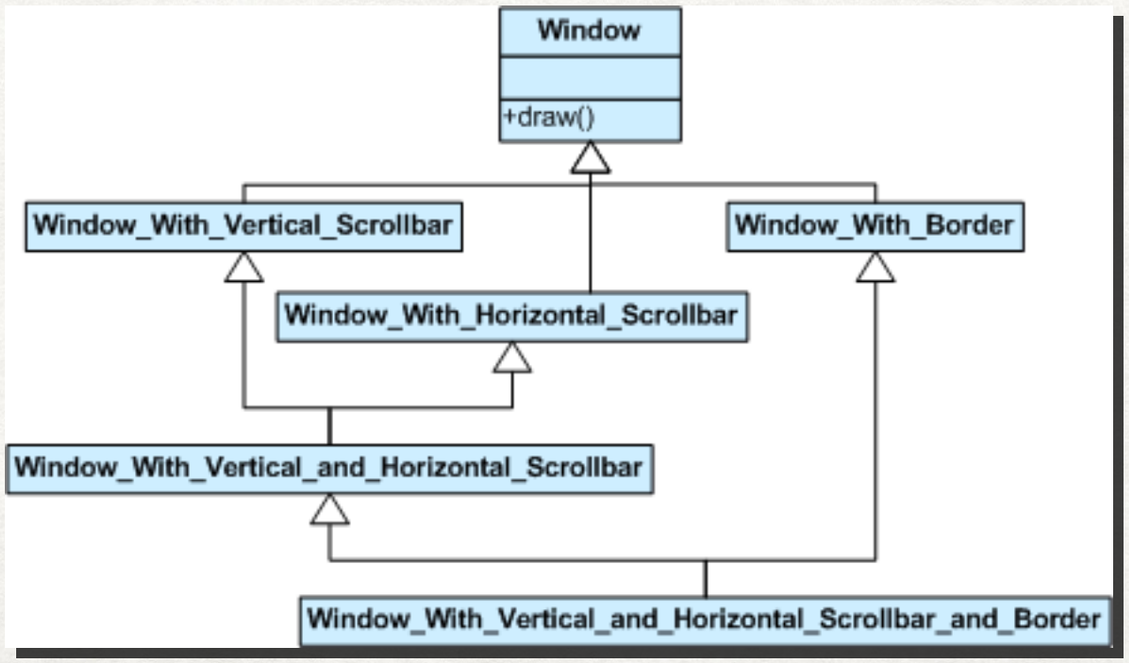
- Паттерн Decorator динамически добавляет новые обязанности объекту. Декораторы являются гибкой альтернативой порождению подклассов для расширения функциональности.
- Рекурсивно декорирует основной объект.
- Паттерн Decorator использует схему "обертываем подарок, кладем его в коробку, обертываем коробку".

DECORATOR

ОПИСАНИЕ

- Вы хотите добавить новые обязанности в поведении или состоянии отдельных объектов во время выполнения программы. Использование наследования не представляется возможным, поскольку это решение статическое и распространяется целиком на весь класс.

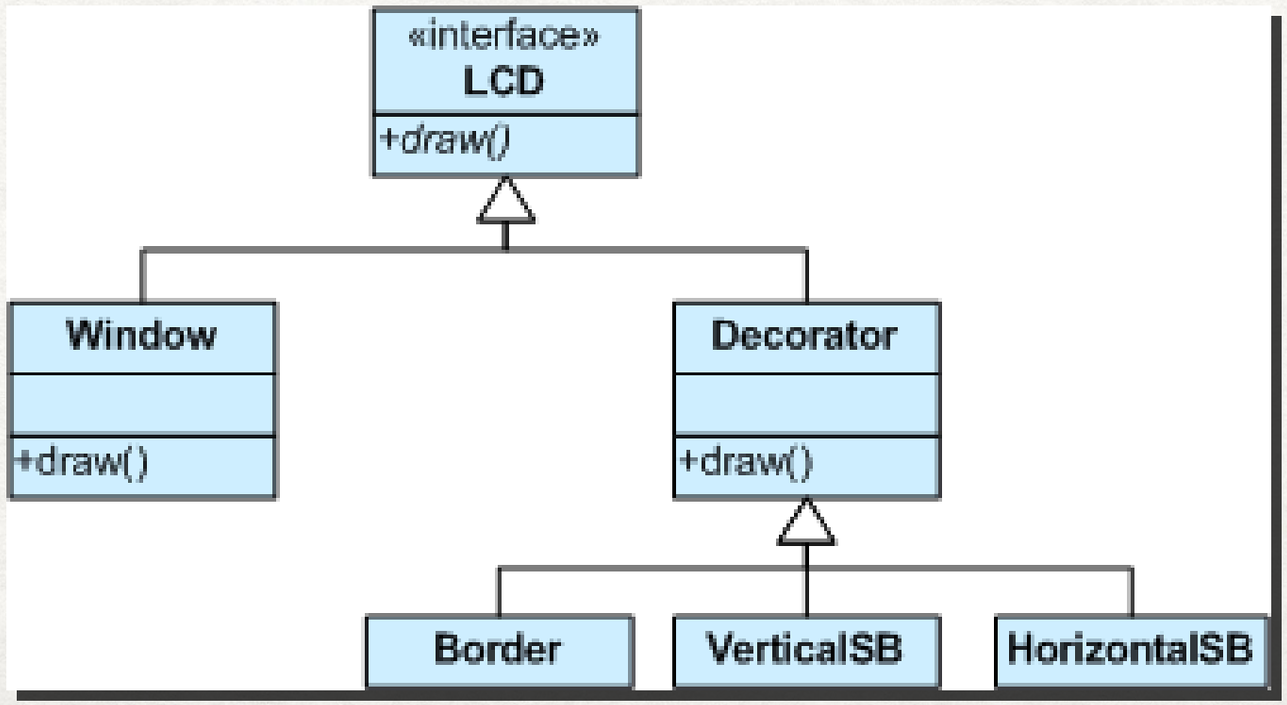
- Предположим, вы работаете над библиотекой для построения графических пользовательских интерфейсов и хотите иметь возможность добавлять в окно рамку и полосу прокрутки. Тогда вы могли бы определить иерархию наследования следующим образом…



DECORATOR

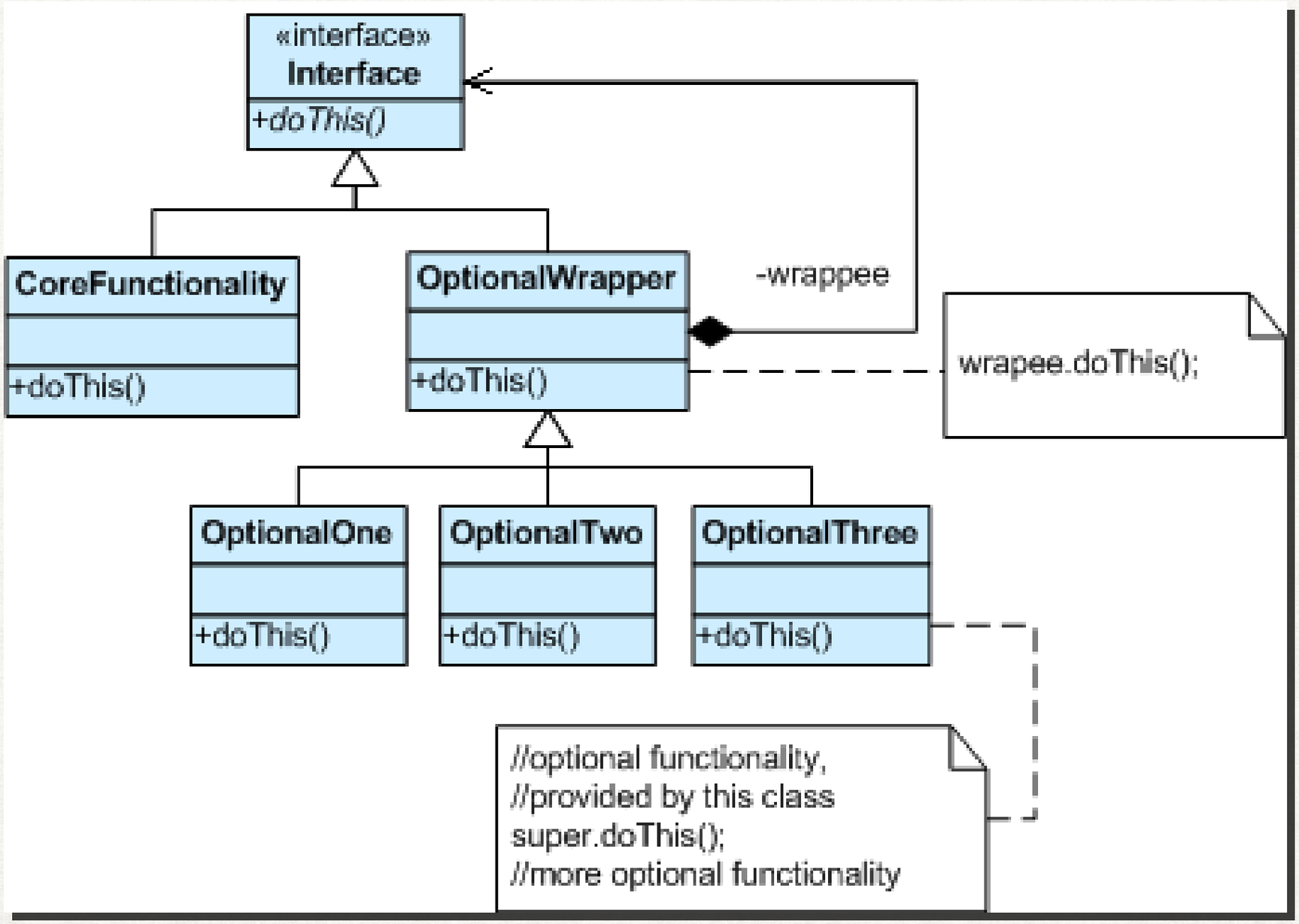
ОПИСАНИЕ

- Эта схема имеет существенный недостаток - число классов сильно разрастается.
- Паттерн Decorator дает клиенту возможность задавать любые комбинации желаемых "особенностей".
- Гибкость может быть достигнута следующим дизайном.



DECORATOR

СТРУКТУРА



DECORATOR

ИСПОЛЬЗОВАНИЕ

- Подготовьте исходные данные: один основной компонент и несколько дополнительных (необязательных) "оберток".
- Создайте общий для всех классов интерфейс по принципу "наименьшего общего знаменателя НОЗ" (lowest common denominator LCD). Этот интерфейс должен делать все классы взаимозаменяемыми.
- Создайте базовый класс второго уровня (Decorator) для поддержки дополнительных декорирующих классов.
- Основной класс и класс Decorator наследуют общий НОЗ-интерфейс.
- Класс Decorator использует отношение композиции. Указатель на НОЗ-объект инициализируется в конструкторе.
- Класс Decorator делегирует выполнение операции НОЗ-объекту.
- Для реализации каждой дополнительной функциональности создайте класс, производный от Decorator.
- Подкласс Decorator реализует дополнительную функциональность и делегирует выполнение операции базовому классу Decorator.
- Клиент несет ответственность за конфигурирование системы: устанавливает типы и последовательность использования основного объекта и декораторов.

DECORATOR

ОСОБЕННОСТИ

- Adapter придает своему объекту новый интерфейс, Proxy предоставляет тот же интерфейс, а Decorator обеспечивает расширенный интерфейс.

- Adapter изменяет интерфейс объекта. Decorator расширяет ответственность объекта. Decorator, таким образом, более прозрачен для клиента. Как следствие, Decorator поддерживает рекурсивную композицию, что невозможно с чистыми адаптерами.

- Decorator можно рассматривать как вырожденный случай Composite с единственным компонентом. Однако Decorator добавляет новые обязанности и не предназначен для агрегирования объектов.

- Decorator позволяет добавлять новые функции к объектам без наследования. Composite фокусирует внимание на представлении, а не декорировании. Эти характеристики являются различными, но взаимодополняющими, поэтому Composite и Decorator часто используются вместе.

- Decorator и Proxy имеют разное назначение, но схожие структуры. Их реализации хранят ссылку на объект, которому они отправляют запросы.

- Decorator позволяет изменить внешний облик объекта, Strategy – его внутреннее содержание.

DECORATOR

ПРОЕКТИРОВАНИЕ ПО ШАГАМ

- Создайте "наименьший общий знаменатель", делающий классы взаимозаменяемыми.
- Создайте базовый класс второго уровня для реализации дополнительной функциональности.
- Основной класс и класс-декоратор используют отношение "является".
- Класс-декоратор "имеет" экземпляр "наименьшего общего знаменателя".
- Класс Decorator делегирует выполнение операции объекту "имеет".
- Для реализации каждой дополнительной функциональности создайте подклассы Decorator.
- Подклассы Decorator делегируют выполнение операции базовому классу и реализуют дополнительную функциональность.
- Клиент несет ответственность за конфигурирование нужной функциональности.

FACADE (ФАСАД)

НАЗНАЧЕНИЕ

Предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

- Паттерн Facade предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Facade определяет интерфейс более высокого уровня, упрощающий использование подсистемы.
- Паттерн Facade "обертывает" сложную подсистему более простым интерфейсом.

FACADE

ОПИСАНИЕ

- Клиенты хотят получить упрощенный интерфейс к общей функциональности сложной подсистемы.

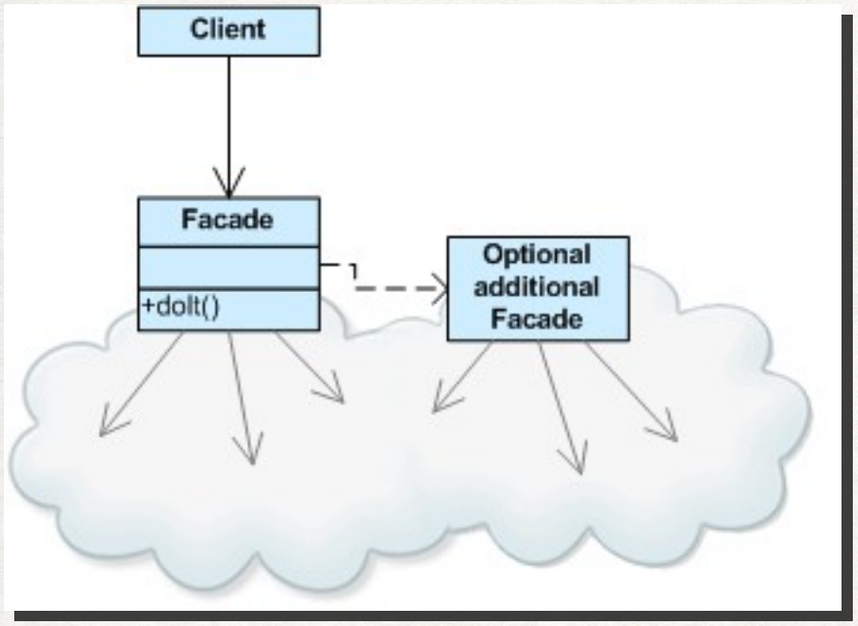
- Паттерн Facade инкапсулирует сложную подсистему в единственный интерфейсный объект. Это позволяет сократить время изучения подсистемы, а также способствует уменьшению степени связанности между подсистемой и потенциально большим количеством клиентов. С другой стороны, если фасад является единственной точкой доступа к подсистеме, то он будет ограничивать возможности, которые могут понадобиться "продвинутым" пользователям.

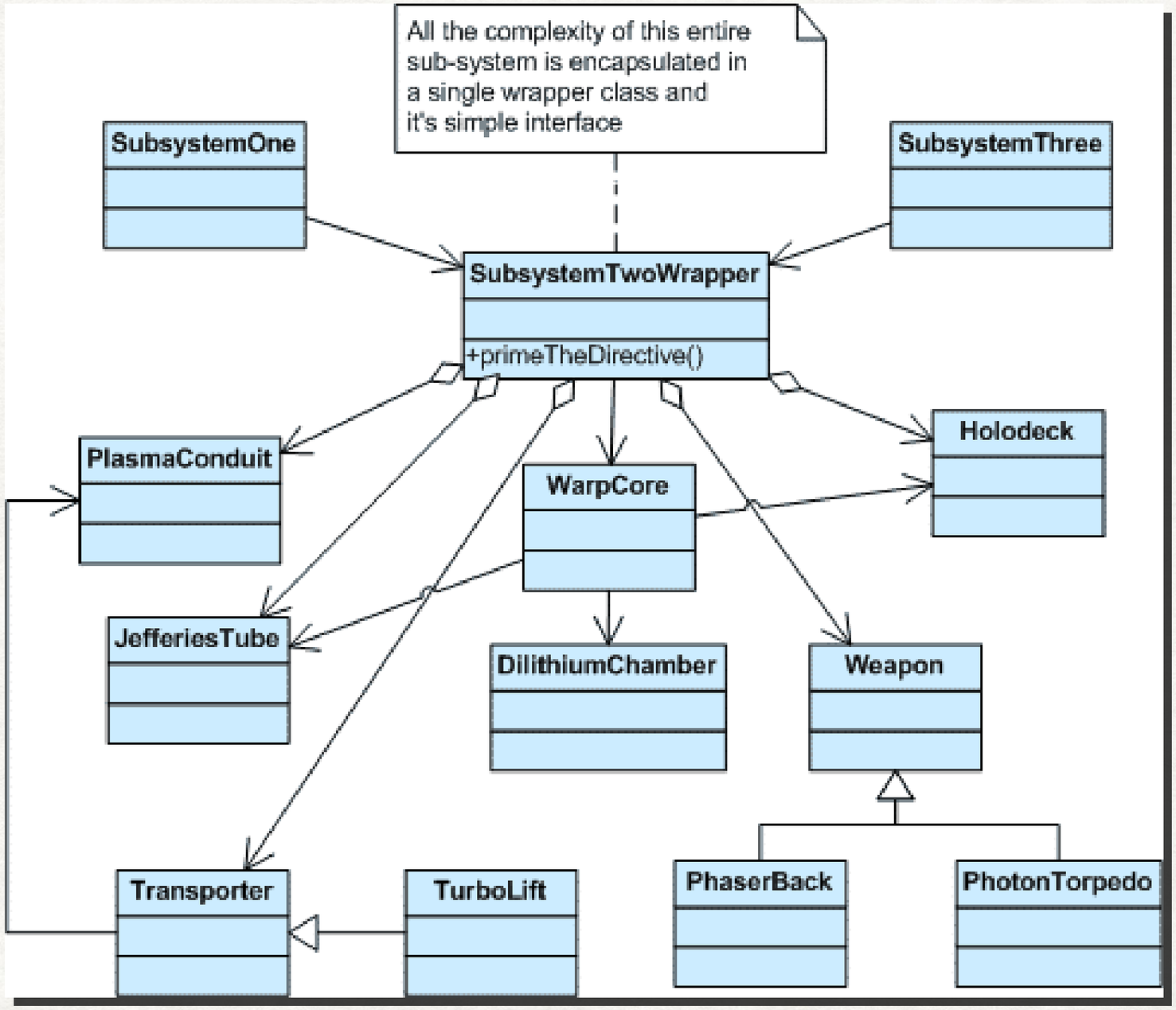
- Объект Facade, реализующий функции посредника, должен оставаться довольно простым и не быть всезнающим «оракулом".

- Клиенты общаются с подсистемой через Facade. При получении запроса от клиента объект Facade переадресует его нужному компоненту подсистемы. Для клиентов компоненты подсистемы остаются "тайной, покрытой мраком".

FACADE

СТРУКТУРА





FACADE

ИСПОЛЬЗОВАНИЕ

- Определите для подсистемы простой, унифицированный интерфейс.
- Спроектируйте класс "обертку", инкапсулирующий подсистему.
- Вся сложность подсистемы и взаимодействие ее компонентов скрыты от клиентов. "Фасад" / "обертка" переадресует пользовательские запросы подходящим методам подсистемы.
- Клиент использует только "фасад".
- Рассмотрите вопрос о целесообразности создания дополнительных "фасадов".

FACADE

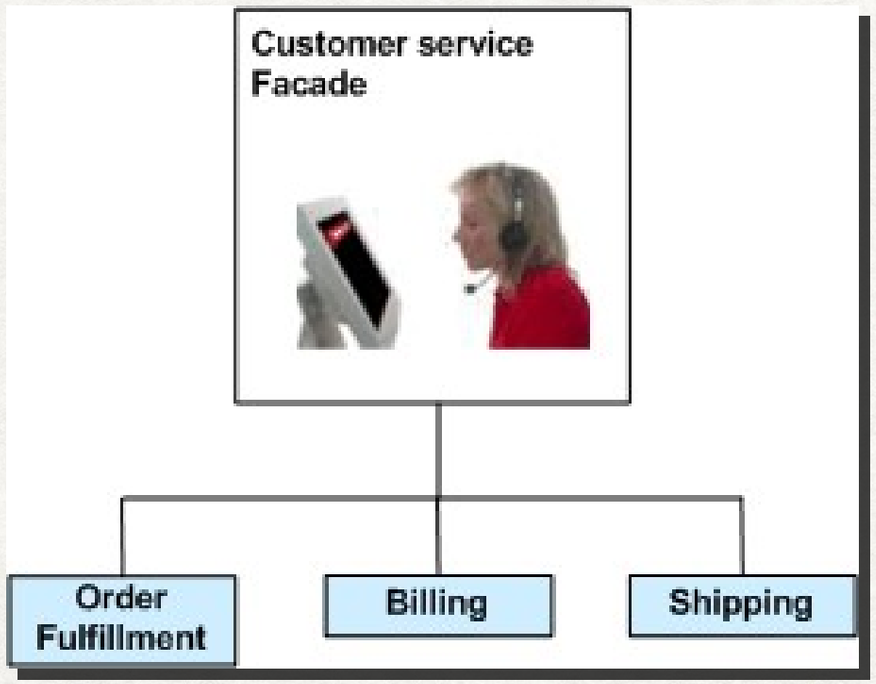
ОСОБЕННОСТИ

- Facade определяет новый интерфейс, в то время как Adapter использует уже имеющийся. Помните, Adapter делает работающими вместе два существующих интерфейса, не создавая новых.
- Если Flyweight показывает, как сделать множество небольших объектов, то Facade показывает, как сделать один объект, представляющий целую подсистему.
- Mediator похож на Facade тем, что абстрагирует функциональность существующих классов. Однако Mediator централизует функциональность между объектами-коллегами, не присущую ни одному из них. Коллеги обмениваются информацией друг с другом через Mediator. С другой стороны, Facade определяет простой интерфейс к подсистеме, не добавляет новой функциональности и не известен классам подсистемы.
- Abstract Factory может применяться как альтернатива Facade для сокрытия платформенно-зависимых классов.
- Объекты "фасадов" часто являются Singleton, потому что требуется только один объект Facade.
- Adapter и Facade в являются "обертками", однако эти "обертки" разных типов. Цель Facade – создание более простого интерфейса, цель Adapter – адаптация существующего интерфейса. Facade обычно "обертывает" несколько объектов, Adapter "обертывает" один объект.

FACADE

ПРИМЕР

- Паттерн Facade определяет унифицированный высокоуровневый интерфейс к подсистеме, что упрощает ее использование. Покупатели сталкиваются с фасадом при заказе каталожных товаров по телефону. Покупатель звонит в службу поддержки клиентов и перечисляет товары, которые хочет приобрести. Представитель службы выступает в качестве "фасада", обеспечивая интерфейс к отделу исполнения заказов, отделу продаж и службе доставки.



FLYWEIGHT (ЛЕГКОВЕС, ПРИСПОСОБЛЕНЕЦ)

НАЗНАЧЕНИЕ

- Паттерн Flyweight использует разделение для эффективной поддержки большого числа мелких объектов.
- Является стратегией Motif GUI для замены "тяжеловесных" виджетов "легковесными" гаджетами.
- Решаемая проблема - Проектирование системы из объектов самого низкого уровня обеспечивает оптимальную гибкость, но может быть неприемлемо "дорогим" решением с точки зрения производительности и расхода памяти.

FLYWEIGHT

ОПИСАНИЕ

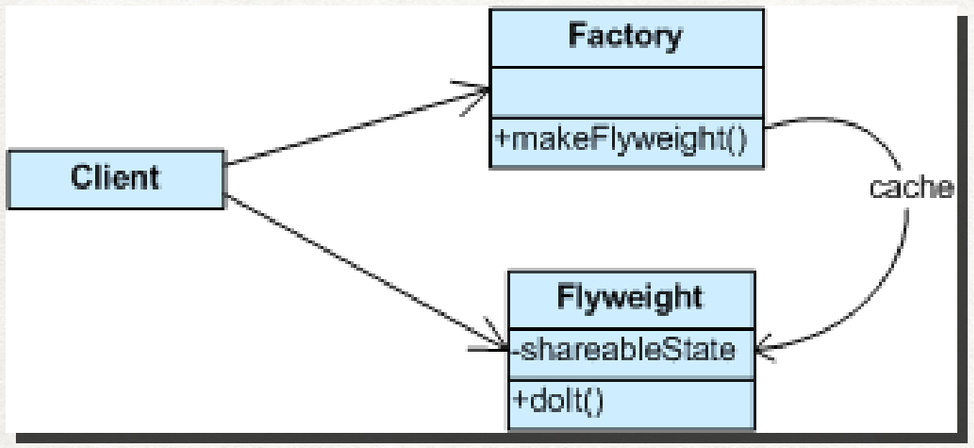
- Паттерн Flyweight описывает, как совместно разделять очень мелкие объекты без чрезмерно высоких издержек. Каждый объект-приспособленец имеет две части: внутреннее и внешнее состояния. Внутреннее состояние хранится (разделяется) в приспособленце и состоит из информации, не зависящей от его контекста. Внешнее состояние хранится или вычисляется объектами-клиентами и передается приспособленцу при вызове его методов.

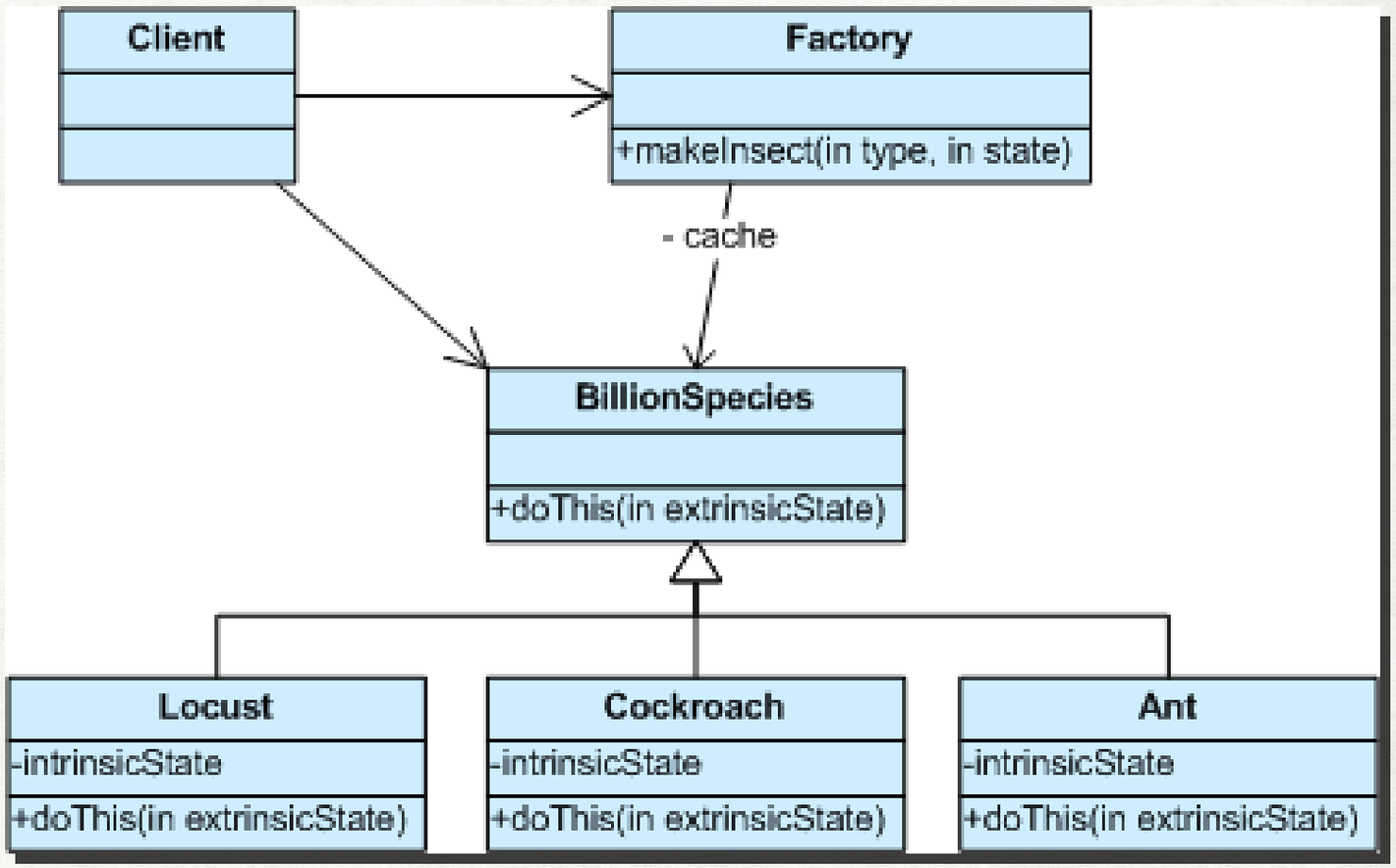
- Motif–библиотека для разработки приложений с графическим интерфейсом под X Window System. Появилась в конце 1980-х и на данный момент считается устаревшей.

- В Motif элементы графического интерфейса (кнопки, полосы прокрутки, меню и т.д.) строятся на основе виджетов. Каждый виджет имеет свое окно. Исторически окна считаются "тяжеловесными" объектами. Если приложение с графическим интерфейсом использует множество виджетов, то производительность системы может упасть. Для решения этой проблемы в Motif предусмотрены гаджеты, являющиеся безоконными аналогами виджетов. Гаджеты управляются менеджерами виджетов, использующих схему "parent-children".

FLYWEIGHT

СТРУКТУРА



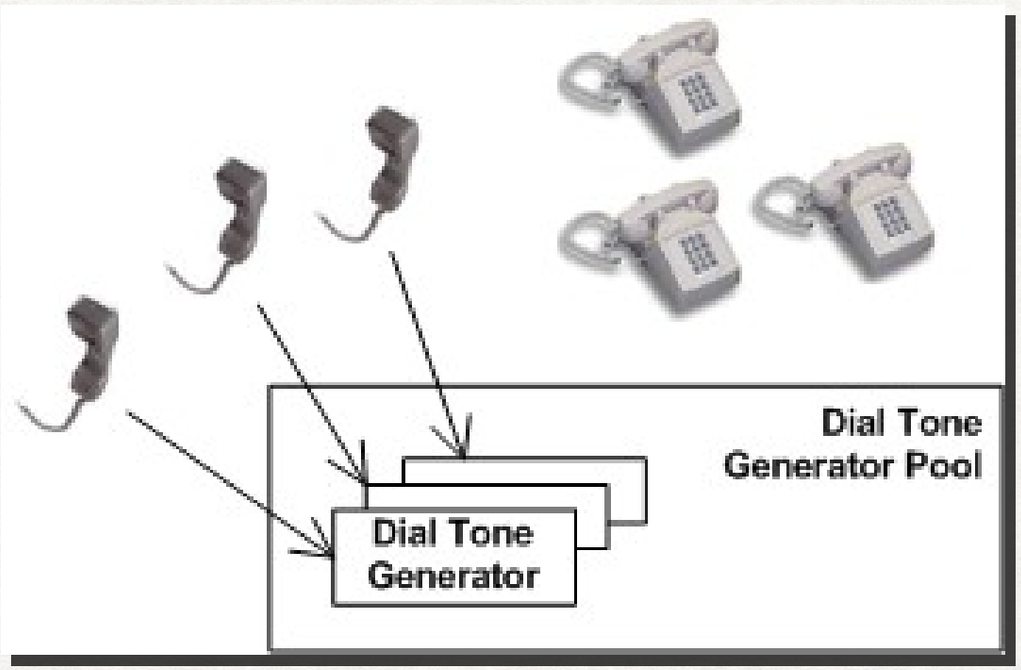


Клиенты не создают приспособленцев напрямую, а запрашивают их у фабрики. Любые атрибуты (члены данных класса), которые не могут разделяться, являются внешним состоянием. Внешнее состояние передается приспособленцу при вызове его методов. При этом наибольшая экономия памяти достигается в том случае, если внешнее состояние не хранится, а вычисляется при вызове.

FLYWEIGHT

ПРИМЕР

Паттерн Flyweight использует разделение для эффективной поддержки большого числа мелких объектов. Телефонная сеть общего пользования ТФОП является примером Flyweight. Такие ресурсы как генераторы тональных сигналов (Занято, КПВ и т.д.), приемники цифр номера абонента, набираемого в тоновом наборе, являются общими для всех абонентов. Когда абонент поднимает трубку, чтобы позвонить, ему предоставляется доступ ко всем нужным разделяемым ресурсам.

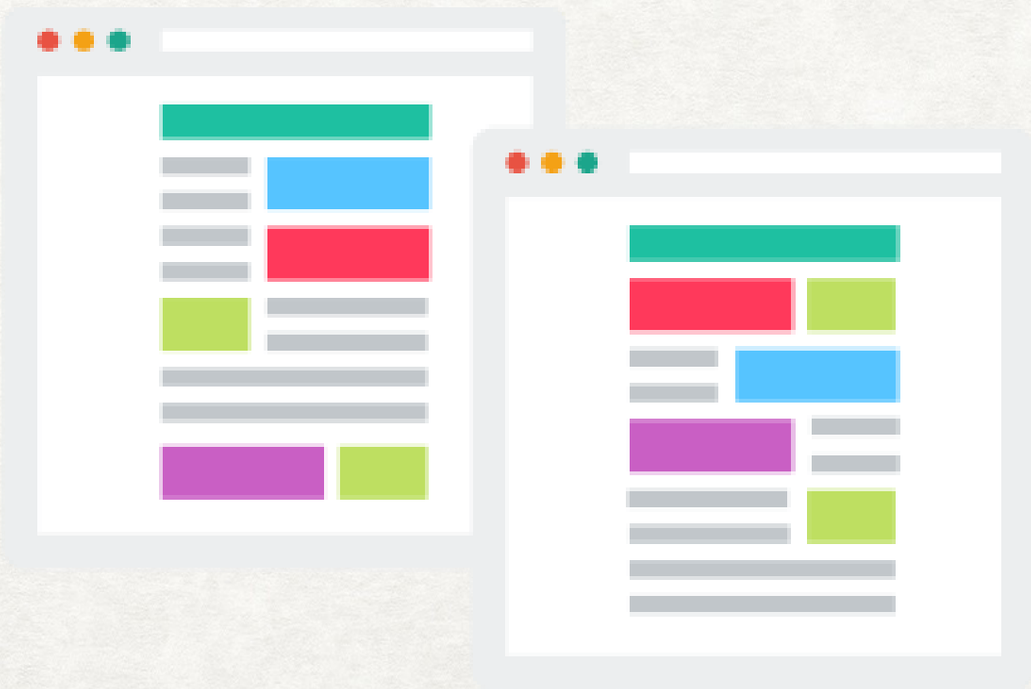


FLYWEIGHT

ПРИМЕР

Современные веб-браузеры используют этот метод для предотвращения загрузки одинаковых изображений дважды. Когда браузер загружает веб-страницу, он просматривает все изображения на этой странице. Браузер загружает все новые изображения из Интернета и помещает их в внутренний кеш. Для уже загруженных изображений создается flyweight объект, который имеет некоторые уникальные данные, такие как позиция внутри страницы.

Browser loads images just once and then reuses them from pool:



FLYWEIGHT

ИСПОЛЬЗОВАНИЕ

- Убедитесь, что существует проблема повышенных накладных расходов.
- Разделите состояние целевого класса на разделяемое (внутреннее) и неразделяемое (внешнее).
- Удалите из атрибутов (членов данных) класса неразделяемое состояние и добавьте его в список аргументов, передаваемых методам.
- Создайте фабрику, которая может кэшировать и повторно использовать существующие экземпляры класса.
- Для создания новых объектов клиент использует эту фабрику вместо оператора new.
- Клиент (или третья сторона) должен находить или вычислять неразделяемое состояние и передавать его методам класса.

FLYWEIGHT

ОСОБЕННОСТИ

- Если Flyweight показывает, как сделать множество небольших объектов, то Facade показывает, как представить целую подсистему одним объектом.
- Flyweight часто используется совместно с Composite для реализации иерархической структуры в виде графа с разделяемыми листовыми вершинами.
- Терминальные символы абстрактного синтаксического дерева Interpreter могут разделяться при помощи Flyweight.
- Flyweight объясняет, когда и как могут разделяться объекты State.

PROXY (ПРОКСИ, ЗАМЕСТИТЕЛЬ)

НАЗНАЧЕНИЕ

- Паттерн Proxy является суррогатом или заместителем другого объекта и контролирует доступ к нему.
- Предоставляя дополнительный уровень косвенности при доступе к объекту, может применяться для поддержки распределенного, управляемого или интеллектуального доступа.
- Являясь "оберткой" реального компонента, защищает его от излишней сложности.
- Вам нужно управлять ресурсоемкими объектами. Вы не хотите создавать экземпляры таких объектов до момента их реального использования.

PROXY

ОПИСАНИЕ

- Суррогат или заместитель это объект, интерфейс которого идентичен интерфейсу реального объекта. При первом запросе клиента заместитель создает реальный объект, сохраняет его адрес и затем отправляет запрос этому реальному объекту. Все последующие запросы просто переадресуются инкапсулированному реальному объекту.

- Существует четыре ситуации, когда можно использовать паттерн Proxy:

  - Виртуальный proxy является заместителем объектов, создание которых обходится дорого. Реальный объект создается только при первом запросе/доступе клиента к объекту.

  - Удаленный proxy предоставляет локального представителя для объекта, который находится в другом адресном пространстве ("заглушки" в RPC и CORBA).

  - Защитный proxy контролирует доступ к основному объекту. "Суррогатный" объект предоставляет доступ к реальному объекту, только вызывающий объект имеет соответствующие права.

  - Интеллектуальный proxy выполняет дополнительные действия при доступе к объекту.

PROXY

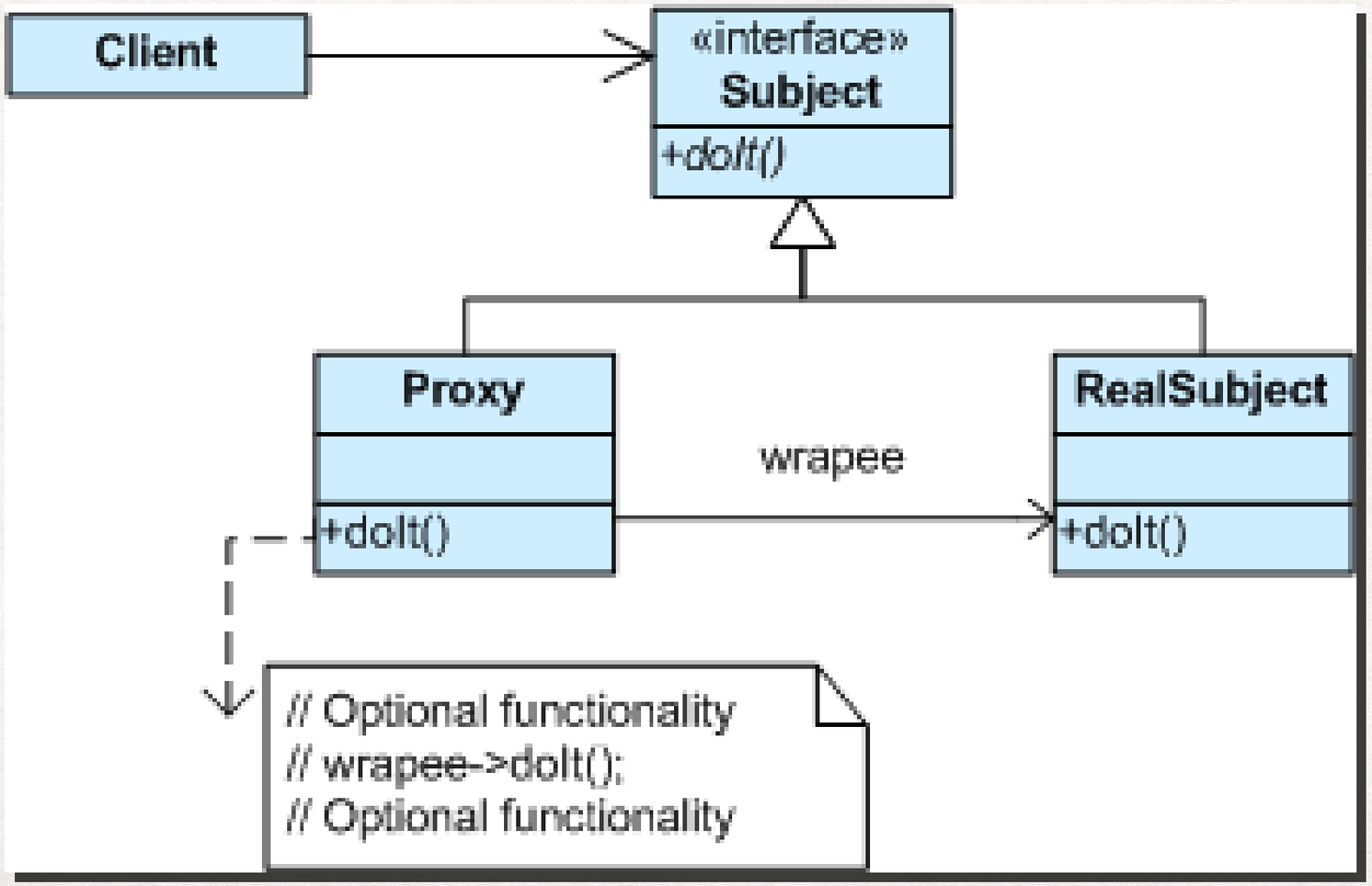
ОПИСАНИЕ

- Вот типичные области применения интеллектуальных proxy:
- Подсчет числа ссылок на реальный объект. При отсутствии ссылок память под объект автоматически освобождается (известен также как интеллектуальный указатель или smart pointer).
- Загрузка объекта в память при первом обращении к нему.
- Установка запрета на изменение реального объекта при обращении к нему других объектов.

PROXY

СТРУКТУРА

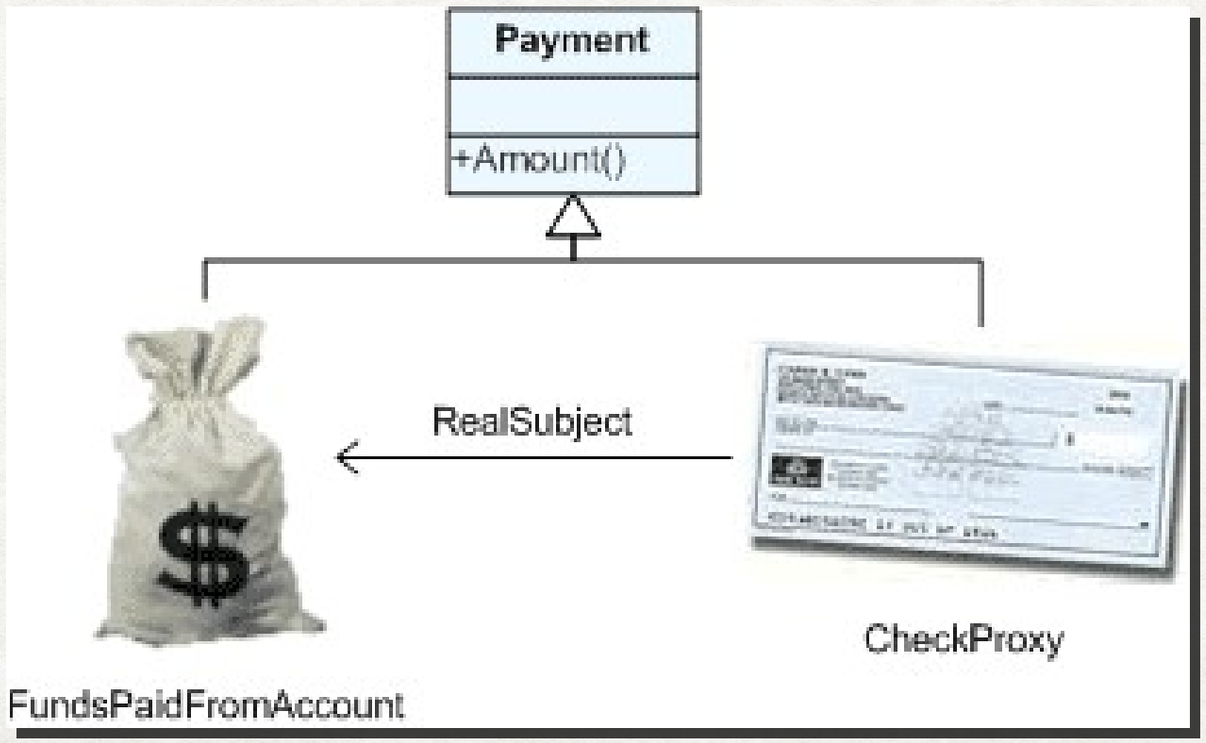
- Заместитель Proxy и реальный объект RealSubject имеют одинаковые интерфейсы класса Subject, поэтому заместитель может использоваться "прозрачно" для клиента вместо реального объекта.



PROXY

ПРИМЕР

- Паттерн Proxy для доступа к реальному объекту использует его суррогат или заместитель. Банковский чек является заместителем денежных средств на счете. Чек может быть использован вместо наличных денег для совершения покупок и, в конечном счете, контролирует доступ к наличным деньгам на счете чекодателя.



PROXY

ИСПОЛЬЗОВАНИЕ

- Определите ту часть системы, которая лучше всего реализуется через суррогата.
- Определите интерфейс, который сделает суррогата и оригинальный компонент взаимозаменяемыми.
- Рассмотрите вопрос об использовании фабрики, инкапсулирующей решение о том, что желательно использовать на практике: оригинальный объект или его суррогат.
- Класс суррогата содержит указатель на реальный объект и реализует общий интерфейс.
- Указатель на реальный объект может инициализироваться в конструкторе или при первом использовании.
- Методы суррогата выполняют дополнительные действия и вызывают методы реального объекта.

PROXY

ОСОБЕННОСТИ

- Adapter предоставляет своему объекту другой интерфейс . Proxy предоставляет тот же интерфейс. Decorator предоставляет расширенный интерфейс.

- Decorator и Proxy имеют разные цели, но схожие структуры. Оба вводят дополнительный уровень косвенности: их реализации хранят ссылку на объект, на который они отправляют запросы.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Decorator (Декоратор): (сложность 2)

Компания производит сувениры Статуэтка и Фоторамка. К нима опционально может быть добавлена Подставка и Подсветка. Каждый компонент имеет свой вес. Курьеру необходимо поставить три набора разным заказчикам: Статуэтка+Подставка, Рамка+Подставка, Рамка+Подставка+Подсветка. Необходимо напечатать вес и название каждого набора в накладную курьера.

Composite (Компоновщик): (сложность 2)

Организация состоит из сотрудников, а сотрудники, в свою очередь групируются в отделы.

Flyweight (Легковес, Приспособленец): (сложность 3)

Программа для моделирования города. Город состоит из отдельных домов, поэтому необходимо создать объекты этих домов. Они могут иметь разный вид, отличаться по различным признакам. Однако, как правило, многие дома делаются по стандартным проектам. И фактически можно выделить несколько типов домов, например, пятиэтажные кирпичные хрущевки, многоэтажные панельные высотки и так далее.

Proxy (Заместитель): (сложность 1)

Клиент получает деньги от банка через банкомат.

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Adapter (Адаптер): сложность (1)

Есть путешественник, который путешествует на машине. Но в какой-то момент ему приходится передвигаться по пескам пустыни, где он не может ехать на машине. Зато он может использовать для передвижения верблюда. Однако в классе путешественника использование класса верблюда не предусмотрено.

Bridge (Мост): сложность (3)

Есть сайт с разными страницами, и пользователь должен иметь возможность изменять тему.

Facade (Фасад): (сложность 1)

Отправка почтовой посылки

- Заказывается курьер
- Упаковывается посылка
- Оплачивается пересылка