Частное учреждение образования

«Колледж бизнеса и права»

УТВЕРЖДАЮ

Ведущий

методист колледжа

\_\_\_\_\_\_\_\_\_\_\_ Е.В. Паскал

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021

|  |  |
| --- | --- |
| Специальность: 2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Технология разработки программного обеспечения» |

**Лабораторная работа № 21**

**Инструкционно-технологическая карта**

Тема: Создание программной системы, использующей структурные шаблоны

Цель: Научиться создавать программные системы, используя структурные шаблоны

Время выполнения: 2 часа

**Контрольныевопросы**

1. Назовите и опишите структурные шаблоны проектирования

**Теоретические сведения для выполнения работы**

**Адаптер**

Адаптер (англ. Adapter или англ. Wrapper — Обёртка) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

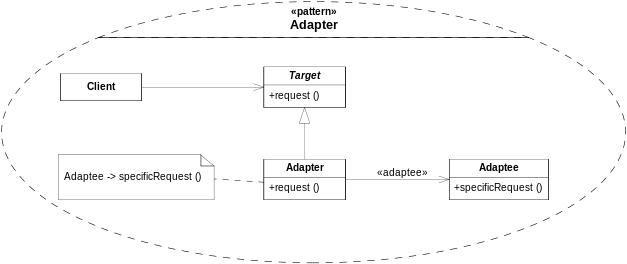
Задача

Система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс.

Участники

Класс Adapter приводит интерфейс класса Adaptee в соответствие с интерфейсом класса Target (наследником которого является Adapter). Это позволяет объекту Client использовать объект Adaptee (посредством адаптера Adapter) так, словно он является экземпляром класса Target.

Таким образом Client обращается к интерфейсу Target, реализованному в наследнике Adapter, который перенаправляет обращение к Adaptee.



Следствия

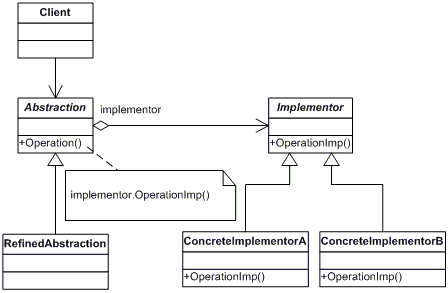
Шаблон Адаптер позволяет включать уже существующие объекты в новые объектные структуры, независимо от различий в их интерфейсах.

**Мост**

Шаблон мост (англ. Bridge) — структурный шаблон проектирования, используемый в проектировании программного обеспечения чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон bridge (от англ. — мост) использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.

Цель

При частом изменении класса преимущества объектно-ориентированного подхода становятся очень полезными, позволяя делать изменения в программе, обладая минимальными сведениями о реализации программы. Шаблон bridge является полезным там, где часто меняется не только сам класс, но и то, что он делает.



Описание

Когда абстракция и реализация разделены, они могут изменяться независимо. Другими словами, при реализации через паттерн мост, изменение структуры интерфейса не мешает изменению структуры реализации. Рассмотрим такую абстракцию как фигура. Существует множество типов фигур, каждая со своими свойствами и методами. Однако есть что-то, что объединяет все фигуры. Например, каждая фигура должна уметь рисовать себя, масштабироваться и т. п. В то же время рисование графики может отличаться в зависимости от типа ОС, или графической библиотеки. Фигуры должны иметь возможность рисовать себя в различных графических средах, но реализовывать в каждой фигуре все способы рисования или модифицировать фигуру каждый раз при изменении способа рисования непрактично. В этом случае помогает шаблон bridge, позволяя создавать новые классы, которые будут реализовывать рисование в различных графических средах. При использовании такого подхода очень легко можно добавлять как новые фигуры, так и способы их рисования.

Связь, изображаемая стрелкой на диаграммах, может иметь 2 смысла: а) "разновидность", в соответствии с принципом подстановки Б. Лисков и б) одна из возможных реализаций абстракции. Обычно в языках используется наследование для реализации как а), так и б), что приводит к разбуханию иерархий классов.

Мост служит именно для решения этой проблемы: объекты создаются парами из объекта класса иерархии А и иерархии B, наследование внутри иерархии А имеет смысл "разновидность" по Лисков, а для понятия "реализация абстракции" используется ссылка из объекта A в парный ему объект B.

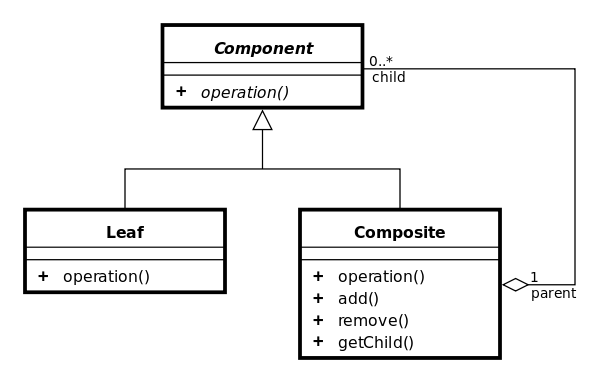
**Компоновщик**

Компоновщик (англ. Composite pattern) — структурный шаблон проектирования, относится к структурным паттернам, объединяет объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.

Цель

Паттерн определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

Описание



**Декоратор**

Декоратор (англ. Decorator) — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

Задача

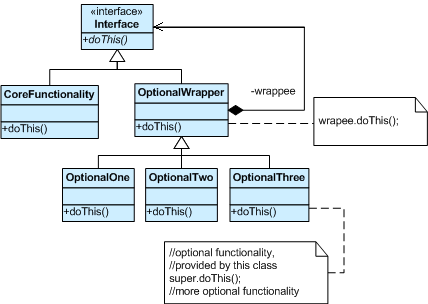
Объект, который предполагается использовать, выполняет основные функции. Однако может потребоваться добавить к нему некоторую дополнительную функциональность, которая будет выполняться до, после или даже вместо основной функциональности объекта.

Способ решения

Декоратор предусматривает расширение функциональности объекта без определения подклассов.

Участники

Класс ConcreteComponent — класс, в который с помощью шаблона Декоратор добавляется новая функциональность. В некоторых случаях базовая функциональность предоставляется классами, производными от класса ConcreteComponent. В подобных случаях класс ConcreteComponent является уже не конкретным, а абстрактным. Абстрактный класс Component определяет интерфейс для использования всех этих классов.



Следствия

Добавляемая функциональность реализуется в небольших объектах. Преимущество состоит в возможности динамически добавлять эту функциональность до или после основной функциональности объекта ConcreteComponent.

Позволяет избегать перегрузки функциональными классами на верхних уровнях иерархии

Декоратор и его компоненты не являются идентичными

**Фасад**

Шаблон фасад (англ. Facade) — структурный шаблон проектирования, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

Проблема

Как обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой, если нежелательно высокое связывание с этой подсистемой или реализация подсистемы может измениться?

Описание

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".



Решение

Определить одну точку взаимодействия с подсистемой — фасадный объект, обеспечивающий общий интерфейс с подсистемой, и возложить на него обязанность по взаимодействию с её компонентами. Фасад — это внешний объект, обеспечивающий единственную точку входа для служб подсистемы. Реализация других компонентов подсистемы закрыта и не видна внешним компонентам. Фасадный объект обеспечивает реализацию GRASP паттерна Устойчивый к изменениям (Protected Variations) с точки зрения защиты от изменений в реализации подсистемы.

Особенности применения

Шаблон применяется для установки некоторого рода политики по отношению к другой группе объектов. Если политика должна быть яркой и заметной, следует воспользоваться услугами шаблона Фасад. Если же необходимо обеспечить скрытность и аккуратность (прозрачность), более подходящим выбором является шаблон Заместитель (Proxy).

**Приспособленец**

Приспособленец (англ. Flyweight) — структурный шаблон проектирования, при котором объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым.

Цель

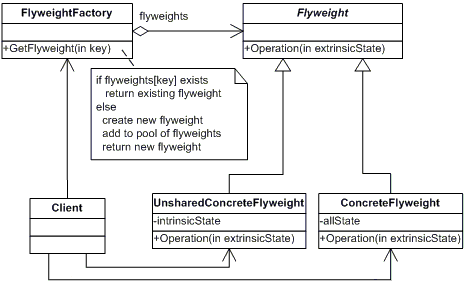
Оптимизация работы с памятью, путем предотвращения создания экземпляров элементов, имеющих общую сущность.

Описание

Flyweight используется для уменьшения затрат при работе с большим количеством мелких объектов. При проектировании приспособленца необходимо разделить его свойства на внешние и внутренние. Внутренние свойства всегда неизменны, тогда как внешние могут отличаться в зависимости от места и контекста применения и должны быть вынесены за пределы приспособленца.

Flyweight дополняет паттерн Factory таким образом, что Factory при обращении к ней клиента для создания нового объекта ищет уже созданный объект с такими же параметрами, что и у требуемого, и возвращает его клиенту. Если такого объекта нет, то фабрика создаст новый.

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.



**Заместитель**

Заместитель (англ. Proxy) — структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

Проблема

Необходимо управлять доступом к объекту так, чтобы не создавать громоздкие объекты «по требованию». Т.е. вы не хотите создавать экземпляры таких объектов до момента их реального использования.

Решение

Создать суррогат громоздкого объекта. «Заместитель» хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту (объект класса «Заместитель» может обращаться к объекту класса «Субъект», если интерфейсы «Реального Субъекта» и «Субъекта» одинаковы). Поскольку интерфейс «Реального Субъекта» идентичен интерфейсу «Субъекта», так, что «Заместителя» можно подставить вместо «Реального Субъекта», контролирует доступ к «Реальному Субъекту», может отвечать за создание или удаление «Реального Субъекта». «Субъект» определяет общий для «Реального Субъекта» и «Заместителя» интерфейс, так, что «Заместитель» может быть использован везде, где ожидается «Реальный Субъект». При необходимости запросы могут быть переадресованы «Заместителем» «Реальному Субъекту».

Виды

**Протоколирующий прокси:** сохраняет в лог все вызовы «Субъекта» с их параметрами.

**Удаленный заместитель (англ. remote proxies):** обеспечивает связь с «Субъектом», который находится в другом адресном пространстве или на удалённой машине. Так же может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному «Субъекту»,

**Виртуальный заместитель (англ. virtual proxies):** обеспечивает создание реального «Субъекта» только тогда, когда он действительно понадобится. Так же может кэшировать часть информации о реальном «Субъекте», чтобы отложить его создание,

**Копировать-при-записи:** обеспечивает копирование «субъекта» при выполнении клиентом определённых действий (частный случай «виртуального прокси»).

**Защищающий заместитель (англ. protection proxies):** может проверять, имеет ли вызывающий объект необходимые для выполнения запроса права.

**Кэширующий прокси:** обеспечивает временное хранение результатов расчёта до отдачи их множественным клиентам, которые могут разделить эти результаты.

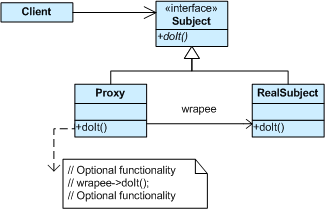
**Экранирующий прокси:** защищает «Субъект» от опасных клиентов (или наоборот).

**Синхронизирующий прокси:** производит синхронизированный контроль доступа к «Субъекту» в асинхронной многопоточной среде.

**Smart reference proxy:** производит дополнительные действия, когда на «Субъект» создается ссылка, например, рассчитывает количество активных ссылок на «Субъект».

Описание

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



**Порядок выполнения работы**

1. Изучите теоретические сведения.

2. Реализовать программу использующую один из структурных шаблонов по теме индивидуального задания.

Преподаватель \_\_\_\_\_\_\_\_ А.С. Комаровский

Рассмотрено на заседании цикловой

комиссии ПОИТ №10

Протокол № от «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021

Председатель ЦК \_\_\_\_\_\_\_\_ В.Ю. Михалевич