Шаблоны проектирования

Демьяненко К.О.
Левин И.В.
ВМИ - 302

История зарождения

Кристофер Александер

- Архитектор и дизайнер
- Создатель более 200 архитектурных проектов в Калифорнии, Японии, Мексике и в других частях мира.
- Создал и внедрял на практике «язык шаблонов» в архитектуре.
- Основные труды:

  “Notes On The Synthesis Of Form” (1964)

  “The Pattern Language” (1977)

  “The Timeless Way of Building” (1979)
- Основной задачей при декомпозиции системы является осуществление следующих двух условий:

  1. максимизация связей внутри компонентов (высокое сцепление, high cohesion);

  2. минимизация связей между компонентами (низкая связанность, low coupling).

Дальнейшие упоминания

- Эд Йордон (Ed Yourdon) и Ларри Константайн (Larry L. Constantine) - “Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design” (1979): определили понятия модульности ПО – сцепление и связность.
- Вард Каннингем (Ward Cunningham) и Кент Бек (Kent Beck) – «Пользователь должен сам писать себе программы».
- Эрих Гамма (Erich Gamma), Ричард Хелм (Richard Helm), Ральф Джонсон (Ralph Johnson), Джон Влиссидс (John Vlissides), также известные как «Банда четырёх», Gang of Four, GoF - Design Patterns: Elements of Reusable Object-Oriented Software, первый каталог шаблонов проектирования.
- Продолжения идеи: существуют шаблоны кодирования, шаблоны рефакторинга, шаблоны реализации корпоративных приложений, шаблоны работы с базами данных,, шаблоны работы с многопоточностью и множество других.

Шаблоны (паттерны) проектирования

- Оптимизированные, универсальные решения для наиболее часто встречающихся проблем программирования.
- Шаблон не является классом или библиотекой, которые можно просто включить в нашу систему — это более широкое понятие, а именно архитектура, которая может быть применена в соответствующем случае. Хороший шаблон должен быть совместим с большинством языков программирования.

Зачем их использовать?

- Любой шаблон проектирования может стать палкой о двух концах: если он будет применен не к месту, то может создать много проблем в будущем. В то же время, реализованный в нужном месте, в нужное время, он может стать настоящим спасителем.
- Шаблоны проектирования — хорошо продуманные решения проблем в программировании. Многие программисты сталкивались с этими проблемами ранее и использовали решения для их устранения.

Классификация шаблонов

- **Структурные** шаблоны в целом определяют взаимодействия между объектами, облегчая их совместное применение.
- **Порождающие** шаблоны обеспечивают механизмы инстанцирования, с помощью чего оптимизируется процесс создания объектов под конкретные ситуации.
- **Поведенческие** шаблоны используются для обеспечения связей между объектами и делают их более простыми и гибкими, другими словами обеспечивает оптимизацию коммуникаций между ними.

Структурные шаблоны

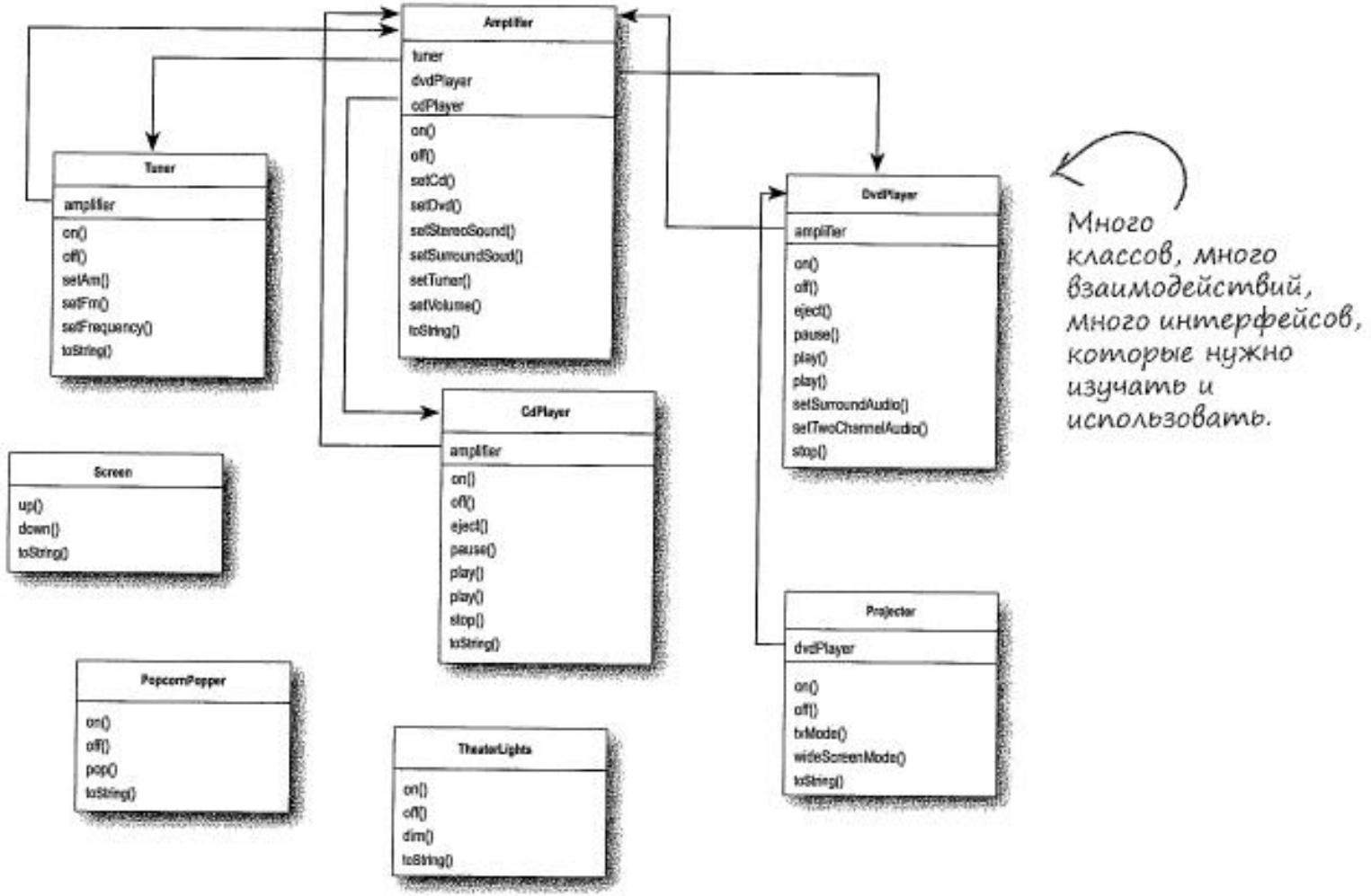
- Фасад (Facade)
- Адаптер (Adapter)
- Компоновщик (Composite)
- Декоратор (Decorator)
- Мост (Bridge)

Фасад (Facade)

Описание

- Шаблон **Фасад** — структурный шаблон проектирования, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

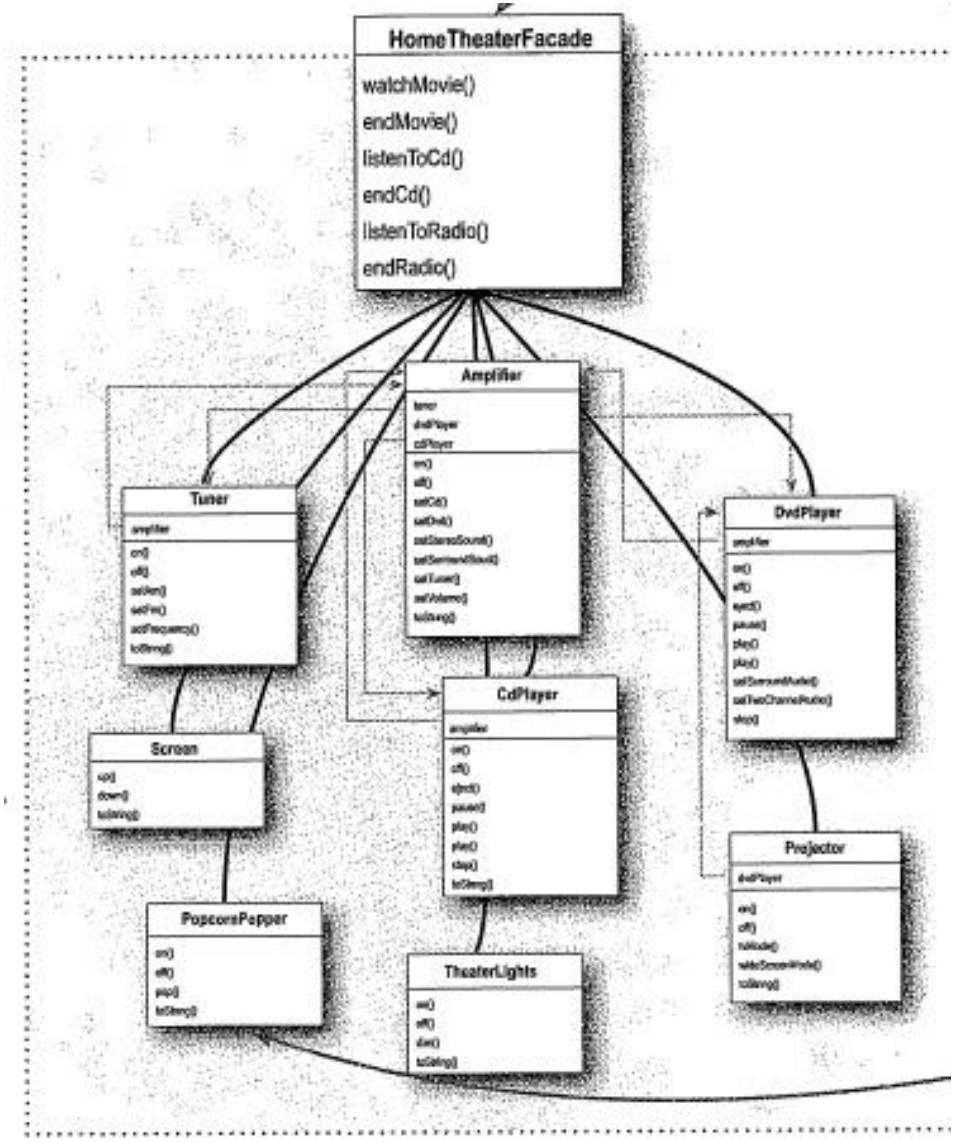
Пример сложной системы



Как включить фильм?

1. Включить аппарат для попкорна.
2. Запустить приготовление попкорна.
3. Выключить свет.
4. Опустить экран.
5. Включить проектор.
6. Связать вход проектора с выходом DVD.
7. Включить полноэкранный режим на проекторе.
8. Включить усилитель.
9. Связать вход усилителя с выходом DVD.
10. Включить на усилителе режим окружающего звука
11. Установить на усилителе среднюю громкость (5).
12. Включить DVD-проигрыватель.
13. Включить воспроизведение на DVD-проигрывателе.

Создаем фасад



Пример кода на JAVA

```
public class HomeTheaterFacade {
    Amplifier amp;
    Tuner tuner;
    DvdPlayer dvd;
    CdPlayer cd;
    Projector projector;
    TheaterLights lights;
    Screen screen;
    PopcornPopper popper;

    public HomeTheaterFacade(Amplifier amp,
                             Tuner tuner,
                             DvdPlayer dvd,
                             CdPlayer cd,
                             Projector projector,
                             Screen screen,
                             TheaterLights lights,
                             PopcornPopper popper) {

        this.amp = amp;
        this.tuner = tuner;
        this.dvd = dvd;
        this.cd = cd;
        this.projector = projector;
        this.screen = screen;
        this.lights = lights;
        this.popper = popper;
    }

        // Другие методы

}
```

Композиция: компоненты подсистемы, которые мы собираемся использовать.

В конструкторе фасада передаются ссылки на все компоненты. Фасад присваивает их соответствующим переменным экземпляра.

Сейчас мы займемся ими...

Пример кода на JAVA 2

```
public void watchMovie(String movie) {
    System.out.println("Get ready to watch a movie...");
    popper.on();
    popper.pop();
    lights.dim(10);
    screen.down();
    projector.on();
    projector.wideScreenMode();
    amp.on();
    amp.setDvd(dvd);
    amp.setSurroundSound();
    amp.setVolume(5);
    dvd.on();
    dvd.play(movie);
}

public void endMovie() {
    System.out.println("Shutting movie theater down...");
    popper.off();
    lights.on();
    screen.up();
    projector.off();
    amp.off();
    dvd.stop();
    dvd.eject();
    dvd.off();
}
```

Метод watchMovie() выполняет те же операции, которые ранее выполнялись нами вручную. Обратите внимание: выполнение каждой операции делегируется соответствующему компоненту подсистемы.

Метод endMovie() выключает всю аппаратуру за нас. И снова каждая операция делегируется соответствующему компоненту подсистемы.

Фасад

Фасад не только упрощает интерфейс, но и обеспечивает логическую изоляцию клиента от подсистемы, состоящей из многих компонентов.

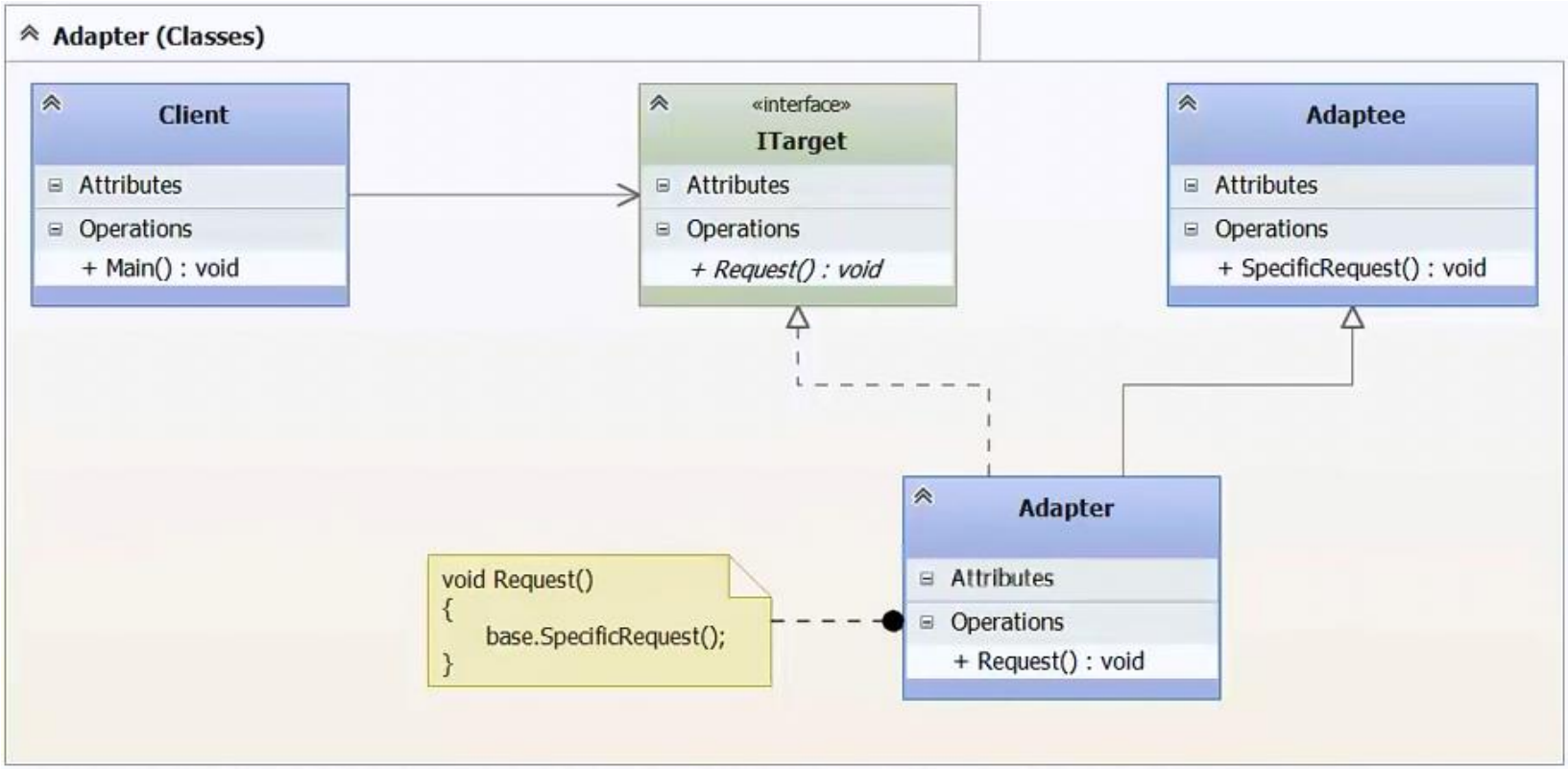
Адаптер (Adapter)

aka Обертка (Wrapper)

Назначение

- Шаблон Адаптер преобразует интерфейс (набор имен методов) одного класса в интерфейс другого класса, который ожидают клиенты. Адаптер обеспечивает совместную работу классов с несовместимыми интерфейсами, такая работа без Адаптера была бы невозможна.

Адаптер уровня классов



- 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 уровня классов

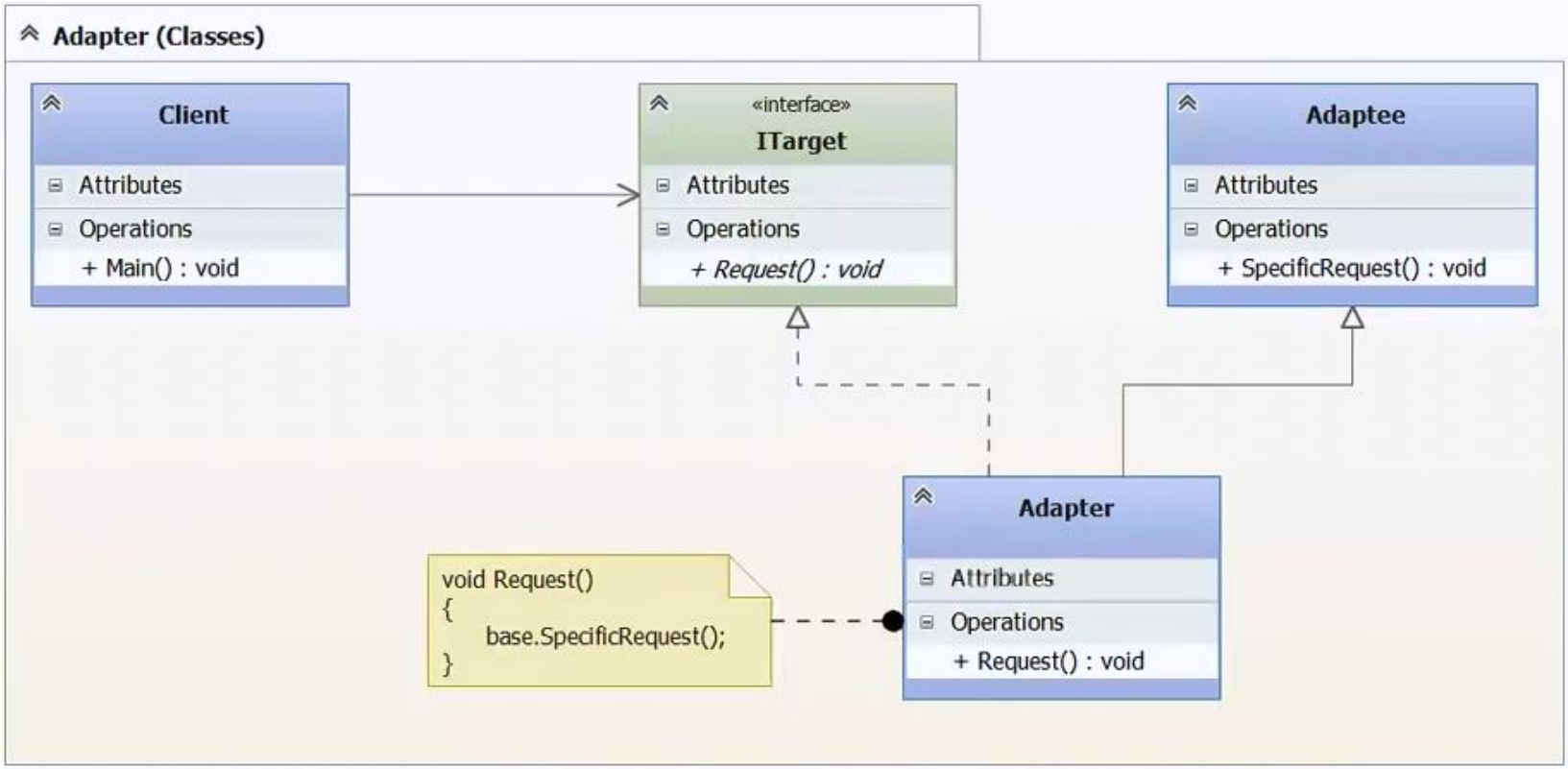
```
class Program
{
    static void Main()
    {
        ITarget target = new Adapter();
        target.Request();
    }
}
```

```
interface ITarget
{
    void Request();
}
```

```
class Adaptee
{
    public void SpecificRequest()
    {
        Console.WriteLine("SpecificRequest");
    }
}
```

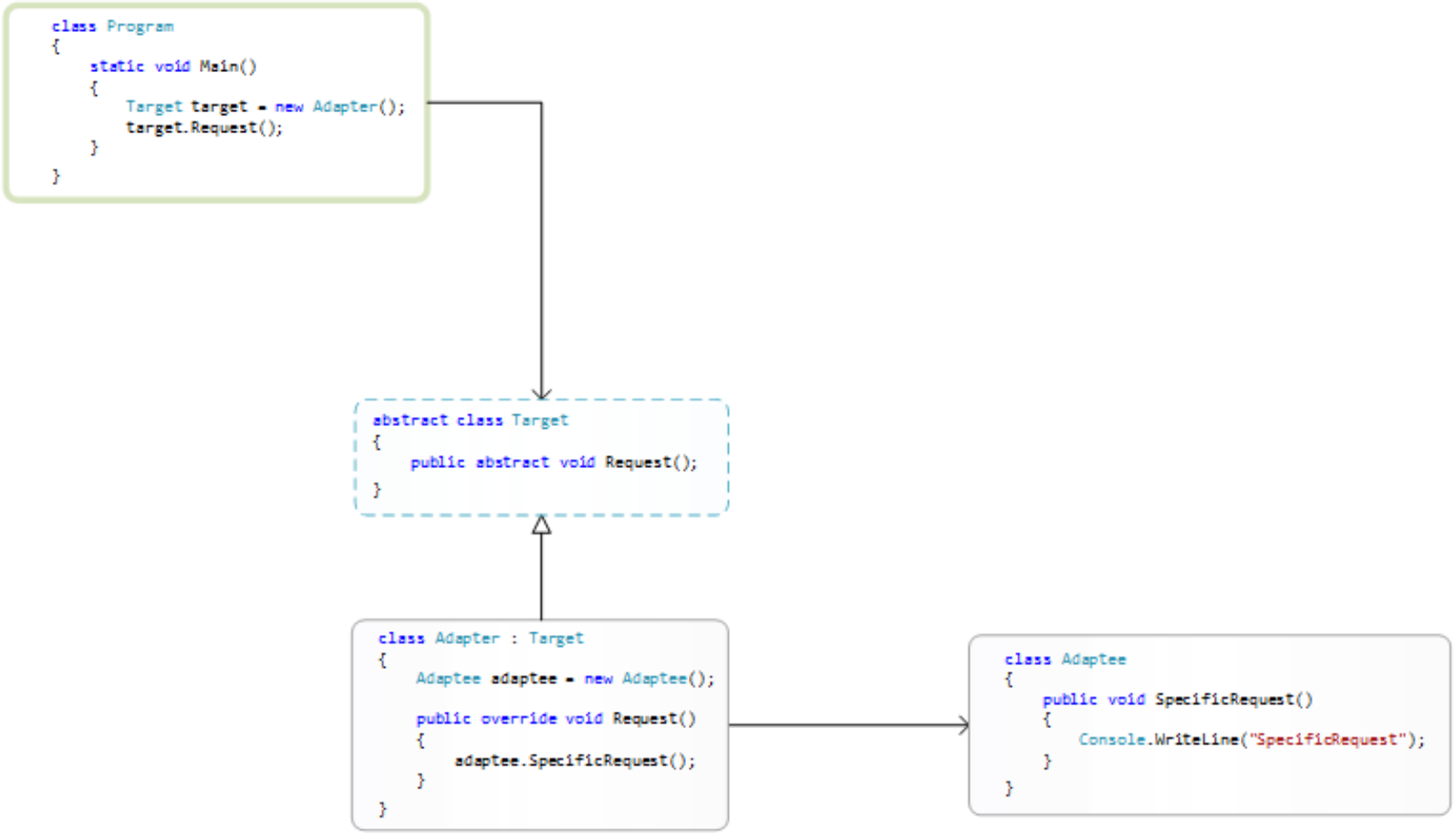
```
class Adapter : Adaptee, ITarget
{
    public void Request()
    {
        SpecificRequest();
    }
}
```

Адаптер уровня классов



- 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 уровня объектов



Применимость шаблона Адаптер

Паттерн Адаптер рекомендуется использовать, когда:

- Требуется использовать уже существующий класс, но его интерфейс (набор методов) не соответствует требованиям клиента.
- Требуется создать повторно используемый класс который должен взаимодействовать с классами имеющими не совместимые интерфейсы.

Особенности шаблона

Объем работ по адаптации.

- Устройство разных адаптеров может сильно отличаться. Класс Adapter может просто изменять имена методов класса Adaptee, но может и расширять методы класса Adaptee.

Сменные адаптеры.

- Степень многократного использования (reusable) класса «A» будет высокой, тогда и только тогда, когда разработчик класса «A» не будет делать предположений о том, какие классы будут использовать класс «A».

Двусторонние адаптеры.

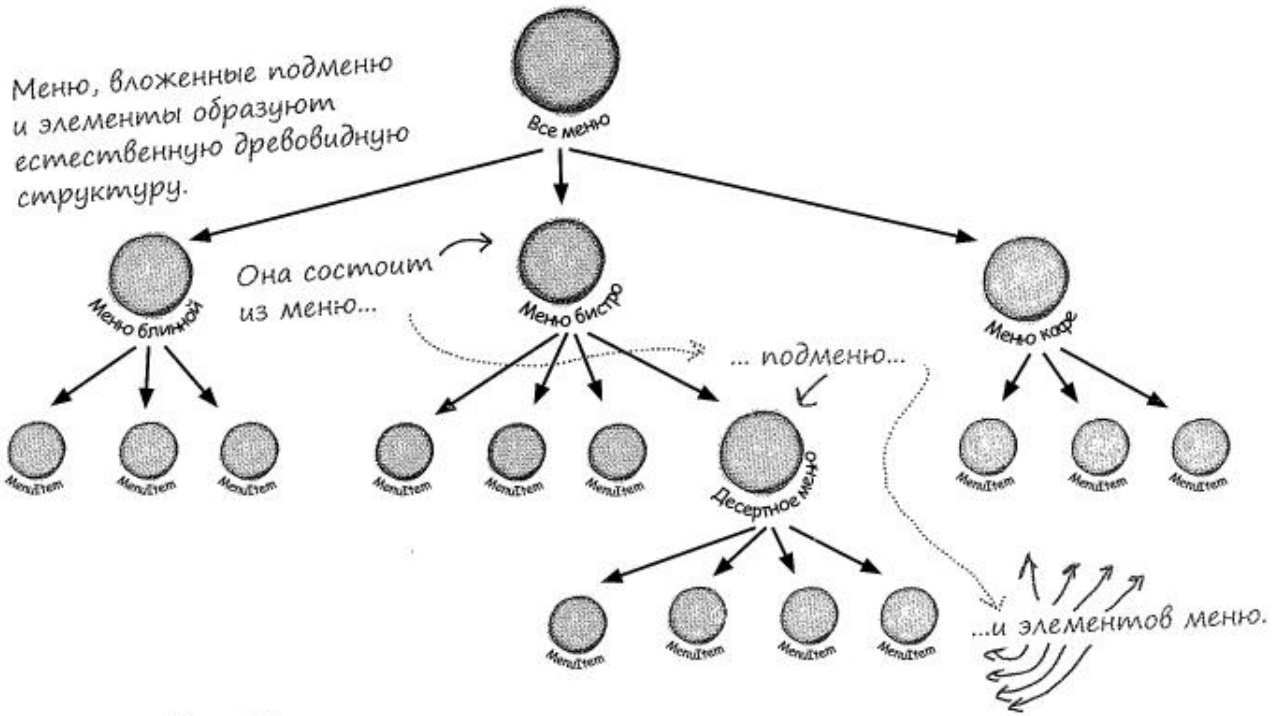
- Способны обеспечить возможность использовать адаптер там, где мог использоваться адаптируемый интерфейс.

Компоновщик (Composite)

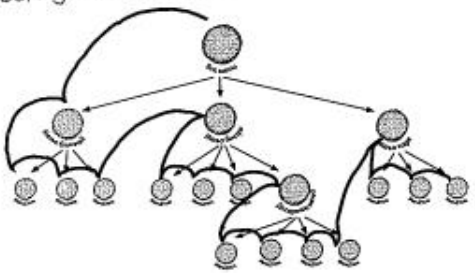
Назначение

- **Компоновщик** — структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.
- Паттерн определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

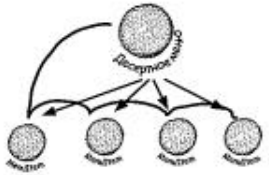
Структура паттерна



Как и прежде, нам понадобится механизм перебора всех узлов дерева.

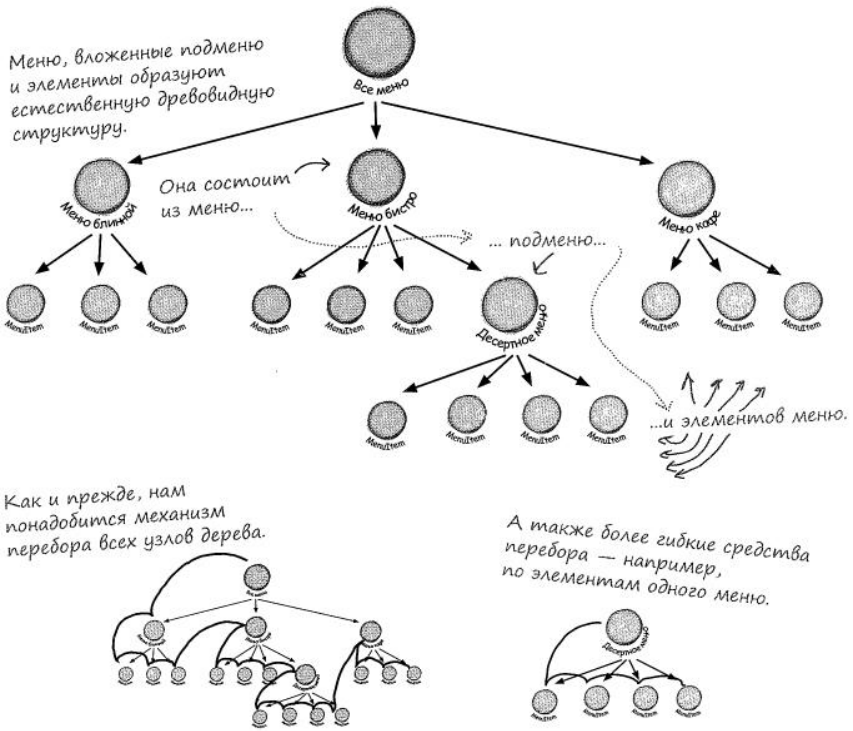


А также более гибкие средства перебора — например, по элементам одного меню.



Компоновщик

Паттерн Компоновщик объединяет объекты в древовидные структуры для представления иерархий «часть/целое». Компоновщик позволяет клиенту выполнять однородные операции с отдельными объектами и их совокупностями.



Рассмотрим это определение в контексте наших меню: паттерн дает возможность создать древовидную структуру, которая может работать с вложенными группами меню *и* элементами меню. Размещая меню и элементы в одной структуре, мы создаем иерархию «часть/целое». Иначе говоря, дерево объектов, которое состоит из отдельных частей (меню и элементы меню), но при этом может рассматриваться как единое целое (одно большое «суперменю»).

Построив «суперменю», мы можем использовать этот паттерн для того, чтобы «выполнять однородные операции с отдельными объектами и их комбинациями». Что это означает? Если у вас есть древовидная структура из меню, подменю (и, возможно, под-подменю) с элементами, любое меню представляет собой «комбинацию», так как оно может содержать другие меню и команды меню. Отдельными объектами являются только элементы меню — они не могут содержать других объектов. Применение в архитектуре паттерна Компоновщик позволит нам написать простой код, который применяет одну и ту же операцию (например, вывод!) ко всей структуре меню.

Компоновщик

Паттерн Компоновщик позволяет создавать древовидные структуры, узлами которых являются как комбинации, так и отдельные объекты.

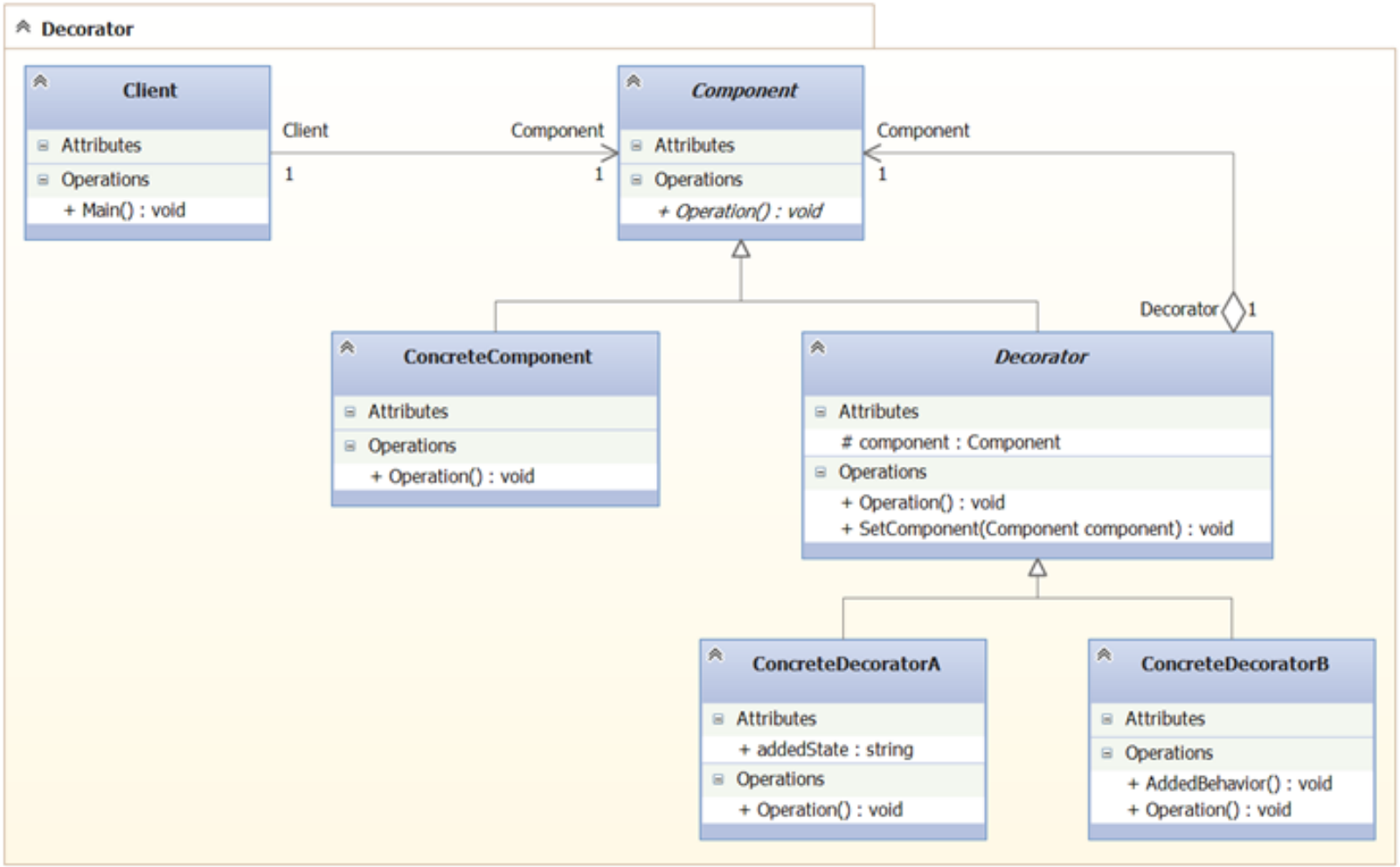
В такой структуре одни и те же операции могут применяться и к комбинациям, и к отдельным объектам. Иначе говоря, во многих случаях различия между комбинациями и отдельными объектами игнорируются.

Декоратор (Decorator)

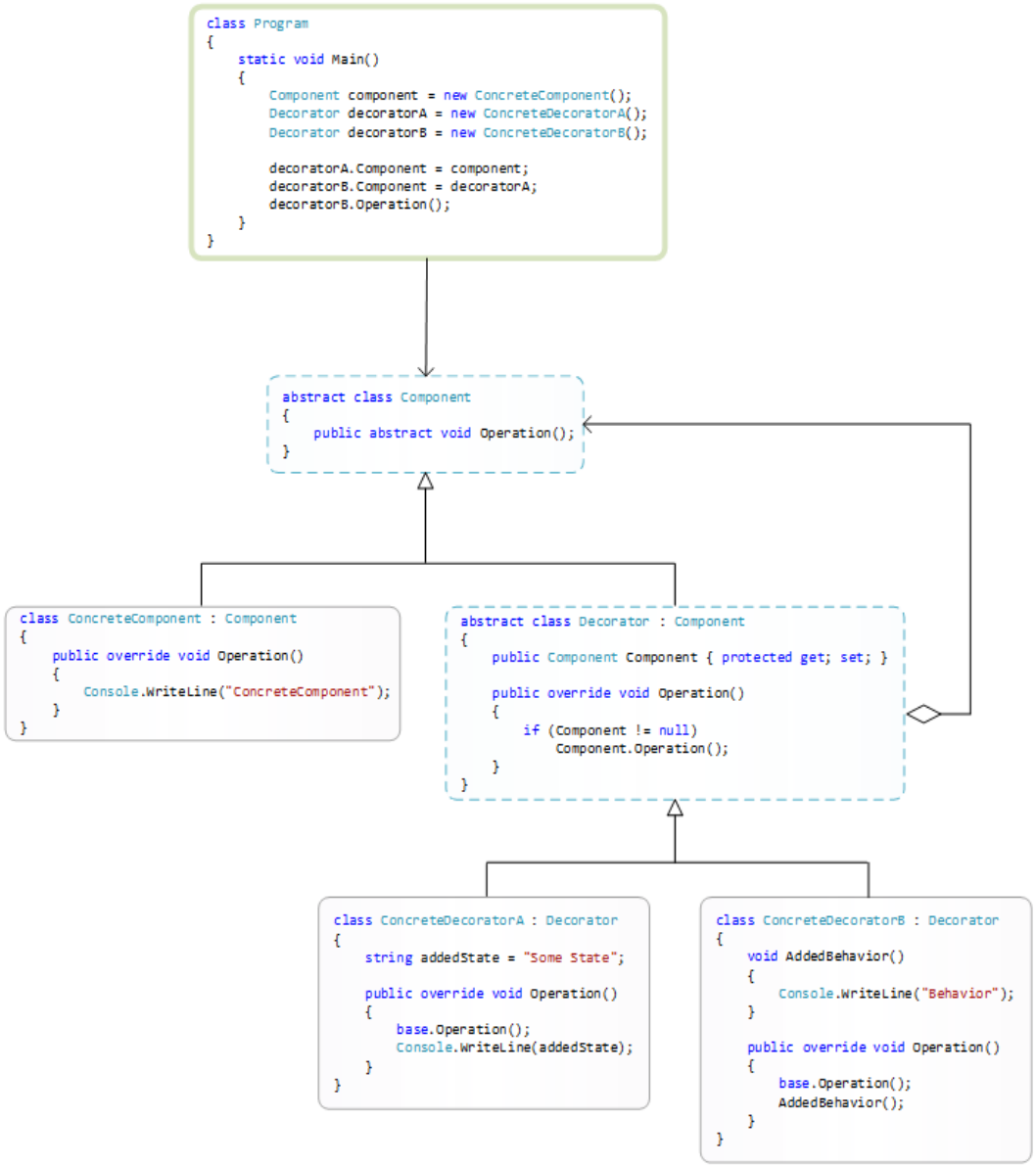
Назначение

- Динамически (в ходе выполнения программы) добавляет объекту новые возможности (состояние и/или поведение).
- Композиция, используемая при реализации паттерна, является гибкой альтернативой наследованию (порождению подклассов) с целью расширения функциональности.

Структура паттерна на языке UML



Структура паттерна на языке C#



Применимость шаблона Декоратор

Паттерн Адаптер рекомендуется использовать, когда:

- Требуется организовать возможность динамического расширения объектов.
- Требуется временно расширить объект новой функциональностью, а позже эту функциональность убрать.

Особенности шаблона

+ Гибкость

- Это свойство программной системы или компонента, позволяющие быстро, легко и безопасно вносить изменения в существующий код или расширять код новой функциональностью.

+ Необходимая достаточность.

- Паттерн позволяет организовать динамическое добавление новой функциональности объектам по мере необходимости.

- Потеря идентичности.

- Идентичность позволяет унифицировано представлять функциональные возможности объектов причисляя их к определенной группе. Декорированные объекты теряют идентичность и приобретают индивидуальность.

- Наличие большого числа мелких объектов-декораторов.

- Программист, знающий паттерны и разбирающийся в устройстве такой системы, сможет легко настраивать разные комбинации вариантов декорирования. Но новичку будет немного сложно изучать и отлаживать такую систему.