Шаблоны проектирования

Литература

Фримен, Э. Head First. Паттерны проектирования. Обновленное юбилейное издание./Фримен Э., Робсон Э., Сьерра К., Бейтс Б. — СПб.: Питер, 2018. — 656 с.: ил.

Vaskaran S. Design Patterns in C#. A Hands-on Guide with Real-world Examples 2nd ed. – Apress, 2020

Gang Of Four – Design patterns

Введение

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Введение

Паттерны (или шаблоны) проектирования описывают типичные способы решения часто встречающихся проблем при проектировании программ.

Введение

Шаблоны проектирования являются важными строительными блоками для проектирования и моделирования приложений на всех платформах.

Классификация шаблонов проектирования

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Классификация шаблонов проектирования

Шаблоны проектирования можно условно разделить на три основных группы:

- Порождающие шаблоны
- Структурные шаблоны
- Поведенческие шаблоны

Порождающие шаблоны (Creational patterns)

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Порождающие шаблоны (creational patterns)

Абстрагируют процесс инстанцирования.

Они позволяют сделать систему независимой от способа создания, композиции и представления объектов.

Порождающие шаблоны (creational patterns)

Порождающие шаблоны инкапсулируют знания о конкретных классах, которые применяются в системе, и при этом скрывают детали того, как эти классы создаются и взаимодействуют.

Единственная информация об объектах, известная системе, — это их интерфейсы, определенные с помощью абстрактных классов

Порождающие шаблоны (creational patterns)

Примеры:

- абстрактная фабрика (abstract factory);
- строитель (builder);
- фабричный метод (factory method);
- ленивая инициализация (lazy initialization);
- прототип (prototype);
- одиночка (singleton);

Фабричный метод

Определение **GOF**

Определяет интерфейс для создания объекта, но пусть подклассы (дочерние классы) решают, какой класс создавать. Фабричный метод позволяет классу откладывать создание экземпляров до подклассов.

Фабричный метод

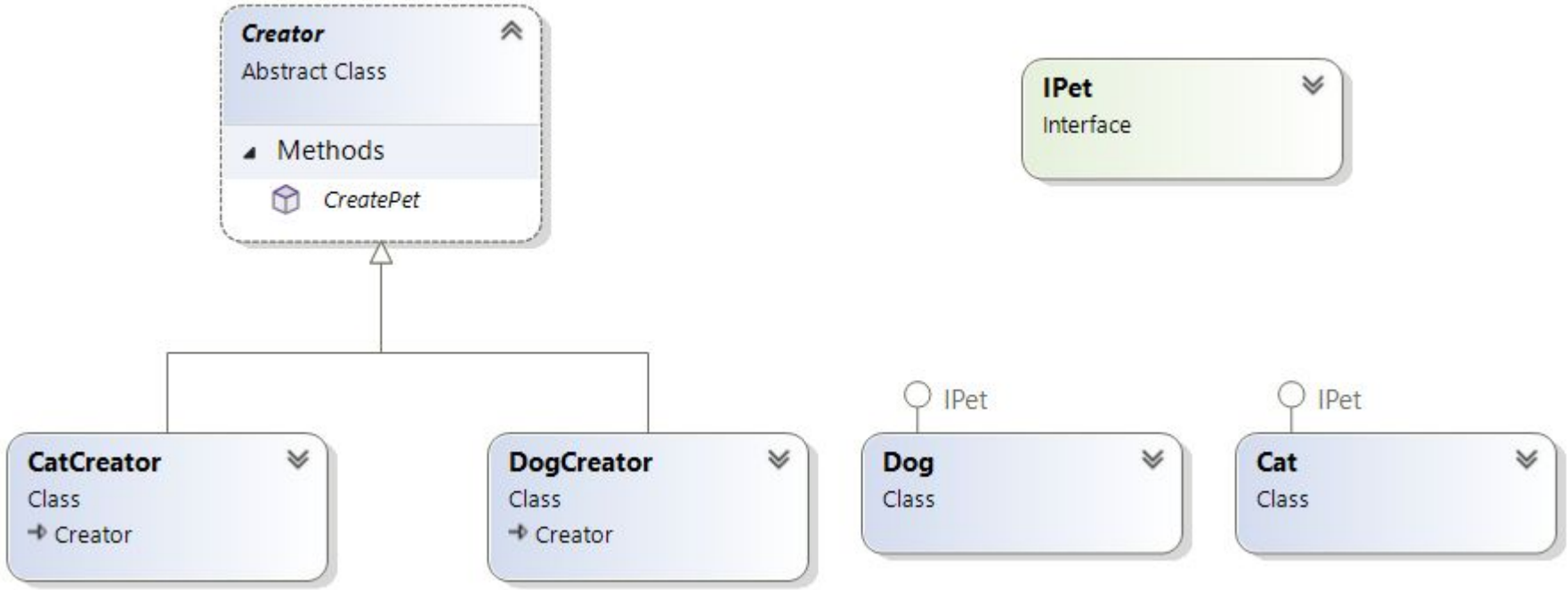
Позволяет сделать код создания объектов более универсальным, не привязываясь к конкретным классам, а оперируя лишь общим интерфейсом, описанном в базовом абстрактном классе;

Фабричный метод

Минусы:

Необходимость создавать наследника Creator для каждого нового типа объекта

Фабричный метод



Фабричный метод

```
Creator[] creators = { new DogCreator(), new CatCreator() };


foreach (Creator creator in creators)
{
    IPet pet = creator.CreatePet();
    Console.WriteLine("Created {0}", pet.GetType());
}
```

Абстрактная фабрика (Abstract factory)

Определение **GOF:**

Предоставляет интерфейс для создания семейств связанных или зависимых объектов без указания их конкретных классов.

Абстрактная фабрика (Abstract factory)

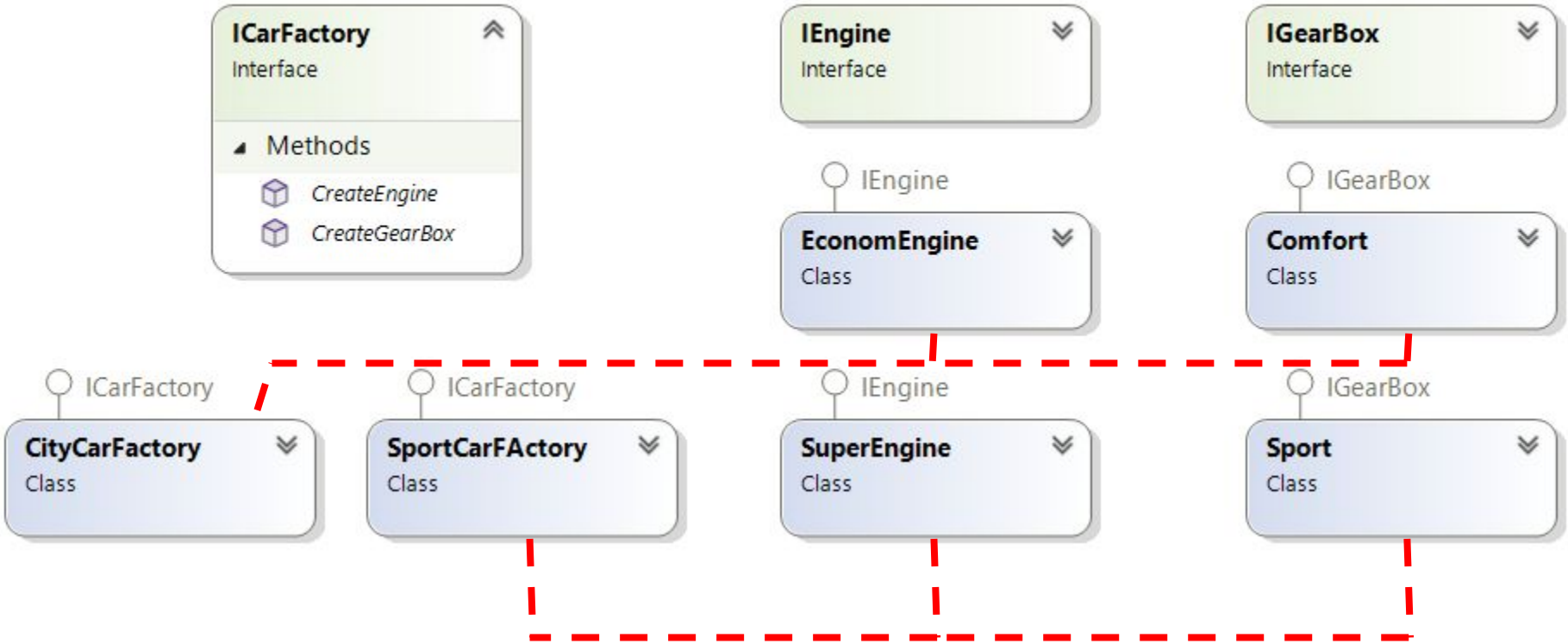
Абстрактная фабрика позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение.

Абстрактная фабрика (Abstract factory)

Абстрактная фабрика применяется, если:

- Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты.
- Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения.
- Система должна конфигурироваться одним из семейств составляющих ее объектов.

Абстрактная фабрика (Abstract factory)



Абстрактная фабрика (Abstract factory)

```
ICarFactory factory = new CityCarFactory();
```

Строитель (Builder)

Определение **GOF:**

Отделяет конструирование сложного объекта от его представления (конкретной реализации), так что в результате одного и того же процесса конструирования могут получаться разные представления.

Строитель (Builder)

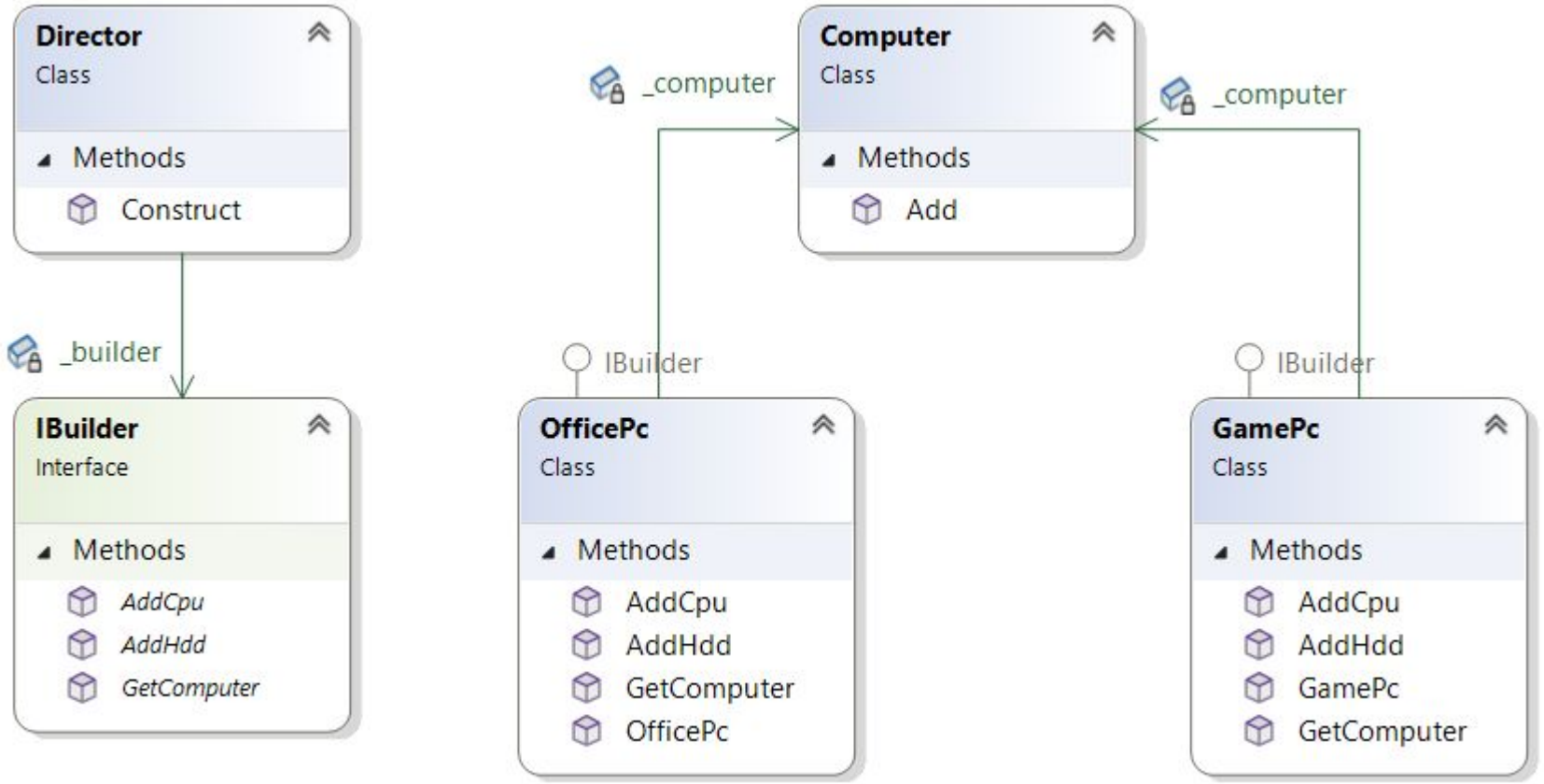
Применяется, когда алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой

Строитель (Builder)

Пример:

сборка компьютеров разной конфигурации

Строитель (Builder)



Строитель (Builder)

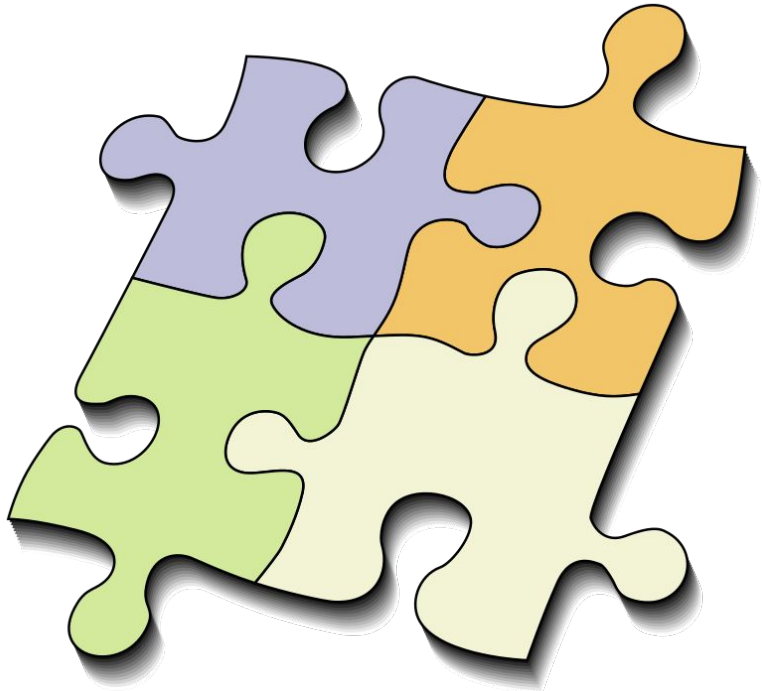
```
var director = new Director();
IBuilder gamePcBuilder = new GamePc();
var pc = director.Construct(gamePcBuilder);
```

Структурные шаблоны (Structural patterns)

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Структурные шаблоны (Structural patterns)

Структурные шаблоны — шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры.



Структурные шаблоны (structural patterns)

Примеры структурных шаблонов проектирования:

- адаптер (adapter);
- мост (bridge);
- компоновщик (composite patern);
- декоратор (decorator);
- фасад (facade)

Структурные шаблоны. Адаптер

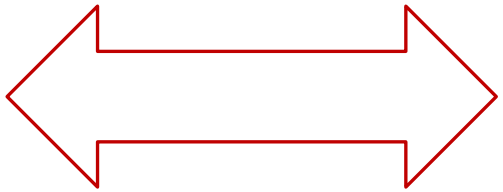
Определение **GOF:**

Преобразовывает интерфейс класса в другой интерфейс, ожидаемый клиентом.

Адаптер позволяет классам работать вместе, что в противном случае было бы невозможно из-за несовместимых интерфейсов.

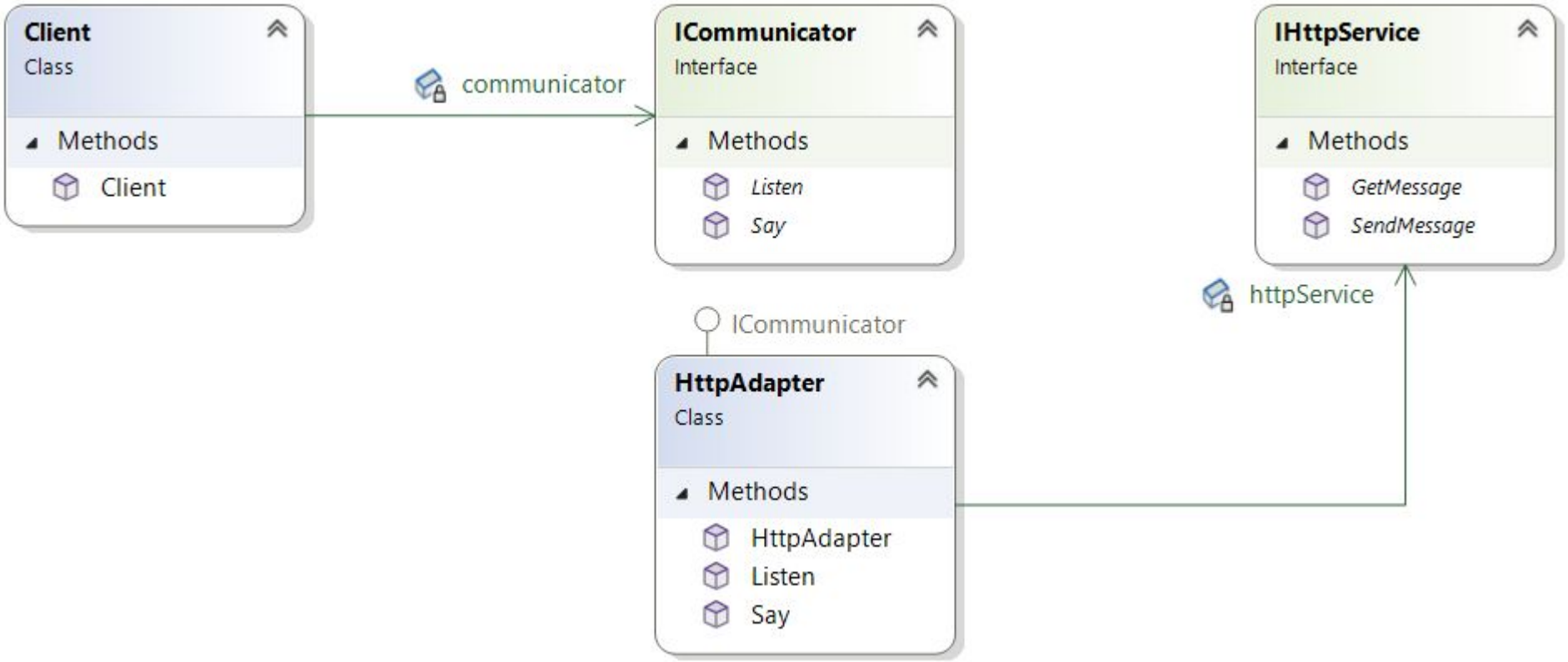
Структурные шаблоны. Адаптер

?



Структурные шаблоны. Адаптер

Структурные шаблоны. Адаптер



Структурные шаблоны. Адаптер

```
internal class HttpAdapter : ICommunicator
    {
        private readonly IHttpService httpService;

        public HttpAdapter(IHttpService httpService)
        {
            this.httpService = httpService;
        }
        public string Listen()
            => httpService.GetMessage();

        public void Say(string message)
            =>httpService.SendMessage(message);
    }
```

Структурные шаблоны: Компоновщик (Composite)

Определение **GOF:**

Объединяет объекты в древовидные структуры, чтобы представлять иерархию часть-целое.

Composite позволяет клиентам единообразно обрабатывать отдельные объекты и композиции объектов.

Структурные шаблоны: Компоновщик (Composite)

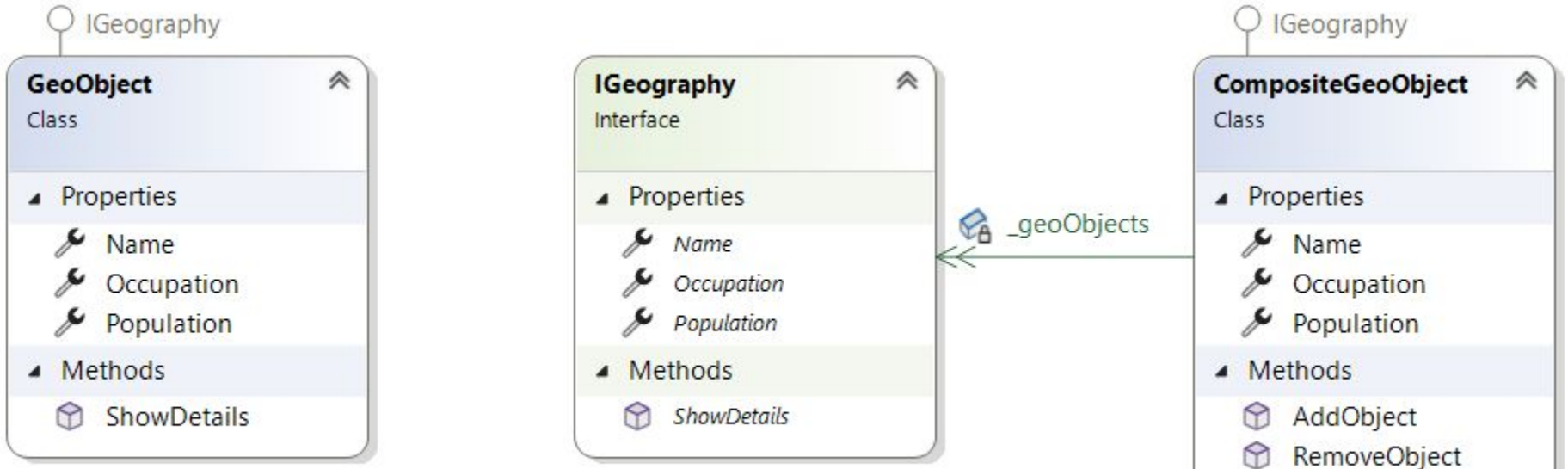
В объектно-ориентированном программировании составной объект — это объект, состоящий из одного или нескольких подобных объектов, где каждый из этих объектов имеет сходную функциональность. (Это также известно как отношение «has-a» между объектами.)

Использование этого шаблона распространено в данных с древовидной структурой. Когда вы реализуете этот шаблон в такой структуре данных, вам не нужно различать ветвь и конечные узлы дерева - вы одинаково взаимодействуете с этими двумя объектами.

Структурные шаблоны: Компоновщик (Composite)

https://edufuture.biz/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Informatika_9_313v.jpg

Структурные шаблоны: Компоновщик (Composite)



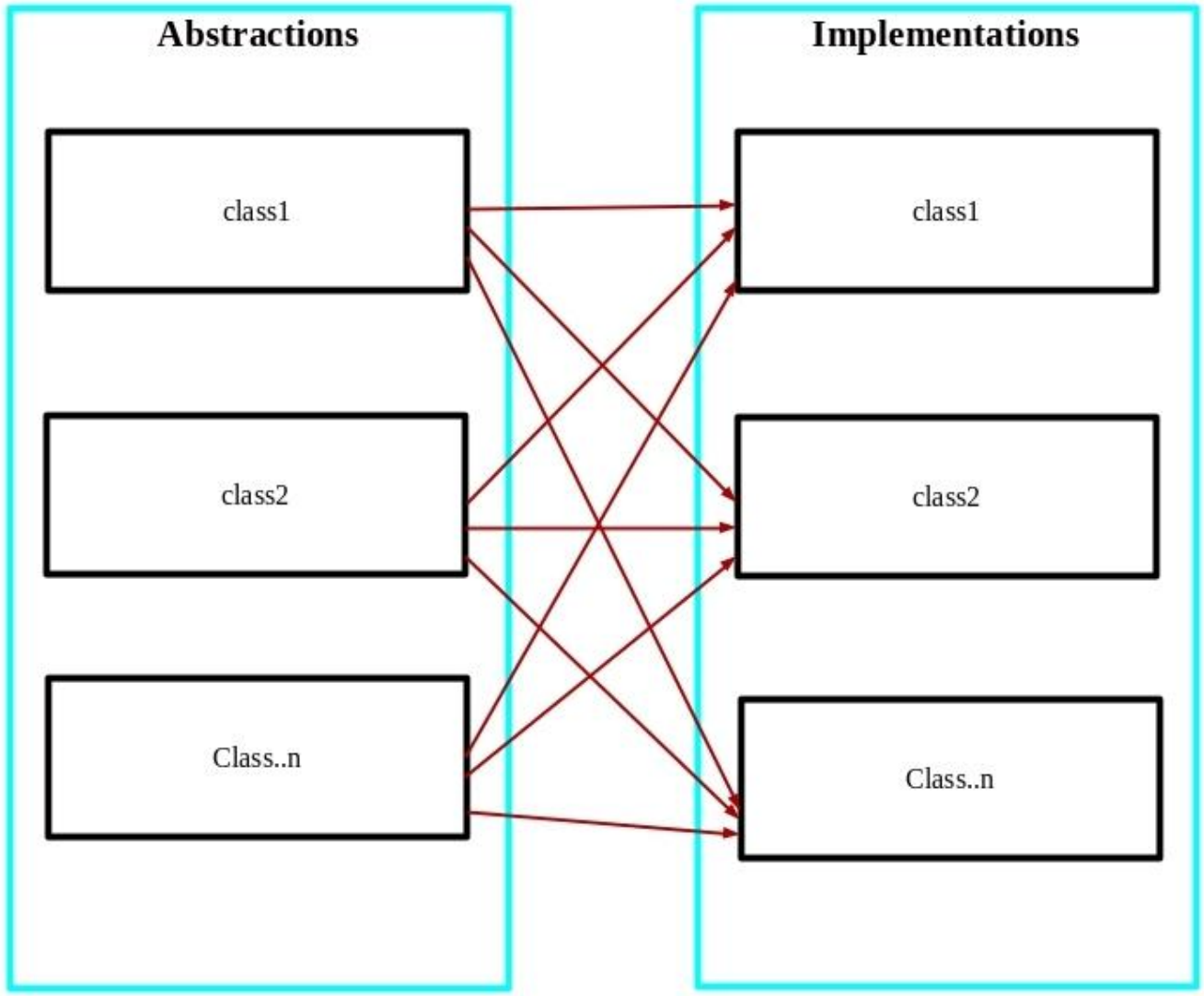

```
foreach (var geoObject in _geoObjects)
            geoObject.ShowDetails();
```

Структурные шаблоны. Мост (Bridge)

Определение **GOF:**

Отделяет абстракцию от ее реализации, чтобы они могли варьироваться независимо.

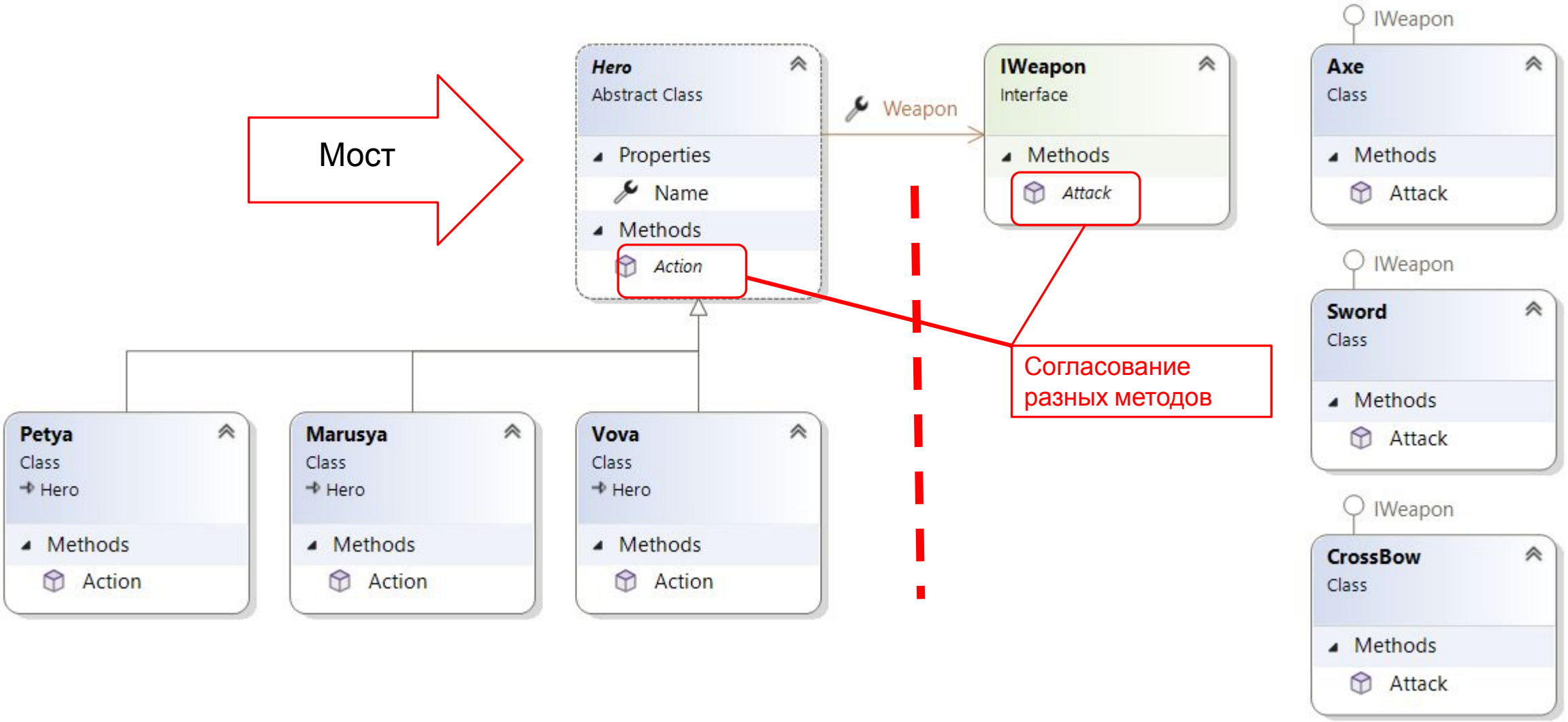
Структурные шаблоны. Мост (Bridge)



https://webmobtuts.com/backend-development/design-patterns-bridge-design-pattern-with-an-example/

Структурные шаблоны. Мост (Bridge)

Структурные шаблоны. Мост (Bridge)



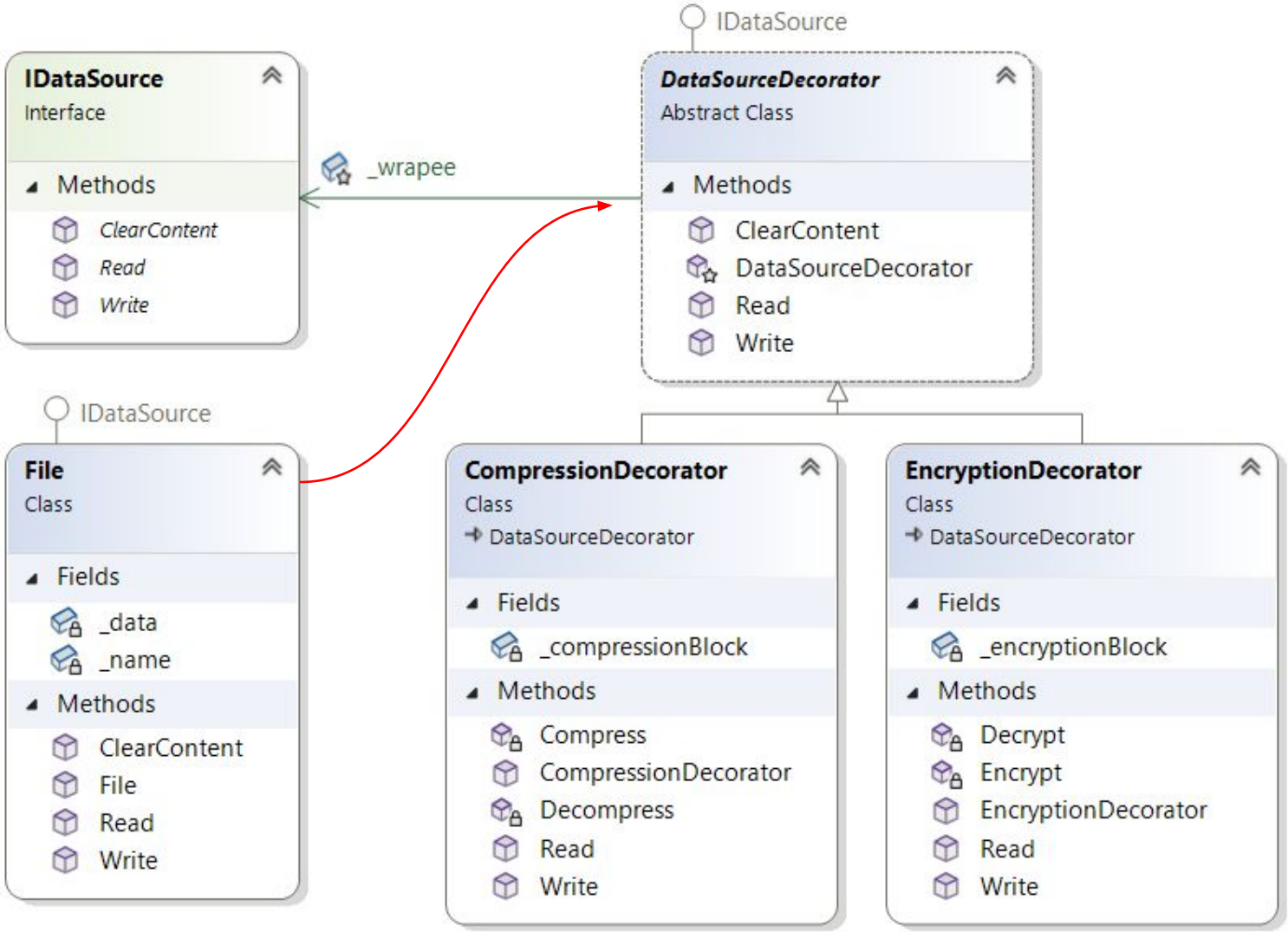
Структурные шаблоны: Декоратор (Decorator)

Определение **GOF:**

Динамическое подключение дополнительного поведения к объекту.

Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.

(Decorator)



Пример взят отсюда: https://github.com/nemanjarogic/DesignPatternsLibrary/

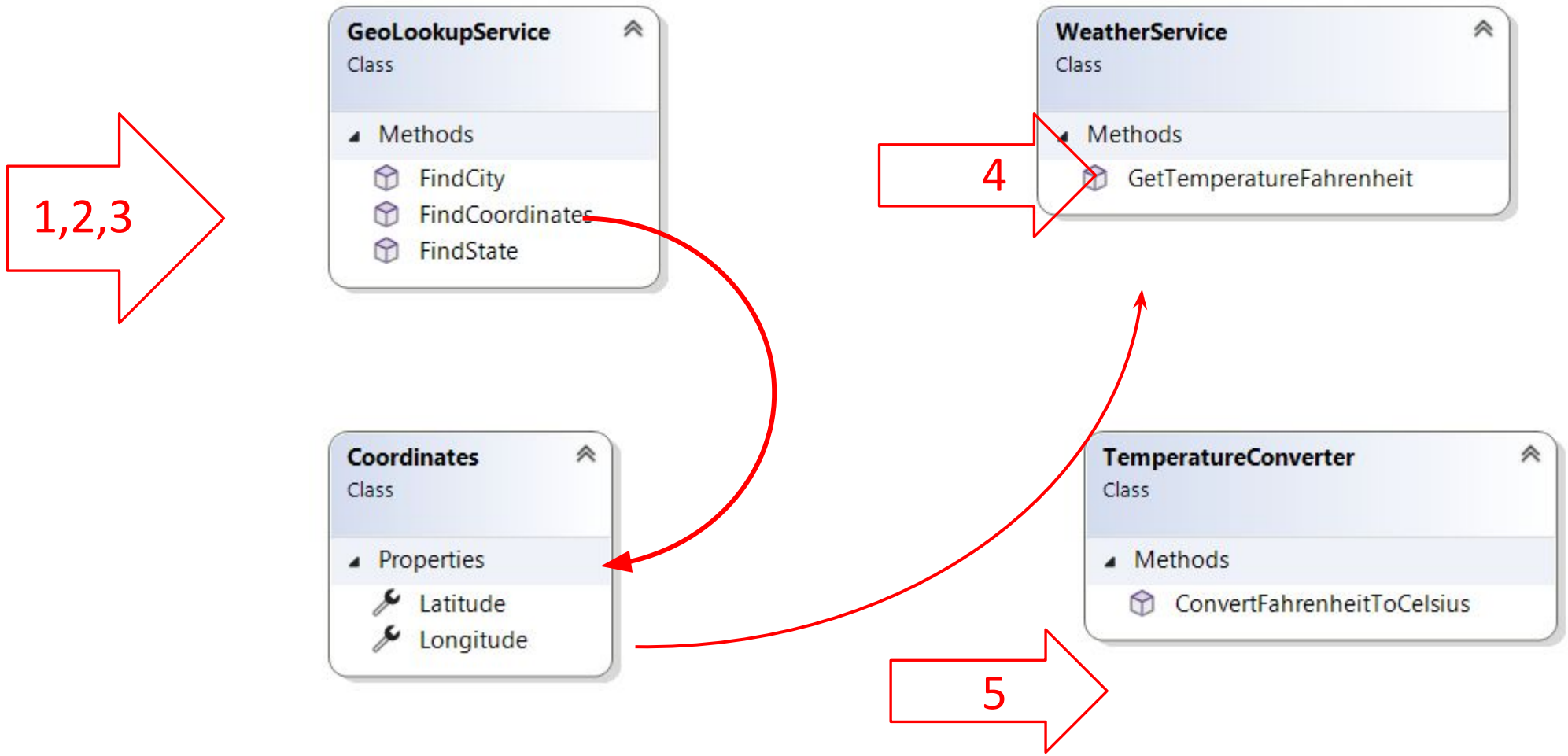
Структурные шаблоны. Фасад (Facade)

Определение **GOF:**

Предоставляет унифицированный интерфейс для набора интерфейсов в подсистеме.

Фасад определяет высокоуровневый интерфейс, упрощающий использование подсистемы.

Структурные шаблоны. Фасад (Facade)



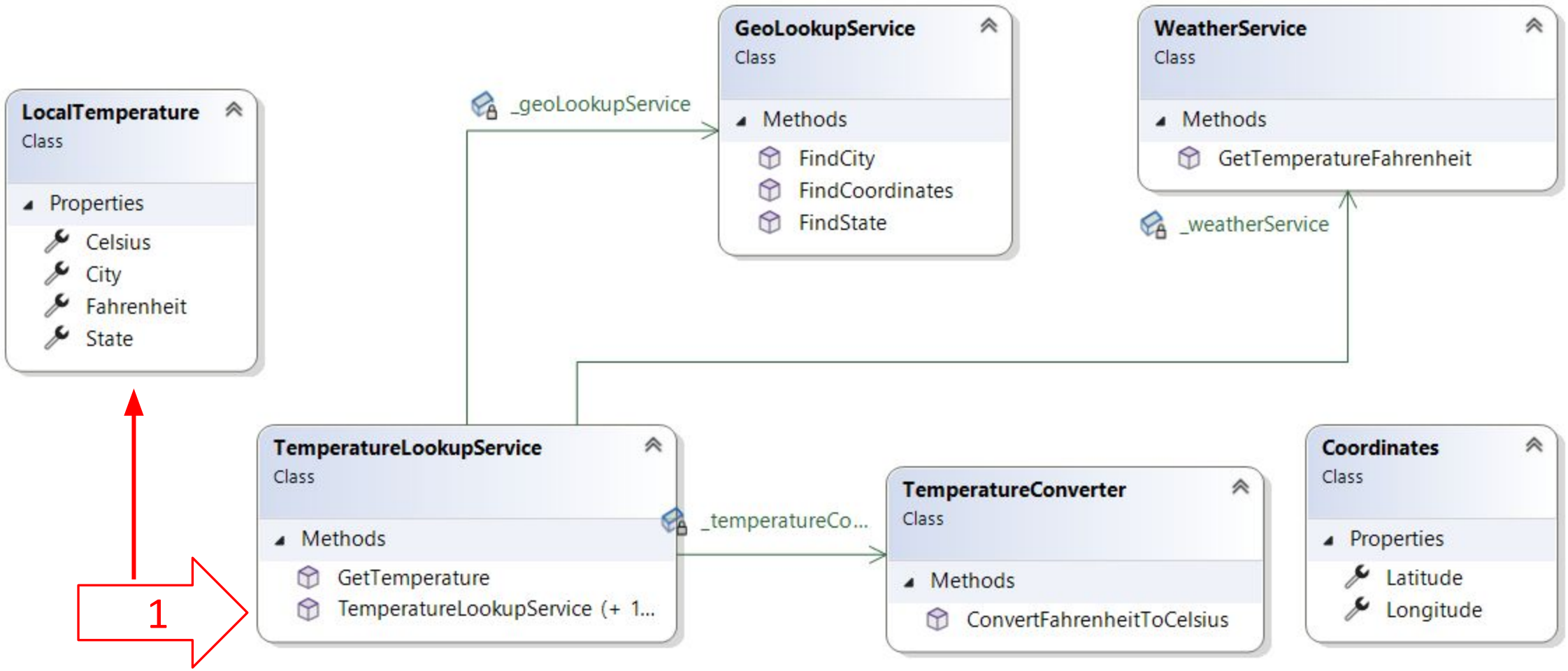
Пример взят отсюда: https://github.com/nemanjarogic/DesignPatternsLibrary/

Структурные шаблоны. Фасад (Facade)

Ог

Пр
ин

Фа
уп



Пример взят отсюда: https://github.com/nemanjarogic/DesignPatternsLibrary/

Поведенческие шаблоны (Behaviiour patterns)

ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Поведенческие шаблоны (behavioral patterns)

Шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов.

patterns)

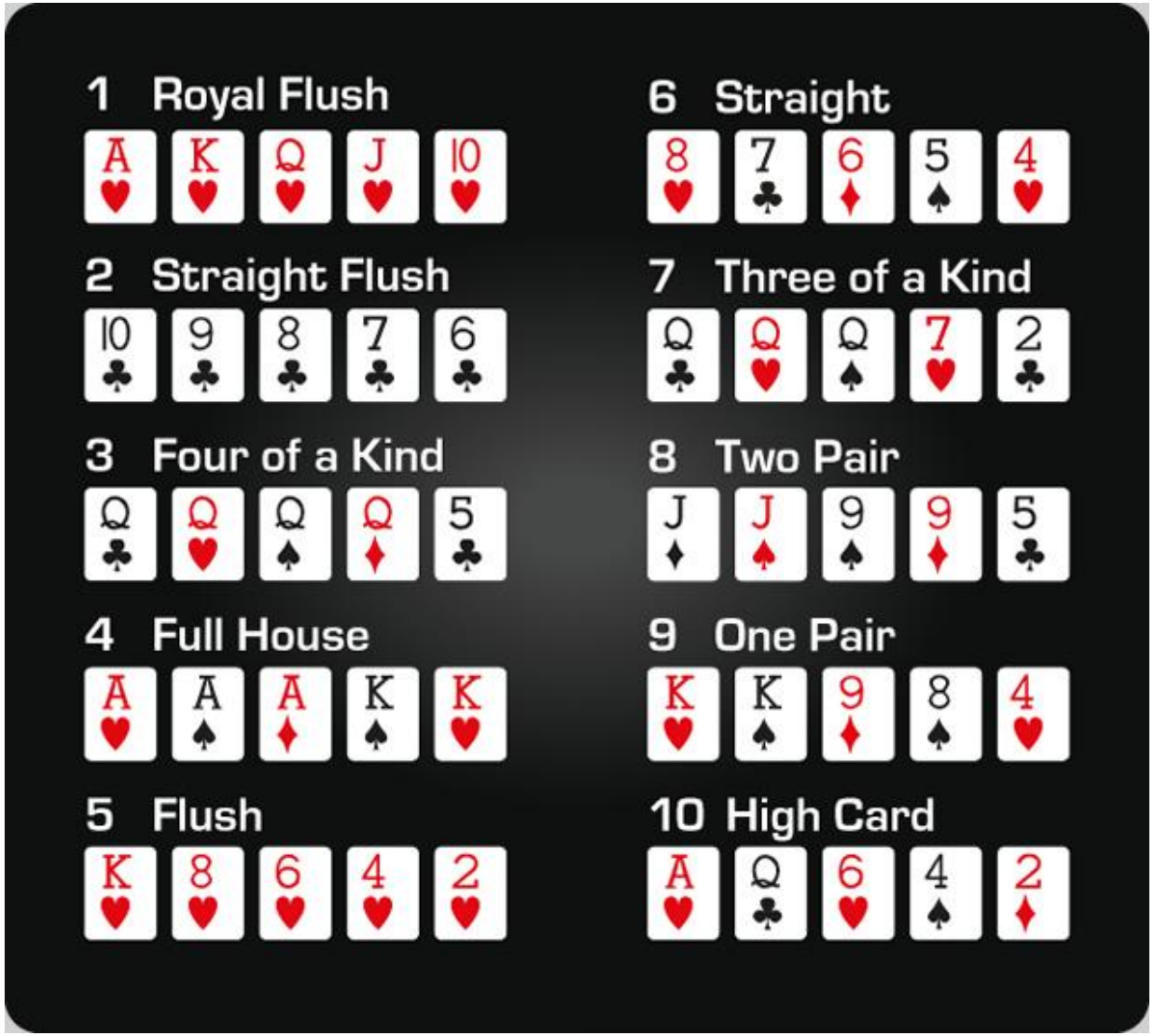
- цепочка ответственности (chain of responsibility);
- команда (action, transaction);
- итератор (cursor);
- посредник (mediator);
- наблюдатель (observer, dependents, publish-subscribe, listener);
- состояние (objects for states);
- стратегия (strategy);
- шаблонный метод (template method);
- посетитель (visitor);

Цепочка ответственности (chain of responsibility)

Определение **GOF**:

Устраняет связь отправителя запроса с его получателем, предоставляя более чем одному объекту возможность обработать запрос. Связывает принимающие объекты и передает запрос по цепочке, пока конечный объект его не обработает.

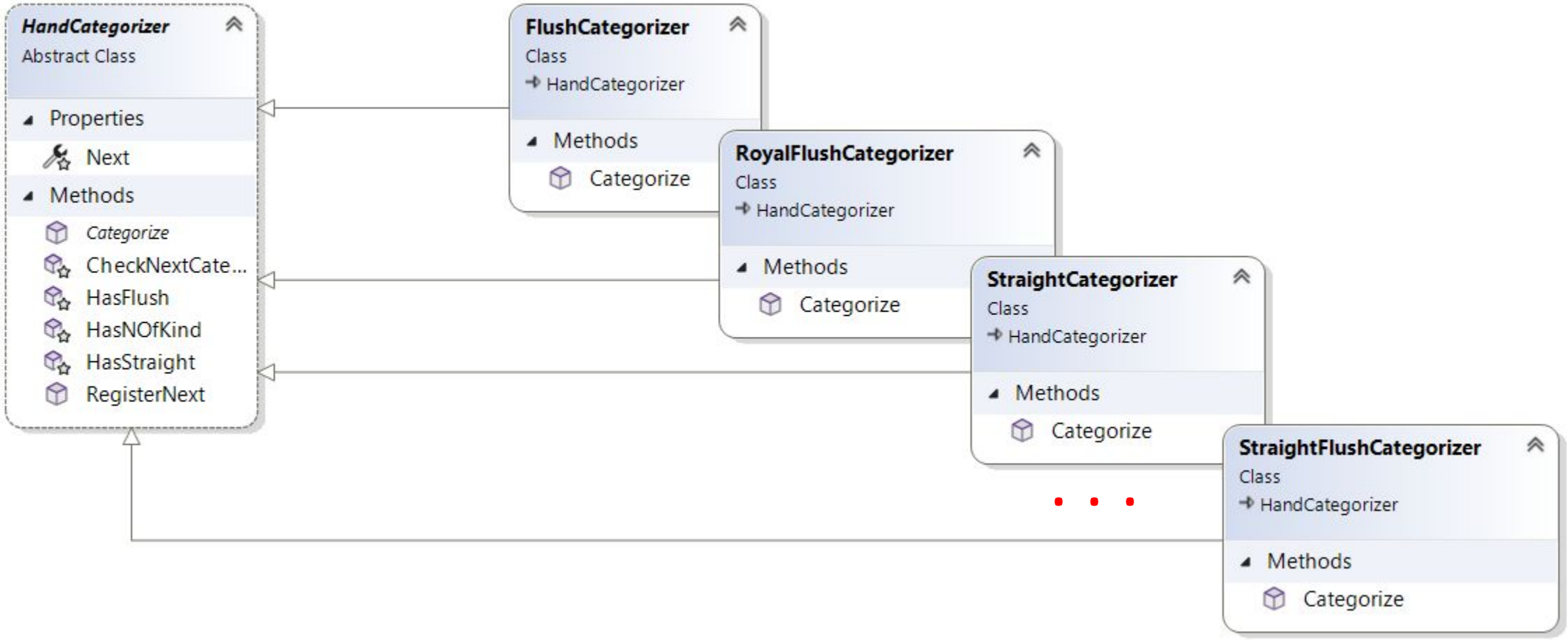
Цепочка ответственности (chain of responsibility)



Цепочка ответственности (chain of responsibility)

Ог

Ус
пр
об
пе
об



Пример взят отсюда: https://github.com/nemanjarogic/DesignPatternsLibrary/

Цепочка ответственности (chain of responsibility)

```
public class RoyalFlushCategorizer : HandCategorizer
{
    public override HandRanking Categorize(Hand hand)
    {
        if (HasFlush(hand) && HasStraight(hand) && hand.HighCard.Value == Value.Ace)
        {
            return HandRanking.RoyalFlush;
        }

        return Next?.Categorize(hand) ??
                            HandRanking.HighCard;;
    }
}
```

Цепочка ответственности (chain of responsibility)

```
Head = new RoyalFlushCategorizer();
Head.RegisterNext(new StraightFlushCategorizer())
    .RegisterNext(new FourOfAKindCategorizer())
    .RegisterNext(new FullHouseCategorizer())
    .RegisterNext(new FlushCategorizer())
    .RegisterNext(new StraightCategorizer())
    .RegisterNext(new ThreeOfAKindCategorizer())
    .RegisterNext(new TwoPairCategorizer())
    .RegisterNext(new PairCategorizer())
    .RegisterNext(new HighCardCategorizer());
```

Команда (command, action, transaction);

Определение **GOF**:

Инкапсулирует запрос как объект, тем самым позволяя вам параметризовать клиентов с различными запросами, очередями или журналами запросов, а также поддерживать отмену операции.

Команда (command, action, transaction);

В целом, здесь важны четыре термина:

- вызывающая сторона (invoker),
- Клиент (client),
- Команда (command),
- Получатель (receiver),

Команда (command, action, transaction);

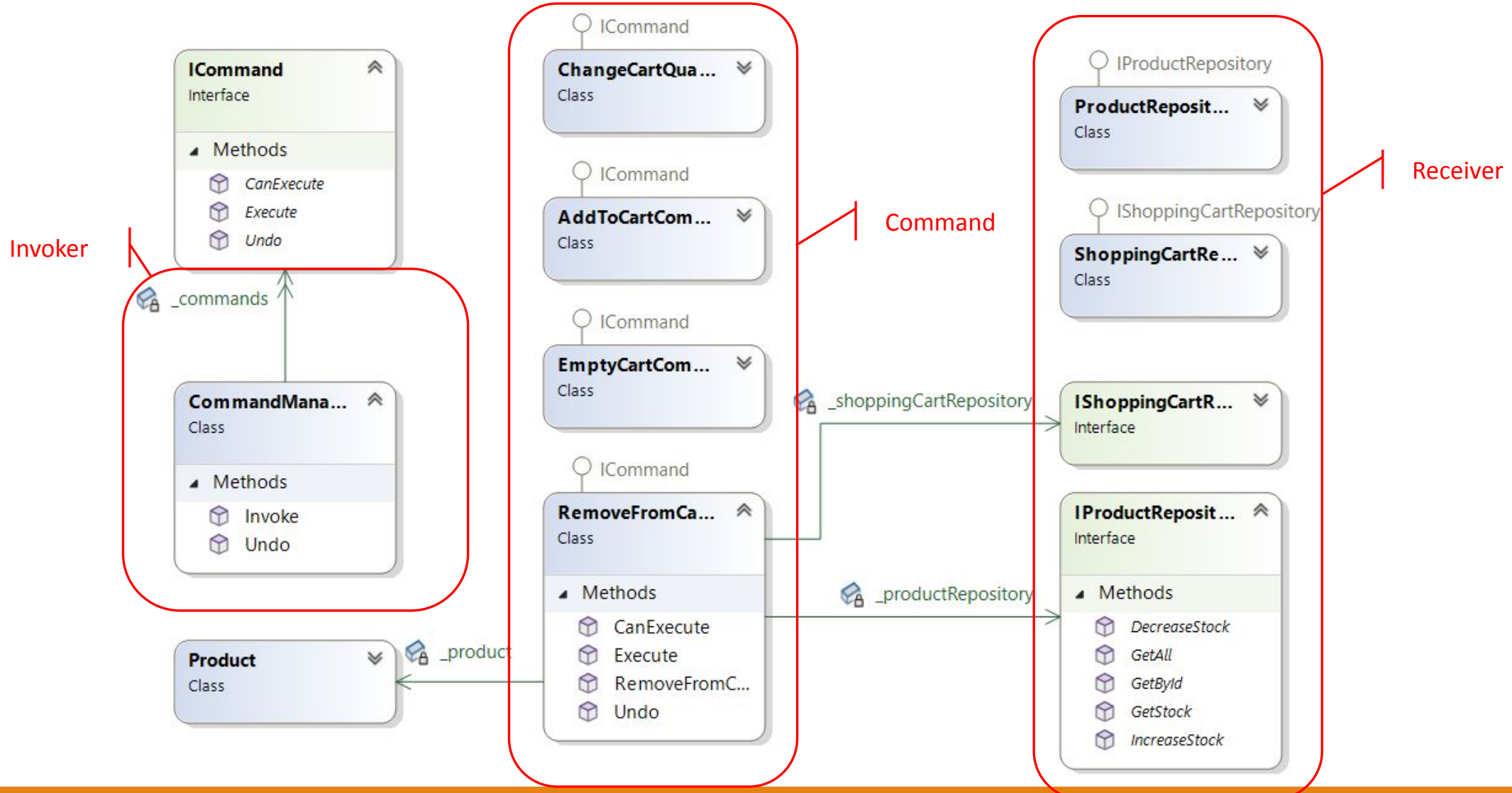
Объект **команды** состоит из действий, которые выполняет **получатель**.

Объект **команды** может вызывать метод получателя способом, характерным для этого класса получателя. Затем **получатель** начинает обработку задания (или действия).

Объект **команды** отдельно передается **вызывающему объекту** для вызова команды. **Вызывающий объект** содержит методы, с помощью которых клиент может выполнять задание, не беспокоясь о том, как целевой получатель выполняет фактическое задание.

Объект-клиент содержит **вызывающий объект** и объекты-**команды**. **Клиент** только принимает решение (т. е. какие команды выполнять), а затем передает команду вызывающему объекту для выполнения.

Команда (command, action, transaction);



Итератор (iterator, cursor)

Определение **GOF**:

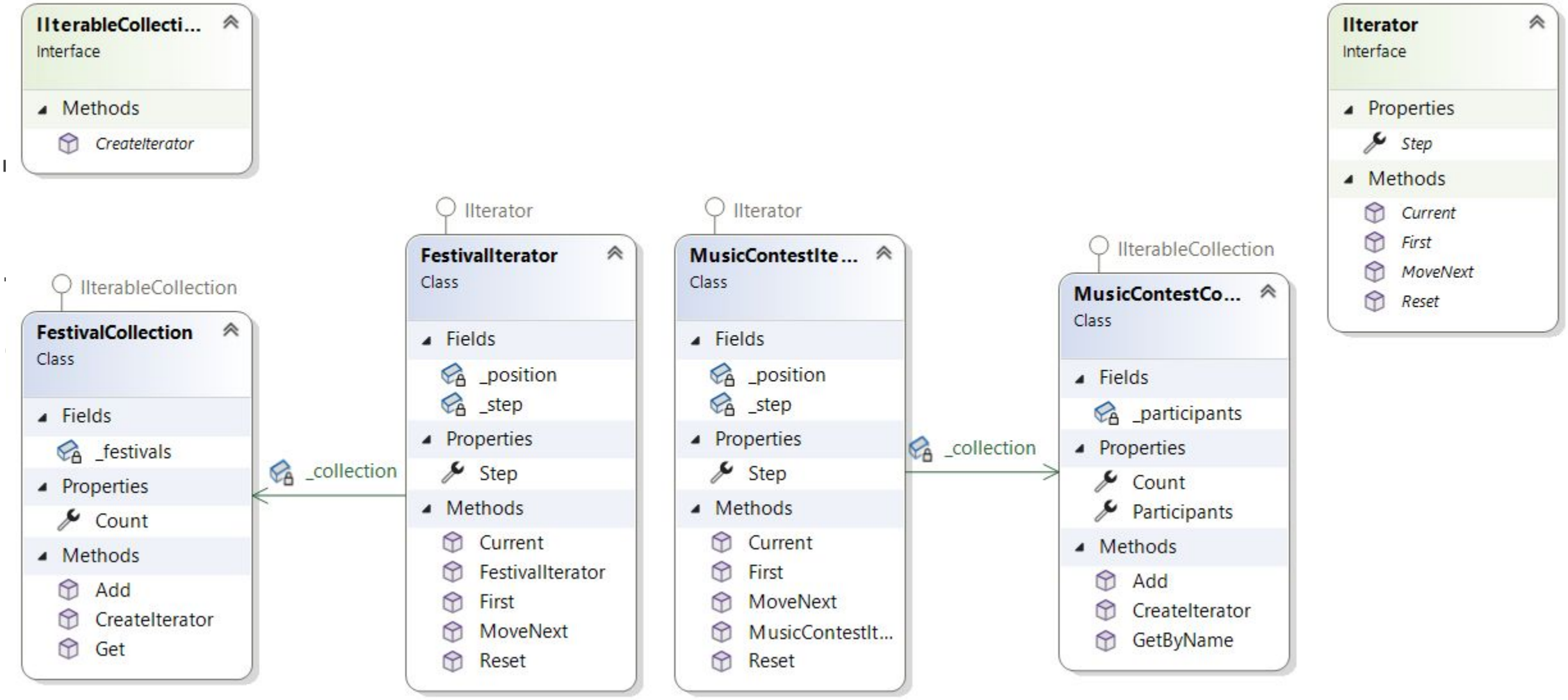
Предоставляет способ доступа к элементам агрегатного объекта последовательно, не раскрывая его базовое представление.

Итератор (iterator, cursor)

Итераторы обычно используются для обхода контейнера (или набора объектов) для доступа к его элементам, не зная, как данные хранятся внутри.

Это очень полезно, когда нужно перемещаться по **разным типам** коллекций объектов стандартным и унифицированным способом.

Итератор (iterator, cursor)



Посредник (mediator)

Определение **GOF**:

Определяет объект, который инкапсулирует взаимодействие набора объектов.

Посредник способствует слабой связи, не позволяя объектам явно ссылаться друг на друга, и позволяет независимо изменять их взаимодействие.

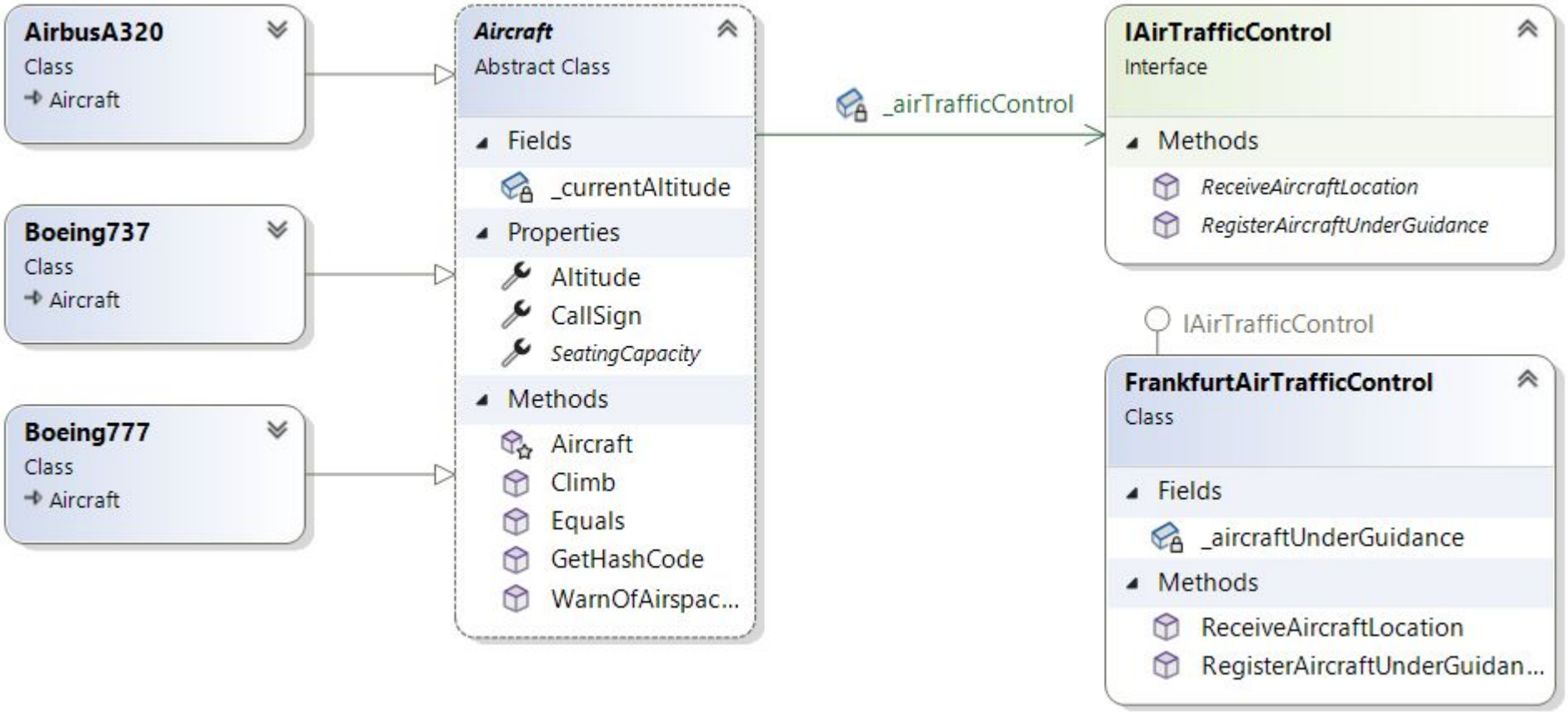
Посредник (mediator)

Пример

Пилоты разных самолетов (которые приближаются или вылетают из зоны аэродрома) общаются с вышками аэропорта.

Они не общаются явно с другими пилотами разных авиакомпаний. Они просто отправляют свой статус только на диспетчерскую вышку. Эти башни посылают сигналы, чтобы подтвердить, кто может взлететь (или приземлиться).

Посредник (mediator)



Наблюдатель (observer, dependents, publish-subscribe, listener)

Определение **GOF**:

Определяет зависимость «один ко многим» между объектами, чтобы при изменении состояния одного объекта все его зависимые объекты уведомлялись и обновлялись автоматически.

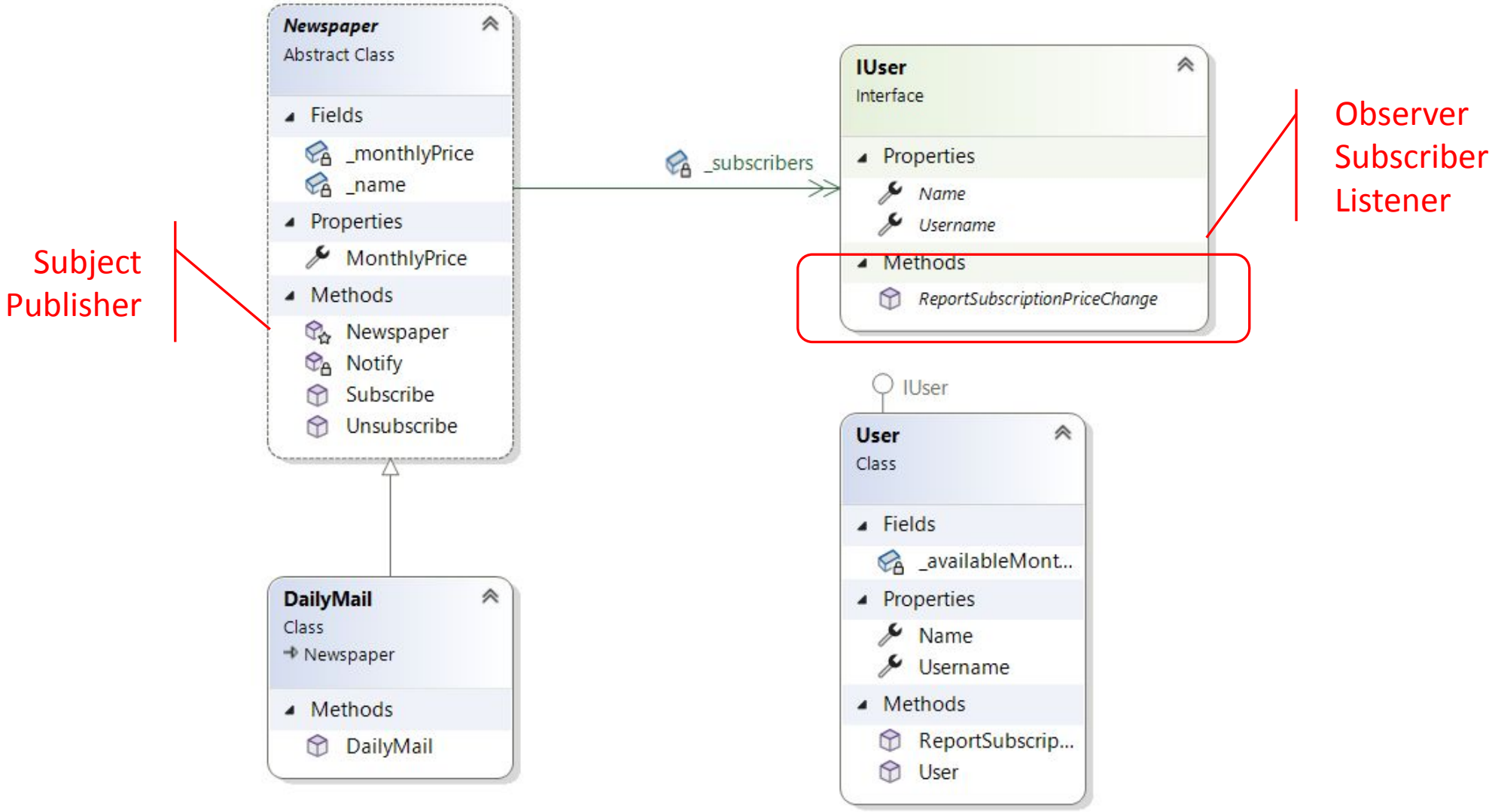
Наблюдатель (observer, dependents, publish-subscribe, listener)

Пример: знаменитость, у которой много подписчиков в социальных сетях.

Каждый из подписчиков хочет получать все последние обновления от своей любимой знаменитости. Когда они теряют интерес, они просто не следуют за той знаменитостью.

В данном случае каждый из этих подписчиков или последователей является наблюдателем, а знаменитость - субъектом.

Наблюдатель (observer, dependents, publish-subscribe, listener)



Состояние (objects for states)

Определение **GOF**:

Разрешает объекту изменять свое поведение при изменении его внутреннего состояния.

Состояние (objects for states)

В этом шаблоне выявляются возможные состояниях вашего приложения (объекта) и соответствующим образом разделяется код.

В идеале каждое из состояний независимо от других состояний.

Вы отслеживаете эти состояния, и ваш код реагирует в соответствии с поведением текущего состояния.

Состояние (objects for states)

Например, если телевизор включен (состояние «Power On»), и вы нажмете кнопку отключения звука (Mute) на пульте дистанционного управления телевизора, на телевизоре звук выключится.

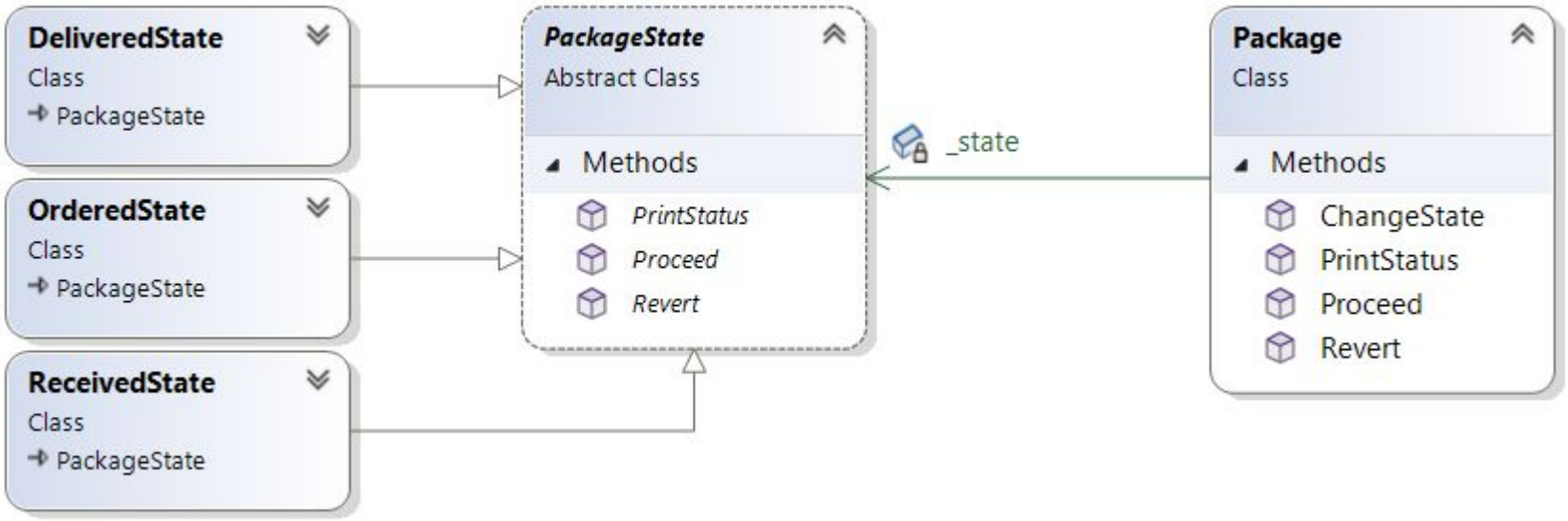
Но, если телевизор уже находится в выключенном режиме (состояние «Power Off»), то команда Mute не выполняется.

Состояние (objects for states)

Пример: товар на складе.

- Состояние: получен - можно заказать товар
- Состояние : заказан –можно отгрузить товар или отменить заказ
- Состояние : отгружен – можно распечатать отчет

Состояние (objects for states)



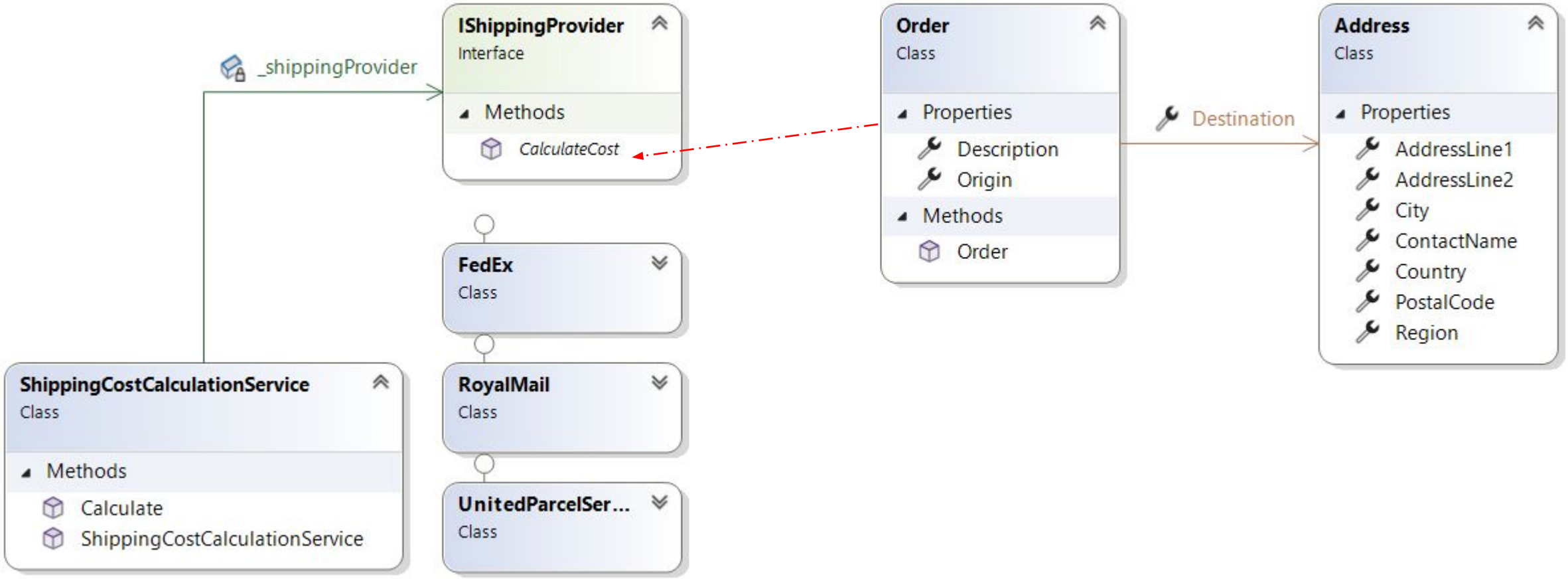
Стратегия (strategy)

Определение **GOF**:

Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

Стратегия позволяет алгоритму изменяться независимо от клиентов, которые его используют.

Стратегия (strategy)



Шаблонный метод (template method)

Определение **GOF**:

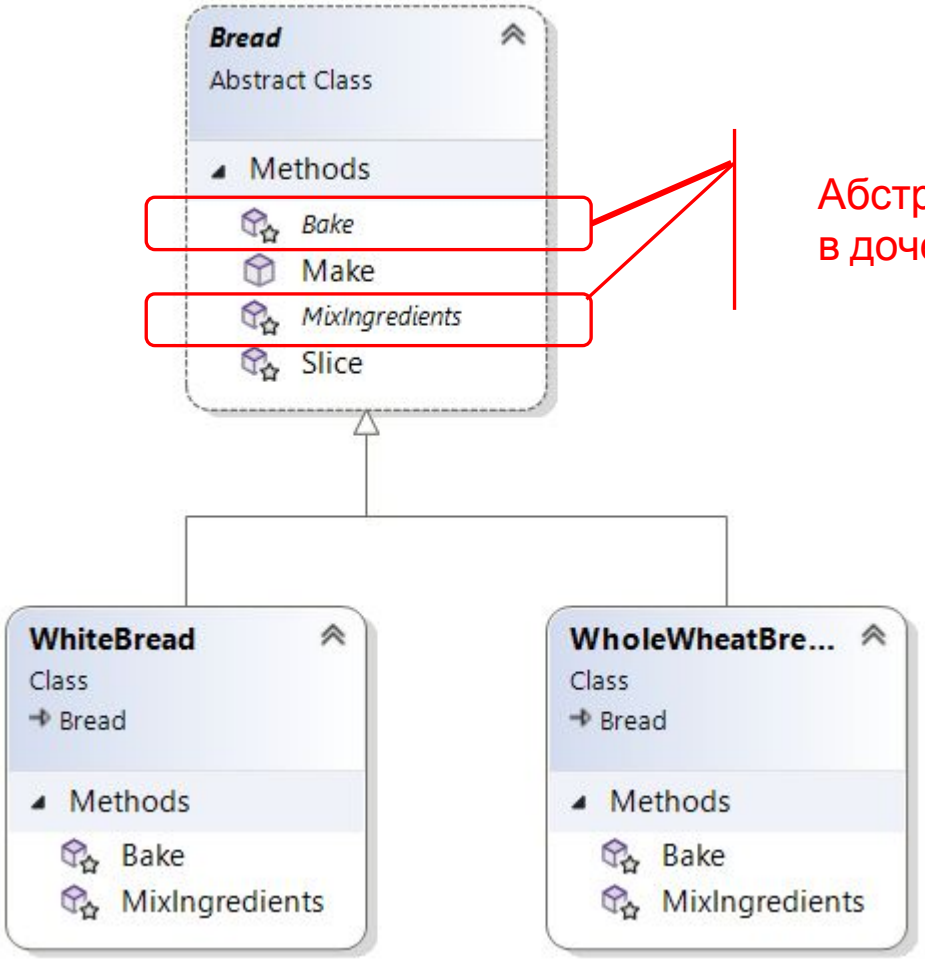
Определяет скелет алгоритма в методе, откладывая некоторые шаги до подклассов. Шаблонный метод позволяет подклассам переопределять определенные шаги алгоритма без изменения структуры алгоритма.

Шаблонный метод (template method)

При заказе пиццы, можно использовать базовый механизм для приготовления пиццы, но также может выбрать дополнительные ингредиенты (начинки).

Таким образом, непосредственно перед доставкой пиццы шеф-повар может включить эти варианты.

Шаблонный метод (template method)



Абстрактные методы для реализации в дочерних классах

Посетитель (visitor)

Определение **GOF**:

Представляет операцию, которая должна выполняться над элементами структуры объекта.

Посетитель позволяет определить новую операцию без изменения классов элементов, над которыми она работает.

Посетитель (visitor)

В этом шаблоне вы отделяете алгоритм от структуры объекта. Таким образом, вы можете добавлять новые операции над объектами, не изменяя их существующую архитектуру.

Этот шаблон поддерживает S**O**LID принцип Open/Close (в котором говорится, что расширение разрешено, но модификации запрещены для таких сущностей, как класс, функция и т. д.).

Посетитель (visitor)