## [Структурные шаблоны](https://ru.wikipedia.org/wiki/Структурные_шаблоны_проектирования) ([Structural](https://en.wikipedia.org/wiki/Structural_pattern)) определяют различные сложные структуры, которые изменяют [интерфейс](https://ru.wikipedia.org/wiki/Интерфейс_(объектно-ориентированное_программирование)) уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

## Компоновщик (Composite)

Его задача – агрегировать объекты, обладающие общим интерфейсом, и обеспечить выполнение операций этого интерфейса над всеми агрегируемыми объектами.

Дизайн шаблона Компоновщика показан на рис. 6. То, что и компоновщик, и отдельный объект реализуют один и тот же интерфейс, удобно с точки зрения клиента, который не видит разницы между ними в плане функциональности. Обычно компоновщика снабжают методами добавления, удаления, редактирования и поиска отдельных агрегируемых компонент.

Описание: Composite.emf

## Приспособленец (Flyweight)

Шаблон Приспособленец предлагает эффективный способ разделить общую информацию, находящуюся в небольших объектах. Суть шаблона в том, чтобы разделить состояние некоторого объекта на состояния трех типов. *Внутреннее состояние* (intrinsic state) принадлежит самому объекту. Тип Flyweight реализует интерфейс IFlyweight, который определяет операции, в которых заинтересована остальная часть системы. Клиент владеет *общим* (неразделяемым) *состоянием* (unshared state), а также коллекцией приспособленцев, которых производит класс-фабрика (FlyweightFactory). Наконец, *внешнее состояние* (extrinsic state) не появляется в системе как таковое. Если оно понадобится, то будет вычислено уже во время выполнения программы для каждого внутреннего состояния.

## Адаптер (Adapter)

Шаблон Адаптер предназначен для обеспечения совместной работы классов, которые изначально не были предназначены для совместного использования. Такие ситуации часто возникают, когда идет работа с существующими библиотеками кода. Нередко их интерфейс не отвечает требованиям клиента, но изменить библиотеку возможности нет. Возникает задача адаптации библиотеки для клиента.

Описание: Adapter.emf

В адаптере четко видно преимущество программирования согласно интерфейсам. Client работает в соответствии с требованиями своей предметной области, эти требования отражены в целевом интерфейсе ITarget (интерфейс, в котором заинтересован клиент). Адаптируемый класс Adaptee обладает требуемой функциональностью, но неподходящим интерфейсом. Adapter реализует интерфейс ITarget и перенаправляет вызовы от Client к Adaptee, изменяя при необходимости параметры и возвращаемые значения.

Особенность адаптеров в том, что они могут добавлять дополнительное поведение к тому поведению, что специфицируется в ITarget и в Adaptee. Другими словами, адаптеры могут быть *прозрачными* для клиента и *непрозрачными*. В примере кода показан последний случай, где Adapter добавляет "Rough:". Эта добавка показывает, что вызов Request() был адаптирован (изменен) перед тем, как был вызван метод SpecificRequest().

## Фасад (Façade)

Назначение шаблона Фасад состоит в предоставлении различных высокоуровневых представлений подсистем, детали реализации которых скрыты от клиента. В общем случае набор операций, желаемый для клиента, может формироваться в виде набора из разных частей подсистемы. Сокрытие деталей – это ключевая концепция программирования.

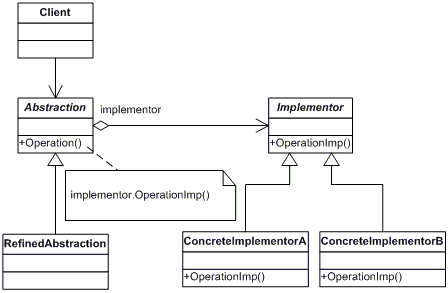
Из возможных вариантов отметим *прозрачные фасады* (в этом случае компоненты подсистемы могут быть доступны и через фасад, и в обход его) и *статические фасады* (фасад является статическим классом – скрываемые объекты не агрегируются, а создаются в методах фасада по необходимости).

Описание: facade.emf

***Шаблон мост***

[Структурный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%8B%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), используемый в [проектировании программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) чтобы «разделять [абстракцию](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и [реализацию](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) так, чтобы они могли изменяться независимо». Шаблон bridge использует [инкапсуляцию](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [агрегирование](https://ru.wikipedia.org/wiki/%D0%90%D0%B3%D1%80%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и может использовать наследование для того, чтобы разделить ответственность между классами.

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.

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Bridgeuml.gif)

Когда абстракция и реализация разделены, они могут изменяться независимо. Другими словами, при реализации через паттерн мост, изменение структуры интерфейса не мешает изменению структуры реализации. 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.

**Декоратор**

[Структурный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%8B%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), предназначенный для динамического подключения дополнительного поведения к [объекту](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Шаблон Декоратор предоставляет гибкую альтернативу практике создания [подклассов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) с целью расширения функциональности.

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. Декоратор предусматривает расширение функциональности объекта без определения подклассов. 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](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81). Абстрактный класс *Component* определяет интерфейс для использования всех этих классов.

1. Добавляемая функциональность реализуется в небольших объектах. Преимущество состоит в возможности динамически добавлять эту функциональность до или после основной функциональности объекта ConcreteComponent.  
   2. Позволяет избегать перегрузки функциональными классами на верхних уровнях иерархии  
   3. Декоратор и его компоненты не являются идентичными

Создается абстрактный класс, представляющий как исходный класс, так и новые, добавляемые в класс функции. В классах-декораторах новые функции вызываются в требуемой последовательности — до или после вызова последующего объекта.

При желании остаётся возможность использовать исходный класс (без расширения функциональности), если на его объект сохранилась ссылка.

**Заместитель**

### [Структурный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%8B%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), который предоставляет[объект](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию [контейнера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))).

### *Проблема*

Необходимо управлять доступом к объекту так, чтобы не создавать громоздкие объекты «по требованию».

### *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](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)«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.
* **Удалённый заместитель** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *remote proxies*): обеспечивает связь с «Субъектом», который находится в другом адресном пространстве или на удалённой машине. Также может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному «Субъекту»,
* **Виртуальный заместитель** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *virtual proxies*): обеспечивает создание реального «Субъекта» только тогда, когда он действительно понадобится. Также 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.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *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**](https://ru.wikipedia.org/wiki/Firewall)**прокси**: защищает «Субъект» от опасных клиентов (или наоборот).
* **Синхронизирующий прокси**: производит синхронизированный контроль доступа к «Субъекту» в асинхронной [многопоточной](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) среде.
* **«Умная» ссылка** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *smart reference proxy*): производит дополнительные действия, когда на «Субъект» создается ссылка, например, рассчитывает количество активных [ссылок](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) на «Субъект».

