Декоратор (шаблон проектирования)

**Декоратор** ([англ.](http://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) *Decorator*) — [структурный](http://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](http://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](http://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](http://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.

Основные характеристики

**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](http://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.

**Следствия**

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

**Реализация**

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.

Замечания и комментарии

* Хотя объект-декоратор может добавлять свою функциональность до или после функциональности основного объекта, цепочка создаваемых объектов всегда должна заканчиваться объектом класса ConcreteComponent.
* Базовые классы языка [Java](http://ru.wikipedia.org/wiki/Java) широко используют шаблон Декоратор для организации обработки операций ввода-вывода.
* И декоратор, и адаптер являются обертками вокруг объекта — хранят в себе ссылку на оборачиваемый объект и часто передают в него вызовы методов. Отличие декоратора от адаптера в том, что адаптер имеет внешний интерфейс, отличный от интерфейса оборачиваемого объекта, и используется именно для стыковки разных интерфейсов. Декоратор же имеет точно такой же интерфейс, и используется для добавления функциональности.
* Для расширения функциональности класса возможно использовать как декораторы, так и стратегии. Декораторы оборачивают объект снаружи, стратегии же вставляются в него внутрь по неким интерфейсам. Недостаток стратегии: класс должен быть спроектирован с возможностью вставления стратегий, декоратор же не требует такой поддержки. Недостаток декоратора: он оборачивает ровно тот же интерфейс, что предназначен для внешнего мира, что вызывает смешение публичного интерфейса и интерфейса кастомизации, которое не всегда желательно.

Применение шаблона

Драйверы-фильтры в ядре [Windows](http://ru.wikipedia.org/wiki/Windows) (архитектура [WDM](http://ru.wikipedia.org/wiki/WDM)) представляют собой декораторы. Несмотря на то, что WDM реализована на не-объектном языке Си, в ней четко прослеживаются паттерны проектирования — декоратор, цепочка ответственности, и команда (объект IRP).

Архитектура [COM](http://ru.wikipedia.org/wiki/COM) не поддерживает наследование реализаций, вместо него предлагается использовать декораторы (в данной архитектуре это называется «агрегация»). При этом архитектура решает (с помощью механизма pUnkOuter) проблему object identity, возникающую при использовании декораторов — identity агрегата есть identity его самого внешнего декоратора.