# Паттерн Фасад (Facade)

**Назначение:** предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистемы.

## Мотивация

Библиотеки классов обычно предназначены для решения целого спектра задач. С помощью ADO.NET можно выполнять простые команды на стороне базе данных, а можно использовать транзакции и выходные (output) параметры хранимых процедур. Наличие сложных сценариев делает библиотеку полезной, но это же может усложнить решение с ее помощью простых задач. Это приводит к появлению всевозможных оболочек даже над стандартными библиотеками, которые упрощают решение простых задач, или сценариев, специфичных для конкретного приложения.

Такие оболочки являются фасадами, которые скрывают исходную сложность библиотеки или модуля, за более простым и, возможно, специфичным для приложения интерфейсом.

Давайте рассмотрим конкретный пример небольшой оболочки, которая упрощает работу с SQL Server-ом (рис 2.1):

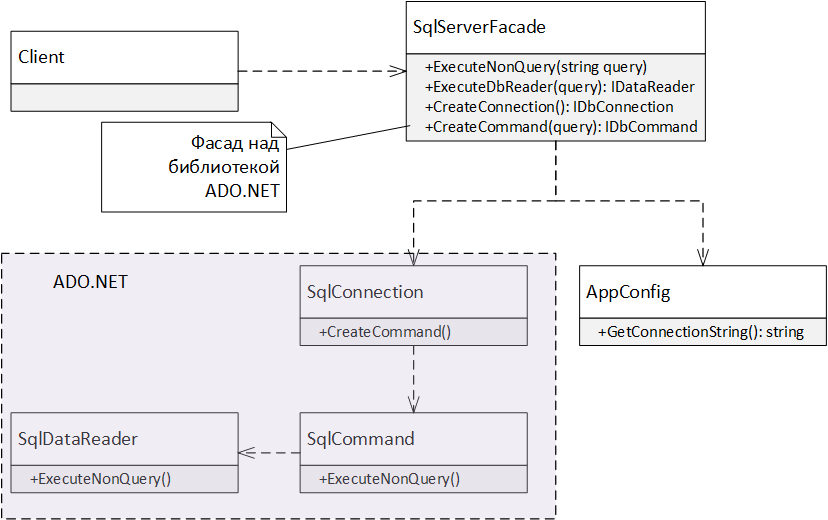


Рисунок 2.1 - Фасад для работы с SQL Server

В большинстве приложений используется лишь часть функциональности сложной библиотеки, и фасад позволяет сделать более простой интерфейс, максимально подходящий для специфических сценариев. Класс SqlServerFacade выступает в роли фасада, который упрощает решение приложением типовых задач взаимодействия с базой данных: прячет логику получения строки подключения, а также ряд второстепенных деталей библиотеки ADO.NET.

## Классическая диаграмма классов паттерна Фасад

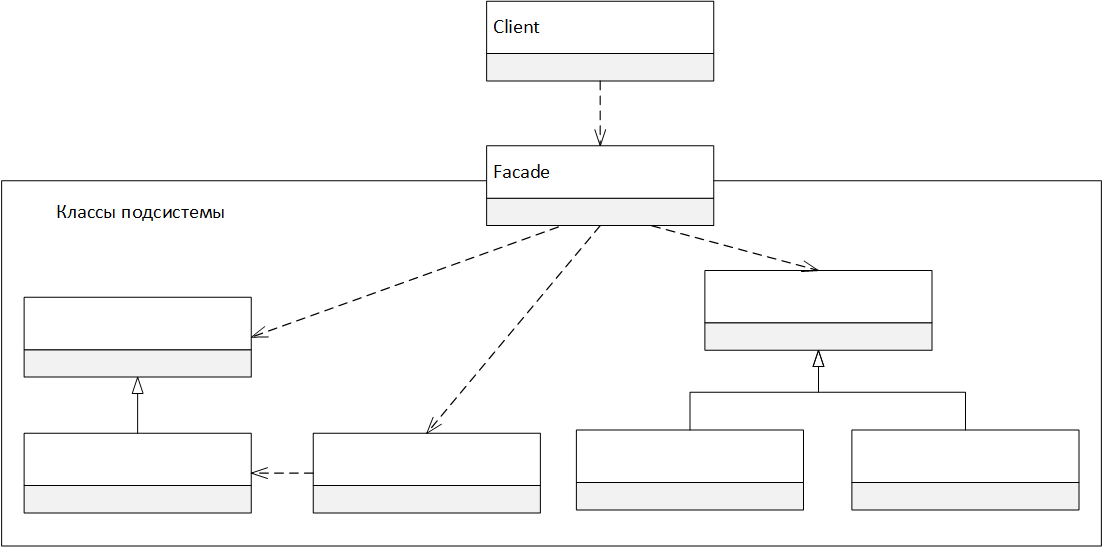


Рисунок 2.2 - Классическая диаграмма классов паттерна Фасад

**Участники**

* Facade (SqlServerFacade) - фасадный класс, который прячет детали реализации подсистемы от клиентов
* Client - клиент фасада, который работает с фасадом, а не с классами подсистемы.

## Обсуждение паттерна Фасад

Очень многие библиотеки или подсистемы приложений содержат встроенные фасады, которые являются высокоуровневыми классами, предназначенными для решения типовых операций. Фасады делают базовые сценарии простыми, а сложные сценарии - возможными. Если клиенту нужна лишь базовая функциональность - достаточно воспользоваться фасадом, если же его функциональности недостаточно, то можно использовать более низкоуровневые классы модуля или библиотеки напрямую.

### Инкапсуляция стороннего кода

Использование фасадов не только упрощает использование библиотек или сторонних компонентов, но и решает ряд насущных проблем.

* **Повторное использование кода и лучших практик**. Многие библиотеки достаточно сложные, что требует определенных навыков для их корректного использования. Инкапсуляция работы с ними в одном месте позволяет корректно их использовать всеми разработчиками, независимо от их опыта.
* **Переход на новую версию библиотеки**. При выходе новой версии библиотеки достаточно будет протестировать лишь фасад, чтобы принять решение, стоит на нее переходить или нет.
* **Переход с одной библиотеки на другую**. Благодаря фасаду, приложение не так сильно завязано на библиотеку, так что переход на другую библиотеку потребует лишь создание еще одного фасада. А использование адаптера (\*) сделает этот переход менее болезненным.

(\*) Сноска: паттерн Адаптер был рассмотрен в предыдущей главе.

### Повышение уровня абстракции

Фасад повышает уровень абстракции и упрощает решение задач, специфичных для текущего приложения. Фасад скрывает низкоуровневые детали, но может предоставлять интерфейс, специфичный для конкретного приложения за счет использования в качестве параметров доменных объектов.

Например, в случае фасада для работы с Sql Server, класс SqlServerProvider может принимать класс Configuration из которого он будет получать строку подключения и другие параметры.

public static IDbCommand CreateCommand(Configuration config) {...}

Листинг 2.1 - Пример использования фасадом доменного типа

## Применимость

Я предпочитаю использовать фасады для работы с большинством сторонних библиотек или подсистем. Это уменьшает связанность (coupling) системы с внешними зависимостями, позволяет лучше понять сторонний код, позволяет контролировать качество новых версий, а также избавляет код приложения от излишних низкоуровневых деталей.

Фасад **не нужно применять**, когда в повышении уровня абстракции нет никакого смысла. В большинстве случаев нет смысла в фасаде для библиотеки логирования, достаточно выбрать одно из решений (log4net, NLog, .NET Traces) и использовать его в коде приложения. Фасады бесполезны, когда они обладают громоздким интерфейсом и пользоваться ими сложнее, чем исходной библиотекой.

## Примеры в .NET Framework

* XmlSerializer прячет сложность сериализации, генерацию временной сборки и многие другие низкоуровневые подробности.
* Класс ThreadPool.QueueUserWorkItem является фасадом для работы с пулом потоков.
* Класс Parallel из TPL является фасадом для упрощения параллельного программирования. Он скрывает низкоуровневые детали работы планировщиков, стратегии партиционирования (partitioning) и другие подробности по созданию и управлению объектами Task<T>.
* Класс System.Runtime.CompilerServices.RuntimeHelpers является фасадом с низкоуровневыми служебными операциями, такими как вызов статического конструктора типа и т.п.
* Класс System.Console является фасадом для работы с консолью ввода-вывода.