Паттерн Adapter

Название и классификация паттерна
Адаптер - паттерн, структурирующий классы и объекты.

Назначение
Преобразует интерфейс одного класса в интерфейс другого, который ожидают клиенты. Адаптер обеспечивает совместную работу классов с несовместимыми интерфейсами, которая без него была бы невозможна.

Мотивация
Иногда класс из инструментальной библиотеки, спроектированный для повторного использования, не удается использовать только потому, что его интерфейс не соответствует тому, который нужен конкретному приложению. Рассмотрим, например, графический редактор, благодаря которому пользователи могут рисовать на экране графические элементы (линии, многоугольники, текст и т.д.) и организовывать их в виде картинок и диаграмм. Основной абстракцией графического редактора является графический объект, который имеет изменяемую форму и изображает сам себя. Интерфейс графических объектов определен абстрактным классом Shape. Редактор определяет подкласс класса Shape для каждого вида графических объектов: LineShape для прямых, PolygonShape для многоугольников и т. д. Классы для элементарных геометрических фигур, например LineShape и PolygonShape, реализовать сравнительно просто, поскольку заложенные в них возможности рисования и редактирования крайне ограничены. Но подкласс Text Shape, умеющий отображать и редактировать текст, уже значительно сложнее, поскольку даже для простейших операций редактирования текста нужно нетривиальным образом обновлять экран и управлять буферами. В то же время, возможно, существует уже готовая библиотека для разработки пользовательских интерфейсов, которая предоставляет развитый класс Text View, позволяющий отображать и редактировать текст. В идеале мы хотели бы повторно использовать Text View для реализации Text Shape, но библиотека разрабатывалась без учета классов Shape, поэтому заставить объекты Text View и Shape работать совместно не удается. Так каким же образом существующие и независимо разработанные классы вроде Text View могут работать в приложении, которое спроектировано под другой, несовместимый интерфейс? Можно было бы так изменить интерфейс класса Text View, чтобы он соответствовал интерфейсу Shape, только для этого нужен исходный код. Но даже если он доступен, то вряд ли разумно изменять Text View; библиотека не должна приспосабливаться к интерфейсам каждого конкретного приложения. Вместо этого мы могли бы определить класс Text Shape так, что он будет адаптировать интерфейс Text View к интерфейсу Shape. Это допустимо сделать двумя способами: наследуя интерфейс от Shape, а реализацию от Text View; включив экземпляр Text View в Text Shape и реализовав Text Shape в терминах интерфейса Text View. Два данных подхода соответствуют вариантам паттерна адаптер в его классовой и объектной ипостасях. Класс Text Shape мы будем называть адаптером.
На этой диаграмме показан адаптер объекта. Видно, как запрос BoundingBox, объявленный в классе Shape, преобразуется в запрос Get Extent, определенный Паттерн Adapter в классе Text View. Поскольку класс Text Shape адаптирует TextView к интерфейсу Shape, графический редактор может воспользоваться классом TextView, хотя тот и имеет несовместимый интерфейс. Часто адаптер отвечает за функциональность, которую не может предоставить адаптируемый класс. На диаграмме показано, как адаптер выполняет такого рода функции. У пользователя должна быть возможность перемещать любой объект класса Shape в другое место, но в классе TextView такая операция не предусмотрена. Text Shape может добавить недостающую функциональность, самостоятельно реализовав операцию CreateManipulator класса Shape, которая возвращает экземпляр подходящего подкласса Manipulator. Manipulator - это абстрактный класс объектов, которым известно, как анимировать Shape в ответ на такие действия пользователя, как перетаскивание фигуры в другое место. У класса Manipulator имеют-

ся подклассы для различных фигур. Например, TextManipulator - подкласс для Text Shape. Возвращая экземпляр TextManipulator, объект класса TextShape добавляет новую функциональность, которой в классе TextView нет, а классу Shape требуется.

Применимость
Применяйте паттерн адаптер, когда:

- хотите использовать существующий класс, но его интерфейс не соответствует вашим потребностям;
- собираетесь создать повторно используемый класс, который должен взаимодействовать с заранее неизвестными или не связанными с ним классами, имеющими несовместимые интерфейсы;
- (только для адаптера объектов!) нужно использовать несколько существующих подклассов, но непрактично адаптировать их интерфейсы путем порождения новых подклассов от каждого. В этом случае адаптер объектов может приспосабливать интерфейс их общего родительского класса.

Структура
Адаптер класса использует множественное наследование для адаптации одного интерфейса к другому.
Адаптер объекта применяет композицию объектов.

Участники

- Target (Shape) — целевой:
  - определяет зависящий от предметной области интерфейс, которым пользуется Client;
- Client (DrawingEditor) — клиент:
  - вступает во взаимоотношения с объектами, удовлетворяющими интерфейсу Target;
- Adaptee (Textview) — адаптируемый:
  - определяет существующий интерфейс, который нуждается в адаптации;
- Adapter (Text Shape) — адаптер:
  - адаптирует интерфейс Adaptee к интерфейсу Target.

Отношения
Клиенты вызывают операции экземпляра адаптера Adapter. В свою очередь адаптер вызывает операции адаптируемого объекта или класса Adaptee, который и выполняет запрос.

Результаты
Результаты применения адаптеров объектов и классов различны.
Адаптер класса:

- адаптирует Adaptee к Target, перепоручая действия конкретному классу Adaptee. Поэтому данный паттерн не будет работать, если мы захотим одновременно адаптировать класс и его подклассы;
- позволяет адаптеру Adapter заместить некоторые операции адаптируемого класса Adaptee, так как Adapter есть не что иное, как подкласс Adaptee;
- вводит только один новый объект. Чтобы добраться до адаптируемого класса, не нужно никакого дополнительного обращения по указателю.

Адаптер объектов:

- позволяет одному адаптеру Adapter работать со многим адаптируемыми объектами Adaptee, то есть с самим Adaptee и его подклассами (если таковые имеются). Адаптер может добавить новую функциональность сразу всем адаптируемым объектам;
- затрудняет замещение операций класса Adaptee. Для этого потребуется породить от Adaptee подкласс и заставить Adapter ссылаться на этот подкласс, а не на сам Adaptee.

Ниже приведены вопросы, которые следует рассмотреть, когда вы решаете применить паттерн адаптер:

- объем работы по адаптации. Адаптеры сильно отличаются по тому объему работы, который необходим для адаптации интерфейса Adaptee к интерфейсу Target. Это может быть как простейшее преобразование, например , изменение имен операций, так и поддержка совершенно другого набора операций. Объем работы зависит от того, насколько сильно отличаются друг от друга интерфейсы целевого и адаптируемого классов;
- использование двусторонних адаптеров для обеспечения прозрачности. Адаптеры непрозрачны для всех клиентов. Адаптированный объект уже не обладает интерфейсом Adaptee, так что его нельзя использовать там, где Adaptee был применим. Двусторонние адаптеры способны обеспечить такую прозрачность. Точнее, они полезны в тех случаях, когда клиент должен видеть объект по-разному.

Реализация

Хотя реализация адаптера обычно не вызывает затруднений, кое о чем все же стоит помнить:

- реализация адаптеров классов в C++. В C++ реализация адаптера класса Adapter открыто наследует от класса Target и закрыто - от Adaptee. Таким образом, Adapter должен быть подтипом Target, но не Adaptee;

Родственные паттерны

Структура паттерна мост аналогична структуре адаптера, но у моста иное назначение. Он отделяет интерфейс от реализации, чтобы то и другое можно было изменять независимо. Адаптер же призван изменить интерфейс существующего объекта.

Паттерн декоратор расширяет функциональность объекта, изменяя его интерфейс. Таким образом, декоратор более прозрачен для приложения, чем адаптер. Как следствие, декоратор поддерживает рекурсивную композицию, что для «чистых» адаптеров невозможно.

Заместитель определяет представителя или суррогат другого объекта, но не изменяет его интерфейс.