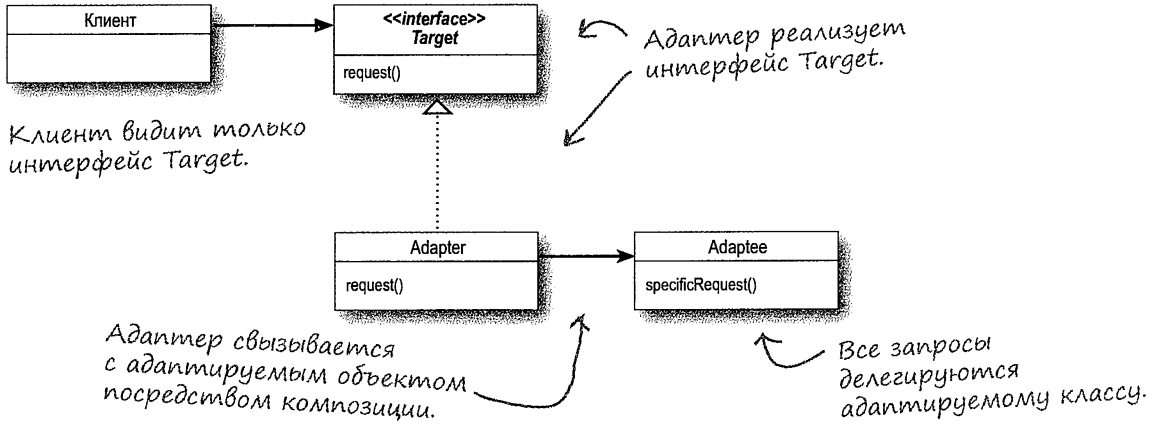
Определение паттерна Адаптер

Довольно уток и индюшек; ниже приведено официальное определение паттерна Адаптер.

> **Паттерн Адаптер** преобразует интерфейс класса к другому интерфейсу, на который рассчитан клиент. Адаптер обеспечивает совместную работу классов, невозможную в обычных условиях из-за несовместимости интерфейсов.

Итак, чтобы использовать клиента с несовместимым интерфейсом, мы создаем адаптер, который выполняет преобразование. Таким образом клиент отделяется от реализованного интерфейса; и если мы ожидаем, что интерфейс будет изменяться со временем, адаптер инкапсулирует эти изменения, чтобы клиента не приходилось изменять каждый раз, когда ему потребуется работать с новым интерфейсом.

Мы проанализировали поведение паттерна в ходе выполнения; давайте также рассмотрим его диаграмму классов:



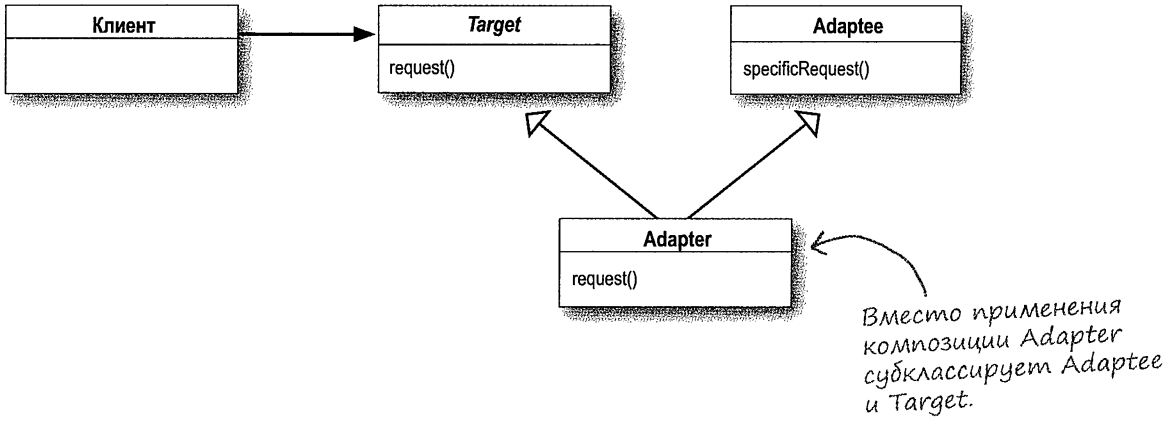
В паттерне Адаптер проявляются многие признаки качественного ОО-проектирования: обратите внимание на использование композиции для «упаковки» адаптируемого объекта в измененный интерфейс. Дополнительное преимущество такого решения заключаетс в том, что адаптер будет работать с любым субклассом адаптируемого объекта.

Также обратите внимание на то, что паттерн связывает клиента с интерфейсом, а не с реализацией; мы можем использовать несколько адаптеров, каждый из которых выполняет преобразование для своего набора классов. Кроме того, новые реализации могут добавляться позднее. Единственным ограничением является лишь их соответствие интерфейсу Target.

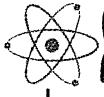
Адаптеры объектов и классов

Мы привели формальное определение паттерна, но еще не рассказали всей истории. На самом деле существует *две* разновидности адаптеров: адаптеры *объектов* и адаптеры *классов*. В этой главе рассматриваются адаптеры объектов, а на диаграмме классов на предыдущей странице также изображен адаптер объектов.

Что же такое адаптер *классов*, и почему мы о них ничего не рассказывали? Потому что для их реализации необходимо множественное наследование, запрещенное в языке Java. Но это не означает, что необходимость в адаптерах классов не возникнет, когда вы будете работать на языке с множественным наследованием! Рассмотрим диаграмму классов с множественным наследованием.



Знакомая картина? Все верно – единственное различие заключается в том, что с адаптером классов мы субклассируем Target и Adaptee, а с адаптером объектов для передачи запросов Adaptee используется механизм композиции.



МОЗГОВОЙ ШТУРМ

Адаптеры объектов и адаптеры классов используют два различных способа адаптации (композиция и наследование). Как эти различия в реализации влияют на гибкость адаптера?