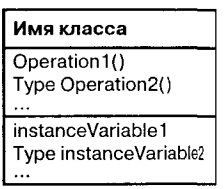
Паттерны проектирования специфицируют также отношения между интерфейсами. В частности, нередко они содержат требование, что некоторые классы должны иметь схожие интерфейсы, а иногда налагают ограничения на интерфейсы классов. Так, декоратор и заместитель требуют, чтобы интерфейсы объектов этих паттернов были идентичны интерфейсам декорируемых и замещаемых объектов соответственно. Интерфейс объекта, принадлежащего паттерну посетитель, должен отражать все классы объектов, с которыми он будет работать.

Специфицирование реализации объектов

До сих пор мы почти ничего не сказали о том, как же в действительности определяется объект. Реализация объекта определяется его *классом*. Класс специфицирует внутренние данные объекта и его представление, а также операции, которые объект может выполнять.

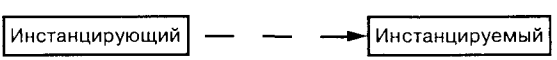
В нашей нотации, основанной на OMT (см. приложение B), класс изображается в виде прямоугольника, внутри которого жирным шрифтом написано имя класса. Ниже обычным шрифтом перечислены операции. Любые данные, которые определены для класса, следуют после операций. Имя класса, операции и данные разделяются горизонтальными линиями.

Типы возвращаемого значения и переменных экземпляра необязательны, поскольку мы не ограничиваем себя языками программирования с сильной типизацией.



Объекты создаются с помощью *инстанцирования* класса. Говорят, что объект является *экземпляром* класса. В процессе инстанцирования выделяется память для *переменных экземпляра* (внутренних данных объекта), и с этими данными ассоциируются операции. С помощью инстанцирования одного класса можно создать много разных объектов-экземпляров.

Пунктирная линия со стрелкой обозначает класс, который инстанцирует объекты другого класса. Стрелка направлена в сторону класса инстанцированного объекта.



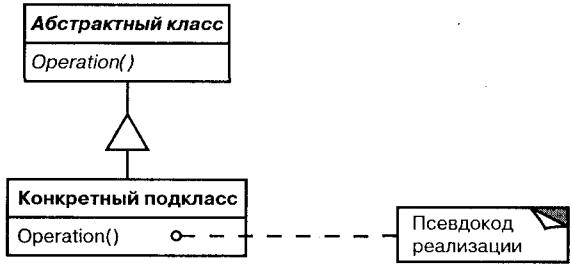
Новые классы можно определить в терминах существующих с помощью *наследования классов*. Если *подкласс* наследует *родительскому классу*, то он включает определения всех данных и операций, определенных в родительском классе. Объекты, являющиеся экземплярами подкласса, будут содержать все данные, определенные как в самом подклассе, так и во всех его родительских классах. Такой объект сможет выполнять все операции, определенные в подклассе и его предках. Отношение *«является подклассом»* обозначается вертикальной линией с треугольником.

Класс называется *абстрактным*, если его единственное назначение – определить общий интерфейс для всех своих подклассов. Абстрактный класс делегирует

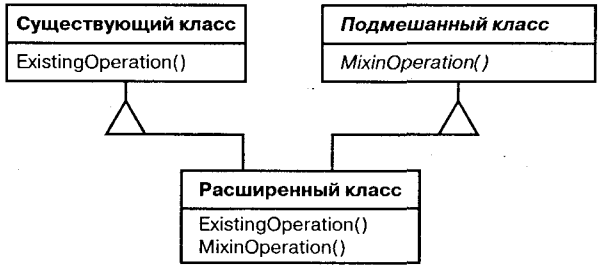
реализацию всех или части своих операций подклассам, поэтому у него не может быть экземпляров. Операции, объявленные, но не реализованные в абстрактном классе, называются *абстрактными*. Класс, не являющийся абстрактным, называется *конкретным*.

Подклассы могут уточнять или переопределять поведение своих предков. Точнее, класс может *заместить* операцию, определенную в родительском классе. Замещение дает подклассам возможность обрабатывать запросы, адресованные родительским классам. Наследование позволяет определять новые классы, просто расширяя возможности старых. Тем самым можно без труда определять семейства объектов со схожей функциональностью.

Имена абстрактных классов оформлены курсивом, чтобы отличать их от конкретных. Курсив используется также для обозначения абстрактных операций. На диаграмме может изображаться псевдокод, описывающий реализацию операции; в таком случае код представлен в прямоугольнике с загнутым уголком, соединенном пунктирной линией с операцией, которую он реализует.



Подмешанным (mixin class) называется класс, назначение которого – предоставить дополнительный интерфейс или функциональность другим классам. Он родственен абстрактным классам в том смысле, что не предполагает непосредственного инстанцирования. Для работы с подмешанными классами необходимо множественное наследование.



Наследование класса и наследование интерфейса

Важно понимать различие между *классом* объекта и его *типом*.

Класс объекта определяет, как объект реализован, то есть внутреннее состояние и реализацию операций объекта. Напротив, тип относится только к интерфейсу

объекта – множеству запросов, на которые объект отвечает. У объекта может быть много типов, и объекты разных классов могут иметь один и тот же тип.

Разумеется, между классом и типом есть тесная связь. Поскольку класс определяет, какие операции может выполнять объект, то заодно он определяет и его тип. Когда мы говорим «объект является экземпляром класса», то подразумеваем, что он поддерживает интерфейс, определяемый этим классом.

В языках вроде C++ и Eiffel классы используются для специфицирования, типа и реализации объекта. В программах на языке Smalltalk типы переменных не объявляются, поэтому компилятор не проверяет, что тип объекта, присваиваемого переменной, является подтипом типа переменной. При отправке сообщения необходимо проверять, что класс получателя реализует реакцию на сообщение, но проверка того, что получатель является экземпляром определенного класса, не нужна.

Важно также понимать различие между наследованием класса и наследованием интерфейса (или порождением подтипов). В случае наследования класса реализация объекта определяется в терминах реализации другого объекта. Проще говоря, это механизм разделения кода и представления. Напротив, наследование интерфейса (порождение подтипов) описывает, когда один объект можно использовать вместо другого.

Две эти концепции легко спутать, поскольку во многих языках явное различие отсутствует. В таких языках, как C++ и Eiffel, под наследованием понимается одновременно наследование интерфейса и реализации. Стандартный способ реализации наследования интерфейса в C++ – это открытое наследование классу, в котором есть исключительно виртуальные функции. Истинное наследование интерфейса можно аппроксимировать в C++ с помощью открытого наследования абстрактному классу. Истинное наследование реализации или класса аппроксимируется с помощью закрытого наследования. В Smalltalk под наследованием понимается только наследование реализации. Переменной можно присвоить экземпляры любого класса при условии, что они поддерживают операции, выполняемые над значением этой переменной.

Хотя в большинстве языков программирования различие между наследованием интерфейса и реализации не поддерживается, на практике оно существует. Программисты на Smalltalk обычно предпочитают считать, что подклассы – это подтипы (хотя имеются и хорошо известные исключения [Coo92]). Программисты на C++ манипулируют объектами через типы, определяемые абстрактными классами.

Многие паттерны проектирования зависят от этого различия. Например, объекты, построенные в соответствии с паттерном цепочка обязанностей, должны иметь общий тип, но их реализация обычно различна. В паттерне компоновщик отдельный объект (компонент) определяет общий интерфейс, но реализацию часто определяет составной объект (композиция). Паттерны команда, наблюдатель, состояние и стратегия часто реализуются абстрактными классами с исключительно виртуальными функциями.

Программирование в соответствии с интерфейсом, а не с реализацией

Наследование классов – это не что иное, как механизм расширения функциональности приложения путем повторного использования функциональности родительских классов. Оно позволяет быстро определить новый вид объектов в терминах уже имеющегося. Новую реализацию вы можете получить посредством наследования большей части необходимого кода из ранее написанных классов.

Однако не менее важно, что наследование позволяет определять семейства объектов с *идентичными* интерфейсами (обычно за счет наследования от абстрактных классов). Почему? Потому что от этого зависит полиморфизм.

Если пользоваться наследованием осторожно (некоторые сказали бы *правильно*), то все классы, производные от некоторого абстрактного класса, будут обладать его интерфейсом. Отсюда следует, что подкласс добавляет новые или замещает старые операции и не скрывает операций, определенных в родительском классе. *Все* подклассы могут отвечать на запросы, соответствующие интерфейсу абстрактного класса, поэтому они являются подтипами этого абстрактного класса.

У манипулирования объектами строго через интерфейс абстрактного класса есть два преимущества:

- клиенту не нужно иметь информации о конкретных типах объектов, которыми он пользуется, при условии, что все они имеют ожидаемый клиентом интерфейс;
- клиенту необязательно «знать» о классах, с помощью которых реализованы объекты. Клиенту известно только об абстрактном классе (или классах), определяющих интерфейс.

Данные преимущества настолько существенно уменьшают число зависимостей между подсистемами, что можно даже сформулировать принцип объектно-ориентированного проектирования для повторного использования: *программируйте в соответствии с интерфейсом, а не с реализацией.*

Не объявляйте переменные как экземпляры конкретных классов. Вместо этого придерживайтесь интерфейса, определенного абстрактным классом. Это одна из наших ключевых идей.

Конечно, где-то в системе вам придется инстанцировать конкретные классы, то есть определить конкретную реализацию. Как раз это и позволяют сделать порождающие паттерны: абстрактная фабрика, строитель, фабричный метод, прототип и одиночка. Абстрагируя процесс создания объекта, эти паттерны предоставляют вам разные способы прозрачно ассоциировать интерфейс с его реализацией в момент инстанцирования. Использование порождающих паттернов гарантирует, что система написана в терминах интерфейсов, а не реализации.

Механизмы повторного использования

Большинству проектировщиков известны концепции объектов, интерфейсов, классов и наследования. Трудность в том, чтобы применить эти знания для построения гибких, повторно используемых программ. С помощью паттернов проектирования вы сможете сделать это проще.