фразы существительные и глаголы, после чего создать соответствующие классы и операции. Другой путь – сосредоточиться на отношениях и разделении обязанностей в системе. Можно построить модель реального мира или перенести выявленные при анализе объекты на свой дизайн. Согласие по поводу того, какой подход самый лучший, никогда не будет достигнуто.

Многие объекты возникают в проекте из построенной в ходе анализа модели. Но нередко появляются и классы, у которых нет прототипов в реальном мире. Это могут быть классы как низкого уровня, например массивы, так и высокого. Паттерн компоновщик вводит такую абстракцию для единообразной трактовки объектов, у которой нет физического аналога. Если придерживаться строгого моделирования и ориентироваться только на реальный мир, то получится система, отражающая сегодняшние потребности, но, возможно, не учитывающая будущего развития. Абстракции, возникающие в ходе проектирования, – ключ к гибкому дизайну.

Паттерны проектирования помогают выявить не вполне очевидные абстракции и объекты, которые могут их использовать. Например, объектов, представляющих процесс или алгоритм, в действительности нет, но они являются неотъемлемыми составляющими гибкого дизайна. Паттерн стратегия описывает способ реализации взаимозаменяемых семейств алгоритмов. Паттерн состояние позволяет представить состояние некоторой сущности в виде объекта. Эти объекты редко появляются во время анализа и даже на ранних стадиях проектирования. Работа с ними начинается позже, при попытках сделать дизайн более гибким и пригодным для повторного использования.

Определение степени детализации объекта

Размеры и число объектов могут сильно варьироваться. С их помощью может быть представлено все, начиная с уровня аппаратуры и до законченных приложений. Как же решить, что должен представлять собой объект?

Здесь и потребуются паттерны проектирования. Паттерн фасад показывает, как представить в виде объекта целые подсистемы, а паттерн приспособленец – как поддержать большое число объектов при высокой степени детализации. Другие паттерны указывают путь к разложению объекта на меньшие подобъекты. Абстрактная фабрика и строитель описывают объекты, единственной целью которых является создание других объектов, а посетитель и команда – объекты, отвечающие за реализацию запроса к другому объекту или группе.

Специфицирование интерфейсов объекта

При объявлении объектом любой операции должны быть заданы: имя операции, объекты, передаваемые в качестве параметров, и значение, возвращаемое операцией. Эту триаду называют *сигнатурой* операции. Множество сигнатур всех определенных для объекта операций называется *интерфейсом* этого объекта. Интерфейс описывает все множество запросов, которые можно отправить объекту. Любой запрос, сигнатура которого соответствует интерфейсу объекта, может быть ему послан.

Тип – это имя, используемое для обозначения конкретного интерфейса. Говорят, что объект имеет тип `Window`, если он готов принимать запросы на выполнение любых операций, определенных в интерфейсе с именем `Window`. У одного объекта может быть много типов. Напротив, сильно отличающиеся объекты могут разделять общий тип. Часть интерфейса объекта может быть охарактеризована одним типом, а часть – другим. Два объекта одного и того же типа должны разделять только часть своих интерфейсов. Интерфейсы могут содержать другие интерфейсы в качестве подмножеств. Мы говорим, что один тип является *подтипом* другого, если интерфейс первого содержит интерфейс второго. В этом случае второй тип называется *супертипом* для первого. Часто говорят также, что подтип *наследует* интерфейс своего супертипа.

В объектно-ориентированных системах интерфейсы фундаментальны. Об объектах известно только то, что они сообщают о себе через свои интерфейсы. Никакого способа получить информацию об объекте или заставить его что-то сделать в обход интерфейса не существует. Интерфейс объекта ничего не говорит о его реализации; разные объекты вправе реализовывать сходные запросы совершенно по-разному. Это означает, что два объекта с различными реализациями могут иметь одинаковые интерфейсы.

Когда объекту посылается запрос, то операция, которую он будет выполнять, зависит как от запроса, так и от объекта-адресата. Разные объекты, поддерживающие одинаковые интерфейсы, могут выполнять в ответ на такие запросы разные операции. Ассоциация запроса с объектом и одной из его операций во время выполнения называется *динамическим связыванием*.

Динамическое связывание означает, что отправка некоторого запроса не определяет никакой конкретной реализации до момента выполнения. Следовательно, допустимо написать программу, которая ожидает объект с конкретным интерфейсом, точно зная, что любой объект с подходящим интерфейсом сможет принять этот запрос. Более того, динамическое связывание позволяет во время выполнения подставить вместо одного объекта другой, если он имеет точно такой же интерфейс. Такая взаимозаменяемость называется *полиморфизмом* и является важнейшей особенностью объектно-ориентированных систем. Она позволяет клиенту не делать почти никаких предположений об объектах, кроме того, что они поддерживают определенный интерфейс. Полиморфизм упрощает определение клиентов, позволяет отделить объекты друг от друга и дает объектам возможность изменять взаимоотношения во время выполнения.

Паттерны проектирования позволяют определять интерфейсы, задавая их основные элементы и то, какие данные можно передавать через интерфейс. Паттерн может также «сказать», что не должно проходить через интерфейс. Хорошим примером в этом отношении является хранитель. Он описывает, как инкапсулировать и сохранить внутреннее состояние объекта таким образом, чтобы в будущем его можно было восстановить точно в таком же состоянии. Объекты, удовлетворяющие требованиям паттерна хранитель, должны определить два интерфейса: один ограниченный, который позволяет клиентам держать у себя и копировать хранители, а другой привилегированный, которым может пользоваться только сам объект для сохранения и извлечения информации о состоянии их хранителя.

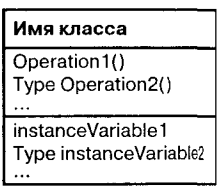
Паттерны проектирования специфицируют также отношения между интерфейсами. В частности, нередко они содержат требование, что некоторые классы должны иметь схожие интерфейсы, а иногда налагают ограничения на интерфейсы классов. Так, декоратор и заместитель требуют, чтобы интерфейсы объектов этих паттернов 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.

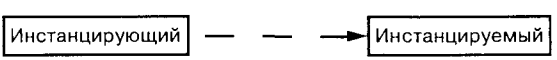
В нашей нотации, основанной на OMT (см. приложение B), класс изображается в виде прямоугольника, внутри которого жирным шрифтом написано имя класса. Ниже обычным шрифтом перечислены операции. Любые данные, которые определены для класса, следуют после операций. Имя класса, операции и данные разделяются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