



Глава 1. Введение в паттерны проектирования

Проектирование объектно-ориентированных программ – нелегкое дело, а если их нужно *использовать повторно*, то все становится еще сложнее. Необходимо подобрать подходящие объекты, отнести их к различным классам, соблюдая разумную степень детализации, определить интерфейсы классов и иерархию наследования и установить существенные отношения между классами. Дизайн должен, с одной стороны, соответствовать решаемой задаче, с другой – быть общим, чтобы удалось учесть все требования, которые могут возникнуть в будущем. Хотелось бы также избежать вовсе или, по крайней мере, свести к минимуму необходимость перепроектирования. Поднаторевшие в объектно-ориентированном проектировании разработчики скажут вам, что обеспечить «правильный», то есть в достаточной мере гибкий и пригодный для повторного использования дизайн, с первого раза очень трудно, если вообще возможно. Прежде чем считать цель достигнутой, они обычно пытаются опробовать найденное решение на нескольких задачах, и каждый раз модифицируют его.

И все же опытным проектировщикам удается создать хороший дизайн системы. В то же время новички испытывают шок от количества возможных вариантов и нередко возвращаются к привычным не объектно-ориентированным методикам. Проходит немало времени перед тем, как становится понятно, что же такое удачный объектно-ориентированный дизайн. Опытные проектировщики, очевидно, знают какие-то тонкости, ускользающие от новичков. Так что же это?

Прежде всего, опытному разработчику понятно, что *не нужно* решать каждую новую задачу с нуля. Вместо этого он старается повторно воспользоваться теми решениями, которые оказались удачными в прошлом. Отыскав хорошее решение один раз, он будет прибегать к нему снова и снова. Именно благодаря накопленному опыту проектировщик и становится экспертом в своей области. Во многих объектно-ориентированных системах вы встретите повторяющиеся паттерны, состоящие из классов и взаимодействующих объектов. С их помощью решаются конкретные задачи проектирования, в результате чего объектно-ориентированный дизайн становится более гибким, элегантным, и им можно воспользоваться повторно. Проектировщик, знакомый с паттернами, может сразу же применять их к решению новой задачи, не пытаясь каждый раз изобретать велосипед.

Поясним нашу мысль через аналогию. Писатели редко выдумывают совершенно новые сюжеты. Вместо этого они берут за основу уже отработанные в мировой литературе схемы, жанры и образы. Например, трагический герой – Макбет,

Гамлет и т.д., мотив убийства – деньги, месть, ревность и т.п. Точно так же в объектно-ориентированном проектировании используются такие паттерны, как «представление состояния с помощью объектов» или «декорирование объектов, чтобы было проще добавлять и удалять их свойства».

Все мы знаем о ценности опыта. Сколько раз при проектировании вы испытывали *дежавю*, чувствуя, что уже когда-то решали такую же задачу, только никак не сообразить, когда и где? Если бы удалось вспомнить детали старой задачи и ее решения, то не пришлось бы придумывать все заново. Увы, у нас нет привычки записывать свой опыт на благо другим людям да и себе тоже.

Цель этой книги состоит как раз в том, чтобы документировать опыт разработки объектно-ориентированных программ в виде *паттернов проектирования*. Каждому паттерну мы присвоим имя, объясним его назначение и роль в проектировании объектно-ориентированных систем. Некоторые из наиболее распространенных паттернов формализованы и сведены в единый каталог.

Паттерны проектирования упрощают повторное использование удачных проектных и архитектурных решений. Представление прошедших проверку временем методик в виде паттернов проектирования облегчает доступ к ним со стороны разработчиков новых систем. С помощью паттернов можно улучшить качество документации и сопровождения существующих систем, позволяя явно описать взаимодействия классов и объектов, а также причины, по которым система была построена так, а не иначе. Проще говоря, паттерны проектирования дают разработчику возможность быстрее найти «правильный» путь.

Как уже было сказано, в книгу включены только такие паттерны, которые неоднократно применялись в разных системах. По большей части они никогда ранее не документировались и либо известны самым квалифицированным специалистам по объектно-ориентированному проектированию, либо были частью какой-то удачной системы.

Хотя книга получилась довольно объемной, паттерны проектирования – лишь малая часть того, что необходимо знать специалисту в этой области. В издание не включено описание паттернов, имеющих отношение к параллельности, распределенному программированию и программированию систем реального времени. Отсутствуют и сведения о паттернах, специфичных для конкретных предметных областей. Из этой книги вы не узнаете, как строить интерфейсы пользователя, как писать драйверы устройств и как работать с объектно-ориентированными базами данных. В каждой из этих областей есть свои собственные паттерны, и, может быть, кто-то их и систематизирует.

1.1. Что такое паттерн проектирования

По словам Кристофера Александра, «любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип ее решения, причем таким образом, что это решение можно потом использовать миллион раз, ничего не изобретая заново» [AIS+77]. Хотя Александр имел в виду паттерны, возникающие при проектировании зданий и городов, но его слова верны и в отношении паттернов объектно-ориентированного проектирования. Наши решения выражаются

в терминах объектов и интерфейсов, а не стен и дверей, но в обоих случаях смысл паттерна – предложить решение определенной задачи в конкретном контексте.

В общем случае паттерн состоит из четырех основных элементов:

1. *Имя*. Сославшись на него, мы можем сразу описать проблему проектирования, ее решения и их последствия. Присваивание паттернам имен позволяет проектировать на более высоком уровне абстракции. С помощью словаря паттернов можно вести обсуждение с коллегами, упоминать паттерны в документации, в тонкостях представлять дизайн системы. Нахождение хороших имен было одной из самых трудных задач при составлении каталога.
2. *Задача*. Описание того, когда следует применять паттерн. Необходимо сформулировать задачу и ее контекст. Может описываться конкретная проблема проектирования, например способ представления алгоритмов в виде объектов. Иногда отмечается, какие структуры классов или объектов свидетельствуют о негибком дизайне. Также может включаться перечень условий, при выполнении которых имеет смысл применять данный паттерн.
3. *Решение*. Описание элементов дизайна, отношений между ними, функций каждого элемента. Конкретный дизайн или реализация не имеются в виду, поскольку паттерн – это шаблон, применимый в самых разных ситуациях. Просто дается абстрактное описание задачи проектирования и того, как она может быть решена с помощью некоего весьма обобщенного сочетания элементов (в нашем случае классов и объектов).
4. *Результаты* – это следствия применения паттерна и разного рода компромиссы. Хотя при описании проектных решений о последствиях часто не упоминают, знать о них необходимо, чтобы можно было выбрать между различными вариантами и оценить преимущества и недостатки данного паттерна. Здесь речь идет и о выборе языка и реализации. Поскольку в объектно-ориентированном проектировании повторное использование зачастую является важным фактором, то к результатам следует относить и влияние на степень гибкости, расширяемости и переносимости системы. Перечисление всех последствий поможет вам понять и оценить их роль.

То, что один воспринимает как паттерн, для другого просто строительный блок. В этой книге мы рассматриваем паттерны на определенном уровне абстракции. *Паттерны проектирования* – это не то же самое, что связанные списки или хэш-таблицы, которые можно реализовать в виде класса и повторно использовать без каких бы то ни было модификаций. Но это и не сложные, предметно-ориентированные решения для целого приложения или подсистемы. Здесь под паттернами проектирования понимается *описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте*.

Паттерн проектирования именуется, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна. Он вычленяет участвующие классы и экземпляры, их роль и отношения, а также функции. При описании каждого паттерна внимание акцентируется на конкретной задаче объектно-ориентированного проектирования. Анализируется, когда следует применять паттерн, можно ли

его использовать с учетом других проектных ограничений, каковы будут последствия применения метода. Поскольку любой проект в конечном итоге предстоит реализовывать, в состав паттерна включается пример кода на языке C++ (иногда на Smalltalk), иллюстрирующего реализацию.

Хотя, строго говоря, паттерны используются в проектировании, они основаны на практических решениях, реализованных на основных языках объектно-ориентированного программирования типа Smalltalk и C++, а не на процедурных (Pascal, C, Ada и т.п.) или объектно-ориентированных языках с динамической типизацией (CLOS, Dylan, Self). Мы выбрали Smalltalk и C++ из прагматических соображений, поскольку чаще всего работаем с ними и поскольку они завоевывают все большую популярность.

Выбор языка программирования безусловно важен. В наших паттернах подразумевается использование возможностей Smalltalk и C++, и от этого зависит, что реализовать легко, а что – трудно. Если бы мы ориентировались на процедурные языки, то включили бы паттерны **наследование**, **инкапсуляция** и **полиморфизм**. Некоторые из наших паттернов напрямую поддерживаются менее распространенными языками. Так, в языке CLOS есть мультиметоды, которые делают ненужным паттерн **посетитель**. Собственно, даже между Smalltalk и C++ есть много различий, из-за чего некоторые паттерны проще выражаются на одном языке, чем на другом (см., например, паттерн итератор).

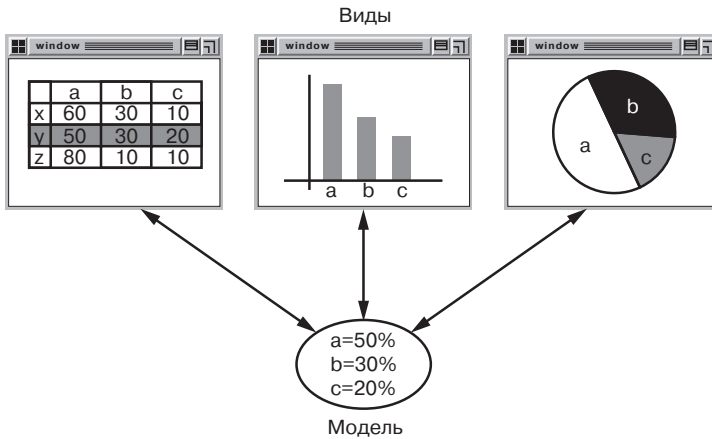
1.2. Паттерны проектирования в схеме MVC в языке Smalltalk

В Smalltalk-80 для построения интерфейсов пользователя применяется тройка классов **модель/вид/контроллер** (Model/View/Controller – MVC) [KP88]. Знакомство с паттернами проектирования, встречающимися в схеме MVC, поможет вам разобраться в том, что мы понимаем под словом «паттерн».

MVC состоит из объектов трех видов. *Модель* – это объект приложения, а *вид* – экранное представление. *Контроллер* описывает, как интерфейс реагирует на управляющие воздействия пользователя. До появления схемы MVC эти объекты в пользовательских интерфейсах смешивались. MVC отделяет их друг от друга, за счет чего повышается гибкость и улучшаются возможности повторного использования.

MVC отделяет вид от модели, устанавливая между ними протокол взаимодействия «подписка/оповещение». Вид должен гарантировать, что внешнее представление отражает состояние модели. При каждом изменении внутренних данных модель оповещает все зависящие от нее виды, в результате чего вид обновляет себя. Такой подход позволяет присоединить к одной модели несколько видов, обеспечив тем самым различные представления. Можно создать новый вид, не переписывая модель.

На рисунке ниже показана одна модель и три вида. (Для простоты мы опустили контроллеры.) Модель содержит некоторые данные, которые могут быть представлены в виде электронной таблицы, гистограммы и круговой диаграммы.



Модель оповещает свои виды при каждом изменении значений данных, а виды обращаются к модели для получения новых значений.

На первый взгляд, в этом примере продемонстрирован просто дизайн, отделяющий вид от модели. Но тот же принцип применим и к более общей задаче: разделение объектов таким образом, что изменение одного отражается сразу на нескольких других, причем изменившийся объект не имеет информации о деталях реализации объектов, на которые он оказал воздействие. Этот более общий подход описывается паттерном проектирования **наблюдатель**.

Еще одно свойство MVC заключается в том, что виды могут быть вложенными. Например, панель управления, состоящую из кнопок, допустимо представить как составной вид, содержащий вложенные, – по одной кнопке на каждый. Пользовательский интерфейс инспектора объектов может состоять из вложенных видов, используемых также и в отладчике. MVC поддерживает вложенные виды с помощью класса `CompositeView`, являющегося подклассом `View`. Объекты класса `CompositeView` ведут себя так же, как объекты класса `View`, поэтому могут использоваться всюду, где и виды. Но еще они могут содержать вложенные виды и управлять ими.

Здесь можно было бы считать, что этот дизайн позволяет обращаться с составным видом, как с любым из его компонентов. Но тот же дизайн применим и в ситуации, когда мы хотим иметь возможность группировать объекты и рассматривать группу как отдельный объект. Такой подход описывается паттерном **компоновщик**. Он позволяет создавать иерархию классов, в которой некоторые подклассы определяют примитивные объекты (например, `Button` – кнопка), а другие – составные объекты (`CompositeView`), группирующие примитивы в более сложные структуры.

MVC позволяет также изменять реакцию вида на действия пользователя. При этом визуальное представление остается прежним. Например, можно изменить реакцию на нажатие клавиши или использовать всплывающее меню вместо командных клавиш. MVC инкапсулирует механизм определения реакции в объекте `Controller`. Существует иерархия классов контроллеров, и это позволяет без труда создать новый контроллер как вариант уже существующего.

Вид пользуется экземпляром класса, производного от `Controller`, для реализации конкретной стратегии реагирования. Чтобы реализовать иную стратегию, нужно просто подставить другой контроллер. Можно даже заменить контроллер вида во время выполнения программы, изменив тем самым реакцию на действия пользователя. Например, вид можно деактивировать, так что он вообще не будет ни на что реагировать, если передать ему контроллер, игнорирующий события ввода.

Отношение **вид-контроллер** – это пример паттерна проектирования **стратегия**. **Стратегия** – это объект для представления алгоритма. Он полезен, когда вы хотите статически или динамически подменить один алгоритм другим, если существует много вариантов одного алгоритма или когда с алгоритмом связаны сложные структуры данных, которые хотелось бы инкапсулировать.

В MVC используются и другие паттерны проектирования, например **фабричный метод**, позволяющий задать для вида класс контроллера по умолчанию, и **декоратор** для добавления к виду возможности прокрутки. Но основные отношения в схеме MVC описываются паттернами **наблюдатель**, **компоновщик** и **стратегия**.

1.3. Описание паттернов проектирования

Как мы будем описывать паттерны проектирования? Графических обозначений недостаточно. Они просто символизируют конечный продукт процесса проектирования в виде отношений между классами и объектами. Чтобы повторно воспользоваться дизайном, нам необходимо документировать решения, альтернативные варианты и компромиссы, которые привели к нему. Важны также конкретные примеры, поскольку они позволяют увидеть применение паттерна.

При описании паттернов проектирования мы будем придерживаться единого принципа. Описание каждого паттерна разбито на разделы, перечисленные ниже. Такой подход позволяет единообразно представить информацию, облегчает изучение, сравнение и применение паттернов.

Название и классификация паттерна

Название паттерна должно четко отражать его назначение. Классификация паттернов проводится в соответствии со схемой, которая изложена в разделе 1.5.

Назначение

Лаконичный ответ на следующие вопросы: каковы функции паттерна, его обоснование и назначение, какую конкретную задачу проектирования можно решить с его помощью.

Известен также под именем

Другие распространенные названия паттерна, если таковые имеются.

Мотивация

Сценарий, иллюстрирующий задачу проектирования и то, как она решается данной структурой класса или объекта. Благодаря мотивации можно лучше понять последующее, более абстрактное описание паттерна.