Глава 1
Введение в объектно-ориентированные концепции

Хотя многие программисты не осознают этого, объектно-ориентированная разработка программного обеспечения существует с начала 1960-х годов. Только во второй половине 1990-х годов объектно-ориентированная парадигма начала набирать обороты, несмотря на тот факт, что популярные объектно-ориентированные языки программирования вроде Smalltalk и C++ уже широко использовались.

Расцвет объектно-ориентированных технологий совпал с началом применения Интернета в качестве платформы для бизнеса и развлечений. А после того как стало очевидным, что Глобальная сеть активно проникает в жизнь людей, объектно-ориентированные технологии уже заняли удобную позицию для того, чтобы помочь в разработке новых веб-технологий.

Важно подчеркнуть, что название этой главы звучит как «Введение в объектно-ориентированные концепции». В качестве ключевого здесь использовано слово «концепции», а не «технологии». Технологии в индустрии программного обеспечения очень быстро изменяются, в то время как концепции эволюционируют. Я использовал термин «эволюционируют» потому, что, хотя концепции остаются относительно устойчивыми, они все же претерпевают изменения. Это очень интересная особенность, заметная при тщательном изучении концепций. Несмотря на их устойчивость, они постоянно подвергаются повторным интерпретациям, а это предполагает весьма любопытные дискуссии.

Эту эволюцию можно легко проследить за последние два десятка лет, если понаблюдать за прогрессом различных индустриальных технологий, начиная с первых примитивных браузеров второй половины 1990-х годов и заканчивая мобильными/телефонными/веб-приложениями, доминирующими сегодня. Как и всегда, новые разработки окажутся не за горами, когда мы будем исследовать гибридные приложения и пр. На всем протяжении путешествия объектно-ориентированные концепции присутствовали на каждом этапе. Вот почему вопросы, рассматриваемые в этой главе, так важны. Эти концепции сегодня так же актуальны, как и 20 лет назад.

Фундаментальные концепции

Основная задача этой книги — заставить вас задуматься о том, как концепции используются при проектировании объектно-ориентированных систем. Исторически

сложилось так, что объектно-ориентированные языки определяются следующими концепциями: *инкапсуляцией*, *наследованием* и *полиморфизмом*. Поэтому если тот или иной язык программирования не реализует все эти концепции, то он, как правило, не считается объектно-ориентированным. Наряду с этими тремя терминами я всегда включаю в общую массу композицию; таким образом, мой список объектно-ориентированных концепций выглядит так:

- инкапсуляция;
- наследование;
- полиморфизм;
- композиция.

Мы детально рассмотрим все эти концепции в данной книге.

Одна из трудностей, с которыми мне пришлось бороться еще с самого первого издания книги, заключается в том, как эти концепции соотносятся непосредственно с текущими методиками проектирования, ведь они постоянно изменяются. Например, все время ведутся дебаты об использовании наследования при объектно-ориентированном проектировании. Нарушает ли наследование инкапсуляцию на самом деле? (Эта тема будет рассмотрена в последующих главах.) Даже сейчас многие разработчики стараются избегать наследования, насколько это представляется возможным.

Мой подход, как и всегда, состоит в том, чтобы придерживаться концепций. Независимо от того, будете вы использовать наследование или нет, вам как минимум потребуется понять, что такое наследование, благодаря чему вы сможете сделать обоснованный выбор методики проектирования. Как уже отмечалось во введении к этой книге, ее целевой аудиторией являются люди, которым требуется *общее введение в фундаментальные объектно-ориентированные концепции*. Исходя из этой формулировки в текущей главе я представляю фундаментальные объектно-ориентированные концепции с надеждой обеспечить моим читателям твердую основу для принятия важных решений касательно проектирования. Рассматриваемые здесь концепции затрагивают большинство, если не все темы, охватываемые в последующих главах, в которых соответствующие вопросы исследуются намного подробнее.

Объекты и унаследованные системы

По мере того как объектно-ориентированное программирование получало широкое распространение, одна из проблем, с которыми сталкивались разработчики, заключалась в интеграции объектно-ориентированных технологий с существующими системами. В то время разграничивались объектно-ориентированное и структурное (или процедурное) программирование, которое было доминирующей парадигмой разработки на тот момент. Мне всегда это казалось странным, поскольку, на мой взгляд, объектно-ориентированное и структурное программирование не конкурируют друг с другом. Они являются взаимодополняющими, так как объекты хорошо интегрируются со структурированным кодом. Даже сейчас я часто слышу такой вопрос: «Вы занимаетесь структурным или объектно-ориентированным программированием?» Недолго думая, я бы ответил: «И тем, и другим».

В том же духе объектно-ориентированный код не призван заменить структурированный код. Многие не являющиеся объектно-ориентированными *унаследованные системы* (то есть более старые по сравнению с уже используемыми) довольно хорошо справляются со своей задачей. Зачем же тогда идти на риск столкнуться с возможными проблемами, изменяя или заменяя эти унаследованные системы? В большинстве случаев вам не следует изменять их, по крайней мере не ради лишь внесения изменений. В сущности, в системах, основанных не на объектно-ориентированном коде, нет ничего плохого. Однако совершенно новые разработки, несомненно, подталкивают задуматься об использовании объектно-ориентированных технологий (в некоторых случаях нет иного выхода, кроме как поступить именно так).

Хотя на протяжении последних 20 лет наблюдалось постоянное и значительное увеличение количества объектно-ориентированных разработок, зависимость мирового сообщества от сетей вроде Интернета и мобильных инфраструктур поспособствовала еще более широкому их распространению. Буквально взрывной рост количества транзакций, осуществляемых в браузерах и мобильных приложениях, открыл совершенно новые рынки, где значительная часть разработок программного обеспечения была новой и главным образом не обремененной заботами, связанными с унаследованными системами. Но даже если вы все же столкнетесь с такими заботами, то на этот случай есть тенденция, согласно которой унаследованные системы можно заключать в *объектные обертки*.

ОБЪЕКТНЫЕ ОБЕРТКИ

Объектные обертки представляют собой объектно-ориентированный код, в который заключается другой код. Например, вы можете взять структурированный код (вроде циклов и условий) и заключить его в объект, чтобы этот код выглядел как объект. Вы также можете использовать объектные обертки для заключения в них функциональности, например параметров, касающихся безопасности, или непереносимого кода, связанного с аппаратным обеспечением, и т. д. Обертывание структурированного кода детально рассматривается в главе 6.

Сегодня одной из наиболее интересных областей разработки программного обеспечения является интеграция унаследованного кода с мобильными и веб-системами. Во многих случаях мобильное клиентское веб-приложение в конечном счете «подключается» к данным, располагающимся на мейнфрейме. Разработчики, одновременно обладающие навыками в веб-разработке как для мейнфреймов, так и для мобильных устройств, весьма востребованны.

Вы сталкиваетесь с объектами в своей повседневной жизни, вероятно, даже не осознавая этого. Вы можете столкнуться с ними, когда едете в своем автомобиле, разговариваете по сотовому телефону, используете свою домашнюю развлекательную систему, играете в компьютерные игры, а также во многих других ситуациях. Электронные соединения по сути превратились в соединения, основанные на объектах. Тяготея к мобильным веб-приложениям, бизнес тяготеет к объектам, поскольку технологии, используемые для электронной торговли, по своей природе в основном являются объектно-ориентированными.

МОБИЛЬНАЯ ВЕБ-РАЗРАБОТКА

Несомненно, появление Интернета значительно поспособствовало переходу на объектно-ориентированные технологии. Дело в том, что объекты хорошо подходят для использования в сетях. Хотя Интернет был в авангарде этой смены парадигмы, мобильные сети теперь заняли не последнее место в общей массе. В этой книге термин «мобильная веб-разработка» будет использоваться в контексте концепций, которые относятся как к разработке мобильных веб-приложений, так и к веб-разработке. Термин «гибридные приложения» иногда будет применяться для обозначения приложений, которые работают в браузерах как на веб-устройствах, так и на мобильных аппаратах.

Процедурное программирование в сравнении с объектно-ориентированным

Прежде чем мы углубимся в преимущества объектно-ориентированной разработки, рассмотрим более существенный вопрос: что именно такое объект? Это одновременно и сложный, и простой вопрос. Сложный он потому, что изучение любого метода разработки программного обеспечения не является тривиальным. А простой он в силу того, что люди уже мыслят объектно.

Например, когда вы смотрите на какого-то человека, вы видите его как объект. При этом объект определяется двумя компонентами: атрибутами и поведением. У человека имеются такие атрибуты, как цвет глаз, возраст, вес и т. д. Человек также обладает поведением, то есть он ходит, говорит, дышит и т. д. В соответствии со своим базовым определением *объект* — это сущность, *одновременно* содержащая данные и поведения.

Слово *«одновременно»* в данном случае определяет ключевую разницу между объектно-ориентированным программированием и другими методологиями программирования. Например, при процедурном программировании код размещается в полностью отдельных функциях или процедурах. В идеале, как показано на рис. 1.1, эти процедуры затем превращаются в «черные ящики», куда поступают входные данные и откуда потом выводятся выходные данные. Данные размещаются в отдельных структурах, а манипуляции с ними осуществляются с помощью этих функций или процедур.

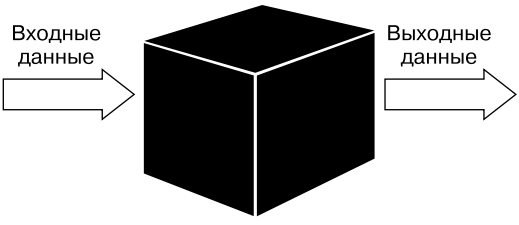

Рис. 1.1. Черный ящик

РАЗНИЦА МЕЖДУ ОБЪЕКТНО-ОРИЕНТИРОВАННЫМ И СТРУКТУРНЫМ ПРОЕКТИРОВАНИЕМ

При объектно-ориентированном проектировании атрибуты и поведения размещаются в рамках одного объекта, в то время как при процедурном или структурном проектировании атрибуты и поведения обычно разделяются.

При росте популярности объектно-ориентированного проектирования один из фактов, который изначально тормозил его принятие людьми, заключался в том, что использовалось много систем, которые не являлись объектно-ориентированными, но отлично работали. Таким образом, с точки зрения бизнеса не было никакого смысла изменять эти системы лишь ради внесения изменений. Каждому, кто знаком с любой компьютерной системой, известно, что то или иное изменение может привести к катастрофе, даже если предполагается, что это изменение будет незначительным.

В то же время люди не принимали объектно-ориентированные базы данных. В определенный момент при появлении объектно-ориентированной разработки в какой-то степени вероятным казалось то, что такие базы данных смогут заменить реляционные базы данных. Однако этого так никогда и не произошло. Бизнес вложил много денег в реляционные базы данных, а совершению перехода препятствовал главный фактор — они работали. Когда все издержки и риски преобразования систем из реляционных баз данных в объектно-ориентированные стали очевидными, неоспоримых доводов в пользу перехода не оказалось.

На самом деле бизнес сейчас нашел золотую середину. Для многих методик разработки программного обеспечения характерны свойства объектно-ориентированной и структурной методологий разработки.

Как показано на рис. 1.2, при структурном программировании данные зачастую отделяются от процедур и являются глобальными, благодаря чему их легко модифицировать вне области видимости вашего кода. Это означает, что доступ к данным неконтролируемый и непредсказуемый (то есть у множества функций может быть доступ к глобальным данным). Во-вторых, поскольку у вас нет контроля над тем, кто сможет получить доступ к данным, тестирование и отладка намного усложняются. При работе с объектами эта проблема решается путем объединения данных и поведения в рамках одного элегантного полного пакета.

ПРАВИЛЬНОЕ ПРОЕКТИРОВАНИЕ

Мы можем сказать, что при правильном проектировании в объектно-ориентированных моделях нет такого понятия, как глобальные данные. По этой причине в объектно-ориентированных системах обеспечивается высокая степень целостности данных.

Вместо того чтобы заменять другие парадигмы разработки программного обеспечения, объекты стали эволюционной реакцией. Структурированные программы содержат комплексные структуры данных вроде массивов и т. д. C++ включает структуры, которые обладают многими характеристиками объектов (классов).

Однако объекты представляют собой нечто намного большее, чем структуры данных и примитивные типы вроде целочисленных и строковых. Хотя объекты содержат такие сущности, как целые числа и строки, используемые для представления атрибутов, они также содержат методы, которые характеризуют поведение. В объектах методы применяются для выполнения операций с данными, а также

для совершения других действий. Пожалуй, более важно то, что вы можете управлять доступом к членам объектов (как к атрибутам, так и к методам). Это означает, что отдельные из этих членов можно скрыть от других объектов. Например, объект с именем `Math` может содержать две целочисленные переменные с именами `myInt1` и `myInt2`. Скорее всего, объект `Math` также содержит методы, необходимые для извлечения значений `myInt1` и `myInt2`. Он также может включать метод с именем `sum()` для сложения двух целочисленных значений.

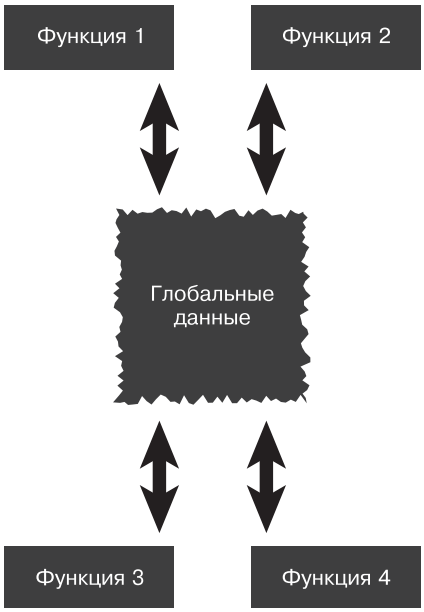

Рис. 1.2. Использование глобальных данных

СКРЫТИЕ ДАННЫХ

В объектно-ориентированной терминологии данные называются атрибутами, а поведения — методами. Ограничение доступа к определенным атрибутам и/или методам называется скрытием данных.

Объединив атрибуты и методы в одной сущности (это действие называется *инкапсуляцией*), мы можем управлять доступом к данным в объекте `Math`. Если определить целочисленные переменные `myInt1` и `myInt2` в качестве «запретной зоны», то другая логически несвязанная функция не будет иметь возможности осуществлять манипуляции с ними, и только объект `Math` сможет делать это.

РУКОВОДСТВО ПО ПРОЕКТИРОВАНИЮ КЛАССОВ SOUND

Можно создать неудачно спроектированные объектно-ориентированные классы, которые не ограничивают доступ к атрибутам классов. Суть в том, что при объектно-ориентированном проектировании вы можете создать плохой код с той же легкостью, как и при использовании любой другой методологии программирования. Просто примите меры для того, чтобы придерживаться руководства по проектированию классов Sound (см. главу 5).