Е. В. Пышкин

ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

ОСНОВНЫЕ КОНЦЕПЦИИ И МЕХАНИЗМЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

- *Объектная модель и ее развитие в современных языках программирования*
- *Иерархическое проектирование типов и программ*
- *Введение в обобщенное программирование*
- *Управляемые среды проектирования*
- *Основы компонентного программирования*
- *Демонстрационные программы, вопросы и упражнения для самостоятельной работы, примеры заданий практикума*

УЧЕБНОЕ ПОСОБИЕ

bhv

Е. В. Пышкин

ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

ОСНОВНЫЕ КОНЦЕПЦИИ И МЕХАНИЗМЫ ОБЪЕКТНО–ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Рекомендовано Учебно-методическим объединением по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 553000 – «Системный анализ и управление»

Санкт-Петербург

«БХВ-Петербург»

2005

УДК 681.3.068(075.8)
ББК 32.973.26-018.1я73
П94

Пышкин Е. В.

П94 Основные концепции и механизмы объектно-ориентированного программирования. — СПб.: БХВ-Петербург, 2005. — 640 с.: ил.

ISBN 5-94157-554-8

Рассматривается понятие объектной модели и анализируются механизмы управления вычислительным процессом, лежащие в основе объектно-ориентированного подхода: классы и интерфейсы, динамическое связывание, обработка исключений, пространства имен. Подробно рассматривается конструирование обобщенных типов и библиотека ввода-вывода применительно к программированию на C++. Содержится информация об управляемом коде, свойствах, делегатах, событиях, специализированных атрибутах, отражении, основах компонентной архитектуры.

Размещенные на компакт-диске демонстрационные программы разработаны с использованием современных языков программирования: C++, Java, C#, Visual Basic.

Для студентов и преподавателей технических вузов

УДК 681.3.068(075.8)
ББК 32.973.26-018.1я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Мирошенков*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

РЕЦЕНЗЕНТЫ:
Жабко А. П., доктор физ.-мат. наук, проф., зав. каф. теории управления СПбГУ
Евдокимов В. Е., к. т. н., доцент каф. информационно-измерительных технологий СПбГПУ

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 15.06.05.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 51,6.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-554-8

Оглавление

Посвящается моим любимым
Галине и Матвею

Предисловие

Книга, предлагаемая вниманию читателей, основывается на книге "Основные концепции объектно-ориентированного программирования" [Пышкин, 2003-2], представляя ее существенно переработанный и расширенный вариант. Использование материалов книги при проведении занятий со студентами Санкт-Петербургского государственного политехнического университета (СПбГПУ) позволило уточнить и улучшить содержание многих разделов книги, а также дополнить их новой информацией, актуальной в свете развития современных технологий программирования.

С одной стороны, опыт преподавания программирования в рамках конкретного академического курса способствовал написанию книги и выбору обозреваемых разделов, образуя методическую основу книги. С другой стороны, благодаря написанию этой книги, содержание курса, безусловно, стало более стройным и систематическим. При этом справедливым можно считать тезис, что задача учебной книги не только в том, чтобы помочь студентам в освоении нового материала, но и в том, чтобы помочь самому автору более детально разобраться в обсуждаемых вопросах и методике преподавания дисциплины и, как следствие, улучшить и сделать более интересным процесс обучения студентов.

Преподавание: исторический экскурс

В этой книге анализируются основные концепции и механизмы, используемые и развиваемые объектно-ориентированными языками программирования. Рассмотрение отдельных концепций иллюстрируется фрагментами исходных текстов программ, преимущественно написанных на языке C++. Я и мои коллеги, преподаватели СПбГПУ, работающие на кафедре автоматики и вычислительной техники, начали использовать языки C и C++ в качестве базовых при преподавании программирования примерно с середины 1990-х го-

дов. До этого в учебном процессе использовался преимущественно язык Pascal, заместивший, в свою очередь, языки Fortran и PL/1.

Будучи студентом, автор сначала изучал в рамках университетского курса язык Fortran, а затем — именно язык PL/1 (о чем вспоминает с воодушевлением и не без некоторой ностальгии). Язык PL/1 был сконструирован, исходя из принципа, что любая полезная конструкция, которая может быть реализована, должна по возможности включаться в язык. Это обусловило сложность и большой размер языка PL/1, который включает около 200 служебных слов. Многим студентам было нелегко справиться с этой сложностью (хотя автору и удавалось провести "интенсивный двухдневный курс для отстающих" — это было изматывающее предприятие, закончившееся тем, что подопечные сдали экзамен со средним баллом 4.5, — может быть, именно тогда желание заниматься преподавательской деятельностью обрело конкретные очертания).

Многие специалисты полагают, что из перечисленных языков нуждам академического процесса лучше всего соответствует язык Pascal, который, будучи несомненным успехом Вирта, первоначально и был ориентирован на учебные цели. Pascal, являясь удачным сочетанием простоты и выразительности, характеризуется довольно строгой системой типов, жестким контролем преобразований типов и своей очевидной ориентированностью на принципы структурного программирования. Некоторые ограничения, присущие оригинальной версии языка, были преодолены в различных диалектах. Одной из наиболее распространенных и удачных версий многие считают язык Turbo Pascal, реализованный в системах проектирования Borland. Большой популярностью среди программистов продолжает пользоваться объектно-ориентированный язык Object Pascal, положенный в основу системы программирования Delphi (собственно, в наши дни даже более употребительным является наименование "язык Delphi").

При этом следует отметить, что с середины 1990-х годов популярность языка Pascal стала падать как в промышленности, так и в университетах (в особенности в США, что, впрочем, оказало дальнейшее влияние и на Европу, и на Россию) [Sebesta, 2002]. Все большая часть коммерческого программного обеспечения стала разрабатываться на языках C и C++. Таким образом, исходя из нацеленности кафедры на подготовку специалистов-практиков, необходимо было включить в учебные программы язык C++. В отличие от языка Pascal, ни язык C, ни язык C++ не были ориентированы на учебные цели, а создавались их авторами для программистов-профессионалов, поэтому преподавание (а следовательно, и изучение) языка C и особенно языка C++ — не такое уж простое дело.

Долгое время сложность преподавания и изучения языка C++ была связана и с отсутствием общемирового стандарта языка. В то самое время, когда мы начали преподавать язык C++ (т. е. в середине 1990-х годов), программисты

шутили, что процесс стандартизации постоянно находился в состоянии "2 года до завершения" [Зуев, Кротов, Сухомлин, 1996]. При этом Адель Голдберг, бывшая одним из авторов языка Smalltalk, вообще называет C++ большим и сложным языком без целостной философии, за исключением требования сохранить пользовательскую базу языка C: "Одна из основных целей этого языка — сохранение эффективности и особенностей языка C одновременно с обеспечением преимуществ объектно-ориентированного программирования. Некоторые люди интуитивно чувствуют, что эти свойства языка C++ не всегда хорошо согласуются друг с другом" [Sebesta, 2002] (однако продолжают активно использовать его в качестве инструмента проектирования сложных систем и комплексов — Е. П.). Мысли Страуструпа по этому поводу довольно подробно изложены в книге [Stroustrup, 1994].

Основная задача книги

Приступая к работе над книгой, автор стремился избежать стиля книг, упрощающих изучаемый предмет, книг типа "Язык X — для чайников", "Программирование — это просто" и т. д. Несмотря на то, что формат курса, равно как и формат учебного пособия, приводит к почти неизбежной выборочности изучаемых разделов, автор старался не избегать сложных и тонких моментов, а явно указывать на них, одновременно ориентируя студентов в большом объеме издаваемой литературы. Это означает, что, по замечанию Зуева, нельзя о сложных вещах говорить упрощенно, дабы у читателя не сформировался усеченный и выхолощенный образ рассматриваемых инструментов программирования: "Настоящее пособие по сложному языку должно иметь форму, близкую к (...) книге Эллис и Страуструпа [Ellis, Stroustrup, 1990], — комментируемый стандарт. Да, читать и пытаться понять строгий, сложно построенный, местами даже занудный текст весьма непросто — но кто сказал, что профессия программиста проста?" [Зуев, 1997].

Несмотря на то, что большинство примеров реализовано на языке C++, в отдельных главах рассматриваются особенности реализации некоторых концепций и механизмов объектно-ориентированного и компонентного программирования в языках Java и C#. Автор немедленно принимает очевидную критику относительно выбора языков (все они являются, по существу, языками C-группы), и в связи с этим хочет подчеркнуть три момента:

1. Выбор обусловлен наличием личного практического опыта последних лет.
2. Данное учебное пособие нацелено на обеспечение академического курса по объектно-ориентированному программированию. В связи с этим методически удобно ограничиться каким-либо базовым языком и привлекать другие языки либо для иллюстрации концепций, не поддерживаемых ба-

зовым языком, либо для иллюстрации существенных отличий в реализации.

3. Включение в рассмотрение других современных объектно-ориентированных языков (Eiffel, Ada95, Visual Basic, языки платформы Microsoft .NET) является отдельной задачей и требует отдельного решения, которое лучше всего вписывается в рамки курса "Языки программирования", необходимого для обеспечения всестороннего образования в области программных технологий. Не подлежит сомнению, что хороший программный инженер должен знать несколько языков программирования [Meyer, 2001].

Сопровождающие текст примеры в основном довольно просты и непохожи на реальные задачи проектирования. Ценность таких "игрушечных" примеров заключается в том, что, изучая их, студенты смогут понять основные иллюстрируемые ими концепции и механизмы, не оказываясь при этом под давлением объемных исходных текстов.

Отметим, что в данной книге не анализируются во всей полноте достоинства и недостатки объектно-ориентированных технологий, а также вопросы сложности разработки программного обеспечения. Отличные книги, содержащие исследование этой проблематики, упоминаются в списке источников (см., в частности, [Booch, 1994], [Brooks, 1995], [Sebesta, 2002], [Stroustrup, 1994], [Лекарев, 1997]).

Благодарности

Хочу выразить свою благодарность всем коллегам, работающим на кафедре автоматики и вычислительной техники факультета технической кибернетики СПбГПУ, за постоянную поддержку в преподавании и научно-педагогическом творчестве. Благодарю сотрудников учебно-методического объединения по университетскому политехническому образованию за помощь в совершенствовании структуры учебного пособия.

Ряд материалов, использованных при подготовке книги, прошли апробацию в ходе курсов по разработке программного обеспечения в Central Ostrobothnia Polytechnic, Ylivieska, Финляндия. Организация этих занятий была бы невозможна без участия и поддержки моих финских коллег Вейкко Бракса и Юхи Хуумонена. Благодарю профессора Ханспетера Мессенбека из университета г. Линца (Австрия) за лекции, прочитанные им на одной из конференций Microsoft. Систематичность изложения основных элементов программирования на языке C# остается для меня одним из ориентиров в педагогической практике.

Благодарю студентов Андрея Власовских, Александра Григоряна, Алексея Рытенкова, Юрия Викторова, Михаила Веренцова, Андрея Хилько, Алек-

сандра Павлова, Илью Антоненко, Анатолия Карпенко, Ивана Пивоварова, Марата Ахина за их замечания и находки, способствовавшие устранению ошибок и улучшению содержания книги.

Особая признательность — Вере Павловне Тихомировой, в чьем гостеприимном доме в деревне Найдениха Тверской области было написано большинство глав книги.

Спасибо музыке Франца Шуберта, помогавшей мне в плодотворной и комфортной работе над книгой.

Введение

> Специалисты по программному обеспечению — лучшие в мире специалисты по созданию хаоса. Мы должны объяснить студентам, что это непреложный факт и показать им, что следует в связи с этим предпринимать.
>
> *Бертран Мейер*

Предпосылки

Автор исходит из предположения, что читатели книги имеют представление об основных принципах процедурного программирования и, в частности, знакомы с концепциями, лежащими в основе структурного программирования. Необходимой предпосылкой для успешного освоения материала книги является знакомство с основными средствами языка программирования C++, не связанными с объектно-ориентированным программированием.

Восприятию текстов программ, помещенных в книге, читателям поможет знание английского языка. Впрочем, его желательно знать любому квалифицированному специалисту, работающему в области проектирования и использования вычислительных систем.

Для самостоятельного практикума по программированию на C++ читателю потребуются средства разработки на C++. При организации занятий со студентами автор преимущественно использует средства Microsoft Visual Studio 6.0 и Microsoft Visual Studio .NET. На факультативных занятиях студенты используют средства проектирования на языках Java (в рамках изучаемого материала достаточно иметь одну из версий JDK, поддерживающих платформу Java2) и C# (для этого требуется Microsoft .Net SDK или развернутая среда Visual Studio .NET).

Организация данной книги

Книга состоит из введения, четырех частей и заключения, а также содержит три приложения.

Часть I "Основание объектно-ориентированного программирования" включает главы 1—2.

В *главе 1* анализируется развитие языков программирования с точки зрения развития воплощаемых ими концепций проектирования и положенных в основу языков абстрактных моделей. Вкратце представляется генеалогия современных языков программирования и место тех языков, которые используются в книге для иллюстраций.

В *главе 2* представляется понятие объектной модели и анализируются ее основные элементы: абстрагирование, инкапсуляция, иерархия и модульность. В главе рассматриваются отношения между основными абстракциями и простые примеры их использования. Для иллюстрации применяются диаграммы UML (Unified Modeling Language) [Booch, 1999]. При написании первых двух глав частично использованы материалы работы, написанной автором для студентов Central Ostrobothnia Polytechnic (Финляндия) [Пышкин, 2003-1].

Часть II "Конструирование типов" посвящена концепциям и механизмам, лежащим в основе конструирования типов, построения иерархий типов и управления доступом к данным, и включает главы 3—6.

Глава 3 посвящена углубленному изучению классов как основного механизма абстракций в объектно-ориентированном программировании. Рассматривается структура и содержание определения класса, разновидности членов класса и основные принципы конструирования новых типов. Анализируются основные проблемы инициализации и разрушения объектов, перегрузка методов класса и операций, обеспечение правильной семантики копирования объектов.

В *главе 4* представлен основной инструмент построения иерархий классов — наследование. Обсуждаются права доступа к членам классов в связи с наследованием. Анализируется концепция сокрытия имен, рассматриваются различия реализации этой концепции различными языками программирования. Сопоставляются объектно-ориентированные модели организации данных на базе однокоренной иерархии классов и без оной.

Глава 5 посвящена важнейшей концепции объектно-ориентированного программирования — полиморфизму (динамическому связыванию). Анализируются различия механизмов реализации полиморфизма в разных объектно-ориентированных языках и причины этих различий. Студенты получают представление об определении типа времени выполнения и динамическом преобразовании типов.

Глава 6 посвящена анализу механизма исключений как основного инструмента обработки ошибок в объектно-ориентированных языках. Рассматриваются альтернативные варианты реализации обработки ошибок в компьютер-

ной программе и преимущества, получаемые программистом при использовании обработки исключений. Подробно рассматриваются особенности реализации механизма обработки исключений в языке C++, включая разрушение объектов в ходе раскрутки стека, организацию обработки неожидаемых и неперехваченных исключений, исключения, возникающие в ходе ошибок распределения памяти. В главе анализируются задачи спецификации исключений и вкратце рассматривается реализация спецификации исключений в разных языках программирования.

Часть III "Организация и взаимодействие типов" посвящена обсуждению концепций и механизмов, традиционно вызывающих трудности у начинающих программистов, в том числе в связи с реализацией множественного наследования, поддержкой пространств имен и обобщенного программирования. Третья часть включает главы 7—9.

В *главе 7* рассматриваются подробности реализации множественного наследования в C++ и обсуждаются причины, из-за которых многие объектно-ориентированные языки не поддерживают множественное наследование. Студенты получают представление о концепции одиночного наследования с интерфейсами (с примером на языке C#).

В *главе 8* рассматриваются пространства имен как инструмент обеспечения модульности программ на уровне логической организации компонентов. Анализируются присущие C++ ограничения в реализации концепции пространств имен и реализация пространств имен в других языках (Java, C#).

Глава 9 посвящена элементам обобщенного программирования: рассматриваются шаблонные и обобщенные функции, шаблонные классы. Глава содержит введение в обработку контейнерных объектов с помощью обобщенных функций. Рассматриваются основные принципы построения контейнеров, используемые в стандартной библиотеке шаблонов (STL, Standard Template Library). Подробно рассматривается понятие итератора контейнера и приводятся примеры проектирования контейнерных классов в стиле стандартной библиотеки шаблонов.

Часть IV посвящена анализу развития объектно-ориентированных моделей в современных языках и технологиях программирования и включает главы 10—12.

В *главе 10* представляются основы организации управляемого кода и основные механизмы управления типами в средах, построенных на основе концепции управляемого кода.

Глава 11 посвящена развитию моделей управления типами в современных объектно-ориентированных языках, отличных от C++ (анализируются модульность, полиморфизм и версии виртуальных методов, свойства, индекса-

торы, делегаты, специализированные атрибуты, события, документирующие комментарии).

Глава 12 содержит введение в компонентное программирование. В главе анализируется влияние компонентной модели на структуру объектно-ориентированного кода, рассматриваются основные проблемы обеспечения межъязыковой и межплатформенной совместимости компонентов.

Заключение посвящено читателям, дочитавшим эту книгу до конца.

Приложения содержат примеры заданий практикума по объектно-ориентированному программированию, описание основных средств библиотеки ввода-вывода C++, необходимых в работе над учебными проектами, а также описание структуры и содержания материалов, предоставляемых на прилагаемом к книге компакт-диске.

Исходные тексты программ и листинги программ

Исходные тексты примеров программ, использованных в книге, предоставляются на компакт-диске в виде единого пакета. Набор файлов с исходными текстами сопровождается файлом license.txt, содержащим информацию об авторском праве и условиях использования исходных текстов в собственной работе.

Часть представленных в книге исходных текстов пронумерована. Листинги нумеруются в пределах отдельных глав. Все пронумерованные листинги соответствуют программам, распространяемым вместе с книгой в пакете исходных текстов программ. В этом случае листинги, представленные в книге, являются копиями или фрагментами программ с компакт-диска, которые были оттранслированы компиляторами Visual C++ 6.0 и Visual C++ 7.0. Случаи, когда результаты исполнения программы зависят от версии компилятора, оговариваются в комментариях. Незначительные отличия текстов программ, приведенных в книге, от компилируемых текстов на компакт-диске могут быть связаны только с исключением из текста книги подробных заголовочных комментариев и с особенностями форматирования текста, обусловленными требованиями издателя. Такие различия не затрагивают содержание программного кода.

О соответствии стандарту C++

При разработке примеров программ автор стремился обеспечить соответствие получаемых программ требованиям стандартного C++ *(ISO/IEC 14882 "Standard for the C++ Programming Language")*. Исходя из сложившейся на

сегодняшний день организации учебного процесса, в ходе которого студентам сначала представляется процедурная парадигма программирования и концепции структурного императивного программирования *(см. во введении разд. "Предпосылки")*, в некоторых примерах программ могут встречаться отклонения от рекомендаций по следованию стандарту, изложенных, в частности, Страуструпом [Stroustrup, 2000, приложение Б]. Это означает, что некоторые программы, не нарушая стандарта с формальной точки зрения, могут отклоняться от стиля программирования на стандартном C++. Повторяю, что это обусловлено исключительно методическими задачами и особенностями учебного процесса и затрагивает лишь некоторые элементы языка, которые перечисляются ниже.

В частности, в приводимых учебных примерах используются средства стандартной библиотеки C++, определенные в пространстве имен `std`. Поэтому в тех случаях, когда необходимы стандартные средства языка C (например, при работе со строками ANSI C), используются заголовочные файлы без расширения, например

```
# include <cstring>
```

вместо

```
# include <string.h>
```

При использовании средств стандартной библиотеки C++ в программу помещается явное указание на используемое пространство имен, например:

```
#include <iostream>
using namespace std;
```

Это позволяет обеспечить совместимость кода с требованиями стандарта, не вызывая серьезных проблем в восприятии кода даже теми студентами, которые до этого использовали библиотеки языка C. Укажем, что подробности реализации пространств имен в C++ и механизмы обеспечения совместимости со старым кодом *рассматриваются в главе 8*.

В большинстве программ, приводимых до раздела 6.3, при использовании операции резервирования памяти `new` результат ее выполнения проверяется на 0. В соответствии с требованиями стандарта, в старом коде проверки на равенство 0 результата, возвращаемого `new`, должны быть заменены перехватом исключений `bad_alloc` или применением `new (nothrow)` [Stroustrup, 2000, §Б.3.4]. После рассмотрения реализации механизма обработки исключения `bad_alloc` в разделе 6.3, в дальнейших главах при распределении памяти используется версия `new`, генерирующая исключение `bad_alloc` в случае ошибки.

Несмотря на рекомендацию использовать вместо C-строк более высокоуровневый тип `string` (см., например, [Stroustrup, 2000, §20.4.1]), в ряде учебных

проектов продолжают применяться строки в стиле C. Такое решение обусловлено тем обстоятельством, что в этих примерах моделируется процесс создания более высокоуровневых типов на основе существующих механизмов низкого уровня. Это позволяет, в частности, рассмотреть проблемы копирования объектов и резервирования-освобождения динамической памяти в процессе функционирования объектов. В тех случаях, где строки нужны как таковые, используется преимущественно тип `string`.

В целом, автор стремился руководствоваться справедливым замечанием Страуструпа о том, что общее подмножество C и C++ не является лучшим подмножеством для изучения начинающими программистами [Stroustrup, 2000, раздел Б.4]. В связи с этим автор заранее приносит извинения за возможные отклонения от этой рекомендации, которые может заметить внимательный и строгий читатель.

Одно замечание о терминологии

В ряде переводных изданий, посвященных объектно-ориентированному программированию и в особенности программированию на языке C++, переводчики изменили сложившуюся в русском языке терминологическую традицию. Наиболее подробное объяснение этого факта содержится в редакционном предисловии к книге "Язык программирования C++" Страуструпа ([Stroustrup, 2000]). Позволю себе привести пространную цитату из этого текста:

> "Слово "statement" принято переводить на русский язык как "оператор". Мы привыкли к тому, что if, while, case и т. д. — это операторы. Увы, в контексте C++ такой подход неприемлем. Дело в том, что в C++ слово "operator" (которое и переведено как "оператор") имеет совсем другое значение. Оно применяется для обозначения сложения (оператор `+`), разыменования (оператор `*`), выяснения размера (оператор `sizeof()`) и в других подобных случаях. (...) Что же касается термина "statement", то из предлагаемых различными словарями вариантов "утверждение", "предложение", "инструкция" мы выбрали последний, т. к. он, по-видимому, лучше всего соответствует сущности обозначаемых словом "statement" конструкций C++ и, кроме того, периодически встречается в книгах в нужном значении".

По поводу процитированного фрагмента хотелось бы высказать следующие соображения:

1. (Общее соображение). По-моему, в принципе неверно из контекста использования какого-либо термина в ограниченной области, изменять сложившиеся языковые традиции, иначе как общаться с людьми, не работающими в этой ограниченной области?

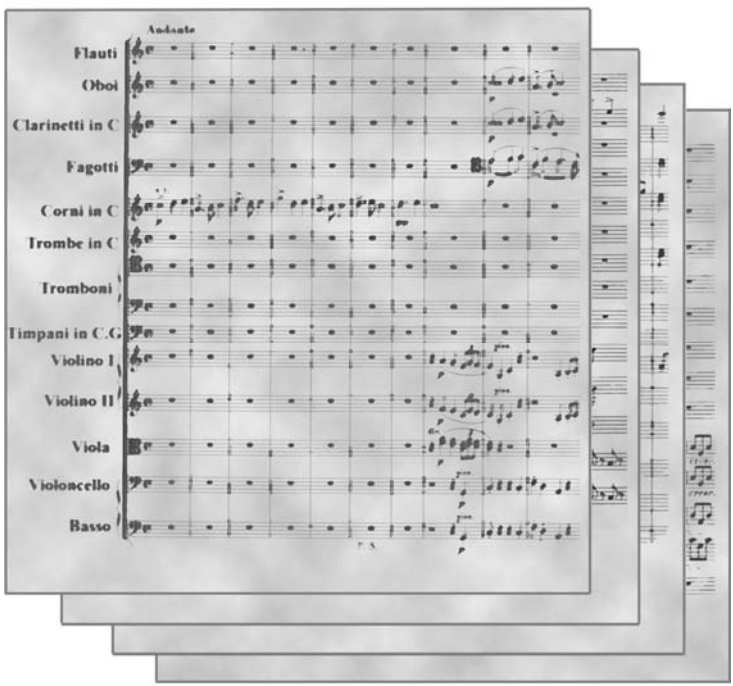
2. (Частное соображение). Не совсем ясно, что означает фраза "в C++ слово "operator" имеет совсем другое значение". Оно имеет то значение, которое и предписано ему английским языком: операция, знак операции! Собственно, вполне корректно сказать, что перегрузка операций является приданием новой семантики знаку операции, и в этом смысле использование служебного слова "operator" довольно точно соответствует существу дела. При этом словари дают вполне адекватный перевод этого термина на русский язык, не идущий вразрез с установившейся традицией.
3. (A propos). Язык C++ включает в себя в качестве подмножества язык C. Одной из существенных сложностей создания и развития C++ является поддержание пользовательской базы языка C. Мне кажется, что пользовательская база включает также и терминологию, используемую в связи с определением конструкций языка. А ведь язык C создавался без служебного слова "operator"!
4. (Наблюдение). Не ссылаясь на авторов, замечу, что мне попадались книги по C++, где понятия "операция" и "оператор" были весьма причудливо перемешаны: видимо, авторы не отнеслись с должным вниманием к означенному редакционному предисловию и использовали эти термины как придется.

На основании изложенных соображений я склонен использовать слово "операция" там, где речь идет об операциях языка (operators). Но чтобы у читателя не возникало излишней путаницы, я буду использовать вполне разумный эквивалент "инструкция" для английского термина "statement". При этом я постараюсь минимизировать использование конфликтного слова "оператор" (хотя кое-где оно все же может встретиться в тексте в своем "исконном" значении, очень уж трудно произносить "инструкция if" вместо "условный оператор" или "инструкция goto" вместо "оператор goto").

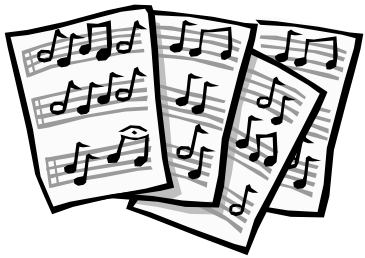
> Может быть, программисты и лучшие в мире специалисты по созданию хаоса, но определенно не единственные.

ЧАСТЬ I

Основание объектно-ориентированного программирования



ГЛАВА 1



Развитие языков как развитие абстрактных моделей

> Язык и мышление теснейшим образом связаны. Если язык обеднеет, обеднеет и мышление. Колоссальное значение имеет терминология, грамматические системы, способствующие самовыражению человека.
>
> *Д. С. Лихачев*

Объектная модель представляет собой концептуальную основу объектно-ориентированного программирования, поэтому изучение основных элементов объектной модели и их взаимодействия позволяет лучше разобраться в принципах организации объектно-ориентированных программ, а также понять, в чем сходства и различия поддержки объектно-ориентированной парадигмы различными языками.

Прежде чем перейти к изучению элементов объектной модели как таковых, проанализируем вкратце процесс развития языков программирования. Это позволит нам выяснить концептуальные и методологические различия изобразительных средств, предоставляемых программисту языками программирования, и причины успешности и востребованности объектно-ориентированного подхода.

1.1. Концепции проектирования и языки программирования

Объектно-ориентированное программирование (ООП) является, вероятно, самой распространенной в современной проектной практике парадигмой программирования, которая поддерживается большинством современных языков программирования. При этом реализации концепций и механизмов, состав-

ляющих основание ООП, могут отличаться в разных языках. Поэтому изучение ООП предполагает не столько изучение какого-либо конкретного языка программирования, сколько лежащих в его основе концепций и механизмов как таковых. К самым важным вещам, которые должен знать профессиональный программный инженер, относится набор фундаментальных концепций, которые вновь и вновь используются в его работе и которые в первую очередь следует изучать [Meyer, 2001]. С одной стороны, такой подход к изучению программирования позволяет лучше разобраться в проблемах, возникающих перед разработчиками программного обеспечения (ПО), с другой стороны, позволяет лучше понять причины появления и успешного развития различных языков и технологий программирования.

Основная задача любой технологии заключается в том, чтобы предоставить разработчикам средства решения основных проблем проектирования. Широкое применение технологии (которая всегда является реализацией определенных концепций) доказывает ее успешность. В частности, это и означает, что данная технология (а значит, и лежащие в ее основе концепции, модели, методы проектирования) позволяет разработчикам успешно решать актуальные проблемы проектирования.

Вообще говоря, появление новых технологий программирования в большой степени обусловлено трансформацией проблем проектирования ПО с его растущей сложностью. В свою очередь, рост сложности программного обеспечения обусловлен развитием областей жизнедеятельности человека, где находят применение новейшие информационные и компьютерные технологии. Это становится возможным благодаря развитию средств проектирования и возможностей вычислительной техники, удешевлению аппаратных ресурсов и увеличению продуктивности разработчиков, использующих современные средства проектирования. С другой стороны, сейчас уже никто не удивляется тому, что новейшие технологии проектирования зачастую требуют более мощного аппаратного обеспечения для своей эффективной работы (хотя это иногда и связано с неэффективностью алгоритмов, используемых в ходе разработки программ), подталкивая пользователей к модернизации используемого оборудования и вычислительных средств. Таким образом, новое более мощное оборудование способствует более эффективному использованию программных средств, это порождает новые области применения и дает толчок к возникновению новых задач проектирования, требующих для своего решения новых технологий разработки программ [Пышкин, 2003-1]. Графически описанный процесс представлен на рис. 1.1.

Язык программирования — инструмент, вытекающий из концепций проектирования, инструмент, облекающий в синтаксическую форму основное содержание концепций, их суть. Поэтому изучение языка самого по себе вне связи с моделями проектирования, заложенными в основу языка, не влечет улуч-

шения качества проектирования и не обязательно способствует повышению квалификации разработчика.

Рис. 1.1. Прогресс информационных технологий и рост сложности задачи проектирования

Разумеется, нужно отдавать себе отчет в том, что *качество проектирования*, прежде всего, *определяется квалификацией и творческими способностями проектировщиков*, а не моделью программирования: выдающиеся проекты создают выдающиеся проектировщики [Brooks, 1995]. При этом технология проектирования помогает грамотному разработчику продемонстрировать свои умения и построить качественный программный продукт. Таким образом, *необходимо изучать не языки программирования сами по себе, а концепции, выражаемые этими языками* [Eckel, 2000-1].

1.2. Абстрактные модели, лежащие в основе языков программирования

Языки программирования — это всегда некоторые модели (виртуальные вычислительные машины), позволяющие наиболее эффективно использовать возможности вычислительных средств, существенные для конкретных областей применения. По сути дела, при написании программ ориентируются не на вычислительную машину как таковую, а на некоторую абстрактную модель вычислительного устройства [Minsky, 1967]. Качество абстрактной модели, ее соответствие содержанию решаемых задач во многом определяет качество и эффективность проектирования с использованием этой модели. По меткому замечанию Эккеля, компьютеры не столько машины, сколько средства усиления мысли [Eckel, 2000-1].