

Стивен Дьюкерст



Стт

**СВЯЩЕННЫЕ
ЗНАНИЯ**

C++ Common Knowledge

Essential Intermediate Programming

Stephen C. Dewhurst

ПРОФЕ  ИОНАЛЬНО

C++

СВЯЩЕННЫЕ ЗНАНИЯ

Основы успеха для будущих профессионалов

Издание исправленное

Стивен С. Дьюхерст



Санкт-Петербург — Москва
2012

Серия «Профессионально»

Стивен С. Дьюхерст

C++. Священные знания

Перевод Н. Шатохиной

Главный редактор	<i>А. Галунов</i>
Зав. редакцией	<i>Н. Макарова</i>
Научный редактор	<i>Н. Лощинин</i>
Редактор	<i>В. Овчинников</i>
Художник	<i>В. Гренда</i>
Корректор	<i>С. Минин</i>
Верстка	<i>Д. Орлова</i>

Дьюхерст С.

C++. Священные знания. – Пер. с англ. – СПб.: Символ-Плюс, 2012. – 240 с., ил.
ISBN 978-5-93286-095-3

Стивен Дьюхерст, эксперт по C++ с более чем 20-летним опытом применения C++ в различных областях, рассматривает важнейшие, но зачастую неправильно понимаемые темы программирования и проектирования на C++, отсеивая при этом ненужные технические тонкости. В один тонкий том Стив уместил то, что он и его рецензенты, опытные консультанты и авторы, считают самым необходимым для эффективного программирования на C++.

Книга адресована тем, кто имеет опыт программирования на C++ и испытывает необходимость быстро повысить свое знание C++ до профессионального уровня. Издание полезно и квалифицированным программистам на C или Java, имеющим небольшой опыт проектирования и разработки сложного кода на C++ и склонным программировать на C++ в стиле Java.

ISBN 978-5-93286-095-3
ISBN 0-321-32192-8 (англ)

© Издательство Символ-Плюс, 2008, 2012

Authorized translation of the English edition © 2005 Pearson Education Inc. This translation is published and sold by permission of Pearson Education Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛПН N 000054 от 25.12.98.

Подписано в печать 26.12.2011. Формат 70x90^{1/16}.
Печать офсетная. Объем 15 печ. л.

Оглавление

	Предисловие	11
Тема 1	Абстракция данных	19
Тема 2	Полиморфизм	21
Тема 3	Паттерны проектирования	25
Тема 4	Стандартная библиотека шаблонов	29
Тема 5	Ссылки – это псевдонимы, а не указатели	32
Тема 6	Массив как тип формального параметра	35
Тема 7	Константные указатели и указатели на константу	38
Тема 8	Указатели на указатели	41
Тема 9	Новые операторы приведения	44
Тема 10	Смысл константной функции-члена	48
Тема 11	Компилятор дополняет классы	52
Тема 12	Присваивание и инициализация – это не одно и то же	55
Тема 13	Операции копирования	58
Тема 14	Указатели на функции	61
Тема 15	Указатели на члены класса – это не указатели	64
Тема 16	Указатели на функции-члены – это не указатели	67
Тема 17	Разбираемся с операторами объявления функций и массивов	70

Тема 18	Объекты-функции	73
Тема 19	Команды и Голливуд	77
Тема 20	Объекты-функции STL	81
Тема 21	Перегрузка и переопределение – это не одно и то же	84
Тема 22	Шаблонный метод	86
Тема 23	Пространства имен	89
Тема 24	Поиск функции-члена	94
Тема 25	Поиск, зависимый от типов аргументов	96
Тема 26	Поиск операторной функции	98
Тема 27	Запросы возможностей	100
Тема 28	Смысл сравнения указателей	103
Тема 29	Виртуальные конструкторы и Прототип	105
Тема 30	Фабричный метод	108
Тема 31	Ковариантные возвращаемые типы	111
Тема 32	Предотвращение копирования	114
Тема 33	Как сделать базовый класс абстрактным	115
Тема 34	Ограничение на размещение в куче	118
Тема 35	Синтаксис размещения new	120
Тема 36	Индивидуальное управление памятью	123
Тема 37	Создание массивов	126
Тема 38	Аксиомы надежности исключений	129
Тема 39	Надежные функции	132
Тема 40	Методика RAII	135
Тема 41	Операторы new, конструкторы и исключения	139
Тема 42	Умные указатели	141
Тема 43	Указатель auto_ptr – штука странная	143
Тема 44	Арифметика указателей	145

Тема 45	Терминология шаблонов	149
Тема 46	Явная специализация шаблона класса	151
Тема 47	Частичная специализация шаблонов	155
Тема 48	Специализация членов шаблона класса	159
Тема 49	Устранение неоднозначности с помощью ключевого слова <code>typename</code>	162
Тема 50	Члены-шаблоны	166
Тема 51	Устранение неоднозначности с помощью ключевого слова <code>template</code>	170
Тема 52	Создание специализации для получения информации о типе . .	173
Тема 53	Встроенная информация о типе	178
Тема 54	Свойства	181
Тема 55	Параметры-шаблоны шаблона	186
Тема 56	Политики	191
Тема 57	Логический вывод аргументов шаблона	195
Тема 58	Перегрузка шаблонов функций	199
Тема 59	Концепция SFINAE	202
Тема 60	Универсальные алгоритмы	206
Тема 61	Мы создаем экземпляр того, что используем	210
Тема 62	Стражи включения	213
Тема 63	Необязательные ключевые слова	215
	Библиография	218
	Алфавитный указатель	219

ОТЗЫВЫ О КНИГЕ

«Мы живем во время, когда, как это ни удивительно, только начинают появляться самые лучшие печатные работы по C++. Эта книга – одна из них. Хотя C++ стоит в авангарде нововведений и продуктивности разработки программного обеспечения более двух десятилетий, только сейчас мы достигли полного его понимания и использования. Данная книга – это один из тех редких вкладов, ценный как для практиков, так и для теоретиков. Это не трактат тайных или академических знаний. Скорее, книга дает доскональное описание известных, казалось бы, вещей, непонимание которых рано или поздно даст о себе знать. Очень немногие овладели C++ и проектированием программного обеспечения так, как это сделал Стив. Практически никто не обладает такой рассудительностью и спокойствием, когда речь идет о разработке программного обеспечения. Он знает, что необходимо знать, поверьте мне. Когда он говорит, я всегда внимательно слушаю. Советую и вам делать так же. Вы (и ваши заказчики) будете благодарны за это.»

*Чак Эллисон (Chuck Allison),
редактор журнала «The C++ Source»*

«Стив обучил меня C++. Это было в далеком 1982 или 1983. Я думаю, он тогда только вернулся с последипломной практики, которую проходил вместе с Бьерном Страуструпом (Bjarne Stroustrup) [создателем C++] в научно-исследовательском центре Bell Laboratories. Стив – один из невоспетых героев, стоящих у истоков. Все, что им написано, я отношу к обязательному и первоочередному чтению. Эта книга легко читается, она вобрала в себя большую часть обширных знаний и опыта Стива. Настоятельно рекомендую с ней ознакомиться.»

*Стен Липман (Stan Lippman),
соавтор книги «C++ Primer, Fourth Edition»*

«Я приветствую обдуманый профессиональный подход коротких и толковых книг.»

*Мэтью П. Джонсон (Matthew P. Johnson),
Колумбийский университет*

«Согласен с классификацией программистов [сделанной автором]. В своей практике разработчика мне доводилось встречаться с подобными людьми. Эта книга должна помочь им заполнить пробел в образовании... Я думаю, данная книга дополняет другие, такие как «Effective C++» Скотта Мейерса (Scott Meyers). Вся информация в ней представлена кратко и проста для восприятия.»

*Моуаз Кэмел (Moataz Kamel), ведущий разработчик
программного обеспечения, компания Motorola, Канада*

«Дьюхерст написал еще одну очень хорошую книгу. Она необходима людям, использующим C++ (и думающим, что они уже все знают о C++).»

*Кловис Тондо (Clovis Tondo),
соавтор книги «C++ Primer Answer Book»*

Предисловие

*Успешность книги определяется не ее содержанием,
а тем, что осталось вне ее рассмотрения.*

Марк Твен

...просто, насколько возможно, но не проще.

Альберт Эйнштейн

*...писатель, ставящий под вопрос
умственные способности читателя,
вообще не писатель, а просто прожектор.*

Е. В. Уайт

Заняв должность редактора ныне закрывшегося журнала «C++ Report», непоседливый Герб Саттер (Herb Sutter) предложил мне вести колонку и выбрать тему на мое усмотрение. Я согласился и решил назвать колонку «Общее знание». Герб представлял эту колонку как «систематический обзор основных профессиональных знаний, которыми должен располагать каждый практикующий программист на C++». Однако, сделав пару выпусков в этом ключе, я заинтересовался методиками метапрограммирования шаблонов, и темы, рассматриваемые в «Общем знании», с этого момента стали далеко не такими «общими».

Проблема индустрии программирования на C++, обусловившая выбор темы, осталась. В своей практике преподавателя и консультанта я сталкиваюсь со следующими типами личностей:

- программисты, в совершенстве владеющие C, но имеющие только базовые знания C++ и, возможно, некоторую неприязнь к нему;

- талантливые новобранцы, прямо с университетской скамьи, имеющие глубокую теоретическую подготовку в C++, но небольшой опыт работы с этим языком;
- высококвалифицированные Java-программисты, имеющие небольшой опыт работы с C++ и склонные программировать на C++ в стиле Java;
- программисты на C++ с опытом сопровождения готовых приложений на C++, у которых не было необходимости изучать что-либо сверх того, что требуется для сопровождения кода.

Хотелось бы сразу перейти к делу, но многим из тех, с кем я сотрудничаю или кого обучаю, требуется предварительное изучение различных возможностей языка программирования C++, шаблонов и методик написания кода. Хуже того, я подозреваю, что большая часть кода на C++ написана в неведении о некоторых из этих основ и, следовательно, не может быть признана специалистами качественным продуктом.

Данная книга направлена на решение этой всепроникающей проблемы. В ней основные знания, которыми должен обладать каждый профессиональный программист на C++, представлены в такой форме, что могут быть эффективно и четко усвоены. Значительная часть материала доступна в других источниках или уже входит в неписанные рекомендации, известные всем экспертам в C++. Преимущество книги в том, что вся информация, в отборе которой я руководствовался своим многолетним опытом преподавания и консультирования, расположена компактно.

Очень может быть, что самое важное в шестидесяти трех коротких темах, составляющих данную книгу, заключается в том, что осталось вне рассмотрения, а не в том, что они содержат. Многие из этих вопросов потенциально очень сложны. Если бы автор не был осведомлен об этих сложностях, то недостоверные сведения ввели бы читателя в заблуждение; в то же время профессиональное обсуждение вопроса с полным представлением всех сложных аспектов могло бы его запутать. Все слишком сложные вопросы были отсеяны. Надеюсь, то, что осталось, представляет собой квинтэссенцию знаний, необходимых для продуктивного программирования на C++. Знатоки заметят, что я не рассматриваю некоторые вопросы, интересные и даже важные с теоретической точки зрения, незнание которых, однако, как правило, не влияет на способность читать и писать код на C++.

Другим толчком для написания данной книги стала моя беседа с группой известных экспертов в C++ на одной из конференций. Все они были настроены мрачно, считая современный C++ слишком сложным и потому недоступным для понимания «среднестатистическим» программистом. (Конкретно речь шла о связывании имен в контексте шаблонов и пространств имен. Да, если ты раздражаешься по такому поводу, значит, пора

больше общаться с нормальными людьми.) Поразмыслив, должен сказать, что наша позиция была высокомерной, а пессимизм – неоправданным. У нас, «экспертов», нет таких проблем, и программировать на C++ так же просто, как говорить на (намного более сложном) естественном языке, даже если ты не можешь схематически представить глубинную структуру каждого своего высказывания. Основная идея данной книги: даже если полное описание нюансов конкретной возможности языка выглядит устрашающе, ее рутинное использование вполне может быть бесхитростным и естественным.

Возьмем перегрузку функций. Полное описание занимает немалую часть стандарта и одну или несколько глав многих учебников по C++. И тем не менее, столкнувшись с таким кодом

```
void f( int );  
void f( const char * );  
//...  
f( "Hello" );
```

любой практикующий программист на C++ безошибочно определит, какая *f* вызывается. Знание всех правил разрешения вызова перегруженной функции полезно, но очень редко бывает необходимым. То же самое можно сказать о многих других якобы сложных областях и идиомах языка программирования C++.

Сказанное не означает, что вся книга проста; «она проста настолько, насколько это возможно, но не проще». В программировании на C++, как в любой другой достойной внимания интеллектуальной деятельности, многие важные детали нельзя уместить на карточке картотеки. Более того, эта книга не для «чайников». Я чувствую огромную ответственность перед теми, кто тратит свое драгоценное время на чтение моих книг. Я уважаю этих людей и стараюсь общаться с ними как с коллегами. Нельзя писать для профессионалов, как для восьмиклассников, потому что это чистая профанация.

Многие темы книги опровергают заблуждения, с которыми я многократно сталкивался и на которые просто надо обратить внимание (например, посвященные порядку областей видимости при поиске функции-члена, разнице между переопределением и перегрузкой). В других обсуждаются вопросы, знание которых постепенно становится обязательным для профессионалов C++, но нередко ошибочно считающиеся сложными и потому замалчиваемые (например, частичная специализация шаблонов класса и параметры-шаблоны шаблонов). Я был раскритикован экспертами, рецензировавшими рукопись этой книги, за то, что слишком много внимания (примерно треть книги) уделил вопросам шаблонов, которые не относятся к общим знаниям. Однако каждый из этих специалистов указал

один, два и более вопросов по шаблонам, которые, по их мнению, следовало включить в книгу. Примечательно то, что предлагаемые ими вопросы немного перекрывались. В итоге каждый из рассмотренных мною аспектов шаблонов получил, по крайней мере, одного сторонника.

В этом и состояла главная трудность отбора тем для данной книги. Вряд ли найдутся читатели, абсолютно не сведущие ни в одном из рассматриваемых вопросов. Скорее всего, некоторым из них будет хорошо знаком весь материал. Очевидно, что если читатель не знает конкретной темы, то ему было бы полезно (я так думаю) изучить ее. Но рассмотрение даже знакомого вопроса под новым углом способно развеять некоторые заблуждения или помочь глубже понять вопрос. А более опытным программистам на C++ книга поможет сберечь драгоценное время, которое они нередко вынуждены (как уже отмечалось) отрывать от своей работы, без конца отвечая на одни и те же вопросы. Я предлагаю интересующимся сначала прочитать книгу, а потом спрашивать, и думаю, что это поможет направить усилия специалистов на решение сложных проблем, туда, где это действительно необходимо.

Сначала я пытался четко сгруппировать шестьдесят три темы в главы, но испытал неожиданные затруднения. Темы стали объединяться самовольно – иногда вполне предсказуемо, а иногда совершенно неожиданным образом. Например, темы, посвященные исключениям и управлению ресурсами, образуют довольно естественную группу. Взаимосвязь тем «Запросы возможностей», «Смысл сравнения указателей», «Виртуальные конструкторы и Прототип», «Фабричный метод» и «Ковариантные возвращаемые типы» хотя и тесна, но несколько неожиданна. «Арифметику указателей» я решил представить вместе с «Умными указателями», а не с изложенным ранее материалом по указателям и массивам. Вместо того чтобы навязать жесткую группировку тем по главам, я предоставил читателю свободу ассоциации. Конечно, между темами, которые могли бы быть расположены в простом линейном порядке, существует масса других взаимосвязей, поэтому в темах делается множество внутренних ссылок друг на друга. Это разбитое на группы, но взаимосвязанное сообщество.

Хотя основной идеей книги является краткость, обсуждение темы иногда включает дополнительные детали, не касающиеся непосредственно рассматриваемого вопроса. Эти подробности не всегда относятся к теме дискуссии, но читателю сообщается об определенной возможности или методике. Например, шаблон `Heap`, появляющийся в нескольких темах, мимоходом информирует читателя о существовании полезных, но редко рассматриваемых алгоритмов STL работы с кучей. Обсуждение синтаксиса размещения `new` представляет техническую базу сложных методик управления буферами, широко используемых стандартной библиотекой. Также везде, где это казалось уместным, я пытался включить обсуждение

вспомогательных вопросов в рассмотрение конкретной проблемы. Поэтому тема «Методика RAII» содержит краткое обсуждение порядка вызова конструктора и деструктора, в теме «Логический вывод аргументов шаблона» обсуждается применение вспомогательных функций для специализации шаблонов классов, а «Присваивание и инициализация – не одно и то же» включает рассмотрение вычислительных конструкторов. В этой книге запросто могло бы быть в два раза больше тем. Но и группировка самих тем, и связь вспомогательных вопросов с конкретной темой помещают проблему в контекст и помогают читателю эффективно усваивать материал.

Я с неохотой включил несколько вопросов, которые не могут быть рассмотрены корректно в формате коротких тем, принятом в данной книге. В частности, вопросы шаблонов проектирования и проектирования стандартной библиотеки шаблонов представлены смехотворно кратко и неполно. Но все же они вошли сюда, просто чтобы положить конец некоторым распространенным заблуждениям, подчеркнуть свою важность и подтолкнуть читателя к более глубокому изучению.

Традиционные примеры являются частью нашей культуры программирования, как истории, которые рассказывают на семейных праздниках. Поэтому здесь появляются Shape, String, Stack и далее по списку. Всеобщее понимание этих базовых примеров обеспечивает тот же эффект при общении, что и паттерны проектирования. Например, «Предположим, я хочу вращать Shape, кроме...» или «При конкатенации двух String...». Простое упоминание обычного примера определяет направление беседы и устраняет необходимость длительного обсуждения предпосылок.

В отличие от моей предыдущей книги, здесь я пытаюсь избегать критики плохих практик программирования и неверного использования возможностей языка C++. Пусть этим занимаются другие книги, лучшие из которых я привел в списке литературы. (Однако мне не вполне удалось избежать менторского тона; некоторые плохие практики программирования просто необходимо упомянуть, хотя бы вскользь.) Цель данной книги – рассказать читателю о технической сущности производственного программирования на C++ максимально эффективным способом.

Стивен С. Дьюхерст,
Карвер, Массачусетс, январь 2005

Благодарности

Питер Гордон (Peter Gordon) – редактор от Бога и экстраординарная личность – удивительно долго выдерживал мое нытье по поводу состояния образования в сообществе разработчиков на C++, пока не предложил мне самому попытаться что-то сделать. В результате появилась эта книга. Ким Бодигхеймер (Kim Boedigheimer) как-то сумел проконтролировать весь проект, ни разу не оказав существенного давления на автора.

Опытные технические редакторы – Мэтью Джонсон (Matthew Johnson), Моатаз Кэмел (Moataz Kamel), Дэн Сакс (Dan Saks), Кловис Тондо (Clovis Tondo) и Мэтью Вилсон (Matthew Wilson) – нашли несколько ошибок и множество погрешностей в языке рукописи, чем помогли сделать эту книгу лучше. Но я упрямец и последовал не всем рекомендациям, поэтому любые ошибки или погрешности языка – полностью моя вина.

Некоторые материалы данной книги появлялись в немного иной форме в моей колонке «Общее знание» журнала «C/C++ Users Journal», и многое из представленного здесь можно найти в веб-колонке «Once, Weakly» по адресу semantics.org. Я получил множество развернутых комментариев как на печатные, так и на опубликованные в сети статьи от Чака Эллисона (Chuck Allison), Аттилы Фахира (Attila Feher), Келвина Хенни (Kevin Henney), Торстена Оттосена (Thorsten Ottosen), Дэна Сакса, Тери Слеттебо (Terje Slettebo), Герба Саттера (Herb Sutter) и Леора Золмана (Leor Zolman). Некоторые глубокие дискуссии с Дэном Саксом улучшили мое понимание разницы между специализацией и созданием экземпляра шаблона и помогли прояснить различие между перегрузкой и представлением перегрузки при ADL и поиске инфиксного оператора.

Я в долгу перед Брэндоном Голдфеддером (Brandon Goldfedder) за аналогию алгоритмов и шаблонов, проводимую в теме, посвященной шаблонам проектирования, и перед Кловисом Тондо за мотивирование и помощь

в поиске квалифицированных рецензентов. Мне повезло в течение многих лет вести курсы на базе книг Скотта Мейерса [5, 6, 7], что позволило из первых рук узнать, какую информацию обычно упускают учащиеся, использующие эти отражающие промышленный стандарт книги по C++ для специалистов среднего уровня. Эти наблюдения помогли сформировать набор тем данной книги. Работа Андрея Александреску (Andrei Alexandrescu) вдохновила меня на эксперименты с метапрограммированием шаблонов. А работа Герба Саттера и Джека Ривза (Jack Reeves), посвященная исключениям, помогла лучше понять их использование.

Я также хотел бы поблагодарить моих соседей и хороших друзей Дика и Джуди Ворд (Dick, Judy Ward), которые периодически выгоняли меня из-за компьютера, чтобы поработать на сборе урожая клюквы. Тому, чья профессиональная деятельность связана преимущественно с упрощенными абстракциями реальности, полезно увидеть, что убедить клюквенный куст плодоносить настолько же сложно, как все то, что может делать программист на C++ с частичной специализацией шаблона.

Сара Дж. Хьюинс (Sarah G. Hewins) и Дэвид Р. Дьюхерст (David R. Dewhurst), как всегда, обеспечили этому проекту одновременно и неоценимую поддержку, и крайне необходимые препятствия.

Мне нравится считать себя спокойным человеком с устойчивыми привычками, больше предрасположенным к спокойному созерцанию, чем громкому предъявлению требований. Однако, подобно тем, кто претерпевает трансформацию личности, оказавшись за рулем автомобиля, закончив рукопись, я стал совершенно другим человеком. Замечательная команда специалистов по модификации поведения издательства Addison-Wesley помогла мне преодолеть эти личностные проблемы. Чанда Лери-Коту (Chanda Leary-Coutu) с Питером Гордоном и Кимом Бодигхеймером работали над переводом моих разглагольствований в рациональные бизнес-предложения и утверждением их у сильных мира сего. Молли Шарп (Molly Sharp) и Джулия Нагил (Julie Nahil) не только превратили неудобный документ Word в лежащие перед вами аккуратные страницы, но и умудрились устранить множество недостатков рукописи, позволив мне при этом сохранить архаичную структуру своих предложений, необычный стиль и характерную расстановку переносов. Несмотря на мои постоянно меняющиеся запросы, Ричард Эванс (Richard Evans) смог уложиться в график и создать предметный указатель. Чути Прасертсис (Chuti Prasertsith) разработала великолепную обложку с клюквенными мотивами. Всем огромное спасибо.

Принятые обозначения

Как упоминалось в предисловии, в этой книге много перекрестных ссылок. При ссылке на какую-либо тему приводится и ее номер, и полное название, чтобы читателю не приходилось постоянно сверяться с оглавлением с целью понять, чему посвящена данная тема. Например, ссылка «64 „Ешьте свои овощи“» говорит нам, что тема под названием «Ешьте свои овощи» имеет порядковый номер 64.

Примеры кода выделены моноширинным шрифтом, чтобы их можно было отличить от остального текста. Примеры неверного или нерекондуемого кода выделены серым фоном. Код без ошибок представлен без фона.

Тема 1 | Абстракция данных

Для программиста *тип данных* – это набор операций, а *абстрактный тип* – это набор операций с реализацией. Идентифицируя объекты в предметной области, мы прежде всего должны интересоваться, что можно сделать с помощью этого объекта, а не как он реализован. Следовательно, если в описании задачи присутствуют служащие, контракты и платежные ведомости, то выбранный язык программирования должен содержать типы `Employee` (Служащий), `Contract` (Контракт) и `PayrollRecord` (Платежная ведомость). Тем самым обеспечивается эффективная двусторонняя трансляция между предметной областью и областью решения. Код, написанный таким образом, содержит меньше «трансляционного шума», он проще и в нем меньше ошибок.

В языках программирования общего назначения, к которым относится C++, нет специальных типов, таких как `Employee`. Зато есть кое-что получше, а именно средства для создания сложных абстрактных типов данных. По сути, назначение абстрактного типа данных состоит в расширении языка программирования в соответствии с нуждами конкретной предметной области.

В C++ нет универсальной процедуры проектирования абстрактных типов данных. В этой области программирования все еще есть место для творчества и вдохновения. Однако самые успешные подходы подразумевают выполнение похожих шагов.

1. Типу присваивается описательное имя. Если найти имя трудно, значит, нет достаточной информации о том, что именно предполагается реализовывать. Необходим дополнительный анализ. Абстрактный тип данных должен представлять одно четко определенное понятие, и имя этого понятия должно быть очевидным.

2. Перечисляются операции, которые может осуществлять тип. Абстрактный тип данных определяется тем, что можно сделать с его помощью. Нельзя забывать об инициализации (конструкторы), очистке ресурсов (деструктор), копировании (операции копирования) и преобразованиях (неявные конструкторы с одним аргументом и операторы преобразования). Ни в коем случае нельзя просто определять набор операций `get/set` над элементами данных реализации. Это не абстракция данных, а лень и недостаток воображения.
3. Проектируется интерфейс типа. Как говорит Скотт Мейерс (Scott Meyers), тип должен «помогать программировать правильно и мешать программировать неправильно». Абстрактный тип данных расширяет язык программирования. По сути, надо спроектировать язык. Попробуйте поставить себя на место пользователя этого типа и напишите немного кода, задействовав свой интерфейс. Правильная конструкция интерфейса – это вопрос как психологии и взаимопонимания, так и проявления технического мастерства.
4. Тип реализуется. Реализация не должна оказывать влияние на интерфейс типа. Должен реализовываться контракт, оговоренный интерфейсом типа. Необходимо помнить, что реализации большинства абстрактных типов данных будут меняться намного чаще, чем их интерфейсы.

Тема 2 | Полиморфизм

В одних книгах по программированию полиморфизму приписывается мистический статус, другие о полиморфизме молчат вовсе, а на самом деле это простая и полезная концепция, поддерживаемая языком C++. Согласно стандарту *полиморфный тип* – это класс, имеющий виртуальную функцию. С точки зрения проектирования *полиморфный объект* – это объект, имеющий более одного типа. И *полиморфный базовый класс* – это базовый класс, спроектированный для использования полиморфными объектами.

Рассмотрим тип финансового опциона AmOption (Американский опцион), представленный на рис. 2.1.

У объекта AmOption четыре типа: он одновременно выступает в ипостасях AmOption, Option (Опцион), Deal (Сделка) и Priceable (Подлежащий оплате).

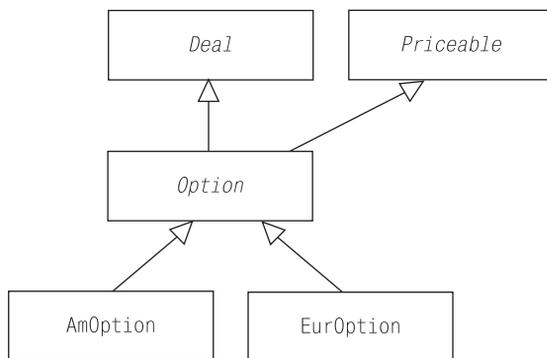


Рис. 2.1. Полиморфизм в иерархии финансового опциона. У Американского опциона четыре типа

Поскольку тип – это набор операций (см. темы 1 «*Абстракция данных*» и 27 «*Запросы возможностей*»), с объектом `AmOption` можно работать посредством одного из его четырех интерфейсов. Это означает, что объектом `AmOption` может управлять код, написанный для интерфейсов `Deal`, `Priceable` и `Option`. Таким образом, реализация `AmOption` может применять и повторно использовать весь этот код. Для полиморфного типа, такого как `AmOption`, самым важным из наследуемого от базовых классов являются интерфейсы, а не реализации. Часто бывает желательно, чтобы базовый класс не включал ничего, кроме интерфейса (см. тему 27 «*Запросы возможностей*»).

Конечно, здесь есть своя тонкость. Чтобы эта иерархия классов работала, полиморфный класс должен уметь замещать любой из своих базовых классов. Другими словами, если универсальный код, написанный для интерфейса `Option`, получает объект `AmOption`, этот объект должен вести себя как `Option`!

Речь идет не о том, что `AmOption` должен вести себя идентично `Option`. (Прежде всего, многие операции базового класса `Option` нередко представляют собой чисто виртуальные функции без реализации.) Лучше рассматривать полиморфный базовый класс (`Option`) как контракт. Базовый класс дает определенные обещания пользователям его интерфейса: сюда входят твердые синтаксические гарантии вызова определенных функций-членов с определенными типами аргументов, а также обещания, которые сложнее проверить, касающиеся того, что на самом деле произойдет при вызове конкретной функции-члена. Конкретные производные классы, такие как `AmOption` и `EurOption` (Европейский опцион), представляют собой субконтракты, которые реализуют контракт, устанавливаемый классом `Option` с его клиентами, как показано на рис. 2.2.

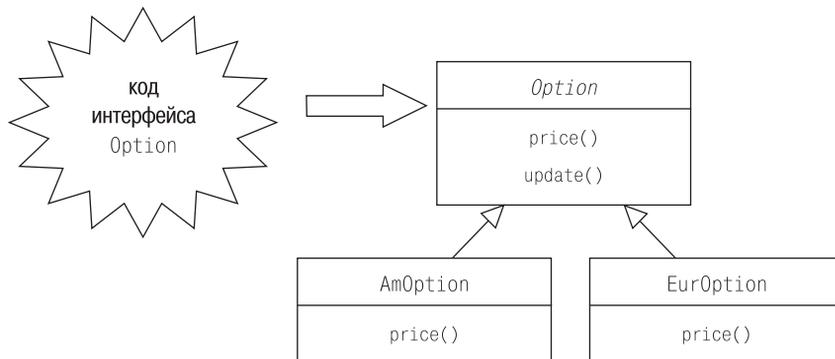


Рис. 2.2. *Полиморфный контракт и его субконтракты. Базовый класс `Option` определяет контракт*

Например, если в классе `Option` есть чисто виртуальная функция-член `price` (цена), вычисляющая текущее значение `Option`, то оба класса, `AmOption` и `EurOption`, должны реализовывать эту функцию. Очевидно, что в этих двух типах `Option` ее поведение не будет идентичным, но она должна вычислять и возвращать цену, а не звонить по телефону или распечатывать файл.

С другой стороны, если вызвать функцию `price` двух разных интерфейсов *одного* объекта, результат должен быть одним и тем же. По сути, любой вызов должен быть связан с одной и той же функцией:

```
AmOption *d = new AmOption;
Option *b = d;
d->price(); // если здесь вызывается AmOption::price...,
b->price(); // ...то же самое должно происходить здесь!
```

Это логично. (Просто удивительно, но углубленные аспекты объектно-ориентированного программирования по большей части есть не что иное, как проявление здравого смысла, скрытое завесой синтаксиса.) Вопросы: «Каково текущее значение этого Американского опциона?» и «Каково текущее значение этого опциона?», по моему мнению, требовали бы одного ответа.

Аналогичные рассуждения, конечно же, применимы и к не виртуальным функциям объектов:

```
b->update(); // если здесь вызывается Option::update...,
d->update(); // ... то же самое должно происходить здесь!
```

Контракт, предоставляемый базовым классом, — это то, что позволяет «полиморфному» коду интерфейса базового класса работать с конкретными опционами, оставаясь при этом в полезном неведении об их существовании. Иначе говоря, полиморфный код может управлять объектами `AmOption` и `EurOption`, рассматривая их как объекты класса `Option`. Различные конкретные типы могут добавляться или удаляться без всякого воздействия на универсальный код, который знает только о базовом классе `Option`. Если в какой-то момент появится `AsianOption` (Азиатский опцион), полиморфный код, знающий только `Option`, сможет работать с ним в блаженном неведении о его конкретном типе. И если позже этот класс исчезнет, его отсутствие не будет замечено.

Справедливо и то, что конкретным типам опционов, таким как `AmOption` и `EurOption`, необходимо знать только о базовых классах, чьи контракты они реализуют. Они не зависят от изменений универсального кода. В принципе базовый класс может знать только о себе. Практически конструкция его интерфейса будет учитывать требования предполагаемых пользователей. Он должен проектироваться таким образом, чтобы произ-

водные классы могли без труда проследить и реализовать его контракт (см. тему 22 «*Шаблонный метод*»). Однако у базового класса не должно быть никакой конкретной информации о любом из его производных классов, потому что это неизбежно усложнит добавление или удаление производных классов в иерархии.

В объектно-ориентированном проектировании, как и в жизни, действует правило «много будешь знать – скоро состаришься» (см. также темы 29 «*Виртуальные конструкторы и Прототип*» и 30 «*Фабричный метод*»).