Содержание

Об авторе 12

О рецензенте 13

Предисловие 14

Глава 1. Введение в наследование и полиморфизм 20

Классы и объекты 20
Наследование и иерархии классов 22
Полиморфизм и виртуальные функции 27
Множественное наследование 31
Резюме 33
Вопросы 33
Для дальнейшего чтения 33

Глава 2. Шаблоны классов и функций 34

Шаблоны в C++ 34
  Шаблоны функций 35
  Шаблоны классов 35
  Шаблоны переменных 36
  Параметры шаблонов, не являющиеся типами 36
Конкретизация шаблона 37
  Шаблоны функций 38
  Шаблоны классов 41
Специализация шаблона 42
  Явная специализация 43
  Частичная специализация 44
Перегрузка шаблонных функций 47
Шаблоны с переменным числом аргументов 50
Лямбда-выражения 54
Резюме 58
Вопросы 58
Для дальнейшего чтения 58

Глава 3. Владение памятью 59

Технические требования 59
Что такое владение памятью? 59
  Правильно спроектированное владение памятью 60

Плохо спроектированное владение памятью....61
Выражение владения памятью в C++....62
Выражения невладения....63
Выражение монопольного владения....64
Выражение передачи монопольного владения....65
Выражение совместного владения....66
Резюме....68
Вопросы....68
Для дальнейшего чтения....69

Глава 4. От простого к нетривиальному....70

Технические требования....70
Обмен и стандартная библиотека шаблонов....70
Обмен и контейнеры STL....71
Свободная функция swap....73
Обмен как в стандарте....74
Когда и для чего использовать обмен....75
Обмен и безопасность относительно исключений....75
Другие распространенные идиомы обмена....77
Как правильно реализовать и использовать обмен....78
Реализация обмена....78
Правильное использование обмена....82
Резюме....83
Вопросы....84

Глава 5. Все о захвате ресурсов как инициализации....85

Технические требования....85
Управление ресурсами в C++....86
Установка библиотеки эталонного микротестирования....86
Установка Google Test....87
Подсчет ресурсов....87
Опасности ручного управления ресурсами....88
Ручное управление ресурсами чревато ошибками....88
Управление ресурсами и безопасность относительно исключений....91
Идиома RAII....93
RAII в двух словах....93
RAII для других ресурсов....97
Досрочное освобождение....98
Аккуратная реализация RAII-объектов....101
Недостатки RAII....104
Резюме....106
Вопросы....106
Для дальнейшего чтения....107

Глава 6. Что такое стирание типа 108
Технические требования 108
Что такое стирание типа? 108
Стирание типа на примере 109
Как стирание типа реализовано в C++? 112
Очень старый способ стирания типа 112
Объектно-ориентированное стирание типа 113
Противоположность стиранию типа 116
Стирание типа в C++ 117
Когда использовать стирание типа, а когда избегать его 119
Стирание типа и проектирование программ 119
Установка библиотеки эталонного микротестирования 121
Издержки стирания типа 121
Резюме 123
Вопросы 124

Глава 7. SFINAE и управление разрешением перегрузки 125
Технические требования 125
Разрешение перегрузки и множество перегруженных вариантов 125
Перегрузка функций в C++ 126
Шаблонные функции 129
Подстановка типов в шаблонных функциях 131
Выведение и подстановка типов 132
Неудавшаяся подстановка 133
Неудавшаяся подстановка – не ошибка 135
Управление разрешением перегрузки 137
Простое применение SFINAE 138
Продвинутое применение SFINAE 140
Еще раз о продвинутом применении SFINAE 150
SFINAE без компромиссов 155
Резюме 160
Вопросы 161
Для дальнейшего чтения 161

Глава 8. Рекурсивный шаблон 162
Технические требования 162
Укладываем CRTP в голове 162
Что не так с виртуальной функцией? 163
Введение в CRTP 165
CRTP и статический полиморфизм 168
Полиморфизм времени компиляции 168
Чисто виртуальная функция времени компиляции 170
Деструкторы и полиморфное удаление 171

CRTP и управление доступом ... 173
CRTP как паттерн делегирования ... 174
Расширение интерфейса ... 175
Резюме ... 180
Вопросы ... 180

Глава 9. Именованные аргументы и сцепление методов ... 181

Технические требования ... 181
Проблема аргументов ... 181
Что плохого в большом количестве аргументов? ... 182
Агрегатные параметры ... 185
Именованные аргументы в C++ ... 187
Сцепление методов ... 188
Сцепление методов и именованные аргументы ... 188
Производительность идиомы именованных аргументов ... 191
Сцепление методов в общем случае ... 194
Сцепление и каскадирование методов ... 194
Сцепление методов в общем случае ... 195
Сцепление методов в иерархиях классов ... 196
Резюме ... 198
Вопросы ... 199

Глава 10. Оптимизация локального буфера ... 200

Технические требования ... 200
Издержки выделения небольших блоков памяти ... 200
Стоимость выделения памяти ... 201
Введение в оптимизацию локального буфера ... 204
Основная идея ... 204
Эффект оптимизации локального буфера ... 206
Дополнительные оптимизации ... 209
Оптимизация локального буфера в общем случае ... 209
Короткий вектор ... 210
Объекты со стертым типом и вызываемые объекты ... 212
Оптимизация локального буфера в библиотеке C++ ... 215
Недостатки оптимизации локального буфера ... 216
Резюме ... 217
Вопросы ... 217
Для дальнейшего чтения ... 217

Глава 11. Охрана области видимости ... 218

Технические требования ... 218
Обработка ошибок и идиома RAII ... 219
Безопасность относительно ошибок и исключений ... 219
Захват ресурса есть инициализация ... 222

Паттерн ScopeGuard....225
Основы ScopeGuard....226
ScopeGuard в общем виде....231
ScopeGuard и исключения....236
Что не должно возбуждать исключения....236
ScopeGuard, управляемый исключениями....239
ScopeGuard со стертым типом....243
Резюме....246
Вопросы....246

Глава 12. Фабрика друзей....247
Технические требования....247
Друзья в C++....247
Как предоставить дружественный доступ в C++....247
Друзья и функции-члены....248
Друзья и шаблоны....252
Друзья шаблонов классов....252
Фабрика друзей шаблона....255
Генерация друзей по запросу....255
Фабрика друзей и Рекурсивный шаблон....257
Резюме....259
Вопросы....260

Глава 13. Виртуальные конструкторы и фабрики....261
Технические требования....261
Почему конструкторы не могут быть виртуальными....261
Когда объект получает свой тип?....262
Паттерн Фабрика....265
Основа паттерна Фабричный метод....265
Фабричные методы с аргументами....266
Динамический реестр типов....267
Полиморфная фабрика....270
Похожие на Фабрику паттерны в C++....272
Полиморфное копирование....272
CRTP-фабрика и возвращаемые типы....273
CRTP-фабрика с меньшим объемом копирования и вставки....274
Резюме....276
Вопросы....277

Глава 14. Паттерн Шаблонный метод и идиома невиртуального интерфейса....278
Технические требования....278
Паттерн Шаблонный метод....279

Шаблонный метод в C++ 279
Применения Шаблонного метода 280
Предусловия, постусловия и действия 282
Невиртуальный интерфейс 283
Виртуальные функции и контроль доступа 283
Идиома NVI в C++ 285
Замечание о деструкторах 287
Недостатки невиртуального интерфейса 288
Компонуемость 288
Проблема хрупкого базового класса 289
Резюме 291
Вопросы 291
Для дальнейшего чтения 291

Глава 15. Одиночка – классический объектно-ориентированный паттерн 292

Технические требования 292
Паттерн Одиночка – для чего он предназначен, а для чего – нет 292
Что такое Одиночка? 293
Когда использовать паттерн Одиночка 294
Типы Одиночек 297
Статический Одиночка 299
Одиночка Мейерса 301
Утекающие Одиночки 308
Резюме 310
Вопросы 311

Глава 16. Проектирование на основе политик 312

Технические требования 312
Паттерн Стратегия и проектирование на основе политик 312
Основы проектирования на основе политик 313
Реализация политик 319
Использование объектов политик 322
Продвинутое проектирование на основе политик 329
Политики для конструкторов 329
Применение политик для тестирования 337
Адаптеры и псевдонимы политик 339
Применение политик для управления открытым интерфейсом 341
Перепривязка политики 347
Рекомендации и указания 349
Достоинства проектирования на основе политик 349
Недостатки проектирования на основе политик 350
Рекомендации по проектированию на основе политик 352

Почти политики ... 354
Резюме ... 360
Вопросы ... 361

Глава 17. Адаптеры и Декораторы ... 362

Технические требования ... 362
Паттерн Декоратор ... 362
  Основной паттерн Декоратор ... 363
  Декораторы на манер C++ ... 366
  Полиморфные декораторы и их ограничения ... 371
  Компонуемые декораторы ... 373
Паттерн Адаптер ... 375
  Основной паттерн Адаптер ... 375
  Адаптеры функций ... 378
  Адаптеры времени компиляции ... 381
Адаптер и Политика ... 384
Резюме ... 388
Вопросы ... 389

Глава18. Паттерн Посетитель и множественная диспетчеризация ... 390

Технические требования ... 390
Паттерн Посетитель ... 391
  Что такое паттерн Посетитель? ... 391
  Простой Посетитель на C++ ... 393
  Обобщения и ограничения паттерна Посетитель ... 397
Посещение сложных объектов ... 401
  Посещение составных объектов ... 401
  Сериализация и десериализация с помощью Посетителя ... 403
Ациклический посетитель ... 409
Посетители в современном C++ ... 412
  Обобщенный посетитель ... 412
  Лямбда-посетитель ... 414
  Обобщенный Ациклический посетитель ... 418
Посетитель времени компиляции ... 421
Резюме ... 427
Вопросы ... 428

Ответы на вопросы ... 429

Предметный указатель ... 448

Об авторе

Федор Г. Пикус – главный конструктор в проектном отделе компании Mentor Graphics (подразделение Siemens), он отвечает за перспективное техническое планирование линейки продуктов Calibre, проектирование архитектуры программного обеспечения и исследование новых технологий. Ранее работал старшим инженером-программистом в Google и главным архитектором ПО в Mentor Graphics. Федор – признанный эксперт по высокопроизводительным вычислениям и C++. Он представлял свои работы на конференциях CPPCon, SD West, DesignCon и в журналах по разработке ПО, также является автором издательства O'Reilly. Федор – обладатель более 25 патентов и автор свыше 100 статей и докладов на конференциях по физике, автоматизации проектирования, электронике, проектированию ПО и C++.

Эта книга не появилась бы на свет без поддержки моей жены Галины, которая заставляла меня двигаться дальше в минуты сомнений в собственных силах. Спасибо моим сыновьям Аарону и Бенджамину за энтузиазм и моему коту Пушку, который разрешал использовать свою подстилку в качестве моего ноутбука

О рецензенте

Кэйл Данлэп (Cale Dunlap) начал писать код на разных языках еще в старших классах, когда в 1999 году разработал свою первую моду для видеоигры Half-Life. В 2002 году он стал соразработчиком более-менее популярной моды Firearms для Half-Life и в конечном итоге способствовал переносу этой моды в ядро игры Firearms-Source. Получил профессиональный диплом по компьютерным информационным системам, а затем степень бакалавра по программированию игр и имитационному моделированию. С 2005 года работал программистом в небольших компаниях, участвуя в разработке различных программ, начиная с веб-приложений и заканчивая моделированием в интересах военных. В настоящее время работает старшим разработчиком в креативном агентстве Column Five в городе Оранж Каунти, штат Калифорния.

Спасибо моей невесте Элизабет, нашему сыну Мэйсону и всем остальным членам семьи, которые поддерживали меня, когда я писал свою первую рецензию на книгу

Предисловие

Еще одна книга по паттернам проектирования в C++? Зачем и почему именно сейчас? Разве написано еще не все, что можно сказать о паттернах?

Есть несколько причин для написания еще одной книги по *паттернам проектирования*, но прежде всего эта книга о C++ – не о *паттернах проектирования* в C++, а о паттернах проектирования *в C++*, и это различие в акцентах очень важно. C++ обладает всеми возможностями традиционного объектно-ориентированного языка, поэтому на нем можно реализовать все классические объектно-ориентированные паттерны, например Фабрику и Стратегию. Некоторые из них рассматриваются в этой книге. Но мощь C++ в полной мере раскрывается при использовании его средств обобщенного программирования. Напомним, что паттерн проектирования – это как часто встречающаяся задача проектирования, так и ее общепринятое решение, и обе эти грани одинаково важны. Понятно, что при появлении новых инструментов открывается возможность для нового решения. Со временем сообщество выбирает из этих решений наиболее предпочтительное, и тогда появляется на свет новый вариант старого паттерна проектирования – задача та же, но предпочтительное решение иное. Однако расширение возможностей также раздвигает границы – коль скоро в нашем распоряжении появляются новые инструменты, возникают и новые задачи проектирования.

В этой книге наше внимание будет обращено на те паттерны проектирования, для которых C++ может привнести нечто существенное хотя бы в одну из граней паттерна. С одной стороны, существуют паттерны, например Посетитель, для которых средства обобщенного программирования C++ позволяют предложить лучшее решение. Оно стало возможным благодаря новой функциональности, появившейся в последних версиях языка, от C++11 до C++17. С другой стороны, обобщенное программирование по-прежнему остается программированием (только выполнение программы производится на этапе компиляции), программирование нуждается в проектировании, а в проектировании возникают типичные проблемы, не так уж сильно отличающиеся от проблем традиционного программирования. Поэтому у многих традиционных паттернов есть близнецы или, по крайней мере, близкие родственники в обобщенном программировании, и именно они будут интересовать нас в этой книге. Характерный пример – паттерн Стратегия, который в обобщенном программировании больше известен под названием Политика (Policy). Наконец, в таком сложном языке, как C++, неизбежно присутствуют собственные идиосинкразии, которые часто приводят к специфическим для C++ проблемам, для которых имеются типичные, или *стандартные*, решения. Эти идиомы C++ не вполне заслуживают называться паттернами, но тоже рассматриваются в данной книге.

Итак, для написания этой книги было три основные причины:

- рассмотреть специфичные для C++ решения общих *классических* паттернов проектирования;
- продемонстрировать специфичные для C++ варианты паттернов, появляющиеся, когда старые задачи проектирования возникают в новом окружении обобщенного программирования;
- показать, как видоизменяются паттерны по мере эволюции языка.

Предполагаемая аудитория

Эта книга адресована программистам на C++, которые хотят почерпнуть из *коллективной мудрости сообщества* – от признанно хороших решений до часто встречающихся проблем проектирования. Можно сказать и по-другому: эта книга открывает для программиста возможность учиться на чужих ошибках.

Это не *учебник C++*; предполагается, что целевая аудитория состоит в основном из программистов, хорошо владеющих средствами и синтаксисом языка и интересующихся тем, как и почему эти средства следует использовать. Однако книга будет полезна и тем программистам, которые хотят больше узнать о C++, но предпочитают учиться на конкретных практических примерах (таким читателям я рекомендую держать под рукой какой-нибудь справочник по C++). Наконец, я надеюсь, что программисты, желающие узнать, не просто что нового появилось в версиях C++11, C++14 и C++17, а для чего эти новшества можно использовать, тоже найдут эту книгу интересной.

Структура книги

В главе 1 «Введение в наследование и полиморфизм» приводится краткое введение в объектно-ориентированные средства C++. Эта глава – не столько справочник по объектно-ориентированному программированию на C++, сколько описание аспектов языка, наиболее важных для последующих глав.

В главе 2 «Шаблоны классов и функций» кратко описываются средства обобщенного программирования в C++ – шаблоны классов, шаблоны функций и лямбда-выражения. Здесь рассмотрены конкретизации и специализации шаблонов, а также выведение аргументов и разрешение перегрузки шаблонной функции. И тут закладывается фундамент для более сложных применений шаблонов в последующих главах.

В главе 3 «Владение памятью» описываются современные идиоматические способы выражения различных видов владения памятью в C++. Это набор соглашений или идиом – компилятор не проверяет выполнение этих правил, но программистам проще понимать друг друга, если все пользуются общим словарем идиом.

В главе 4 «Обмен – от простого к нетривиальному» исследуется одна из основополагающих операций C++ – обмен двух значений. У этой операции на

удивление сложные взаимодействия с другими средствами C++, и они тоже обсуждаются здесь.

Глава 5 «Все о захвате ресурсов как инициализации» посвящена детальному разбору одной из фундаментальных концепций C++ – управлению ресурсами. Здесь вводится, пожалуй, самая популярная идиома C++, RAII (захват ресурса есть инициализация).

В главе 6 «Что такое стирание типа» обсуждается техника, которая существовала в C++ давно, но лишь с принятием стандарта C++11 завоевала популярность и приобрела важность. Механизм стирания типа позволяет писать абстрактные программы, в которых некоторые типы не упоминаются явно.

В главе 7 «SFINAE и управление разрешением перегрузки» рассматривается идиома C++ SFINAE, которая, с одной стороны, является важной составной частью механизма шаблонов в C++ и в этом смысле прозрачна для программиста, а с другой – для ее целенаправленного применения требуется ясное понимание тонкостей шаблонов.

В главе 8 «Рекурсивные шаблоны» описывается заковыристый паттерн, в котором достоинства объектно-ориентированного программирования сочетаются с гибкостью шаблонов. Объясняется идея шаблона и рассказывается, как правильно применять его для решения практических задач. Предполагается, что читатель будет готов распознать этот паттерн в последующих главах.

В главе 9 «Именованные аргументы и сцепление методов» рассматривается необычная техника вызова функций в C++ с использованием именованных аргументов вместо позиционных. Это еще одна идиома, которая неявно используется в каждой программе на C++, тогда как ее явное целенаправленное применение требует некоторых размышлений.

Глава 10 «Оптимизация локального буфера» – единственная в этой книге, целиком посвященная производительности. Производительность и эффективность – критически важные аспекты, учитываемые в каждом проектном решении, оказывающем влияние на сам язык, – ни одно языковое средство не включается в стандарт без всестороннего обсуждения с точки зрения эффективности. Поэтому неудивительно, что целая глава посвящена широко распространенной идиоме, призванной повысить производительность программ на C++.

В главе 11 «Охрана области видимости» описывается старый паттерн C++, который в последних версиях изменился почти до неузнаваемости. Речь идет о паттерне, который позволяет без труда писать безопасный относительно исключений и вообще безопасный относительно ошибок код на C++.

В главе 12 «Фабрика друзей» описывается старый паттерн, который нашел новые применения в современном C++. Он применяется для порождения функций, ассоциированных с шаблонами, например арифметических операторов для каждого порождаемого по шаблону типа.

В главе 13 «Виртуальные конструкторы и фабрики» рассматривается еще один классический объектно-ориентированный паттерн в C++ – Фабрика. По-

путно показано, как добиться видимости полиморфного поведения от конструкторов C++, хотя они и не могут быть виртуальными.

В главе 14 «Паттерн Шаблонный метод и идиома невиртуального интерфейса» описывается интересный гибрид классического объектно-ориентированного паттерна, шаблона и идиомы, специфичной только для C++. В совокупности получается паттерн, который описывает оптимальное использование виртуальных функций в C++.

В главе 15 «Одиночка – классический паттерн ООП» рассказано еще об одном классическом объектно-ориентированном паттерне, Одиночка, в контексте C++. Обсуждается, когда разумно применять этот паттерн, а когда следует его избегать. Демонстрируется несколько распространенных реализаций Одиночки.

Глава 16 «Проектирование на основе политик» посвящена одной из жемчужин проектирования в C++ – паттерну Политика (больше известному под названием Стратегия). Он применяется на этапе компиляции, т. е. является не объектно-ориентированным паттерном, а паттерном обобщенного программирования.

В главе 17 «Адаптеры и декораторы» обсуждаются два широко используемых и тесно связанных паттерна в контексте C++. Рассматривается их применение как в объектно-ориентированном, так и в обобщенном коде.

Глава 18 «Посетитель и множественная диспетчеризация» завершает галерею классических объектно-ориентированных паттернов неувядаемым паттерном Посетитель. Сначала объясняется сам паттерн, а затем рассматривается, как современный C++ позволяет реализовать его проще, надежнее и устойчивее к ошибкам.

Что необходимо для чтения этой книги

Для выполнения примеров из этой книги вам понадобится компьютер с операционной системой Windows, Linux или macOS (программы на C++ можно собирать даже на таком маленьком компьютере, как Raspberry Pi). Также понадобится современный компилятор C++, например GCC, Clang, Visual Studio или еще какой-то поддерживающий язык на уровне стандарта C++17. Необходимо также уметь работать на базовом уровне с GitHub и Git, чтобы клонировать проект, содержащий примеры.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте **www.dmkpress.com** или **www.дмк.рф** на странице с описанием соответствующей книги.

Код примеров из этой книги размещен также на сайте GitHub по адресу **https://github.com/PacktPublishing/Hands-On-Design-Patterns-with-CPP**. Все обновления выкладываются в репозиторий на GitHub.

В разделе **https://github.com/PacktPublishing/** есть и другие пакеты кода для нашего обширного каталога книг и видео. Не пропустите!

Обозначения и графические выделения

В этой книге применяется ряд соглашений о наборе текста.

`CodeInText`: код в тексте, имена таблиц базы данных, папок и файлов, расширения имен файлов, пути к файлам, данные, вводимые пользователем, и адреса в Твиттере. Например: «`overload_set` – *шаблон класса с переменным числом аргументов*».

Отдельно стоящие фрагменты кода набраны так:

```
template <typename T>
T increment(T x) { return x + 1; }
```

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте **www.dmkpress.com**, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты **dmkpress@gmail.com**.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Глава 1

Введение в наследование и полиморфизм

C++ – прежде всего объектно-ориентированный язык, и объекты – фундаментальные строительные блоки программы на C++. Для описания связей и взаимодействий между различными частями программной системы, для определения и реализации интерфейсов между компонентами и для организации данных и кода применяются иерархии классов. И хотя эта книга – не учебник по C++, цель настоящей главы – сообщить читателю достаточно информации о тех касающихся классов и наследования языковых средствах, которые будут использоваться в последующих главах. Мы не будем пытаться полностью описать все возможности C++ для работы с классами, а лишь дадим введение в понятия и конструкции языка, которые нам понадобятся.

В этой главе рассматриваются следующие вопросы:

- что такое классы и какую роль они играют в C++;
- что такое иерархии классов и как в C++ используется наследование;
- что такое полиморфизм времени выполнения и как он применяется в C++.

Классы и объекты

Объектно-ориентированное программирование – это способ структурировать программу, объединив алгоритм и данные, которыми он оперирует, в единую сущность, именуемую **объектом**. Большинство объектно-ориентированных языков, в том числе и C++, основано на классах. Класс – это определение объекта, он описывает алгоритм и данные, формат объекта и связи с другими классами. Объект – это конкретный экземпляр класса, т. е. переменная. У объекта есть адрес, по которому он расположен в памяти. Класс – это тип, определенный пользователем. Вообще говоря, по определению, предоставленному классом, можно создать сколь угодно много объектов (некоторые классы ограничивают количество своих объектов, но это исключение, а не правило).

В C++ данные, составляющие класс, организуются в виде набора данных-членов, т. е. переменных различных типов. Алгоритмы реализованы в виде

функций – методов класса. Язык не требует, чтобы данные-члены класса были как-то связаны с реализацией его методов, но одним из признаков правильного проектирования является хорошая инкапсуляция данных в классах и ограниченное взаимодействие методов с внешними данными.

Идея инкапсуляции является центральной для классов в C++ – язык позволяет управлять тем, какие данные-члены и методы открыты (public), т. е. видимы извне класса, а какие закрыты (private), т. е. являются внутренними для класса. В хорошо спроектированном классе многие данные-члены, или даже все они, закрыты, а для выражения открытого интерфейса класса, т. е. того, что он делает, нужны только открытые методы. Этот открытый интерфейс можно уподобить контракту – проектировщик класса обещает, что класс будет предоставлять определенные возможности и операции. Закрытые данные и методы класса – часть его реализации, они могут изменяться при условии, что открытый интерфейс, т. е. заключенный нами контракт, остается неизменным. Например, следующий класс представляет рациональное число и поддерживает операцию инкремента, что и выражено в его открытом интерфейсе:

```
class Rational {
    public:
    Rational& operator+=(const Rational& rhs);
};
```

Хорошо спроектированный класс не раскрывает больше деталей реализации, чем необходимо его открытому интерфейсу. Реализация не является частью контракта, хотя документированный интерфейс может налагать на нее некоторые ограничения. Например, если мы обещаем, что числитель и знаменатель рационального числа не имеют общих множителей, то операция сложения должна включать шаг их сокращения. Для этого очень пригодилась бы закрытая функция-член, которую могли бы вызывать другие операции, но клиенту класса вызывать ее никогда не пришлось бы, потому что любое рациональное число уже сделано неприводимым до передачи вызывающей программе:

```
class Rational {
    public:
    Rational& operator+=(const Rational& rhs);
    private:
    long n_;     // числитель
    long d_;     // знаменатель
    void reduce();
};
Rational& Rational::operator+=(const Rational& rhs) {
    n_ = n_*rhs.d_ + rhs.n_*d_;
    d_ = d_*rhs.d_;
    reduce();
    return *this;
}
Rational a, b;
a += b;
```

Методам класса разрешен специальный доступ к данным-членам – они могут обращаться к закрытым данным класса. Отметим различие между классом и объектом: `operator+=()` – метод класса `Rational`, но вызывается от имени объекта `a`. Однако этот метод имеет также доступ к закрытым данным объекта `b`, поскольку `a` и `b` – объекты одного класса. Если функция-член ссылается на член класса по имени, без указания дополнительных квалификаторов, значит, она обращается к члену того объекта, от имени которого вызвана (мы можем указать это явно, написав `this->n_` и `this->d_`). Для доступа к членам другого объекта того же класса необходимо добавить указатель или ссылку на этот объект, но больше он ничем не ограничен – в отличие от случая, когда запрашивается доступ к закрытому члену из функции, не являющейся членом класса.

Кстати говоря, C++ поддерживает и структуры в стиле языка C. Но в C++ структура – не просто агрегат данных-членов, она может иметь методы, модификаторы доступа `public` и `private` и все остальное, что есть в классах. С точки зрения языка, единственное различие между классом и структурой состоит в том, что все члены и методы класса по умолчанию закрыты, а в структуре они по умолчанию открыты. Если не считать этого нюанса, использовать структуры или классы – вопрос соглашения; традиционно ключевое слово `struct` применяется для описания структур в стиле C (т. е. таких, которые были бы допустимы в программе на C) и *почти* в стиле C, например структуры, в которую добавлен только конструктор. Конечно, эта граница подвижна и определяется стилем и практикой кодирования, принятыми в конкретном проекте или команде.

Помимо уже описанных методов и данных-членов, C++ поддерживает статические данные и методы. Статический метод похож на обычную функцию, не являющуюся членом, – он не вызывается от имени конкретного объекта, и единственный способ предоставить ему доступ к объекту, вне зависимости от типа, – передать объект в качестве аргумента. Однако, в отличие от свободной функции, не являющейся членом, статический метод сохраняет привилегированный доступ к закрытым данным класса.

Уже сами по себе классы – полезный способ сгруппировать алгоритмы с данными, которыми они манипулируют, и ограничить доступ к некоторым данным. Но свои богатейшие объектно-ориентированные возможности классы C++ получают благодаря наследованию и возникающим на его основе иерархиям классов.

Наследование и иерархии классов

Иерархии классов в C++ играют двоякую роль. С одной стороны, они позволяют выразить отношения между объектами, а с другой – строить сложные типы как композиции простых. То и другое достигается при помощи наследования.

Концепция наследования является центральной для использования классов и объектов в C++. Наследование позволяет определять новые классы как расширения существующих. Производный класс, наследующий базовому, со-

держит в той или иной форме все данные и алгоритмы, присутствующие в базовом классе, и добавляет свои собственные. В C++ важно различать два основных типа наследования: открытое и закрытое.

В случае открытого наследования наследуется интерфейс класса. Наследуется и его реализация – данные-члены базового класса являются также членами производного. Но именно наследование интерфейса – отличительная черта открытого наследования; это означает, что частью открытого интерфейса производного класса являются все открытые функции-члены базового.

Напомним, что открытый интерфейс подобен контракту – мы обещаем клиентам класса, что он будет поддерживать определенные операции, сохранять некоторые инварианты и подчиняться специфицированным ограничениям. Открыто наследуя базовому классу, мы связываем производный класс тем же контрактом (и, возможно, расширяем его, если решим определить дополнительные открытые интерфейсы). Поскольку производный класс соблюдает интерфейс базового класса, мы вправе использовать производный класс всюду, где допустим базовый; возможно, мы не сможем воспользоваться расширениями интерфейса (код ожидает получить базовый класс и не знает ни о каких расширениях), но интерфейс и ограничения базового класса остаются в силе.

Часто эту мысль формулируют в виде *принципа «является»* – экземпляр производного класса является также экземпляром базового класса. Однако способ интерпретации отношения *является* в C++ интуитивно не вполне очевиден. Например, является ли квадрат прямоугольником? Если да, то мы можем произвести класс `Square` от класса `Rectangle`:

```
class Rectangle {
    public:
    double Length() const { return length_; }
    double Width() const { return width_; }
    ...
    private:
    double l_;
    double w_;
};
class Square : public Rectangle {
    ...
};
```

Сразу видно, что здесь не все в порядке – в производном классе два члена, задающих измерения, тогда как в действительности нужен лишь один. Необходимо как-то гарантировать, что их значения одинаковы. Вроде бы ничего страшного – интерфейс класса `Rectangle` допускает любые положительные значения длины и ширины, а класс `Square` налагает дополнительные ограничения. Но на самом деле все гораздо хуже – контракт класса `Rectangle` разрешает пользователю задать разные измерения. Это даже можно выразить явно:

```
class Rectangle {
    public:
```