## Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — это особый концептуальный подход к проектированию программ, и C++ расширяет язык С средствами, облегчающими применение такого подхода. Ниже перечислены наиболее важные характеристики ООП: 1)абстракция; 2)инкапсуляция и сокрытие данных; 3)полиморфизм; 4)наследование; 5)повторное использование кода. Класс — это единственное наиболее важное расширение C++, предназначенное для реализации этих средств и связывающее их между собой.

## Типы данных

Прежде всего, о типе данных мы думаем в терминах того, как он выглядит — каким образом хранится в памяти. Например, тип char занимает 1 байт памяти, a double — обычно 8 байт. Но если рассуждать далее, то мы придем к заключению, что тип данных также определен в терминах операций, которые допустимо выполнять с ним. Например, к типу int можно применять все арифметические операции. Целые числа можно складывать, вычитать, умножать, делить. Можно также применять операцию взятия модуля (%).

С другой стороны, рассмотрим указатели. Указатель может требовать для своего хранения столько же памяти, сколько и тип int. Он даже может иметь внутреннее представление в виде целого числа. Но указатель не позволяет выполнять над собой те же операции, что и целое. Например, перемножить два указателя не удастся. Эта концепция не имеет смысла, поэтому в C++ она не реализована. Таким образом, когда вы объявляете переменную как int или указатель на float, то не просто выделяете память для нее, но также устанавливаете, какие операции допустимы с этой переменной. Значит, спецификация базового типа выполняет три вещи. **1.** Определяет, сколько памяти нужно объекту. **2.** Определяет, как интерпретируются биты памяти. Типы long и float могут занимать одинаковое количество бит памяти, но транслируются в числовые значения по-разному. **3.** Определяет, какие операции, или методы, могут быть применены с использованием этого объекта данных.

Для встроенных типов информация об операциях встроена в компилятор. Но когда вы определяете пользовательский тип в C++, то должны предоставить эту информацию самостоятельно.

## Классы в C++

Класс — это двигатель C++, предназначенный для трансляции абстракции в пользовательские типы. Он комбинирует представление данных и методов для манипулирования этими данными в пределах одного пакета. Давайте взглянем на класс, представляющий акционерный капитал некоторой кампании.

Вначале нужно подумать о том, как представлять акции. Вы можете взять в качестве базовой единицы один пакет акций и определить класс, представляющий этот пакет. Однако это потребует создания 100 объектов для представления 100 пакетов, что явно не практично. Вместо этого в качестве базовой единицы можно представить персональную долю владельца в определенном пакете. Количество акций, находящихся во владении, будут частью представления данных.

Реалистичный подход должен позволять поддерживать хранение информации о таких вещах, как начальная стоимость и дата покупки; это необходимо для налогообложения. Кроме того, он должен предусматривать обработку таких событий, как разделение пакетов.

Открытый интерфейс класса содержит следующие функции: 1) приобретение начального пакета акций компании (будущий метод acquire()); 2) покупка дополнительных акций в имеющийся пакет (будущий метод buy()); 3) продажа акций (sell()); 4) обновление объема доли в пакете акций, update());

5) отображение информации об акциях, находящихся у клиента (show()).

Этот список можно использовать для определения открытого интерфейса класса и при необходимости позже добавлять дополнительные средства.

Для поддержки этого интерфейса необходимо сохранять некоторую информацию:

1) название компании, представляющий акционерный капитал (поле company);

2) количество акций, находящихся во владении клиента (поле shares);

3) стоимость одной акции (поле share\_val);

4) общая стоимость всех акций конкретной компании (поле total\_val).

Далее можно определить класс. Обычно спецификация класса состоит из двух частей.

• Объявление класса, описывающее компоненты данных в терминах членов данных, а также открытый интерфейс в терминах функций-членов, называемых методами.

• Определения методов класса, которые описывают, как реализованы определенные функции-члены.

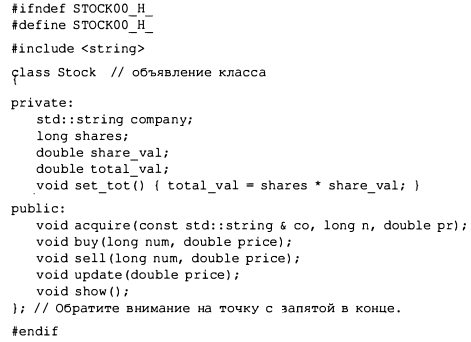
Таким образом, объявление класса предоставляет общий обзор класса, в то время как определения методов снабжают необходимыми деталями.

Обычно помещают интерфейс, имеющий форму определения класса, в заголовочный файл, а реализацию в форме кода для методов класса — в файл исходного кода. Это является обычной практикой. В примере 1.1 представлена первая стадия, пробное объявление класса под именем Stock. В этом файле применяется #ifndef и т.п. для защиты против многократного включения файла.

Чтобы упростить идентификацию классов, есть соглашение о написании имен классов с заглавной буквы. Обратите внимание, на объявление класса, на функции-члены, а также на разделы public и private. Рассмотрим, как все работает.

**Пример.** В программе создать класс, который имитирует продажу и покупку акций компаний на бирже. Необходимо хранить следующую информацию: 1) название компании, представляющий акционерный капитал (поле company); 2) количество акций, находящихся во владении клиента (поле shares); 3) стоимость одной акции (поле share\_val); 4) общая стоимость всех акций конкретной компании (поле total\_val). Открытый интерфейс класса содержит следующие функции: 1) приобретении начального пакета акций компании (будущий метод acquire()); 2) покупка дополнительных акций в имеющийся пакет (будущий метод buy()); 3) продажа акций (sell()); 4) обновление объема доли в пакете акций (update()); 5) отображение информации об акциях, находящихся у клиента (show()). Создать объект класса, например, клиента по имени Оливер, который купил 20 акций компании «NanoSmart» по цене 12,5$ за акцию, смоделировать различные ситуации с дальнейшей покупкой, продажей и общей стоимостью пакета акций.

Пример 1.1 (stock00.h).



Ключевое слово **class** в C++ идентифицирует код в примере 1.1 (stock00.h) как определение класса. В таком контексте ключевые слова class и typename не являются синонимами, как это было в параметрах шаблона; typename здесь использовать нельзя. Данный синтаксис идентифицирует Stock в качестве имени типа для нового класса. Это позволяет объявлять переменные, которые называются объектами или экземплярами типа Stock. Каждый индивидуальный объект этого типа представляет отдельный пакет акций, находящийся во владении некоторого человека. Например, следующее объявление создает два объекта с именами sally и sollу:



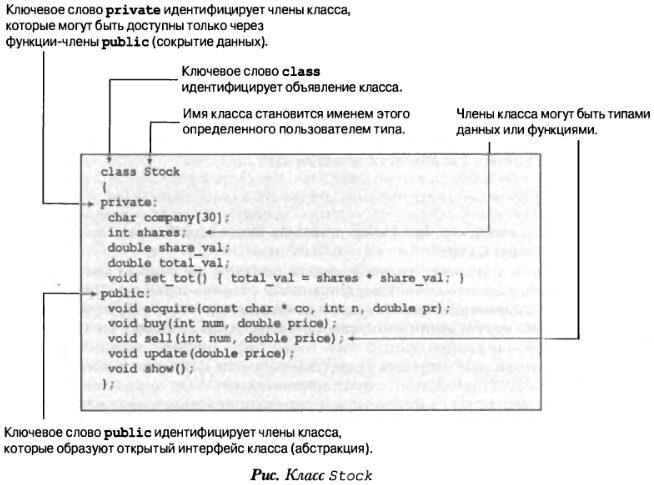
Объект sally, например, представляет пакет акций определенной компании, принадлежащий Салли. Аналогично, объект solly, - это пакет акций, которым владеет Солли.

Информация, которую мы решили сохранять, появляется в форме данных-членов класса, таких как company и shares. Член company объекта sally, например, хранит название компании, член share содержит количество долей общего пакета акций компании, которыми владеет Салли, член share\_val соответствует объему каждой доли, а член total\_val — общему объему всех долей. Аналогично необходимые операции представлены в виде функций-членов (или методов), таких как sell() и update(). Функция-член может быть определена на месте, как, например, set\_tot(), либо подобно остальным функции-члены в этом классе представлена с помощью прототипа. Полные определения остальных функций-членов появятся позже, в файле реализации, но прототипов уже достаточно для описания их интерфейса. Связывание данных и методов в единое целое — наиболее замечательное свойство класса. При таком проектном решении создание объекта типа Stock автоматически устанавливает правила, регулирующие его использованием.

**Замечание.** Известно, что классы istream и ostream имеют функции-члены вроде get() и getline(). Прототипы функций в определении класса Stock демонстрируют установку функций-членов. Заголовочный файл iostream, например, содержит прототип getline() в объявлении класса iostream.

## Управление доступом

Рассмотрим ключевые слова private и public. Эти метки позволяют управлять доступом к членам класса. Любая программа, которая использует объект определенного класса, может иметь непосредственный доступ к членам из раздела public. Доступ к членам объекта из раздела private программа может получить только через открытые функции-члены из раздела public (или же через дружественные функции). Например, единственный способ изменить переменную shares класса Stock — это воспользоваться одной из функций-членов класса Stock.



Таким образом, открытые функции-члены действуют в качестве посредников между программой и закрытыми членами объекта; они предоставляют интерфейс между объектом и программой. Эта изоляция данных от прямого доступа со стороны программы называется **сокрытием данных**. Сокрытие данных является хорошей практикой в программировании, поскольку предохраняет целостность данных.

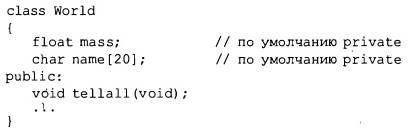
Проектное решение класса должно быть таким, чтобы отделить открытый интерфейс от реализации. Открытый интерфейс представляет абстрактный компонент проектного решения. Собрание деталей реализации в одном месте и отделение их от абстракции называется инкапсуляцией. Сокрытие данных (помещение данных в раздел private класса) является примером **инкапсуляции**, и поэтому оно скрывает функциональные детали реализации в разделе private, как это сделано в классе Stock с функцией set\_tot(). Другим примером инкапсуляции может служить обычная практика помещения определений функций класса в файл, отдельный от объявления класса.

Следует отметить, что сокрытие данных не только предотвращает прямой доступ к данным, но также избавляет вас (в роли пользователя этого класса) от необходимости знать то, как представлены данные. Например, метод show() отображает, общую сумму пакета акций, находящегося во владении. Это значение может быть сохранено в виде части объекта, как это делает код в примере 1.1, либо при необходимости может быть вычислено. С точки зрения пользователя класса нет разницы, какой подход применяется. Необходимо знать только то, что делают различные функции-члены — т.е. какие аргументы они принимают, и какие типы значений возвращают. Принцип состоит в том, чтобы отделить детали реализации от проектного решения интерфейса. Если позже вы найдете лучший способ реализации представления данных или деталей внутреннего устройства функций-членов, то сможете изменить их без изменения программного интерфейса, что значительно облегчает поддержку и сопровождение программ.

## Управление доступом к членам: public или private

Объявлять члены класса — будь они элементами данных или функциями-членами — можно как в открытом (public), так и в закрытом (private) разделе класса. Но поскольку одним из главных принципов ООП является сокрытие данных, то единицы данных обычно размещаются в разделе private. Функции-члены, которые образуют интерфейс класса, размещаются в разделе public; в противном случае вызвать эти функции из программы не удастся. Как показывает объявление класса Stock, вы все же можете поместить функции-члены в раздел private. Вызвать такие функции из программы непосредственно не получится, но их могут использовать открытые методы. Как правило, закрытые функции-члены применяются для управления деталями реализации, которые не формируют часть открытого интерфейса.

Использовать ключевое слово private в объявлении класса не обязательно, поскольку это спецификатор доступа к объектам класса по умолчанию:



Однако существует рекомендация метку private указываться явно, чтобы подчеркнуть концепцию сокрытия данных.

**Замечание (о классы и структурах)**. Описания классов выглядят очень похожими на объявления структур с дополнениями в виде функций-членов и меток видимости private и public. Фактически C++ расширяет на структуры те же самые свойства, которые есть у классов. Единственная разница состоит в том, что типом доступа по умолчанию у структур является public, в то время как у классов — private. Программисты обычно используют классы для реализации описаний классов, тогда как ограниченные структуры применяются для чистых объектов данных, которые часто называются простыми старыми структурами данных (plain-old data — POD).

## Реализация функций-членов класса

Необходимо определять вторую часть спецификации класса, т.е. предоставлять код для тех функций-членов, которые описаны с помощью прототипов в объявлении класса. Определения функций-членов очень похожи на определения обычных функций. Каждое из них имеет заголовок и тело. Определения функций-членов могут иметь тип возврата и аргументы. Но, кроме того, с ними связаны две специфических характеристики. **1.**При определении функции-члена для идентификации класса, которому принадлежит функция, используется **операция разрешения контекста** (::). **2.**Методы класса имеют доступ к private-компонентам класса.

Рассмотрим все это подробнее. В заголовке функции-члена для идентификации класса, которому она принадлежит, применяется операция разрешения контекста (::). Например, заголовок для функции-члена update () выглядит следующим образом:



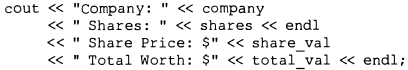
Эта нотация означает, что вы определяете функцию update (), которая является членом класса Stock. Но это означает не только то, что функция update () является функцией-членом, но также и то, что такое же имя можно использовать для функций-членов другого класса. Например, функция update() для класса Button будет иметь следующий заголовок:



Таким образом, операция разрешения контекста идентифицирует класс, к которому данный метод относится. Говорят, что идентификатор update () имеет область видимости класса. Другие функции-члены класса Stock могут при необходимости использовать метод update () без операции разрешения контекста. Это связано с тем, что они принадлежат одному классу, и, следовательно, имеют общую область видимости. Использование update () за пределами объявления класса и определений методов, однако, требует соблюдения специальных мер.

Единственный способ однозначного разрешения имен методов — использовать полное имя, включающее имя класса. Stock::update() называется уточненным именем функции. Простое имя update(), с другой стороны, является сокращением (неуточненным именем) полного имени и может применяться только в контексте класса.

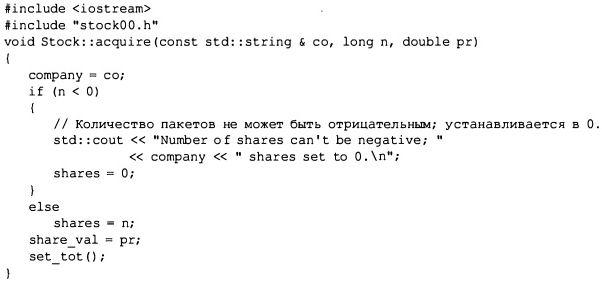
Следующей специальной характеристикой методов является то, что метод может иметь доступ к закрытым членам класса. Например, метод show() может использовать код вроде следующего:

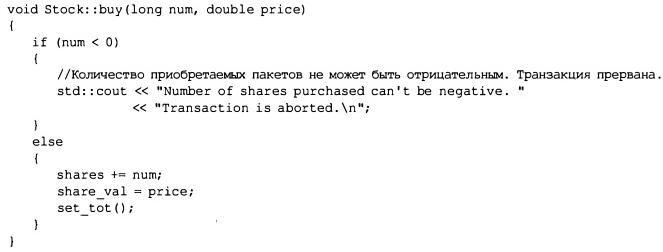


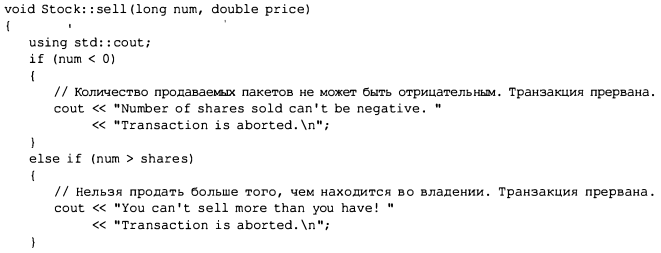
Здесь company, shares и т.д. являются закрытыми данными-членами класса Stock. Если вы попытаетесь воспользоваться для доступа к этим данным-членам функцией, которая не является членом, то компилятор воспрепятствует этому (исключением являются дружественные функции).

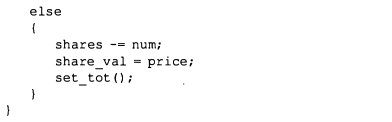
Памятуя об этих двух обстоятельствах, методы класса можно реализовать, как показано в примере 1.2. Эти определения методов могут быть помещены в отдельный файл либо в тот же файл, где находится объявление класса. В данном примере они помещены в отдельный файл реализации, поэтому в нем потребуется включить заголовочный файл stock00.h, чтобы компилятор имел доступ к определению класса. Для демонстрации возможности работы с пространствами имен в одних методах используется квалификатор std::, а в других — объявления using.

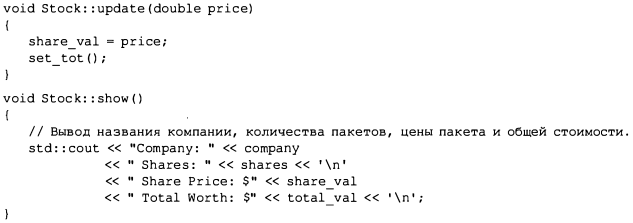
Пример 1.2 (stocks00.cpp).











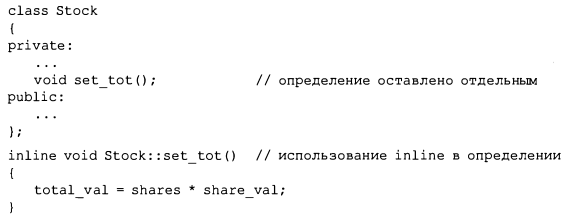
Функция acquire() управляет первоначальной покупкой пакета акций некоторой компании, в то время как buy() и sell() — дополнительной покупкой и продажей акций из существующего пакета. Методы buy() и sell() гарантируют, что количество купленных или проданных акций не будет отрицательным. Кроме того, если пользователь пытается продать больше акций, чем у него есть, функция sell() отменит транзакцию. Объявление данных закрытыми и ограничения доступа к открытым функциям предоставляет контроль над использованием данных. В данном случае это позволяет предпринять защитные меры против недопустимых транзакций.

Четыре из определенных функций-членов устанавливают или сбрасывают значение члена totalval. Вместо того чтобы повторять в коде вычисление этого значения четыре раза, каждая из открытых функций-членов вызывает функцию set\_tot(). Поскольку эта функция представляет собой просто реализацию внутреннего кода, а не является частью открытого интерфейса, в классе она объявлена как закрытая функция-член. То есть set\_tot() представляет собой функцию-член, которая используется разработчиком класса, но не теми, кто пишет код, использующий класс. Если вычисления, выполняемые функцией, оказываются сложными, это поможет также уменьшить общий объем исходного кода. Однако здесь главное в том, что за счет использования вызова этой функции вместо повторения кода вычислений обеспечивается гарантия того, что всегда будет применяться абсолютно идентичный алгоритм. Кроме того, если его понадобится изменить, это нужно будет сделать только в одном месте.

## Встроенные методы

Любая функция с определением внутри объявления класса автоматически становится встроенной. Это значит, что Stock::settot() является встроенной функцией. Объявления класса часто используют встроенные функции для коротких функций-членов, и set\_tot() — пример такого случая.

Второй способ состоит в том, что можно определить функцию-член вне объявления класса и, тем не менее, сделать ее встроенной. Чтобы это сделать, просто необходимо использовать квалификатор **inline** при определении функции в разделе реализации класса:



Специальные правила для встроенных функций требуют, чтобы они были определены в каждом файле, в котором используются. Самый простой способ гарантировать, что встроенные определения доступны всем файлам в многофайловой программе — поместить эти определения в тот же заголовочный файл, где объявлен класс. Некоторые системы разработки снабжены интеллектуальными компоновщиками, которые разрешают размещать встроенные определения в отдельном файле реализации.

Кстати, согласно правилу перезаписи, определение метода внутри объявления класса эквивалентно замене определения метода прототипом и последующей перезаписью определения в виде встроенной функции немедленно после объявления класса. То есть исходное определение set\_tot () в примере 1.1 эквивалентно только что показанному, где определение следует за объявлением класса.

## Объекты и методы

Мы подошли к одному из наиболее важных аспектов использования объектов: каким образом метод класса применяется к объекту? Код представленный ниже использует член shares некоторого объекта:



Но какого объекта? Чтобы ответить на него, давайте вначале посмотрим, как создается объект. Наиболее простой способ объявления переменных класса выглядит так:



Это создает два объекта класса Stock, один по имени kate, а второй — joe.

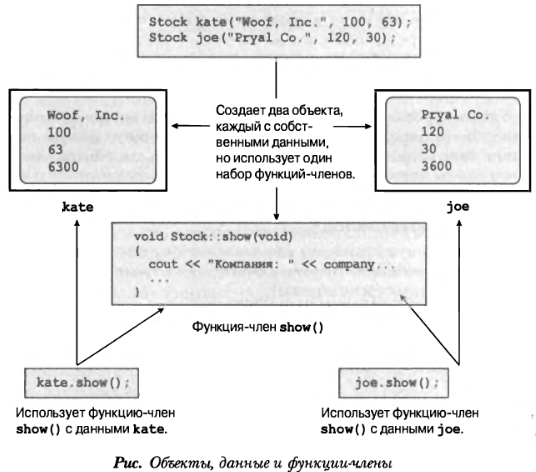
Теперь рассмотрим, как использовать функцию-член с одним из этих объектов. Ответ, как и в случае структур и членов структур, состоит в применении операции членства:



Первый оператор вызывает show () как член объекта kate. Это значит, что метод интерпретирует shares как kate.shares, a share\_val — как kate.share\_val. Аналогично вызов joe.show() заставляет метод show() интерпретировать shares как joe.shares, a share\_val — как joe.share\_val, соответственно.

Подобным же образом вызов kate. sell () запускает функцию set\_tot (), как если бы это была kate.set\_tot(), позволяя ей получать доступ к данным объекта kate.

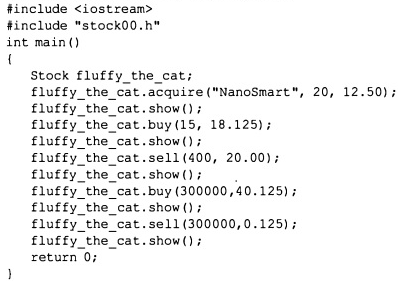
Каждый вновь созданный вами объект содержит хранилище для собственных внутренних переменных-членов класса, однако все объекты одного класса разделяют общий набор методов, по одной копии каждого. Предположим, например, что kate и joe — это объекты класса Stock. В этом случае kate.shares занимает один фрагмент памяти, a joe. shares — другой. Но kate. show() и joe. show() представляют собой один и тот же метод, т.е. оба выполняют один и тот же блок кода, только применяют этот код к разным данным. Вызов функции-члена — это то, что в некоторых объектно-ориентированных языках называется отправкой сообщения. Таким образом, отправка сообщения двум разным объектам вызывает один и тот же метод, который применяется к двум разным объектам (см. рис.).



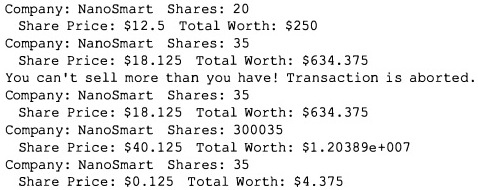
## Использование классов

Создавать объект класса можно за счет объявления переменной этого класса либо использования операция new для размещения в памяти объекта этого класса. Объекты можно передавать в аргументах, возвращать их из функций, присваивать один объект другому. Язык C++ предоставляет средства для инициализации объектов, для «обучения» cin и cout распознавать объекты и даже для выполнения автоматического приведения типов между объектами подобных классов. В примере 1.3 приведен код программы, которая использует файлы интерфейса и реализации. В коде создается объект типа Stock по имени fluffy\_the\_cat.

Пример 1.3 (usestok0.cpp).



Результат работы программы:

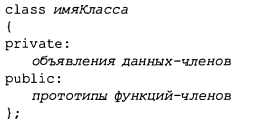


Обратите внимание, что main () — это просто механизм для тестирования класса Stock. Когда класс Stock заработает должным образом, его можно будет применять в качестве пользовательского типа в других программах. Важнейшим моментом для использования нового типа является понимание того, что делают функции-члены; вы не должны задумываться о деталях реализации.

**Замечание** (клиент-серверная модель). Программисты, соблюдающие принципы ООП, часто обсуждают проект программ в терминах клиент-серверной модели. Согласно этой концепции, клиентом является программа, которая использует класс. Объявление класса, включая его методы, образует сервер, который является ресурсом, доступным нуждающейся в нем программе. Клиент взаимодействует с сервером только через открытый (public) интерфейс. Это означает, что единственной ответственностью клиента и, как следствие — программиста, является знание интерфейса. Ответственностью сервера и, как следствие — его разработчика, является обеспечение того, чтобы его реализация надежно и точно соответствовала интерфейсу. Любые изменения, вносимые разработчиком сервера в класс, должны касаться деталей реализации, но не интерфейса. Это позволяет программистам разрабатывать клиент и сервер независимо друг от друга, без внесения в сервер таких изменений, которые нежелательным образом отобразятся на поведении клиента.

## Проектирование класса

Первый шаг в проектировании класса заключается в предоставлении объявления класса. Объявление класса смоделировано на основе объявления структуры и может включать в себя данные-члены и функции-члены. Объявление имеет раздел private, и члены, объявленные в этом разделе, могут быть доступны только через функции-члены. Объявление также содержит раздел public, и объявленные в нем члены могут быть непосредственно доступны программе, использующей объекты класса. Как правило, данные-члены попадают в закрытый раздел, а функции-члены — в открытый, поэтому типичное объявление класса имеет следующую форму:



Содержимое открытого раздела включает абстрактную часть проектного решения — открытый интерфейс. Инкапсуляция данных в закрытом разделе защищает их целостность и называется сокрытием данных. Таким образом, использование классов — это способ, который предлагает C++ для облегчения реализации абстракций, сокрытия данных и инкапсуляции ООП.

Второй шаг в спецификации класса — это реализация функций-членов класса. Вместо прототипов в объявление можно включать полное определение функций, однако общепринятая практика состоит в том, чтобы использовать прототипы, а определять функции отдельно. В этом случае вам понадобится операция разрешения контекста для индикации того, к какому классу данная функция-член принадлежит. Например, предположим, что класс Bozo имеет функцию-член Retort (), которая возвращает указатель на тип char. Заголовок функции должен выглядеть примерно так:



Другими словами, Retort () — не только функция типа char \*, это функция типа char \*, принадлежащая классу Bozo. Полное, или уточненное, имя функции будет выглядеть как Bozo::Retort(). Имя Retort (), с другой стороны, является сокращением уточненного имени, и оно должно использоваться только в определенных случаях, таких как в коде методов класса.

Другой способ описания этой ситуации — это говорить о том, что Retort имеет область видимости класса, поэтому необходима операция разрешения контекста для уточнения имени, когда оно встречается вне объявления и вне методов класса.

Для создания объекта, который является частным примером класса, применяется имя класса, как если бы оно было именем типа:



Это работает потому, что класс является типом, определенным пользователем.

Функция-член класса, или метод, вызывается с использованием объекта класса. Это делается с помощью операции членства (точки):



Код вызывает функцию-член Retort (), и всякий раз, когда код этой функции обращается к определенным данным-членам, используются значения членов объекта bozetta.

## Некоторые изменения реализации

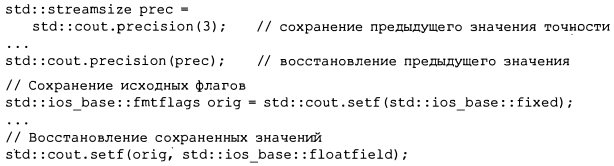
С выводом программы связан один момент, который может не устраивать — неподходящее форматирование чисел. Имеется возможность улучшить реализацию, не затрагивая интерфейс. Класс ostream содержит функции-члены, которые управляют форматированием. Не особо вдаваясь в детали, скажем, что с помощью метода setf() можно избавиться от экспоненциальной нотации:



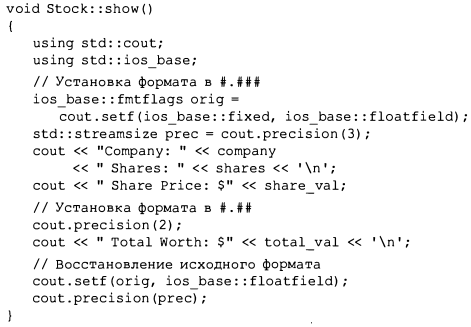
Этот вызов устанавливает флаг, который заставляет объект cout использовать нотацию с фиксированной точкой. Подобным же образом следующий оператор заставляет cout выводить три десятичных знака после точки:



Эти средства можно использовать в методе show () для управления форматированием, но следует учесть еще один момент. В случае изменения реализации метода, внесенные модификации не должны влиять на другие части клиентской программы. Изменения в формате будут оставаться активными вплоть до следующих изменений, поэтому они могут повлиять на последующий вывод в клиентской программе. Следовательно, в show () должен быть предусмотрен возврат к состоянию форматирования, которое было до вызова этого метода. Это можно сделать, с применением возвращаемых значений операторов установки формата:



Во-первых, fmtflags — это тип, определенный в классе iosbase, который находится в пространстве имен std, отсюда и такое довольно длинное имя типа для orig. Во-вторых, orig хранит все флаги, и оператор сброса использует эту информацию для восстановления установок в разделе floatfield, который включает флаги для нотации с фиксированной точкой и экспоненциальной нотации. В-третьих, давайте не будем здесь сильно беспокоиться о деталях. Главный момент в том, что изменения ограничиваются файлом реализации и не влияют на программу, использующую этот класс. Итак, изменим определение метода в файле реализации следующим образом:



После этой замены вывод будет выглядеть так:

