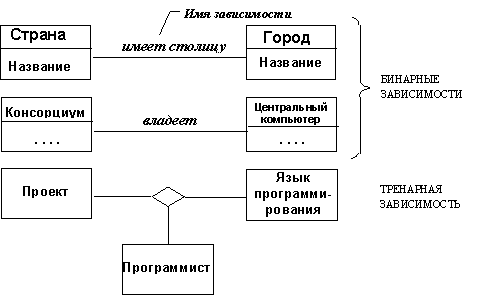
**13. Зависимости между классами (объектами).**  
С каждым объектом связана структура данных, полями которой являются атрибуты этого объекта и указатели функций (фрагментов кода), реализующих операции этого объекта (отметим, что указатели функций в результате оптимизации кода обычно заменяются на обращения к этим функциям). Таким образом, объект - это некоторая структура данных, тип которой соответствует классу этого объекта.

Между объектами можно устанавливать *зависимости* по данным. Эти *зависимости* выражают *связи* или *отношения* между классами указанных объектов. Примеры таких зависимостей изображены на рисунке 2.6 (первые две зависимости - бинарные, третья зависимость - тренарная). Зависимость изображается линией, соединяющей классы над которой надписано имя этой зависимости, или указаны роли объектов (классов) в этой зависимости (указание ролей - наиболее удобный способ идентификации зависимости).



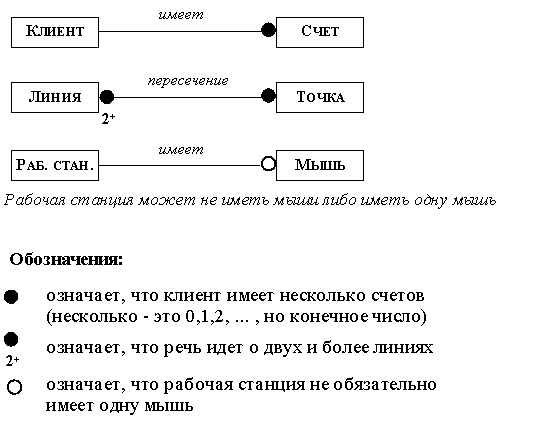
*Рис. 2.6. Зависимости между классами*

Зависимости между классами являются двусторонними: все классы в зависимости равноправны. Это так даже в тех случаях, когда имя зависимости как бы вносит направление в эту зависимость. Так, в первом примере на рисунке 2.6 имя зависимости имеет\_столицу предполагает, что зависимость направлена от класса страна к классу город (двусторонность зависимости вроде бы пропала); но следует иметь в виду, что эта зависимость двусторонняя в том смысле, что одновременно с ней существует и обратная зависимость является\_столицей. Точно таким же образом, во втором примере на рисунке 2.6 можно рассматривать пару зависимостей владеет-принадлежит. Подобных недоразумений можно избежать, если идентифицировать зависимости не по именам, а по наименованиям ролей классов, составляющих зависимость.

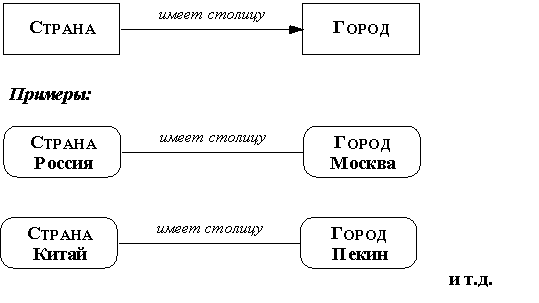
В языках программирования зависимости между классами (объектами) обычно реализуются с помощью ссылок (указателей) из одного класса (объекта) на другой. Представление зависимостей с помощью ссылок обнаруживает тот факт, что зависимость является свойством пары классов, а не какого-либо одного из них, т.е. зависимость - это отношение. Отметим, что хотя зависимости между объектами двунаправлены, их не обязательно реализовать в программах как двунаправленные, оставляя ссылки лишь в тех классах, где это необходимо для программы.

Дальнейшие примеры зависимостей между классами приведены на рисунке 2.7. Первый пример показывает зависимость между клиентом банка и его счетами. Клиент банка может иметь одновременно несколько счетов в этом банке, либо вовсе не иметь счета (когда он впервые становится клиентом банка). Таким образом, нужно изобразить зависимость между клиентом и несколькими счетами, что и сделано на рисунке 2.7. Второй пример показывает зависимость между пересекающимися кривыми (в частности, прямыми) линиями. Можно рассматривать 2, 3, и более таких линий, причем они могут иметь несколько точек пересечения. Наконец, третий пример показывает необязательную (optional) зависимость: компьютер может иметь, а может и не иметь мышь.

Зависимостям между классами соответствуют зависимости между объектами этих классов. На рисунке 2.8 показаны зависимости между объектами для первого примера рисунка 2.6; на рисунке 2.9 показаны зависимости между объектами для примеров, изображенных на рисунке 2.7.



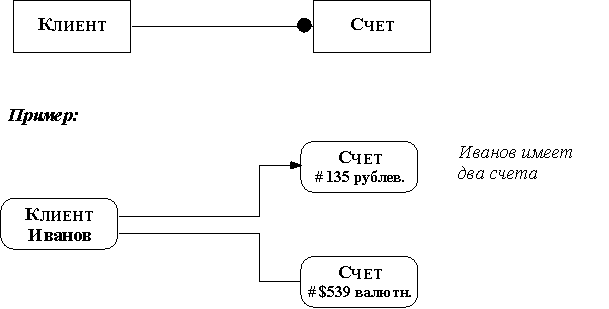
*Рис. 2.7. Дальнейшие примеры зависимостей. Обозначения*

**

*Рис. 2.8. Зависимости между объектами*

Отметим, что при изображении зависимостей между объектами мы, как правило, знаем количество объектов и не нуждаемся в таких обозначениях как "несколько", "два и более", "не обязательно".

При проектировании системы удобнее оперировать не объектами, а классами.



*Рис. 2.9. Более сложные зависимости между объектами*

Понятие зависимости перенесено в объектно-ориентированную технологию проектирования программных систем из технологии проектирования (и моделирования) баз данных, где зависимости используются с давних пор. Языки программирования, как правило, не поддерживают явного описания зависимостей. Тем не менее описание зависимостей очень полезно при разработке программных систем. Технология OMT использует зависимости при интерпретации диаграмм, описывающих систему.

**14. 1. Определение классов.**

Анализ внешних требований к проектируемой прикладной системе позволяет определить объекты и классы объектов, связанные с прикладной проблемой, которую должна решать эта система. Все классы должны быть осмыслены в рассматриваемой прикладной области; классов, связанных с компьютерной реализацией, как например список, стэк и т.п. на этом этапе вводить не следует.

Начать нужно с выделения возможных классов из письменной постановки прикладной задачи (технического задания и другой документации, предоставленной заказчиком). Следует иметь в виду, что это очень сложный и ответственный этап разработки, так как от него во многом зависит дальнейшая судьба проекта.

При определении возможных классов нужно постараться выделить как можно больше классов, выписывая имя каждого класса, который приходит на ум. В частности, каждому существительному, встречающемуся в предварительной постановке задачи, может соответствовать класс. Поэтому при выделении возможных классов каждому такому существительному обычно сопоставляется возможный класс.

Далее список возможных классов должен быть проанализирован с целью исключения из него *ненужных* классов. Такими классами являются:

* *избыточные классы:* если два или несколько классов выражают одинаковую информацию, следует сохранить только один из них;
* *нерелевантные* (не имеющие прямого отношения к проблеме) *классы*: для каждого имени возможного класса оценивается, насколько он необходим в будущей системе (оценить это часто бывает весьма непросто); нерелевантные классы исключаются;
* *нечетко определенные* (с точки зрения рассматриваемой проблемы) *классы* (см. примеры таких классов в п. [2.3.1](http://citforum.ru/programming/oop_rsis/glava2_3_1.shtml));
* *атрибуты*: некоторым существительным больше соответствуют не классы, а атрибуты; такие существительные, как правило, описывают свойства объектов (например, имя, возраст, вес, адрес и т.п.);
* *операции*: некоторым существительным больше соответствуют не классы, а имена операций (например, телефонный\_вызов вряд ли означает какой-либо класс);
* *роли*: некоторые существительные определяют имена ролей в объектной модели (например, владелец, водитель, начальник, служащий; все эти имена связаны с ролями в различных зависимостях объектов класса человек);
* *реализационные конструкции*: именам, больше связанным с программированием и компьютерной аппаратурой, не следует на данном этапе сопоставлять классов, так как они не отражают особенностей проектируемой прикладной системы; примеры таких имен: подпрограмма, процесс, алгоритм, прерывание и т.п.

После исключения имен всех ненужных (лишних) возможных классов будет получен предварительный список классов, составляющих проектируемую систему.

**14.2. Подготовка словаря данных.**

Приведем часть словаря данных, содержащую определения классов, используемых в проекте.

ATM (банкомат) - терминал, который дает возможность клиенту осуществлять свою собственную проводку, используя для идентификации свою карточку. ATM (банкомат) взаимодействует с клиентом, чтобы получить необходимую информацию для проводки, посылает информацию для проводки центральному\_компьютеру, чтобы он проверил ее и в дальнейшем использовал при выполнении проводки и выдает деньги и квитанцию клиенту. Предполагается, что ATM (банкомату) не требуется работать независимо от сети.

Банк - финансовая организация, которая содержит счета своих клиентов и выпускает карточки, санкционирующие доступ к счетам через сеть ATM (банкоматов).

Карточка - пластиковая карточка, врученная банком своему клиенту, которая санкционирует доступ к счетам через сеть ATM (банкоматов). Каждая карточка содержит код банка и номер карточки, закодированные в соответствии с национальными стандартами на банковские карточки. Код\_банка однозначно идентифицирует банк внутри консорциума. Номер\_карточки определяет счета, к которым карточка имеет доступ. Карточка не обязательно обеспечивает доступ ко всем счетам клиента. Каждой карточкой может владеть только один клиент, но у нее может существовать несколько копий, так что необходимо рассмотреть возможность одновременного использования одной и той же карточки с разных ATM (банкоматов).

Кассир - служащий банка, который имеет право осуществлять проводки с кассовых\_ терминалов, а также принимать и выдавать деньги и чеки клиентам. Проводки, деньги и чеки, с которыми работает каждый кассир должны протоколироваться и правильно учитываться.

Кассовый\_терминал - терминал, с которого кассир осуществляет проводки для клиентов. Когда кассир принимает и выдает деньги и чеки, кассовый\_терминал печатает квитанции. Кассовый\_терминал взаимодействует с компьютером\_банка, чтобы проверить и выполнить проводку.

Клиент - держатель одного или нескольких счетов в банке. Клиент может состоять из одного или нескольких лиц, или организаций. То же самое лицо, держащее счет и в другом банке рассматривается как другой клиент.

Компьютер\_банка - компьютер, принадлежащий банку, который взаимодействует с сетью ATM (банкоматов) и собственными кассовыми\_терминалами банка. Банк может иметь свою внутреннюю компьютерную сеть для обработки счетов, но здесь мы рассматриваем только тот компьютер\_банка, который взаимодействует с сетью ATM.

Консорциум - объединение банков, которое обеспечивает работу сети ATM (банкоматов). Сеть передает в консорциум проводки банков.

Проводка - единичный интегрированный запрос на выполнение некоторой последовательности операций над счетами одного клиента. Было сделано предположение, что ATM (банкоматы) только выдают деньги, однако для них не следует исключать возможности печати чеков или приема денег и чеков. Хотелось бы также обеспечить гибкость системы, которая в дальнейшем обеспечит возможность одновременной обработки счетов разных клиентов, хотя пока этого не требуется. Различные операции должны быть правильно сбалансированы.

Счет - единичный банковский счет, над которым выполняются проводки. Счета могут быть различных типов; клиент может иметь несколько счетов.

Центральный\_компьютер - компьютер, принадлежащий консорциуму, который распределяет проводки и их результаты между ATM (банкоматами) и компьютерами\_банков. Центральный\_компьютер проверяет коды банков, но не выполняет проводок самостоятельно.

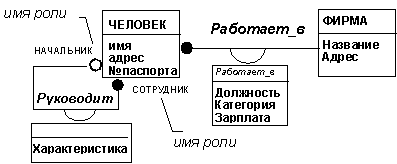
**15. Атрибуты зависимостей.**

Зависимости, как и классы, могут иметь атрибуты: например, при организации доступа пользователя к файлу разрешение\_на\_доступ является атрибутом зависимости доступен: см. рисунок 2.10, на котором атрибут зависимости обозначается прямоугольником, связанным дугой с прямой, изображающей зависимость. Такое обозначение атрибутов зависимостей принято в технологии OMT. Отметим, что разрешение на доступ связано как с пользователем, так и с файлом, и не может быть атрибутом ни пользователя, ни файла в отдельности.



*Рис. 2.10. Пример атрибута зависимости*

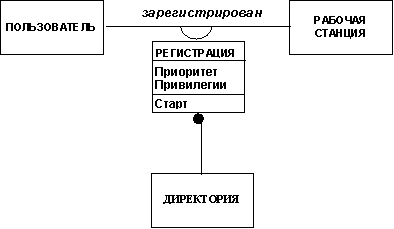
Еще один пример зависимостей, имеющих атрибуты, показан на рисунке 2.11. Из примера видно, что зависимость может иметь несколько атрибутов. Кроме того, на рисунке 2.11 указаны роли различных объектов в зависимости (подробнее см. ниже). Зависимость руководит на этом рисунке удобнее именовать как начальник-сотрудник.



*Рис. 2.11. Атрибуты двух зависимостей между одним и многими*

Иногда зависимости, имеющие много атрибутов, представляют с помощью классов. Такие зависимости в базах данных представляются временными таблицами, организуемыми в процессе обращения с базой данных. Пример зависимости, представленной через класс, показана на рисунке 2.12, на котором представлена информация о регистрации пользователей на рабочих станциях.

Пользователь может быть зарегистрирован на нескольких рабочих станциях, каждая регистрация содержит приоритет пользователя и его привилегии доступа (атрибуты зависимости). Пользователь может иметь свою директорию для каждой зарегистрированной рабочей станции, но одна и та же директория может принадлежать одновременно нескольким пользователям или нескольким рабочим станциям.



*Рис. 2.12. Представление зависимости в виде класса*

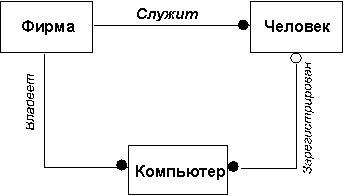
**16.1. Определение зависимостей.**

На следующем этапе построения объектной модели определяются зависимости между классами. Прежде всего из классов исключаются атрибуты, являющиеся явными ссылками на другие классы; такие атрибуты заменяются зависимостями. Смысл такой замены в том, что зависимости представляют собой абстракцию того же уровня, что и классы, и потому не оказывают непосредственного влияния на будущую реализацию (ссылка на класс лишь один из способов реализации зависимостей).

Аналогично тому, как имена возможных классов получались из существительных, встречающихся в предварительной постановке прикладной задачи, имена возможных зависимостей могут быть получены из *глаголов* или *глагольных оборотов*, встречающихся в указанном документе. Так обычно описываются: физическое положение (следует\_за, является\_частью, содержится\_в), направленное действие (приводит\_в\_движение), общение (разговаривает\_с), принадлежность (имеет, является\_частью) и т.п. Пример выделения явных и неявных глагольных оборотов из предварительной постановки конкретной прикладной задачи рассмотрен в п. [2.3.3.](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml)

Затем следует убрать ненужные или неправильные зависимости, используя следующие критерии:

* *зависимости между исключенными классами* должны быть исключены, либо переформулированы в терминах оставшихся классов (см. пример в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml));
* *нерелевантные зависимости* и зависимости, связанные с реализацией, должны быть исключены (см. пример в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml));
* *действия:* зависимость должна описывать структурные свойства прикладной области, а не малосущественные события (см. примеры в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml));
* *тренарные зависимости:* большую часть зависимостей между тремя или большим числом классов можно разложить на несколько бинарных зависимостей, используя в случае необходимости квалификаторы (см. примеры в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml)); в некоторых (очень редких) случаях такое разложение осуществить не удается; например, тренарная зависимость "Профессор читает курс в аудитории 628" не может быть разложена на бинарные без потери информации;
* *производные зависимости:* нужно исключать зависимости, которые можно выразить через другие зависимости, так как они избыточны (см. пример в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml)); при исключении избыточных (производных) зависимостей нужно быть особенно осторожным, так как не все дублирующие одна другую зависимости между классами избыточны; в некоторых случаях другие зависимости позволяют установить только существование еще одной производной зависимости, но не позволяют установить кратность этой зависимости; например, в случае, представленном на рисунке 2.36, фирма имеет много служащих и владеет многими компьютерами; каждому служащему предоставлено для персонального использования несколько компьютеров, кроме того, имеются компьютеры общего пользования; кратность зависимости предоставлен\_для\_использования не может быть выведена из зависимостей служит и владеет; хотя производные зависимости и не добавляют новой информации, они часто бывают удобны; в этих случаях их можно указывать на диаграмме, пометив косой чертой.



*Рис. 2.36. Неизбыточные зависимости*

Удалив избыточные зависимости, нужно уточнить семантику оставшихся зависимостей следующим образом:

* *неверно названные зависимости:* их следует переименовать, чтобы смысл их стал понятен (см. пример в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml));
* *имена ролей:* нужно добавить имена ролей там, где это необходимо; имя роли описывает роль, которую играет соответствующий класс в данной зависимости с точки зрения другого класса, участвующего в этой зависимости; если имя роли ясно из имени класса, его можно не указывать (см. пример в п. [2.3.3](http://citforum.ru/programming/oop_rsis/glava2_3_3.shtml));
* *квалификаторы:* добавляя квалификаторы там, где это необходимо, мы вносим элементы контекста, что позволяет добиться однозначной идентификации объектов; квалификаторы позволяют также упростить некоторые зависимости, понизив их кратность;
* *кратность:* необходимо добавить обозначения кратности зависимостей; при этом следует помнить, что кратность зависимостей может меняться в процессе дальнейшего анализа требований к системе;
* *неучтенные зависимости* должны быть выявлены и добавлены в модель.

**16.2. Уточнение атрибутов.**

На следующем этапе уточняется система атрибутов: корректируются атрибуты классов, вводятся, в случае необходимости, новые атрибуты. Атрибуты выражают свойства объектов рассматриваемого класса, либо определяют их текущее состояние.

Атрибуты обычно соответствуют существительным; например цвет\_автомобиля (свойство объекта), позиция\_курсора (состояние объекта). Атрибуты, как правило, слабо влияют на структуру объектной модели.

Не следует стремиться определить как можно больше атрибутов: большое количество атрибутов усложняет модель, затрудняет понимание проблемы. Необходимо вводить только те атрибуты, которые имеют отношение к проектируемой прикладной системе, опуская случайные, малосущественные и производные атрибуты.

Наряду с атрибутами объектов необходимо ввести и атрибуты зависимостей между классами (связей между объектами).

При уточнении атрибутов руководствуются следующими критериями:

* *Замена атрибутов на объекты*. Если наличие некоторой сущности важнее, чем ее значение, то это объект, если важнее значение, то это атрибут: например, начальник - это объект (неважно, кто именно начальник, главное, чтобы кто-то им был), зарплата - это атрибут (ее значение весьма существенно); город - всегда объект, хотя в некоторых случаях может показаться, что это атрибут (например, город как часть адреса фирмы); в тех случаях, когда нужно, чтобы город был атрибутом, следует определить зависимость (скажем, находится) между классами фирма и город.
* *Квалификаторы*. Если значение атрибута зависит от конкретного контекста, его следует сделать квалификатором (см. примеры в п. [2.3.4](http://citforum.ru/programming/oop_rsis/glava2_3_4.shtml)).
* *Имена*. Именам обычно лучше соответствуют квалификаторы, чем атрибуты объектов; во всех случаях, когда имя позволяет сделать выбор из объектов некоторого множества, его следует сделать квалификатором (см. примеры в п. [2.3.4](http://citforum.ru/programming/oop_rsis/glava2_3_4.shtml)).
* *Идентификаторы*. Идентификаторы объектов связаны с их реализацией. На ранних стадиях проектирования их не следует рассматривать в качестве атрибутов.
* *Атрибуты связей*. Если некоторое свойство характеризует не объект сам по себе, а его связь с другим объектом (объектами), то это атрибут связи, а не атрибут объекта.
* *Внутренние значения*. Атрибуты, определяющие лишь внутреннее состояние объекта, незаметное вне объекта, следует исключить из рассмотрения.
* *Несущественные детали*. Атрибуты, не влияющие на выполнение большей части операций, рекомендуется опустить.