2. Первая фаза жизненного цикла - анализ требований и предварительное проектирование системы. Объектно-ориентированное моделирование

Как известно, проектирование прикладной программной системы начинается с анализа требований, которым она должна будет удовлетворять. Такой анализ проводится с целью понять назначение и условия эксплуатации системы настолько, чтобы суметь составить ее предварительный проект.

При объектно-ориентированном подходе анализ требований к системе сводится к разработке моделей этой системы. Моделью системы (или какого-либо другого объекта или явления) мы называем формальное описание системы, в котором выделены основные объекты, составляющие систему, и отношения между этими объектами. Построение моделей - широко распространенный способ изучения сложных объектов и явлений. В модели опущены многочисленные детали, усложняющие понимание. Моделирование широко распространено и в науке, и в технике.

Модели помогают:

- проверить работоспособность разрабатываемой системы на ранних этапах ее разработки;
- общаться с заказчиком системы, уточняя его требования к системе;
- вносить (в случае необходимости) изменения в проект системы (как в начале ее проектирования, так и на других фазах ее жизненного цикла).

В настоящее время существует несколько технологий объектно-ориентированной разработки прикладных программных систем, в основе которых лежит построение и интерпретация на компьютере моделей этих систем. Мы подробно ознакомимся с одной из таких технологий - OMT (Object Modeling Techniques). Эта технология оказала большое влияние на других разработчиков объектно-ориентированных технологий, а книга, в которой она описана, является одной из наиболее часто цитируемых книг по данному направлению. Более того, система обозначений (графический язык) для описания моделей, предложенная в этой книге, широко применяется в других технологиях и в статьях по объектно-ориентированной разработке программных систем.

В технологии OMT проектируемая программная система представляется в виде трех взаимосвязанных моделей:

- объектной модели, которая представляет статические, структурные аспекты системы, в основном связанные с данными;
- динамической модели, которая описывает работу отдельных частей системы;
- функциональной модели, в которой рассматривается взаимодействие отдельных частей системы (как по данным, так и по управлению) в процессе ее работы.

Эти три вида моделей позволяют получить три взаимно-ортогональных представления системы в одной системе обозначений. Совокупность моделей системы может быть проинтерпретирована на компьютере (с помощью инструментального программного обеспечения), что позволяет продемонстрировать заказчику характер работы с будущей системой и существенно упрощает согласование предварительного проекта системы.

Модели, разработанные и отлаженные на первой фазе жизненного цикла системы, продолжают использоваться на всех последующих его фазах, облегчая программирование системы, ее отладку и тестирование, сопровождение и дальнейшую модификацию.

Как будет показано в дальнейшем, модели системы не связаны с языком программирования, на котором будет реализована система.

2.1. Объектная модель системы

Объектная модель описывает структуру объектов, составляющих систему, их атрибуты, операции, взаимосвязи с другими объектами. В объектной модели должны быть отражены те понятия и объекты реального мира, которые важны для разрабатываемой системы. В объектной модели отражается прежде всего прагматика разрабатываемой системы, что выражается в использовании терминологии прикладной области, связанной с использованием разрабатываемой системы.

2.1.1. Объекты и классы

Объекты

По определению будем называть объектом понятие, абстракцию или любую вещь с четко очерченными границами, имеющую смысл в контексте рассматриваемой прикладной проблемы. Введение объектов преследует две цели:

- понимание прикладной задачи (проблемы);
- введение основы для реализации на компьютере.

Примеры объектов: форточка, Банк "Империал", Петр Сидоров, дело № 7461, сберкнижка и т.д.

Цель разработки объектной модели - описать объекты, составляющие в совокупности проектируемую систему, а также выявить и указать различные зависимости между объектами. Декомпозиция проблемы на объекты - творческий, плохо формализуемый процесс.

Все объекты могут быть отличены один от другого: пусть у нас есть два яблока, имеющие одинаковый цвет, форму, вес и вкус; все равно это два яблока (а не одно), в чем легко убедиться, съев одно из них (другое останется). Между объектами можно установить отношение тождества: объекты, удовлетворяющие этому отношению, одинаковы (тождественны), как вышеупомянутые яблоки. В случае с яблоками иногда говорят о двух экземплярах объекта яблоко. Мы будем считать здесь, что объект и экземпляр объекта - это одно и то же.

Классы

Два яблока из предыдущего примера принадлежат одному и тому же классу объектов (именно с этим связана их одинаковость). Цвет, форма, вес и вкус яблока - это его атрибуты: совокупность атрибутов и их значений (например, красное, овальное, стограммовое, кисло-сладкое) характеризует объект.

Все объекты одного и того же класса характеризуются одинаковыми наборами атрибутов. Однако объединение объектов в классы определяется не наборами атрибутов, а семантикой. Так, например, объекты конюшня и лошадь могут иметь одинаковые атрибуты: цена и

возраст. При этом они могут относиться к одному классу, если рассматриваются в задаче просто как товар, либо к разным классам, что более естественно.

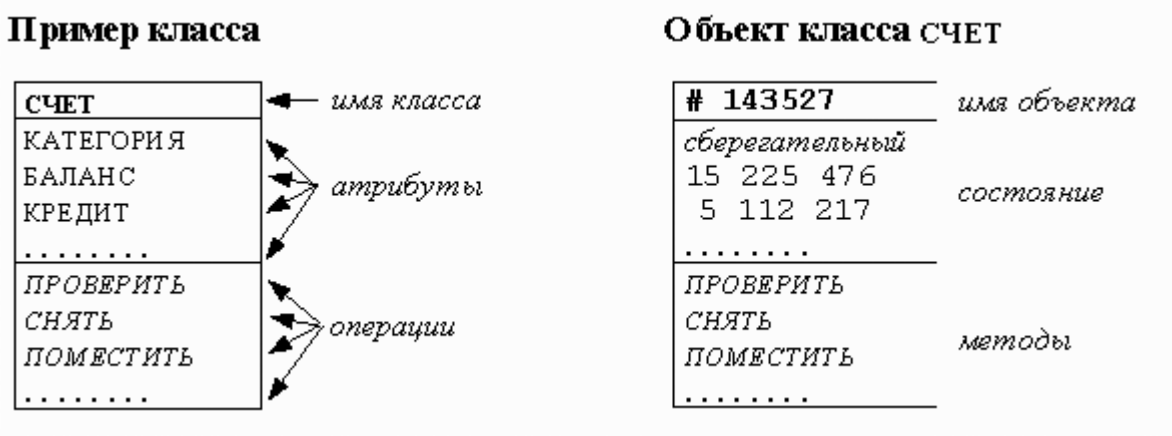


Рис. 2.1. Пример класса и объекта этого класса

Объединение объектов в классы позволяет ввести в задачу абстракцию и рассмотреть ее в более общей постановке. Класс имеет имя (например лошадь), которое относится ко всем объектам этого класса. Кроме того, в классе вводятся имена атрибутов, которые определены для объектов. В этом смысле описание класса аналогично описанию типа структуры (записи); при этом каждый объект имеет тот же смысл, что и экземпляр структуры (переменная или константа
соответствующего типа). Пример класса и объекта этого класса приведен на рисунке 2.1.

2.1.2. Атрибуты объектов

Атрибут - это значение, характеризующее объект в его классе. Примеры атрибутов: категория, баланс, кредит (атрибуты объектов класса счет); имя, возраст, вес (атрибуты объектов класса человек) и т.д.

Среди атрибутов различаются постоянные атрибуты (константы) и переменные атрибуты. Постоянные атрибуты характеризуют объект в его классе (например, номер счета, категория, имя человека и т.п.). Текущие значения переменных атрибутов характеризуют текущее состояние объекта (например, баланс счета, возраст человека и т.п.); изменяя значения этих атрибутов, мы изменяем состояние объекта.

Атрибуты перечисляются во второй части прямоугольника, изображающего класс (см. рисунок 2.1). Иногда указывается тип атрибутов (ведь каждый атрибут - это некоторое значение) и начальное значение переменных атрибутов (совокупность начальных значений этих атрибутов задает начальное состояние объекта).

Следует отметить, что, говоря об объектах и их классах, мы не подразумеваем никакого объектно-ориентированного языка программирования. Это, в частности, выражается в том, что на данном этапе разработки программной системы следует рассматривать только такие атрибуты, которые имеют смысл в реальности (все атрибуты объектов класса счет - рисунок 2.1 - обладают этим свойством). Атрибуты связаны с особенностями общей реализации. Например, если известно, что будет использоваться база данных, в которой каждый объект имеет уникальный идентификатор, то включать этот идентификатор в число атрибутов объекта на

данном этапе не следует. Дело в том, что, вводя такие атрибуты, мы ограничиваем возможности реализации системы. Так, вводя в качестве атрибута уникальный идентификатор объекта в базе данных, мы уже в самом начале проектирования отказываемся от использования СУБД, которые такой идентификатор не поддерживают.

2.1.3. Операции и методы

Операция - это функция (или преобразование), которую можно применять к объектам данного класса. Примеры операций: проверить, снять, поместить (для объектов класса счет - рисунок 2.1), открыть_на_чтение, читать, закрыть (для объектов класса файл - рисунок 2.2) и т.п.

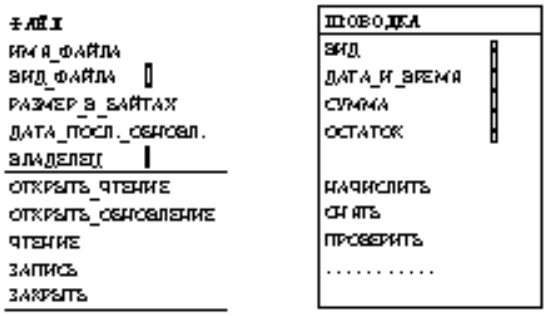


Рис. 2.2. Другие примеры классов

Все объекты данного класса используют один и тот же экземпляр каждой операции (т.е. увеличение количества объектов некоторого класса не приводит к увеличению количества загруженного программного кода). Объект, из которого вызвана операция, передается ей в качестве ее неявного аргумента (параметра).

Одна и та же операция может, вообще говоря, применяться к объектам разных классов: такая операция называется полиморфной, так как она может иметь разные формы для разных классов. Например, для объектов классов вектор и комплексное_число можно определить операцию +; эта операция будет полиморфной, так как сложение векторов и сложение комплексных чисел, вообще говоря, разные операции.

Каждой операции соответствует метод - реализация этой операции для объектов данного класса. Таким образом, операция - это спецификация метода, метод - реализация операции. Например, в классе файл может быть определена операция печать (print). Эта операция может быть реализована разными методами: (а) печать двоичного файла; (б) печать текстового файла и др. Логически эти методы выполняют одну и ту же операцию, хотя реализуются они разными фрагментами кода.

Каждая операция имеет один неявный аргумент - объект к которому она применяется. Кроме того, операция может иметь и другие аргументы (параметры).
Эти дополнительные аргументы параметризуют операцию, но не связаны с выбором метода. Метод связан только с классом и объектом (некоторые объектно-ориентированные языки, например C++, допускают одну и ту же операцию с разным числом аргументов, причем используя то или иное число аргументов, мы практически выбираем один из методов, связанных с такой операцией; на этапе предварительного проектирования системы удобнее

считать эти операции различными, давая им разные имена, чтобы не усложнять проектирование).

Операция (и реализующие ее методы) определяется своей сигнатурой, которая включает, помимо имени операции, типы (классы) всех ее аргументов и тип (класс) результата (возвращаемого значения). Все методы, реализующие операцию должны иметь такую же сигнатуру, что и реализуемая ими операция.

Операции перечисляются в третьей части прямоугольника (рисунок 2.1), описывающего класс. Каждая операция должна быть представлена своей сигнатурой, однако на ранних стадиях проектирования можно ограничиваться указанием имени операции, отложив полное определение сигнатуры на конец рассматриваемой фазы жизненного цикла (либо даже на последующие фазы). В графическом языке технологии OMT тип любого объекта данных указывается вслед за именем этого объекта после двоеточия (как в языке Паскаль).

При моделировании системы, полезно различать операции, имеющие побочные эффекты (эти эффекты выражаются в изменении значений атрибутов объекта, т.е. в изменении его состояния), и операции, которые выдают требуемое значение, не меняя состояния объекта. Эти последние операции называются запросами.

Значения некоторых атрибутов объекта могут быть доступны только операциям этого объекта. Такие атрибуты называются закрытыми. На рисунке 2.3 показаны закрытые атрибуты для объектов класса счет. Значения закрытых атрибутов объекта можно узнать вне объекта только в том случае, если среди операций этого объекта определены соответствующие запросы. Аналогично, в объекте можно определить и закрытые (вспомогательные) операции, однако на ранних стадиях проектирования этого, как правило, не делают, так как выделение закрытых операций связано, в основном, с реализацией системы.

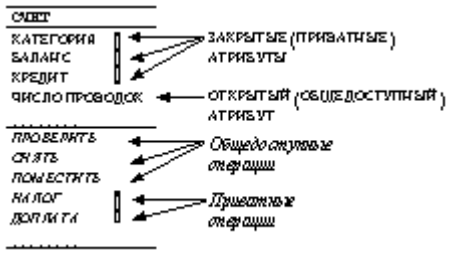


Рис 2.3. Открытые и закрытые атрибуты и операции

Запросы без аргументов (за исключением неявного аргумента - объекта, к которому применяется операция) могут рассматриваться как производные атрибуты. Значения производных атрибутов зависят от значений основных атрибутов. В этом их отличие от основных атрибутов, значения которых независимы. Следовательно, значения основных атрибутов объекта определяют как его состояние, так и значения его производных атрибутов. Так, например, длина, ширина и высота комнаты - ее основные атрибуты, а площадь и кубатура - производные атрибуты; такой атрибут как кубатура нужен для того, чтобы не вычислять кубатуру комнаты всякий раз, когда понадобится ее значение.

Выбор основных атрибутов объектов произволен, но в число основных атрибутов не следует включать такие атрибуты, значения которых определяются значениями других атрибутов, так что на самом деле они являются производными.

Таким образом, для задания класса необходимо указать имя этого класса, а затем перечислить его атрибуты и операции (или методы). Полное описание объекта на графическом языке OMT имеет вид, изображенный на рисунке 2.4. Однако иногда удобно бывает пользоваться сокращенным описанием класса, когда в прямоугольнике, изображающем этот класс, указывается только имя класса. Так, на рисунке 2.5 приведены сокращения обозначения классов для нашего основного примера - системы обслуживания клиентов банковского консорциума.

ИМЯ_КЛАССА
имя_атрибута 1: тип_1 = значение_по_умолчанию_1
имя_атрибута 2: тип_2 = значение_по_умолчанию_2
. . . . . .
имя_операции_1 (список_аргументов_1): тип_результата
сигнатура_операции_2
сигнатура_операции_3
. . . . . .

Рис. 2.4. Полное представление объекта в OMT

Рис. 2.5. Возможные классы для системы AMT (банковское обслуживание)

2.1.4. 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).

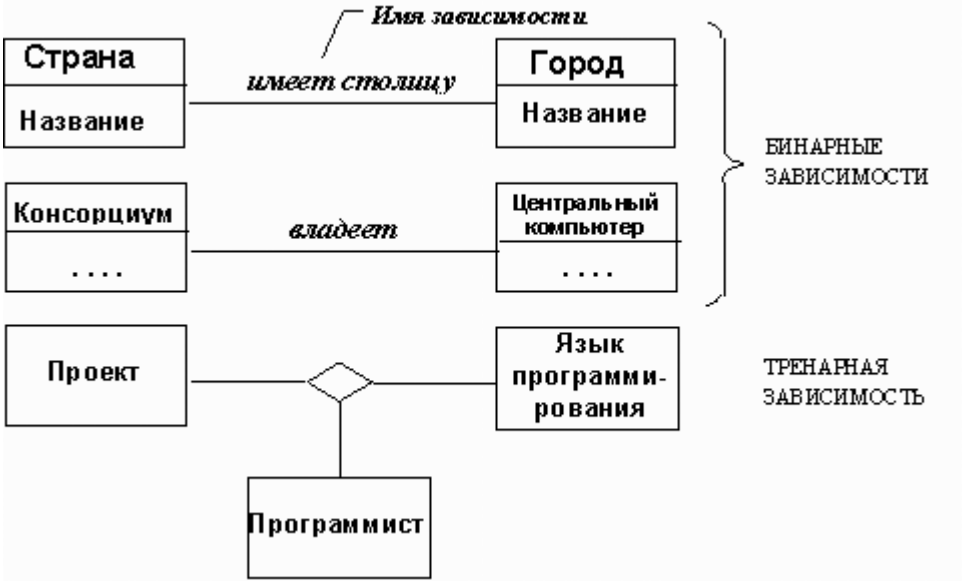


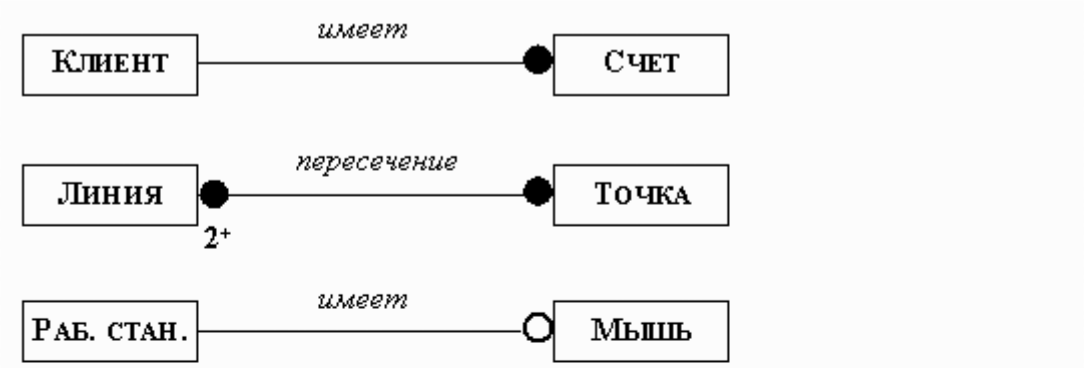
Рис. 2.6. Зависимости между классами

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.



Рабочая станция может не иметь мыши либо иметь одну мышь

Обозначения:

● означает, что клиент имеет несколько счетов (несколько - это 0,1,2, ... , но конечное число)

● 2+ означает, что речь идет о двух и более линиях

○ означает, что рабочая станция не обязательно имеет одну мышь

Рис. 2.7. Дальнейшие примеры зависимостей. Обозначения

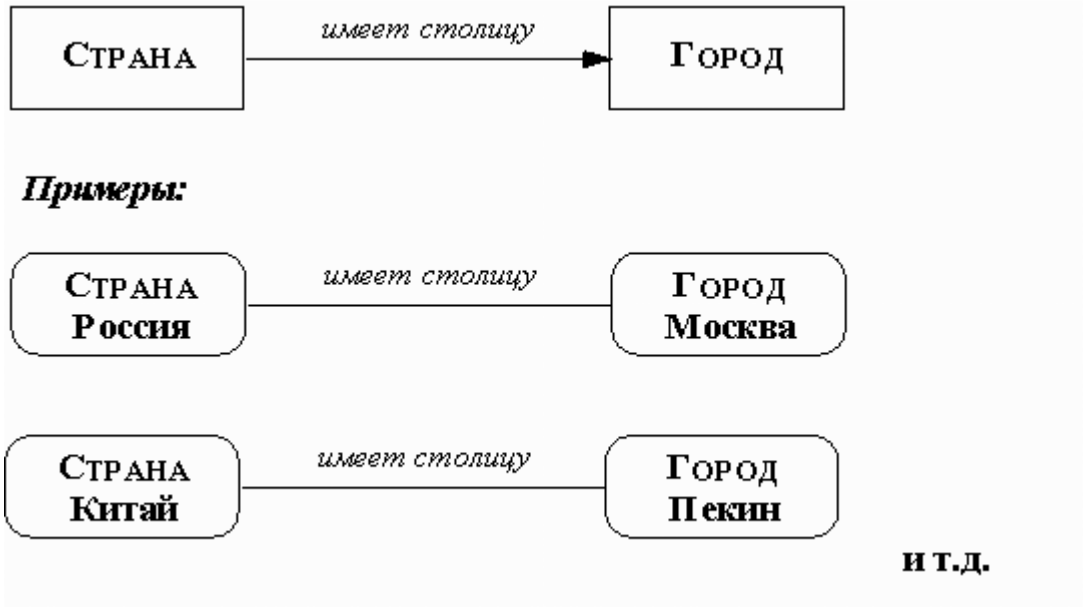


Рис. 2.8. Зависимости между объектами

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.

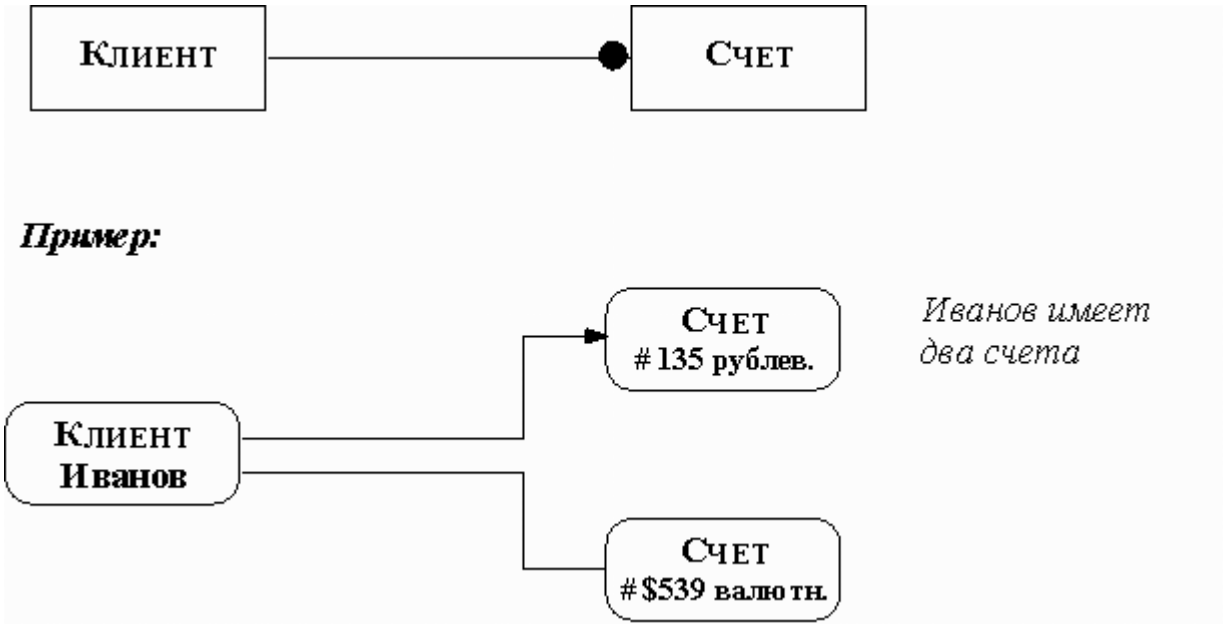


Рис. 2.9. Более сложные зависимости между объектами

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.

2.1.5. 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.

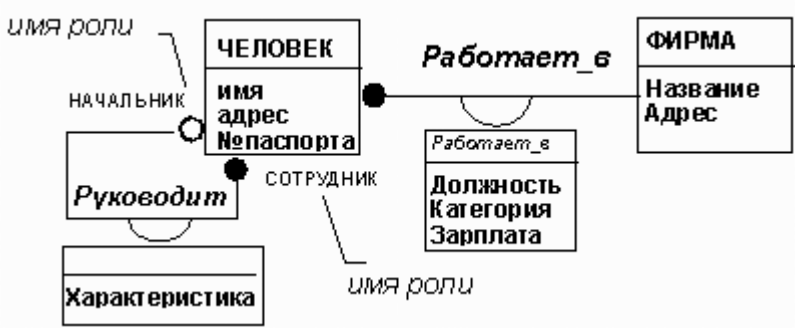


Рис. 2.11. Атрибуты двух зависимостей между одним и многими

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.

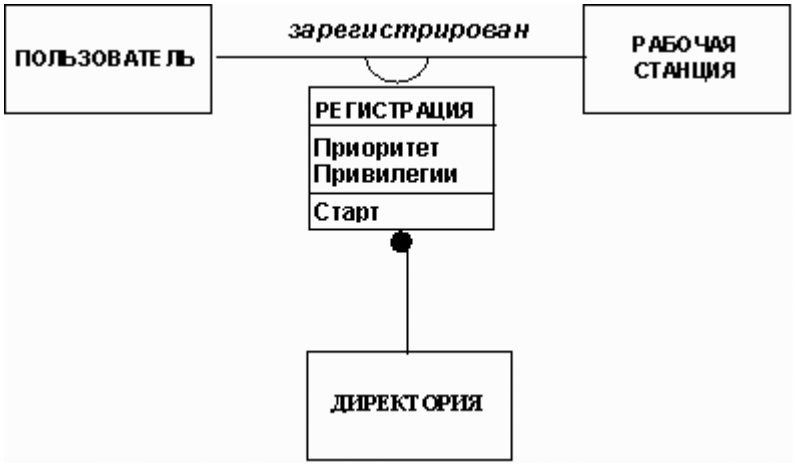


Рис. 2.12. Представление зависимости в виде класса

2.1.6. Имена ролей, квалификаторы

Роль определяет одну сторону зависимости. В бинарной зависимости определены две роли. Имя роли однозначно определяет одну сторону зависимости. Роли дают возможность

рассматривать бинарную зависимость как связь между объектом и множеством зависимых объектов: каждая роль является обозначением объекта или множества объектов, связанных зависимостью с объектом на другом конце зависимости. Имя роли можно рассматривать как производный атрибут, множеством значений которого является множество связанных с этой ролью объектов. В бинарной зависимости пара имен ролей может использоваться для идентификации этой зависимости.

На рисунке 2.11 имена начальник и сотрудник в зависимости руководит - это имена ролей; как уже было отмечено, эту зависимость удобнее назвать начальник-сотрудник. Еще один пример имен ролей показан на рисунке 2.13.

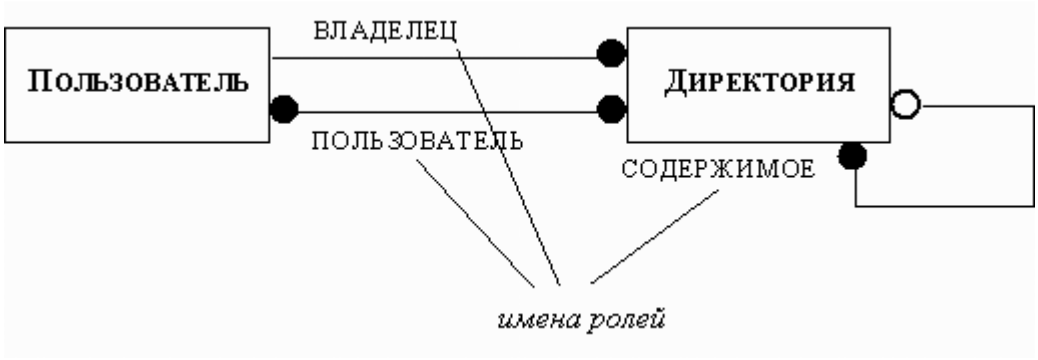


Рис. 2.13. Имена ролей

Пользователь может быть либо владельцем, либо зарегистрированным пользователем директории; директория может содержать в себе другие директории.

Имена ролей следует обязательно указывать в тех случаях, когда зависимость устанавливается между объектами одного и того же класса (как в случаях, показанных на рисунках 2.11 и 2.13). Имена ролей должны быть уникальны, так как они используются для различения объектов, участвующих в зависимости.

Квалификатором называется некоторый атрибут, который позволяет снизить эффективную кратность зависимости. Квалификаторы применяются в зависимостях типов "один-ко-многим" или "много-ко-многим". Так в примере, показанном на рисунке 2.14, использование квалификатора имя_файла позволяет привести зависимость дает_доступ от вида, приведенного на рисунке 2.14(а), к виду, показанному на рисунке 2.14(б), сократив число зависимых объектов до одного.
Еще один пример использования квалификатора показан на рисунке 2.14(в): использование квалификаторов и здесь позволяет сократить кратность зависимости до одного объекта. Квалификаторы указываются на схемах в прямоугольничках, пририсованных к прямоугольнику, изображающему соответствующий класс.

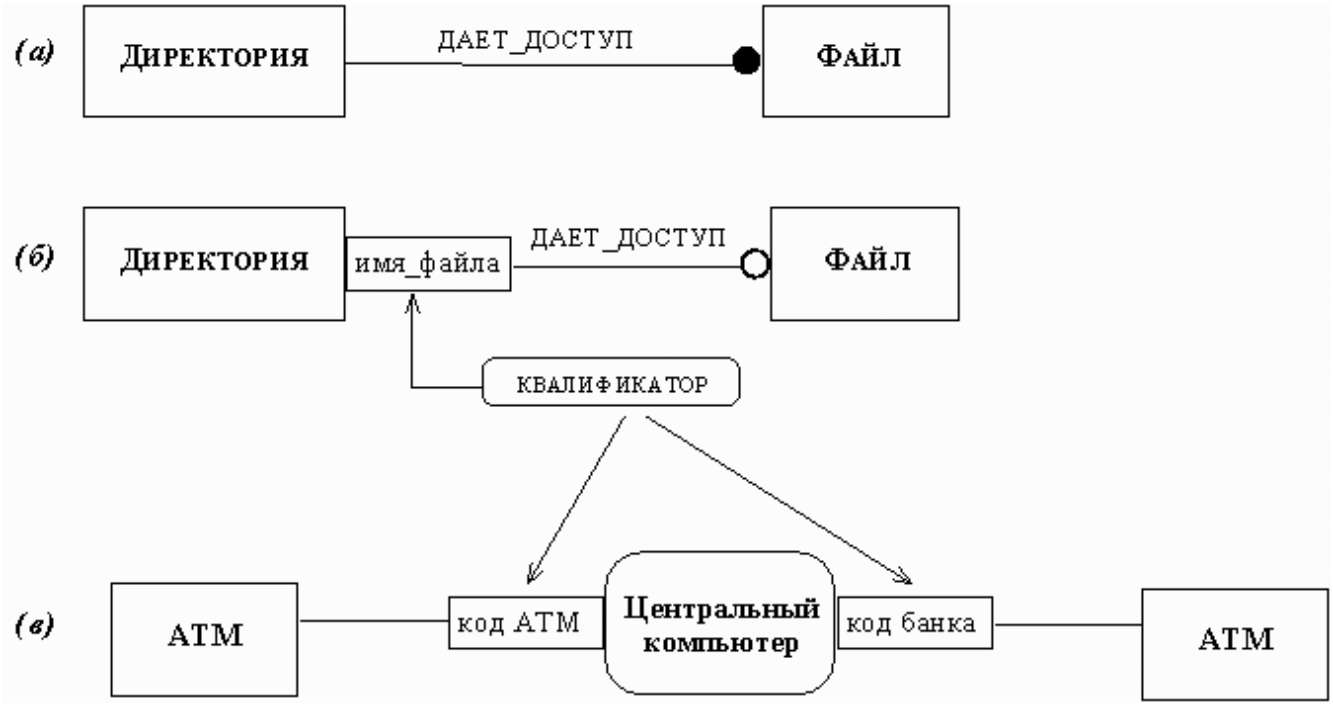


Рис. 2.14. Использование квалификаторов

Рисунок 2.14(в) может быть проинтерпретирован следующим образом: центральный компьютер + код_АТМ определяют конкретную АТМ (отметим, что код_АТМ - имя одного из атрибутов класса АТМ, а не класса центральный_компьютер); аналогично центральный_компьютер + код_банка определяют конкретный компьютер банка. Использование квалификаторов повышает точность описания семантики и наглядность описания зависимостей.

2.1.7. Агрегация

Агрегация - это зависимость между классом составных объектов и классами, представляющими компоненты этих объектов (отношение "целое"-"часть"). Агрегация обозначается ромбиком: на рисунке 2.15 приведен пример агрегации; этот пример интерпретируется следующим образом: документ состоит из нескольких (нуля, или более) абзацев; каждый абзац состоит из нескольких (нуля, или более) предложений.

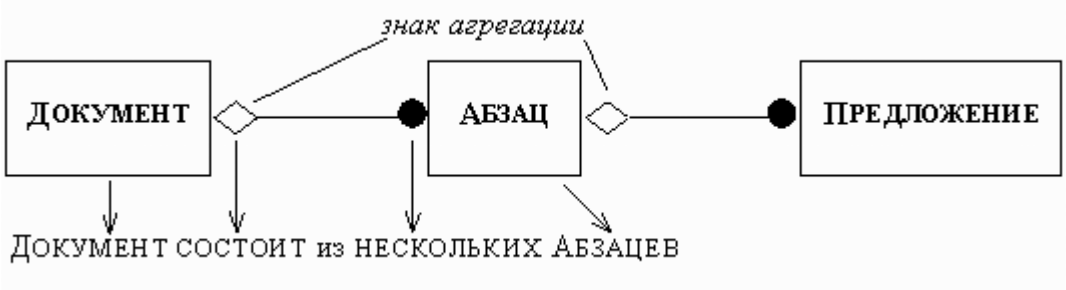


Рис. 2.15. Агрегация

Наиболее важным свойством отношения агрегации является его транзитивность (если А есть часть В, а В есть часть С, то А есть часть С): так, из рисунка 2.15 можно заключить, что документ состоит из нескольких (нуля, или более) предложений. Легко видеть, что отношение агрегации антисимметрично (если А есть часть В, то В не есть часть А). Отметим также, что часть свойств целого может быть перенесена и на его части, возможно, с несущественными изменениями (например, контекст каждого оператора некоторой функции совпадает с внутренним контекстом всей функции).

Дальнейшие примеры агрегации показаны на рисунке 2.16. Отметим, что обе агрегации, показанные на рисунке 2.16(а), следует рассматривать не как зависимости между пятерками классов, а как четверки зависимостей между парами классов. Только при таком рассмотрении можно говорить о транзитивности и антисимметричности отношения агрегации.

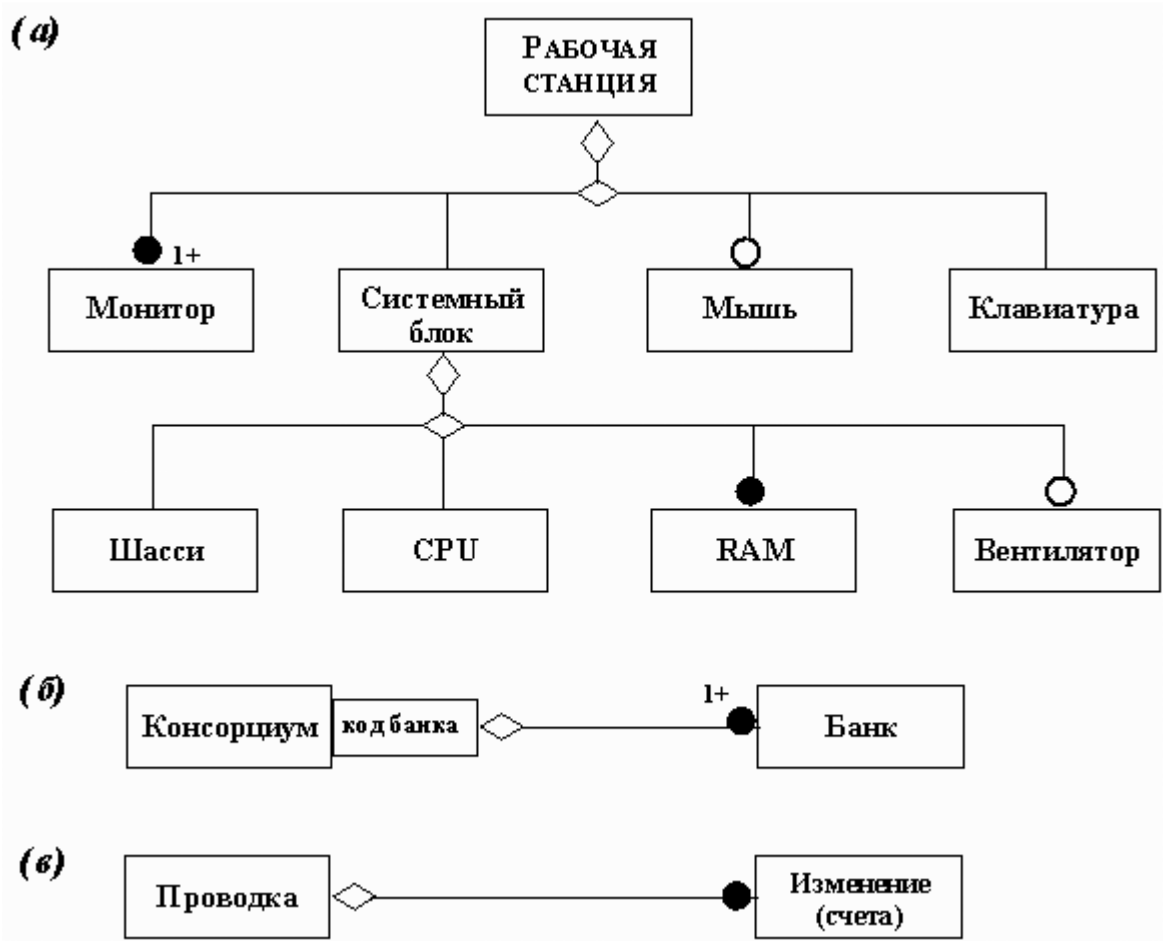


Рис. 2.16. Примеры агрегации

2.1.8. Обобщение и наследование

Обобщение и наследование позволяют выявить аналогии между различными классами объектов, определяют многоуровневую классификацию объектов. Так, в графических системах могут существовать классы, определяющие обрисовку различных геометрических

фигур: точек, линий (прямых, дуг окружностей и кривых, определяемых сплайнами), многоугольников, кругов и т.п. (рисунок 2.17).

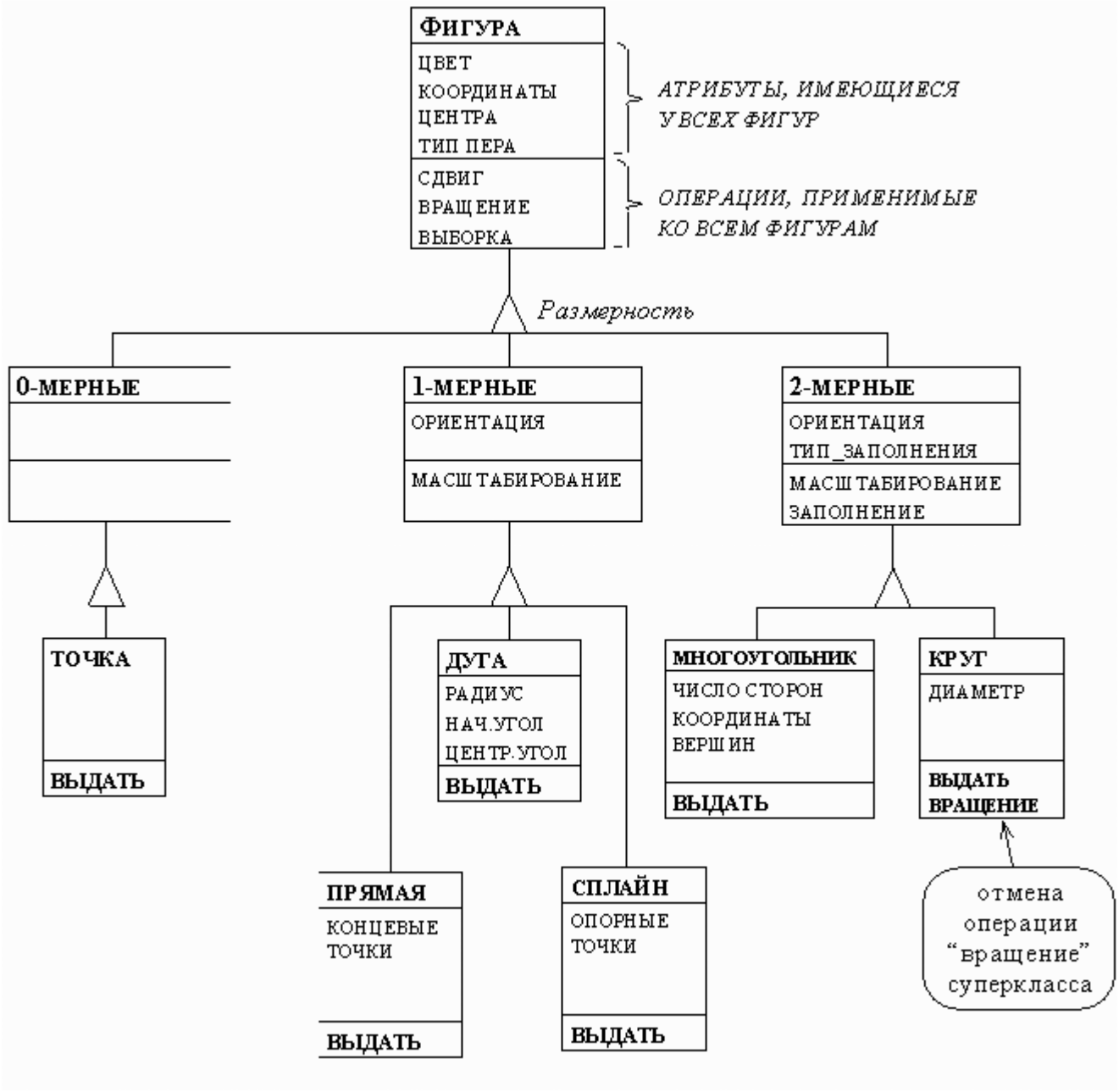


Рис. 2.17. Обобщение (выделение суперклассов)

Обобщение позволяет выделить класс одномерные фигуры и считать классы прямая, дуга и сплайн подклассами класса одномерные фигуры, а класс одномерные фигуры - суперклассом классов прямая, дуга и сплайн. Если при этом принять соглашение, что атрибуты и операции суперкласса действительны в каждом из его подклассов (говорят, что эти атрибуты и операции наследуются подклассами), то одинаковые атрибуты и операции классов прямая, дуга и сплайн (подклассов) могут быть вынесены в класс одномерные фигуры (суперкласс). Аналогично можно поступить и с двумерными фигурами, определив для классов многоугольник и круг суперкласс двумерная фигура. Наконец, можно определить класс

фигура как суперкласс классов нульмерная фигура, одномерная фигура и двумерная фигура. Легко видеть, что отношения "подкласс - суперкласс" (обобщение) и "суперкласс - подкласс" (наследование) транзитивны, что позволяет строить классификационные деревья. При этом атрибуты и операции каждого суперкласса наследуются его подклассами всех уровней (мы как бы выносим за скобки одинаковые операции). Это значительно облегчает и сокращает описание классов.

На схемах обобщение (наследование) изображается треугольничком (рисунок 2.17). Треугольничек следует ставить даже в том случае, когда суперкласс имеет всего один подкласс. Слово размерность, следующее за верхним треугольничком на рисунке 2.17, является дискриминатором.

Дискриминатор - это атрибут типа "перечисление", показывающий, по какому из свойств объектов сделано данное обобщение.
Другие примеры обобщения (наследования) показаны на рисунке 2.18 (эти примеры связаны с основным нашим примером - системой обслуживания клиентов банковским консорциумом).

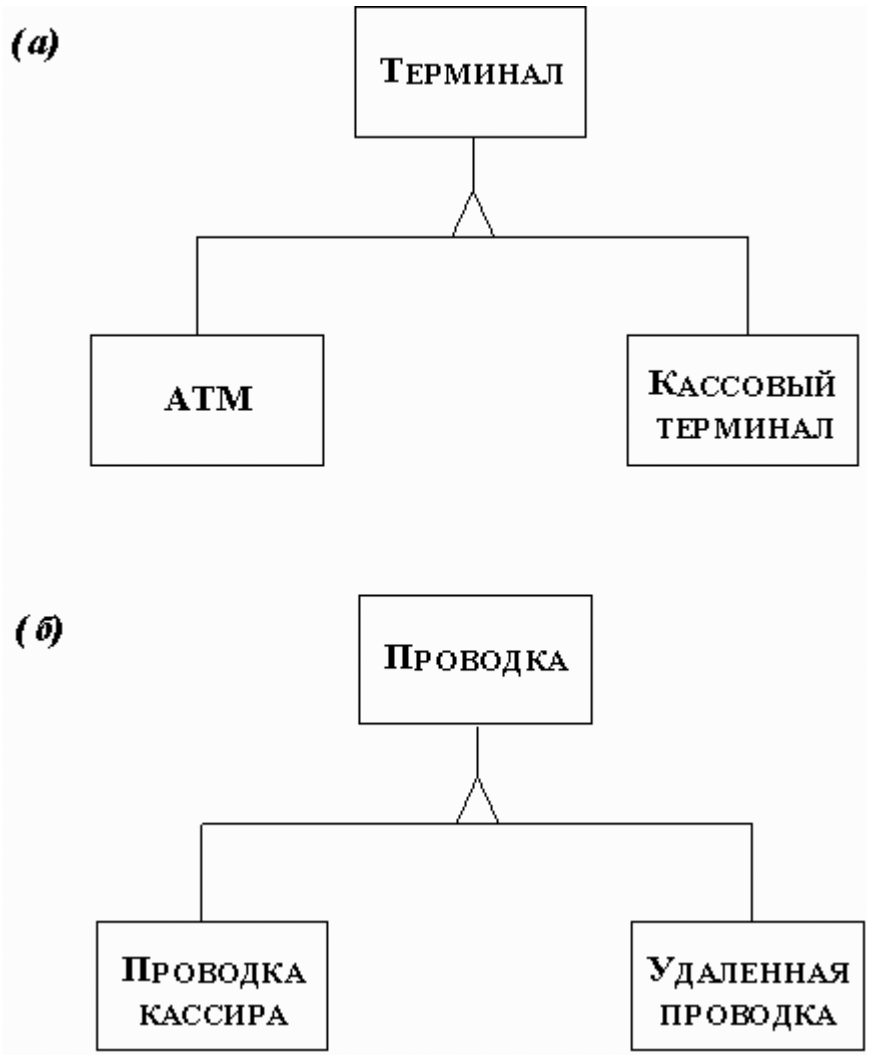


Рис. 2.18. Другие примеры обобщения (наследования)

Необходимо отметить, что, как показывает опыт практического проектирования систем, следует избегать обширных многоуровневых классификаций, так как поведение подклассов

низших уровней многоуровневой классификации бывает трудно понять: большая часть (а нередко и все) атрибутов и операций таких классов определена в их суперклассах различных уровней. Если количество уровней классификации стало непомерно большим, нужно слегка изменить структурирование системы. Чтобы понять, какое число уровней является непомерно большим, можно руководствоваться следующими оценками: два-три уровня наследования, как правило, приемлемы всегда (мне известна одна фирма, разрабатывающая программные системы, в которой издан стандарт фирмы, запрещающий более чем трехуровневые классификации в программах); десятиуровневая классификация почти всегда неприемлема; пять-шесть уровней, как правило, достаточно для программистов и не слишком обременяет администрацию.

Обобщение и наследование широко применяются не только при анализе требований к программным системам и их предварительном проектировании, но и при их реализации.

Иногда в подклассе бывает необходимо переопределить операцию, определенную в одном из его суперклассов. Для этого операция, которая может быть получена из суперкласса в результате наследования, определяется и в подклассе; это ее повторное определение "заслоняет" ее определение в суперклассе, так что в подклассе применяется не унаследованная, а переопределенная в нем операция. Напомним, что каждая операция определяется своей сигнатурой; следовательно, сигнатура переопределения операции должна совпадать с сигнатурой операции из суперкласса, которая переопределяется данной операцией. Так, в примере, изображенном на рисунке 2.17, в классе круг переопределяется операция вращение
его суперкласса фигура (при повороте круга его изображение не меняется, что позволяет сделать операцию вращение в классе круг пустой).

Переопределение может преследовать одну из следующих целей:

- расширение: новая операция расширяет унаследованную операцию, учитывая влияние атрибутов подкласса;
- ограничение: новая операция ограничивается выполнением лишь части действий унаследованной операции, используя специфику объектов подкласса;
- оптимизация: использование специфики объектов подкласса позволяет упростить и ускорить соответствующий метод (например, переопределяя операцию max класса IntegerSet в его подклассе SortedIntegerSet, мы можем резко упростить ее, используя специфические свойства упорядоченных множеств);
- удобство.

Целесообразно придерживаться следующих семантических правил наследования:

- все операции-запросы (операции, которые используют значения атрибутов, но не изменяют их) должны наследоваться всеми подклассами;
- все операции, изменяющие значения атрибутов, должны наследоваться во всех их расширениях;
- все операции, изменяющие значения ограниченных атрибутов, или атрибутов, определяющих зависимости, должны блокироваться во всех их расширениях (например, операция размер_по_оси_х естественна для класса эллипс, но должна быть заблокирована (заменена пустой операцией) в его подклассе круг);
- операции не следует переопределять *коренным образом*; все методы, реализующие одну и ту же операцию, должны осуществлять сходное преобразование атрибутов;
- унаследованные операции можно уточнять, добавляя дополнительные действия.

Следуя этим правилам, которые, к сожалению, редко поддерживаются объектно-ориентированными языками программирования, можно сделать разрабатываемую программу более понятной, легче модифицируемой, менее подверженной влиянию различных ошибок и недосмотров.

2.1.9. Абстрактные классы

Рассмотрим пример, представленный на рисунке 2.19. В нем рассмотрена операция подсчет выплат для различных категорий служащих фирмы: временных служащих с почасовой оплатой труда, постоянных служащих с понедельной оплатой труда и руководящих работников фирмы с помесячной оплатой.

Каждая из категорий служащих представлена своим подклассом класса служащий, от которого они наследуют атрибут годовой_доход и операцию подсчет_выплат. Но подсчет выплат для каждой категории служащих производится по-своему, с учетом значений их собственных (неунаследованных) атрибутов; поэтому в каждом из подклассов операция подсчет_выплат переопределяется. Следовательно, в суперклассе операция подсчет_выплат может быть определена произвольным образом, так как она никогда не будет выполняться. В то же время сигнатуры всех операций подсчет_выплат в суперклассе и в подклассах должны быть одинаковыми (иначе это будут разные операции). Из сказанного следует, что в суперклассе можно задать только сигнатуру операции подсчет_выплат, это обеспечит одинаковые сигнатуры этой операции во всех подклассах. Методы, реализующие операцию подсчет_выплат, достаточно определить только в подклассах класса служащий. Суперкласс, в котором заданы только атрибуты и сигнатуры операций, но не определены методы, реализующие его операции, называется абстрактным классом. Методы, реализующие операции абстрактного класса, определяются в его подклассах, которые называются конкретными классами.

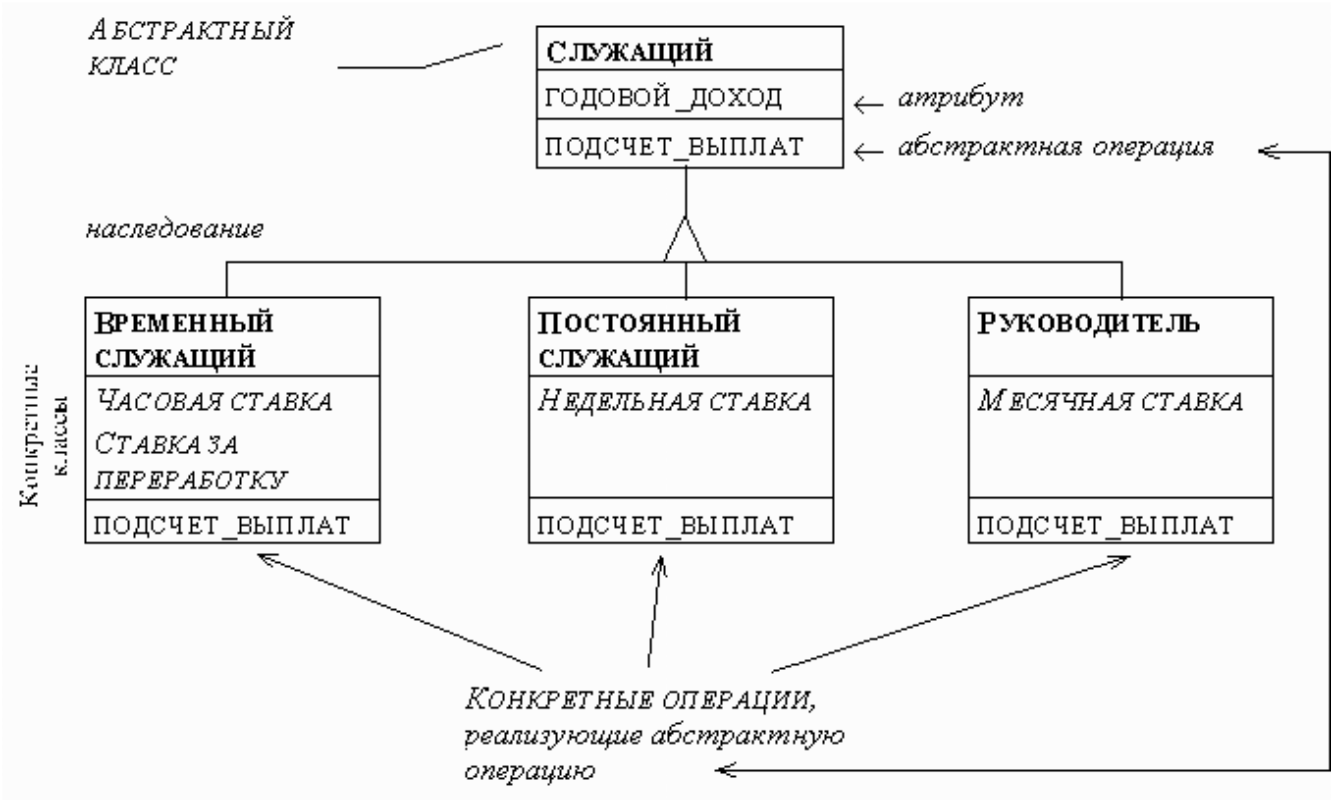


Рис. 2.19. Абстрактный класс

Абстрактный класс не может иметь объектов, так как в нем не определены операции над объектами; объекты должны принадлежать конкретным подклассам абстрактного класса. Абстрактные классы используются для спецификации интерфейсов операций (методы, реализующие эти операции впоследствии определяются в подклассах абстрактного класса). Абстрактные классы удобны на фазе анализа требований к системе, так как они позволяют выявить аналогию в различных, на первый взгляд, операциях, определенных в анализируемой системе.

2.1.10. Множественное наследование

Множественное наследование позволяет классу иметь более одного суперкласса, наследуя свойства (атрибуты и операции) всех своих суперклассов. Класс, имеющий несколько суперклассов, называется объединенным классом. Свойства класса-предка, встречающегося более, чем один раз, в графе наследования, наследуются только в одном экземпляре. Конфликты между параллельными определениями порождают двусмысленности, которые должны разрешаться во время реализации. На практике следует избегать таких двусмысленностей или плохого понимания даже в тех случаях, когда конкретный язык программирования, выбранный для реализации системы, предоставляет возможность их разрешения, используя приоритеты или какие-либо другие средства.

Пример множественного наследования приведен на рисунке 2.20, на котором рассмотрена классификация транспортных средств. Класс транспортное средство имеет два подкласса сухопутное_ТС и водное_ТС (зачерненный треугольник, используемый для обозначения наследования, означает, что подклассы имеют непустое пересечение). Класс амфибии имеет два суперкласса сухопутное_ТС и водное_ТС, наследуя все свойства (атрибуты и операции) как класса сухопутное_ТС, так и класса водное_ТС.

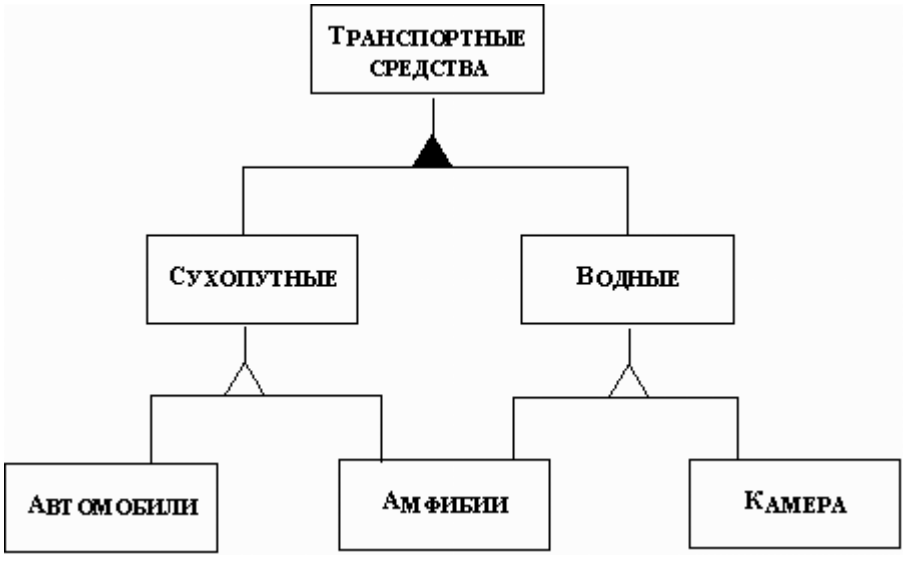


Рис. 2.20. Множественное наследование

Еще один пример множественного наследования приведен на рисунке 2.21, где рассмотрено множественное наследование от непересекающихся классов. В этом случае, который наиболее типичен для применения множественного наследования, свойства, унаследованные от разных предков, дополняют друг друга.

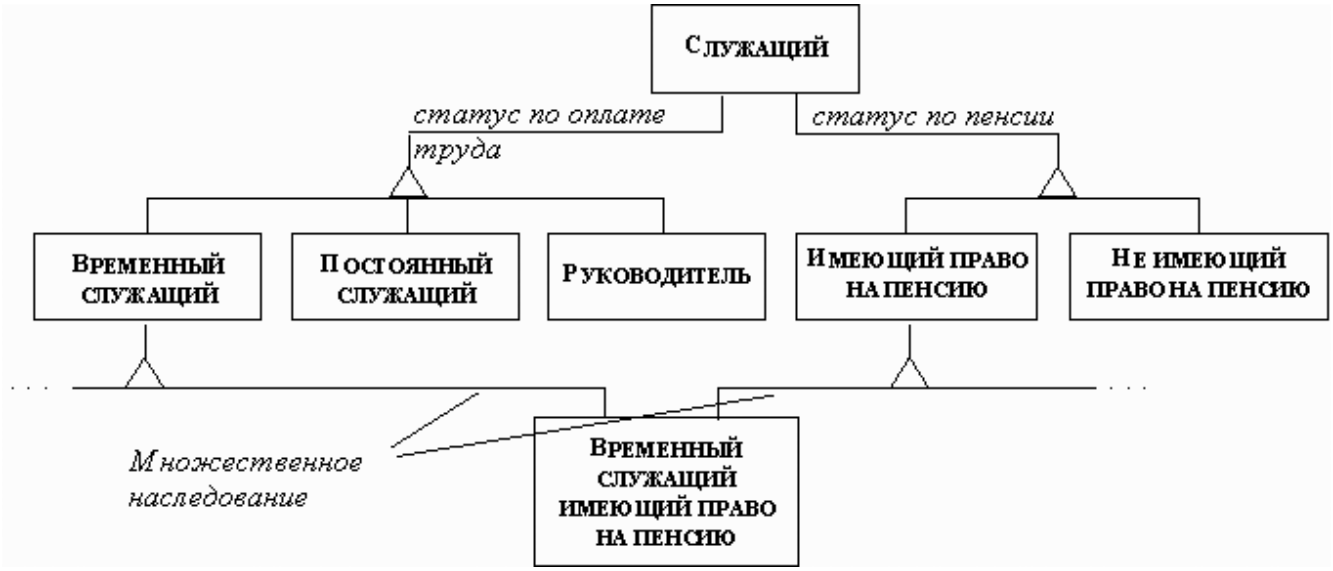


Рис. 2.21. Множественное наследование от непересекающихся классов

В случае, если множественное наследование не поддерживается языком программирования, выбранным для реализации, оно может быть заменено одним из следующих способов.

- Использование вложенного простого наследования представлено на рисунке 2.22.
- Делегирование с использованием агрегации ролей показано на рисунке 2.23. Делегированием называется механизм реализации, в котором объект, ответственный за операцию, пересылает (делегирует) эту операцию другому объекту; в объектно-ориентированных языках делегирование реализуется путем присоединения методов непосредственно к объектам, а не к классам.

В рассматриваемом примере операции классов оплата_труда и пенсионное_обеспечение делегируются объектам класса служащий, который можно рассматривать как результат агрегации классов оплата_труда и пенсионное_обеспечение. Более подробно делегирование будет рассмотрено в разделе 5.

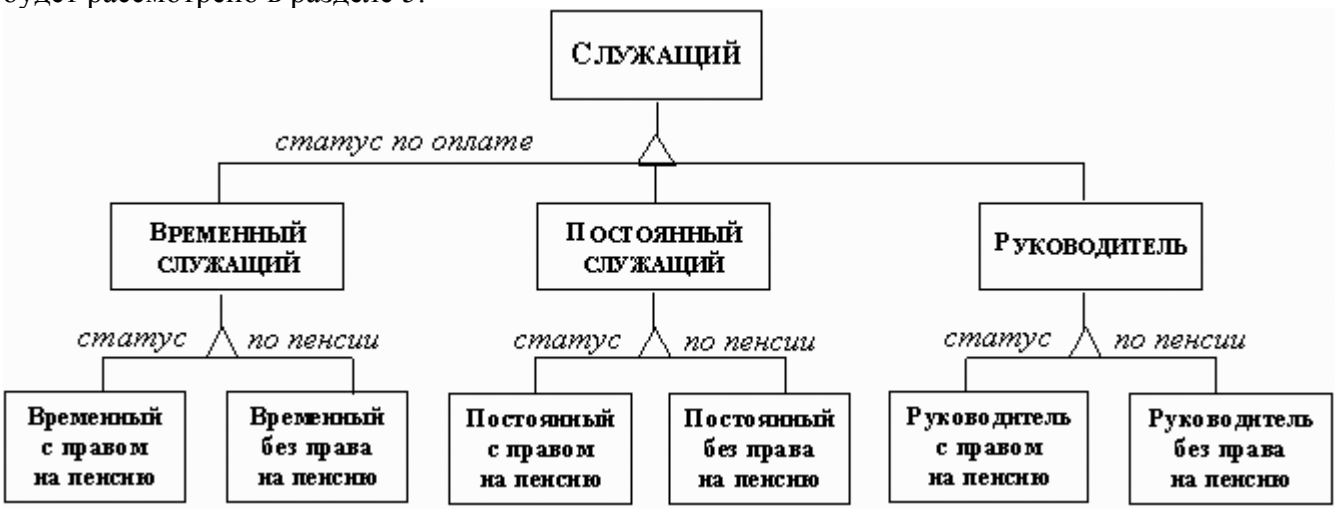


Рис. 2.22. Реализация множественного наследования с помощью вложенного простого наследования

Рис. 2.23. Реализация множественного наследования путем делегирования с использованием агрегации ролей

Еще один способ использования делегирования для реализации множественного наследования показан на рисунке 2.24. При этом способе суперкласс, наиболее существенный по передаче своих свойств, остается единственным суперклассом рассматриваемого подкласса, а свойства остальных суперклассов делегируются объектам этого подкласса.

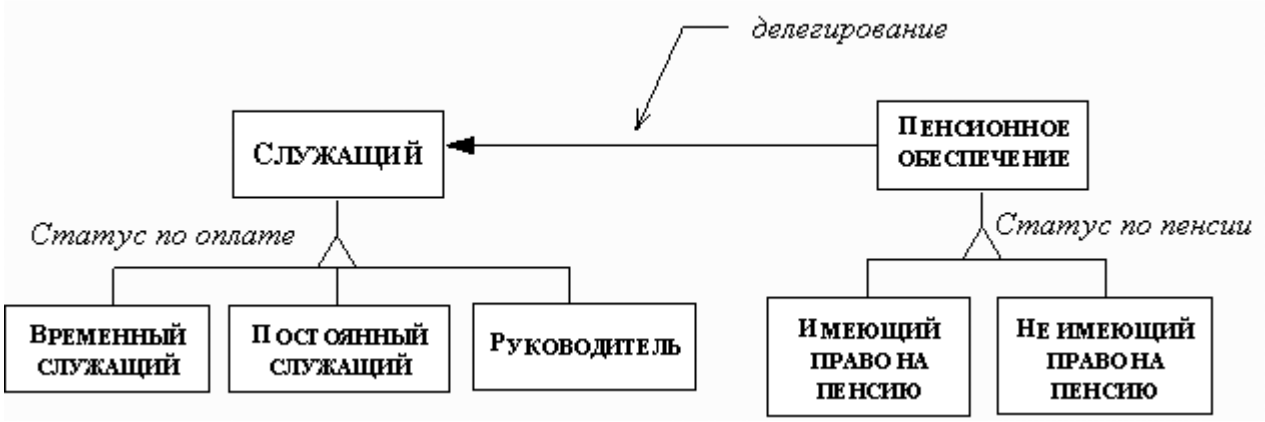


Рис. 2.24. Реализация множественного наследования с использованием простого наследования и делегирования

Возможны и другие способы замены множественного наследования. Во всех случаях при выборе способа замены множественного наследования нужно руководствоваться следующими правилами:

- если подкласс имеет несколько суперклассов, каждый из которых одинаково существен, лучше всего использовать делегирование (рисунок 2.23);
- если наиболее существенным является только один из суперклассов, а остальные не так важны, наилучшим способом является реализация множественного наследования через простое наследование и делегирование (рисунок 2.24);
- если число возможных комбинаций групп наследуемых свойств невелико, можно использовать вложенное простое наследование (рисунок 2.22); в случае большого числа комбинаций этот способ применять не следует;

- если один из суперклассов передает подклассу намного большее число свойств, чем остальные суперклассы, следует сохранить наследование по этому пути (это возможно в ситуациях, представленных на рисунках 2.22 и 2.24);
- если решено использовать вложенное простое наследование, то на первый уровень вложенности следует поместить наиболее существенный по передаче свойств суперкласс, затем наиболее существенный из оставшихся суперклассов и т.д. (рисунок 2.22);
- следует избегать использования вложенного простого наследования (рисунок 2.22), если это ведет к дублированию достаточно больших частей программы;
- следует помнить, что только вложенное простое наследование (рисунок 2.22) обеспечивает полную тождественность множественному наследованию.

2.1.11. Связь объектов с базой данных

В объектно-ориентированном проектировании мы имеем дело с множествами взаимосвязанных объектов. Каждый объект может рассматриваться как переменная или константа структурного типа (при таком рассмотрении методы, описываемые в объекте, трактуются как адреса функций, которые разрешено применять к этому объекту). Следовательно, множество объектов - это множество взаимосвязанных данных, т.е. нечто очень похожее на базу данных. Поэтому применение понятий баз данных часто оказывается полезным при объектно-ориентированном анализе и объектно-ориентированном проектировании прикладных программных систем.

Рассмотрим на примерах некоторые особенности применения указанных понятий.

Метаданные

Метаданными называются данные, описывающие другие данные. Например, определение класса - это метаданные, так как класс описывает другие данные - объекты этого класса. Модели являются метаданными, так как они описывают моделируемые объекты. Еще одним примером метаданных является абстрактный класс (см. раздел 2.1.9). Определение абстрактного и конкретного классов может быть представлено с помощью объектной модели, изображенной на рисунке 2.25.

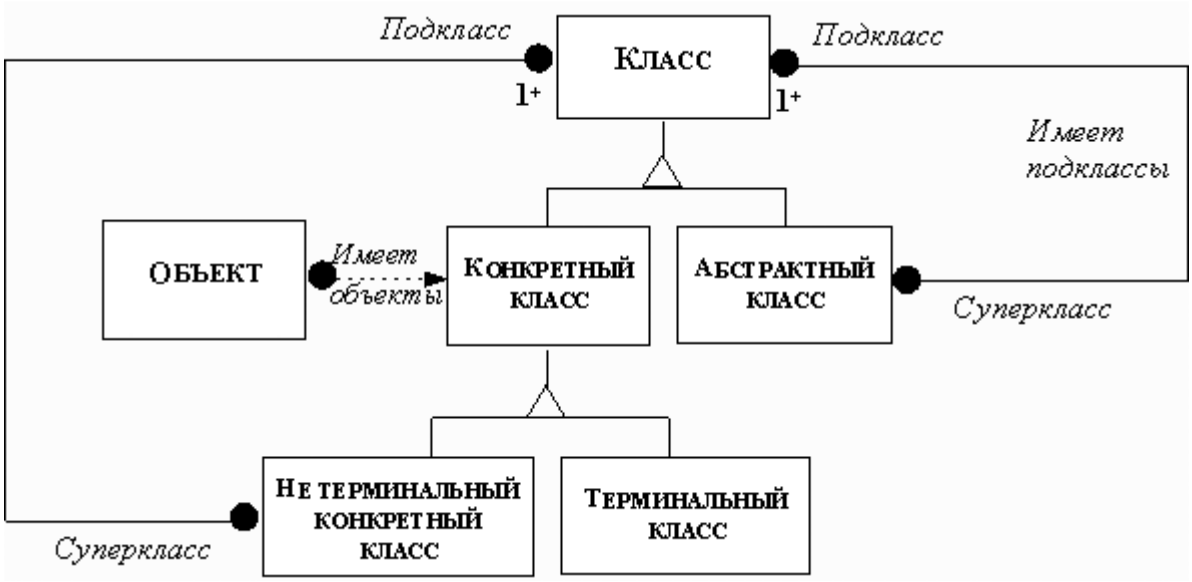


Рис. 2.25. Объектная модель, определяющая абстрактный и конкретный классы

Класс описывает множество объектов - экземпляров этого класса. Объекты данного класса порождаются по описанию класса с помощью процесса, называемого тиражированием. Процесс тиражирования можно распространить и на другие случаи порождения экземпляров объектов по образцам. Рассмотрим, например, модели автомобилей, выпускаемых различными производителями. Класс модель_автомобиля имеет свои атрибуты (как например, имя_модели, год_выпуска, базовая_цена) и зависимости (в частности, этот класс связан зависимостью с классом фирма). Но каждая модель_автомобиля может рассматриваться как метакласс, описывающий множество автомобилей, принадлежащих конкретным людям - их владельцам. Каждый класс автомобиль получает атрибуты от своего метакласса, но может иметь и собственные атрибуты (как например, серийный_номер, цвет, комплектация). При этом гораздо удобнее и экономнее получать различные классы автомобиль тиражированием метакласса модель_автомобиля (схематически этот процесс тиражирования представлен на рисунке 2.26). Тиражирование изображается пунктирной стрелкой.

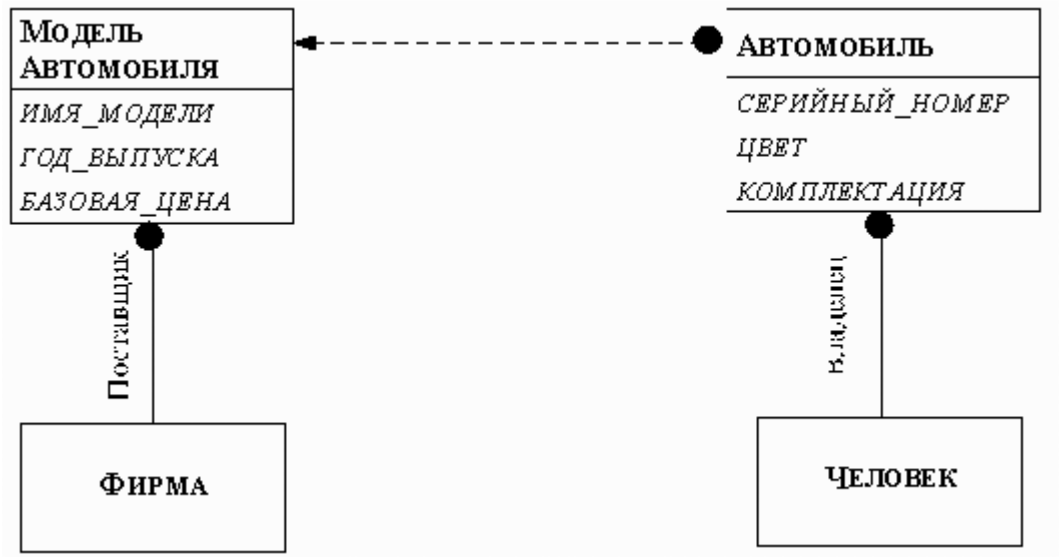


Рис. 2.26. Тиражирование метакласса

Использование возможных ключей

Возможным ключом (этот термин чаще употребляется в базах данных) называется минимальный набор атрибутов, который однозначно идентифицирует объект или связь. Связь - это экземпляр зависимости: каждая зависимость между классами порождает связи между объектами этих классов. Возможный ключ является минимальным набором атрибутов в том смысле, что если убрать из него хотя бы один из атрибутов, он уже не будет иметь различные значения для всех объектов или связей. Класс или зависимость могут иметь один или несколько возможных ключей, каждый из которых может представлять собой одну из комбинаций некоторого числа атрибутов. Одним из возможных ключей для любого класса всегда является идентификатор объекта. Одна или более комбинаций зависимых объектов являются возможными ключами для зависимости.

На рисунке 2.27 показаны возможные ключи для различных бинарных зависимостей. Связь возможных ключей с кратностью зависимостей очевидна и в комментариях не нуждается.

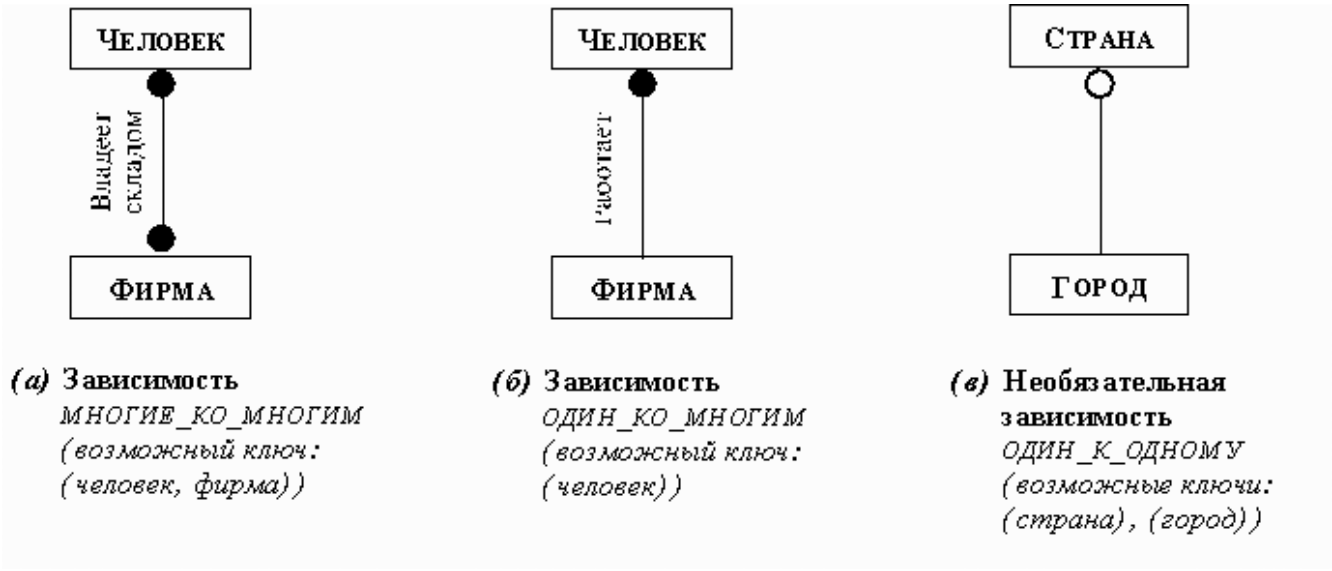


Рис. 2.27. Возможные ключи бинарных зависимостей

На рисунке 2.28 рассмотрены две тренарные зависимости. Первая из них определяется таблицей:

Проект	Исполнитель	Язык
САПР радиосхем	Иванов	Ada
ПО АСУ	Петров	C
Компилятор C++	Сидоров	C
САПР радиосхем	Кузнецов	Fortran
САПР радиосхем	Сидоров	Ada
САПР радиосхем	Сидоров	C
ПО АСУ	Сидоров	C++

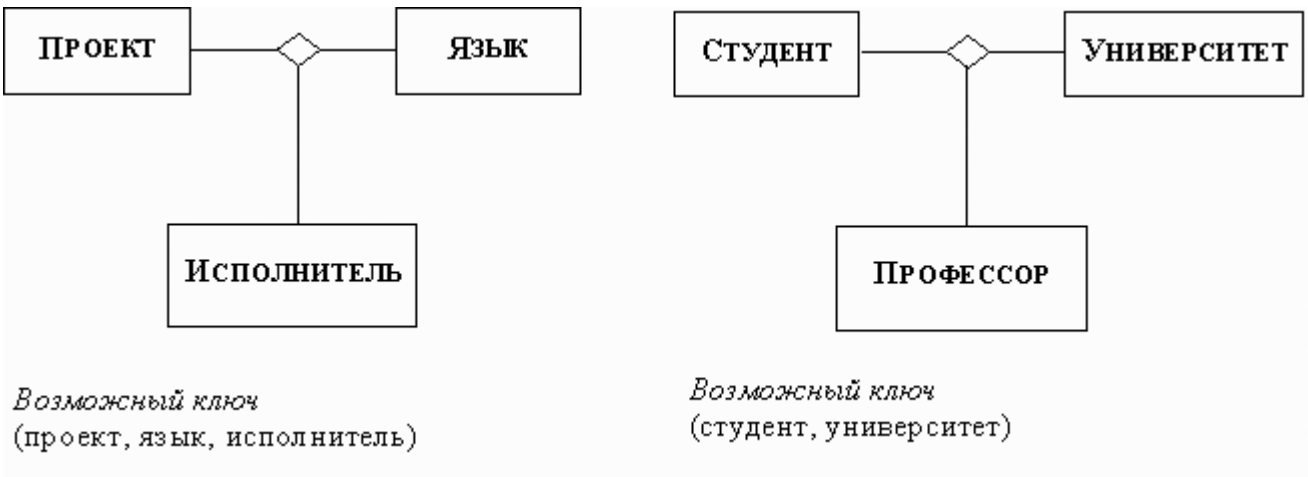


Рис. 2.28. Возможные ключи тренарных зависимостей

Легко видеть, что ни какой-либо из столбцов зависимости, ни какая-либо комбинация из двух столбцов не определяют однозначно строки таблицы. Следовательно, единственным возможным ключом является тройка (проект, исполнитель, язык).

Вторая зависимость определяется таблицей:

Студент	Профессор	Университет
Белова	Лавров	МГУ
Белова	Шапиро	МГТУ
Чернова	Лавров	МГТУ
Чернова	Лавров	МГУ
Халецкий	Уилкинсон	Оксфорд

Студент может посещать более одного университета, профессор может преподавать более, чем в одном университете. В каждом университете студент должен иметь одного научного руководителя.

Легко видеть, что ни один из столбцов не может быть возможным ключом. Рассмотрим различные пары столбцов: (студент, профессор), (студент, университет), (профессор, университет). Отметим, что пары (студент, профессор) и (профессор, университет) не могут быть возможными ключами: значения (Чернова, Лавров) и (Лавров, МГУ) встречаются дважды. Что касается пары (студент, университет), то она может быть возможным ключом. Тройка (студент, профессор, университет) не является возможным ключом, так как это не минимальный набор атрибутов.

Ограничения

Ограничения - это функциональные зависимости между сущностями объектной модели. Под термином сущности подразумеваются объекты, классы, атрибуты, связи и зависимости. Ограничение сокращает количество возможных значений, которые может принимать некоторая сущность.

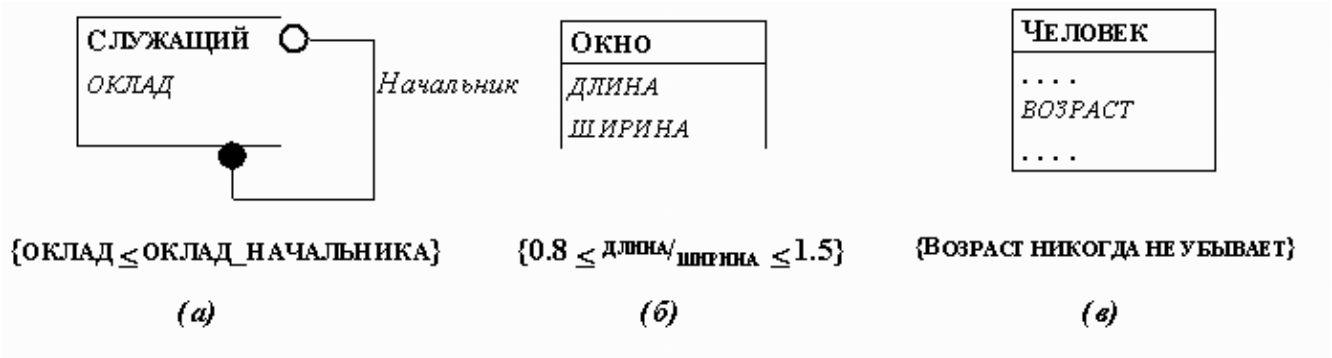


Рис. 2.29. Ограничения на объекты

На рисунке 2.29(а - в) представлены ограничения, накладываемые на объекты:

- (а) зарплата служащего не может превышать зарплаты его начальника (ограничение на значения одного атрибута разных объектов);

- (б) никакое окно (на экране дисплея) не может иметь отношение длины к ширине, не лежащее в интервале от 0.8 до 1.5 (ограничение на значения разных атрибутов одного объекта);

- (в) возраст человека не может убывать (ограничение на изменение значения атрибута во времени).

Ограничения указываются в фигурных скобках под изображением соответствующего класса на объектной диаграмме (они относятся ко всем объектам этого класса). Обычно ограничения могут быть выражены в виде логических функций (предикатов), которые и представляют их в программе. Ограничения дают один из критериев качества объектной модели: "хорошая" объектная модель обычно содержит много ограничений.

Ограничения на зависимости сокращают количество объектов, связанных с данным объектом (соответствующие обозначения уже рассматривались нами ранее). На рисунке 2.30 показан пример другого рода ограничений: показано, что множество объектов на одном из концов зависимости является упорядоченным.

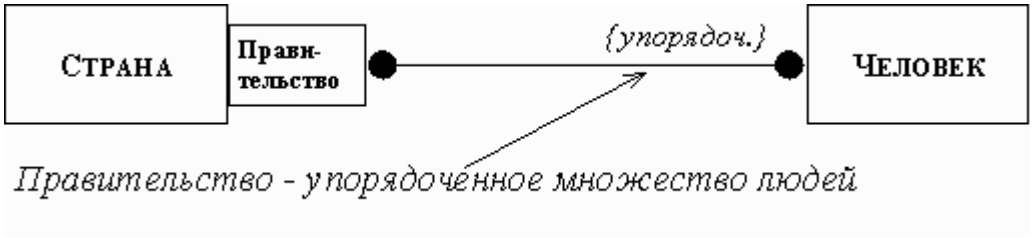


Рис. 2.30. Ограничения на связи

На рисунке 2.31 приведен пример общих ограничений на зависимости: указано, что зависимость является_председателем между объектами человек и комиссия есть подмножество зависимости является_членом. Общие ограничения выражаются, как правило, с помощью уравнений. На диаграммах они обозначаются пунктирными стрелками с комментариями в фигурных скобках.

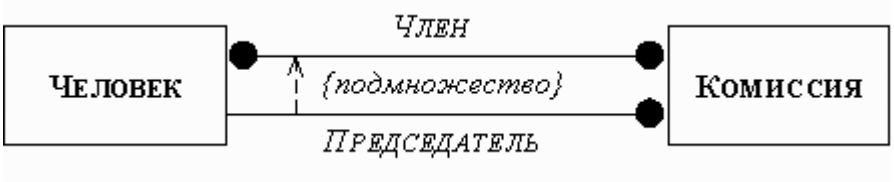


Рис. 2.31. Общее ограничение между зависимостями

Производные объекты, связи и атрибуты

Производный объект определяется как функция от одного или нескольких объектов. Он полностью определяется этими объектами. Следовательно, производный объект избыточен, но он может быть включен в объектную модель для облегчения ее понимания. Аналогичным образом можно определить производные связи и производные атрибуты. Все производные сущности помечаются косой чертой ("/").

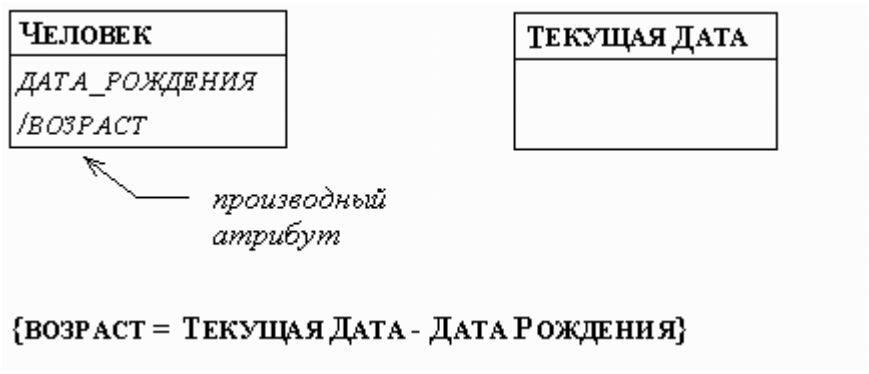


Рис. 2.32. Производный атрибут

На рисунке 2.32 показан производный атрибут возраст класса человек, возраст определяется как разность значений объекта текущая_дата (соответствующий класс также показан на рисунке) и атрибута дата_рождения объекта класса человек.

На рисунке 2.33 показано, что объект класса машина состоит из нескольких объектов класса узел, каждый из которых может состоять из объектов класса деталь. Каждый узел характеризуется смещением в системе координат, связанной с машиной, а каждая деталь характеризуется смещением в системе координат, связанной с узлом.

Для удобства можно для каждого объекта деталь определить систему координат, в которой координаты вычисляются с помощью координат объекта машина, смещения объекта узел и смещения объекта деталь. Эта система координат может быть представлена в производном классе смещение, который связан с каждым объектом деталь производной зависимостью чистое_смещение. Поскольку в реальном мире наблюдается большая избыточность, в объектные модели удобно вводить большое количество производных сущностей.

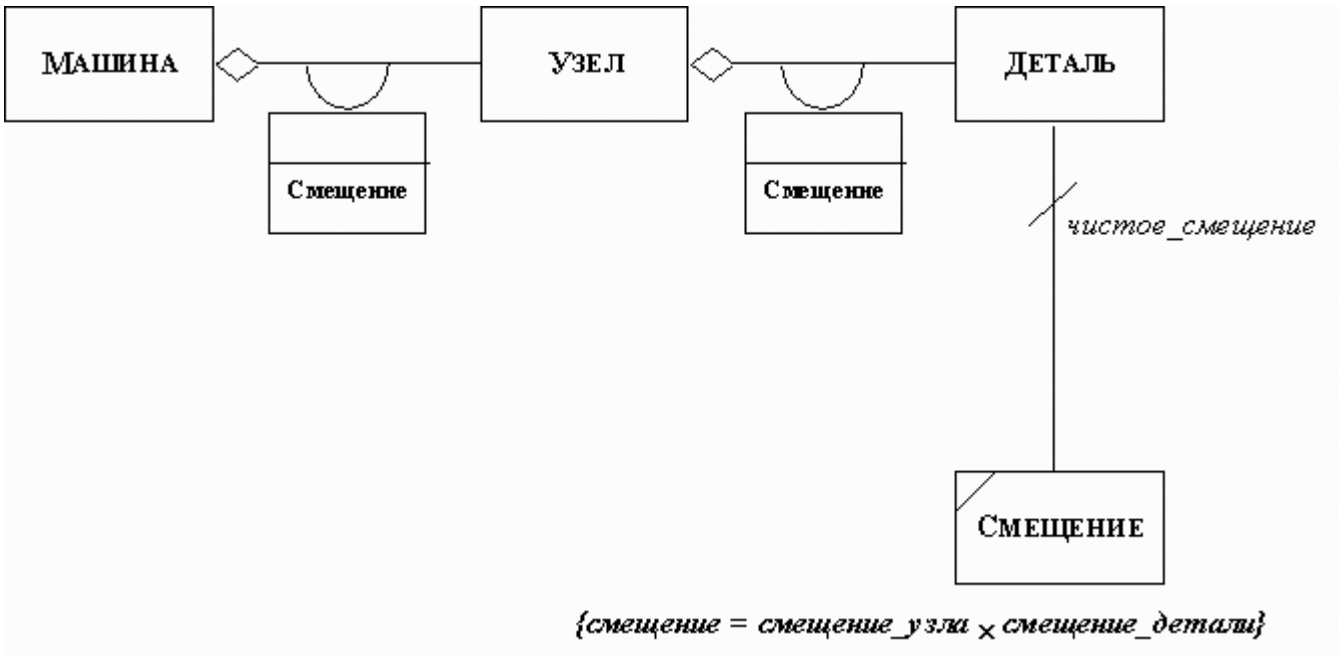


Рис. 2.33. Производный объект и производная зависимость

Гомоморфизмы

Гомоморфизм - это отображение между двумя зависимостями (как показано на рисунке 2.34). Например, деталь, описанная в каталоге деталей автомобиля, может содержать в себе другие детали (составная деталь), также описанные в этом каталоге. Каждая позиция каталога снабжена номером модели; каждая деталь характеризуется своим серийным номером, причем каждая составная деталь содержит в себе соответствующие детали, также имеющие свои серийные номера.

На рисунке 2.34 показаны классы, соответствующие физическим деталям (класс деталь) и их описаниям в каталоге (класс модель_детали), а также зависимости между такими классами (поскольку речь идет о сложных деталях, эти зависимости являются агрегациями зависимости содержит). На рисунке показано отображение диаграмм объектных моделей, соответствующих физическим деталям, в диаграмму объектной модели, соответствующей каталогу. При этом следует иметь в виду, что поскольку разным физическим деталям одного вида (номенклатуры) соответствует одно описание в каталоге, рассматриваемое отображение как бы "склеивает" все диаграммы объектных моделей, соответствующих физическим деталям, в одну диаграмму объектной модели, соответствующей каталогу. Описанное отображение есть один из наиболее типичных примеров гомоморфизма.

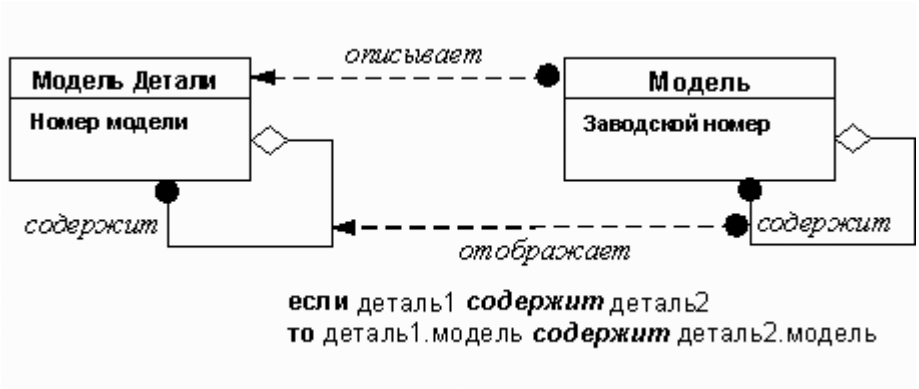


Рис. 2.34. Пример гомоморфизма

Общий случай гомоморфизма показан на рисунке 2.35. Гомоморфизм отображает связи, определяемые зависимостью u, в связи, определяемые зависимостью t, являясь отображением типа "много-в-один". Гоморфизмы обычно используются при изучении связей между метаобъектами и порождаемыми ими объектами (описание детали в каталоге может рассматриваться как метадеталь).

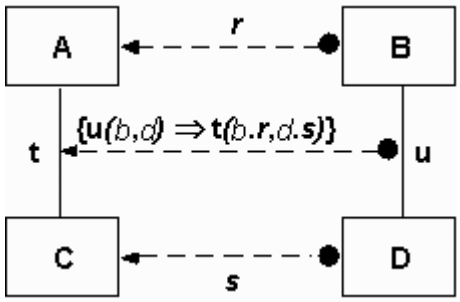


Рис. 2.35. Общий случай гомоморфизма