Лекция 3. Объекты и классы

Объектно-ориентированный подход предлагает описывать системы в виде взаимодействия объектов.

Объект — это сущность, способная сохранять своё состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния.

Объект обладает индивидуальностью, состоянием и поведением. Структура и поведение подобных объектов определены в их общем классе. Термины «экземпляр класса» и «объект» взаимозаменяемы. На рисунке 3.1 приведён пример объекта по имени *Стул*, имеющего определённый набор свойств и операций.

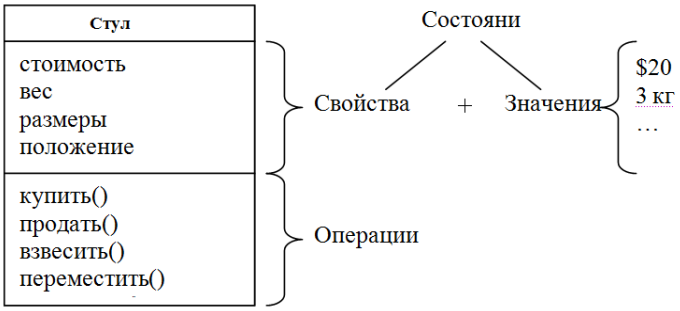

Рис. 3.1. Объект Стул

Индивидуальность — это характеристика объекта, которая отличает его от всех других объектов.

Состояние объекта характеризуется перечнем всех свойств объекта и текущими значениями каждого из этих свойств. Каждый объект характеризуется своим *состоянием*.

Состояние объекта характеризуется текущим значением его *атрибутов*. У счёта есть атрибут — баланс. Он выражается числом — количеством рублей. Операции снятия со счёта и зачисления на счёт изменяют баланс и состояние объекта. Атрибутами могут быть как простейшие величины (числа, логические значения), так и сложные величины, объекты.

Объекты не существуют изолированно друг от друга. Они подвергаются воздействию или сами воздействуют на другие объекты.

Важнейшей характеристикой объекта является описание того, как он может взаимодействовать с окружающим миром. Это описание называется *интерфейсом* объекта.

Объекты взаимодействуют между собой с помощью сообщений. Принимая сообщение, объект выполняет соответствующее действие. Эти действия обычно называются *методами*.

Пример. У объекта «счёт №66579801» имеются методы «снять деньги со счёта», «положить деньги на счёт». Эти два метода составляют интерфейс объекта. У объекта «клиент Иванов» имеется метод «сообщить свой код». У объекта «банкомат на Тверской» есть методы «начать работу», «принять деньги», «сообщить остаток денег на счёте», «выдать деньги».

У объекта «счёт» атрибут «баланс» на является непосредственно частью интерфейса. Другие объекты могут обратиться к этому атрибуту только опосредованно, с помощью метода «сообщить остаток денег на счёте». Они не могут умножить этот атрибут на два или разделить пополам.

Таким образом, наряду с методами и атрибутами, входящими в интерфейс, и доступными другим объектам, у объекта могут быть методы или атрибуты, предназначенные для «внутреннего употребления», к которым может обращаться только сам объект. То есть объект известен другим объектам только по своему интерфейсу. Внутренняя структура его скрыта. Важным следствием является возможность изменения внутренней структуры объекта независимо от других взаимодействующих с ним объектов. С точки зрения программирования легче и безопаснее производить модификации системы.

Поведение характеризует то, как объект воздействует на другие объекты (или подвергается воздействию) в терминах изменений его состояния и передачи сообщений. Поведение объекта является функцией как его состояния, так и выполняемых им операций (купить, продать, взвесить, переместить, покрасить). Говорят, что состояние объекта представляет суммарный результат его поведения.

Операция обозначает обслуживание, которое объект предлагает своим клиентам. Возможны пять видов операций клиента над объектом:

1) модификатор (изменяет состояние объекта);

2) селектор (даёт доступ к состоянию, но не изменяет его);
3) итератор (доступ к содержанию объекта по частям, в строго определённом порядке);
4) конструктор (создает объект и инициализирует его состояние);
5) деструктор (разрушает объект и освобождает занимаемую им память).

В чистых объектно-ориентированных языках программирования операции объявляться только как методы — элементы классов, экземплярами которых являются объекты. Гибридные языки (C++, Ada 95) позволяют писать операции как свободные подпрограммы (вне классов).

В общем случае все методы и свободные подпрограммы, ассоциированные с конкретным объектом, образуют его *протокол*. Таким образом, протокол определяет оболочку допустимого поведения объекта и поэтому заключает в себе цельное (статическое и динамическое) представление объекта.

Большой протокол полезно разделять на логические группировки поведения. Эти группировки, разделяющие пространство поведения объекта, обозначают *роли*, которые может играть объект. Принцип выделения ролей иллюстрирует рисунок. С точки зрения внешней среды важное значение имеет такое понятие, как обязанности объекта. *Обязанности* означают обязательства объекта обеспечить определённое поведение. Обязанностями объекта являются все виды обслуживания, которые он предлагает клиентам. В мире объект играет определённые роли, выполняя свои обязанности.

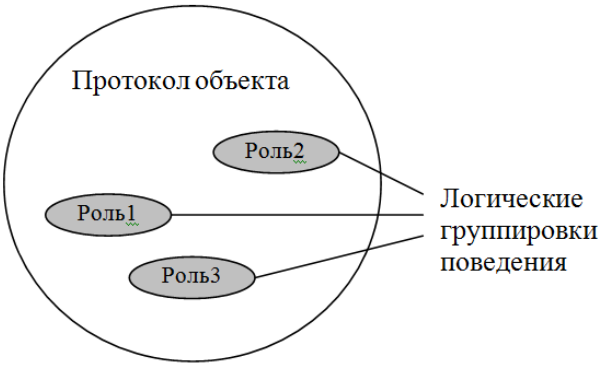

Рис. 3.2. Протокол объекта

Наличие у объекта внутреннего состояния означает, что порядок выполнения им операций очень важен. Иначе говоря, объект может представляться как независимый автомат. По аналогии с автоматами можно выделять активные и пассивные объекты.

Активный объект имеет собственный канал (поток) управления, пассивный — нет. Активный объект автономен, он может проявлять своё поведение без воздействия со стороны других объектов. Пассивный объект, наоборот, может изменять своё состояние только под воздействием других объектов.

Виды отношений между объектами

В поле зрения разработчика программного обеспечения находятся не объекты-одиночки, а взаимодействующие объекты, ведь именно взаимодействие объектов реализует поведение системы. У Г. Буча есть отличная цитата из Галла: «Самолёт — это набор элементов, каждый из которых по своей природе стремится упасть на землю, но ценой совместных непрерывных усилий преодолевает эту тенденцию». Отношения между парой объектов основываются на взаимной информации о разрешённых операциях и ожидаемом поведении. Особо интересны два вида отношений между объектами: связи и агрегация.

Связь — это физическое или понятийное соединение между объектами. Объект сотрудничает с другими объектами через соединяющие их связи. Связь обозначает соединение, с помощью которого:

- объект-клиент вызывает операции объекта-поставщика;
- один объект перемещает данные к другому объекту.

Можно сказать, что связи являются рельсами между станциями-объектами, по которым ездят «трамвайчики сообщений».

Связи между объектами (рис. 3.3) показаны с помощью соединительных линий. Связи представляют возможные пути для передачи сообщений. Сами сообщения показаны стрелками, отмечающими их направления, и помечены именами вызываемых операций.

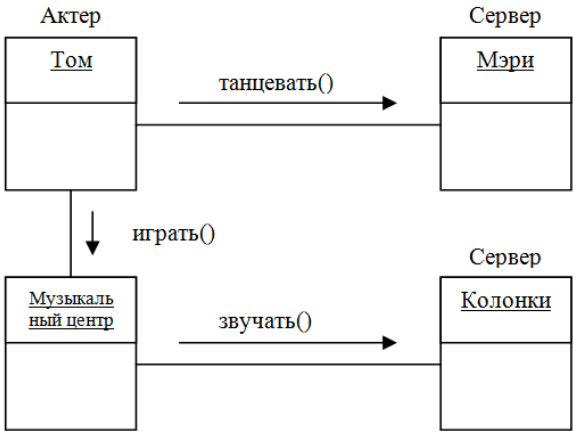

Рис. 3.3. Связи между объектами

Как участник связи объект может играть одну из трех ролей:

- актёр — объект, который может воздействовать на другие объекты, но никогда не подвержен воздействию других объектов;
- сервер — объект, который никогда не воздействует на другие объекты, он только используется другими объектами;
- агент — объект, который может как воздействовать на другие объекты, так и использоваться ими. Агент создаётся для выполнения работы от имени актера или другого агента.

На рисунке 3.3 *Том* — это актёр, *Мэри* и *Колонки* — серверы, *Музыкальный центр* — агент.

Рассмотрим два объекта, A и B, между которыми имеется связь. Для того чтобы объект A мог послать сообщение в объект B, надо, чтобы B был *виден* для A.

Различают четыре *формы видимости* между объектами.

1. Объект-поставщик (сервер) глобален для клиента.
2. Объект-поставщик (сервер) является параметром операции клиента.
3. Объект-поставщик (сервер) является частью объекта-клиента.
4. Объект-поставщик (сервер) является локально объявленным объектом в операции клиента.

На этапе анализа вопросы видимости обычно опускают. На этапах проектирования и реализации вопросы видимости по связям обязательно должны рассматриваться.

Связи обозначают равноправные (клиент—серверные) отношения между объектами. *Агрегация* обозначает отношения объектов в иерархии «целое—часть». Агрегация обеспечивает возможность перемещения от целого (агрегата) к его частям (свойствам).

Агрегация может обозначать, а может и не обозначать физическое включение части в целое. На рисунке 3.4 приведён пример физического включения (композиции) частей (*Двигатель*, *Сиденья*, *Колеса*) в агрегат *Автомобиль*. В этом случае говорят, что части включены в агрегат по величине.

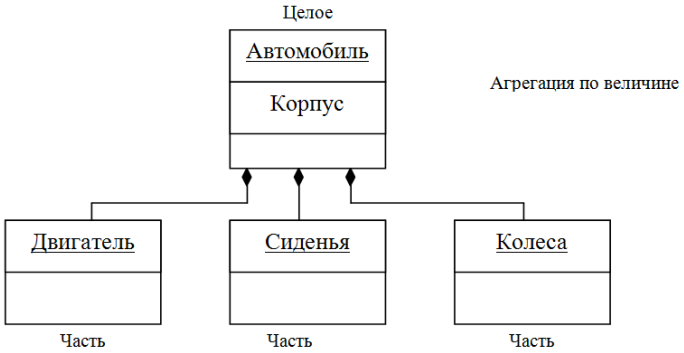

Рис. 3.4. Агрегация по величине

На рисунке 3.5 приведён пример нефизического включения частей (*Студент*, *Преподаватель*) в агрегат *Вуз*. Очевидно, что *Студент* и *Преподаватель* являются элементами *Вуза*, но они не входят в него физически. В этом случае говорят, что части включены в агрегат по ссылке.

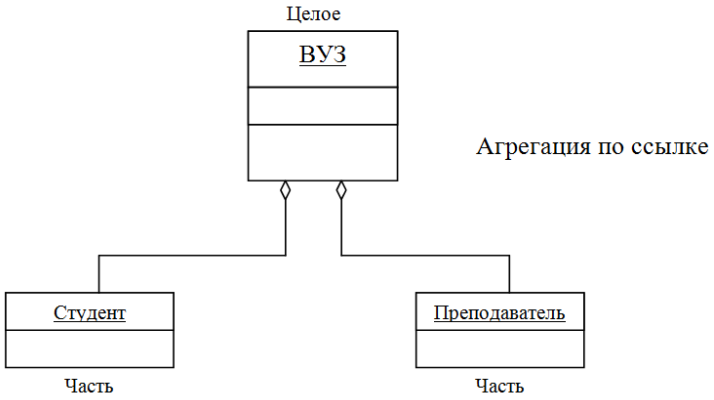

Рис. 3.5. Агрегация по ссылке

При выборе вида отношения между объектами должны учитываться следующие факторы:

- связи обеспечивают низкое сцепление между объектами;
- агрегация инкапсулирует части как секреты целого.

Общая характеристика классов

В объектно-ориентированной системе обычно функционирует множество объектов. Однотипные объекты объединяются в *классы*.

Класс — это описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл). Любой объект является просто экземпляром класса.

Класс — это структурный тип данных, который включает описание полей данных, а также процедур и функций, работающих с этими полями данных. Применительно к классам такие процедуры и функции получили название *методов*.

Все объекты одного и того же класса обладают одинаковым интерфейсом и реализуют этот интерфейс одним и тем же способом. Два объекта одного класса могут отличаться только текущим состоянием.

Пример. У всех объектов-счетов, принадлежащих классу «Счёт», имеется номер и баланс, все они реагируют на сообщения «проверить наличие денег» и «снять сумму со счёта».

Индивидуальные объекты называются *экземплярами* класса, а класс — это *шаблон*, по которому строятся объекты.

Реализация объединения данных с определёнными видами их обработки делает классы пригодными для описания состояния и поведения моделей реальных объектов. Совокупность полей определяется множеством аспектов состояния объекта с точки зрения решаемой задачи, а совокупность методов — множеством аспектов поведения объекта (рис. 3.6).

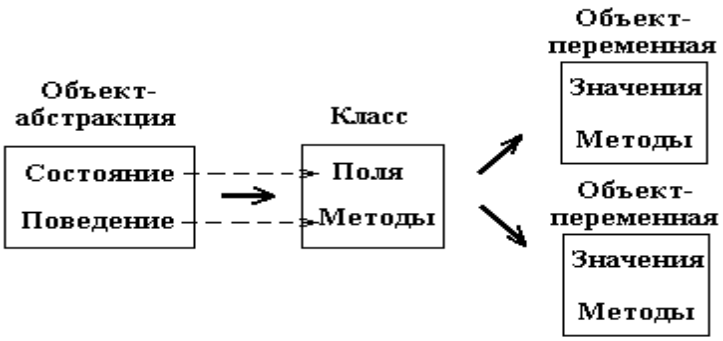

Рис. 3.6. Соответствие объекта-абстракции классу и объектам-переменным

В программах используются переменные типа класса. Такие переменные принято называть просто объектами.

Пример. Банковская система состоит из экземпляров 3-х классов: класса счетов, класса банкоматов и класса клиентов (рис. 3.7). Графические изображая классы и отношения между ними, будем придерживаться обозначений, принятых в *унифицированном языке моделирования UML* (*Unified Modelling Language*).

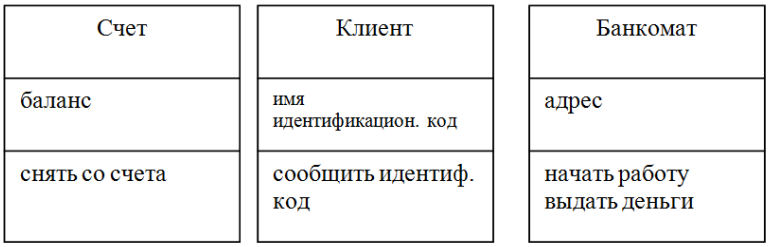

Рис. 3.7. Классы банковской системы

Различают внутреннее представление класса (*реализацию*) и внешнее представление класса (*интерфейс*). На рисунке 3.8 представлена общая структура класса.

Рис. 3.8. Общая структура класса

Интерфейс объявляет возможности (услуги) класса, но скрывает его структуру и поведение. Иными словами, интерфейс демонстрирует внешнему миру абстракцию класса, его внешний облик. Интерфейс в основном состоит из объявлений всех операций, применимых к экземплярам класса. Он может также включать объявления типов, переменных, констант и исключений, необходимых для полноты данной абстракции.

Интерфейс может быть разделен на три части:

1) *публичную* (*public*), объявления которой доступны всем клиентам;
2) *защищённую* (*protected*), объявления которой доступны только самому классу, его подклассам и друзьям;
3) *приватную* (*private*), объявления которой доступны только самому классу и его друзьям.

Другом класса называют класс, который имеет доступ ко всем частям этого класса (публичной, защищённой и приватной). Иными словами, от друга у класса нет секретов. (Другом класса может быть и свободная подпрограмма).

Наличие интерфейса обеспечивает уменьшение возможности «разрушения» (несанкционированного изменения значений полей) объекта извне. Сокрытие особенностей реализации, кроме того, упрощает внесение изменений в реализацию класса, как в процессе отладки, так и при модификации программы.

Реализация класса описывает секреты поведения класса. Она включает реализации всех операций, определенных в интерфейсе класса.

Виды отношений между классами

Классы, подобно объектам, не существуют в изоляции. Напротив, с отдельной проблемной областью связывают ключевые абстракции, отношения между которыми формируют структуру из классов системы.

Для покрытия основных отношений большинство объектно-ориентированных языков программирования поддерживают следующие отношения между классами:

- ассоциация;
- наследование;
- агрегация;
- зависимость.

Ассоциации обеспечивают взаимодействия объектов, принадлежащих разным классам. Они являются клеем, соединяющим воедино все элементы программной системы. Благодаря ассоциациям мы получаем работающую систему. Без ассоциаций система превращается в набор изолированных классов-одиночек.

Наследование — наиболее популярная разновидность отношения *обобщение—специализация*. Альтернативой наследованию считается *делегирование*. При делегировании объекты *делегируют* своё поведение родственным объектам. При этом классы становятся не нужны.

Агрегация обеспечивает отношения *целое—часть*, объявляемые для экземпляров классов.

Зависимость часто представляется в виде частной формы — *использования*, которое фиксирует отношение между клиентом, запрашивающим услугу, и сервером, предоставляющим эту услугу.

Конкретизация выражает другую разновидность отношения *обобщение—специализация*. Применяется в таких языках как Ada 95, C++, Эйфель.

Отношения метаклассов поддерживаются в языках SmallTalk и CLOS. *Метакласс* — это класс классов, понятие, позволяющее обращаться с классами как с объектами.

Реализация определяет отношение, при котором класс-приёмник обеспечивает свою собственную реализацию интерфейса другого класса-источника. Иными словами, здесь речь идёт о наследовании интерфейса. Семантически реализация — это «скрещивание» отношений зависимости и обобщения-специализации.

Ассоциации классов

Ассоциация обозначает семантическое соединение классов.

Пример. В системе обслуживания читателей имеются две ключевые абстракции — *Книга* и *Библиотека*. Класс *Книга* играет роль элемента, хранимого в библиотеке. Класс *Библиотека* играет роль хранилища для книг (рис. 3.9).

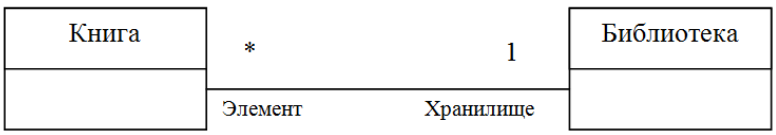

Рис. 3.9. Ассоциация классов

Отношение ассоциации между классами изображено на рисунке. Очевидно, что ассоциация предполагает двухсторонние отношения:

- для данного экземпляра *Книги* выделяется экземпляр *Библиотеки*, обеспечивающий её хранение;
- для данного экземпляра *Библиотеки* выделяются все хранимые *Книги*.

Здесь представлена ассоциация *один-ко-многим*. Каждый экземпляр *Книги* имеет указатель на экземпляр *Библиотеки*. Каждый экземпляр *Библиотеки* имеет набор указателей на несколько экземпляров *Книги*.

Ассоциация обеспечивает только семантическую связь. Она не указывает направление и точную реализацию отношения. Ассоциация пригодна для анализа проблемы, когда нам требуется лишь идентифицировать связи. С помощью создания ассоциаций мы приходим к пониманию участников семантических связей, их ролей, мощности (количества элементов).

Ассоциация один-ко-многим означает, что для каждого экземпляра класса *Библиотека*, есть 0 или более экземпляров класса *Книга*, а для каждого экземпляра класса *Книга* есть один экземпляр *Библиотеки*. Эту множественность обозначает мощность ассоциации. Мощность ассоциации бывает одного из трёх типов:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

Примеры ассоциаций с различными типами мощности приведены на рис. 3.10. Они имеют следующий смысл:

- у европейской жены один муж, а у европейского мужа одна жена;
- у восточной жены один муж, а у восточного мужа сколько угодно жен;
- у заказа один клиент, а у клиента сколько угодно заказов;
- человек может посещать сколько угодно зданий, а в здании может находиться сколько угодно людей.

Рис. 3.10. Мощность ассоциации

Наследование

Важнейшим свойством классов и их принципиальным отличием от абстрактных типов данных является наследование.

Наследование — это отношение, при котором один класс разделяет структуру и поведение, определённые в одном другом (простое наследование) или во многих других (множественное наследование) классах.

Между классами наследование определяет иерархию «является» («is a»), при которой подкласс наследует от одного или нескольких более общих суперклассов. Говорят, что подкласс является специализацией его суперкласса (за счёт дополнения или переопределения существующей структуры или поведения).

Пример. Существуют различные виды банковских счетов. Однако они имеют много общего. Выделив общую часть, создадим класс *Счёт*. Классы *Расчётный счёт* и *Депозит* сохраняют все свойства (как методы, так и атрибуты) класса *Счёт*, дополняя и уточняя его поведение. Говорят, что класс *Депозит* наследует класс *Счёт* (рис. 3.11).

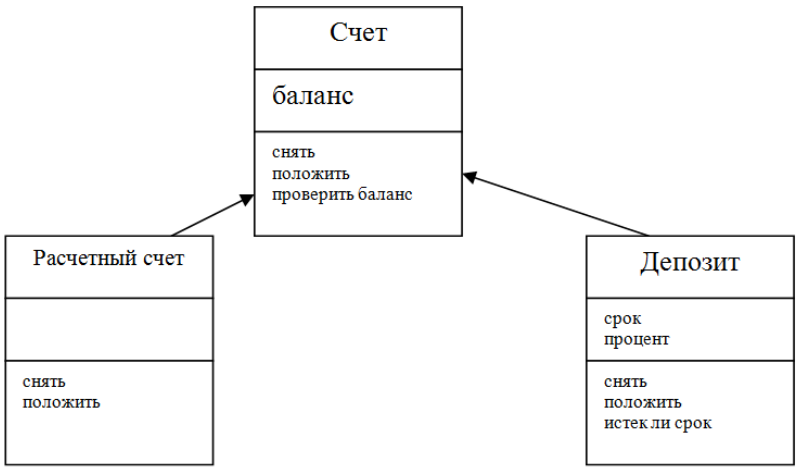

Рис. 3.11. Наследование классов

Класс *Расчётный счёт* использует (наследует) атрибут своего *базового класса* и изменяет (уточняет) два метода: «снять» и «положить». Метод «проверить баланс» наследуется без изменений. Класс *Депозит* не только уточняет эти методы, но и добавляет новые атрибуты и методы.

Таким образом, класс *Расчётный счёт* содержит 1 атрибут и 3 метода, а класс *Депозит* — 3 атрибута и 4 метода.

Уточнение свойств классов может происходить в несколько шагов (рис. 3.12).

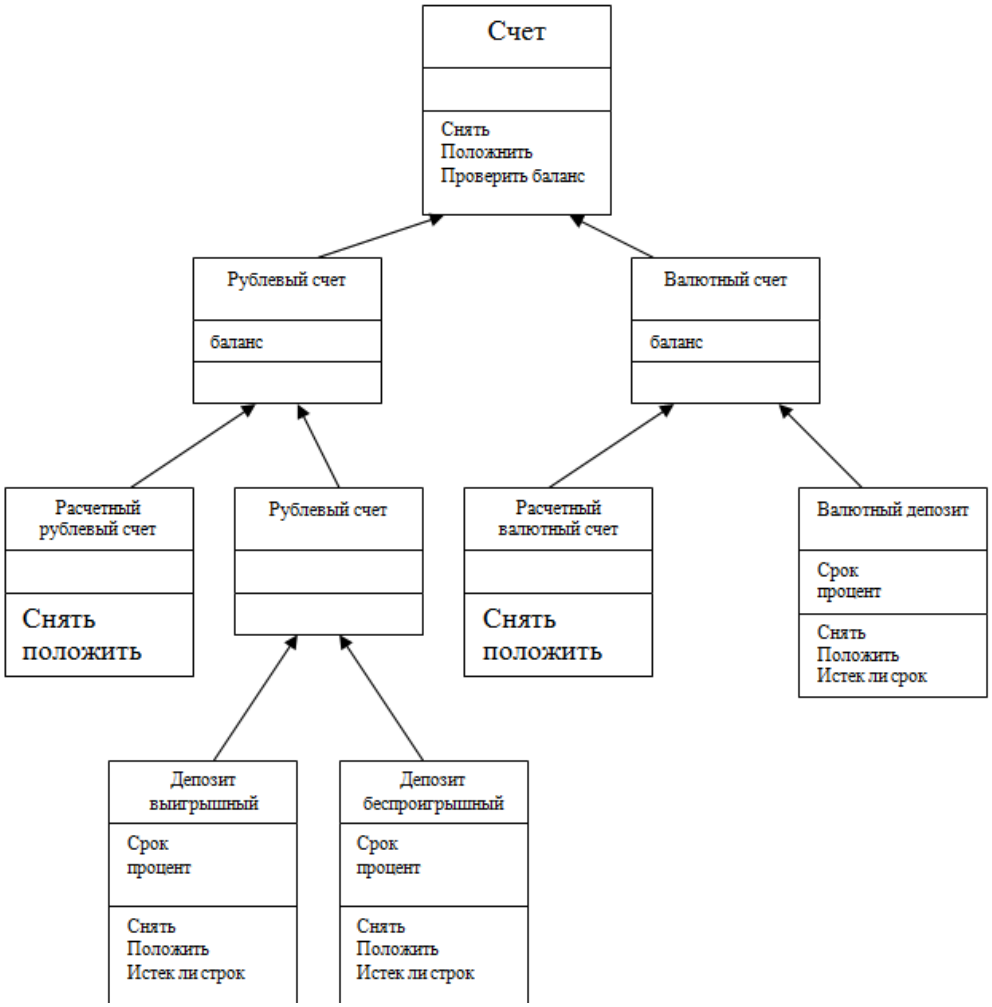

Рис. 3.12. Диаграмма классов банковских счетов (вариант 1)

Построить иерархию классов можно иначе (рис. 3.13).

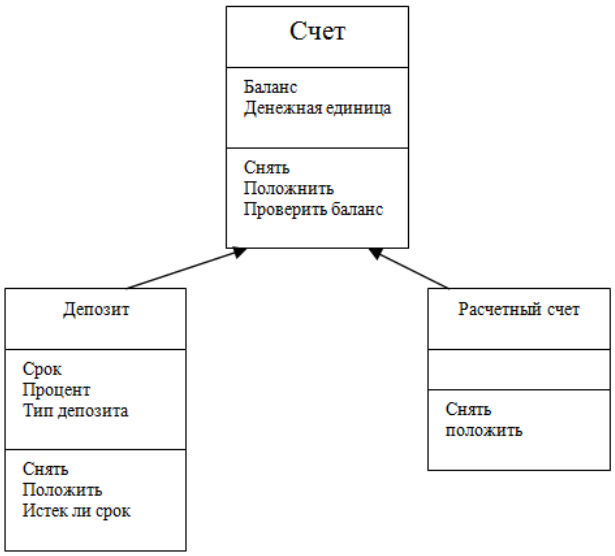

Рис. 3.13. Диаграмма классов банковских счетов (вариант 2)

Отметим, что класс *Счёт* введен для того, чтобы описать общие характеристики всех счетов, а не для того, чтобы создавать объекты этого класса. Классы, для которых не существует экземпляров, называются *абстрактными*. *Конкретными* классами в отличие от них называются классы, экземпляры которых могут существовать в системе. Назначением абстрактных классов является определение общих, наиболее характерных методов и атрибутов наследуемых из них классов. Чаще всего абстрактные классы используются для задания общего интерфейса иерархии конкретных классов, хотя атрибуты, и реализация каких-либо методов может присутствовать в абстрактных классах.

Класс может унаследовать методы и атрибуты одного базового класса – такое наследование называется *одинарное* или *простое*. В случае *множественного наследования* у одного класса имеется несколько базовых.

Множественное наследование позволяет объединять характеристики разных классов в одном.

Пример. При реализации банковской системы объекты, в том числе и счета, должны храниться в базе данных. Система классов, обеспечивающая хранение объектов в базе данных, может состоять из базового класса *Постоянный объект*. Для того чтобы какой-либо конкретный счёт стало возможным хранить в базе данных, он должен быть выведен из класса *Постоянный объект* (рис. 3.14).

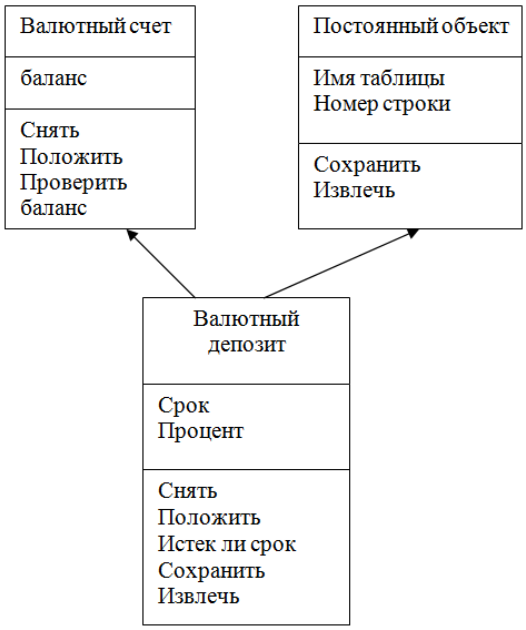

Рис. 3.14. Множественное наследование

Полиморфизм

Полиморфизм означает способность обладать несколькими формами. В объектно-ориентированной разработке несколькими формами обладают сущности, способные во время выполнения присоединяться к объектам разных типов, что контролируется статическими объявлениями.

При создании иерархии классов может обнаружиться, что некоторые свойства объектов, сохраняя название, изменяются по сути. Для реализации таких иерархий в языке программирования должен быть предусмотрен механизм полиморфизма, обеспечивающий возможность задания различных реализаций некоторого единого по названию метода для классов различных уровней иерархии.

Полиморфизм — это возможность с помощью одного имени обозначать операции из различных классов (но относящихся к общему суперклассу). Вызов обслуживания по полиморфному имени приводит к исполнению одной из некоторого набора операций.

Пример. Когда какой-либо объект, например *Банкомат*, посылает сообщение объекту *Счёт* о необходимости снятия денег, конкретный вид счёта не важен. Важно, что объект может правильно обработать сообщение «снять». *Полиморфизмом* называется возможность взаимодействия с объектом, не зная, к какому конкретному классу он относится. Посылая сообщение любому счёту, мы используем полиморфизм счетов.

При использовании полиморфизма используется знание интерфейса объекта, однако поведение конкретного объекта в ответ на полученное сообщение может быть различно в зависимости от конкретного класса этого объекта. Соответственно посылающий объект точно не знает, что произойдет при вызове метода.

Полиморфными объектами или *полиморфными переменными* называются переменные, которым в процессе выполнения программы может быть присвоено значение, тип которого отличается от типа переменной. В языках с жёсткой типизацией такая ситуация может возникнуть:

- при передаче объекта типа класса-потомка в качестве фактического параметра подпрограмме, в которой этот параметр описан, как параметр типа класса-родителя;
- при работе с указателями, когда указателю на объект класса-родителя присваивается адрес объекта класса-потомка.

Тип полиморфного объекта становится известным только на этапе выполнения программы. Соответственно, при вызове полиморфного метода для такого объекта, нужный аспект также должен определяться на этапе выполнения.

Для этого в языке должен быть реализован механизм *позднего связывания*, позволяющий определять тип объекта и, соответственно, аспект полиморфного метода, к которому идет обращение в программе, на этапе её выполнения.

С помощью механизма позднего связывания реализуется оперативная перестройка программы в соответствии с типами используемых объектов.

Агрегация

Отношения агрегации между классами аналогичны отношениям агрегации между объектами.

Агрегация может быть определена как *включение по значению*. Это пример физического включения, означающий, что объект не существует независимо от включающего его экземпляра. Время жизни этих двух объектов неразрывно связано.

Возможен косвенный тип агрегации — *включение по ссылке*. При этом сцепление объектов уменьшено. Экземпляры каждого класса создаются и уничтожаются независимо.

Отметим, что, в сущности, свойства (атрибуты) класса находятся в отношении композиции между всем классом и его элементами-свойствами. Тем не менее в общем случае свойства должны иметь примитивные значения (числа, строки, даты), а не ссылаться на другие классы, так как в «атрибутной» нотации не видны другие отношения классов-частей. Кроме того, свойства классов не могут находиться в совместном использовании несколькими классами.

Зависимость

Зависимость — это отношение, которое показывает, что изменение в одном классе (независимом) может влиять на другой класс (зависимый), который использует его. Графически зависимость изображается как пунктирная стрелка, направленная на класс, от которого зависит. С помощью зависимости уточняют, какая абстракция является клиентом, а какая — поставщиком определённой услуги. Пунктирная стрелка зависимости направлена от клиента к поставщику.

Наиболее часто зависимости показывают, что один класс использует другой класс как аргумент в сигнатуре своей операции.