«Диаграммы классов»

Статические модели обеспечивают представление структуры систем в терминах базовых строительных блоков и отношений между ними. «Статичность» этих моделей состоит в том, что здесь не показывается динамика изменений системы во времени. Вместе с тем следует понимать, что эти модели несут в себе не только структурные описания, но и описания операций, реализующих заданное поведение системы. Основным средством для представления статических моделей являются диаграммы классов. Вершины диаграмм классов нагружены классами, а дуги (ребра) — отношениями между ними. Диаграммы используются:

- в ходе анализа — для указания ролей и обязанностей сущностей, которые обеспечивают поведение системы;
- в ходе проектирования — для фиксации структуры классов, которые формируют системную архитектуру.

Вершины в диаграммах классов

Итак, вершина в диаграмме классов — класс. Обозначение класса показано на рис. 1.

Рис. 1. Обозначение класса

Имя класса указывается всегда, свойства и операции — выборочно. Предусмотрено задание области действия свойства (операции). Если свойство (операция) подчеркивается, его областью действия является класс, в противном случае областью Действия является экземпляр (рис. 2).

Что это значит? Если областью действия свойства является класс, то все его экземпляры (объекты) используют общее значение этого свойства, в противном случае у каждого экземпляра свое значение свойства.

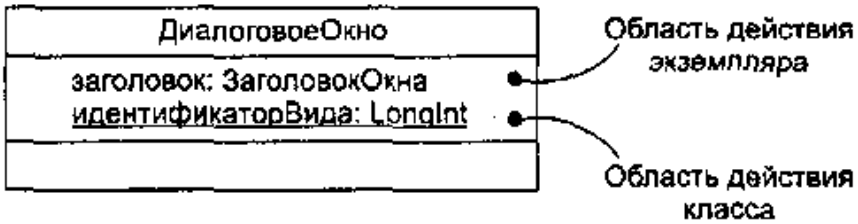

Рис. 2. Свойства уровней класса и экземпляра

Отношения в диаграммах классов

Отношения, используемые в диаграммах классов, показаны на рис. 5.

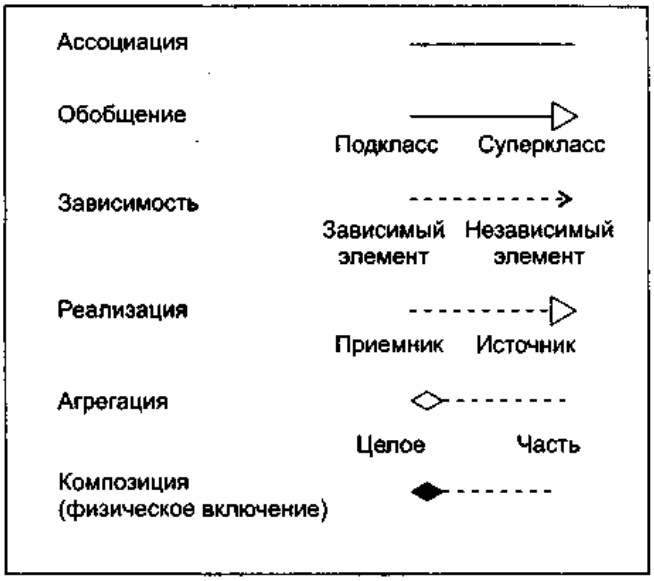

Рис. 5. Отношения в диаграммах классов

Ассоциации отображают структурные отношения между экземплярами классов, то есть соединения между объектами. Каждая ассоциация может иметь метку — *имя,* которое описывает природу отношения. Как показано на рис. 6, имени можно придать направление — достаточно добавить треугольник направления, который указывает направление, заданное для чтения имени.

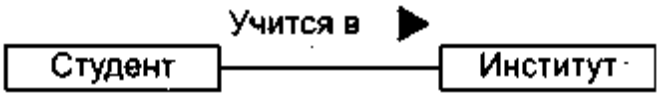

Рис. 6. Имена ассоциаций

Когда класс участвует в ассоциации, он играет в этом отношении определенную роль. Как показано на рис. 11.7, *роль* определяет, каким представляется класс на одном конце ассоциации для класса на противоположном конце ассоциации.

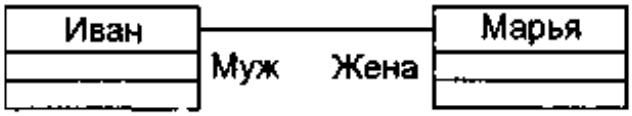

Рис. 7. Роли

Один и тот же класс в разных ассоциациях может играть разные роли. Часто важно знать, как много объектов может соединяться через экземпляр ассоциации. Это количество называется ложностью роли в ассоциации, записывается в виде выражения, задающего диапазон величин или одну величину (рис. 8).

Запись мощности на одном конце ассоциации определяет количество объектов, соединяемых с каждым объектом на противоположном конце ассоциации. Например, можно задать следующие варианты мощности:

- 5 — точно пять;
- * — неограниченное количество;
- 0..* — ноль или более;
- 1..* — один или более;
- 3..7 — определенный диапазон;
- 1..3, 7 — определенный диапазон или число.

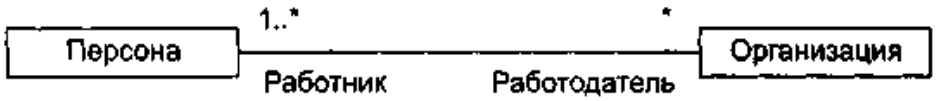

Рис. 8. Мощность

Достаточно часто возникает следующая проблема — как для объекта на одном конце ассоциации выделить набор объектов на противоположном конце? Например, рассмотрим взаимодействие между банком и клиентом — вкладчиком. Как показано на рис. 9, мы устанавливаем ассоциацию между классом Банк и классом Клиент. В контексте Банка мы имеем НомерСчета, который позволяет идентифицировать конкретного Клиента. В этом смысле НомерСчета является атрибутом ассоциации. Он не является характеристикой Клиента, так как Клиенту не обязательно знать служебные параметры его счета. Теперь для данного экземпляра Банка и данного значения НомераСчета можно выявить ноль или один экземпляр Клиента. В UML для решения этой проблемы вводится *квалификатор* — атрибут ассоциации, чьи значения выделяют набор объектов, связанных с объектом через ассоциацию. Квалификатор изображается маленьким прямоугольником, присоединенным к концу ассоциации. В прямоугольник вписывается свойство — атрибут ассоциации.

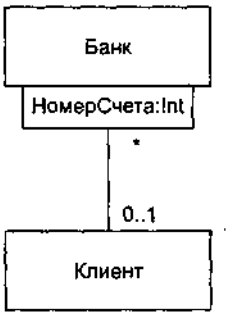

Рис. 9. Квалификация

Кроме того, роли в ассоциациях могут иметь пометки *видимости.* Например, на рис. 10 показаны ассоциации между Начальником и Женщиной, а также между Женщиной и Загадкой. Для данного экземпляра Начальника можно определить соответствующие экземпляры Женщины. С другой стороны, Загадка приватна для Женщины, поэтому она недоступна извне. Как показано на рисунке, из объекта Начальника можно перемещаться к экземплярам Женщины (и наоборот), но нельзя видеть экземпляры Загадки для объектов Женщины.

Рис. 10. Видимость

На конце ассоциации можно задать три уровня видимости, добавляя символ видимости к имени роли:

- по умолчанию для роли задается публичная видимость;
- приватная видимость указывает, что объекты на данном конце недоступны любым объектам вне ассоциации;
- защищенная видимость (protected) указывает, что объекты на данном конце недоступны любым объектам вне ассоциации, за исключением потомков того класса, который указан на противоположном конце ассоциации.

Зависимость является отношением использования между клиентом (зависимым элементом) и поставщиком (независимым элементом). Обычно операции клиента:

- вызывают операции поставщика;
- имеют сигнатуры, в которых возвращаемое значение или аргументы принадлежат классу поставщика.

Например, на рис. 12 показана зависимость класса Заказ от класса Книга, так как Книга используется в операциях проверкаДоступности, добавить и удалить класса Заказ.

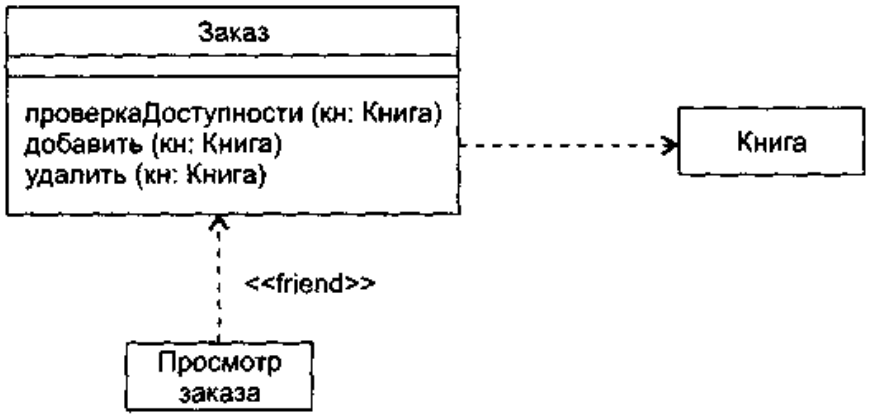

Рис. 12. Отношения зависимости

На этом рисунке изображена еще одна зависимость, которая показывает, что класс Просмотр Заказа использует класс Заказ. Причем Заказ ничего не знает о Просмотре Заказа. Данная зависимость помечена стереотипом «friend», который расширяет простую зависимость, определенную в языке. Отметим, что отношение зависимости очень разнообразно — в настоящее время язык предусматривает 17 разновидностей зависимости, различаемых по стереотипам.

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, которое не является отношением ассоциации, обобщения или реализации. Оно касается только самих элементов модели и не требует множества отдельных примеров для пояснения своего смысла. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("—>" или "<—"). На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику (рис. 13). На данном рисунке изображены два класса: Класс_А и Кяасс_Б, при этом Класс_Б является источником некоторой зависимости, а Класс_А — клиентом этой зависимости.

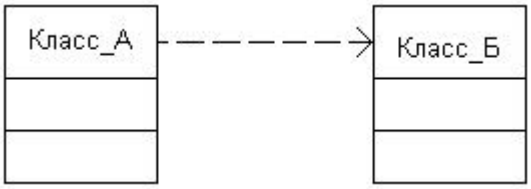

Рис. 13. Графическое изображение отношения зависимости на диаграмме классов

В качестве класса-клиента и класса-источника зависимости могут выступать целые множества элементов модели. В этом случае одна линия со стрелкой, выходящая от источника зависимости, расщепляется в некоторой точке на несколько отдельных линий, каждая из которых имеет отдельную стрелку для класса-клиента. Например, если функционирование Класса_С зависит от особенностей реализации Класса_А и Класса_/>, то данная зависимость может быть изображена следующим образом (рис. 14).

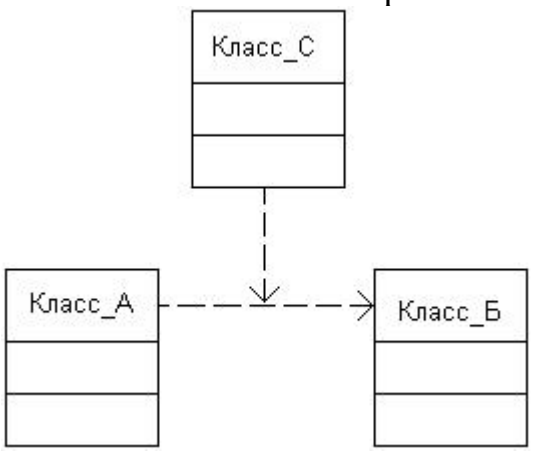

Рис. 14. Графическое представление зависимости между классом-клиентом (Класс_С) и классами-источниками (Класс_Л и Класс_Б)

Стрелка может помечаться необязательным, но стандартным ключевым словом в кавычках и необязательным индивидуальным именем. Для отношения зависимости предопределены ключевые слова, которые обозначают некоторые специальные виды зависимостей. Эти ключевые слова (стереотипы) записываются в кавычках рядом со стрелкой, которая соответствует данной зависимости. Примеры стереотипов для отношения зависимости представлены ниже:

- "access" — служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;
- "bind" — класс-клиент может использовать некоторый шаблон для своей последующей параметризации;
- "derive" — атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;
- "import" — открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем;

- "refine" — указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.

Обобщение — отношение между общим предметом (суперклассом) и специализированной разновидностью этого предмета (подклассом). Подкласс может иметь одного родителя (один суперкласс) или несколько родителей (несколько суперклассов). Во втором случае говорят о множественном наследовании.

Как показано на рис. 15, подкласс Летающий шкаф является наследником суперклассов Летающий предмет и Хранилище вещей. Этому подклассу достаются в наследство все свойства и операции двух классов-родителей.

Множественное наследование достаточно сложно и коварно, имеет много «подводных камней». Например, подкласс Яблочный_Пирог не следует производить от суперклассов Пирог и Яблоко. Это типичное неправильное использование множественного наследования: потомок наследует все свойства от его родителя, хотя обычно не все свойства применимы к потомку. Очевидно, что Яблочный_Пирог является Пирогом, но не является Яблоком, так как пироги не растут на деревьях.

Рис. 15. Множественное наследование

Еще более сложные проблемы возникают при наследовании от двух классов, имеющих общего родителя. Говорят, что в результате образуется ромбовидная решетка наследования (рис. 16).

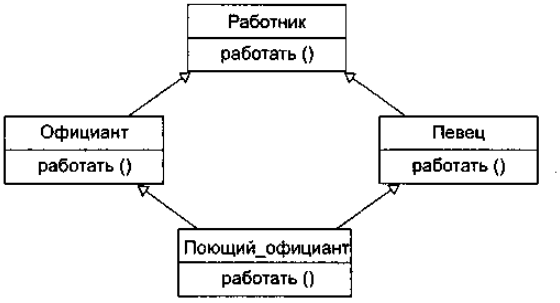

Рис. 16 Ромбовидная решетка наследования

Полагаем, что в подклассах Официант и Певец операция работать суперкласса Работник переопределена в соответствии с обязанностью

подкласса (работа официанта состоит в обслуживании едой, а певца — в пении). Возникает вопрос — какую версию операции работать унаследует Поющий_официант? А что делать со свойствами, доставшимися в наследство от родителей и общего прародителя? Хотим ли мы иметь несколько копий свойства или только одну?

Все эти проблемы увеличивают сложность реализации, приводят к введению многочисленных правил для обработки особых случаев.

Реализация — семантическое отношение между классами, в котором класс-приемник выполняет реализацию операций интерфейса класса-источника. Например, на рис. 17 показано, что класс Каталог должен реализовать интерфейс Обработчик каталога, то есть Обработчик каталога рассматривается как источник, а Каталог — как приемник.

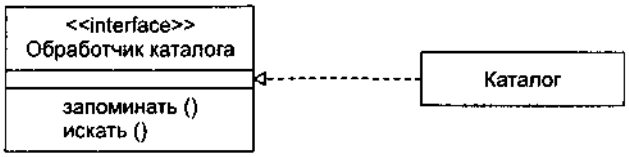

Рис. 17. Реализация интерфейса

Интерфейс Обработчик каталога позволяет клиентам взаимодействовать с объектами класса Каталог без знания той дисциплины доступа, которая здесь реализована (LIFO — последний вошел, первый вышел; FIFO — первый вошел, первый вышел и т. д.).

Отношение обобщения является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 18). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие — на более специальный класс (класс-потомок или подкласс).

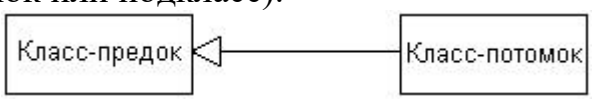

Рис. 18. Графическое изображение отношения обобщения в языке UML

Как правило, на диаграмме может указываться несколько линий для одного отношения обобщения, что отражает его таксономический характер.

В этом случае более общий класс разбивается на подклассы одним отношением Обобщения.

Например, класс Геометрическая_фигура_на_плоскости (курсив обозначает абстрактный класс) может выступать в качестве суперкласса для подклассов, соответствующих конкретным геометрическим фигурам, таким как,Прямоугольник, Окружность, Эллипс и др. Данный факт может быть представлен графически в форме диаграммы классов следующего вида (рис. 19).

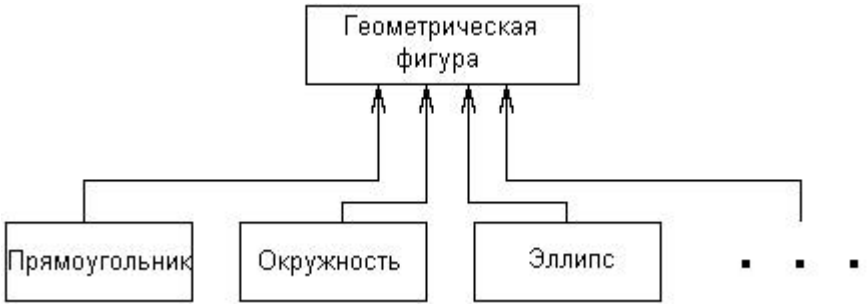

Рис.19. Пример графического изображения отношения обобщения классов

С целью упрощения обозначений на диаграмме классов совокупность линий, обозначающих одно и то же отношение обобщения, может быть объединена в одну линию. В этом случае данные отдельные линии изображаются сходящимися к единственной .стрелке, имеющей с ними общую точку пересечения (рис. 20).

Рис. 20. Вариант графического изображения отношения обобщения классов для случая объединения отдельных линий

Это обозначение по форме соответствует графу специального вида, который рассматривался в главе 2, а именно — иерархическому дереву. В этом случае класс-предок является корнем этого дерева, а классы-потомки — его листьями. Отличие заключается в возможности указания на диаграмме классов потенциальной возможности наличия других классов-потомков, которые не включены в обозначения представленных на диаграмме классов (многоточие вместо прямоугольника).

Рядом со стрелкой обобщения может размещаться строка текста, указывающая на некоторые дополнительные свойства этого отношения. Данный текст будет относиться ко всем линиям обобщения, которые идут к классам-потомкам. Другими словами, отмеченное свойство касается всех подклассов данного отношения. При этом текст следует рассматривать как ограничение, и тогда он записывается в фигурных скобках.

Отношения агрегации и композиции в языке UML считаются разновидностями ассоциации, применяемыми для отображения структурных отношений между «целым» (агрегатом) и его «частями». *Агрегация* показывает отношение по ссылке (в агрегат включены только указатели на части), *композиция* — отношение физического включения (в агрегат включены сами части).

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа "часть-целое". Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой. С точки зрения модели отдельные части системы могут выступать как в виде элементов, так и в виде подсистем, которые, в свою очередь, тоже могут образовывать составные компоненты или подсистемы. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его "частями" (рис. 22).

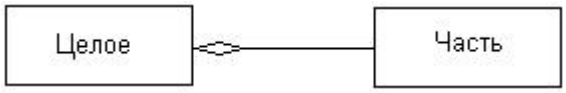

Рис. 22. Графическое изображение отношения агрегации в языке UML

Еще одним примером отношения агрегации может служить известное каждому из читателей деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь. Используя обозначения языка UML, компонентный состав ПК можно представить в виде соответствующей диаграммы классов (рис. 23), которая в данном случае иллюстрирует отношение агрегации.

Рис. 23. Диаграмма классов для иллюстрации отношения агрегации на примере ПК

Отношение композиции

Отношение композиции, как уже упоминалось ранее, является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Возможно, не самый лучший, но наверняка понятный всем пример этого отношения представляет собой живая клетка в биологии. Другой пример — окно интерфейса программы, которое может состоять из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния. Нетрудно понять, что подобное окно представляет собой класс, а его компоненты являются как классами, так и атрибутами или свойствами окна. Последнее обстоятельство весьма характерно для отношения композиции, поскольку отражает различные способы представления данного отношения.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис. 24).

Рис. 24 Графическое изображение отношения композиции в языке UML

В качестве дополнительных обозначений для отношений композиции и агрегации могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, указание кратности класса ассоциации и имени данной ассоциации, которые не являются обязательными. Применительно к описанному выше примеру класса "Окно_программы" его диаграмма классов может иметь следующий вид (рис. 5.11).

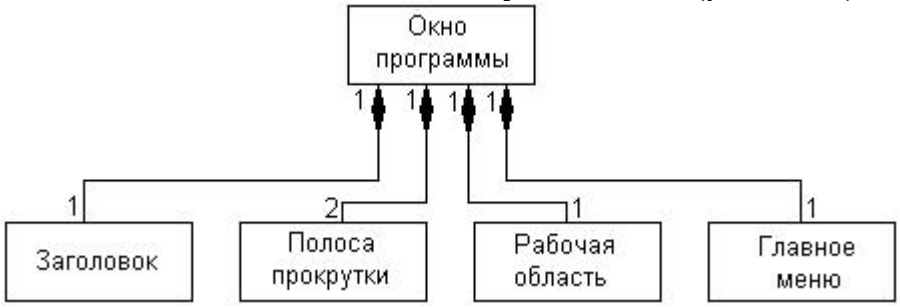

Рис. 25. Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

Данный пример может иллюстрировать и другие особенности разрабатываемой компьютерной программы, которые не указывались в явном виде при описании этого примера. Так, в частности, указание кратности 1 рядом с классом "Рабочая_область" характерно для однодокументных приложений.

Деревья наследования

При использовании отношений обобщения строится иерархия классов. Некоторые классы в этой иерархии могут быть абстрактными. *Абстрактным* называют класс, который не может иметь экземпляров. Имена абстрактных классов записываются курсивом. Например, на рис. 26 показаны абстрактные классы *Млекопитающие, Собаки, Кошки.*

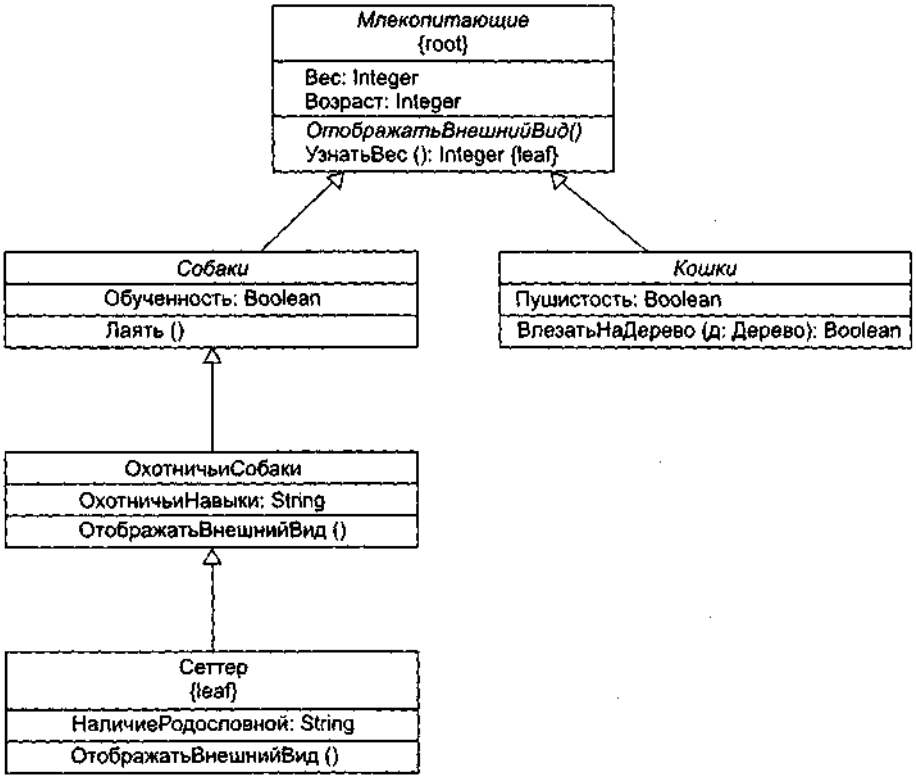

Рис. 26. Абстрактность и полиморфизм

Кроме того, здесь имеются конкретные классы ОхотничьиСобаки, Сеттер, каждый из которых может иметь экземпляры.

Обычно класс наследует какие-то характеристики класса-родителя и передает свои характеристики классу-потомку. Иногда требуется определить *конечный* класс, который не может иметь детей. Такие классы помечаются теговой величиной (характеристикой) leaf, записываемой за именем класса. Например, на рисунке показан конечный класс Сеттер.

Иногда полезно отметить *корневой* класс, который не может иметь родителей. Такой класс помечается теговой величиной (характеристикой) root, записываемой за именем класса. Например, на рисунке показан корневой класс *Млекопитающие.*

Аналогичные свойства имеют и операции. Обычно операция является полиморфной, это значит, что в различных точках иерархии можно определять операции с похожей сигнатурой. Такие операции из дочерних классов переопределяют поведение соответствующих операций из родительских классов. При обработке сообщения (в период выполнения) производится полиморфный выбор одной из операций иерархии в соответствии с типом объекта. Например, ОтображатьВнешнийВид () и

ВлезатьНаДерево (дуб) — полиморфные операции. К тому же операция *Млекопитающие::ОтображатьВнешнийВид ( )* является абстрактной, то есть неполной и требующей для своей реализации потомка. Имя абстрактной операции записывается курсивом (как и имя класса). С другой стороны, Млекопитающие::УзнатьВес () — конечная операция, что отмечается характеристикой leaf. Это значит, что операция не полиморфна и не может перекрываться.

Интерфейсы

Интерфейсы являются элементами диаграммы вариантов. Однако при построении диаграммы классов отдельные интерфейсы могут уточняться, и в этом случае для их изображения используется специальный графический символ — прямоугольник класса с ключевым словом или стереотипом "interface" (рис. 27). При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

Рис. 27. Пример графического изображения интерфейса на диаграмме классов