**11.2. Основные понятия диаграмм классов UML**

*Диаграммой классов* в терминологии UML называется диаграмма, на которой показан набор классов (и некоторых других сущностей[50)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote50), не имеющих явного отношения к проектированию БД), а также связей между этими классами.[51)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote51) Кроме того, диаграмма классов может включать комментарии и ограничения. Ограничения могут неформально задаваться на естественном языке или же могут формулироваться на языке объектных ограничений OCL (Object Constraints Language).[52)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote52) Чуть позже мы обсудим эту тему более подробно.

**11.2.1. Классы, атрибуты, операции**

*Классом* называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Графически класс изображается в виде прямоугольника. У каждого класса должно быть имя (текстовая строка), уникально отличающее его от всех других классов. При формировании имен классов в UML допускается использование произвольной комбинации букв, цифр и даже знаков препинания. Однако на практике рекомендуется использовать в качестве имен классов короткие и осмысленные прилагательные и существительные, каждое из которых начинается с заглавной буквы. Примеры описания классов показаны на [рис. 11.1](http://citforum.ru/database/advanced_intro/31.shtml#image.11.1).

  
*Рис. 11.1.*  Примеры описания классов

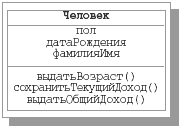
*Атрибутом класса* называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов (в частности, не иметь ни одного атрибута). Свойство, выражаемое атрибутом, является свойством моделируемой сущности, общим для всех объектов данного класса. Так что атрибут является абстракцией состояния объекта. Любой атрибут любого объекта класса должен иметь некоторое значение.

Имена атрибутов представляются в разделе класса, расположенном под именем класса. Хотя UML не накладывает ограничений на способы создания имен атрибутов (имя атрибута может быть произвольной текстовой строкой), на практике рекомендуется использовать короткие прилагательные и существительные, отражающие смысл соответствующего свойства класса. Первое слово в имени атрибута рекомендуется писать с прописной буквы, а все остальные слова – с заглавной. Пример описания класса с указанными атрибутами показан на [рис. 11.2](http://citforum.ru/database/advanced_intro/31.shtml#image.11.2).

  
*Рис. 11.2.*  Класс Человек с указанными именами атрибутов

*Операцией класса* называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция – это абстракция того, что можно делать с объектом. Класс может содержать любое число операций (в частности, не содержать ни одной операции). Набор операций класса является общим для всех объектов данного класса.

Операции класса определяются в разделе, расположенном ниже раздела с атрибутами. При этом можно ограничиться только указанием имен операций, оставив детальную спецификацию выполнения операций на более поздние этапы моделирования. Для именования операций рекомендуется использовать глагольные формы, соответствующие ожидаемому поведению объектов данного класса. Описание операции может также содержать ее *сигнатуру*, т. е. имена и типы всех параметров, а если операция является функцией, то и тип ее значения. Класс Человек с определенными операциями показан на [рис. 11.3](http://citforum.ru/database/advanced_intro/31.shtml#image.11.3).

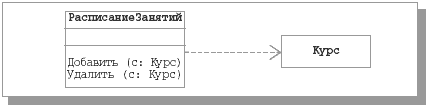
  
*Рис. 11.3.*  Класс Человек с операциями

Для класса Человек мы определили три операции: выдатьВозраст, сохранитьТекущийДоход, выдатьОбщийДоход. В операции выдатьВозраст используются значение атрибута датаРождения и значение текущей даты. Операция сохранитьТекущийДоход позволяет зафиксировать в состоянии объекта сумму и дату поступления дохода данного человека. Операция выдатьОбщийДоход выдает суммарный доход данного человека за указанное время. Заметим, что состояние объекта меняется при выполнении только второй операции. Результаты первой и третьей операций формируются на основе текущего состояния объекта.

**11.2.2. Категории связей. Связь-зависимость**

В диаграмме классов могут участвовать связи трех разных категорий: зависимость (dependency), обобщение (generalization) и ассоциация (association). При проектировании реляционных БД наиболее важны вторая и третья категории связей, поэтому о связях-зависимостях будет сказано только самое основное.

*Зависимостью* называют связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый класс. Чаще всего зависимости применяют в диаграммах классов, чтобы отразить в сигнатуре операции одного класса тот факт, что параметром этой операции могут быть объекты другого класса. Понятно, что если интерфейс второго класса изменяется, это влияет на поведение объектов первого класса. Простой пример диаграммы классов со связью-зависимостью показан на [рис. 11.4](http://citforum.ru/database/advanced_intro/31.shtml#image.11.4).

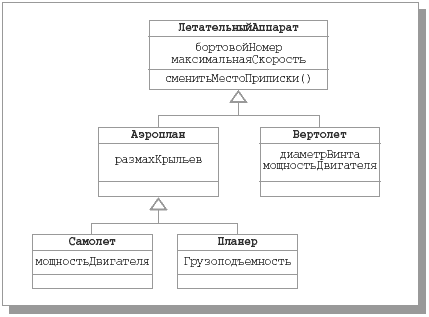
  
*Рис. 11.4.*  Диаграмма классов со связью-зависимостью

Зависимость показывается прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость. Очевидно, что связи-зависимости существенны для объектно-ориентированных систем (в том числе и для ООБД). При проектировании реляционных БД непонятно, что делать с зависимостями (как воспользоваться этой информацией в реляционной БД?).

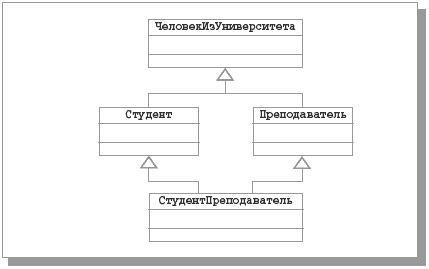
**11.2.3. Связи-обобщения и механизм наследования классов в UML**

*Связью-обобщением* называется связь между общей сущностью, называемой *суперклассом*, или родителем, и более специализированной разновидностью этой сущности, называемой *подклассом*, или потомком. Обобщения иногда называют *связями «is a»*, имея в виду, что класс-потомок является частным случаем класса-предка. Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции.

Объекты класса-потомка могут использоваться везде, где могут использоваться объекты класса-предка. Это свойство называют *полиморфизмом по включению*, имея в виду, что объекты потомка можно считать включаемыми во множество объектов класса-предка. Графически обобщения изображаются в виде сплошной линии с большой незакрашенной стрелкой, направленной к суперклассу. В качестве первой иллюстрации, приведенной на [рис. 11.5](http://citforum.ru/database/advanced_intro/31.shtml#image.11.5), воспользуемся классификацией летательных аппаратов с [рис. 10.12](http://citforum.ru/database/advanced_intro/28.shtml#image.10.12) из предыдущей лекции. На [рис. 11.5](http://citforum.ru/database/advanced_intro/31.shtml#image.11.5) показан пример иерархии *одиночного наследования*: у каждого подкласса имеется только один суперкласс.

  
*Рис. 11.5.*  Иерархия одиночного наследования классов

Одиночное наследование является достаточным в большинстве случаев применения связи-обобщения. Однако в UML допускается и *множественное наследование*, когда один подкласс определяется на основе нескольких суперклассов. В качестве одного из разумных (не слишком распространенных) примеров рассмотрим диаграмму классов на [рис. 11.6](http://citforum.ru/database/advanced_intro/31.shtml#image.11.6) (для упрощения диаграммы имена атрибутов и операций указывать не будем).

  
*Рис. 11.6.*  Пример множественного наследования классов

На этой диаграмме классы Студент и Преподаватель порождены из одного суперкласса ЧеловекИзУниверситета. Вообще говоря, к классу Студент относятся те объекты класса ЧеловекИзУниверситета, которые соответствуют студентам, а к классу Преподаватель – объекты класса ЧеловекИзУниверситета, соответствующие преподавателям. Но, как это часто случается, многие студенты уже в студенческие годы начинают преподавать, так что могут существовать такие два объекта классов Студент и Преподаватель, которым соответствует один объект класса ЧеловекИзУниверситета. Итак, среди объектов класса Студент могут быть преподаватели, а некоторые преподаватели могут быть студентами. Тогда мы можем определить класс СтудентПреподаватель путем множественного наследования от суперклассов Студент и Преподаватель. Объект класса СтудентПреподаватель обладает всеми свойствами и операциями классов Студент и Преподаватель и может быть использован везде, где могут применяться объекты этих классов. Так что полиморфизм по включению продолжает работать. Следует отметить, что множественное наследование, помимо того что не слишком часто требуется на практике, порождает ряд проблем, из которых одной из наиболее известных является проблема именования атрибутов и операций в подклассе, полученном путем множественного наследования. Например, предположим, что при образовании подклассов Студент и Преподаватель в них обоих был определен атрибут с именем номерКомнаты. Очень вероятно, что для объектов класса Студент значениями этого атрибута будут номера комнат в студенческом общежитии, а для объектов класса Преподаватель – номера служебных кабинетов. Как быть с объектами класса СтудентПреподаватель, для которых существенны оба одноименных атрибута (у студента-преподавателя могут иметься и комната в общежитии, и служебный кабинет)? На практике применяется одно из следующих решений:

1. запретить образование подкласса СтудентПреподаватель, пока в одном из суперклассов не будет произведено переименование атрибута номерКомнаты;
2. наследовать это свойство только от одного из суперклассов, так что, например, значением атрибута номерКомнаты у объектов класса СтудентПреподаватель всегда будут номера служебных кабинетов;
3. унаследовать в подклассе оба свойства, но автоматически переименовать оба атрибута, чтобы прояснить их смысл; назвать их, например, номерКомнатыСтудента и номерКомнатыПреподавателя.

Ни одно из решений не является полностью удовлетворительным. Первое решение требует возврата к ранее определенному классу, имена атрибутов и операций которого, возможно, уже используются в приложениях. Второе решение нарушает логику наследования, не давая возможности на уровне подкласса использовать все свойства суперклассов. Наконец, третье решение заставляет использовать длинные имена атрибутов и операций, которые могут стать недопустимо длинными, если процесс множественного наследования будет продолжаться от полученного подкласса.[53)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote53)

Но, конечно, сложность проблемы именования атрибутов и операций несопоставимо меньше сложности реализации множественного наследования в реляционных БД. Поэтому при использовании UML для проектирования реляционных БД нужно очень осторожно использовать наследование классов вообще и стараться избегать множественного наследования.[54)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote54)

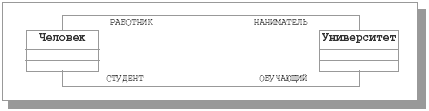
**11.2.4. Связи-ассоциации: роли, кратность, агрегация**

*Ассоциацией* называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса. Допускается, чтобы оба конца ассоциации относились к одному классу. В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих сразу *n* классов (они называются *n-арными ассоциациями*).[55)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote55) Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами.

С понятием ассоциации связаны четыре важных дополнительных понятия: имя, роль, кратность и агрегация. Во-первых, ассоциации может быть присвоено имя, характеризующее природу связи. Смысл имени уточняется с помощью черного треугольника, который располагается над линией связи справа или слева от имени ассоциации. Этот треугольник указывает направление чтения имя связи. Пример именованной ассоциации показан на [рис. 11.7](http://citforum.ru/database/advanced_intro/31.shtml#image.11.7). Треугольник показывает, что именованная ассоциация должна читаться как «Студент учится в Университете».

  
*Рис. 11.7.*  Пример именованной ассоциации

Другим способом именования ассоциации является указание роли каждого класса, участвующего в этой ассоциации. Роль класса, как и имя конца связи в ER-модели, задается именем, помещаемым под линией ассоциации ближе к данному классу. На [рис. 11.8](http://citforum.ru/database/advanced_intro/31.shtml#image.11.8) показаны две ассоциации между классами Человек и Университет, в которых эти классы играют разные роли. Как мы видим, объекты класса Человек могут выступать в роли РАБОТНИКОВ при участии в ассоциации, в которой объекты класса Университет играют роль НАНИМАТЕЛЯ. В другой ассоциации объекты класса Человек играют роль СТУДЕНТА, а объекты класса УНИВЕРСИТЕТ – роль ОБУЧАЮЩЕГО.

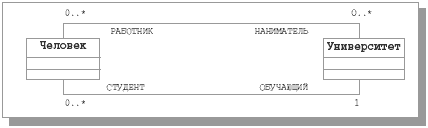
  
*Рис. 11.8.*  Две ассоциации с разными ролями классов

В общем случае, для ассоциации могут задаваться и ее собственное имя, и имена ролей классов. Это связано с тем, что класс может играть одну и ту же роль в разных ассоциациях, так что в общем случае пара имен ролей классов не идентифицирует ассоциацию. С другой стороны, в простых случаях, когда между двумя классами определяется только одна ассоциация, можно вообще не связывать с ней дополнительные имена.

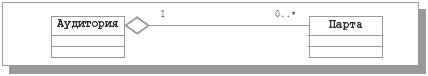
*Кратностью (multiplicity) роли ассоциации* называется характеристика, указывающая, сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации (в UML экземпляр ассоциации называется *соединением – link*, но мы не будем здесь использовать этот термин, чтобы не создавать путаницу – все-таки трудно одновременно говорить про *связи*, *ассоциации* и *соединения*, имея в виду разные понятия). Наиболее распространенным способом задания кратности роли ассоциации является указание конкретного числа или диапазона. Например, указание «1» говорит о том, что каждый объект класса с данной ролью должен участвовать в некотором экземпляре данной ассоциации, причем в каждом экземпляре ассоциации может участвовать ровно один объект класса с данной ролью. Указание диапазона «0..1» говорит о том, что не все объекты класса с данной ролью обязаны участвовать в каком-либо экземпляре данной ассоциации, но в каждом экземпляре ассоциации может участвовать только один объект. Аналогично, указание диапазона «1..\*» говорит о том, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должен участвовать хотя бы один объект (верхняя граница не задана). Толкование диапазона «0..\*» является очевидным расширением случая «0..1».

В более сложных (но крайне редко встречающихся на практике) случаях определения кратности можно использовать списки диапазонов. Например, список «2, 4..6, 8..\*» говорит о том, что все объекты класса с указанной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должны участвовать два, от четырех до шести или более семи объектов класса с данной ролью.

На диаграмме классов с [рис. 11.9](http://citforum.ru/database/advanced_intro/31.shtml#image.11.9) показано, что произвольное (может быть, нулевое) число людей являются служащими произвольного числа университетов. Каждый университет обучает произвольное (может быть, нулевое) число студентов, но каждый студент может быть студентом только одного университета.

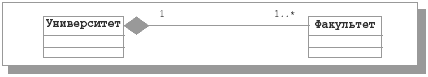
  
*Рис. 11.9.*  Ассоциации с указанными кратностями ролей

Обычная ассоциация между двумя классами характеризует связь между равноправными сущностями: оба класса находятся на одном концептуальном уровне. Но иногда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид «часть-целое». В этом случае класс «целое» имеет более высокий концептуальный уровень, чем класс «часть». Ассоциация такого рода называется *агрегатной*. Графически агрегатные ассоциации изображаются в виде простой ассоциации с незакрашенным ромбом на стороне класса-«целого». Простой пример агрегатной ассоциации показан на [рис. 11.10](http://citforum.ru/database/advanced_intro/31.shtml#image.11.10).

  
*Рис. 11.10.*  Пример агрегатной ассоциации

Объектами класса Аудитория являются студенческие аудитории, в которых проходят занятия. В каждой аудитории должны быть установлены парты. Поэтому в некотором смысле класс Парта является «частью» класса Аудитория. Мы умышленно сделали роль класса Парта необязательной, поскольку могут существовать аудитории без парт (например, класс для занятий танцами) и некоторые парты могут находиться на складе. Обратите внимание, что, хотя аудитории, не оснащенные партами, как правило, непригодны для занятий, объекты классов Аудитория и Парта существуют независимо. Если некоторая аудитория ликвидируется, то находящиеся в ней парты не уничтожаются, а переносятся на склад.

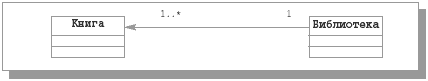
Бывают случаи, когда связь «части» и «целого» настолько сильна, что уничтожение «целого» приводит к уничтожению всех его «частей». Агрегатные ассоциации, обладающие таким свойством, называются *композитными*, или просто композициями. При наличии композиции объект-часть может быть частью только одного объекта-целого (композита). При обычной агрегатной ассоциации «часть» может одновременно принадлежать нескольким «целым». Графически композиция изображается в виде простой ассоциации, дополненной закрашенным ромбом со стороны «целого». Пример композитной агрегатной ассоциации показан на [рис. 11.11](http://citforum.ru/database/advanced_intro/31.shtml#image.11.11).

  
*Рис. 11.11.*  Пример композитной агрегатной ассоциации

Любой факультет является частью одного университета, и ликвидация университета приводит к ликвидации всех существующих в нем факультетов (хотя во время существования университета отдельные факультеты могут ликвидироваться и создаваться).

Заметим, что в контексте проектирования реляционных БД агрегатные и в особенности композитные ассоциации влияют только на способ поддержки ссылочной целостности. В частности, композитная связь является явным указанием на то, что ссылочная целостность между «целым» и «частями» должна поддерживаться путем каскадного удаления частей при удалении целого. Подробнее способы поддержки ссылочной целостности в SQL-ориентированных БД рассматриваются в следующих лекциях.

При наличии простой ассоциации между двумя классами (например, ассоциации между классами Студент и Университет с [рис. 11.7](http://citforum.ru/database/advanced_intro/31.shtml#image.11.7)) предполагается возможность навигации между объектами, входящими в один экземпляр ассоциации. Если известен конкретный объект-студент, то должна обеспечиваться возможность узнать соответствующий объект-университет. Если известен конкретный объект-университет, то должна обеспечиваться возможность узнать все соответствующие объекты-студенты. Другими словами, если не оговорено иное, то навигация по ассоциации может проводиться в обоих направлениях.[56)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote56) Однако бывают случаи, когда желательно ограничить направление навигации для некоторых ассоциаций. В этом случае на линии ассоциации ставится стрелка, указывающая направление навигации. Пример показан на [рис. 11.12](http://citforum.ru/database/advanced_intro/31.shtml#image.11.12).

  
*Рис. 11.12.*  Ассоциация с указанным направлением навигации

В библиотеке должно содержаться некоторое количество книг, и каждая книга должна принадлежать некоторой библиотеке. С точки зрения библиотечного хозяйства разумно иметь возможность найти книгу в библиотеке, т. е. произвести навигацию от объекта-библиотеки к связанным с ним объектам-книгам. Однако вряд ли потребуется по данному экземпляру книги узнать, в какой библиотеке она находится.[57)](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote57)

[50](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote50_back)   В этой лекции мы используем термин сущность настолько же неформально, как в предыдущей лекции использовали термин объект. UML претендует на обеспечение более точного и формального понятия объекта (UML обычно называют языком объектно-ориентированного моделирования). В спецификации языка UML даже присутствует определение понятия объекта средствами самого UML. Однако, по нашему глубокому убеждению, несмотря на эти попытки, понятие объекта в UML остается таким же нечетким, как и понятие сущности в ER-модели. По-прежнему приходится опираться в основном на интуицию и здравый смысл.

[51](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote51_back)   В UML, как и в модели ER-диаграмм, для родового обозначения связей используется термин relationship. Во многих переводах книг про UML на русский язык вместо термина связь применяется термин отношение. Как и в предыдущей лекции, мы используем термин связь.

[52](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote52_back)   Язык OCL является частью общей спецификации UML, но, в отличие от других частей языка, имеет не графическую, а линейную нотацию.

[53](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote53_back)   Как кажется, здесь можно провести некоторую аналогию с ситуацией, по причине наличия которой в реляционной алгебре (см. лекции 4 и 5) была введена операция RENAME.

[54](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote54_back)   Если под «реляционными» базами данных понимать SQL-ориентированные БД.

[55](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote55_back)   Напомним, что в варианте ER-модели, рассмотренном нами в предыдущей лекции, допускались только бинарные связи. В свое время компания Oracle обосновывала это решение тем, что наличие бинарных ассоциаций всегда является достаточным. Здесь мы также ограничимся обсуждением бинарных ассоциаций.

[56](http://citforum.ru/database/advanced_intro/31.shtml" \l "footnote56_back)   Поскольку UML – это высокоуровневый язык моделирования, в нем не уточняется, что такое навигация в реализационном смысле. Но очевидно, что само появление понятия навигации связано с объектно-ориентированной природой UML. Термин «навигация» является почти ругательным в мире реляционных БД, но для мира объектно-ориентированных БД он вполне естественен, поскольку в этом мире на модельном уровне присутствует понятие ссылки, или указателя.