Лекция 5. ОСНОВНЫЕ ЭЛЕМЕНТЫ ОБЪЕКТНОЙ МОДЕЛИ

К основным понятиям объектно-ориентированного подхода (элементам объектной модели) относятся:

- объект;
- класс;
- атрибут;
- операция;
- полиморфизм (интерфейс);
- компонент;
- связи.

Объект определяется как осязаемая сущность (tangible entity) — предмет или явление (процесс), имеющие четко определяемое поведение. Объект может представлять собой абстракцию некоторой сущности предметной области (объект реального мира) или программной системы (архитектурный объект). Любой объект обладает ***состоянием (state), поведением (behavior) и индивидуальностью (identity).***

Состояние объекта — одно из возможных условий, в которых он может существовать, оно изменяется со временем. Состояние объекта характеризуется перечнем всех возможных (статических) свойств данного объекта и текущими значениями (динамическими) каждого из этих свойств. Состояние объекта определяется значениями его свойств *(атрибутов)* и связями с другими объектами.

Поведение определяет действия объекта и его реакцию на запросы от других объектов. Поведение характеризует воздействие объекта на другие объекты, изменяющее их состояние. Иначе говоря, поведение объекта полностью определяется его действиями. Поведение представляется с помощью набора ***сообщений,*** воспринимаемых объектом ***(операций,*** которые может выполнять объект).

Каждый объект обладает уникальной ***индивидуальностью.*** Индивидуальность — это свойства объекта, отличающие его от всех других объектов.

Структура и поведение схожих объектов определяют общий для них класс. Термины «экземпляр класса» и «объект» являются эквивалентными.

Графическое представление объектов в языке моделирования UML (который будет рассматриваться в подразд. 2.5) показано на рис. 2.37.

Рис. 2.37. Графическое представление объектов

Класс — это множество объектов, связанных общностью свойств, поведения, связей и семантики. Класс инкапсулирует (объединяет) в себе данные (атрибуты) и поведение (операции). Класс является абстрактным определением объекта и служит в качестве шаблона для создания объектов. Графическое представление класса в языке UML показано на рис. 2.38. Класс изображается в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй — его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом).

Любой объект является экземпляром (instance) класса. Определение классов и объектов — одна из самых сложных задач объектно-ориентированного проектирования.

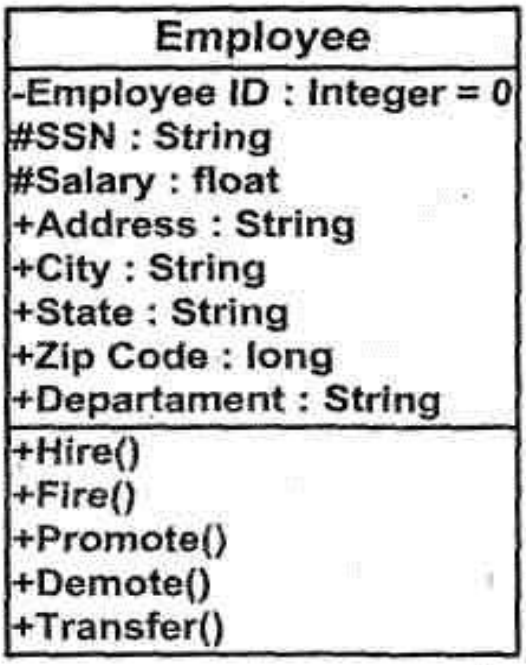

Рис. 2.38. Графическое представление класса

Атрибут — поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства.

Атрибут — это элемент информации, связанный с классом. Например, у класса Company (Компания) могут быть атрибуты Name (Название), Address (Адрес) и NumberOfEmployees (Число служащих).

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attribute visibility).

У атрибута можно определить три возможных значения этого параметра. Рассмотрим каждый из них в контексте примера (см. рис. 2.38). Пусть имеется класс Employee с атрибутом Address и класс Company:

Public (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В таком случае класс Company может изменить значение атрибута Address класса Employee. В соответствии с нотацией UML общему атрибуту предшествует знак «+».

Private (закрытый, секретный). Соответствующий атрибут не виден никаким другим классом. Класс Employee будет знать значение атрибута Address и сможет изменять его, но класс Company не сможет его ни увидеть, ни редактировать. Если это понадобится, он должен попросить класс Employee просмотреть или изменить значение этого атрибута, что обычно делается с помощью общих операций. Закрытый атрибут обозначается знаком «-» в соответствии с нотацией UML.

Protected (защищенный). Такой атрибут доступен только самому классу и его потомкам в иерархии наследования. Допустим, имеется два различных типа сотрудников — с почасовой оплатой и на окладе. Таким образом, потомками класса Employee будут два класса — HourlyEmp и SalariedEmp. Защищенный атрибут Address можно просмотреть или изменить из классов Employee, HourlyEmp и SalariedEmp, но не из класса Company. Нотация UML для защищенного атрибута — это знак «#».

В общем случае атрибуты рекомендуется делать закрытыми или защищенными. При этом удается избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут.

Определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию называется **операцией. Операция — это реализация услуги, которую можно запросить у любого объекта данного класса.**

Операции отражают поведение объекта. Операция-запрос не изменяет состояния объекта. Операция-команда может изменить состояние объекта. Результат операции зависит от текущего состояния объекта.

Как правило, в объектных и объектно-ориентированных языках программирования операции, выполняемые над данным объектом, называются ***методами*** и являются составной частью определения класса.

Операции реализуют связанное с классом поведение (иначе говоря, реализуют ***обязанности класса — responsibilities).*** В широком смысле обязанности класса делятся на две категории.

Знание (определяется атрибутами класса):

- наличие информации о данных или вычисляемых величинах;
- наличие информации о связанных объектах. Действие (определяется операциями класса):
- выполнение некоторых действий самим объектом;
- инициация действий других объектов;
- координация действий других объектов.

Операция включает три части — имя, параметры и тип возвращаемого значения. Параметры — это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент1: тип данных аргумента1, аргумент2: тип данных аргумента2,...): тип возвращаемого значения.

Существуют четыре различных типа операций.

- ***Операции реализации (implementor operations)*** реализуют некоторые функции (процедуры). Такие операции выявляются путем анализа диаграмм взаимодействия UML.
- ***Операции управления (manager operations)*** управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.
- Атрибуты обычно бывают закрытыми или защищенными. Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют ***операции доступа (access operations)***.

- ***Вспомогательными (helper operations)*** называются такие операции класса, которые необходимы ему для выполнения его обязанностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Понятие ***полиморфизма*** может быть интерпретировано, как способность класса принадлежать более чем одному типу.

Полиморфизм — это способность скрывать множество различных реализаций под единственным общим интерфейсом.

Интерфейс — совокупность операций, определяющих набор услуг класса или компонента. Интерфейс не определяет внутреннюю структуру, все его операции имеют открытую видимость

Полиморфизм тесно связан с наследованием. ***Наследование*** означает построение новых классов на основе существующих с возможностью добавления или переопределения свойств (атрибутов) и поведения (операций).

Объектно-ориентированная система изначально строится с учетом ее эволюции. Наследование и полиморфизм обеспечивают возможность определения новой функциональности классов с помощью создания производных классов — потомков базовых классов. Потомки наследуют характеристики родительских классов без изменения их первоначального описания и добавляют при необходимости собственные структуры данных и методы. Определение производных классов, при котором задаются только различия или уточнения, в огромной степени экономит время и усилия при производстве и использовании спецификаций и программного кода.

Компонент — относительно независимая и замещаемая часть системы, выполняющая четко определенную функцию в контексте заданной архитектуры. Компонент представляет собой физическую реализацию проектной абстракции и может быть:

- компонентом исходного кода;
- компонентом времени выполнения (run time);
- исполняемым компонентом.

Компонент обеспечивает физическую реализацию набора интерфейсов.

Между элементами объектной модели существуют различные виды связей. К основным типам связей относятся связи ассоциации, зависимости и обобщения.

Ассоциация (association) — это семантическая связь между классами. Ее изображают на диаграмме классов в виде обыкновенной линии (рис. 2.39). Ассоциация отражает структурные связи между объектами различных классов.

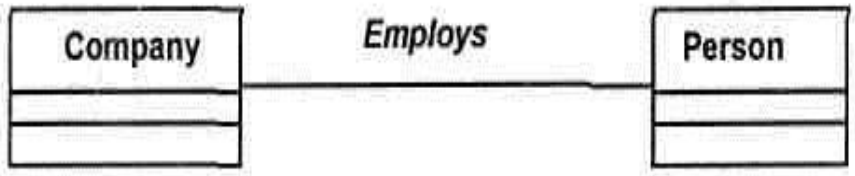

Рис. 2.39. Ассоциация

Агрегация (aggregation) представляет собой форму ассоциации — более сильный тип связи между целым (составным) объектом и его частями (компонентными объектами).

Существуют четыре возможных семантики для агрегации:

1) агрегация типа «Безраздельно обладает»;

2) агрегация типа «Обладает»;

3) агрегация типа «Включает»;

4) агрегация типа «Участник».

Агрегация типа «Безраздельно обладает» устанавливает следующее:

- между компонентными объектами и их составными объектами установлено отношение *зависимости по существованию* (следовательно, удаление составного объекта распространяется вниз по иерархии отношения, так что связанные компонентные объекты также удаляются);
- агрегация *транзитивна* (если объект C1 является частью объекта B1, а B1 — частью A1, тогда C1 является частью A1);

- агрегация *асимметрична (нерефлексивна)* (если объект B1 является частью объекта A1, то объект A1 не является частью B1*);*
- агрегация *стационарна* (если объект B1 является частью объекта A1, то он не может быть частью объекта Ai ($i \neq 1$)).

Агрегация типа «Обладает» поддерживает первые три свойства агрегации «Безраздельно обладает», к которым относятся:

- зависимость по существованию;
- транзитивность;
- асимметричность.

Агрегация типа «Включает» слабее, чем агрегация типа «Обладает», она поддерживает транзитивность и асимметричность.

Агрегация типа «Участник» обладает свойством целенаправленного группирования независимых объектов — группирования, при котором не делается предположений относительно свойства зависимости по существованию, транзитивности, асимметричности или стационарности. Компонентный объект в агрегации типа «Участник» может одновременно принадлежать более чем одному составному объекту.

Язык UML обеспечивает ограниченную поддержку агрегации. Сильная форма агрегации является в UML ***композицией.*** В композиции составной объект может физически содержать компонентные объекты. Компонентный объект может принадлежать только одному составному объекту. Композиция языка UML в большей или меньшей степени соответствует агрегациям типа «Безраздельно обладает» и «Обладает».

Слабая форма агрегации в UML называется просто ***агрегацией.*** При этом составной объект физически не содержит компонентный объект. Один компонентный объект может обладать несколькими связями ассоциации или агрегации. Иначе говоря, агрегация в языке UML соответствует агрегациям типа «Включает» и «Участник».

Агрегация изображается линией между классами с ромбом на стороне целого объекта (рис. 2.40). Сплошной ромб представляет композицию (рис. 2.41).

Рис. 2.40 Агрегация

Рис 2.41. Композиция

Хотя связи ассоциации и агрегации двунаправленные по умолчанию, часто накладываются ограничения на направление навигации (только в одном направлении). Если введено ограничение по направлению, то добавляется стрелка на конце связи. Направление ассоциации можно определить, изучая сообщения между классами. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи — это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Если человек является

сотрудником компании, ассоциацию можно назвать «employs» (нанимает) (см. рис. 2.39).

Имена у связей определять не обязательно. Обычно это делают, если причина создания связи не очевидна. Имя показывают около линии соответствующей связи.

Для уточнения роли, которую играет каждый класс в связях ассоциации или агрегации, применяют ролевые имена (рис. 2.42). Возвращаясь к примеру с классами Person и Company, можно сказать, что класс Person играет роль сотрудника класса Company. Ролевые имена — это обычно имена существительные или основанные на них фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или ролевым именем, или именем связи, но не обоими сразу. Как и имена связей, ролевые имена не обязательны, их дают, только если смысл связи не очевиден.

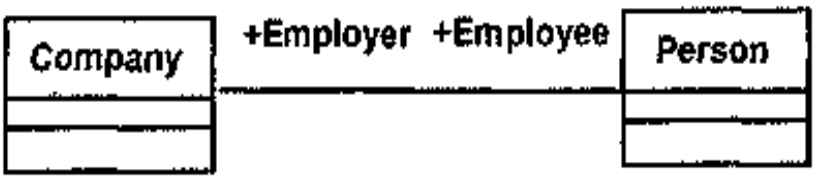

Рис 2.42. Ролевые имена

Мощность (multiplicity) показывает, как много объектов участвует в связи. Мощность — это число объектов одного класса, связанных с одним объектом другого класса. Понятие мощности связи в объектной модели аналогично понятиям мощности и класса принадлежности связи в модели «сущность-связь» (с точностью до расположения показателя мощности на диаграмме). Для каждой связи можно обозначить два показателя мощности — по одному на каждом конце связи.

В языке UML приняты следующие нотации для обозначения мощности.

Значения мощности

Мощность	**Значение**

*****	Много
0	Нуль
1	Один
0..*	Нуль или больше
1..*	Один или больше
0..1	Нуль или один
1..1	Ровно один

Например, при разработке системы регистрации курсов в университете можно определить классы Course (учебный курс) и Student (студент). Между ними установлена связь, означающая посещение курсов студентами. Если один студент может посещать от нуля до четырех курсов, а на одном курсе могут заниматься от 10 до 20 студентов, то на диаграмме классов это можно изобразить следующим образом (рис. 2.43).

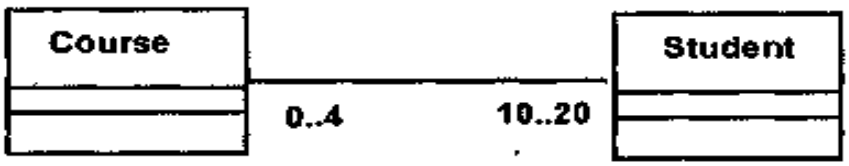

Рис 2.43. Мощность связи

Частным случаем ассоциации является ***ассоциация-класс*** (Association class), которая обладает как свойствами класса, так и свойствами ассоциации. Ассоциация-класс — это место, где хранятся относящиеся к ассоциации атрибуты и операции. Экземплярами ассоциации-класса являются связи, у которых есть не только ссылки на объекты, но и значения атрибутов. Ассоциация-класс подобна связи с атрибутами в модели «сущность-связь».

Допустим, что имеются два класса, Student и Course, и возникла необходимость добавить атрибут Grade (оценка). В таком случае возникает вопрос, в какой класс надо его добавить. Если поместить его внутри класса Student, то придется вводить его для каждого посещаемого студентом курса, что слишком сильно увеличит размер этого класса. Если же поместить его внутри класса Course, то придется задавать его для каждого посещающего этот курс студента.

Чтобы решить эту проблему, можно создать ассоциацию-класс. В этот класс следует поместить атрибут Grade, относящийся к связи между курсом и студентом. Нотация UML для ассоциации-класса представлена на рис. 2.44.

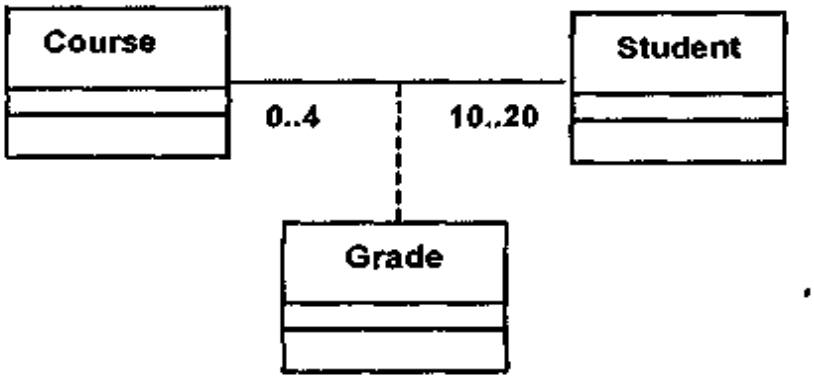

Рис 2.44. Ассоциация-класс

Ассоциация-класс определяет дополнительное ограничение, согласно которому двум участвующим в ассоциации объектам может соответствовать только один экземпляр ассоциации-класса. Диаграмма на рис. 2.44 не допускает, чтобы студент мог получать по курсу более чем одну оценку. Если необходимо, чтобы такое допускалось, то ассоциацию-класс Grade следует преобразовать в обычный класс, связанный ассоциациями с классами Student и Course.

Зависимость (dependency) — связь между двумя элементами модели, при которой изменения в спецификации одного элемента могут повлечь за собой изменения в другом элементе. Зависимость — слабая форма связи между клиентом и сервером (клиент зависит от сервера и не имеет знаний о сервере). Зависимость изображается пунктирной линией, направленной от клиента к серверу (рис. 2.45).

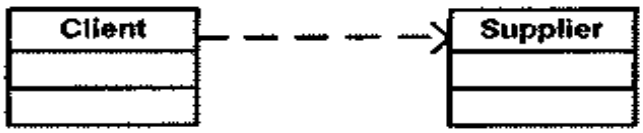

Рис 2.45. Зависимость

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Причины для зависимостей могут быть самыми разными: один класс посылает сообщение другому; один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции. Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Обобщение (generalization) — связь «тип-подтип» реализует механизм наследования (inheritance). Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. В языке UML связи наследования называют обобщениями и изображают в виде стрелок от класса-потомка к классу-предку (рис. 2.46).

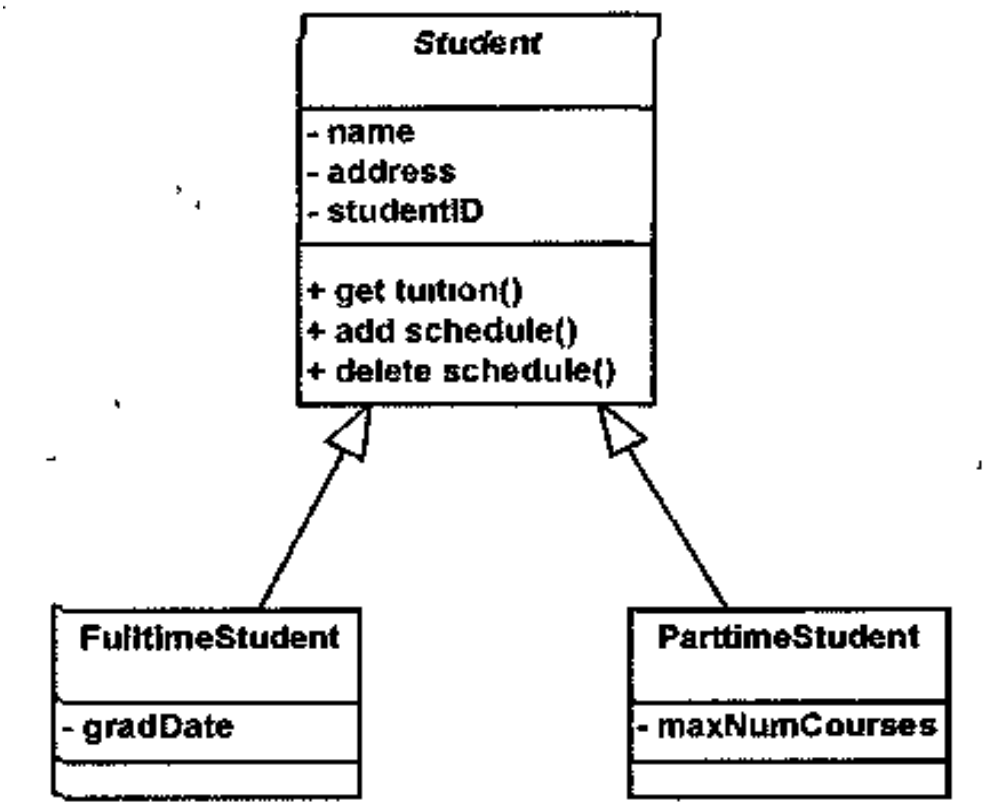

Рис 2.46. Обобщение

Общие атрибуты, операции и/или связи отображаются на верхнем уровне иерархии.

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.