1. Технология построения диаграммы Варианты использования в UML
2. Технология построения диаграммы классов в UML

Вопросы:

1.Диаграммы вариантов использования и их назначение

2.Элементы диаграммы

3.Пример построения диаграммы

4.Понятие класса

5.Отношения между классами

6.Интерфейсы

7.Объекты

8.Шаблоны и параметризованные классы

1. Диаграммы вариантов использования (Use Case)

Визуальное моделирование в UML представляет собой процесс последовательного поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели разрабатываемой программной системы.

Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования.

Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Назначение диаграмм Варианты использования

Разработка диаграммы вариантов использования преследует цели:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть диаграммы use case

Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования.

Актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами.

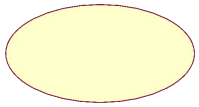
В языке UML пакет Варианты использования является подпакетом пакета Элементы поведения. Последний специфицирует понятия, при помощи которых определяют функциональность моделируемых систем.

Элементы пакета вариантов использования являются первичными по отношению к тем, с помощью которых могут быть описаны сущности, такие как системы и подсистемы. Однако внутренняя структура этих сущностей никак не описывается.

Базовые элементы этого пакета — ***вариант использования и актер.***

Стандартные элементы

Отдельный ***вариант использования*** обозначается на диаграмме эллипсом, внутри которого или рядом с ним содержится его краткое название или имя в форме глагола с пояснительными словами.



Проверить состояние текущего счета

Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности.

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемую сущность или систему по запросу пользователя (актера), т. е. определяет способ применения этой сущности.

Сервис, который инициализируется по запросу пользователя, представляет собой законченную последовательность действий.

- Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности.
- Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы.
- Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы.
- Примерами вариантов использования могут являться следующие действия: проверка состояния текущего счета клиента, оформление заказа на покупку товара, получение дополнительной информации о кредитоспособности клиента, отображение графической формы на экране монитора и другие действия.

Актеры

Актер представляет собой любую ***внешнюю по отношению к моделируемой системе сущность***, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач.

Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования.

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели.

Примерами актеров могут быть: клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

В качестве актеров могут выступать другие системы, подсистемы проектируемой системы или отдельные классы.

Каждый актер определяет некоторое согласованное множество ролей, в которых могут выступать пользователи данной системы в процессе взаимодействия с ней. В каждый момент времени с системой взаимодействует вполне определенный пользователь, при этом он играет или выступает в одной из таких ролей.

Наиболее наглядный пример актера — конкретный пользователь системы со своими собственными параметрами аутентификации.

Интерфейсы

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры.

В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров.

Примечания

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения.

Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Отношения на диаграмме вариантов использования

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов. Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис.

Два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности.

Варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы.

- В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:
- Отношение ассоциации (association relationship)
- Отношение расширения (extend relationship)
- Отношение обобщения (generalization relationship)
- Отношение включения (include relationship)

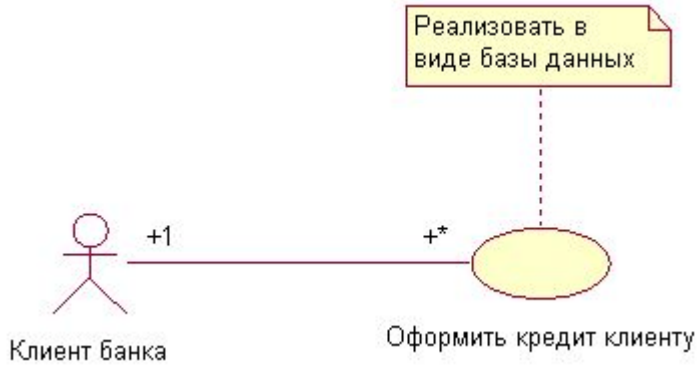
Отношение ассоциации

Отношение ассоциации является одним из фундаментальных понятий в языке UML.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования.

Ассоциация специфицирует семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования.

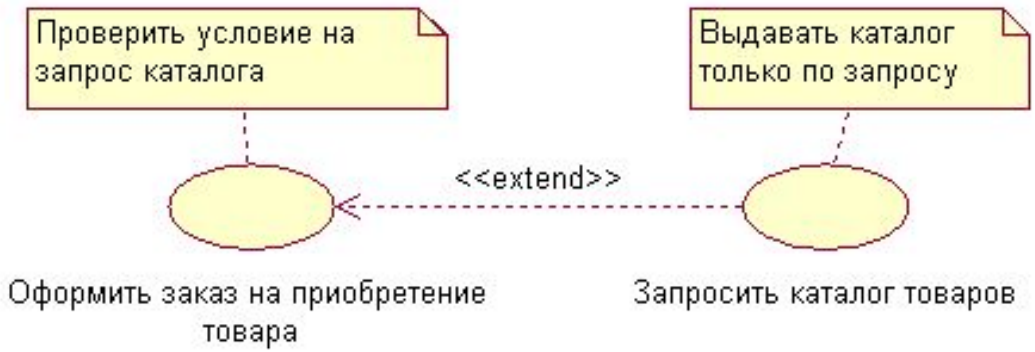
Кратность характеризует общее количество конкретных экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации.
Здесь кратность "*" означает, что каждый отдельный клиент банка может оформить для себя несколько кредитов, при этом их общее число заранее неизвестно и ничем не ограничивается.



Отношение расширения

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров.

Расширение является направленным и указывает, что применительно к отдельным примерам некоторого варианта использования должны быть выполнены конкретные условия, определенные для расширения данного варианта использования. Так, если имеет место отношение расширения от варианта использования A к варианту использования B, то это означает, что свойства экземпляра варианта использования B могут быть дополнены благодаря наличию свойств у расширенного варианта использования A.

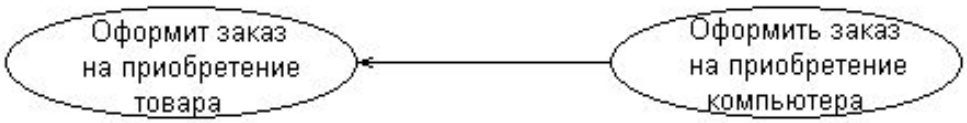


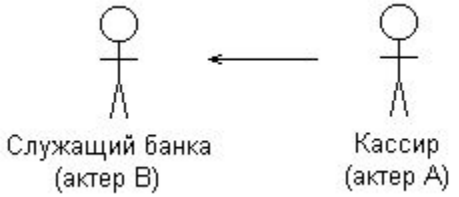
Отношение обобщения

Отношение обобщения служит для указания того факта, что некоторый вариант использования A может быть обобщен до варианта использования B. В этом случае вариант A будет являться специализацией варианта B. При этом B называется предком или родителем по отношению A, а вариант A — потомком по отношению к варианту использования B.

Потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения.

Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов.



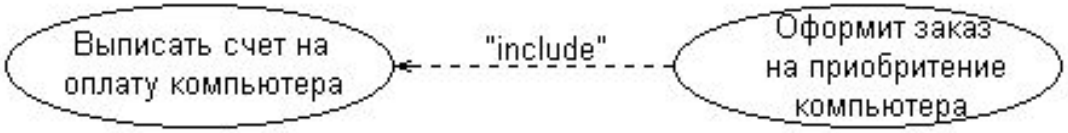


Отношение включения

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения.

Отношение включения, направленное от варианта использования A к варианту использования B, указывает, что каждый экземпляр варианта A включает в себя функциональные свойства, заданные для варианта B. Эти свойства специализируют поведение соответствующего варианта A на диаграмме.



Пример диаграммы вариантов использования

В качестве примера рассмотрим процесс моделирования системы продажи товаров по каталогу, которая может быть использована при создании соответствующих информационных систем.

В качестве актеров данной системы могут выступать два субъекта, один из которых является продавцом, а другой — покупателем. Каждый из этих актеров взаимодействует с рассматриваемой системой продажи товаров по каталогу и является ее пользователем, т. е. они оба обращаются к соответствующему сервису "Оформить заказ на покупку товара".

Первоначальная структура диаграммы может включать в себя только двух указанных актеров и единственный вариант использования

На следующем этапе разработки данной диаграммы вариант использования "Оформить заказ на покупку товара" может быть уточнен на основе введения в рассмотрение четырех дополнительных вариантов использования. Это следует из более детального анализа процесса продажи товаров, что позволяет выделить в качестве отдельных сервисов такие действия, как обеспечить покупателя информацией о товаре, согласовать условия оплаты товара и заказать товар со склада.

С другой стороны, продажа товаров по каталогу предполагает наличие самостоятельного информационного объекта — каталога товаров, который в некотором смысле не зависит от реализации сервиса по обслуживанию покупателей. В нашем случае, каталог товаров может запрашиваться покупателем или продавцом при необходимости выбора товара и уточнения деталей его продажи. Вполне резонно представить сервис "Запросить каталог товаров" в качестве самостоятельного варианта использования.

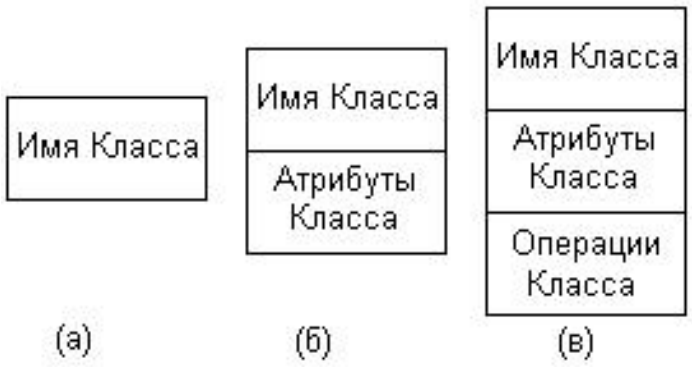
Приведенная диаграмма вариантов использования, в свою очередь, может быть детализирована далее с целью более глубокого уточнения предъявляемых к системе требований и конкретизации деталей ее последующей реализации. В рамках общей парадигмы ООАП подобная детализация может выполняться в двух основных направлениях.

В рамках рассматриваемой системы продажи товаров может иметь самостоятельное значение и специфические особенности отдельная категория товаров — компьютеры. В этом случае диаграмма может быть дополнена вариантом использования "Оформить заказ на покупку компьютера" и актерами "Покупатель компьютера" и "Продавец компьютеров", которые связаны с соответствующими компонентами диаграммы отношением обобщения

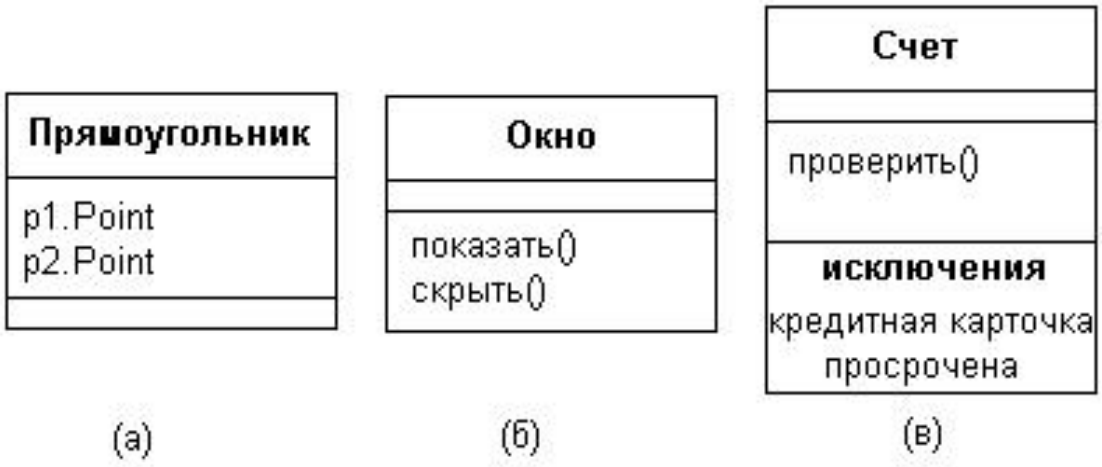
2.Технология построения диаграммы классов в UML

1. Понятие класса

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).



Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.



Имя класса

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов.

Имена классов образуют словарь предметной области при ООАП.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив).

Атрибуты класса

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута>[кратность]:

<тип атрибута> = <исходное значение>{строка-свойство}

Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

- Символ "+" обозначает атрибут с областью видимости типа общедоступный (public).
- Символ "#" обозначает атрибут с областью видимости типа защищенный (protected).
- знак "-" обозначает атрибут с областью видимости типа закрытый (private).

Отсутствие квантора видимости трактуется как public или private

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

[нижняя_граница1 .. верхняя_граница1, нижняя_граница2.. верхняя_грашца2, ..., нижняя_границak .. верхняя_границak]

Нижняя_граница и верхняя_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел. В качестве верхней_границы может использоваться специальный символ "*".

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т. е. в точности 1.

[1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Примеры задания имен и типов атрибутов классов:

цвет: Color — здесь цвет является именем атрибута, Color — именем типа данного атрибута.

имя_сотрудника [1..2] : String — здесь имя_сотрудника является именем атрибута, который служит для представления информации об имени, а возможно, и отчестве конкретного сотрудника.

видимость:Boolean — здесь видимость есть имя абстрактного атрибута, который может характеризовать наличие визуального представления соответствующего класса на экране монитора. В этом случае тип Boolean означает, что возможными значениями данного атрибута является одно из двух логических значений: истина (true) или ложь (false).

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Здесь необходимо придерживаться правила принадлежности значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса. С другой стороны, конструктор соответствующего объекта может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость.

Примеры

- цвет:Color = (255, 0, 0) — в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута.
- видимость:Boolean = истина — может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции — это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут может принимать подмножество значений из некоторой области значений атрибута, определяемой его типом. Эти значения можно рассматривать как набор однотипных записей или массив, которые в совокупности характеризуют каждый объект класса.

Например, некоторый атрибут задан в виде форм: Прямоугольник, то это будет означать, что все объекты данного класса могут иметь несколько различных форм, каждая из которых является прямоугольником.

Другой пример атрибута в виде номер_счета:Integer. что может означать для объекта Сотрудник наличие некоторого подмножества счетов, общее количество которых заранее не фиксируется.

Строка-свойство служит для указания значений атрибута, которые не могут быть изменены в программе при работе с данным типом объектов. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе.

Например, строка-свойство в записи атрибута заработная_плата:Currency = = {$500} может служить для обозначения фиксированной заработной платы для каждого объекта класса "Сотрудник" определенной должности в некоторой организации.

Операция

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка-свойство данной операции:

<квантор видимости><имя операции>(список параметров):

<выражение типа возвращаемого значения>{строка-свойство}

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, отображается при помощи таких же символов.

Имя атрибута является единственным обязательным элементом синтаксического обозначения операции.

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых может быть представлен в следующем виде:

<вид параметра><имя параметра>:<выражение типа>=<значение параметра по умолчанию>.

вид параметра — есть одно из ключевых слов in, out или inout со значением in по умолчанию, в случае если вид параметра не указывается. Имя параметра есть идентификатор соответствующего формального параметра. Выражение типа является зависимой от конкретного языка программирования спецификацией типа возвращаемого значения для соответствующего формального параметра. Наконец, значение по умолчанию в общем случае представляет собой выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нем ограничениям.

Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения.

Строка-свойство служит для указания значений свойств, которые могут быть применены к данному элементу. Строка-свойство не является обязательной, она может отсутствовать, если никакие свойства не специфицированы.

Операция с областью действия на весь класс показывается подчеркиванием имени и строки выражения типа. По умолчанию под областью операции понимается объект класса.

Операция, которая не может изменять состояние системы и, соответственно, не имеет никакого побочного эффекта, обозначается строкой-свойством "{запрос}" ("{query}"). В противном случае операция может изменять состояние системы, хотя нет никаких гарантий, что она будет это делать.

Для повышения производительности системы одни операции могут выполняться параллельно или одновременно, а другие — только последовательно. В этом случае для указания параллельности выполнения операции используется строка-свойство вида "{concurrency = имя}", где имя может принимать одно из следующих значений: последовательная (sequential), параллельная (concurrent), охраняемая (guarded).

Примеры операций:

+создать() — может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров. Эта операция не возвращает никакого значения после своего выполнения.

+нарисовать(форма: Многоугольник = прямоугольник, цвет_заливки: Color = (0, 0, 255)) — может обозначать операцию по изображению на экране монитора прямоугольной области синего цвета, если не указываются другие значения в качестве аргументов данной операции.

выдать_сообщение():{"Ошибка деления на ноль"} — смысл данной операции не требует пояснения, поскольку содержится в строке-свойстве операции. Данное сообщение может появиться на экране монитора в случае попытки деления некоторого числа на ноль, что недопустимо.

2. Отношения между классами

Кроме внутреннего устройства или структуры классов на диаграмме классов указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями или связями в языке UML являются:

Отношение зависимости (dependency relationship) —>

Отношение ассоциации (association relationship)

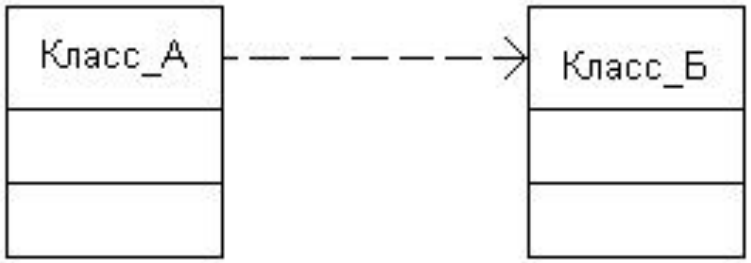
Отношение обобщения (generalization relationship)

Отношение реализации (realization relationship)

Отношение зависимости

В общем случае оно указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику. На рисунке изображены два класса: Класс_А и Класс_Б, при этом Класс_Б является источником некоторой зависимости, а Класс_А — клиентом этой зависимости.



Отношение ассоциации соответствует наличию некоторого отношения между классами. Это отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации.

Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой — вторым (со стороны вершины треугольника).

Тернарная ассоциация и ассоциации более высокой арности связывает некоторым отношением 3 и более классов, при этом один класс может участвовать в ассоциации более чем один раз.

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации.

Некоторый класс может быть присоединен к ромбу пунктирной линией. Это означает, что данный класс обеспечивает поддержку свойств соответствующей N-арной ассоциации, а сама N-арная ассоциация имеет атрибуты, операции и/или ассоциации, т.е является классом .

Данная ассоциация указывает на наличие отношения между этими тремя классами, которое может представлять информацию об играх футбольных команд в национальном чемпионате в течение нескольких последних лет.

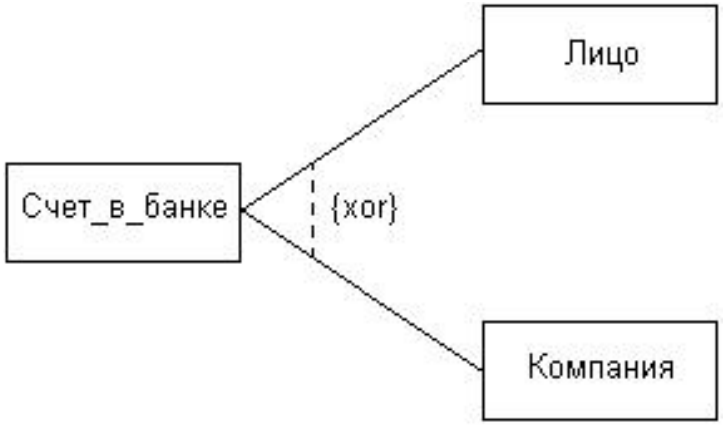
Кратность отдельных классов, являющихся концами ассоциации.

Кратность отдельного класса обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов.

Так, для рассмотренного ранее примера кратность "1" для класса "Компания" означает, что каждый сотрудник может работать только в одной компании. Кратность "1..*" для класса "Сотрудник" означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено.

Частным случаем отношения ассоциации является так называемая исключающая ассоциация (Xor-association). Семантика данной ассоциации указывает на тот факт, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один ее экземпляр.

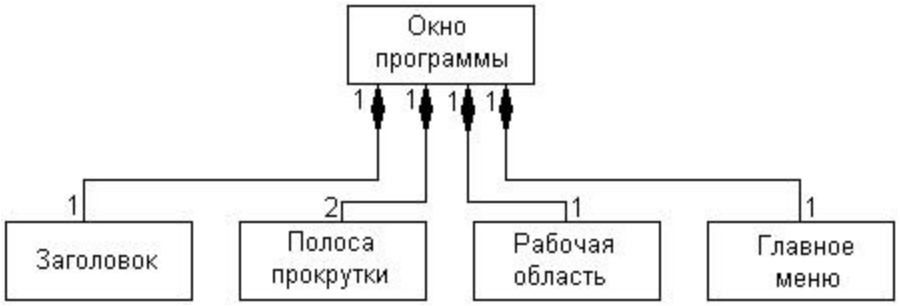
Например, счет в банке может быть открыт для клиента, в качестве которого может выступать физическое лицо (индивидум) или компания, что изображается с помощью исключающей ассоциации.



Частным случаем отношения ассоциации является **отношение агрегации.** Оно 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.

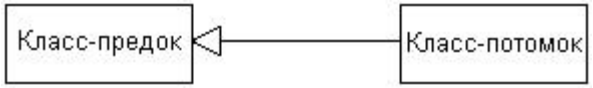
Это отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа "часть-целое".

Отношение композиции 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.



Отношение обобщения является обычным отношением между более общим элементом (родителем или предком) и более частным или специальным элементом

Применительно к диаграмме классов это отношение описывает иерархическое строение классов и наследование их свойств и поведения. (дочерним или потомком).



3. Интерфейсы

Интерфейсы являются элементами диаграммы вариантов использования и были рассмотрены ранее. 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". При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

4. Объекты

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. В силу самых различных причин может возникнуть необходимость показать взаимосвязи не только между классами модели, но и между отдельными объектами, реализующими эти классы. В данном случае может быть разработана диаграмма объектов, которая, хотя и не является канонической в метамодели языка UML, но имеет самостоятельное назначение.

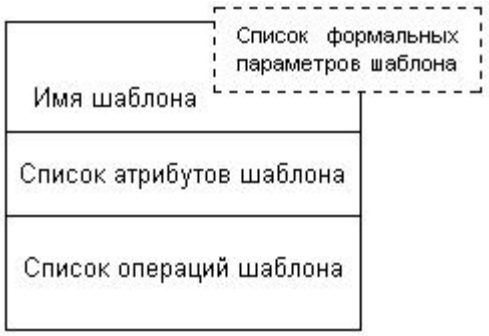
Для графического изображения объектов используется такой же символ прямоугольника, что и для классов. Отличия проявляются при указании имен объектов, которые в случае объектов обязательно подчеркиваются. Запись имени объекта представляет собой строку текста "имя объекта: имя класса", разделенную двоеточием.

Примеры изображения объектов

Анонимный объект

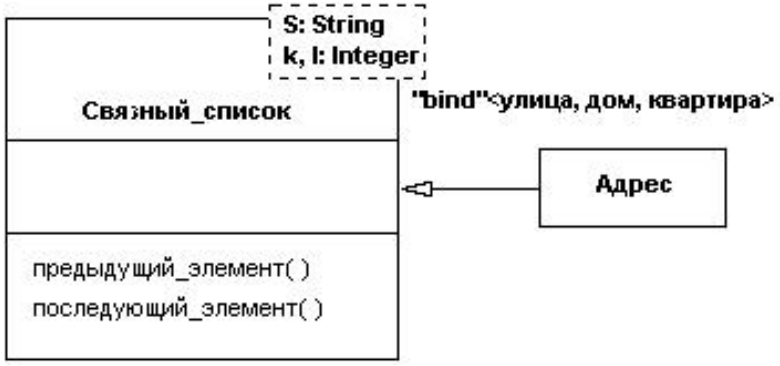
5.Шаблоны или параметризованные классы

Шаблон (template) или параметризованный класс (parametrized class) предназначен для обозначения такого класса, который имеет один (или более) нефиксированный формальный параметр. Он определяет целое семейство или множество классов, каждый из которых может быть получен связыванием этих параметров с действительными значениями. Обычно параметрами шаблонов служат типы атрибутов классов, такие как целые числа, перечисление, массив строк и др. В более сложном случае формальные параметры могут представлять и операции класса.



В верхнем прямоугольнике указывается список формальных параметров для тех классов, которые могут быть получены на основе данного шаблона.

Шаблон не может быть непосредственно использован в качестве класса, поскольку содержит неопределенные параметры. Чаще всего в качестве шаблона выступает некоторый суперкласс, параметры которого уточняются в его классах-потомках.



В данном примере отмечен тот факт, что класс "Адрес" может быть получен из шаблона Связный_список на основе актуализации формальных параметров "S, k, l" фактическими атрибутами "улица, дом, квартира".

Этот же шаблон может использоваться для задания (инстанцирования) другого класса, скажем, класса "Точки_на_плоскости". В этом случае класс "Точки_на_плоскости" актуализирует те же формальные параметры, но с другими значениями, например, “S, k, l” с атрибутами координаты_точки, x, y>.