Содержание

**2**



Введение

1. Обзор Язык UML как средство автоматизированных систем

1.1 Правила языка UML

2. Диаграммы классов

2.1 Имя класса

2.2 Отношения между классами

2.3 Интерфейсы

2.4 Шаблоны или параметризованные классы

2.5 Рекомендации по построению диаграмм классов

## Введение

Унифицированный язык моделирования (UML) является стандартным инструментом для создания "чертежей" программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем.

UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это очень выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию.

*3*



## 1. Обзор Язык UML как средство автоматизированных систем

UML - это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем.

Язык состоит из словаря и правил, позволяющих комбинировать входящие в него слова и получать осмысленные конструкции. В *языке моделирования* словарь и правила ориентированы на концептуальное и физическое представление системы. Язык моделирования, подобный UML, является стандартным средством для составления "чертежей" программного обеспечения.

Моделирование необходимо для понимания системы. При этом единственной модели никогда не бывает достаточно. Напротив, для понимания любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки

Словарь и правила такого языка, как UML, объясняют, как создавать и читать хорошо определенные модели, но ничего не сообщают о том, какие модели и в каких *случаях* нужно создавать. Это задача всего процесса разработки программного обеспечения. Хорошо организованный процесс должен подсказать вам, какие требуются артефакты, какие ресурсы необходимы для их создания, как можно использовать эти артефакты, чтобы оценить выполненную работу и управлять проектом в целом.

UML - это язык визуализации

*2*



Использование UML позволяет решить третью проблему: явная модель облегчает общение.

Некоторые особенности системы лучше всего моделировать в виде текста, другие - графически. На самом деле во всех интересных системах существуют структуры, которые невозможно представить с помощью одного лишь языка программирования. UML - графический язык, что позволяет решить вторую из обозначенных проблем.

UML - это не просто набор графических символов. За каждым из них стоит хорошо определенная семантика. Это значит, что модель, написанная одним разработчиком, может быть однозначно интерпретирована другим - или даже инструментальной программой. Так решается первая из перечисленных выше проблем.

UML - это язык специфицирования

В данном контексте *специфицирование* означает построение точных, недвусмысленных и полных моделей. UML позволяет специфицировать все существенные решения, касающиеся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания системы программного обеспечения.

*4*



UML - это язык конструирования

*5*



UML не является языком визуального программирования, но модели, созданные с его помощью, могут быть непосредственно переведены на различные языки программирования. Иными словами, UML-модель можно отобразить на такие языки, как Java, C++, Visual Basic, и даже на таблицы реляционной базы данных или устойчивые объекты объектно-ориентированной базы данных. Те понятия, которые предпочтительно передавать графически, так и представляются в UML; те же, которые лучше описывать в текстовом виде, выражаются с помощью языка программирования.

Такое отображение модели на язык программирования позволяет осуществлять прямое проектирование: генерацию кода из модели UML в какой-то конкретный язык. Можно решить и обратную задачу: реконструировать модель по имеющейся реализации. Обратное проектирование не представляет собой ничего необычного. Если вы не закодировали информацию в реализации, то эта информация теряется при прямом переходе от моделей к коду. Поэтому для обратного проектирования необходимы как инструментальные средства, так и вмешательство человека. Сочетание прямой генерации кода и обратного проектирования позволяет работать как в графическом, так и в текстовом представлении, если инструментальные программы обеспечивают согласованность между обоими представлениями.

Помимо прямого отображения в языки программирования UML в силу своей выразительности и однозначности позволяет непосредственно исполнять модели, имитировать поведение систем и контролировать действующие системы.

UML - это язык документирования

*6*



Компания, выпускающая программные средства, помимо исполняемого кода производит и другие артефакты, в том числе следующие:

* требования к системе;
* архитектуру;
* проект;
* исходный код;
* проектные планы;
* тесты;
* прототипы;
* версии, и др.

В зависимости от принятой методики разработки выполнение одних работ производится более формально, чем других. Упомянутые артефакты - это не просто поставляемые составные части проекта; они необходимы для управления, для оценки результата, а также в качестве средства общения между членами коллектива во время разработки системы и после ее развертывания.

UML позволяет решить проблему документирования системной архитектуры и всех ее деталей, предлагает язык для формулирования требований к системе и определения тестов,

Где используется UML

Язык UML предназначен прежде всего для разработки программных систем. Его использование особенно эффективно в следующих областях:

информационные системы масштаба предприятия;

*7*



* банковские и финансовые услуги;
* телекоммуникации;
* транспорт;
* оборонная промышленность, авиация и космонавтика;
* розничная торговля;
* медицинская электроника;
* наука;
* распределенные Web-системы.

Строительные блоки UML

Словарь языка UML включает три вида строительных блоков:

* сущности;
* отношения;
* диаграммы.

*Сущности* - это абстракции, являющиеся основными элементами модели. Отношения связывают различные сущности; диаграммы группируют представляющие интерес совокупности сущностей.

*Обобщение* (Generalization) - это отношение "специализация/обобщение", при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (родителя или предка)



**Рис.2.14** Обобщения

Отношения реализации встречаются в двух случаях: во-первых, между интерфейсами и реализующими их классами или компонентами, а во-вторых, между прецедентами и реализующими их кооперациями. Отношение реализации изображается в виде пунктирной линии с незакрашенной стрелкой, как нечто среднее между отношениями обобщения и зависимости (см. рис.2.15).

*2*



Диаграмма в UML - это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации системы с разных точек зрения. Диаграмма - в некотором смысле одна из проекций системы. Как правило, за исключением наиболее тривиальных случаев, диаграммы дают свернутое представление элементов, из которых составлена система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый распространенный вариант), или не присутствовать ни в одной (очень редко). Теоретически диаграммы могут содержать любые комбинации сущностей и отношений. На практике, однако, применяется сравнительно небольшое количество типовых комбинаций, соответствующих пяти наиболее употребительным видам, которые составляют архитектуру программной системы (см. следующий раздел).



**Рис.2.15** Реализации

Таким образом, в UML выделяют девять типов диаграмм:

* диаграммы классов;
* диаграммы объектов;
* диаграммы прецедентов;
* диаграммы последовательностей;
* диаграммы кооперации;

*8*



* диаграммы состояний;

*9*



* диаграммы действий;
* диаграммы компонентов;
* диаграммы развертывания.

## 1.1 Правила языка UML

Строительные блоки UML нельзя произвольно объединять друг с другом. Как и любой другой язык, UML характеризуется набором правил, определяющих, как должна выглядеть хорошо оформленная модель, то есть семантически самосогласованная и находящаяся в гармонии со всеми моделями, которые с нею связаны.

В языке UML имеются семантические правила, позволяющие корректно и однозначно определять:

* *имена*, которые можно давать сущностям, отношениям и диаграммам;
* *область действия* (контекст, в котором имя имеет некоторое значение);
* *видимость* (когда имена видимы и могут использоваться другими элементами);
* *целостность* (как элементы должны правильно и согласованно соотноситься друг с другом);
* *выполнение* (что значит выполнить или имитировать некоторую динамическую модель).

Модели, создаваемые в процессе разработки программных систем, эволюционируют со временем и могут неоднозначно рассматриваться

*2*



разными участниками проекта в разное время. По этой причине создаются не только хорошо оформленные модели, но и такие, которые:

* содержат скрытые элементы (ряд элементов не показывают, чтобы упростить восприятие);
* неполные (отдельные элементы пропущены);
* несогласованные (целостность модели не гарантируется).

## 2. Диаграммы классов

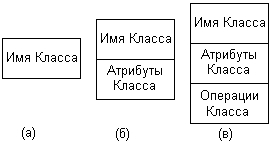
Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени. Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. 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 (рис.2.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

*10*



*2*



**Рис.2.1.** Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис.2.1, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис.2.1, б) и операциями (рис.2.1, в).

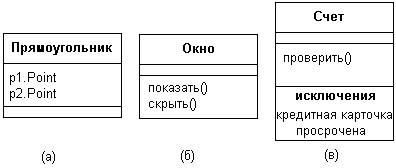
Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рис.2.2. В первом случае для класса "Прямоугольник" (рис.2.2, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис.2.2, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис.2.2, в) дополнительно изображена четвертая секция, в которой указано исключение - отказ от обработки просроченной кредитной карточки.

*11*



*2*



**Рис.2.2.** Примеры графического изображения классов на диаграмме

## 2.1 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.

*12*



В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

*13*



Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

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:

*14*



* Символ "+" обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
* Символ "#" обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.
* И, наконец, знак "-" обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается.

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,.., нuжняя\_гpaнuцak. верхняя\_границаk], где нижняя\_граница и верхняя\_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел, у которого нижняя (верхняя) граница равна значению нижняя\_граница (верхняя\_граница). В целом данное условное обозначение кратности соответствует теоретико-множественному объединению соответствующих интервалов.

*2*



Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. При этом придерживаются следующего правила: соответствующие нижние и верхние границы интервалов включаются в значение кратности. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается единственный знак "\*", то это означает, что кратность атрибута может быть произвольным положительным целым числом или нулем.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1.1, т.е. в точности 1.

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.

*15*



*2*



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.

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.

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.

Операция.

*16*



*17*



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 специального символа. Символ "+" обозначает операцию с областью видимости типа общедоступный (public). Символ "#" обозначает операцию с областью видимости типа защищенный (protected). И, наконец, символ "-" используется для обозначения операции с областью видимости типа закрытый (private).

Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private.

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

*18*



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>.

Здесь 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.

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

*2*



областью операции понимается объект класса. В этом случае имя и строка выражения типа операции не подчеркиваются.

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). При этом придерживаются следующей семантики для данных значений:

* последовательная (sequential) - для данной операции необходимо обеспечить ее единственное выполнение в системе, одновременное выполнение других операций может привести к ошибкам или нарушениям целостности объектов класса.
* параллельная (concurrent) - данная операция в силу своих особенностей может выполняться параллельно с другими операциями в системе, при этом параллельность должна поддерживаться на уровне реализации модели.
* охраняемая (guarded) - все обращения к данной операции должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса, при этом могут быть приняты дополнительные меры по контролю исключительных ситуаций на этапе ее выполнения.

С целью сокращения обозначений допускается использование одного имени в качестве строки-свойства для указания соответствующего значения параллельности. Отсутствие данной строки-свойства означает, что семантика параллельности для операции не определена. Поэтому следует предположить худший с точки зрения

*19*



*2*



производительности случай, когда данная операция требует последовательного выполнения.

Если для некоторой операции необходимо дополнительно указать особенности ее реализации (например, алгоритм), то это может быть сделано в форме примечания, записанного в виде текста, который присоединяется к записи операции в соответствующей секции класса. Если объекты класса принимают и реагируют на некоторый сигнал, то запись данной операции помечается ключевым словом "сигнал" ("signal"). Это обозначение равнозначно обозначению некоторой операции. Реакция объекта на прием сигнала может быть показана в виде некоторого автомата.

## 

## 2.2 Отношения между классами

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме 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)

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

*20*



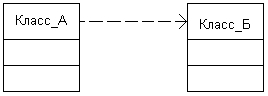
Отношение зависимости.

*21*



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.2.3). 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.



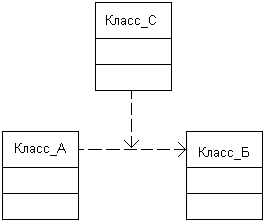
**Рис.2.3.** Графическое изображение отношения зависимости на диаграмме классов

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,

*2*



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.2.4).



**Рис.2.4.** 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;

*22*



*2*



* "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.

Отношение ассоциации

Отношение ассоциации соответствует наличию некоторого отношения между классами. Данное 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. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Наиболее простой случай данного отношения - бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. 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). Отсутствие

*2*

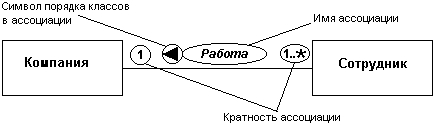


данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

*233*



В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами - классом "Компания" и классом "Сотрудник" (рис.2.5). Они связаны между собой бинарной ассоциацией Работа, имя которой указано на рисунке рядом с линией ассоциации. Для данного отношения определен порядок следования классов, первым из которых является класс "Сотрудник", а вторым - класс "Компания".



**Рис.2.5.** Графическое изображение отношения бинарной ассоциации между классами

Тернарная ассоциация и ассоциации более высокой арности в общем случае называются N-арной ассоциацией (читается - "эн арная ассоциация"). Такая ассоциация связывает некоторым отношением 3 и более классов, при этом один класс может участвовать в ассоциации более чем один раз. Класс ассоциации имеет определенную роль в соответствующем отношении, что может быть явно указано на диаграмме. Каждый экземпляр N-арной ассоциации представляет собой N-арный кортеж значений объектов из соответствующих классов. Бинарная ассоциация является частным случаем N-арной ассоциации, когда значение N=2, и имеет свое собственное обозначение.

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. В этом случае ромб соединяется с символами соответствующих классов сплошными линиями. Обычно линии проводятся от вершин ромба или от середины его сторон. Имя N-арной ассоциации записывается рядом с ромбом соответствующей ассоциации.

*24*



*2*



Порядок классов в N-арной ассоциации, в отличие от порядка множеств в отношении, на диаграмме не фиксируется. 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.

Одним из таких дополнительных обозначений является имя роли отдельного класса, входящего в ассоциацию. Имя роли представляет собой строку текста рядом с концом ассоциации для соответствующего класса. Она указывает специфическую роль, которую играет класс, являющийся концом рассматриваемой ассоциации. Имя роли не является обязательным элементом обозначений и может отсутствовать на диаграмме.

Отношение агрегации

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.

*25*



*2*



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.

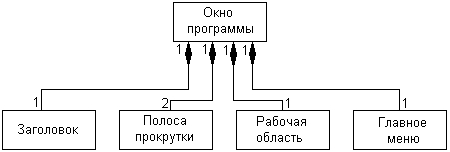
Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис.2.10).



**Рис.2.10.** 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.2.11).

*26*



*27*



**Рис.2.11.** 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.

Отношение обобщения.

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.2.12). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).

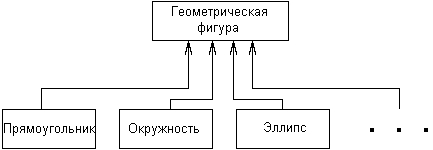


*2*



**Рис.2.12.** 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.2.13).



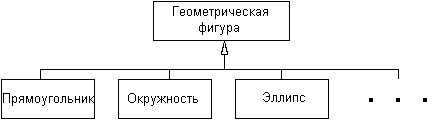
**Рис.2.13.** Пример графического изображения отношения обобщения классов

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.2.14).

*28*



*2*



**Рис.2.14.** 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.

В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

{complete} - означает,

*29*



*2*



* что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может. Пример - класс Клиент\_банка является предком для двух классов: Физическое\_лицо и Компания, и других классов-потомков он не имеет. На соответствующей диаграмме классов это можно указать явно, записав рядом с линией обобщения данную строку-ограничение;
* {disjoint} - означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов. В приведенном выше примере это условие также выполняется, поскольку предполагается, что никакое конкретное физическое лицо не может являться одновременно и конкретной компанией. В этом случае рядом с линией обобщения можно записать данную строку-ограничение;
* {incomplete} - означает случай, противоположный первому. А именно, предполагается, что на диаграмме указаны не все классы-потомки. В последующем возможно восполнить их перечень не изменяя уже построенную диаграмму. Пример - диаграмма класса "Автомобиль", для которой указание всех без исключения моделей автомобилей соизмеримо с созданием соответствующего каталога. С другой стороны, для отдельной задачи, такой как разработка системы продажи автомобилей конкретных моделей, в этом нет необходимости. Но указать неполноту структуры классов-потомков все же следует;
* {overlapping} - означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам. Пример - класс "Многоугольник" является классом-предком для класса "Прямоугольник" и класса "Ромб". Однако существует отдельный класс "Квадрат", экземпляры которого одновременно являются объектами первых двух классов. Вполне

*30*

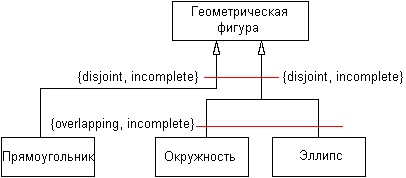


*2*



* естественно такую ситуацию указать явно с помощью данной строки-ограничения.

С учетом возможности использования строк-ограничений диаграмма классов (рис.2.14) может быть изображена без многоточий и без потери информации (рис.2.15).



**Рис.2.15.** Вариант графического изображения отношения обобщения классов с использованием строки-ограничения

## 

## 2.3 Интерфейсы

Интерфейсы являются элементами диаграммы вариантов использования и были рассмотрены в главе 4. 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.2.17). При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.



**Рис.2.17.** Пример графического изображения интерфейса на диаграмме классов

*31*



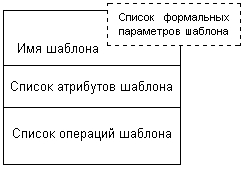
## 2.4 Шаблоны или параметризованные классы

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.

*2*



Графически шаблон изображается прямоугольником, к верхнему правому углу которого присоединен маленький прямоугольник из пунктирных линий (рис.2.19), большой прямоугольник может быть разделен на секции, аналогично обозначению для класса. В верхнем прямоугольнике указывается список формальных параметров для тех классов, которые могут быть получены на основе данного шаблона. В верхней секции шаблона записывается его имя по правилам записи имен для классов.



**Рис.2.19.** Графическое изображение шаблона на диаграмме классов

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. Очевидно, в этом случае между ними существует отношение зависимости с ключевым словом "bind", когда класс-клиент может использовать некоторый шаблон для своей последующей параметризации. В более частном случае между шаблоном и формируемым от него классом имеет место отношение обобщения с наследованием свойств шаблона (рис.2.20). 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".

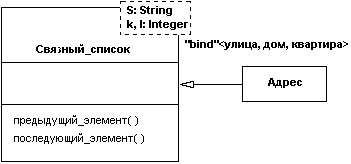
*32*



*2*



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, "ЬтсГ<координаты\_точки, х, у>. Концепция шаблонов является достаточно мощным средством в ООП, и поэтому ее использование в языке UML позволяет не только сократить размеры диаграмм, но и наиболее корректно управлять наследованием свойств и поведения отдельных элементов модели.



**Рис.2.20.** Пример использования шаблона на диаграмме классов

*33*



## 2.5 Рекомендации по построению диаграмм классов

**34**



Процесс разработки диаграммы классов занимает центральное место в ООАП сложных систем. От умения правильно выбрать классы и установить между ними взаимосвязи часто зависит не только успех процесса проектирования, но и производительность выполнения программы. Как показывает практика ООП, каждый программист в своей работе стремится в той или иной степени использовать уже накопленный личный опыт при разработке новых проектов. Это обусловлено желанием свести новую задачу к уже решенным, чтобы иметь возможность использовать не только проверенные фрагменты программного кода, но и отдельные компоненты в целом (библиотеки компонентов).

При определении классов, атрибутов и операций и задании их имен и типов перед отечественными разработчиками всегда встает невольный вопрос: какой из языков использовать в качестве естественного, русский или английский?

Отвечая на поставленный выше вопрос, следует отметить, что наиболее целесообразно придерживаться следующих рекомендаций. При построении диаграммы вариантов использования, являющейся наиболее общей концептуальной моделью проектируемой системы, применение русскоязычных терминов является не только оправданным с точки зрения описания структуры предметной области, но и эффективным с точки зрения коммуникативного взаимодействия с заказчиком и пользователями. При построении остальных типов диаграмм следует придерживаться разумного компромисса.

В частности, на начальных этапах разработки диаграмм целесообразность использования русскоязычных терминов вполне очевидна и оправдана. Однако, по мере готовности графической модели для реализации в виде программной системы и передачи ее для дальнейшей работы программистам, акцент может смещаться в сторону использования англоязычных терминов, которые в той или иной степени отражают особенности языка программирования, на котором предполагается реализация данной модели.

*2*



Более того, использование CASE-инструментариев для автоматизации ООАП, чаще всего, накладывает свои собственные требования на язык спецификации моделей. Именно по этой причине большинство примеров в литературе даются в англоязычном представлении, а при их переводе на русский может быть утрачена не только точность формулировок, но и семантика соответствующих понятий.

*35*

