УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные сведения.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 03.04.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 10
Диаграмма классов (class diagram)

Центральное место в ООАП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна всем, кто когда-либо имел опыт работы с CASE-инструментариями. Схожая нотация применяется и для объектов — экземпляров класса, с тем различием, что к имени класса добавляется имя объекта и вся надпись подчеркивается.

Нотация UML предоставляет широкие возможности для отображения дополнительной информации (абстрактные операции и классы, стереотипы, общие и частные методы, детализированные интерфейсы, параметризованные классы). При этом возможно использование графических изображений для ассоциаций и их специфических свойств, таких как отношение агрегации, когда составными частями класса могут выступать другие классы.

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. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. 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. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов. Декомпозиция некоторого представления на отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области. При этом компоненты диаграммы соответствуют элементам

статической семантической модели. Модель системы, в свою очередь, должна быть согласована с внутренней структурой классов, которая описывается на языке UML.

5.1. Класс

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. 5.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

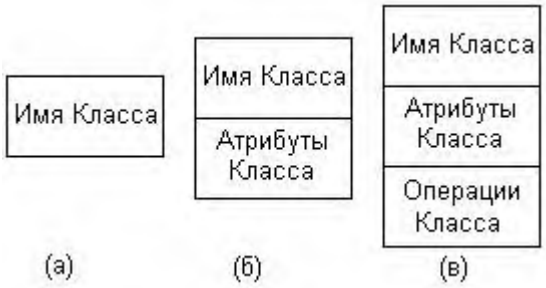

Рис. 1. Графическое изображение класса на диаграмме классов

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. 5.1, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 5.1, б) и операциями (рис. 5.1, в).

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.

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. 5.2. В первом случае для класса "Прямоугольник" (рис. 5.2, а) указаны только его атрибуты — точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 5.2, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 5.2, в) дополнительно изображена четвертая секция, в которой указано исключение — отказ от обработки просроченной кредитной карточки.

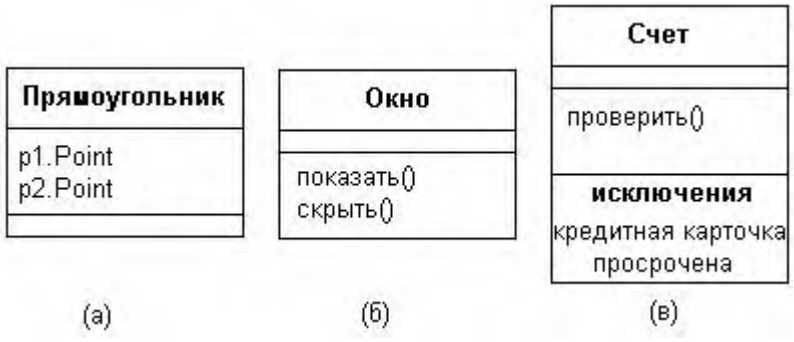

Рис.2. Примеры графического изображения классов на диаграмме

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. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП.

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.

Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Продавец", "Менеджер", "Офис" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

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.

Примечание

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель — двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий <Имя_пакета>::<Имя_класса>. Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

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). 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. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Примечание

Поскольку язык UML инвариантен относительно реализации своих конструкций в конкретных языках программирования, семантика отдельных кванторов видимости не является строго фиксированной. Значения этих кванторов должны дополнительно уточняться пояснительным текстом на естественном языке или соглашением по использованию соответствующих программно-зависимых синтаксических конструкций.

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_границаk .. 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. В качестве верхней_границы может использоваться специальный символ "*", который означает произвольное положительное целое число. Другими словами, это означает неограниченное сверху значение кратности соответствующего атрибута.

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.

В качестве примера рассмотрим следующие варианты задания кратности атрибутов.

- [0..1] означает, что кратность атрибута может принимать значение О или 1. При этом 0 означает отсутствие значения для данного атрибута.
- [0..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа — [*].
- [1.:*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.
- [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.
- [1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.
- [1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.
- [1..3,7..*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

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.

Можно привести следующие примеры задания имен и типов атрибутов классов:

- цвет: Color — здесь цвет является именем атрибута, Color — именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель (красный, зеленый, синий) для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета.
- имя_сотрудника [1..2] : String — здесь имя_сотрудника является именем атрибута, который служит для представления информации об имени, а возможно, и отчестве конкретного сотрудника. Тип атрибута String (Строка) как раз и указывает на тот факт, что отдельное значение имени представляет собой строку текста из одного или двух слов (например, "Кирилл" или "Дмитрий Иванович"). Поскольку во многих языках программирования существует тип данных String, использование соответствующего англоязычного термина не вызывает недоразумения у большинства программистов. Однако, хотя в языке UML все термины даются в англоязычном представлении, использование в качестве типа атрибута Строка в данной ситуации не исключается и определяется только соображениями удобства.
- видимость:Boolean — здесь видимость есть имя абстрактного атрибута (курсив здесь не случаен), 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). При этом значение истина может соответствовать наличию графического изображения на экране монитора, а значение ложь — его отсутствию, о чем дополнительно указывается в пояснительном тексте. Поскольку кратность атрибута видимость не указана, она принимает значение 1 по умолчанию. В этой ситуации англоязычное имя типа атрибута вполне оправдано наличием соответствующего базового типа в языках программирования. Абстрактный характер данного атрибута обозначается курсивным текстом в записи данного атрибута.
- форма:Многоугольник — здесь имя атрибута форма может характеризовать такой класс, который является геометрической фигурой на плоскости. В этом случае тип атрибута Многоугольник указывает на тот факт, что отдельная геометрическая фигура может иметь форму треугольника, прямоугольника, ромба, пятиугольника и любого другого многоугольника, но не окружности или эллипса. Вполне очевидно, что в данной ситуации использование соответствующего англоязычного термина вряд ли целесообразно, поскольку тип Многоугольник не является базовым для языков программирования.

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.

В качестве примеров исходных значений атрибутов можно привести следующие дополненные выше варианты задания атрибутов:

- цвет:Color = (255, 0, 0) — в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута.
- имя_сотрудника[1..2]:String = Иван Иванович — возможно, это нетипичный случай, который, скорее, соответствует ситуации имя_руководителя[2]:81пп§ = Иван Иванович.
- видимость:Boolean = истина — может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.
- форма:Многоугольник = прямоугольник — вряд ли требует комментариев, поскольку здесь речь идет о геометрической форме создаваемого объекта.

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, если некоторый атрибут задан в виде форма: Прямоугольник. то это будет означать, что все объекты данного класса могут иметь несколько различных форм, каждая из которых является прямоугольником. Другим примером может служить задание 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. С другой стороны, запись данного атрибута в виде заработная_плата: Currency = $500 означает уже нечто иное, а именно — при создании нового экземпляра Сотрудник (аналогия — прием на работу нового сотрудника) для него устанавливается по умолчанию заработная плата в $500. Однако для отдельных сотрудников могут быть сделаны исключения как в большую, так и в меньшую сторону, о чем необходимо позаботиться дополнительно в программе.

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 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.

Примечание

Применительно к конкретным языкам программирования могут быть определены дополнительные кванторы видимости. В этом случае подобные дополнения являются расширением базовой нотации и требуют соответствующих пояснений в форме текста на естественном языке или в виде строки-свойства.

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.

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.

Выражение типа возвращаемого значения также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания кратности возвращаемого значения данная спецификация может быть записана в виде списка отдельных выражений.

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. В этом случае имя и строка выражения типа операции не подчеркиваются.

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). 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 производительности случай, когда данная операция требует последовательного выполнения.

Появление сигнатуры операции на самом верхнем уровне объявляет эту операцию на весь класс, при этом данная операция наследуется всеми потомками данного класса. Если в некотором классе операция не выполняется (т. е. некоторый метод не применяется), то такая операция может быть помечена как абстрактная "{abstract}". Другой способ показать абстрактный характер операции — записать ее сигнатуру курсивом. Подчиненное появление записи данной операции без свойства {абстрактная} указывает на тот факт, что соответствующий класс-потомок может выполнять данную операцию в качестве своего "метода.

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. Кроме других случаев эта нотация может быть использована, чтобы показать реакцию объектов класса на ошибочные ситуации или исключения, которые могут моделироваться как сигналы или сообщения.

Поведение операции может быть указано дополнительно в форме присоединенного к операции примечания. В этом случае текст примечания заключается в скобки, если он представляет собой формальную спецификацию на некотором языке программирования и соответствует элементу "семантическое ограничение языка UML". В противном случае текст примечания является простым описанием на естественном языке и обозначается прямоугольником с "загнутым" верхним правым уголком (см. главу 4).

Список формальных параметров и тип возвращаемого значения могут не указываться. Квантор видимости атрибутов и операций может быть указан в виде специального значка или символа, которые используются для графического представления моделей в некотором инструментальном средстве. Имена операций, так же как и атрибутов, записываются со строчной (малой) буквы, а их типы — с заглавной (большой) буквы. При этом обязательной частью строки записи операции является наличие имени операции и круглых скобок.

В качестве примеров записи операций можно привести следующие обозначения отдельных 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 = (O, O, 255)) — может обозначать операцию по изображению на экране монитора прямоугольной области синего цвета, если не указываются другие значения в качестве аргументов данной операции.
- запросить_счет_клиента(номер_счета:1п1е§ег):Сиггепсу — обозначает операцию по установлению наличия средств на текущем счете клиента банка. При этом аргументом данной операции является номер счета клиента, который записывается в виде целого числа (например, "123456"). Результатом выполнения этой операции является некоторое число, записанное в принятом денежном формате (например, $1,500.00).
- 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.

5.2. 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.

Отношение зависимости

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. 5.3). 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.

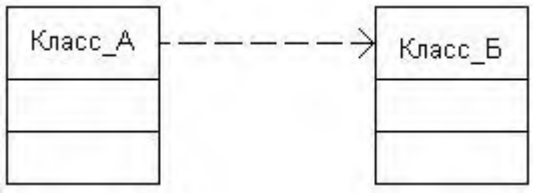

Рис. 3. Графическое изображение отношения зависимости на диаграмме классов

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, 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. 5.4).

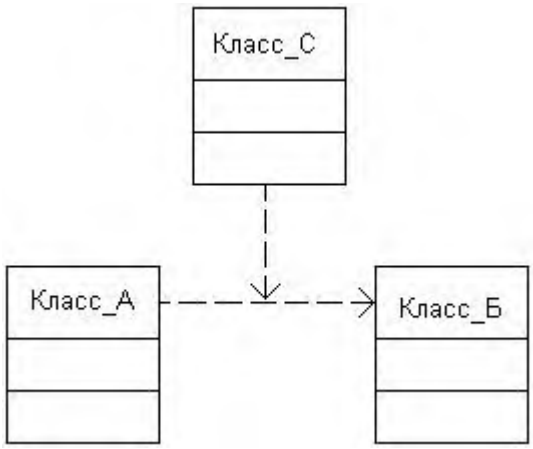

Рис. 4. Графическое представление зависимости между классом-клиентом (Класс_С) и классами-источниками (Класс_Л и Класс_Б)

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;
- "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.

Примечание

Отношение зависимости является наиболее общей формой отношения в языке UML. Все другие типы рассматриваемых отношений можно считать частным случаем данного отношения. Однако важность выделения специфических семантических свойств и дополнительных характеристик для других типов отношений обусловливают их самостоятельное рассмотрение при построении диаграмм.

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

В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами — классом "Компания" и классом "Сотрудник" (рис. 5.5). 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". Отдельным примером или экземпляром данного отношения может являться пара значений (Петров И. И., "Рога&Копыта"). Это означает, что сотрудник Петров И. И. работает в компании "Рога&Копыта".

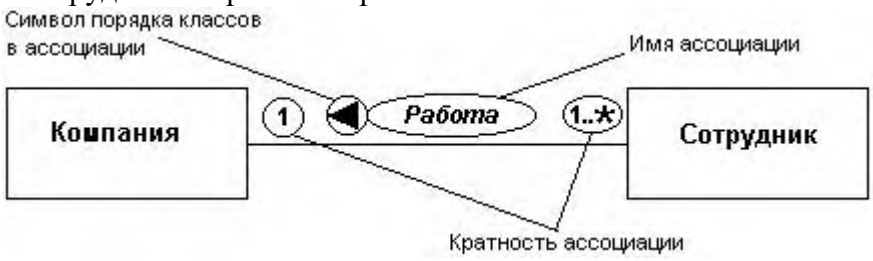

Рис. 5. Графическое изображение отношения бинарной ассоциации между классами

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.

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. Другими словами, такая ассоциация, в свою очередь, является классом с соответствующим обозначением в виде прямоугольника и является самостоятельным элементом языка UML — ассоциацией-классом (Association Class). N-арная ассоциация не может содержать символ агрегации ни для какой из своих ролей.

В качестве примера конкретной тернарной ассоциации рассмотрим отношение между тремя классами: "Футбольная команда", "Год" и "Игра". 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 (рис. 5.6).

Как уже упоминалось, отдельный класс ассоциации имеет собственную роль в отношении. Эта роль может быть изображена графически на диаграмме классов. С этой целью в языке UML вводится в рассмотрение специальный элемент — конец ассоциации (Association End), который графически соответствует точке соединения линии ассоциации с отдельным классом. Конец ассоциации является частью ассоциации, но не класса. Каждая ассоциация имеет два или больше концов ассоциации. Наиболее важные свойства ассоциации указываются на диаграмме рядом с этими элементами ассоциации и должны перемешаться вместе с ними.

Рис. 6. Графическое изображение тернарной ассоциации между тремя классами

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.

Следующий элемент обозначений — кратность отдельных классов, являющихся концами ассоциации. Кратность отдельного класса обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов. Интервал записывается рядом с концом ассоциации и для N-арной ассоциации означает потенциальное число отдельных экземпляров или значений кортежей этой ассоциации, которые могут иметь место, когда остальные N-1 экземпляров или значений классов фиксированы.

Так, для рассмотренного ранее примера (см. рис. 5.5) 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. Заметим, что вместо кратности "1..*" записать только символ "*" нельзя, поскольку последний означает кратность "0..*". Для данного примера это означало бы, что отдельные компании могут совсем не иметь сотрудников в своем штате. Но такая кратность вполне приемлема в других ситуациях, как это видно из рассмотренного выше примера (рис. 5.6).

Что касается других свойств отношения, ассоциации, то в случае их наличия, они могут рассматриваться в качестве атрибутов класса ассоциации и могут быть указаны на диаграмме обычным для класса способом в соответствующей секции прямоугольника класса.

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. На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации, рядом с которой записывается строка-ограничение "{xor}".

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 (рис. 5.7).

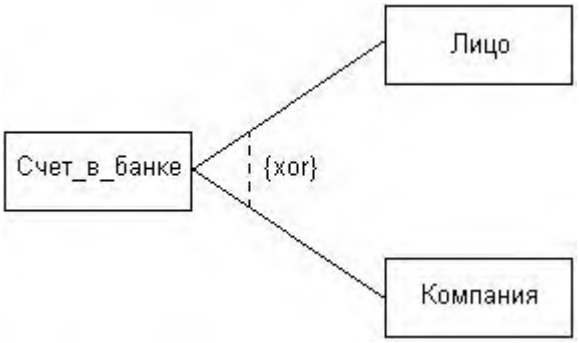

Рис. 7. Графическое изображение исключающей ассоциации между тремя классами

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму — отношение композиции. Поскольку эти отношения имеют свои специальные обозначения и относятся к базовым понятиям языка UML, рассмотрим их последовательно.

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. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Примечание

В связи с рассмотрением данного отношения вполне уместно вспомнить о специальном термине "агрегат", которое служит для обозначения технической системы, состоящей из взаимодействующих составных частей или подсистем. Эта аналогия не случайна и может служить для более наглядного понимания сути рассматриваемого отношения.

Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой некоторую иерархию ее компонентов, однако данная иерархия принципиально отличается от иерархии, порождаемой отношением обобщения. Отличие

заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются вполне самостоятельными сущностями. Более того, части целого обладают своими собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого.

В качестве примера отношения агрегации рассмотрим взаимосвязь типа "часть-целое", которая имеет место между сущностью "Грузовой автомобиль" и такими компонентами, как "Двигатель", "Шасси", "Кабина", "Кузов". Не претендуя на точное соответствие терминологии данной предметной области, нетрудно представить себе, что грузовой автомобиль состоит из двигателя, шасси, кабины и кузова. Именно это отношение между классом "Грузовой_автомобиль" и классами "Двигатель", "Шасси", "Кабина", "Кузов" описывает отношение агрегации.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его "частями" (рис. 5.8).

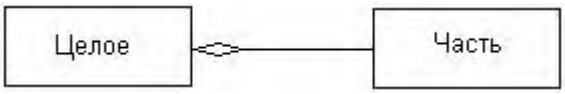

Рис. 8. Графическое изображение отношения агрегации в языке UML

Еще одним примером отношения агрегации может служить известное каждому из читателей деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь. Используя обозначения языка UML, компонентный состав ПК можно представить в виде соответствующей диаграммы классов (рис. 9), которая в данном случае иллюстрирует отношение агрегации.

Рис. 9. Диаграмма классов для иллюстрации отношения агрегации на примере ПК

Отношение композиции

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

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

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис. 5.10).

Рис. 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. 5.11).

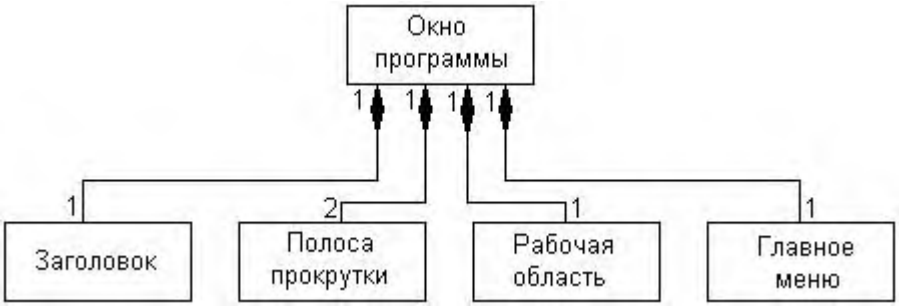

Рис. 11. Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

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

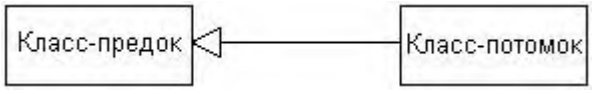

Рис. 5.12. Графическое изображение отношения обобщения в языке UML

Рекомендации по построению диаграмм классов

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).

Такой стереотипный подход позволяет существенно сократить сроки реализации проекта, однако приемлем лишь в том случае, когда новый проект концептуально и технологически не слишком отличается от предыдущих. В противном случае платой за сокращение сроков проекта может стать его реализация на устаревшей технологической базе. Что касается собственно объектной структуризации предметной области, то здесь уместно придерживаться тех рекомендаций, которые накоплены в ООП. Они широко освещены в литературе [1, 2, 4, 10, 13, 18, 20] и поэтому здесь не рассматриваются.

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? С одной стороны, использование родного языка для описания модели является наиболее естественным способом ее представления и в наибольшей степени отражает коммуникативную функцию модели системы. С другой стороны, разработка модели является лишь одним из этапов разработки соответствующей системы, а применение инструментальных средств для ее реализации в абсолютном большинстве случаев требует использования англоязычных терминов. Именно поэтому возникает характерная неоднозначность, с которой, по-видимому, совершенно незнакома англоязычная аудитория.

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.

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.

После разработки диаграммы классов процесс ООАП может быть продолжен в двух направлениях. С одной стороны, если поведение системы тривиально, то можно приступить к разработке диаграмм кооперации и компонентов. Однако для сложных динамических систем поведение представляет важнейший аспект их функционирования. Детализация поведения осуществляется последовательно при разработке диаграмм состояний, последовательности и деятельности.