23. Диаграмма классов (class diagram). Класс. Отношения между классами. Интерфейсы. Объекты

Диаграмма классов (англ. *Static Structure diagram*) — диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Диаграмма классов (Class diagram) — статическая структурная диаграмма, описывающая структуру системы, демонстрирующая классы системы, их атрибуты, методы и зависимости между классами.

Существует два вида:

- Статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
- Аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

- Концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
- Точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
- Точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Класс

Класс - это абстрактное понятие, сравнимое с понятием *категория* в его обычном смысле.

Класс в ООП - это абстрактный тип данных, который включает в себя не только данные, но и функции и процедуры.

Суть отличия классов от других абстрактных типов данных состоит в том, что при задании типа данных класс определяет одновременно и интерфейс, и реализацию для всех своих экземпляров, а вызов метода-конструктора обязателен.

***В UML**

Класс на диаграмме показывается в виде прямоугольника, разделенного на 3 области. В верхней содержится название класса, в средней – описание атрибутов (свойств), в нижней – названия операций – услуг, предоставляемых объектами этого класса.

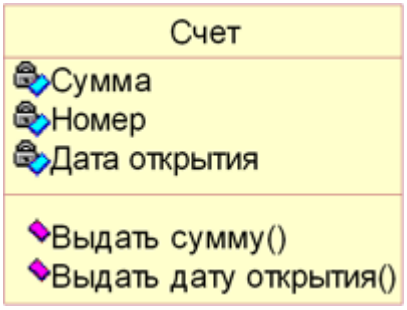


рис.1

Для каждого атрибута класса можно задать видимость (visibility). Эта характеристика показывает, доступен ли атрибут для других классов. В UML определены следующие уровни видимости атрибутов:

- Открытый (public) – атрибут виден для любого другого класса (объекта);
- Защищенный (protected) – атрибут виден для потомков данного класса;
- Закрытый (private) – атрибут не виден внешними классами (объектами) и может использоваться только объектом, его содержащим.

Класс содержит объявления **операций**, представляющих собой определения запросов, которые должны выполнять объекты данного класса. Каждая операция имеет **сигнатуру**, содержащую имя операции, тип возвращаемого значения и список параметров, который может быть пустым. Реализация операции в виде процедуры – это метод, принадлежащий классу. Для операций, как и для атрибутов класса, определено понятие «видимость». Закрытые операции являются внутренними для объектов класса и недоступны из других объектов. Остальные образуют интерфейсную часть класса и являются средством интеграции класса в программную систему.

Отношения между классами

- Наследование (Генерализация) — объекты дочернего класса наследуют все свойства родительского класса.
- Ассоциация — объекты классов вступают во взаимодействие между собой.
- Агрегация — объекты одного класса входят в объекты другого.
- Композиция — объекты одного класса входят в объекты другого и зависят друг от друга по времени жизни.
- Класс-Метакласс — отношение, при котором экземплярами одного класса являются другие классы.

***В UML**

На диаграммах классов обычно показываются ассоциации и обобщения

Каждая **ассоциация** несет информацию о связях между объектами внутри ПС. Наиболее часто используются бинарные ассоциации, связывающие два класса. Ассоциация может иметь название, которое должно выражать суть отображаемой связи (см. рис. 2). Помимо названия, ассоциация может иметь такую характеристику, как **множественность**. Она показывает, сколько объектов каждого класса может участвовать в ассоциации. Множественность указывается у каждого конца ассоциации (полюса) и задается конкретным числом или диапазоном чисел. Множественность, указанная в виде звездочки, предполагает любое количество (в том числе, и ноль).

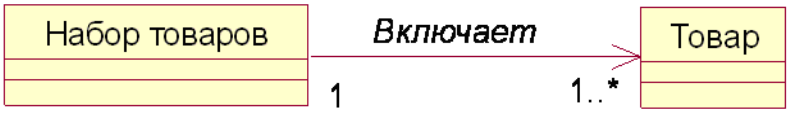


рис.2

Обобщение на диаграммах классов используется, чтобы показать связь между классом-родителем и классом-потомком. Оно вводится на диаграмму, когда возникает разновидность какого-либо класс, а также в тех случаях, когда в системе обнаруживаются несколько классов, обладающих сходным поведением (в этом случае общие элементы поведения выносятся на более высокий уровень, образуя класс-родитель).

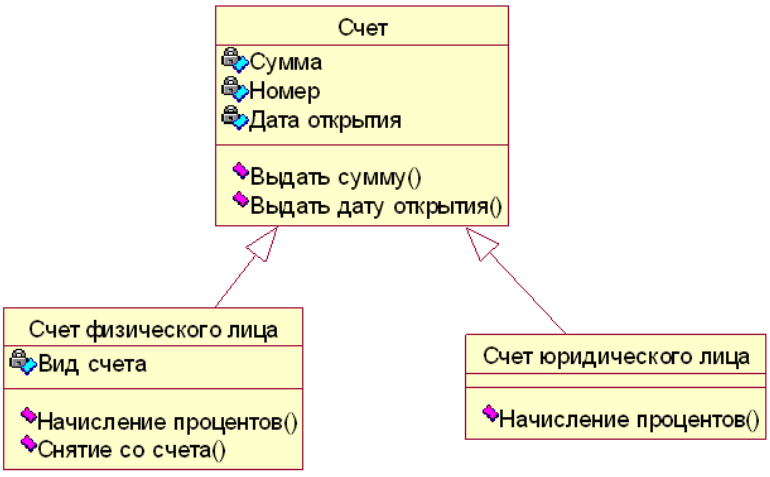


рис.3

UML позволяет строить модели с различным уровнем детализации. На рис.4 показана детализация модели, представленной на рис.2.

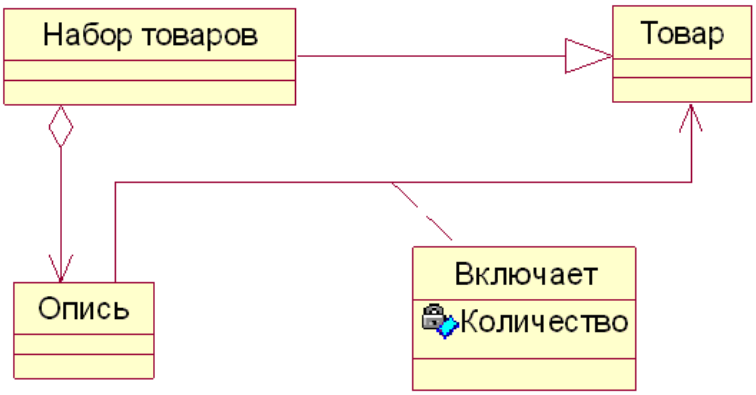


рис.4

Обобщение показывает, что набор товаров – это тоже товар, который может быть предметом заказа, продажи, поставки и т.д. Набор включает опись, в которой указывается, какие товары входят в набор, а класс-ассоциация «включает» определяет количество каждого вида товаров в наборе.

Стереотип класса – это элемент расширения словаря UML, который обозначает отличительные особенности в использовании класса. Стереотип имеет название, которое задается в виде текстовой строки. При изображении класса на диаграмме стереотип показывается в верхней части класса в двойных угловых скобках. Есть четыре стандартных стереотипа классов, для которых предусмотрены специальные графические изображения (см. рис.5).

Стереотип используется для обозначения классов-сущностей (классов данных), стреотип описывает пограничные классы, которые являются посредниками между ПС и внешними по отношению к ней сущностями – актерами, обозначаемыми стереотипом <>. Наконец, стереотип описывает классы и объекты, которые управляют взаимодействиями. Применение стереотипов позволяет, в частности, изменить вид диаграмм классов.

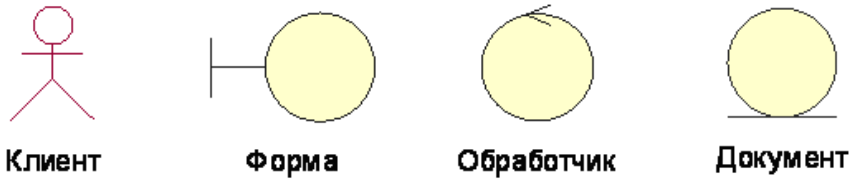


рис.5

Интерфейс

В программировании существует понятие программного интерфейса, означающего перечень возможных вычислений, которые может выполнить та или иная часть программы, включая описание того, какие аргументы и в каком порядке требуется передавать на вход алгоритмам из этого перечня, а также что и в каком виде они будут возвращать. Абстрактный тип данных интерфейс придуман для формализованного описания такого перечня. Сами алгоритмы, то есть действительный программный код, который будет выполнять все эти вычисления, интерфейсом *не* задаётся, программируется отдельно и называется *реализацией интерфейса*.

Описание ООП-интерфейса, если отвлечься от деталей синтаксиса конкретных языков, состоит из двух частей: *имени* и *методов* интерфейса.

- *Имя интерфейса* строится по тем же правилам, что и другие идентификаторы используемого языка программирования. Разные языки и среды разработки имеют различные соглашения по оформлению кода, в соответствии с которыми имена интерфейсов могут формироваться по некоторым правилам, которые помогают отличать имя интерфейса от имён других элементов программы. Например, в технологии COM и во всех поддерживающих её языках действует соглашение, следуя которому, имя интерфейса строится по шаблону «`I<Имя>`», то есть состоит из написанного с заглавной буквы осмысленного имени, которому предшествует прописная латинская буква `I` (`IUnknown`, `IDispatch`, `IStringList` и т. п.).
- *Методы интерфейса*. В описании интерфейса определяются имена и сигнатуры входящих в него методов, то есть процедур или функций класса.

Использование интерфейсов возможно двумя способами:

- Класс может *реализовывать* интерфейс. Реализация интерфейса заключается в том, что в описании класса данный интерфейс указывается как реализуемый, а в коде класса обязательно определяются все методы, которые описаны в интерфейсе, в полном соответствии с сигнатурами из описания этого интерфейса. То есть, если класс реализует интерфейс, для любого экземпляра этого класса существуют и могут быть вызваны все описанные в интерфейсе методы. Один класс может реализовать несколько интерфейсов одновременно.

- Возможно объявление переменных и параметров методов как имеющих тип-интерфейс. В такую переменную или параметр может быть записан экземпляр любого класса, реализующего интерфейс. Если интерфейс объявлен как тип возвращаемого значения функции, это означает, что функция возвращает объект класса, реализующего данный интерфейс.

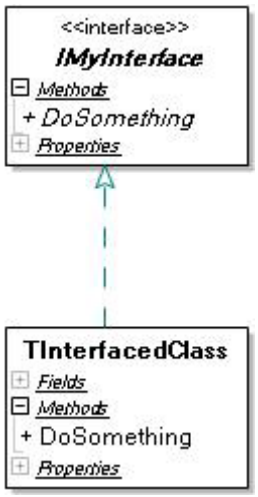
Как правило, в объектно-ориентированных языках программирования интерфейсы, как и классы, могут наследоваться друг от друга. В этом случае интерфейс-потомок включает все методы интерфейса-предка и, возможно, добавляет к ним свои собственные.

Таким образом, с одной стороны, интерфейс — это «договор», который обязуется выполнить класс, *реализующий* его, с другой стороны, интерфейс — это тип данных, потому что его описание достаточно чётко определяет свойства объектов, чтобы наравне с классом типизировать переменные. Следует, однако, подчеркнуть, что интерфейс не является полноценным типом данных, так как он задаёт только внешнее поведение объектов. Внутреннюю структуру и реализацию заданного интерфейсом поведения обеспечивает класс, реализующий интерфейс; именно поэтому «экземпляров интерфейса» в чистом виде не бывает, и любая переменная типа «интерфейс» содержит экземпляры конкретных классов.

Использование интерфейсов — один из вариантов обеспечения полиморфизма в объектных языках и средах. Все классы, реализующие один и тот же интерфейс, с точки зрения определяемого ими поведения, ведут себя внешне одинаково. Это позволяет писать обобщённые алгоритмы обработки данных, использующие в качестве типов параметры интерфейсов, и применять их к объектам различных типов, всякий раз получая требуемый результат.

Интерфейсы в UML используются для визуализации, специфицирования, конструирования и документирования стыковочных UML-узлов между составными частями системы. Типы и UML-роли обеспечивают механизм моделирования статического и динамического соответствия абстракции интерфейсу в конкретном контексте.

В UML интерфейсы изображаются как классы со стереотипом «interface», либо в виде кружочков (в этом случае содержащиеся в интерфейсе UML-операции не отображаются).



Объект в программировании — некоторая сущность в виртуальном пространстве, обладающая определённым состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними (методов)[1]. Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «**экземпляр класса**» и «объект» взаимозаменяемы

Объекты обладают свойствами наследования, инкапсуляции и полиморфизма

***В UML**

Диаграмма объектов (Object diagram) — демонстрирует полный или частичный снимок моделируемой системы в заданный момент времени. На диаграмме объектов отображаются экземпляры классов (объекты) системы с указанием текущих значений их атрибутов и связей между объектами.

Графически диаграмму объектов представляют в виде графа, состоящего из вершин и ребер.

При моделировании статического вида системы с точки зрения проектирования или процессов объектные диаграммы применяют для того, чтобы моделировать структуру объектов.

Моделирование структуры объектов предполагает получение "снимка" объектов системы в некоторый момент времени. Диаграммы объектов предоставляют статическую основу динамического сценария, описываемого диаграммой взаимодействия. Они применяются для визуализации, специфицирования конструирования и документирования определенных экземпляров в системе, а также отношений между этими экземплярами.