Курганский государственный университет
Кафедра программного обеспечения автоматизированных систем

К У Р С Л Е К Ц И Й

по дисциплине

ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

для студентов направления 231000.62
«Программная инженерия»

Лекция 3.4. Разработка логической модели программной системы.

UML-диаграммы классов

План лекции

1. Классы
    1.1. Имя класса
    1.2. Атрибуты класса
    1.3. Операции (методы) класса
2. Отношения между классами
    2.1. Отношение ассоциации
    2.2. Отношение обобщения
    2.3. Отношение агрегации
    2.4. Отношение композиции
    2.5. Отношение зависимости
3. Интерфейсы
4. Шаблоны
5. Пример диаграммы классов

Заключение

Контрольные вопросы и задания

Диаграмма классов

Диаграмма классов (class diagram) используется на стадии разработки логической модели программной системы и служит для представления её статической структуры в терминологии классов объектно-ориентированного программирования.

Диаграмма классов является дальнейшим развитием диаграммы вариантов использования и формируется в результате классификации вариантов использования.

Диаграмма классов описывает внутреннюю структуру сущностей предметной области (объектов и подсистем), а также отражает взаимосвязи (типы отношений) между ними.

Диаграмма классов не содержит информации о временных аспектах функционирования системы и является графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят от времени.

В общем случае статическая структурная модель системы может быть представлена несколькими диаграммами классов, объединенными в некоторый пакет: декомпозиция модели на отдельные диаграммы выполняется для удобства графической визуализации структурных взаимосвязей предметной области.

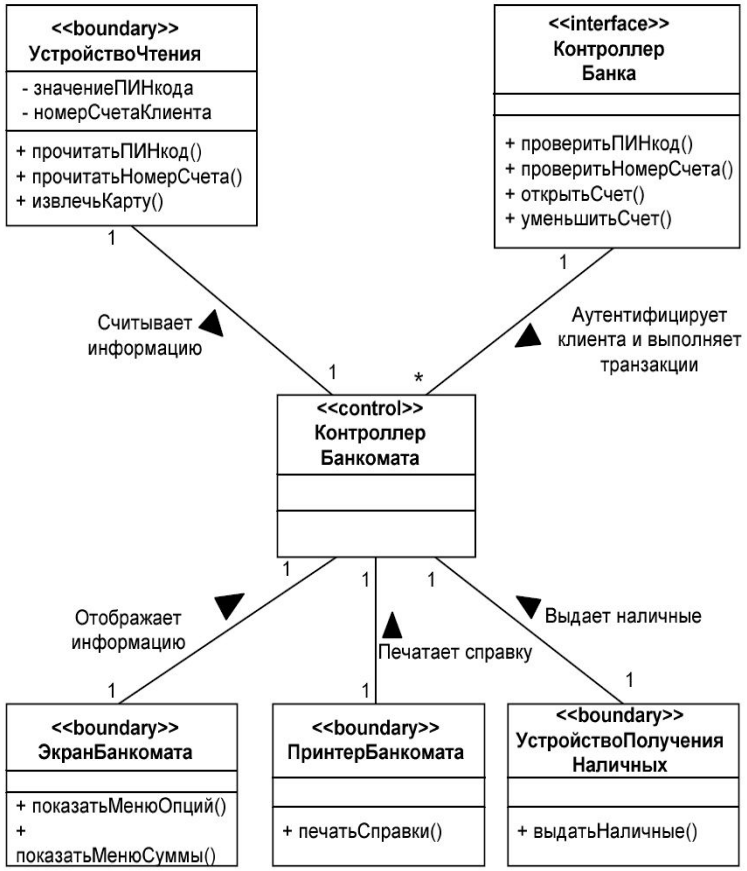
Основные компоненты диаграммы классов

Основными компонентами диаграммы классов являются "***класс***" и "***отношение***".

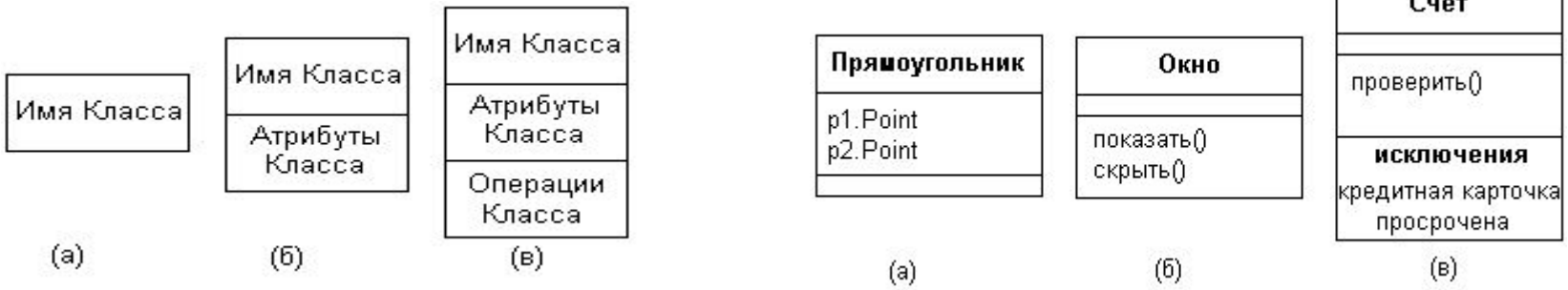
UML-диаграмма классов представляется в виде графа:

Вершины графа - это элементы типа «*класс*»,

Дуги графа - структурные *отношения* различных типов, установленные между классами.



Классы



Класс (class) в языке UML служит для описания и представления множества объектов (сущностей) предметной области, которые обладают одинаковой структурой, поведением и отношениями с объектами других классов.

Графически класс изображается в виде прямоугольника, разделенного на секции, в которых указываются имя класса, его атрибуты (переменные) и операции (методы).

Обязательным элементом обозначения класса является только его имя. На начальных этапах разработки диаграммы классы могут обозначаться простым прямоугольником с указанием только имени класса, а по мере детализации диаграммы описания классов дополняются секциями атрибутов и операций.

Кроме собственно имени класса в первой секции его обозначения может находиться дополнительная информация: ссылки на стандартные шаблоны или абстрактные классы, информация о разработчике данного класса; статус состояния разработки, а также и другие общие свойства этого класса.

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*.

Атрибуты класса

Атрибуты (свойства) класса записываются во второй (сверху) секции в соответствии со следующими синтаксическими правилами:

Каждый атрибут записывается в отдельной строке. Если строка атрибута подчеркнута, это означает, что соответствующий атрибут является *общим*, то есть значение этого атрибута будет одинаковым для всех объектов данного класса.

Строка описания атрибута состоит из следующих параметров атрибута (из которых обязательным является только один параметр – ***имя*** атрибута):

- *Квантор видимости атрибута*
- ***Имя атрибута***
- *Кратность атрибута*
- *Тип атрибута*
- *Исходное значение*
- *Строка-свойство*

Атрибуты класса

Квантор видимости атрибута – обозначается одним из четырех символов*:*

- *"**+**"* (*public*) – обозначает *общедоступный* атрибут, то есть видимый из любого класса пакета, в котором определена диаграмма;
- *"**#**"* (*protected*) – обозначает *защищенный* атрибут, область видимости которого ограничена только данным классом и его подклассами;
- "**-**" (*private*) – обозначает *закрытый* атрибут, область видимости которого ограничена только данным классом.
- "**~**" (*package*) – обозначает *пакетный* атрибут, доступен только для классов того пакета, в котором определен класс – владелец атрибута.

Если квантор видимости атрибута опущен, то это означает только то, что видимость атрибута не указывается, и никакое значение этого параметра по умолчанию не устанавливается.

Атрибуты класса

Имя атрибута – текстовая строка, используемая в качестве идентификатора атрибута; должна быть уникальной в пределах данного класса.

Кратность атрибута – характеризует общее количество экземпляров атрибута данного типа, входящих в состав класса. Кратность записывается в квадратных скобках непосредственно после имени атрибута в виде строки из цифр, разделенных запятыми или многоточием, как это показано в следующих примерах:

- [3] – кратность атрибута строго равна числу 3;
- [1, 3, 12, 22] – одно из указанных в скобках целых чисел;
- [1 .. 5, 7, 9 .. 12] – любое целое число из диапазонов от 1 до 5 и от 9 до 12 (включая границы диапазонов), а также число 7;
- [7 .. *] – любое целое число, большее или равное 7;
- [*] – любое целое неотрицательное число.

Атрибуты класса

Тип атрибута - в простейшем случае указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс (например, String, Boolean или Color). Перед описателем типа атрибута ставится знак "**:**" (двоеточие).

Исходное значение – определяет начальное значение атрибута, которое он получит (по умолчанию) в момент создания экземпляра класса. Исходное значение должно соответствовать заданному типу этого атрибута. Перед описателем исходного значения атрибута ставится знак равенства "**=**".

Если исходное значение атрибута опущено, то на момент создания экземпляра класса значение атрибута будет неопределенным.

{*Строка-свойство*}, заключенная в фигурные скобки, служит для указания значений атрибута, которые фиксируются для всех экземпляров данного класса: это значение атрибут получит в момент создания экземпляра класса, и оно не может быть переопределено в программе при работе с этим объектом.

Примеры описателей атрибутов класса

Атрибуты класса "**Геометрические_Объекты**":

- **цвет:Color=(255,0,0)** – имя атрибута "**цвет**", тип атрибута "**Color**", начальное значение (**255,0,0**). В RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута. Факт подчеркивания строки атрибута указывает на то, что все экземпляры этого класса будут иметь такое же значение данного атрибута (т.е. все создаваемые геометрические объекты будут красного цвета).
- **форма:Многоугольник=прямоугольник** – имя атрибута "**форма**", тип атрибута "**Многоугольник**", начальное значение "**прямоугольник".** Каждый вновь создаваемый экземпляр этого класса будет иметь форму прямоугольника.

Атрибуты класса "**Сотрудники**":

- **имя_сотрудника[1..2]:String="Новый_Сотрудник"** – при создании нового экземпляра класса (то есть при приеме на работу нового сотрудника) атрибут **имя_сотрудника** этого класса (имеющий тип **String** и кратность 1 или 2) получит начальное значение "**Новый_Сотрудник** ".
- **заработная_плата:Money=$500 –** каждому вновь принятому на работу сотруднику назначается 500-долларовая зарплата (размер которой в будущем может быть изменен).
- **заработная_плата:Money={$500}** – строка-свойство, заключенная в фигурные скобки, может означать фиксированную заработную плату для всех экземпляров класса.

Операции (методы) класса

Операция (*operation*), или ***метод*** (*method*) класса – это сервис, предоставляемый каждым экземпляром (объектом) этого класса по требованию его клиентов – других объектов, в том числе и объектов этого же класса.

Совокупность всех операций класса характеризует функциональный аспект его поведения.

Спецификации операций класса записываются в третьей сверху секции графического изображения класса по стандартизованным синтаксическим правилам:

- Спецификация каждой операции класса записывается в отдельной строке.
- Если область действия операции распространяется на все объекты класса, спецификация операции подчеркивается; если строка не подчеркнута, областью действия операции является объект класса.
- Спецификация операции, заданная у класса верхнего уровня ("предка"), по умолчанию наследуется всеми потомками данного класса.
- Если в некотором классе-потомке данная операция не выполняется, то *спецификация этой операции в данном классе должна быть записана курсивом* (что указывает на абстрактный характер этой операции для данного класса).
- Существует и другой способ показать абстрактный характер операции – указать в *строке-свойстве* этой операции значение **{abstract}**.

Спецификация операций класса

Строка операции имеет стандартный формат:

<квантор_видимости><имя_операции>(**список_параметров**):**<тип_возвращаемого_значения>**{**строка-свойство**},

где:

Квантор видимости, как и в случае с атрибутами класса, может принимать одно из четырех возможных значений и обозначается соответствующими символами:

"+" – *public*, "#" – *protected*, "-" – *private*, "**~**" - *package*.

Имя операции – это идентифицирующая операцию строка текста, которая должна быть уникальной в пределах данного класса. Имя операции является единственным обязательным элементом обозначения операции.

Имя операции должно начинаться со строчной (малой) буквы и не должно содержать пробелов.

Пара скобок после имени операции является обязательным элементом спецификации (даже при "пустом" списке параметров).

Спецификация операций класса

Список параметров - это заключенный в скобки перечень формальных параметров, каждый из которых может быть представлен в следующем формате: **<вид параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>, где:**

- **Вид параметра** – одно из ключевых слов: **in, out** или **inout** (по умолчанию - **in**).
- **Имя параметра** - идентификатор соответствующего формального параметра, записывается с прописной (большой) буквы.
- **Выражение типа** - спецификация типа значения параметра, зависимая от языка программирования.
- **Значение по умолчанию** - выражение для значения формального параметра, синтаксис которого зависит от языка программирования.

Спецификация операций класса

Выражение типа возвращаемого значения – указывает на тип данных значения, возвращаемого объектом после выполнения операции. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка выражений.

Строка-свойство заключается в фигурные скобки и служит для указания значений свойств, которые могут быть применены к данному элементу, например:

- Пассивная операция, которая не изменяет состояния системы, обозначается строкой-свойством **{query}** (запрос). В противном случае операция может изменять состояние системы, хотя нет никаких гарантий, что она будет это делать.
- Для указания возможности параллельного выполнения операции используются строки-свойства следующих видов:
  - **{sequential}** – *последовательная* операция, при реализации которой необходимо обеспечить ее единственное выполнение в системе;
  - **{concurrent}** – *параллельная* операция, допускающая одновременное выполнение с другими операциями;
  - **{guarded}** – *охраняемая* операция, все обращения к которой должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса.

Примеры спецификации операций классов

Пример 1.

+нарисовать(форма: Многоугольник = прямоугольник, цвет_заливки: Color = (O, O, 255)

Обозначена общедоступная (+) операция "**нарисовать**" с двумя входными (**in** – по умолчанию) параметрами: "**форма**" типа "**Многоугольник**" с начальным значением "**прямоугольник**", и "**цвет заливки**" типа **Color** с начальным значением**(O,O,255)**.

При выполнении этой операции на экране монитора будет изображена прямоугольная область синего цвета.

Пример 2.

-запросить_счет_клиента(номер_счета:integer):Сиггепсу

Обозначена закрытая (-) операция по установлению наличия средств на текущем счете клиента банка. Входным параметром данной операции является номер счета клиента, который записывается в виде целого числа.

Результатом выполнения операции является некоторое число, записанное в принятом денежном формате:

Отношения между классами

Кроме внутренней структуры классов на диаграмме указываются *отношения* между классами, отражающие взаимосвязи между их экземплярами (объектами).

В языке UML определены *5 базовых типов отношений* между классами, для каждого из которых предусмотрено соответствующее графическое обозначение:

1. отношение *ассоциации* (association relationship),
2. отношение *обобщения* (generalization relationship),
3. отношение *агрегации* (aggregation relationship),
4. отношение *композиции* (composition relationship)
5. отношение *зависимости* (dependency relationship)

Отношение ассоциации

Отношение ассоциации – это наиболее общий случай взаимосвязи между классами.

В определенном смысле все остальные типы отношений являются частными случаями отношения ассоциации, выделенными в отдельные категории ввиду их важности с точки зрения описания структуры моделируемой системы.

Классы, связанные отношением ассоциации, называются "*ролями ассоциации*".

Важнейшая характеристика ассоциации – её *арность*, определяемая количеством классов, участвующих в ассоциации. Простейшая ассоциация – бинарная, в которой участвуют ровно два класса. Если в ассоциации участвуют три класса, она называется *тернарной*, если больше – *тетрарной*, *пентарной* и т.д., в общем случае – "***n***-арной" ассоциацией.

Фактически в ассоциации участвуют объекты - *экземпляры* ассоциированных классов, поэтому допустимой является и *ассоциация*, описывающая взаимосвязь между различными объектами одного и того же класса.

Каждый экземпляр ***n***-арной ассоциации представляет собой ***n***-арный кортеж значений объектов из соответствующих классов, при этом один и тот же объект (экземпляр класса) может участвовать в нескольких экземплярах одной и той же ассоциации.

Обозначение отношения ассоциации

Отношение ассоциации обозначается на диаграмме сплошной линией (со стрелками или без них), соединяющей ассоциированные классы.

Линия ассоциации может быть поименована и помеченная специальными символами, в её разрыве может изображаться ромб, концы ассоциации могут быть помечены именами ролей и/или кратностью ассоциированных классов.

Если имя ассоциации задано, оно записывается с заглавной буквы.

Ромб в разрыве линии бинарной ассоциации, как правило, не изображается.

Кратность отношения ассоциации

Концы ассоциации могут быть помечены параметрами *кратности,* указывающими на количество объектов (экземпляров классов), участвующих в одном экземпляре ассоциации.

В примере, показанном на предыдущем слайде, в одной компании могут работать один и более ("**1..***") сотрудников, а один ("**1**") сотрудник не может работать более, чем в одной компании (кратность "1" на левом конце ассоциации можно было бы и не указывать - это значение параметр кратности получает по умолчанию).

Заметим, что значение "*" параметра кратности правого конца ассоциации было бы ошибочным, так как "*" – это краткая форма записи диапазона значений "**0..***", а компания, в штате которой нет ни одного сотрудника, вряд ли имеет право на существование.

Отношение обобщения

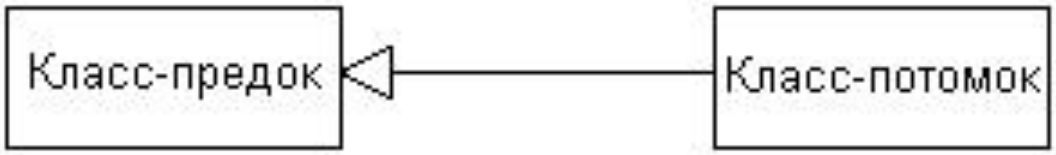
Отношение обобщения устанавливается между более общим элементом (предком) и более частным или специальным элементом (потомком).

Применительно к диаграмме классов отношение обобщения описывает иерархическое строение классов и наследование их свойств и поведения.

При этом предполагается, что класс-потомок (подкласс) обладает всеми свойствами и поведением класса-предка (суперкласса), но также может иметь и свои собственные свойства и поведение, которые отсутствуют у класса-предка.

На диаграммах отношение обобщения обозначается сплошной линией с треугольной незакрашенной внутри стрелкой, указывающей на класс-предок

Класс-предок может иметь несколько связанных с ним классов-потомков: в этом случае множество классов, связанных отношением обобщения, образуют древовидную структуру, корнем которой является класс-предок.



Ограничения отношения обобщения

Рядом со стрелкой обобщения на диаграмме может быть помещен текст, указывающий на дополнительные свойства этого отношения, касающиеся всех подклассов данного отношения.

Если текст содержит ключевое слово, записанное в фигурных скобках, его следует рассматривать как ***ограничение***:

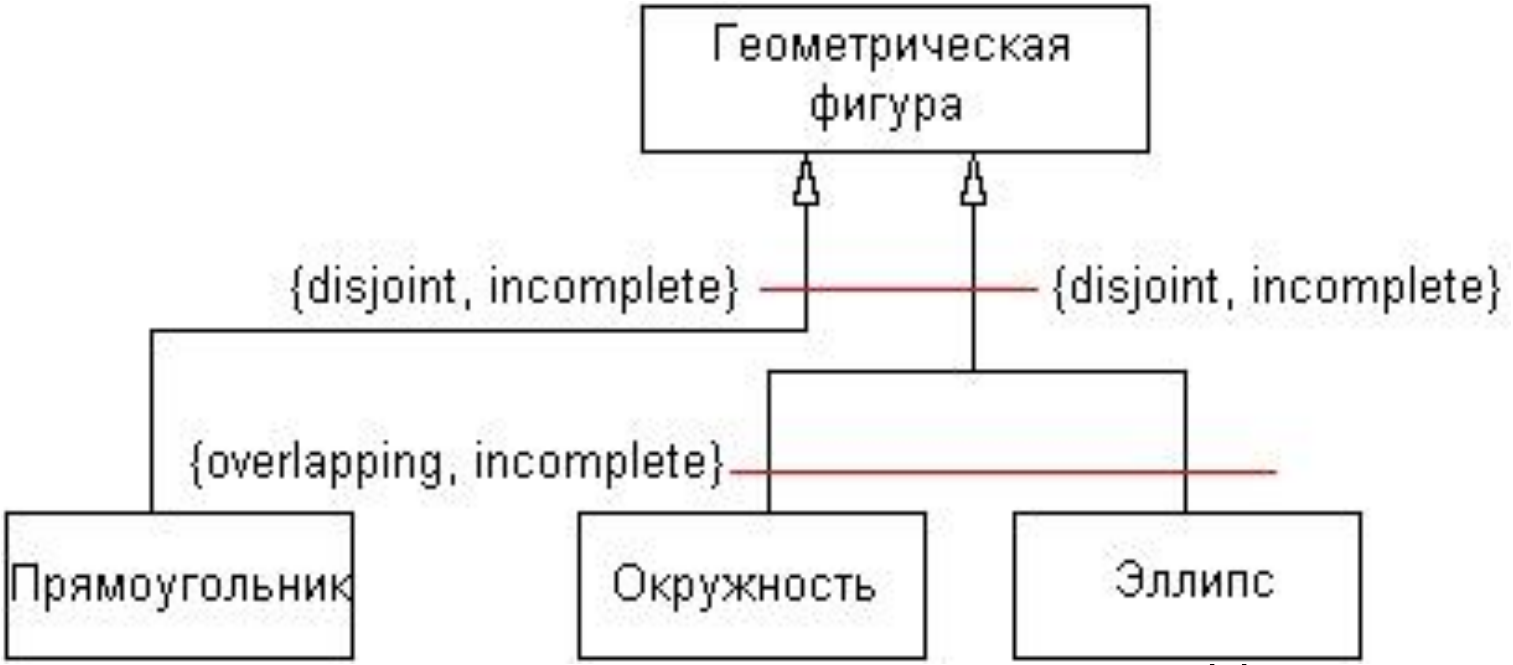
- {**complete**} – означает, что в данном отношении обобщения специфицированы все без исключения классы-потомки, и других классов-потомков данный класс-предок не имеет.
- {**incomplete**} – означает случай, противоположный первому: на диаграмме указаны не все классы-потомки, и впоследствии можно пополнить их перечень, не изменяя уже построенную диаграмму.
- {**disjoint**} – означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов.
- {**overlapping**} – означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам-предкам.

Отношение обобщения

Ограничение {**incomplete**} указывает на тот факт, что кроме прямоугольника, окружности и эллипса существуют и другие типы геометрических фигур;

Ограничение{**disjoint**} – на то, что прямоугольники не могут одновременно являться окружностями и эллипсами (и наоборот);

Ограничение {**overlapping**} – на то, что некоторые экземпляры класса **Эллипс** могут одновременно принадлежать и классу **Окружность**.



Отношение агрегации

Отношение агрегации между несколькими классами представляет системные взаимосвязи типа "целое-часть" между компонентами модели. Это отношение описывает декомпозицию сложной системы на более простые составные части.

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.

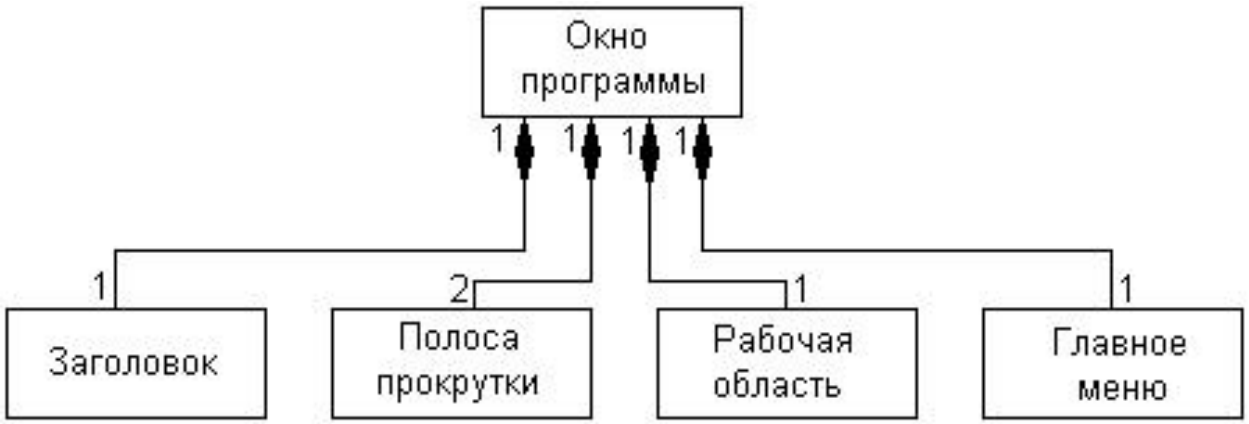
Отличие заключается в том, что *части системы не обязаны наследовать ее свойства и поведение*, поскольку являются вполне самостоятельными сущностями. Более того, части целого обладают своими собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого.

Отношение агрегации изображается на диаграмме сплошной линией, на одном из концов которой (соединенным с классом-агрегатом, представляющим "*целое*") помещается *не закрашенный внутри ромб*.

Отношение композиции

Отношение композиции является частным случаем отношения агрегации и используется для выделения специальной формы взаимосвязи между "целым" и его "частями", при которой части не могут существовать в отрыве от целого и уничтожаются при его уничтожении.

Рисунок иллюстрирует также возможность указания на диаграмме дополнительных параметров отношения композиции, в частности, параметра кратности связей. Например, кратность "1" связи с классом "**Рабочая_область**" указывает на тот факт, что некоторое приложение будет обрабатывать (и отображать а рабочей области окна) одновременно только один документ.



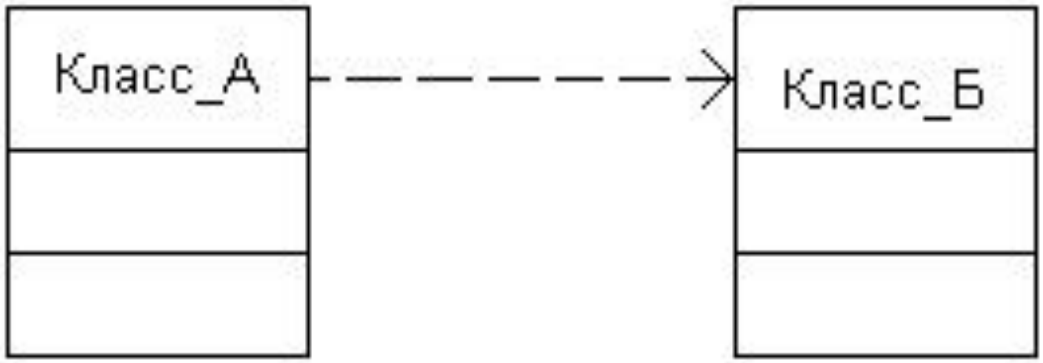
Отношение зависимости

Отношение зависимости указывает на некоторое семантическое отношение между двумя классами, которое не является отношением ассоциации, обобщения или реализации, но при этом изменение одного класса ("независимого") может потребовать изменения другого, зависимого от него класса.

Независимый класс называется "*источником зависимости*", а зависимый от него класс - "*клиентом зависимости*".

Отношение зависимости изображается на диаграмме пунктирной линией со стрелкой, направленной от класса-клиента к классу-источнику зависимости.

На рисунке **Класс_Б** является источником некоторой зависимости, а **Класс_А** – клиентом этой зависимости.



Параметры отношения зависимости

Линия зависимости может помечаться необязательными параметрами: стандартным ключевым словом (стереотипом), заключенным в кавычки, и/или индивидуальным именем.

Для отношения зависимости предопределены следующие стереотипы, каждый из которых обозначает некоторый специальный вид зависимости:

- "**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, когда в ходе работы над проектом появляется некоторая дополнительная информация.

Отношение зависимости может устанавливаться не только между классами, но также и между множествами классов, организованных в *пакеты* – в этом случае один пакет выступает в качестве источника зависимости, а другой пакет – в качестве клиента.

Интерфейсы

Интерфейс является элементом UseCase-диаграммы, однако при построении диаграммы классов отдельные интерфейсы могут уточняться.

Для изображения интерфейса используется графический символ класса:

- В секции «имя» дополнительно указывается стереотип "interface".
- Секция атрибутов у интерфейса отсутствует.
- В секции операций интерфейса указывается соответствующая информация.

Шаблоны

Шаблон (template) - это класс, имеющий нефиксированные формальные параметры. Шаблон, называемый также *параметризованным классом* (parametrized class), определяет множество классов, каждый из которых может быть получен связыванием формальных параметров шаблона с определенными фактическими значениями.

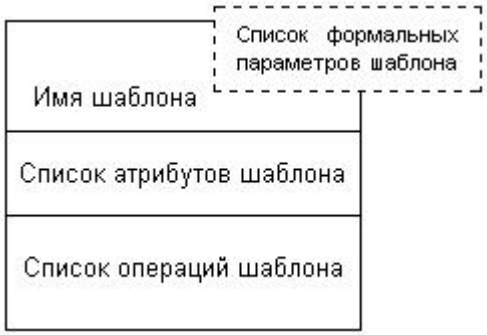
Обычно параметрами шаблонов служат типы атрибутов классов, в более сложных случаях формальные параметры могут представлять и операции класса.

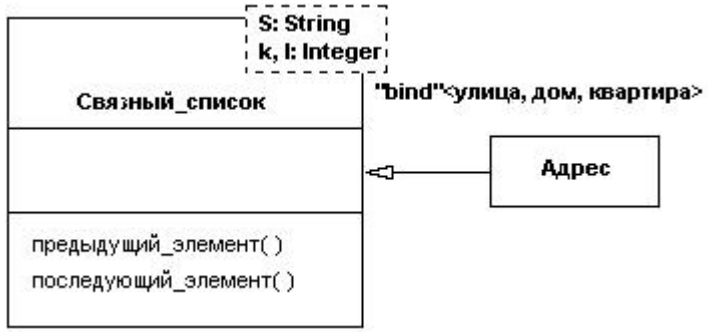
Шаблон не может быть непосредственно использован в качестве класса, поскольку содержит неопределенные параметры.

На диаграмме классов шаблон изображается аналогично классу. В отличие от обычного класса, обозначение шаблона содержит малый прямоугольник, расположенный в верхнем правом углу основного прямоугольника и нарисованный пунктирными линиями.

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

Шаблоны





Чаще всего в качестве шаблона выступает суперкласс, параметры которого уточняются в его классах-потомках. Очевидно, что в этом случае между ними существует *отношение зависимости* со стереотипом "**bind**", когда класс-клиент может использовать шаблон для своей последующей параметризации.

В более частном случае между шаблоном и формируемым от него классом определено *отношение обобщения* с наследованием свойств шаблона.

В приведенном на правом рисунке примере отмечен тот факт, что класс "**Адрес**" может быть получен из шаблона "**Связный_список**" заменой формальных параметров шаблона (**S, k, l**) фактическими атрибутами (**улица, дом, квартира**).

Этот же шаблон может использоваться для определения другого класса, например, класса **Точки_на_плоскости**, связывающего те же формальные параметры (**S,k,l**) с другими атрибутами, например, ("**координаты_точки**", **x**, **y**).

Концепция шаблонов является достаточно мощным средством в ООАП, и поэтому ее использование в языке UML позволяет не только сократить размеры диаграмм, но и наиболее корректно управлять наследованием свойств и поведения элементов модели.

Заключение

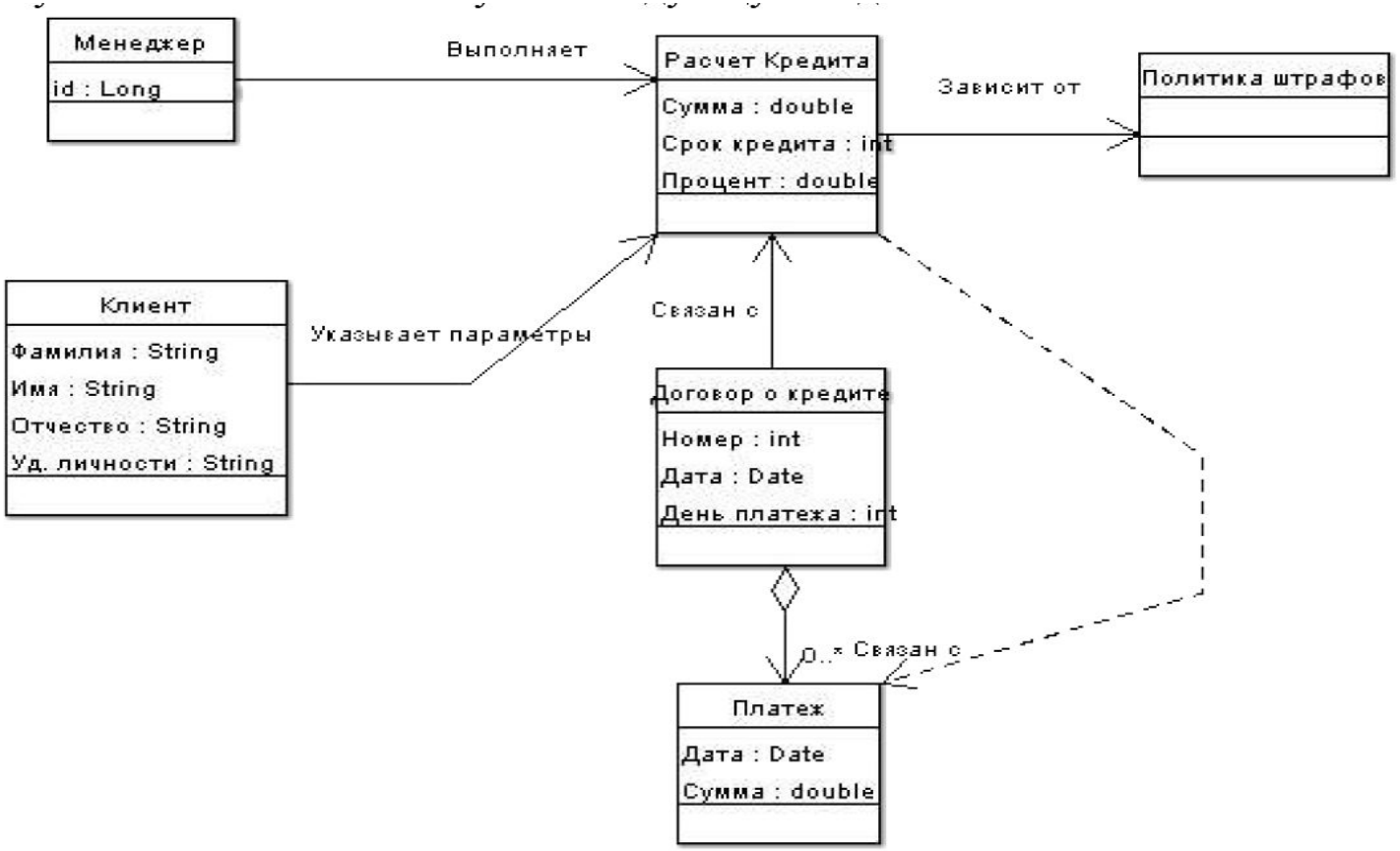
Процесс разработки диаграммы классов занимает центральное место в ООАП сложных систем.

От умения правильно выбрать классы и установить между ними взаимосвязи часто зависит не только успех процесса проектирования, но и производительность выполнения программы.

После разработки диаграммы классов процесс ООАП может быть продолжен в двух направлениях:

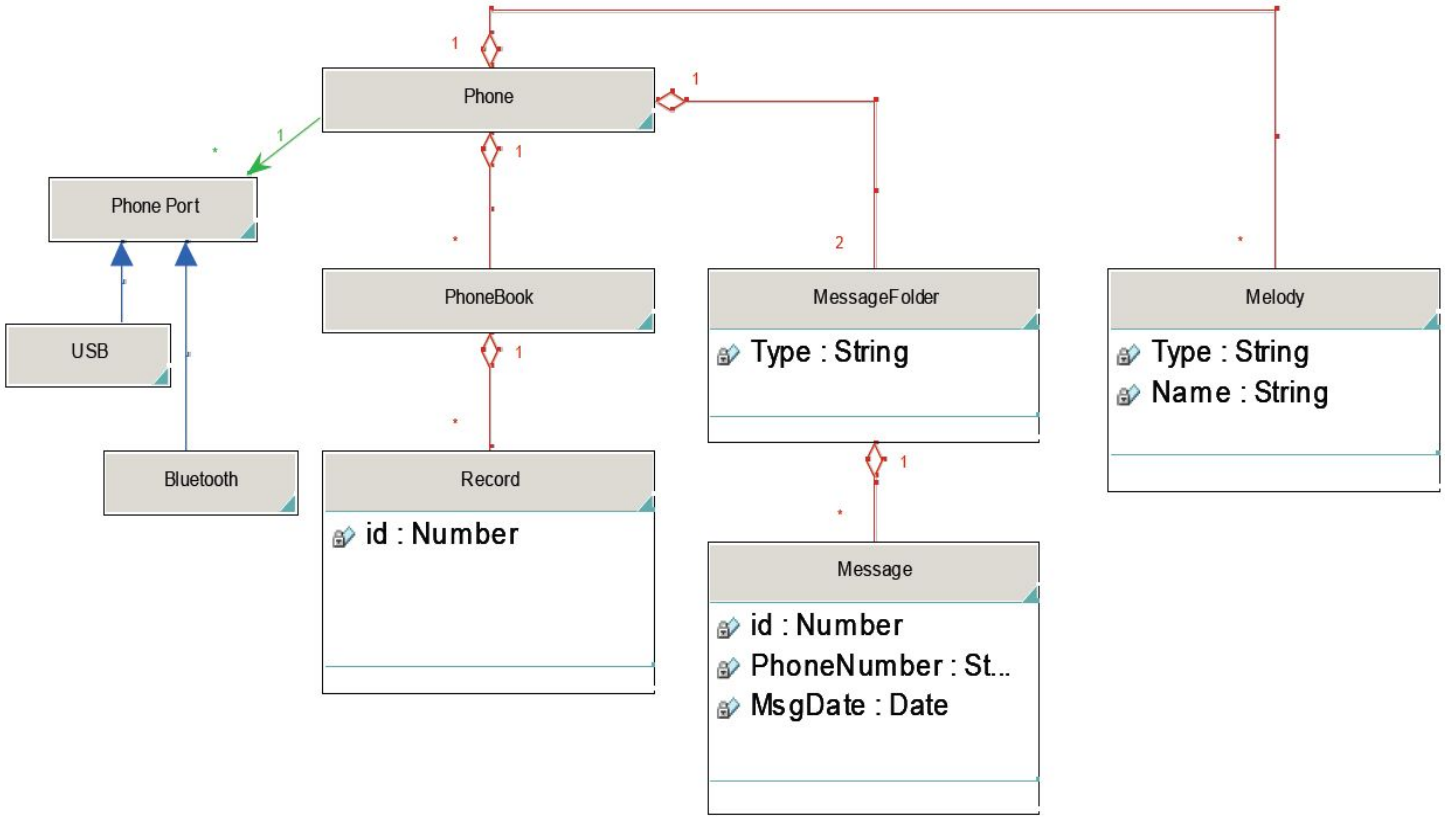
1. Если поведение системы тривиально, то можно приступить к разработке
   - диаграмм кооперации
   - диаграмм компонентов;
2. Для сложных динамических систем поведение представляет важнейший аспект их функционирования, и детализация поведения системы осуществляется последовательно при разработке
   - диаграмм состояний,
   - диаграмм последовательности
   - диаграмм деятельности.

Пример 1:
Расчет платежей по кредиту



Пример 2:
Браузер данных, хранящихся в мобильном телефоне.

Программа считывания поступивших SMS



Пример 3:
Анализатор LOG-файлов