Тема 2.6.3

Диаграмма классов

Содержание

1. Назначение диаграммы

Назначение диаграммы классов

Диаграмма классов занимает центральное место при проектировании ПС с использованием объектно-ориентированного подхода к разработке ПО.

Большинство современных CASE-средств осуществляют автоматическую генерацию кода основываясь именно на этой диаграмме.

Диаграмма классов – диаграмма, предназначенная для представления модели статической структуры программной системы в терминологии классов ООП.

Назначение диаграммы классов

Разработка этой диаграммы преследует следующие **цели**:

- определить сущности предметной области и представить их в форме классов с соответствующими атрибутами и операциями;
- определить взаимосвязи между сущностями предметной области и представить их в форме типовых отношений между классами;
- разработать исходную модель программной системы для последующей реализации в форме физических моделей;
- подготовить документацию для последующей разработки программного кода.

2. Сущности диаграммы

Сущность Класс

Класс – это множество объектов, которые обладают *одинаковой* структурой, поведением и отношениями с объектами из других классов.

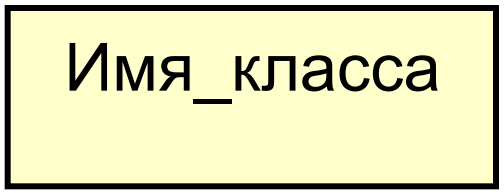
Т.о., класс – это описание набора объектов с общими

- атрибутами,
- операциями
- и семантикой.

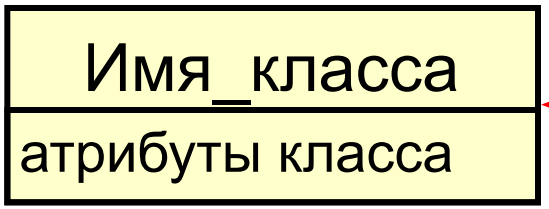
Сущность Класс

Графически класс изображается в виде разделенного на три части прямоугольника, где записаны его

- имя,
- атрибуты
- и операции.



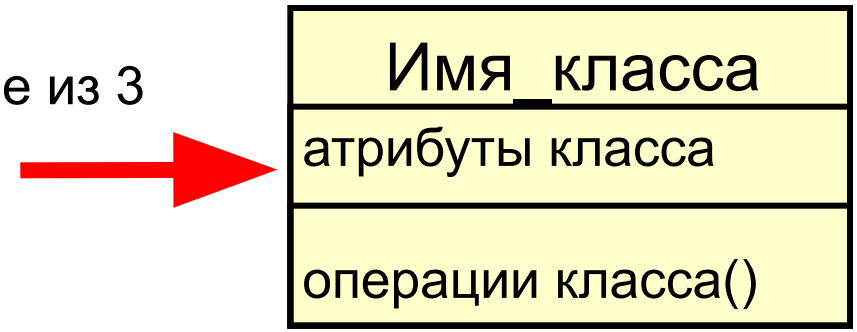
Простейший вид класса состоит только из секции имени



Класс с указанием атрибутов (переменных)

Полное описание класса, состоящее из 3 разделов (секций)

✔секции имени,
✔секции атрибутов,
✔секции операций



Сущность Класс

Примеры:

Имя класса

Основными **характеристиками класса/объекта** являются:

1. ***Уникальное имя***.

   Позволяет отличить объекты друг от друга.

   - ***Имя класса*** должно быть уникально
   - Имя класса должно начинаться с заглавной буквы.
   - Класс может не иметь экземпляров или объектов.
     - ✔ В этом случае он называется ***абстрактным классом***,
     - ✔ а для обозначения его имени используется *курсив* .

Атрибуты класса

2. ***Атрибут*** – именованное свойство объектов класса, которое является общим для всех объектов данного класса.

   Каждый атрибут имеет свое значение для каждого объекта.

   Среди атрибутов различают:

   - статические (условно постоянные свойства);
   - динамические (условно переменные свойства);
   - производные (вычисляемые свойства), значения которых определяются при помощи значений статических и динамических атрибутов объекта.

Атрибуты класса

Общий формат записи атрибутов:

<квантор видимости> <имя атрибута> [кратность]:
<тип атрибута> = <исходное значение>
{строка-свойство}

Квантор видимости может принимать одно из следующих значений:

- ✔«**+**» - атрибут с областью видимости типа ***общедоступный*** (public).
- ✔«**#**» - атрибут с областью видимости типа ***защищенный*** (protected).
- ✔«**-**» - атрибут с областью видимости типа ***закрытый*** (private).
- ✔«**~**» - атрибут с областью видимости типа ***пакетный*** (package).

Атрибуты класса

Имя атрибута представлено в виде уникальной строки текста

- ✔ Имя атрибута является единственным обязательным элементом в синтаксическом обозначении атрибута
- ✔ Должно начинаться со строчной буквы
- ✔ По практическим соображениям записывается без пробелов

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса.

- ✔ Формат:

  [нижняя граница . . верхняя граница]
- ✔ Примеры: [0..1], [0..*], [1..3,5..7]

Атрибуты класса

Тип атрибута – выражение, определяемое некоторым **типом данных** (например, в зависимости от языка программирования).

- ✔ В простейшем случае – *осмысленная строка текста*.
- ✔ Пример:

цвет: Color
имяСотрудника[1..2]: String;
видимость: Boolean

Исходное значение служит для задания некоторого начального значения в момент создания отдельного экземпляра класса

- ✔ Пример:

цвет: Color = (255, 0, 0)
имяСотрудника[1..2]: String = ‘Иван Иванов’;
видимость: Boolean = истина

Атрибуты класса

Строка-свойство служит для указания дополнительных свойств атрибута, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения соответствующей программы.

- ✔ Это значение принимается за ***исходное значение атрибута***, которое не может быть изменено в дальнейшем.
- ✔ Пример:

  заработнаяПлата: Currency = $500 {frozen}

Состояние объекта

3. ***Состояние объекта*** – ситуация в ходе жизни объекта, в течение которой он
   - ✔ удовлетворяет некоторому условию,
   - ✔ выполняет некоторую деятельность
   - ✔ или ожидает некоторого события.
   - Выражается состояние всеми текущими значениями объекта.
   - При изменении значений атрибутов меняется состояние.

Поведение объекта

4. ***Поведение*** – это то, как объект действует и реагирует (то есть, как он меняет свое состояние).
   - Поведение выражается в терминах состояния объекта и передачи сообщений.
   - Поведение может изменять состояние.
   - Понятие ***сообщение*** совпадает с понятием *операция*.

Операции класса

5. ***Операция*** – действие, которое должен выполнить объект для реализации своего поведения, или сервис, который может быть востребован одним объектом у другого.
   - У каждой операции есть свой объект-получатель, то есть объект, для которого операция применяется.
   - Операция может иметь дополнительные аргументы, которые ее параметризуют.
   - Некоторые операции могут приобретать разные формы в различных классах. Такие операции называют ***полиморфными***.

Операции класса

Правила записи операций:

<квантор видимости> <имя операции>
(список параметров): <выражение типа возвращаемого значения> {строка-свойство}

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

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

Операции класса

Строка-свойство служит для указания значений свойств, которые могут быть применены к данной операции.

- Например, для указания последовательности действий будет использована строка-свойство вида:

  {concurrency = имя} ,

  где *имя* может принимать одно из следующих значений:
  - *sequential* (последовательная),
  - *concurrent* (параллельная),
  - *guarded* (охраняемая)

Операции класса

Примеры:

+нарисовать (форма : Многоугольник = прямоугольник, цветЗаливки : Color = (0, 0, 255));

-изменитьСчетКлиента (номерСчета : Integer) : Currency;

#выдатьСообщение() : (‘Ошибка деления на ноль’).

Операции класса

- К стереотипным операциям для всех классов относятся следующие операции:
  - ✔ создать и инициировать *(create* и *initialize)* объект;
  - ✔ удалить *(delete)* объект;
  - ✔ получить *(get…)* значение атрибута;
  - ✔ установить *(set…)* значение атрибута;
  - ✔ добавить объект *(add…)* – добавить связь с другим объектом;
  - ✔ исключить объект *(remove…)* – исключить связь с другим объектом.
- К наиболее распространенным операциям относятся:
  - ✔ *модификатор* – операция, меняющая состояние объекта;
  - ✔ *запрос (селектор)* – операция, которая считывает состояние объекта, но не меняет состояние;
  - ✔ *итератор* – операция, которая позволяет организовать доступ ко всем частям объекта-контейнера для последующего их использования в строго определенной последовательности (например, просмотр очереди);
  - ✔ *преобразование* – операция, производящая объект другого типа.

Методы класса

6. ***Метод*** – конкретная реализация операции.

Для выбора кандидатов в объекты можно воспользоваться списком категорий понятий из таблицы на следующем слайде.

Кандидаты в объекты

Категория понятий	Пример
Физические или материальные объекты	*Самолёт*
Спецификации, элементы дизайна или описания	*Описание полёта*
Места	*Аэропорт*
Транзакции	*Резервирование, Продажа*
Элементы транзакций	*Элемент продажи*
Роли людей	*Пилот*
Контейнеры для других объектов	*Самолёт*
Содержимое контейнеров	*Пассажир*
Другие компьютерные или электромеханические системы, внешние по отношению к данной системе	*Система управления полётами*
Абстрактные понятия	*Боязнь высоты*
Организации	*Авиалиния*
События	*Полёт, Крушение*
Процессы (в основном, не представляются в виде понятий)	*Продажа, Бронирование места*
Правила и политика	*Политика аннулирования заказа*
Каталоги	*Каталог частей*
Записи финансовой, трудовой, юридической и другой деятельности	*Чек, Трудовой контракт, Журнал обслуживания*
Финансовые инструменты и службы	*Кредитная линия, Акция*
Руководства, книги	*Должностные инструкции, Руководства по восстановлению*

Класс Интерфейс

Интерфейс (***interface***) – в контексте языка UML является специальным случаем класса, у которого имеются только операции и отсутствуют атрибуты.

3. Отношения на диаграмме

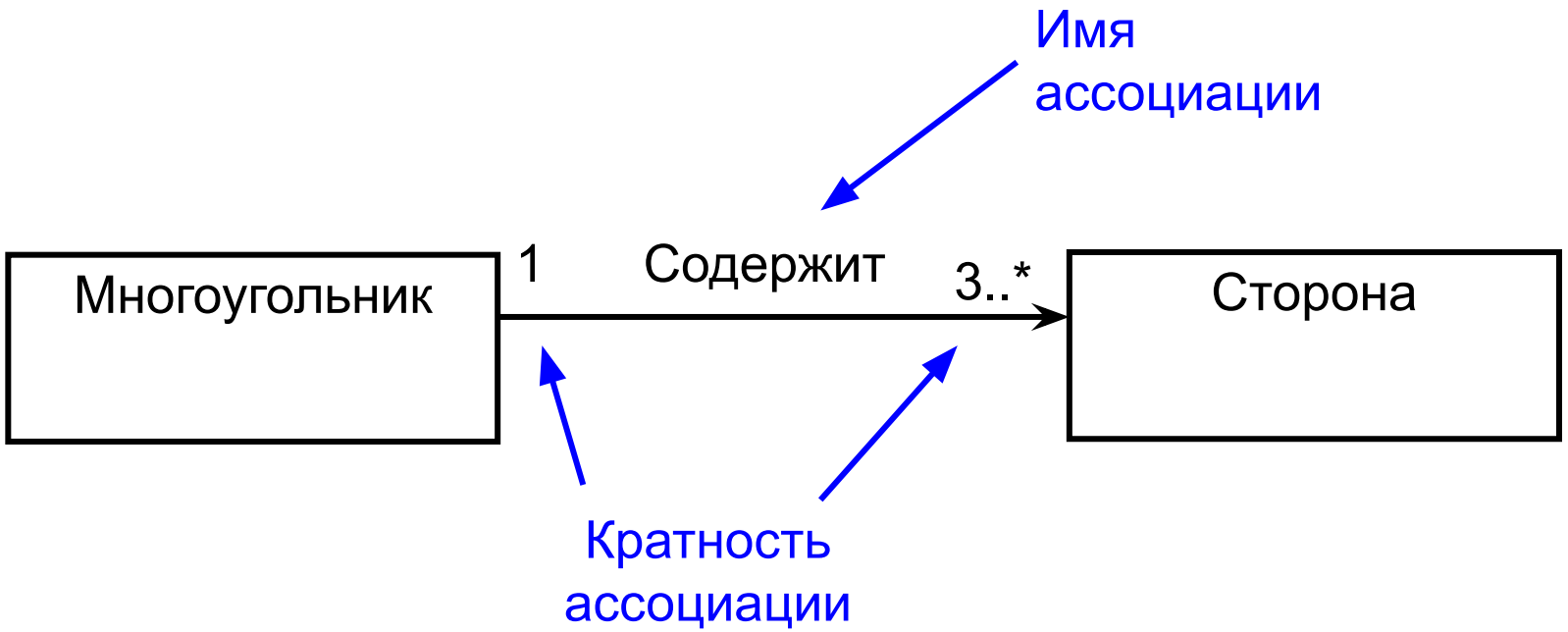
Виды отношений между классами

Базовыми отношениями на диаграмме классов являются:

- отношения ***ассоциации*** (*association*);
- отношения ***обобщения*** (*generalization*);
- отношения ***агрегации*** (*aggregation*);
- отношения ***композиции*** (*composition*);
- отношения ***зависимости*** (*dependency*).

Отношение ассоциации

- Самым распространенным типом отношений является ***ассоциация*** *(association)*, которая отражает значимые и полезные связи объектов.
  - Отношение ассоциации свидетельствует о наличии произвольного отношения между классами.



Отношение ассоциации

Один конец ассоциации называется ***полюсом***. Полюс обладает кратностью, ограничивающей количество связанных между собой объектов.

Кратность – это количество объектов одного класса, которые могут быть связаны с одним объектом класса противоположного конца ассоциации.

- Кратность указывается рядом с полюсом под линией ассоциации и может принимать одно из следующих значений:
  - ✔ 1 – точно один объект;
  - ✔ 0..1 – ноль или один объект;
  - ✔ 0..* – ноль или больше объектов;
  - ✔ 1..* – один или больше объектов;
  - ✔ * – много объектов.

Отношение ассоциации

- По своей природе ассоциация двусторонняя, имеет два конца и может быть прослежена как в одном, так и в обоих направлениях.
- В UML двунаправленная ассоциация обозначается при помощи прямой линии.
- Когда ассоциация однонаправлена, линия заканчивается стрелкой, указывающей направление ассоциации.

На рис. показаны оба вида обозначений:

Отношение ассоциации

На рис. ниже показана двунаправленная ассоциация с кратностью

- ✔ на одном полюсе *много*,
- ✔ а на другом – *один*.

Эта ассоциация двунаправленная, поэтому справедливы оба утверждения:

- ✔ много людей работает на одном предприятии,
- ✔ и на одном предприятии работает много людей.

Отношение ассоциации

Полюс может иметь не только кратность, но и располагать ***именем роли***, которое описывает роль объектов.

- Представление ролей в ассоциации не является обязательным, однако
  - ✔ значительно улучшает понимание диаграмм классов,
  - ✔ обеспечивает простоту добавления уникальных атрибутов при программировании классов.
- Имя роли указывается около конца ассоциации.

Отношение ассоциации

На рис. ниже показаны роли в двух однонаправленных ассоциациях:

- *управляет*
- и *назначает*.

- Роль объектов класса *Человек* в ассоциации *назначает* определена как *менеджер*:
  - ✔ одно предприятие назначает некоторых из многих людей менеджерами.
- В ассоциации *управляет* повторно используется роль *менеджер* и назначается вторая – *рабочий*:
  - ✔ *один человек, являющийся менеджером, управляет многими людьми, которые являются рабочими предприятия*.

Отношение обобщения

- ***Обобщение*** *(generalization)* является частным случаем ассоциации и обозначает отношение типа общее – частное *(is – a)*. Обобщение указывает на наличие общих характеристик между классами.
  - Класс, имеющий наиболее общие для некоторой группы классов характеристики, называется ***суперклассом***,
  - а его специализированная версия – ***подклассом***.

  - Открытые и защищенные атрибуты и операции суперкласса наследуются всеми подклассами.
  - Любой подкласс может добавлять уточняющие характеристики к унаследованным.
  - Дополнительные атрибуты и/или операции в подклассах могут отсутствовать.

Отношение обобщения

Отношение обобщения не является набором связей между объектами, поэтому направление и кратность для обобщения не указываются.

Обобщение допускает многоуровневую иерархию классов,

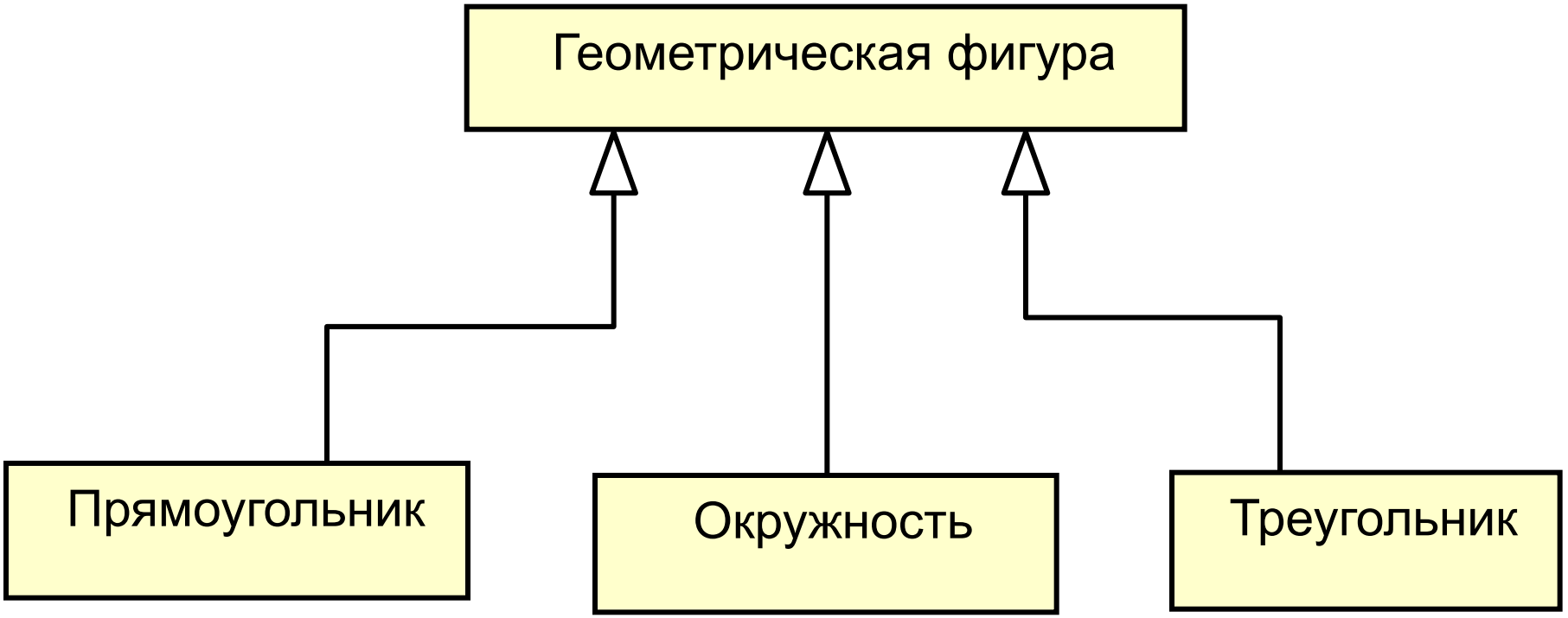
- ✔ но лучше ограничиться тремя уровнями, чтобы не усложнять модификацию программ,
- ✔ так как изменение суперкласса требует корректировки подклассов и дополнительного тестирования.

Программируется обобщение при помощи механизма ***наследования***:

- суперкласс становится базовым классом (как правило, абстрактным);
- подклассы реализуются как производные классы.

Отношение обобщения

В UML обобщение обозначается линией с треугольником на конце. Вершина треугольника направлена в сторону суперкласса (см. рис.).

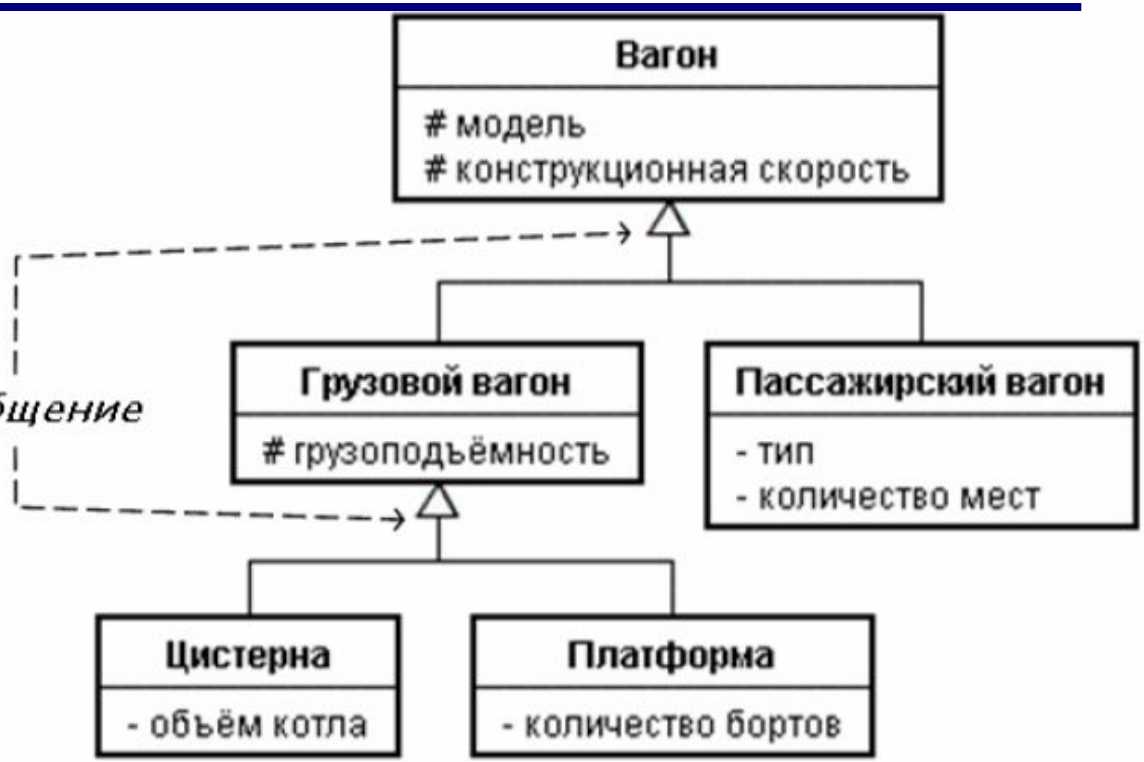


Отношение обобщения

Пример:

- Первый уровень иерархии содержит класс *Вагон* – суперкласс для классов второго уровня:
  - ✔ *Грузовой вагон;*
  - ✔ *Пассажирский вагон*.
- Оба класса наследуют от суперкласса *Вагон* защищенные атрибуты
  - ✔ *модель*
  - ✔ и *конструкционная скорость*,

и имеют дополнительные.



- *Грузовой вагон* служит суперклассом для классов третьего уровня иерархии:
  - ✔ *Цистерна*
  - ✔ и *Платформа*.
- К их закрытым атрибутам добавляются атрибуты классов *Грузовой вагон* и *Вагон*.

Отношение обобщения

Дискриминатор показывает признак, указывающий, по какому свойству объектов сделано обобщение.

- На рис. показано обобщение с указанием дискриминаторов.

Отношение агрегации

- ***Агрегация*** *(aggregation)* – специализированный вид ассоциации, который обозначает отношение часть целого *(part of, has)*, где каждая пара классов определяется отдельно.
  - **Смысл**: один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей другие сущности.
  - Применяется для представления системных взаимосвязей типа «*часть-целое*».

Отношение агрегации

Агрегация имеет два важных **свойства**:

1. ***Транзитивность***
   - ✔ если ***A*** является частью ***B***, а ***B*** – частью ***C***, то ***A*** является частью ***C***.
2. ***Асимметричность***
   - ✔ если ***A*** является частью ***B***, то ***B*** не является частью ***A***.

Отношение агрегации

Для отношения часть целого в UML существует две формы:

- *общая (агрегация)*
- и *частная (композиция)*.

Композиция – особый случай агрегации с дополнительными ограничениями:

- ✔ часть может принадлежать не более чем одному целому;
- ✔ приписанная к некоторому целому часть автоматически получает срок жизни, совпадающий со сроком жизни целого.

Т.о. части не могут выступать в отрыве от целого,

- ✔ т.е. с уничтожением целого уничтожаются составные части.

Отношение агрегации

Агрегация допускает независимую обработку объектов-частей и объекта-целого.

- ✔ Обработка частей в композиции возможна только через объект целого,
- ✔ так же как и доступ внешних объектов к частям.

Кратность присутствует в каждой форме отношении часть целого и ставится на стороне частей.

- Если ее значение не указано, то предполагается, что в отношении участвует много частей и одно целое.

Отношение агрегации

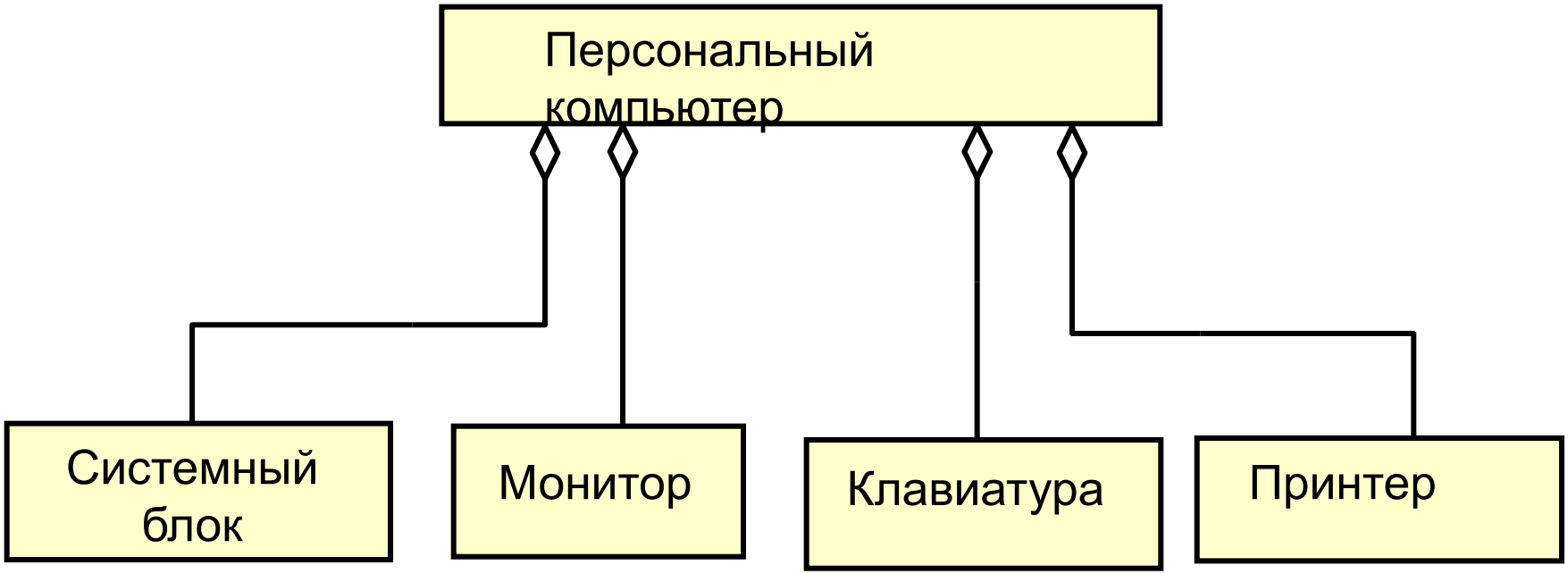
В UML агрегация обозначается линией с полым ромбом около полюса целого, у композиции ромб закрашен (см. рис.).

Из диаграммы видно, что

- в одном вагоне много грузов
  - ✔ при потере вагона не обязательно исчезнут все грузы,
- на одном предприятии много отделов
  - ✔ при ликвидации предприятия отделы перестают существовать.

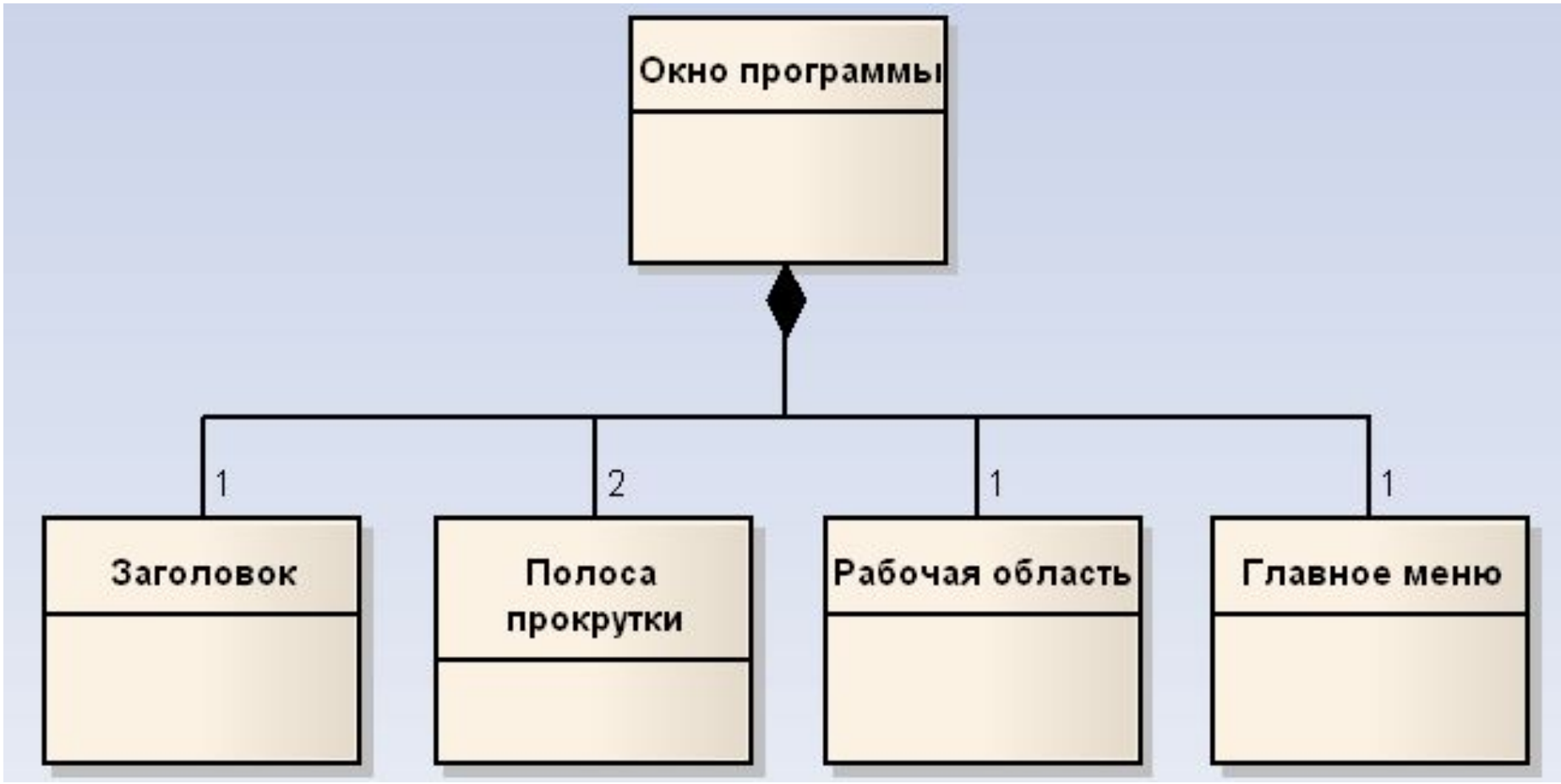
Отношение агрегации

Пример агрегации:



Отношение агрегации

Пример композиции:



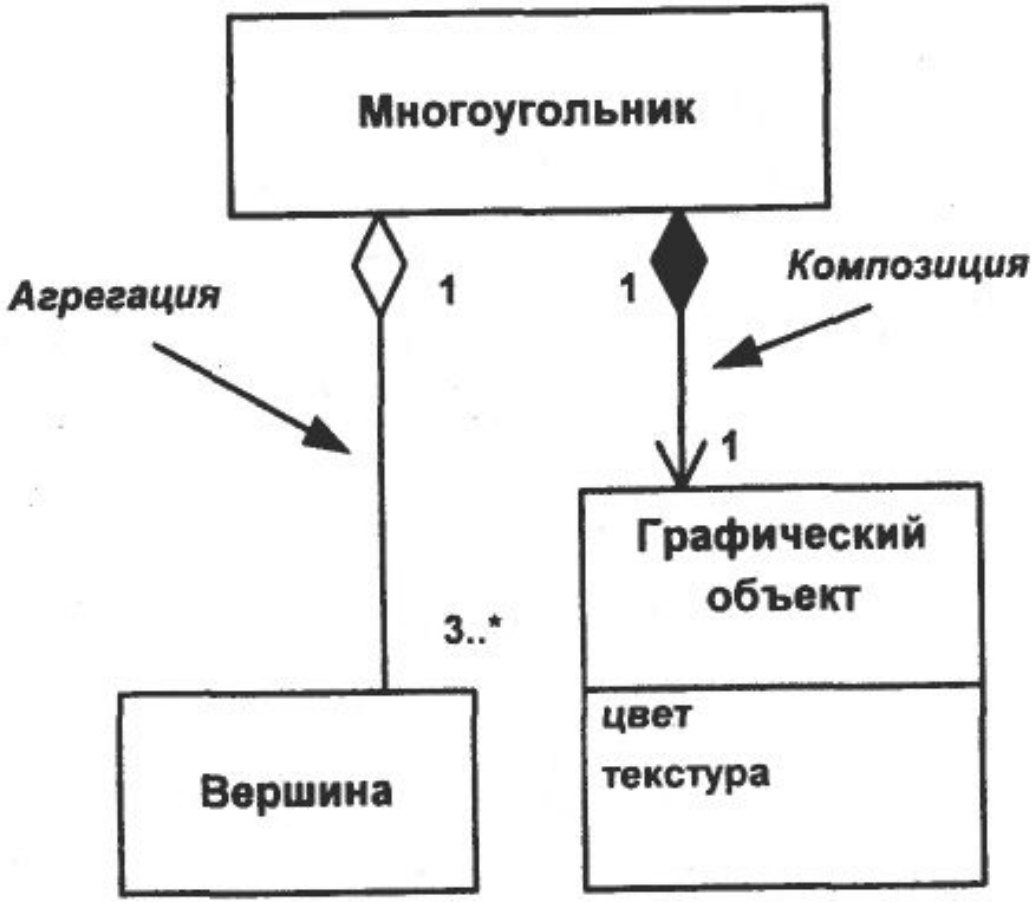
Отношение агрегации

Пример агрегации, где

- тепловоз *(агрегат)* состоит из
  - ✔ секций *(частей)*
  - ✔ и дизелей *(частей)*;
- каждый дизель *(как агрегат)* имеет в своем составе
  - ✔ топливные насосы *(части)*.
- По свойству транзитивности насосы – части тепловоза.
- По свойству асимметричности агрегации тепловоз не является частью топливных насосов.

Отношение агрегации

Пример агрегации и композиции

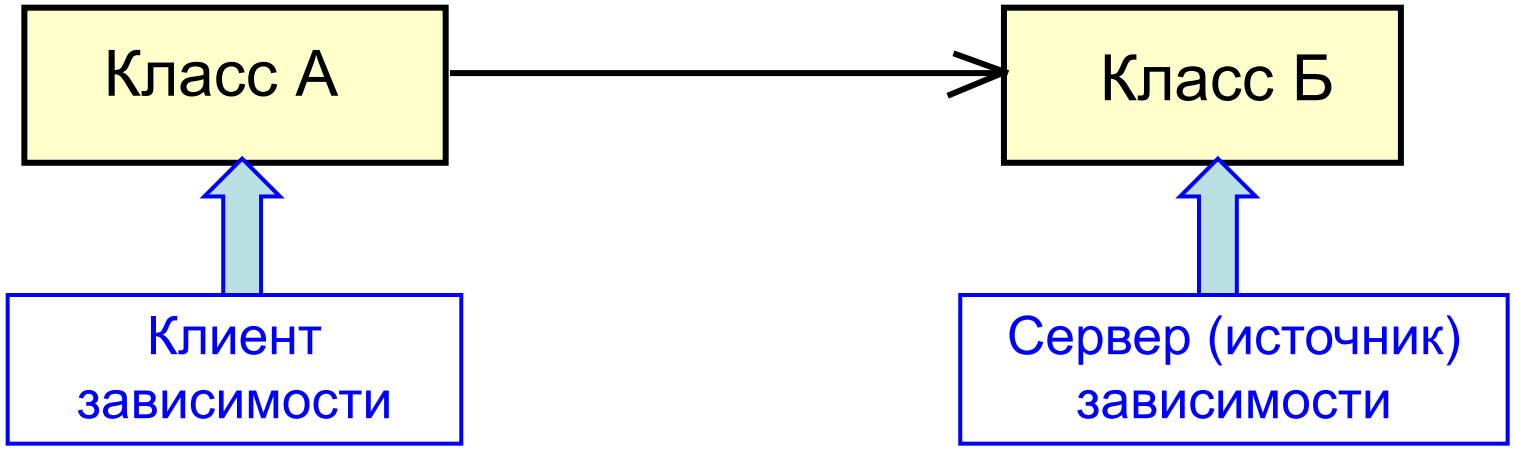


Отношение зависимости

- ***Зависимость*** – это **однонаправленное** отношение использования между двумя классами.
  - На одном конце отношения находится зависимый класс, на втором – независимый.
  - Объект-клиент зависимого класса для своего корректного функционирования пользуется услугами объекта-сервера независимого класса.
  - Зависимость отражает связь между объектами по применению, когда изменение поведения сервера может повлиять на поведение клиента.

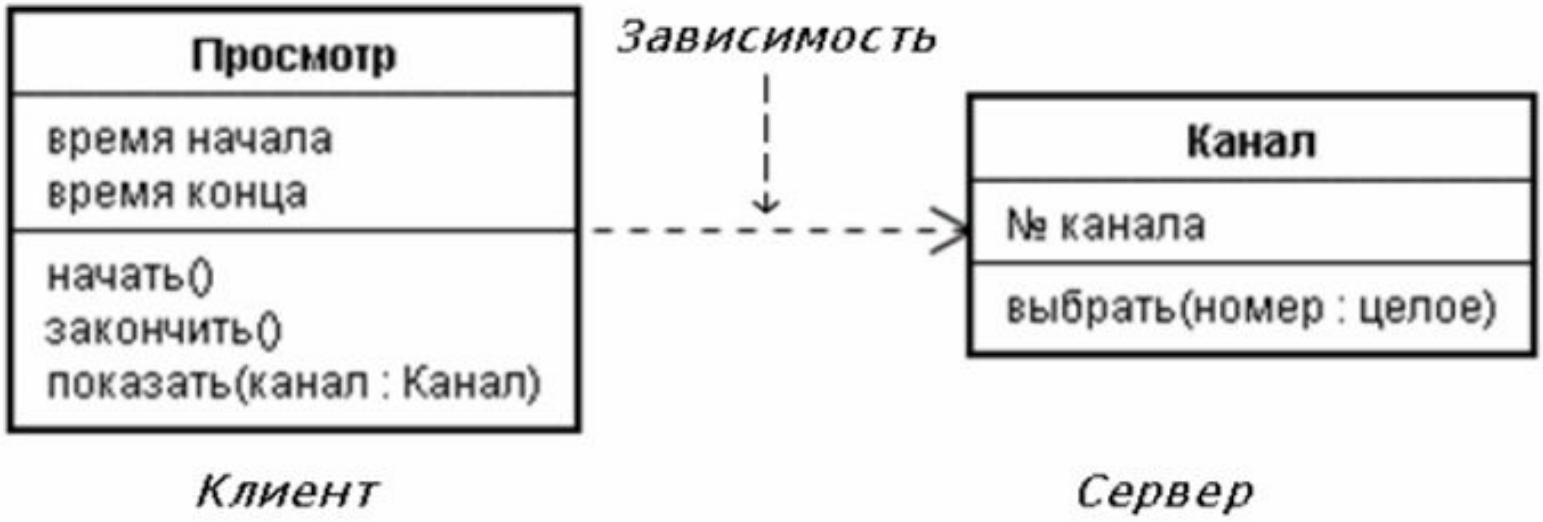
Отношение зависимости

В UML отношение зависимости изображается пунктирной стрелкой, всегда направленной в сторону независимого класса (см. рис.).



Отношение зависимости

Пример:



Показана зависимость просмотра тех или иных передач от выбора телевизионного канала зрителем.

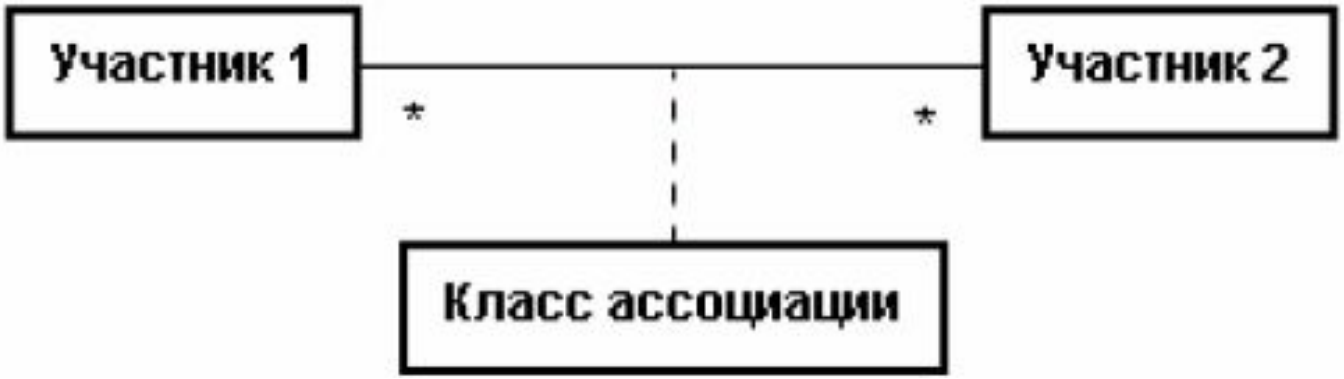
- ✔ Объект-клиент *Просмотр* использует объект-сервер *Канал* для реализации операции *показать ( )*.
- ✔ Состояние *Просмотра* зависит от *Канала*, номер которого может изменяться.
- ✔ Выбранный канал передается операции *показать ( )* как фактический параметр *(канал : Канал)*.
- ✔ Результатом выполнения операции объектом *Просмотр* станет смена его состояния.

Класс ассоциации

- ***Класс ассоциации*** *(association class)* позволяет назначить для ассоциации атрибуты и операции, которые являются принадлежностью связей и не могут быть приписаны ни к одному из объектов-участников.
  - Для одной ассоциации может существовать **только один** класс ассоциации.
    - ✔ Важная особенность класса ассоциации заключается в том, что для объектов-участников создается только один объект этого класса.

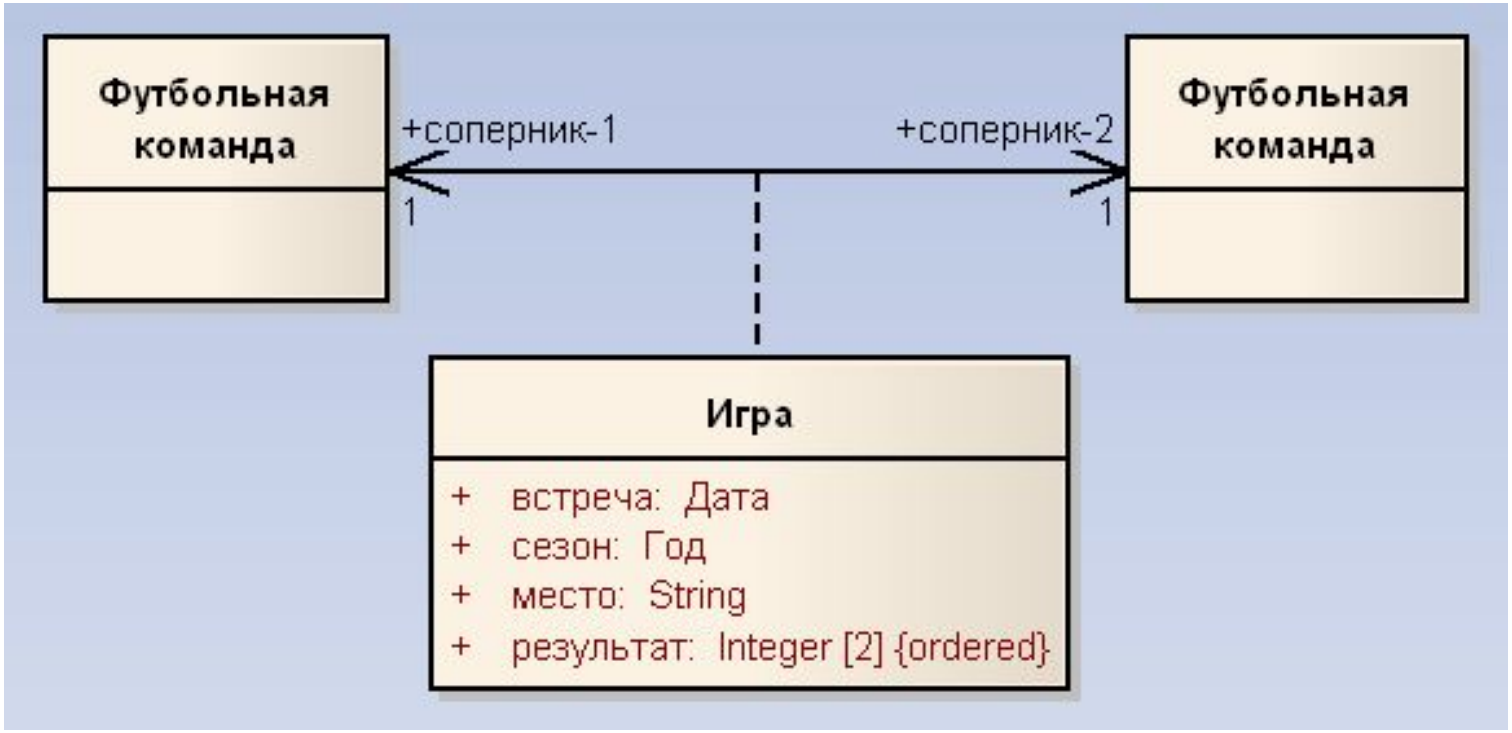
Класс ассоциации

Обозначается класс ассоциации как обычный класс, соединенный пунктирной линией с той ассоциацией, которую он уточняет (см. рис.).



Класс ассоциации

Пример:



Класс ассоциации

Основанием для введения классов ассоциации послужили ассоциации с кратностью *многие–ко–многим*, где возникает неоднозначность в определения атрибутов классов-участников.

Например, в сессию основной экзамен по дисциплине проводится для студента один раз.

- Между классами *Студент* и *Дисциплина* существует ассоциация *многие–ко–многим*:
  - ✔ один студент экзаменуется по многим дисциплинам,
  - ✔ и по одной дисциплине – много студентов.
- Полученные студентом оценки не могут быть атрибутами ни одного названного класса.

Класс ассоциации

- На рис. ниже показана диаграмма с классом ассоциации для регистрации результатов экзаменов.
  - ✔ Дата и оценка экзамена конкретного студента по конкретной дисциплине хранятся объектом класса ассоциации *экзамен*.
  - ✔ Для каждой пары объектов студент – дисциплина будет существовать один объект экзамен.

Класс ассоциации

- Класс ассоциации может участвовать в другой ассоциации, как показано на рис. ниже.
  - ✔ Теперь объект *экзамен* знает об объекте *преподаватель*, чтобы сохранить данные о педагоге, принявшем у студента экзамен по дисциплине.
  - ✔ Так как не все преподаватели участвуют в приеме экзаменов, для объектов класса определена роль *экзаменатор*.

Класс ассоциации

Классы ассоциации существенно отличаются от обычных классов, потому что для них разрешается только один объект.

- Если требуется создание нескольких или многих объектов для одной и той же пары участников, используется отдельный класс.

Пусть для рассмотренного выше примера

- необходимо регистрировать результаты не только основного экзамена, но еще двух дополнительных, которые могут сдавать некоторые студенты.
- Для решения задачи класс ассоциации преобразован в обычный класс для регистрации результатов экзаменов (см. рис. на след. слайде).

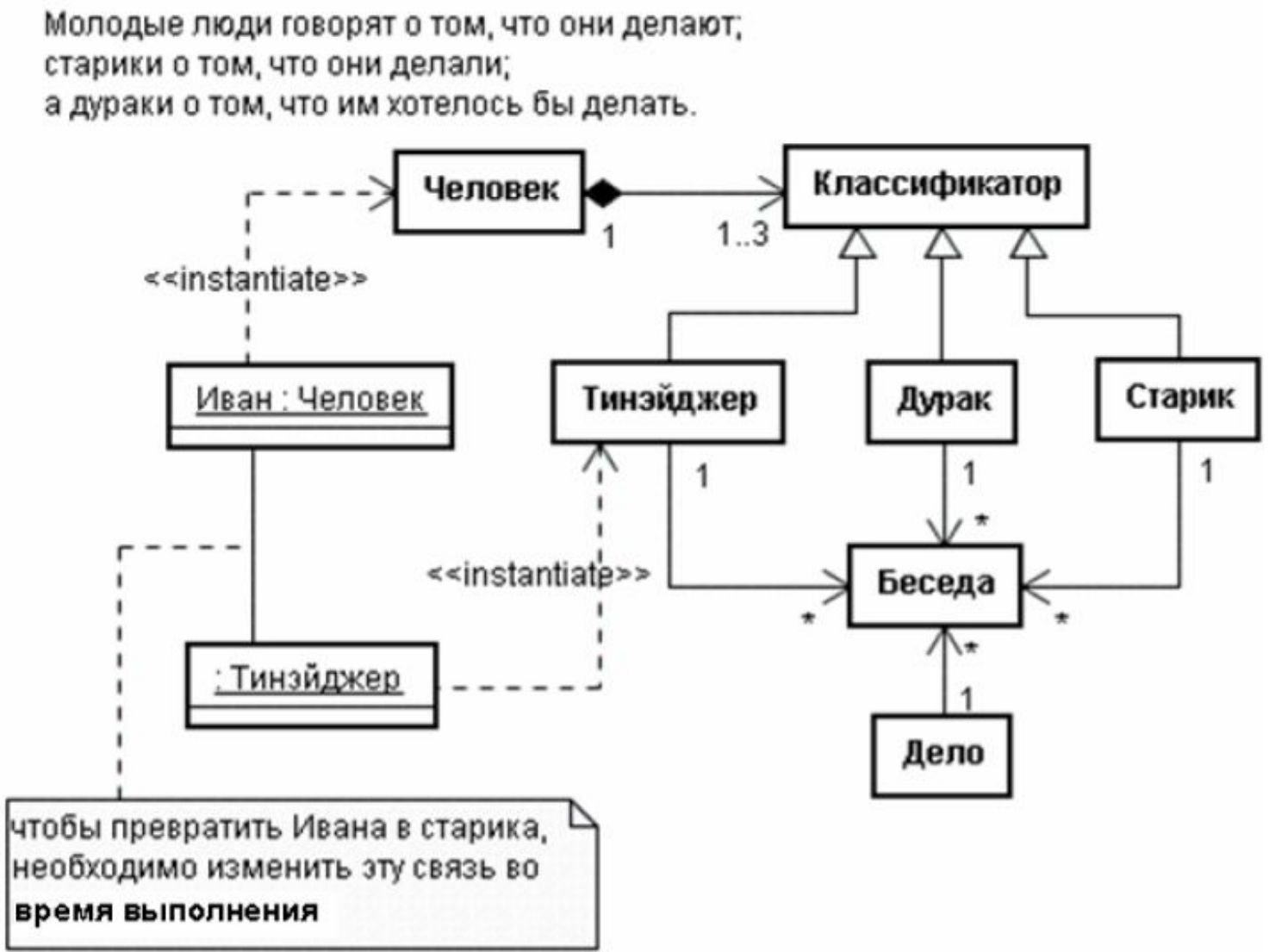
Класс ассоциации

- Здесь класс *Экзамен* уже не является классом ассоциации.
- Это обычный класс, который участвует в трех следующих ассоциациях:
  1. Студент – Экзамен:
     - ✔ Один студент не сдает экзаменов вообще или экзаменуется не более трех раз.
  2. Дисциплина – Экзамен:
     - ✔ По одной дисциплине проводится много экзаменов.
  3. Преподаватель – Экзамен:
     - ✔ Один преподаватель (являющийся экзаменатором) принимает много экзаменов.

Отношения между классами

Для делегирования композиция и обобщение используются совместно, что повышает устойчивость системы.

Пример делегирования показан на рис. ниже.



Отношения между классами

Наличие отношений между классами добавляет в классы дополнительные атрибуты, которые, как правило, реализуются посредством указателей.

Точки зрения на диаграмму классов

Диаграмма классов представляет собой граф, вершинами которого являются классы, между которыми установлены разные отношения.

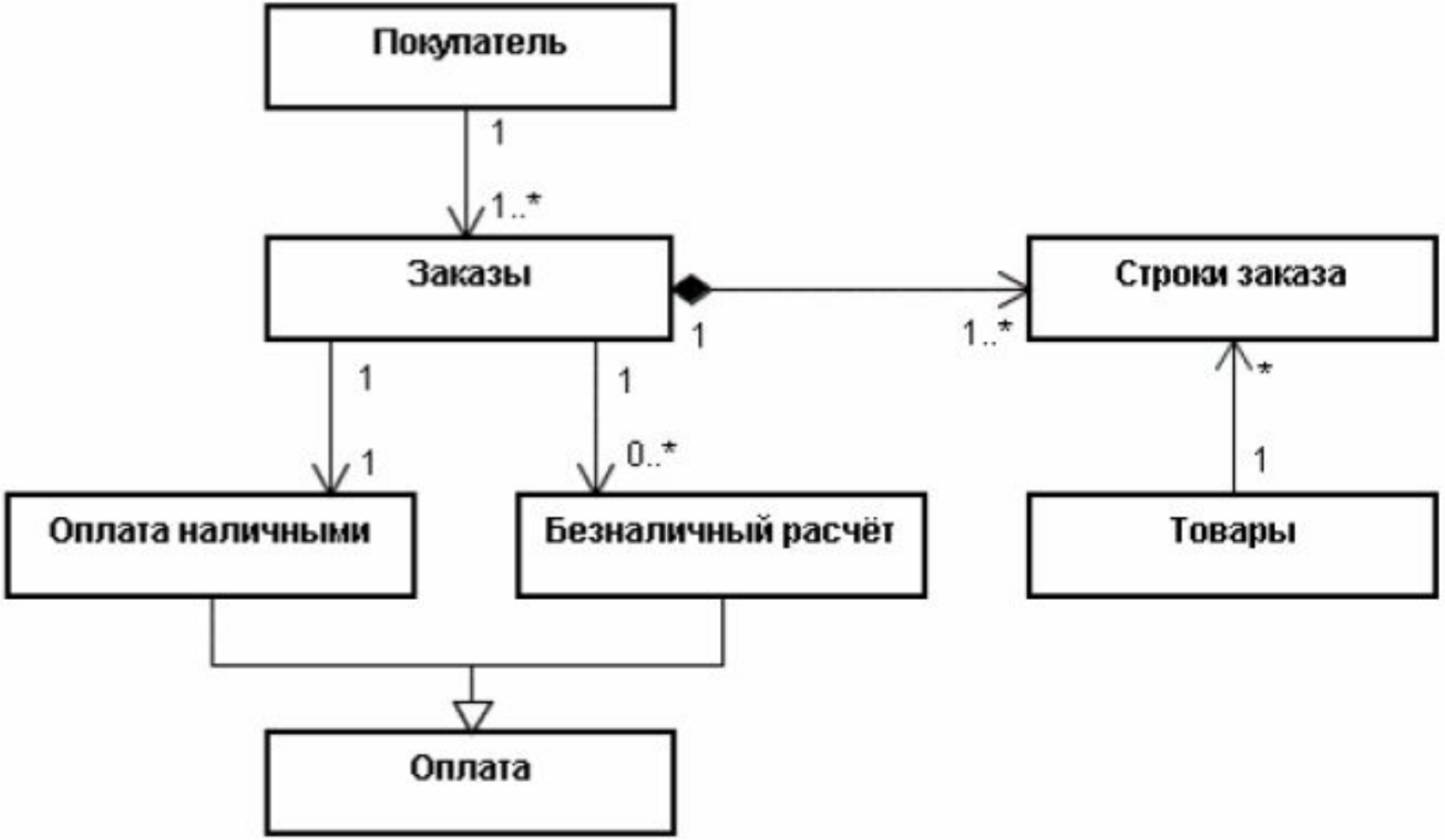
Как правило, для каждого прецедента создается своя диаграмма классов, которая уточняется по мере работы над проектом.

Точки зрения на диаграмму классов

Существует три различные точки зрения на диаграмму классов:

1. *Концептуальная точка зрения*.
   - Диаграмма классов отображает понятия предметной области и не имеет никакого отношения к реализующему ее ПО.
   - Классы показывают без атрибутов и операций.
   - Эти диаграммы полезны на стадии анализа.
   - Пример концептуальной диаграммы классов показан на рис. след. слайда.

Пример концептуальной диаграммы



Точки зрения на диаграмму классов

2. *Точка зрения спецификации*.
   - Моделирование спускается на уровень ПО.
   - Рассматриваются только интерфейсы классов, но без привязки к реализации в ПО.
   - Логические абстракции
     - ✔ классы пользовательского интерфейса,
     - ✔ классы управления
     - ✔ и классы сущностей

     показывают с атрибутами и операциями, а также уточняют отношения.
   - Эти диаграммы полезны на стадии проектирования.

Точки зрения на диаграмму классов

3. *Точка зрения реализации*.
   - Моделирование спускается на уровень реализации.
   - В этом случае логические абстракции проектирования будут представлены на диаграмме классами выбранного объектно-ориентированного языка программирования.
   - Такие диаграммы используются только в том случае, когда необходимо проиллюстрировать конкретный способ реализации.

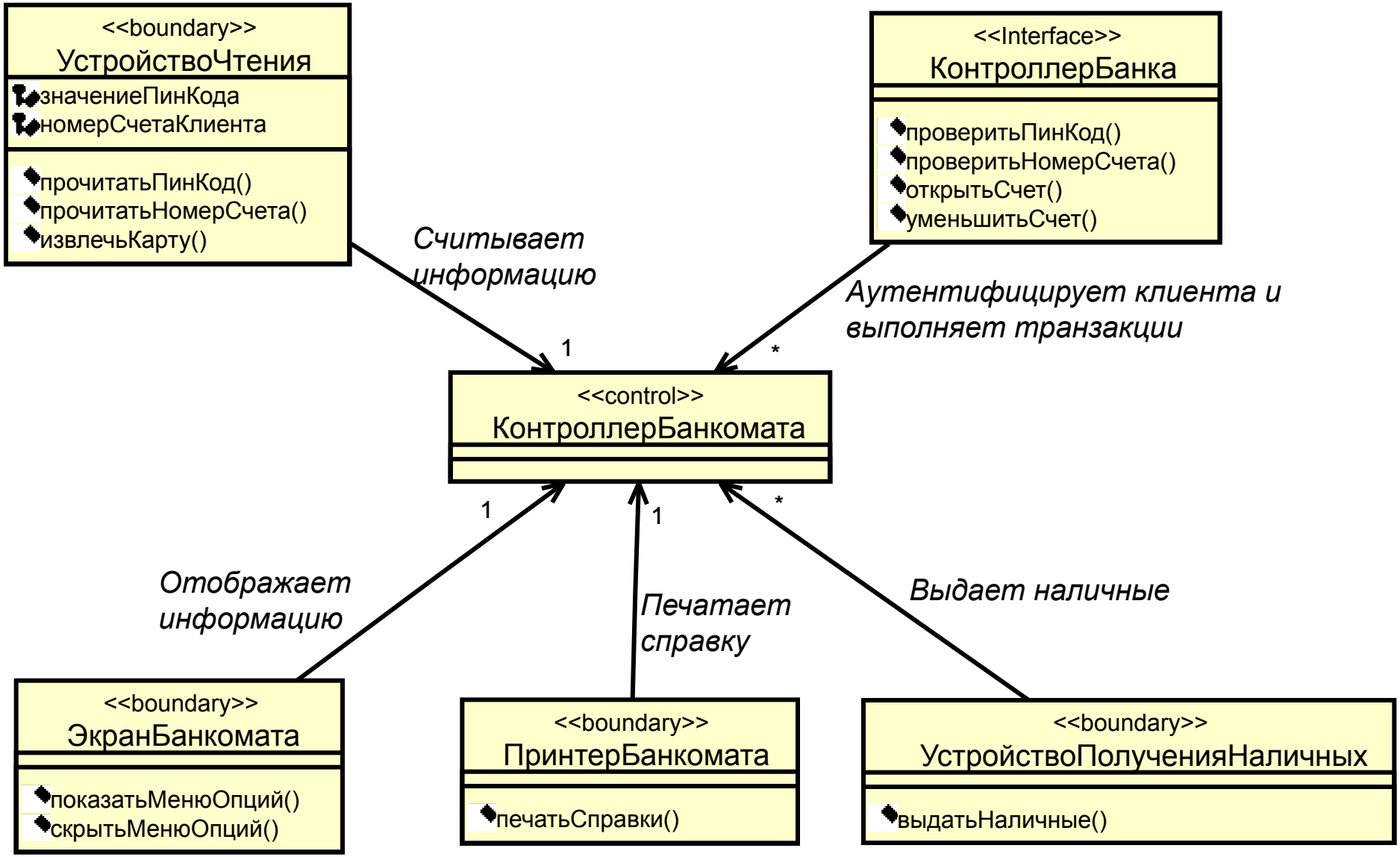
Точки зрения на диаграмму классов

Для представления архитектуры системы разрабатывается столько диаграмм классов, сколько требуется.

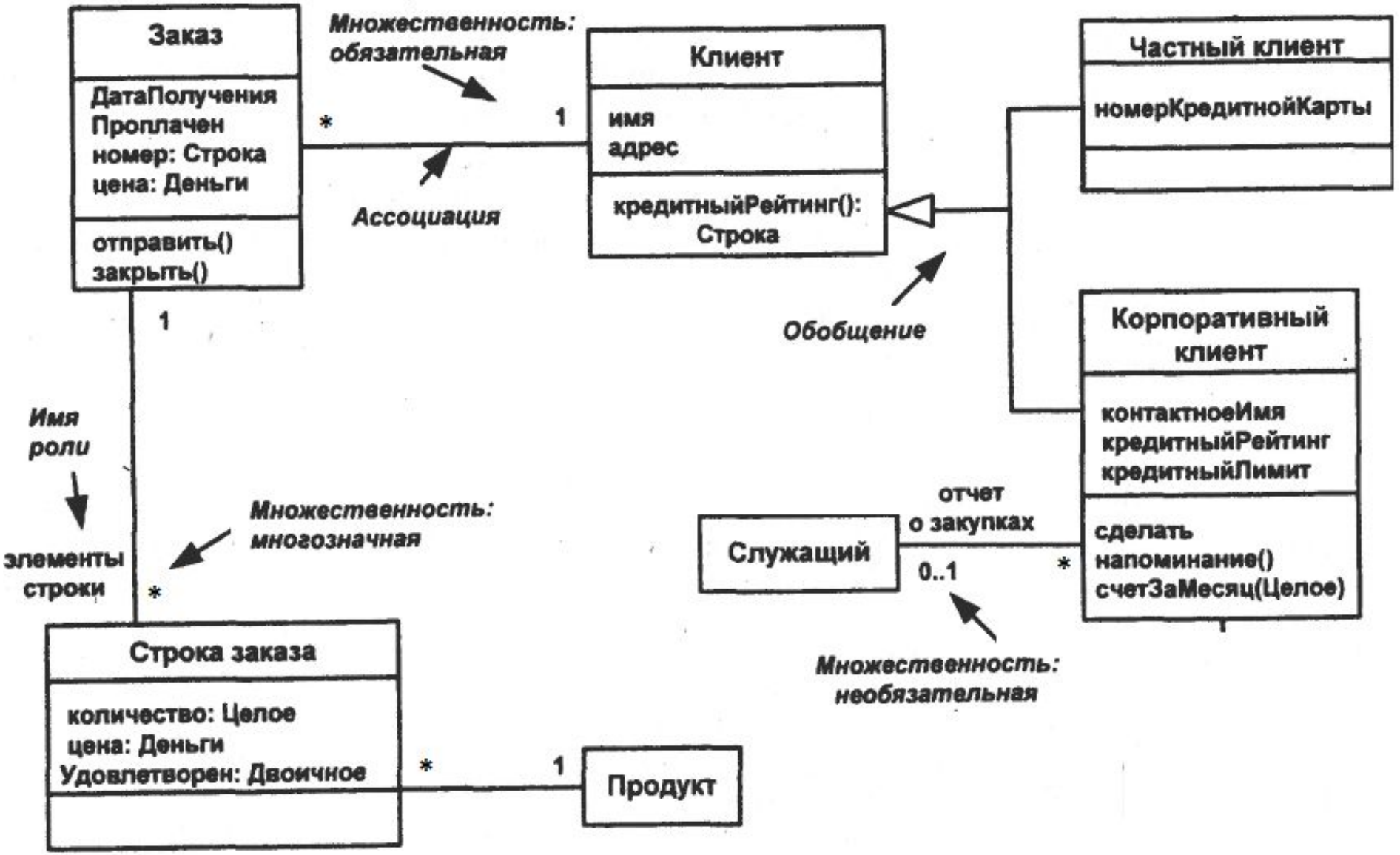
С их помощью разработчики могут видеть и планировать архитектуру еще до фактического написания кода.

4. Примеры диаграммы

Пример диаграммы классов

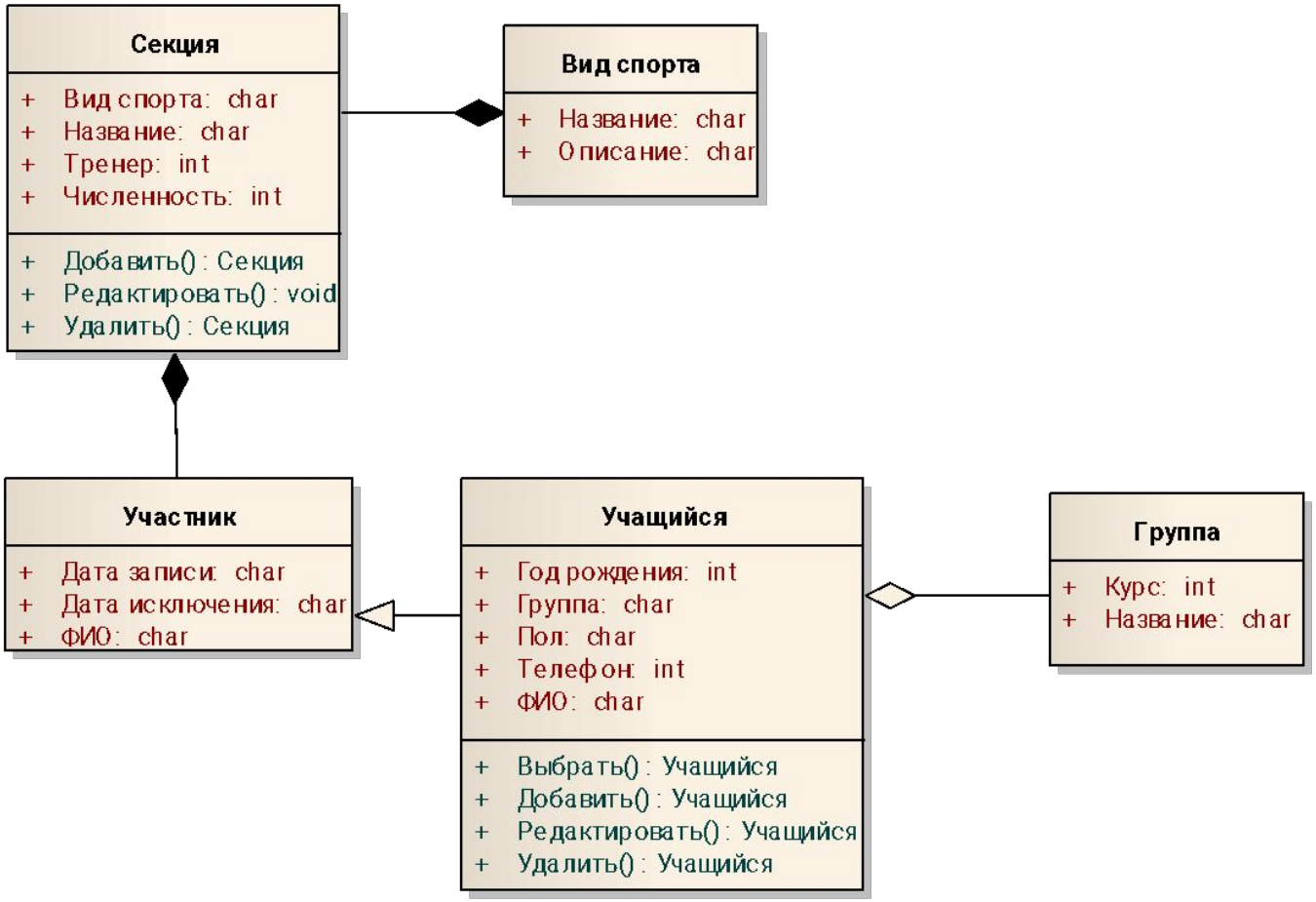


Пример диаграммы классов



Пример диаграммы классов с ошибками

Информационная система «Спортивные секции колледжа»



Пример диаграммы классов с ошибками

Информационная система «Пиццерия»