

Проектирование отношений



Связи

- ▶ Связь представляет собой семантическую взаимосвязь между классами
- ▶ ассоциации
 - ▶ Агрегации
 - ▶ Композиции
- ▶ зависимости
- ▶ обобщения (наследования)

► **UML (Unified Modeling Language)** -
унифицированный язык моделирования



► Классы

секция имени

секция атрибутов

видимость ИМЯ кратность : тип = начальное_значение
{свойства}

секция операций

ClassName

```
+ attribute
- privateAttr
- fio : String = "Novikov"
- array : char [10]

+ operationName ()
- staticOperation ()
+ function () : int
```

Стандартные стереотипы классов

Стереотип	Описание
«actor»	Действующее лицо
«auxiliary»	Вспомогательный класс
«enumeration»	Перечислимый тип данных
«exception»	Исключение (только в UML 1)
«focus»	Основной класс
«implementationClass»	Реализация класса
«interface»	Все составляющие абстрактные
«metaclass»	Экземпляры являются классами
«powertype»	Метакласс, экземплярами которого являются все наследники данного класса (только в UML 1)
«process»	Активный класс
«thread»	Активный класс (только в UML 1)
«signal»	Класс, экземплярами которого являются сигналы
«stereotype»	Новый элемент на основе существующего
«type»	Тип данных
«dataType»	Тип данных
«utility»	Нет экземпляров, служба

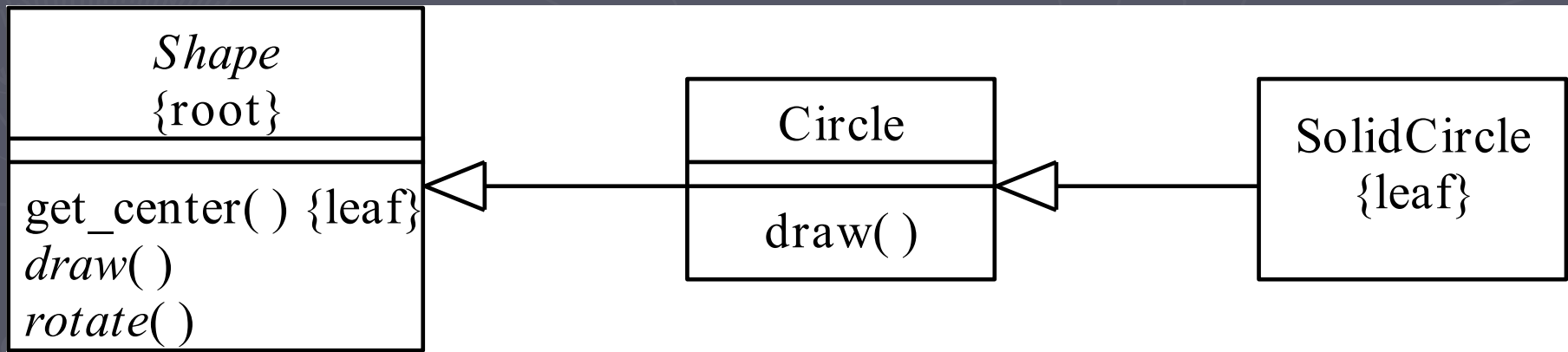
- *открытый* (обозначается знаком + или ключевым словом public);
- *защищенный* (обозначается знаком # или ключевым словом protected);
- *закрытый* (обозначается знаком - или ключевым словом private);
- *пакетный* (обозначается знаком ~ или ключевым словом package).

«utility»
Company

а) Обобщение

Генерализация-расширение-наследование

- ▶ показывают связи наследования между классами
- ▶ Является IS A

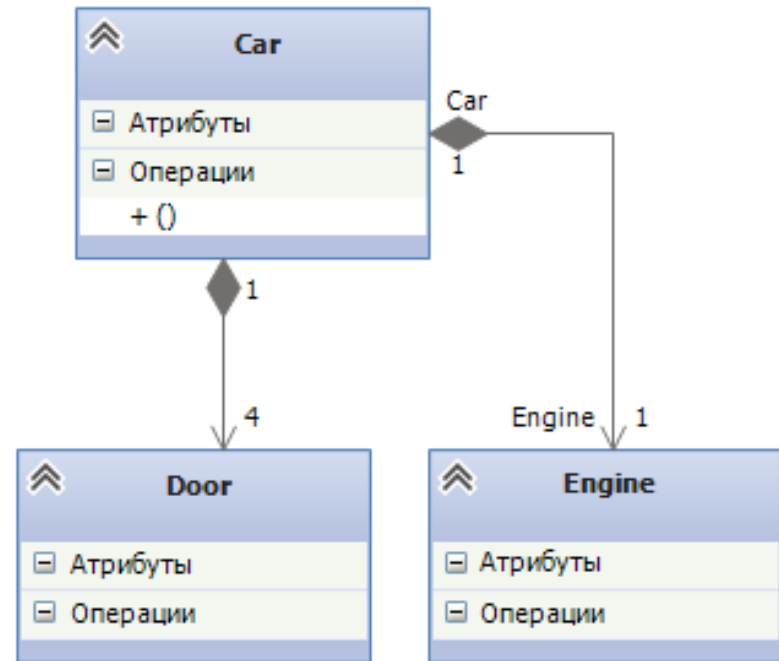


б) КОМПОЗИЦИЯ

Композиция (composition) – это ассоциация между классом А (часть) и классом В (целое), которая дополнительно накладывает более сильные ограничения в сравнении с агрегацией: композиционно часть А может входить только в одно целое В, часть существует, только пока существует целое и прекращает свое существование вместе с целым.

Состоит из
Отношение
«содержит как
часть»
HAS-A

Целое контролирует
время жизни своей
составной части (часть
не существует без
целого) - сильно
связана

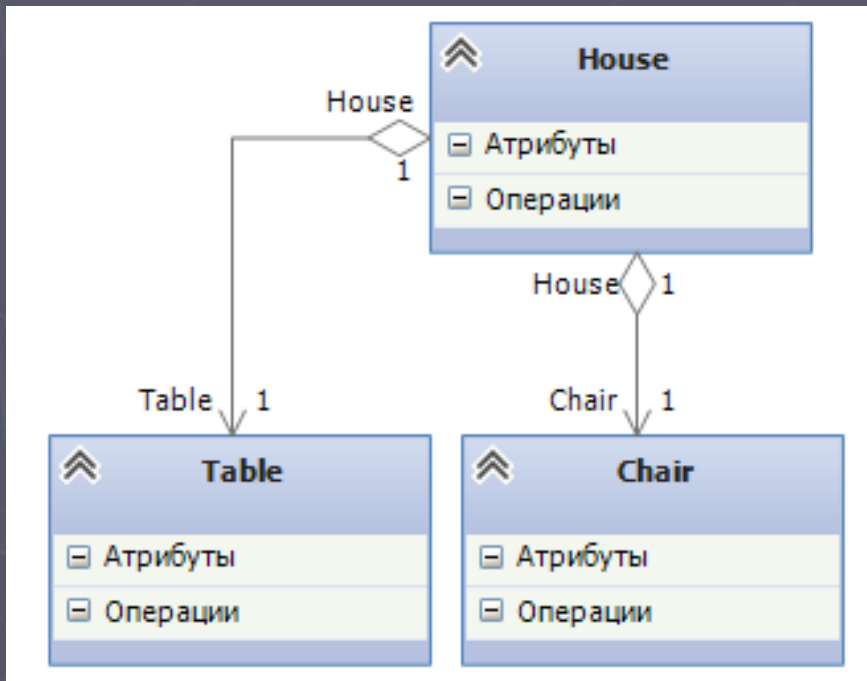


**Целое контролирует время жизни
своей составной части (часть не
существует без целого) - сильно связана**

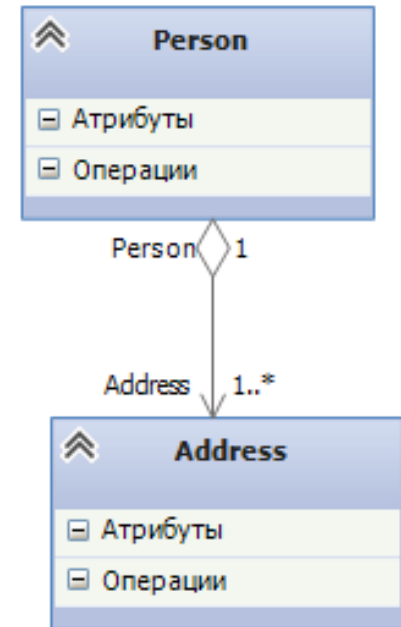


В) Агрегация

- СВЯЗЬ МЕЖДУ ЦЕЛЫМ И ЕГО ЧАСТЬЮ
- Включает в себя - contains
- Объекты м.б. равноправные



Целое хоть и содержит свою составную часть, время их жизни не связано

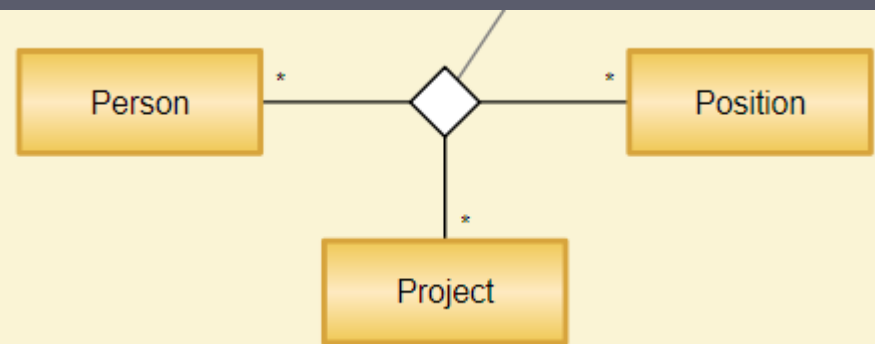


Агрегация (aggregation) – это ассоциация между классом А (часть) и классом В (целое), которая означает, что экземпляры (один или несколько) класса А входят в состав экземпляра класса В.

**Целое хоть и содержит свою
составную часть, время их жизни не
связано**



► Многополюсная ассоциация



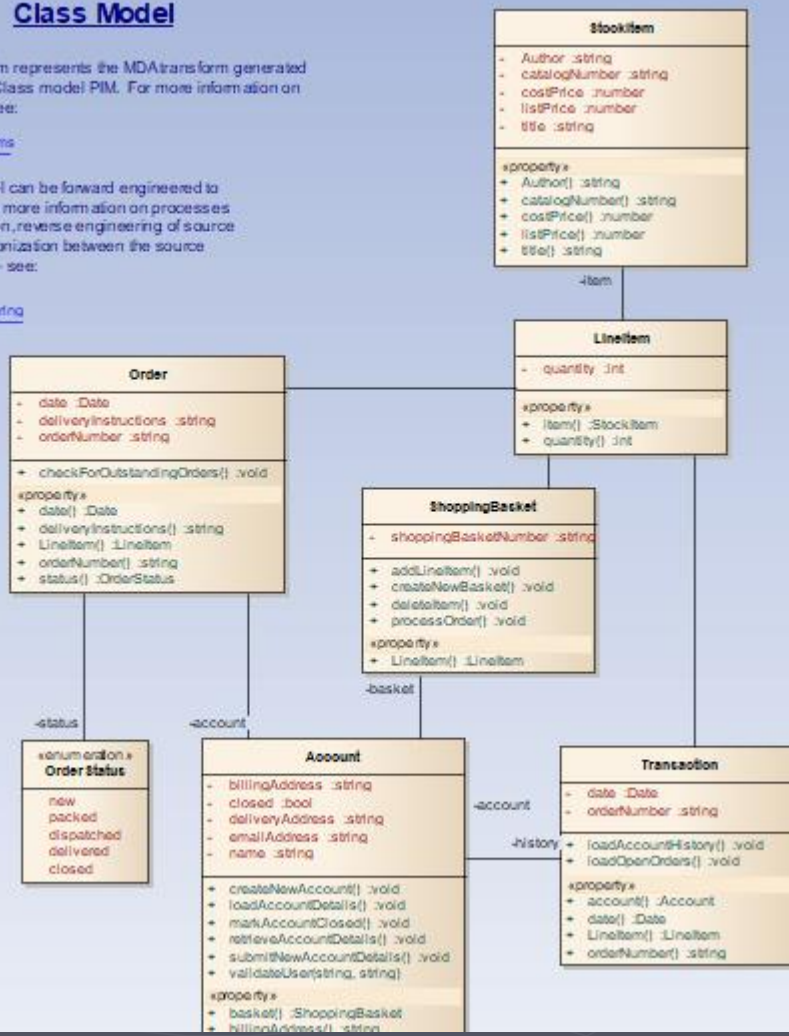
Class Model

This Class diagram represents the MDA transform generated from the Abstract Class model PIM. For more information on MDA transforms see:

[MDA Transforms](#)

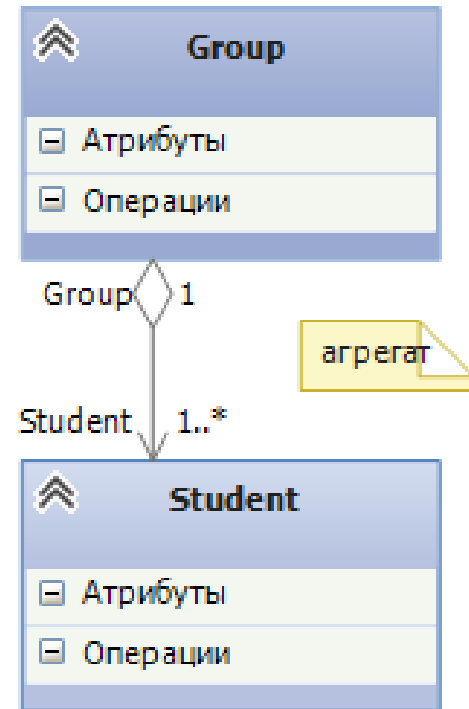
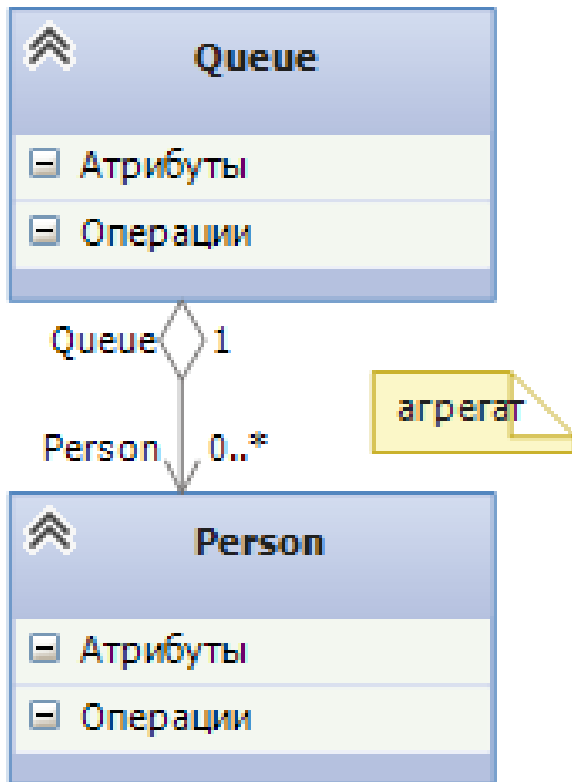
This Class model can be forward engineered to the C# code. For more information on processes of code generation, reverse engineering of source code and synchronization between the source code and model - see:

[Code Engineering](#)



?

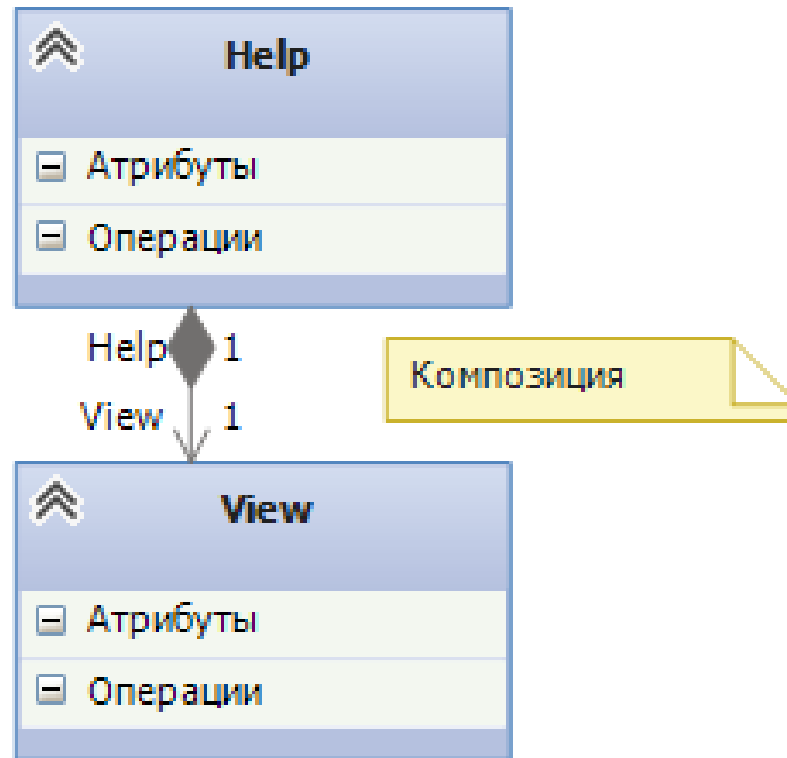
Группа – студент



Человек - очередь

?

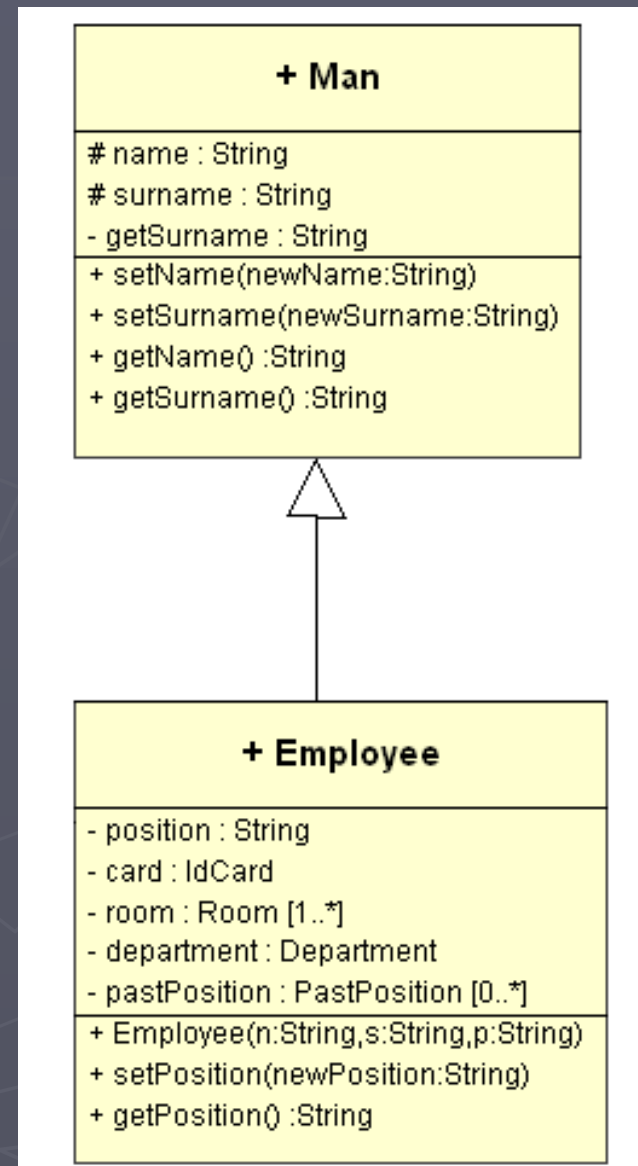
Помощь – окно просмотра



?

► Человек – работник

Человек – цех- завод



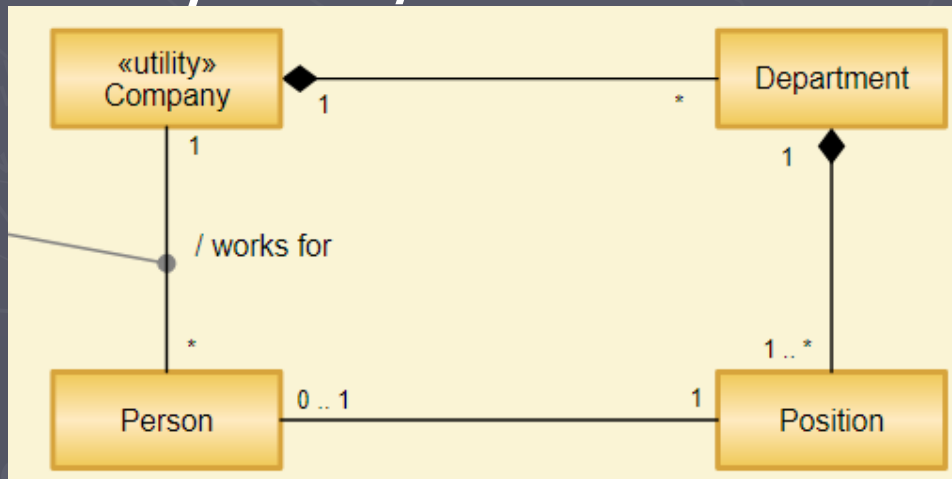
Отличие

- ▶ **композиция** может быть частью одного и только одного целого
- ▶ **агрегация** может быть частью нескольких объектов.

Г) Ассоциации и их дополнения

экземпляры одного класса связаны с экземплярами другого класса

- Имя ассоциации
- Кратность полюса ассоциации
- Агрегация и композиция



Ассоциация показывает отношения между объектами-экземплярами класса двусторонние или односторонние



+ Employee

- position : String
- card : IdCard
- room : Room [1..*]
- department : Department
- pastPosition : PastPosition [0..*]

+ Employee(n:String,s:String,p:String)
+ setPosition(newPosition:String)
+ getPosition() :String
+ setIdCard(newIdCard:IdCard)
+ getIdCard() :IdCard
+ setRoom(newRoom:Room)
+ getRoom() :Room[1..*]
+ deleteRoom(r:Room)
+ setDepartment(d:Department)
+ getDepartment() :Department
+ setPastPosition(p:PastPosition)
+ getPastPosition() :PastPosition[1..*]
+ deletePastPosition(p:PastPosition)

+ выдана

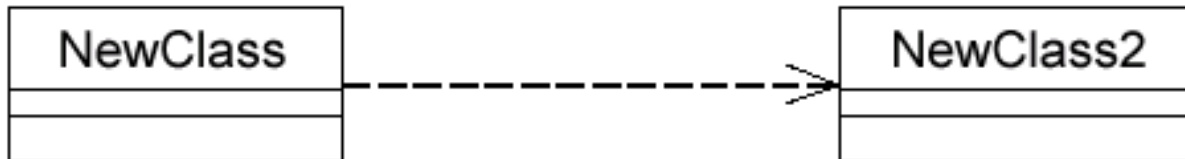
+ 1..1 + 1..1

+ IdCard

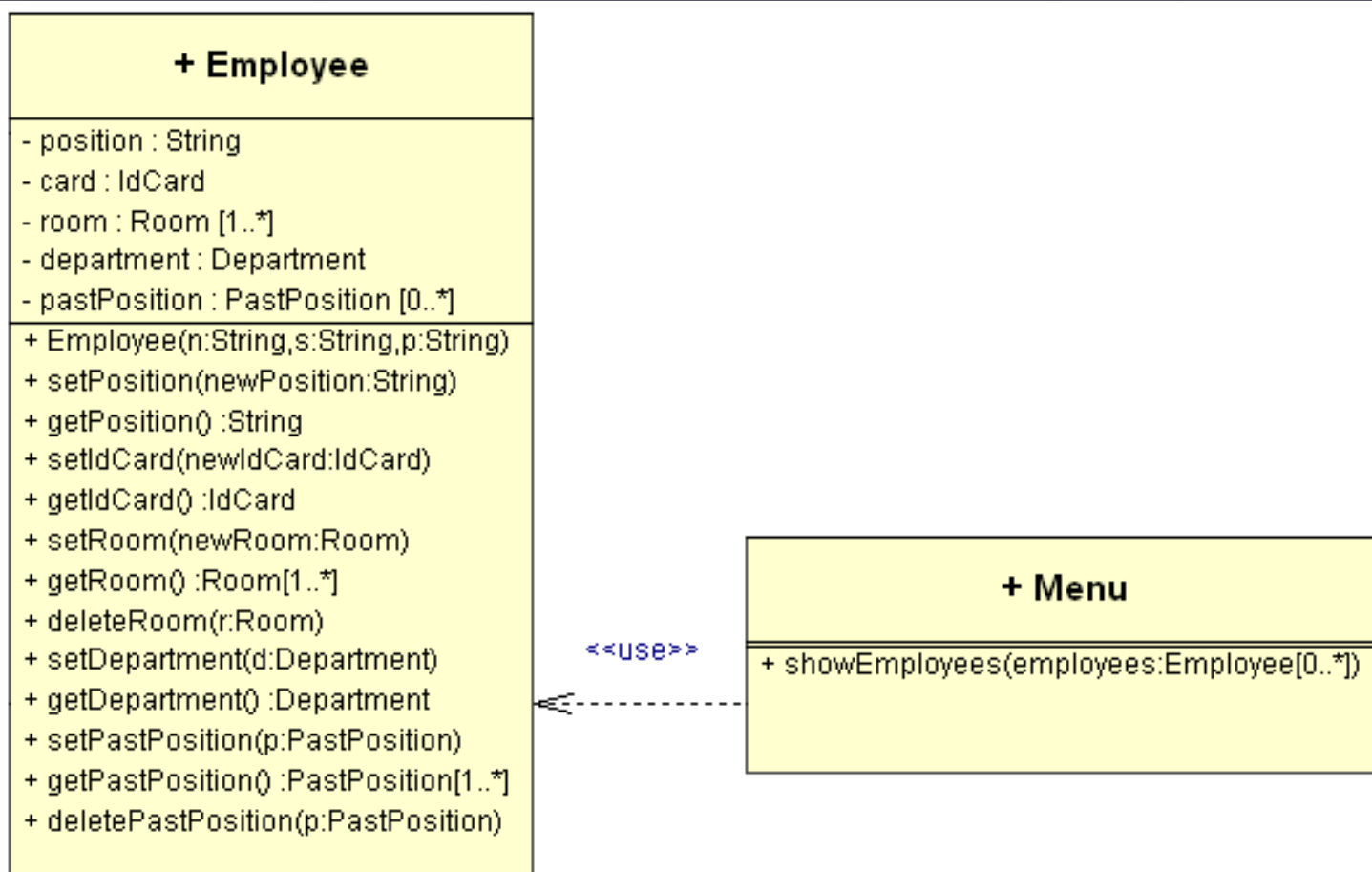
- number : int
- dateExpire : String
+ IdCard(n:int)
+ setNumber(newNumber:int)
+ getNumber() :int
+ setDateExpire(newDate:Date)
+ getDateExpire() :Date

д) Зависимость

- ▶ всегда однонаправленны

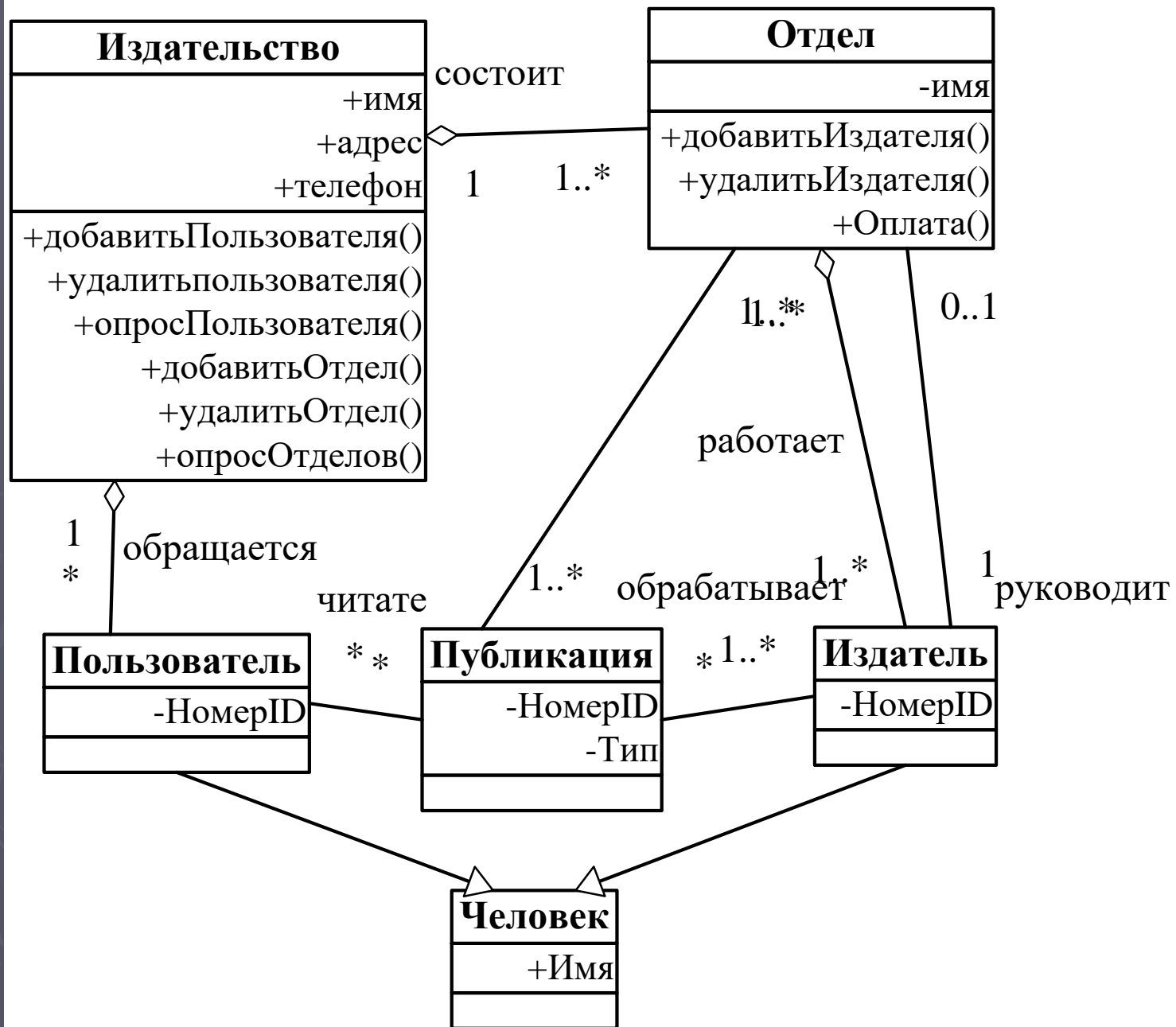


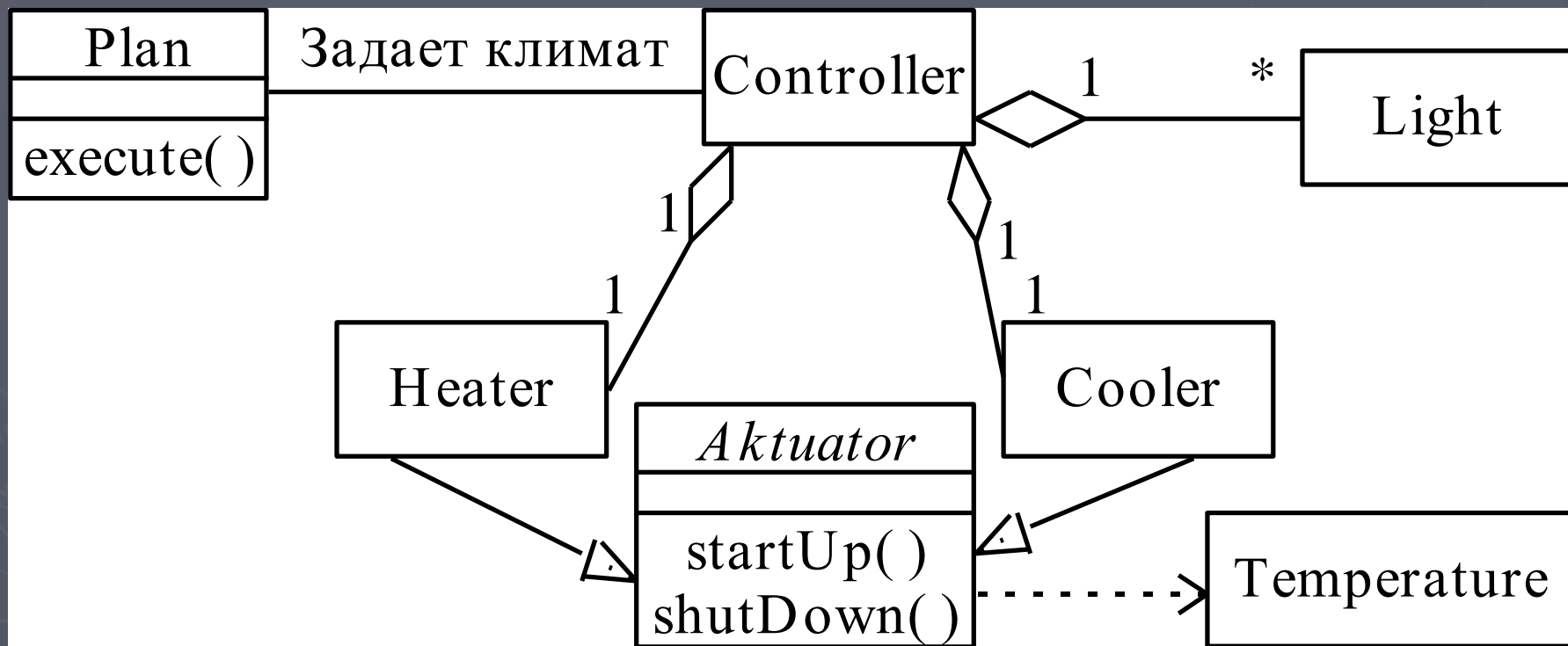
► Работники - отображения



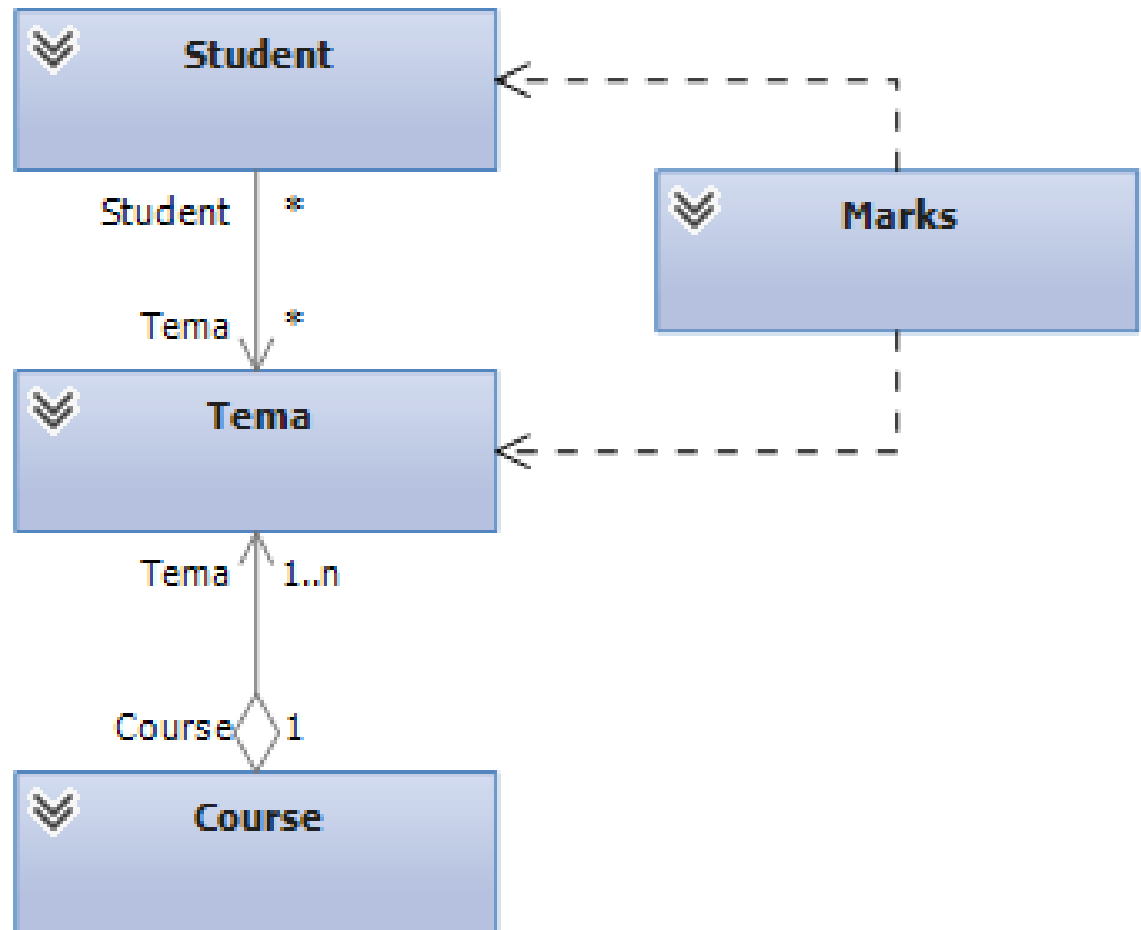
Типы отношений между классами

- ▶ обобщение\специализация (общее-частное)
- ▶ целое\часть
- ▶ семантическое, смысловые отношения или ассоциации





- ▶ Студент
- ▶ Тема
- ▶ Курс
- ▶ Оценки



Диаграммы классов (принципы)

- ▶ Описывать структуру удобнее параллельно с описанием поведения.
- ▶ Каждая итерация должна быть небольшим уточнением
- ▶ Не обязательно включать в модель все классы сразу
- ▶ Не обязательно определять все составляющие класса сразу.
- ▶ Не обязательно определять все отношения между классами сразу