

# Анализ и проектирование на UML

Новиков Федор Александрович

[fedornovikov@rambler.ru](mailto:fedornovikov@rambler.ru)

Курс подготовлен по заказу  
ООО Сан Майкросистемс СПб

## Часть 4

Курс подготовлен при поддержке Sun Microsystems  
Правила использования материалов опубликованы на [www.sun.ru](http://www.sun.ru)

# План лекций

- Введение в UML
- Обзор языка
- Моделирование использования
- ✓ Моделирование структуры
- Моделирование поведения
- Управление моделями
- Тенденции развития языка
- UML и процесс разработки

# 4. Моделирование структуры

- 4.1. ОО статическое моделирование
- 4.2. Классификаторы
- 4.3. Элементы диаграмм классов
- 4.4. Идентификация классов
- 4.5. Отношения на диаграммах классов
- 4.6. Интерфейсы, типы данных и роли
- 4.7. Внутренняя структура объектов
- 4.7. Диаграммы компонентов
- 4.8. Диаграммы размещения → развертывания

# 4.1. ОО статическое моделирование

- **Дескрипторы**
- **Отступление про ООП**
- **Моделирование структуры:  
Какой? Чего?**

# Дескрипторы

- Модель статична и существенно конечна, выполнение динамично и практически бесконечно – структуру чего описывает модель?
- *Дескриптор* – конечное описание (intent) бесконечного множества (extent)
- Литерал – самоопределенное значение
- Почти все элементы модели (классы, варианты использования, узлы, ассоциации ...) суть дескрипторы
- Примечания – литералы

# ООП

- **Объект – инкапсулирует данные и операции для работы с ними**
- **Класс – описание однотипных объектов**
- **Объект = экземпляр (instance) класса**
- **Наследование (inheritance) – способ структуризации описаний классов**
- **Полиморфизм (polymorphism) – соглашение о способе идентификации операций (по сигнатуре, а не по имени)**
- **Каждый объект является непосредственным экземпляром одного класса и косвенным экземпляром всех суперклассов**

# Моделирование структуры: Какой? Чего?

- Структура связей объектов во время выполнения
  - Ассоциации на диаграмме классов
- Структура сложных объектов, состоящих из взаимодействующих частей
  - Диаграмма внутренней структуры классификатора
- Структура хранения данных
  - Дополнения ассоциаций
- Структура программного кода
  - Обобщения
- Структура артефактов в проекте
  - Диаграммы компонентов
- Структура компонентов в приложении
  - Интерфейсы и классы на диаграмме компонентов
- Структура используемых вычислительных ресурсов
  - Диаграммы размещения → развертывания

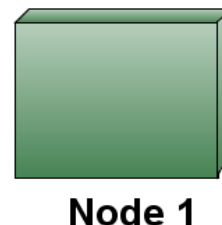
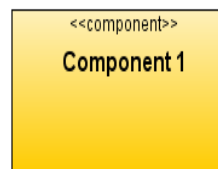
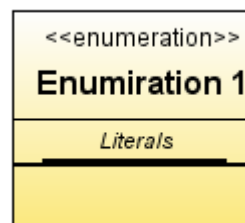
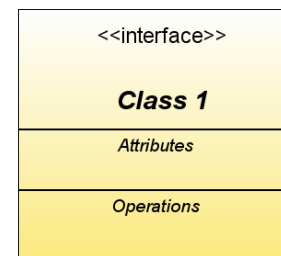
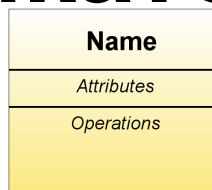
## 4.2. Классификаторы

- **Классификатор (classifier)** – сущность, имеющая экземпляры
- **Классификаторы UML 1.4**
  - **Класс**
  - **Интерфейс** = множество абстрактных операций
  - **Тип данных**: примитивы и enum
  - **Сигнал (signal)**
  - **Компонент (component)**
  - **Узел (node)**
  - **Вариант использования (use case)**
  - **Подсистема (subsystem)**
- **Не классификаторы** – пакет, обобщение
  - **хотя ассоциация имеет экземпляры**



# Классификаторы UML 2.0

- Класс (class)
- Артефакт (artifact)
- Интерфейс (interface)
- Прimitivesкий тип данных (data type)
- Перечисление (enumeration)
- Сигнал (signal)
- Компонент (component)
- Узел (node)
- Вариант использования (use case)
- Действующее лицо (actor)
- Кооперация (cooperation)
- Роль (role)



# Общие свойства классификаторов

- Видимость (visibility)
  - Не только у классификаторов
- Область действия (scope)
  - Не только у классификаторов
- Обобщение (generalization)
  - Только для классификаторов
  - Наследование (inheritance)
- Кратность (multiplicity)
  - Не только у классификаторов

# Видимость

- ***Видимость*** – может ли свойство одного классификатора использоваться в другом классификаторе
- Открытый (+) **public**
- Защищенный (#) **protected**
- Закрытый (-) **private**

# Область действия

- **Область действия** – как проявляет себя свойство классификатора в экземплярах, т.е. имеют экземпляры индивидуальные значения свойства или совместно используют одно
- **Экземпляр (instance)** – по умолчанию
- **Классификатор (classifier)** – подчеркивается (аналогично static)

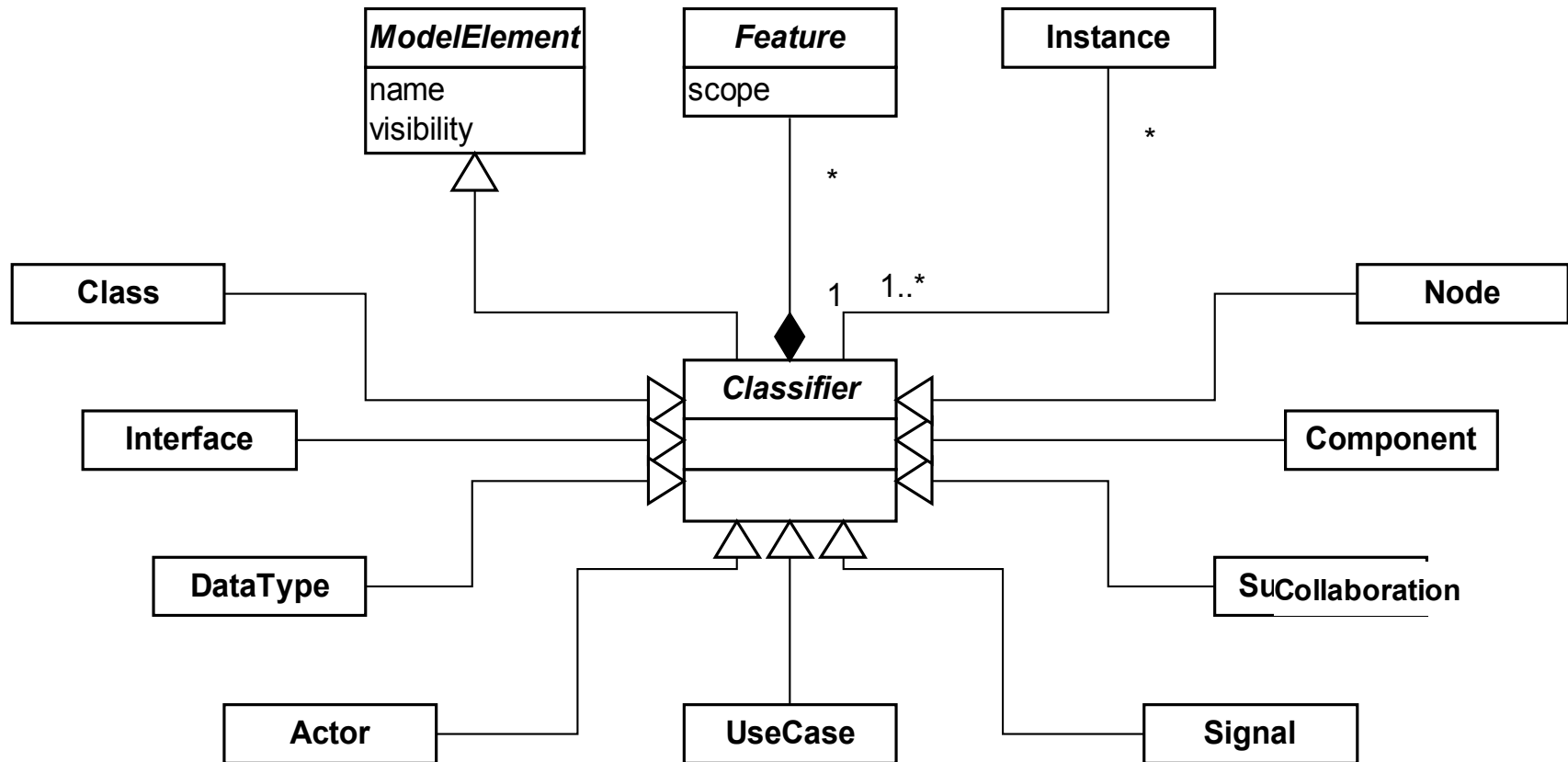
# Наследование

- **Абстрактный** (в UML) – не могущий иметь непосредственных экземпляров
  - Но наследники могут иметь экземпляры
- **Листовой** – не могущий иметь потомков (свойство {leaf})
  - Например, в листовом классе не может быть абстрактных операций
- **Корневой** – не могущий иметь родителей (свойство {root})
  - Используется, если есть несколько независимых иерархий наследования
- **Множественное наследование поощряется**
  - Конфликты наследования – проблема архитектора, а не языка

# Кратность

- Кратность (multiplicity) – ограничение (обычно сверху) на количество экземпляров
  - Вариант перевода – множественность
- Кратность м.б. у классификаторов, атрибутов и полюсов ассоциаций
- Нет экземпляров – служебный класс, служба (utility)
- Ровно один – Singleton (одиночка)
- Фиксированное число экземпляров
  - Например, порты в концентраторе
- Произвольное число экземпляров – по умолчанию
- Кратный атрибут = массив

# Метамодель классификаторов



# 4.3. Элементы диаграмм классов

## ■ Сущности:

- Класс (class)
  - Интерфейс (interface)
  - Тип данных (data type)
- Пакет (package)
- Примечание (note)

## ■ Отношения:

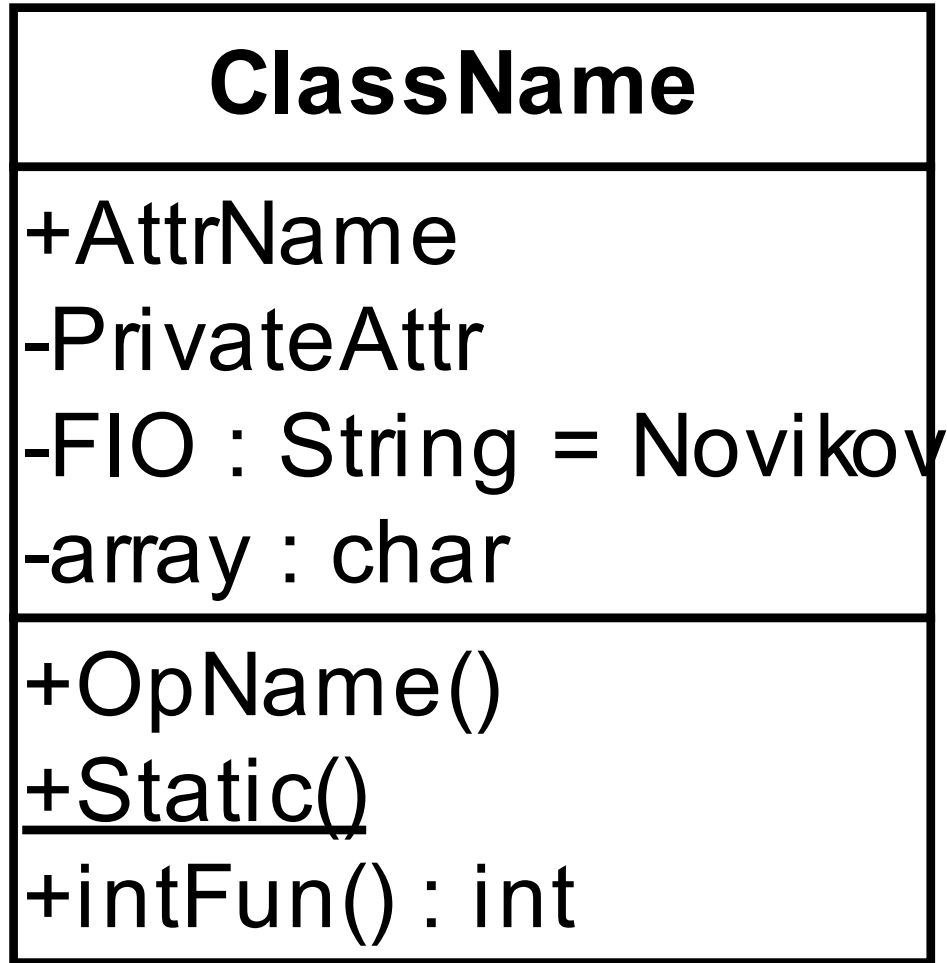
- Ассоциация (в т.ч. агрегирование)
- Обобщение (generalization)
  - Зависимость (dependency)
  - Реализация (implementation)



# Классы в UML

- **Раздел имени**
  - «стереотип» ИМЯ {свойства} кратность
  - Имя: *class*, *abstract*, object
- **Раздел атрибутов**
  - видимость ИМЯ кратность :тип  
= начальное\_значение {свойства}
  - видимость тип ИМЯ кратность  
= начальное\_значение {свойства}
- **Раздел операций**
  - видимость тип ИМЯ (параметры) {свойства}
- **Дополнительные разделы (или примечание)**

# Нотация



# Стандартные стереотипы классов

## (1.x)

- «metaclass» — экземпляры являются классами
- «powertype» — метакласс, экземплярами являются все наследники данного класса
- «stereotype» — стереотип
- «utility» — нет экземпляров = служба
- «interface» — нет атрибутов и все операции абстрактные
- «datatype» — тип данных
- «enumeration» — перечислимый тип данных
- «implementationClass» — реализация класса
- «actor» — действующее лицо
- «signal» — экземплярами являются сообщения
- «exception» — сигнал, распространяемый по иерархии обобщений
- «process», «thread» — активные классы

# Атрибуты

- видимость ИМЯ кратность : тип = начальное\_значение {свойства}
- видимость тип ИМЯ кратность = начальное\_значение {свойства}
- Видимость: + # - (public, protected, private)
- Тип: string, integer, ..., примитивы, перечисления {false, true}, имя типа или класса
- Подчеркивание = static в C++
- Кратность: [0..2], [1..\*]
- Свойства:
  - {changeable} – по умолчанию
  - {addOnly} – добавление значений в массив
  - {frozen} – после создания нельзя менять

# Примеры атрибутов

- **name**
- **+name**
- **+name : String**
- **+name [1..3] : String**
- **+name : String = “Novikov”**
- **+name : String {frozen}**

# Операции

- видимость ИМЯ (параметры) : тип {свойства}
- видимость тип ИМЯ (параметры) {свойства}
- направление передачи ПАРАМЕТР : тип = значение
- направления передачи: in, out, inout, return
- Подчеркивание = static в C++
- Свойства:
  - {isQuery} – функция без побочных эффектов
  - {sequential} – только один поток
  - {guarded} – в каждый момент один
  - {concurrent} – атомарность
- Группирование с помощью стереотипов

# Примеры операций

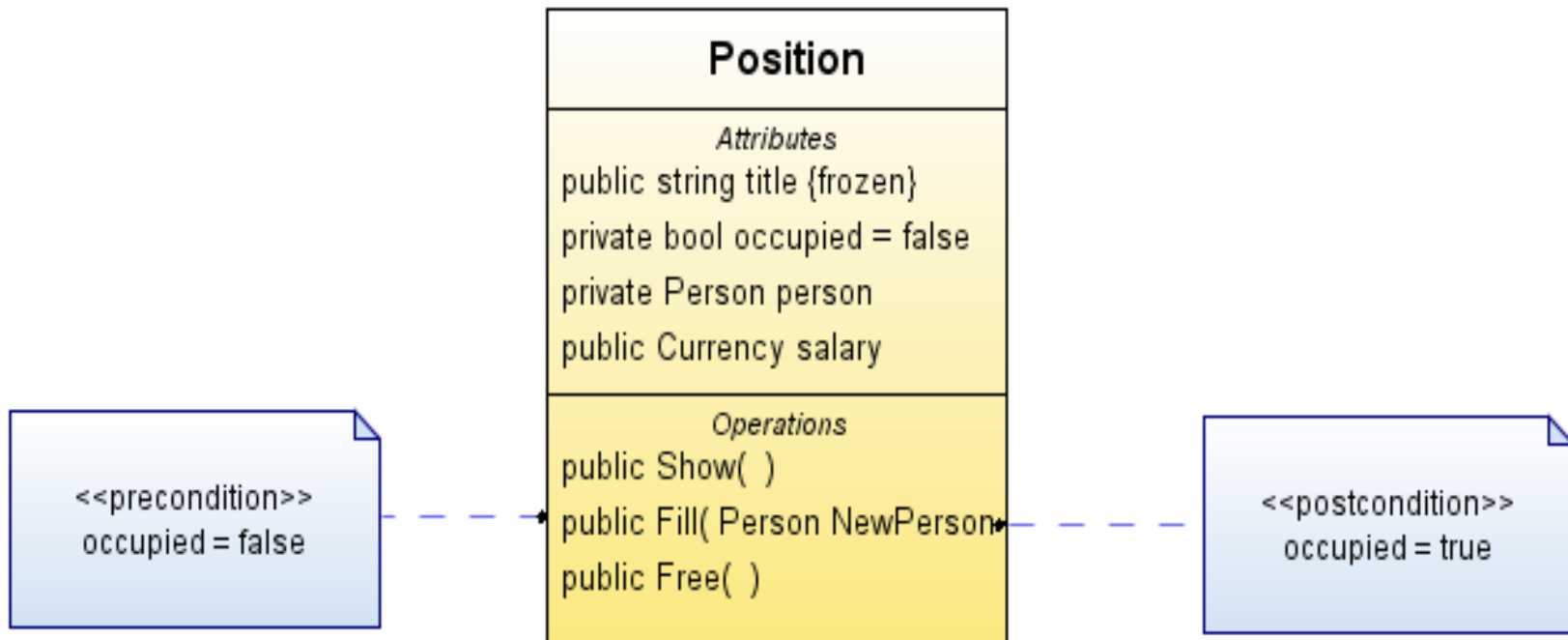
- **move**
- **+move(in from, in to)**
- **+move(from : Dpt, to : Dpt)**
- **+getName() : String {isQuery}**
- **+setPwd(in pwd : String = “password”)**

# Дополнительные разделы

- = Примечание к классу
- Примечания к операциям:  
предусловия и постусловия
- Неформальные, но полезные:
  - CRC – Class Responsibility Card
- Дополнительные разделы
  - Property get-set functions
  - Inner classes



# Пример ИС ОК



# Идентификация классов

- **Словарь предметной области**
  - **Имена существительные в ТЗ**
- **Реализация вариантов использования**
  - **Объекты на диаграммах взаимодействия**
  - **Объекты в состоянии на диаграммах деятельности (UML 1.x)**
- **Образцы (patterns) проектирования**
  - **Образец – параметризованная (по классам) кооперация**

# Словарь предметной области

- Прием, перевод и увольнение **сотрудников**
- Создание и ликвидация **подразделений**
- Создание **вакансий** и сокращение **должностей**

# Реализация вариантов использования

- Реализация HirePerson диаграммой использования
  - Person      Position      HireForm
- Реализация CreatePosition
  - Department
- Реализация CreateDepartment
  - Company

# Применение образца

- Образец View-Model Separation (Model- View-Controller)
  - Тривиальные соображения:
  - Диалоговое окно не должно прямо манипулировать данными
  - Класс, ответственный за хранение и обработку данных не должен содержать интерфейсных элементов
- HireForm

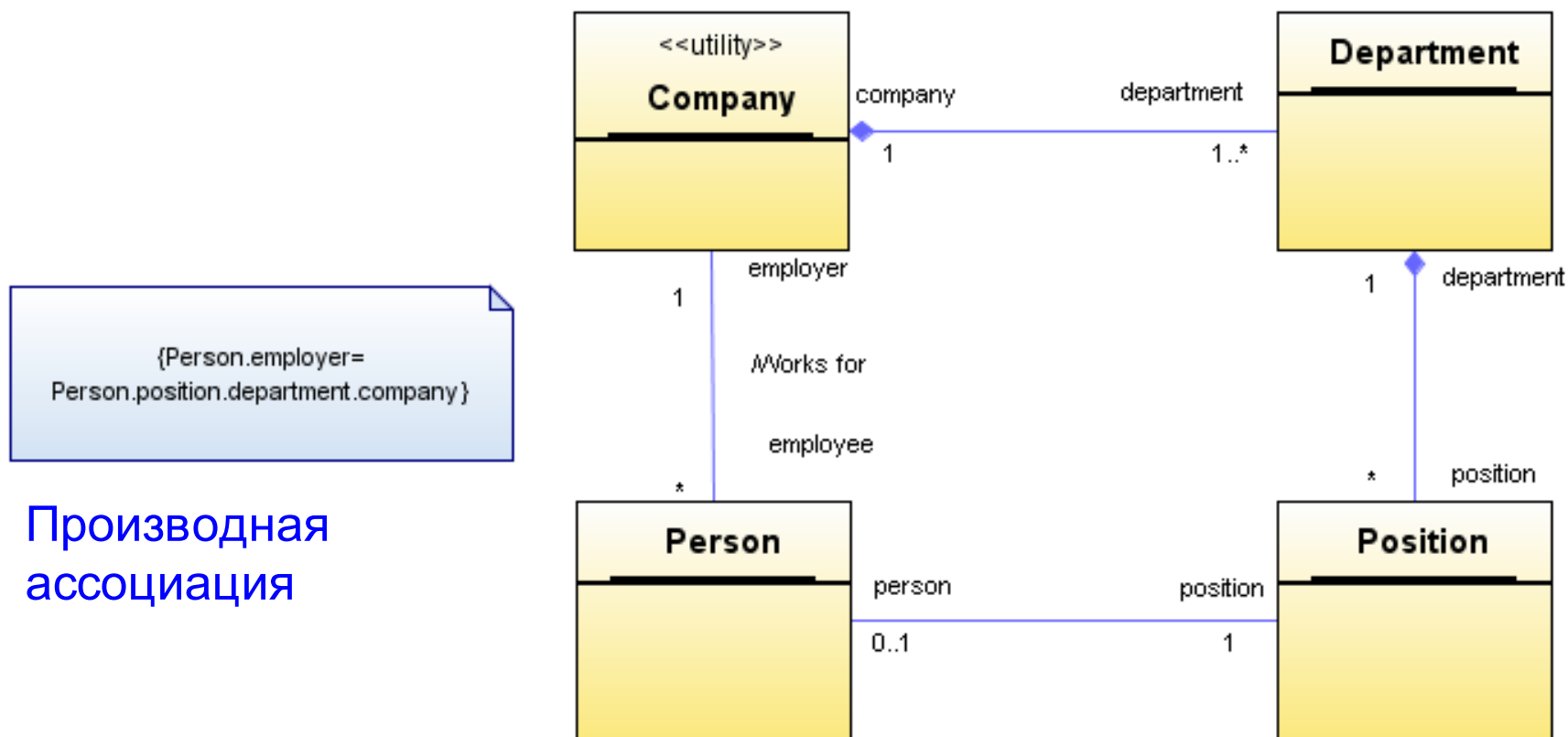
# Отношения

- **Зависимости:**
  - простые и стереотипные
- **Обобщение:**
  - включая множественное наследование
- **Ассоциация:**
  - включая агрегацию и композицию
- **Реализация:**
  - класс реализует интерфейс

# Стереотипные зависимости на диаграмме классов 1.x → 2.0

- «bind» – подставляет параметры в шаблон 🖐️56
- «derive» – производный элемент 🖐️🖐️32
- «friend» – имеет специальные права видимости 🖐️
- «instanceOf» – является экземпляром 🖐️35
- «instantiate» – создает экземпляры 🖐️35
- «powertype» – множество потомков 🖐️ 🖐️34
- «refine» – уточняет/конкретизирует ≈реализует 🖐️6.x
- «use» – использует ≈ зависимость без стереотипа

# Производные элементы (derive)



Производная  
ассоциация



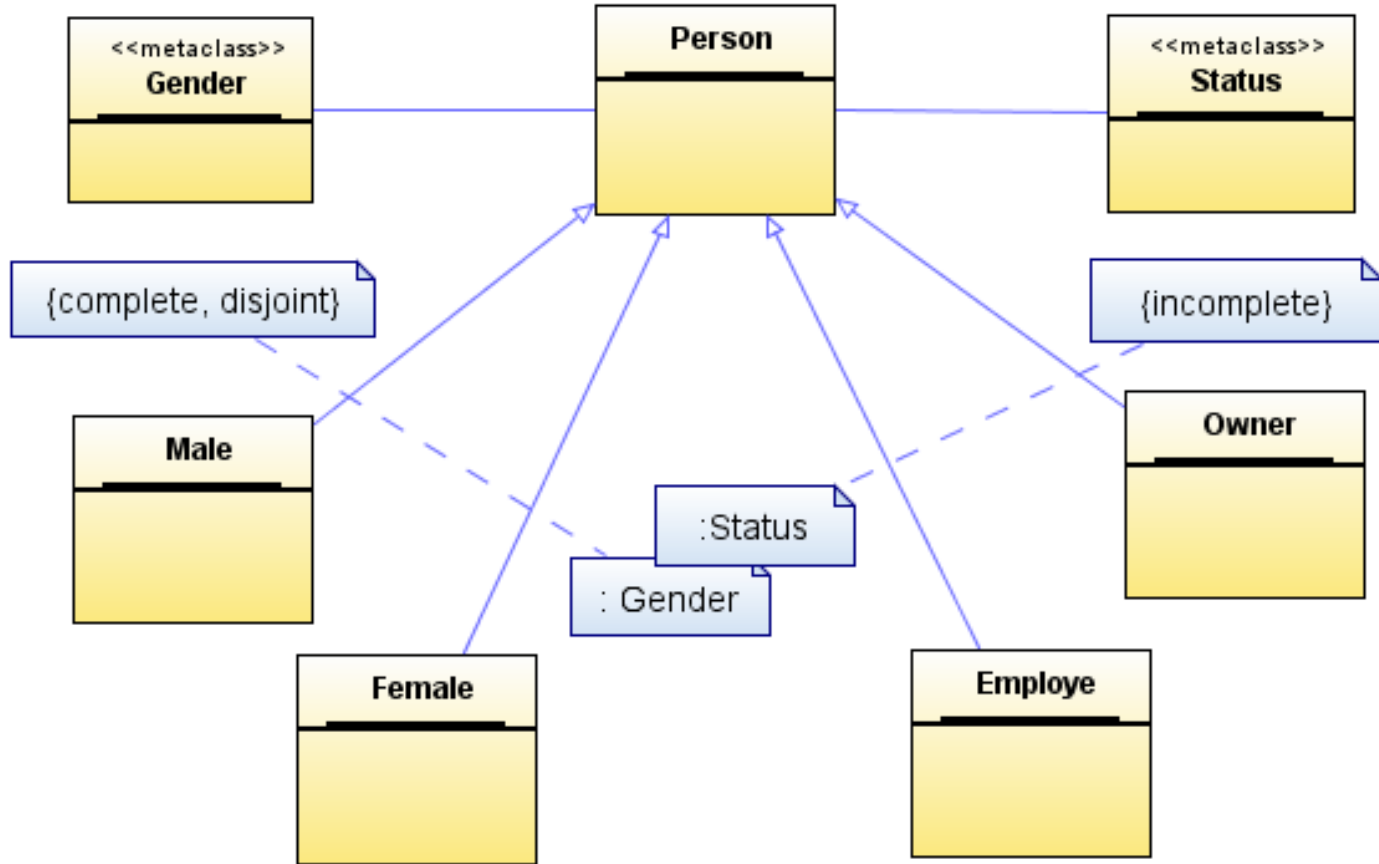


# Обобщение

- Стрелка с треугольником от подкласса к суперклассу
- Одноичное и множественное обобщение: т.е. наследование в UML не иерархия, а решётка
- Стереотип «implementation»
  - закрытое наследование C++
- Ограничения:
  - {complete} – иерархия завершена
  - {incomplete} – иерархия не завершена

# Множества обобщений и подтипов

## Множество подтипов

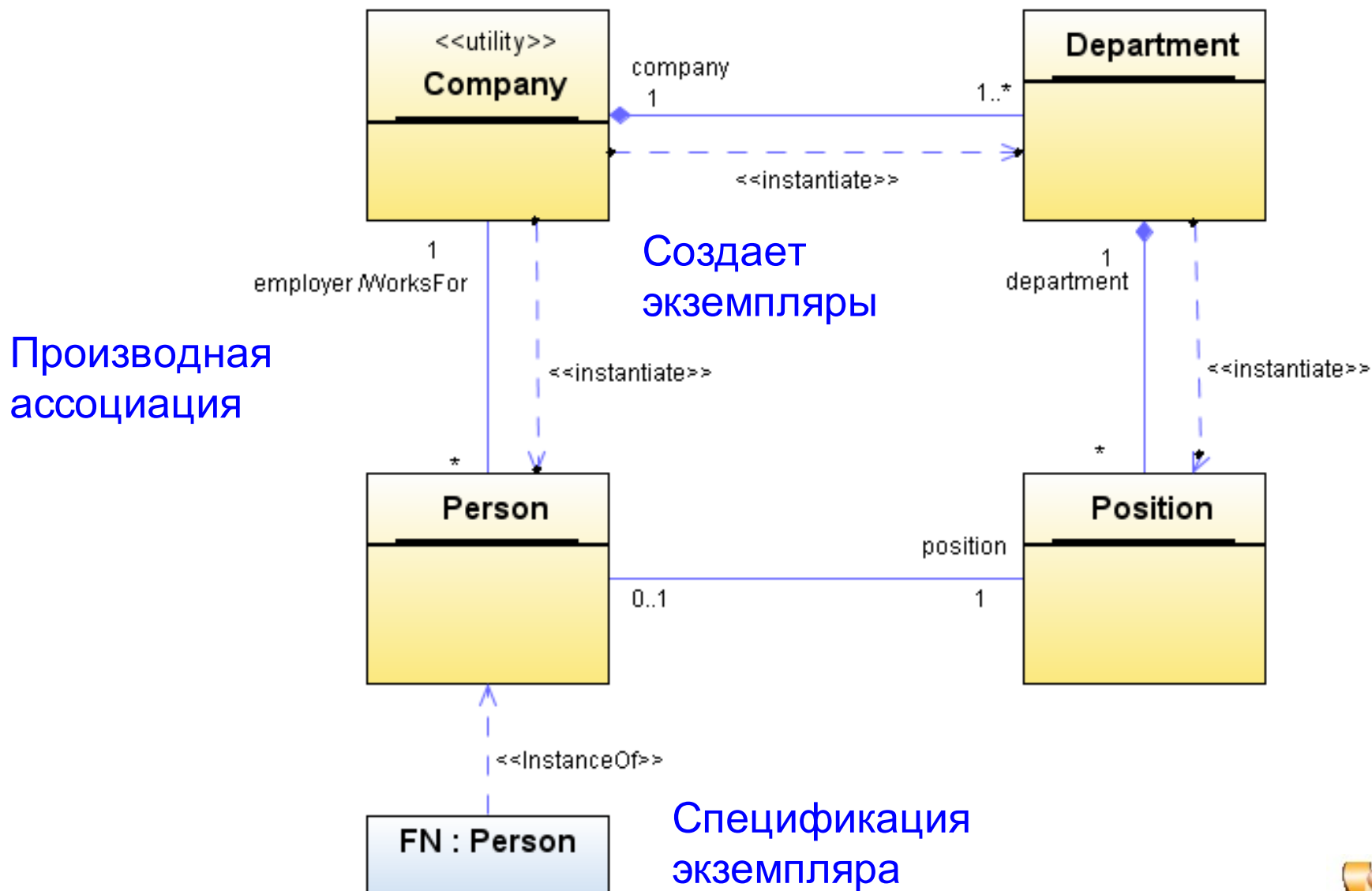


## Множество обобщений

## Множество обобщений



# Спецификация экземпляра

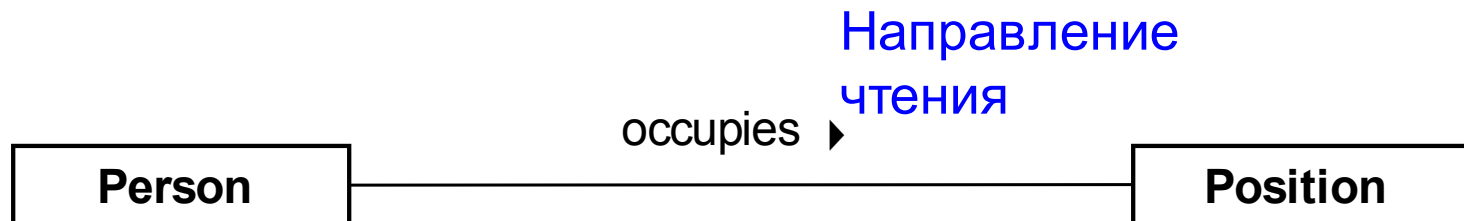
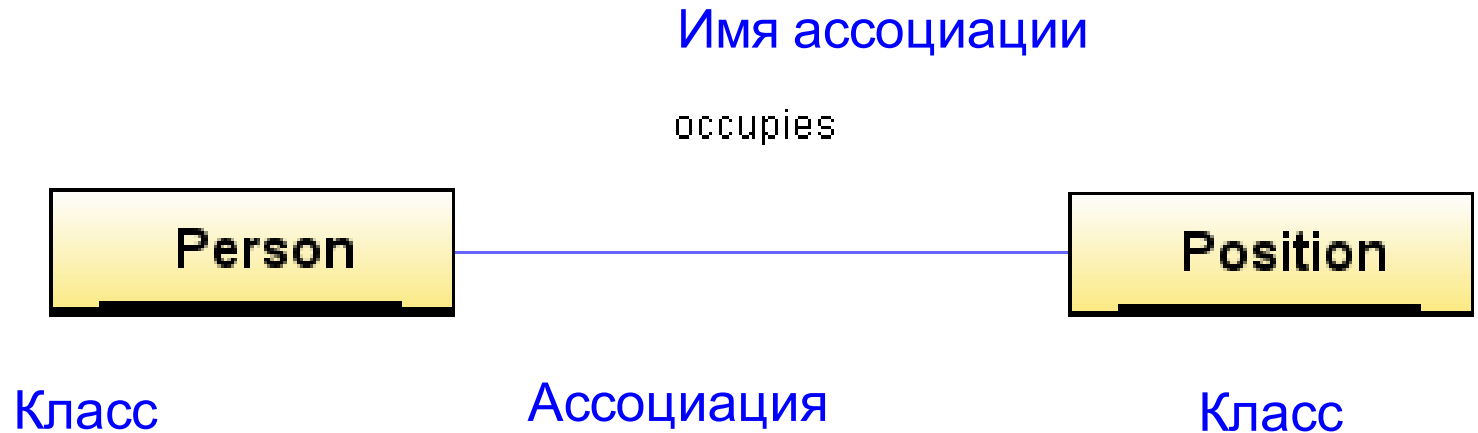


# Ассоциация

- имя ассоциации (возможно, вместе с направлением чтения) 📑37
- кратность полюса ассоциации 📑38
- вид агрегации полюса ассоциации 📑39-41
- роль полюса ассоциации 📑42
- направление навигации полюса ассоциации 📑43
- видимость полюса ассоциации 📑44
- упорядоченность объектов на полюсе ассоциации 📑45
- изменяемость множества объектов на полюсе ассоциации 📑45
- квалификатор полюса ассоциации 📑46
- класс ассоциации 📑47
- многополюсные ассоциации 📑48

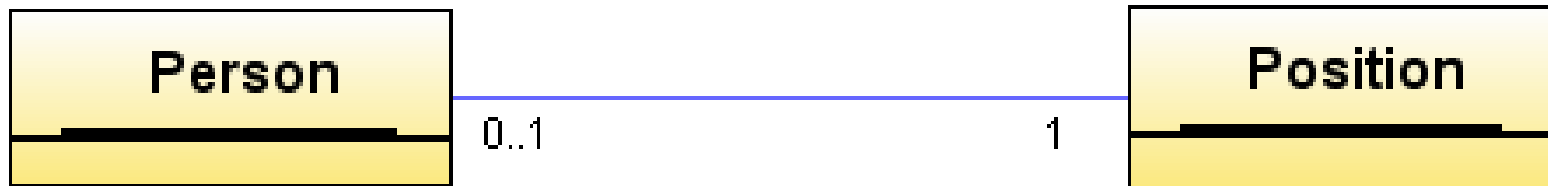
# Имя ассоциации

- Обычно если  $> 2$  полюсов
- Можно указать направление чтения



# Кратность полюса ассоциации

- Сколько экземпляров объектов может быть на данном полюсе связи
- \* - неопределенное число



Кратность задана диапазоном

Кратность задана  
числом



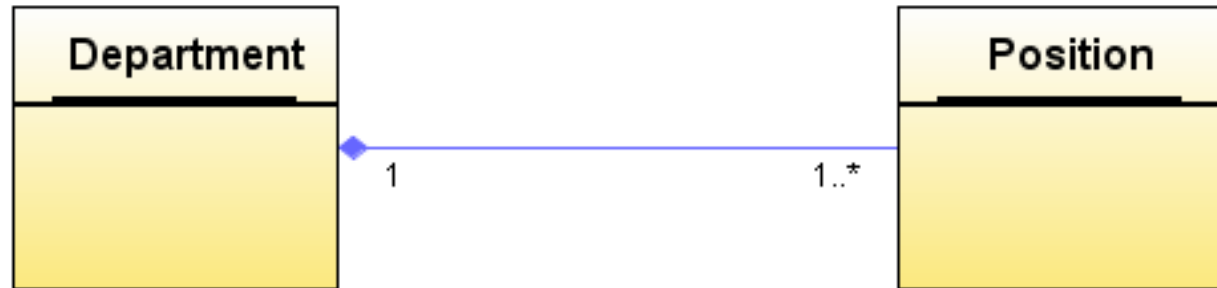
# Агрегация и композиция

- **Агрегация**
  - **Отношение Часть → Целое**
  - **Не изменяет направления навигации и не накладывает ограничений на время жизни**
- **Композиция**
  - **Буквально «состоит из»**
  - **Часть принадлежит только одному целому**
  - **Время жизни частей совпадает с временем жизни целого**
- **Встраивание и ссылка на объект**

# Примеры композиции и агрегации

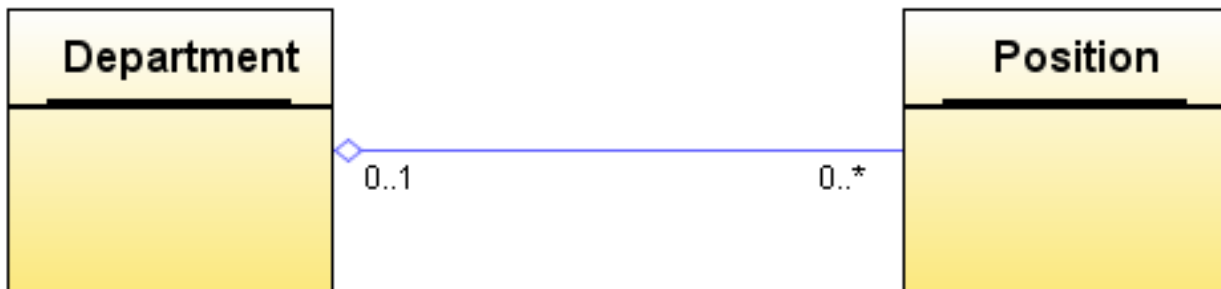
## Композиция

Жесткая  
структура



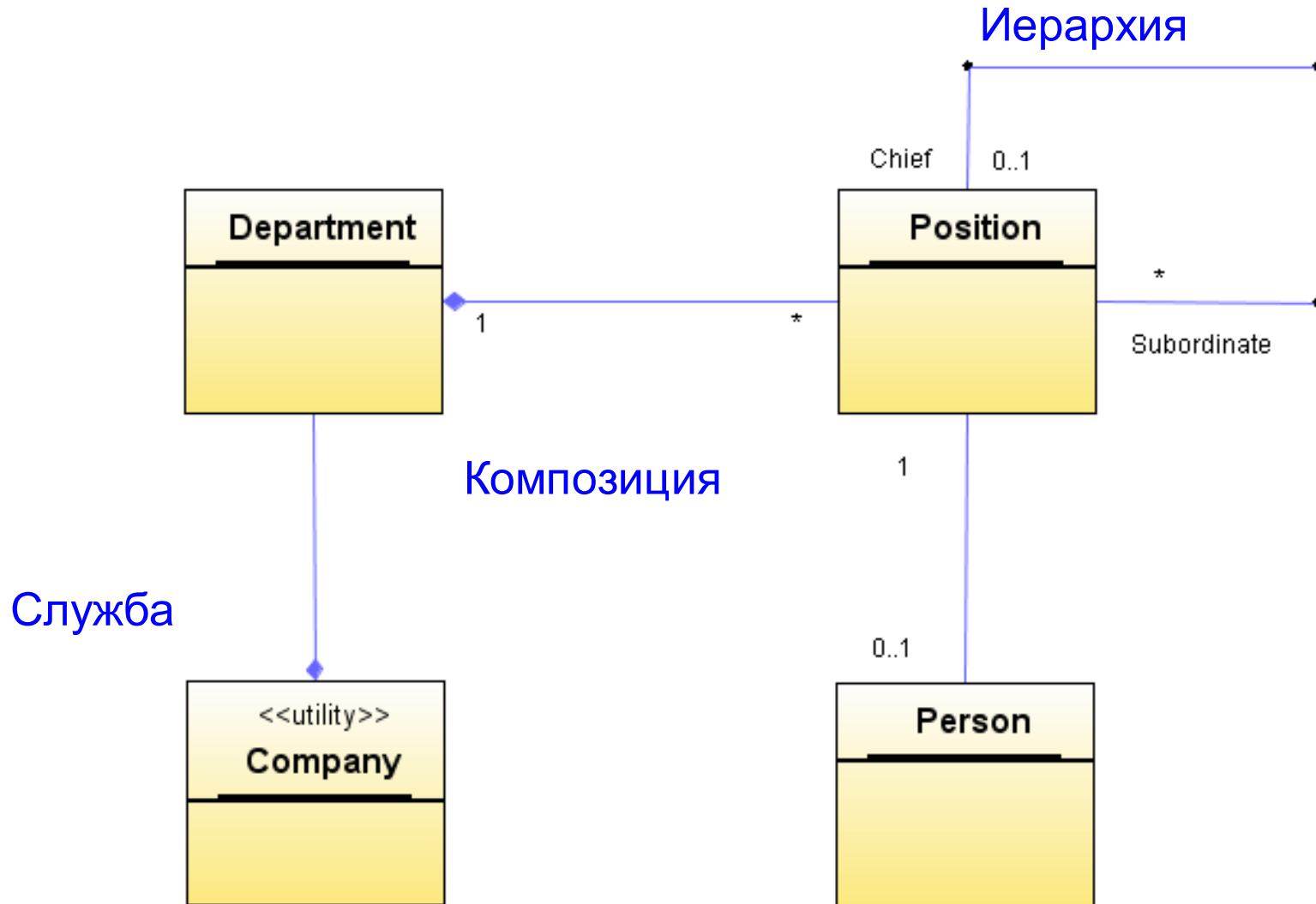
## Агрегация

Мягкая  
структура



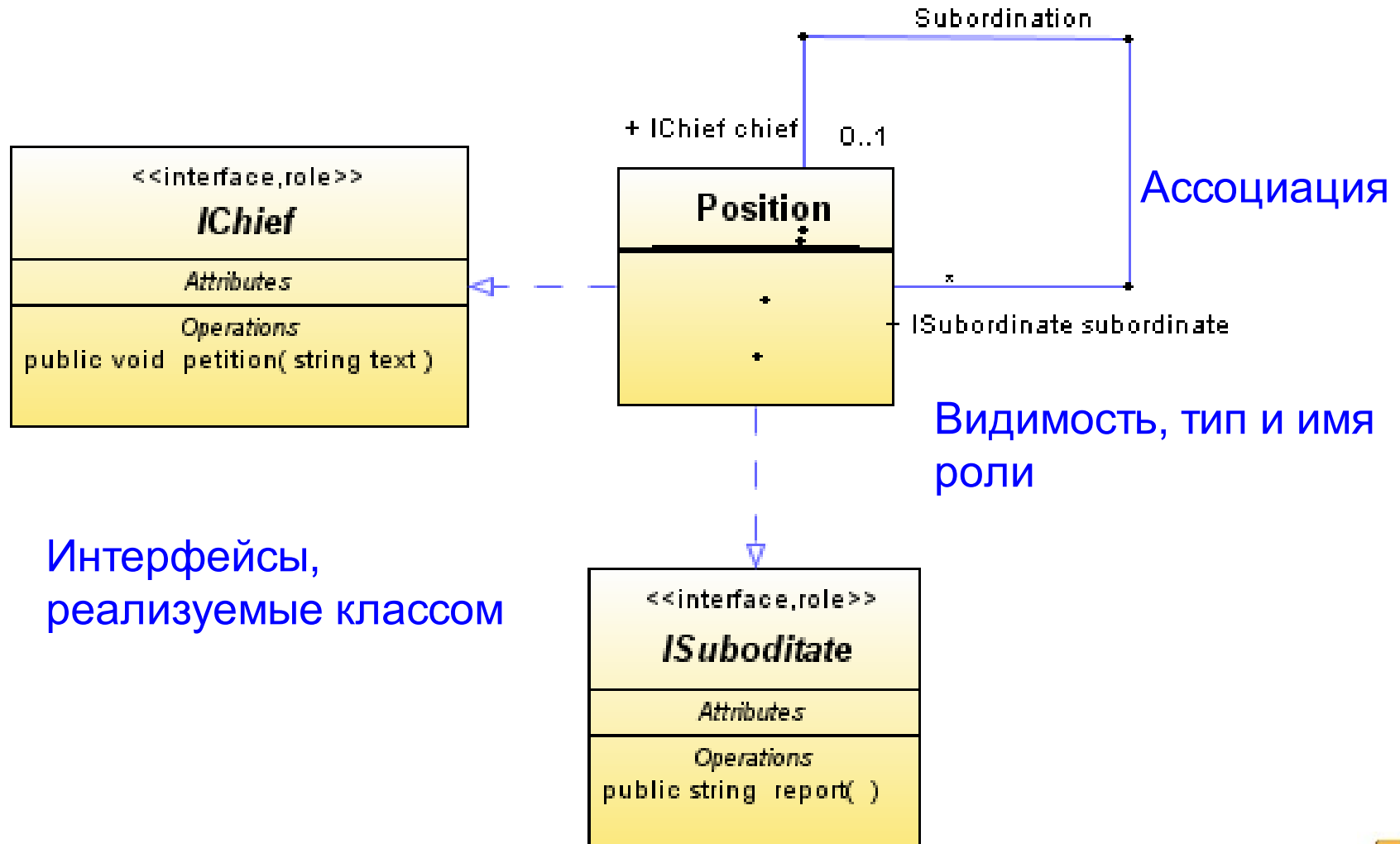


# Примеры ассоциаций



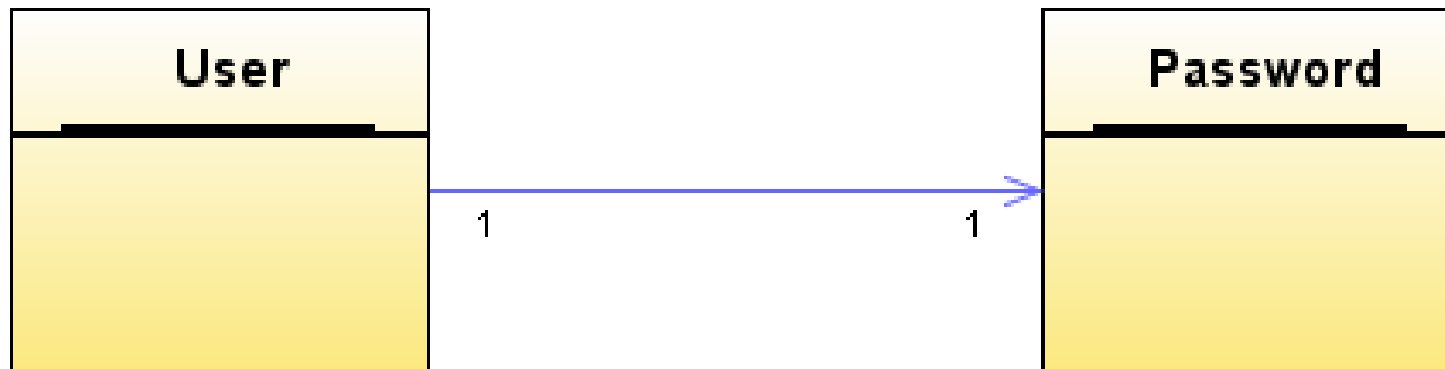
# Роль полюса ассоциации

## ■ видимость ИМЯ : тип



# Направление навигации

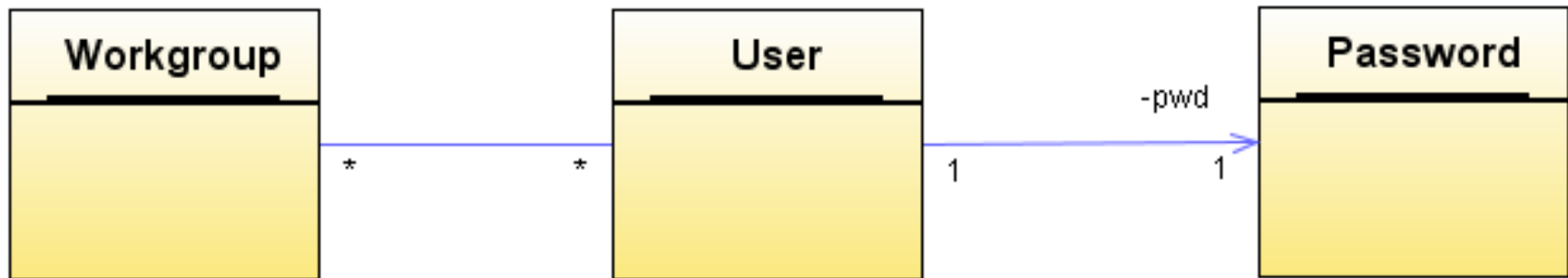
Ассоциация с указанным  
направлением навигации



# Видимость полюса ассоциации

- Для чужих – связанные ассоциацией *всегда* видят друг друга

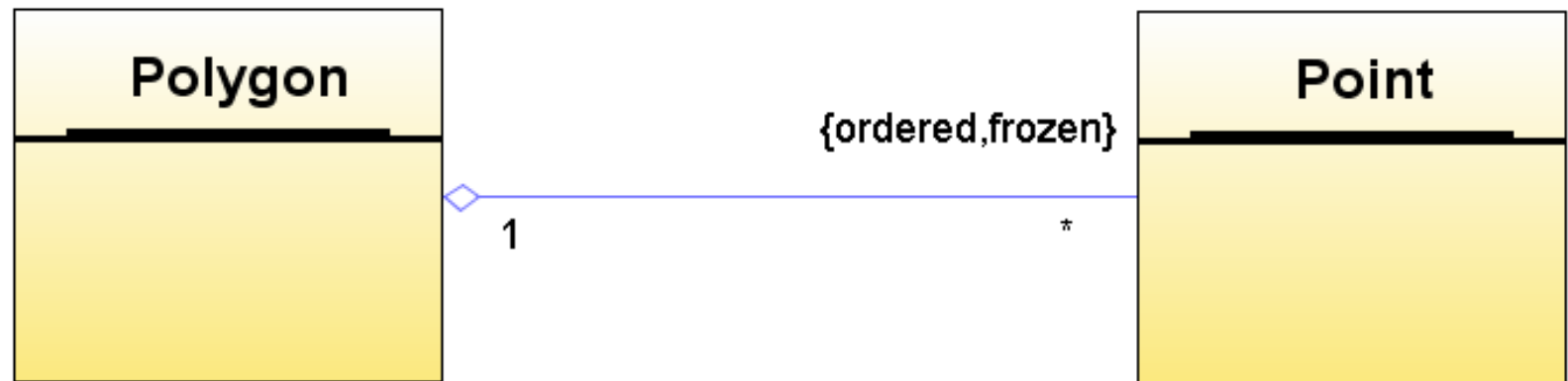
Ассоциация с направлением навигации и ограничением видимости



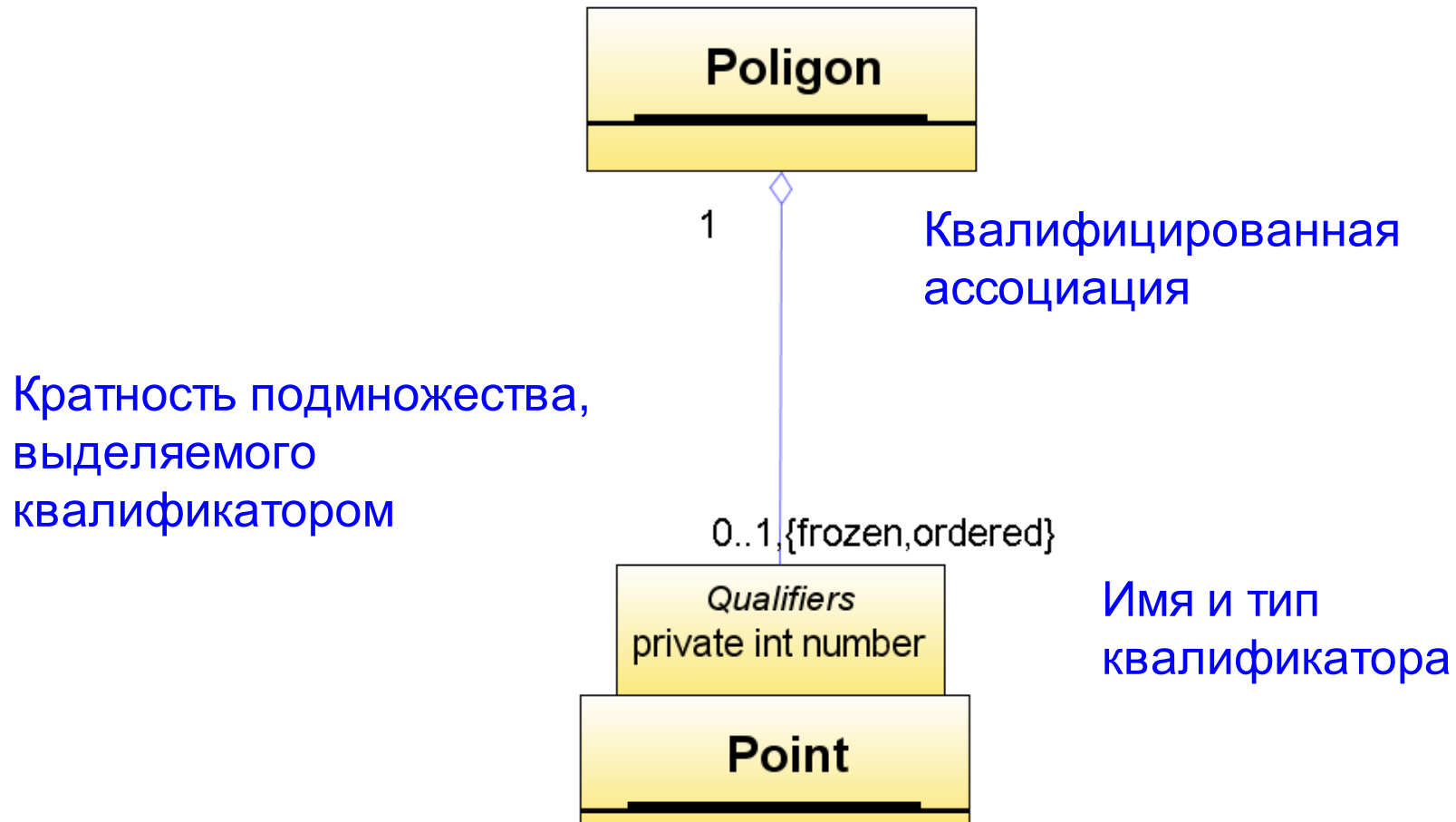
Ассоциация «многие ко многим»



# Упорядоченность и изменяемость множества объектов на полюсе связи



# Квалификатор полюса ассоциации

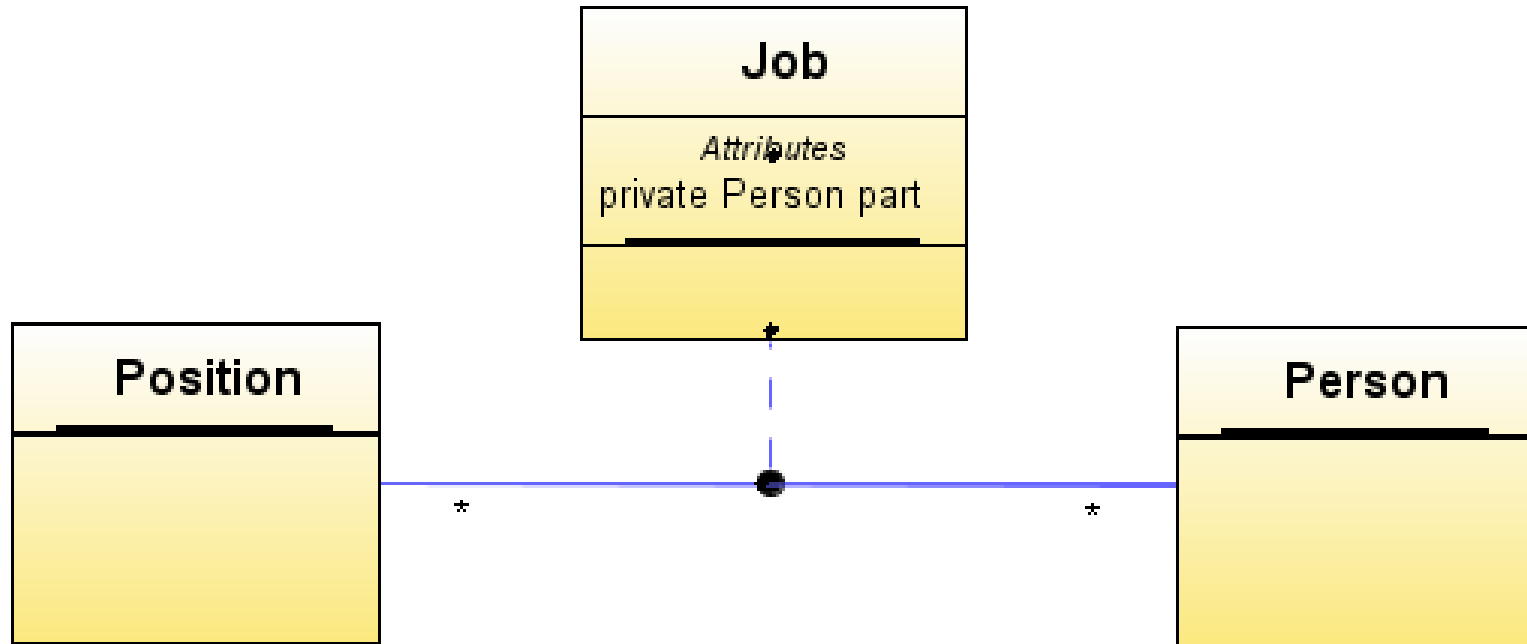


- Квалификатор – это атрибут ассоциации



# Класс ассоциации

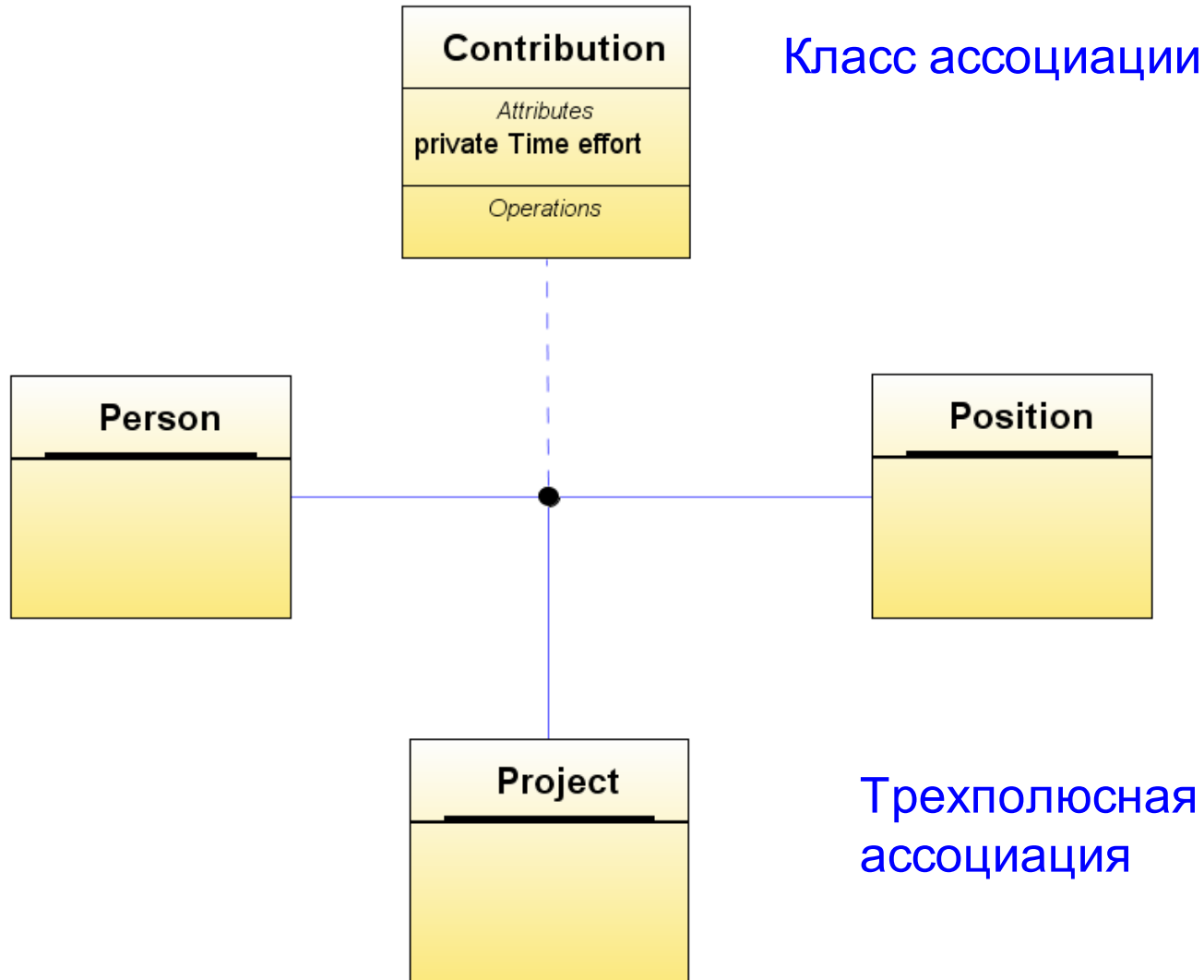
Класс ассоциации



Линия ассоциации

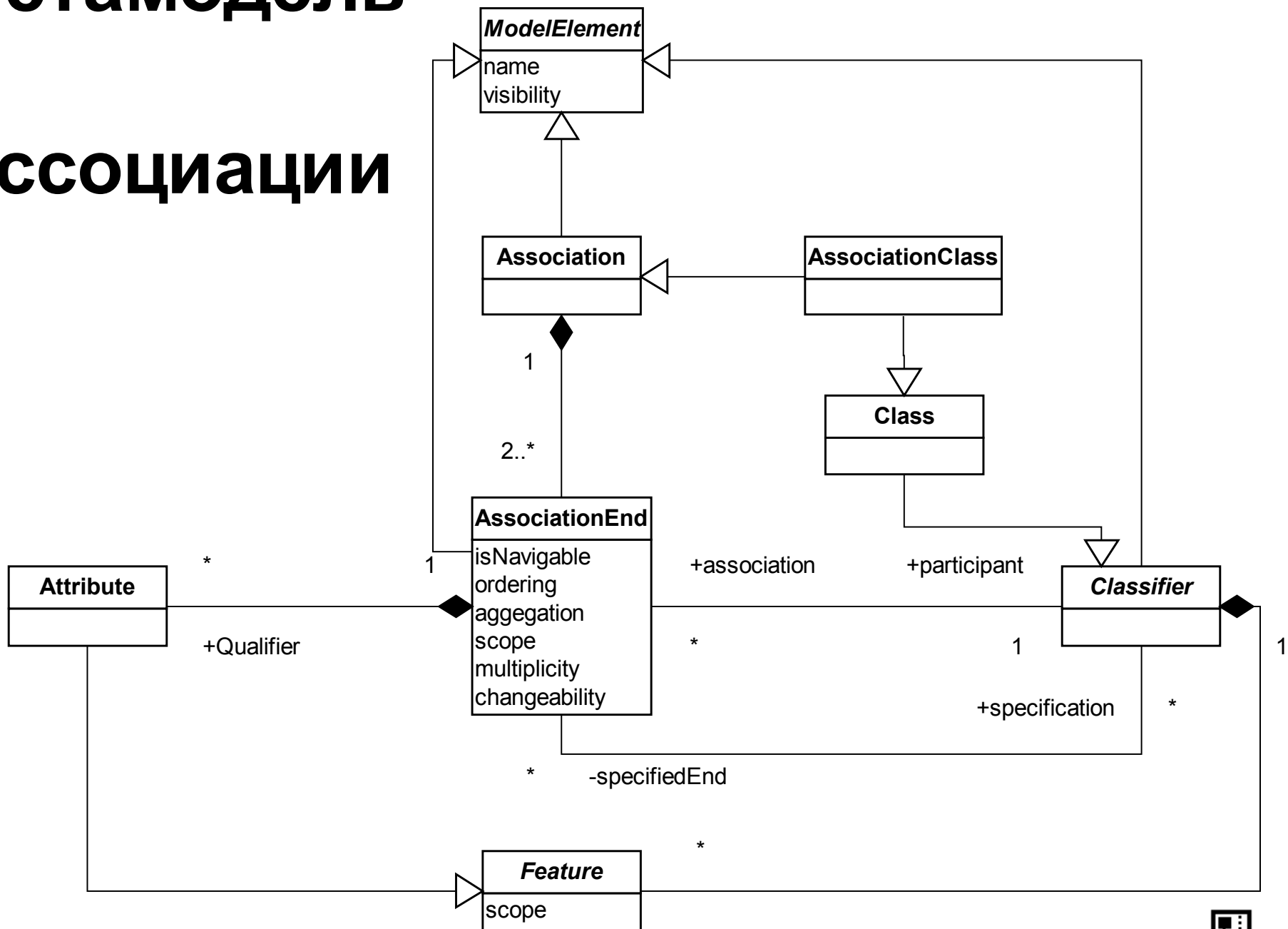


# Многополюсные ассоциации





## ассоциации



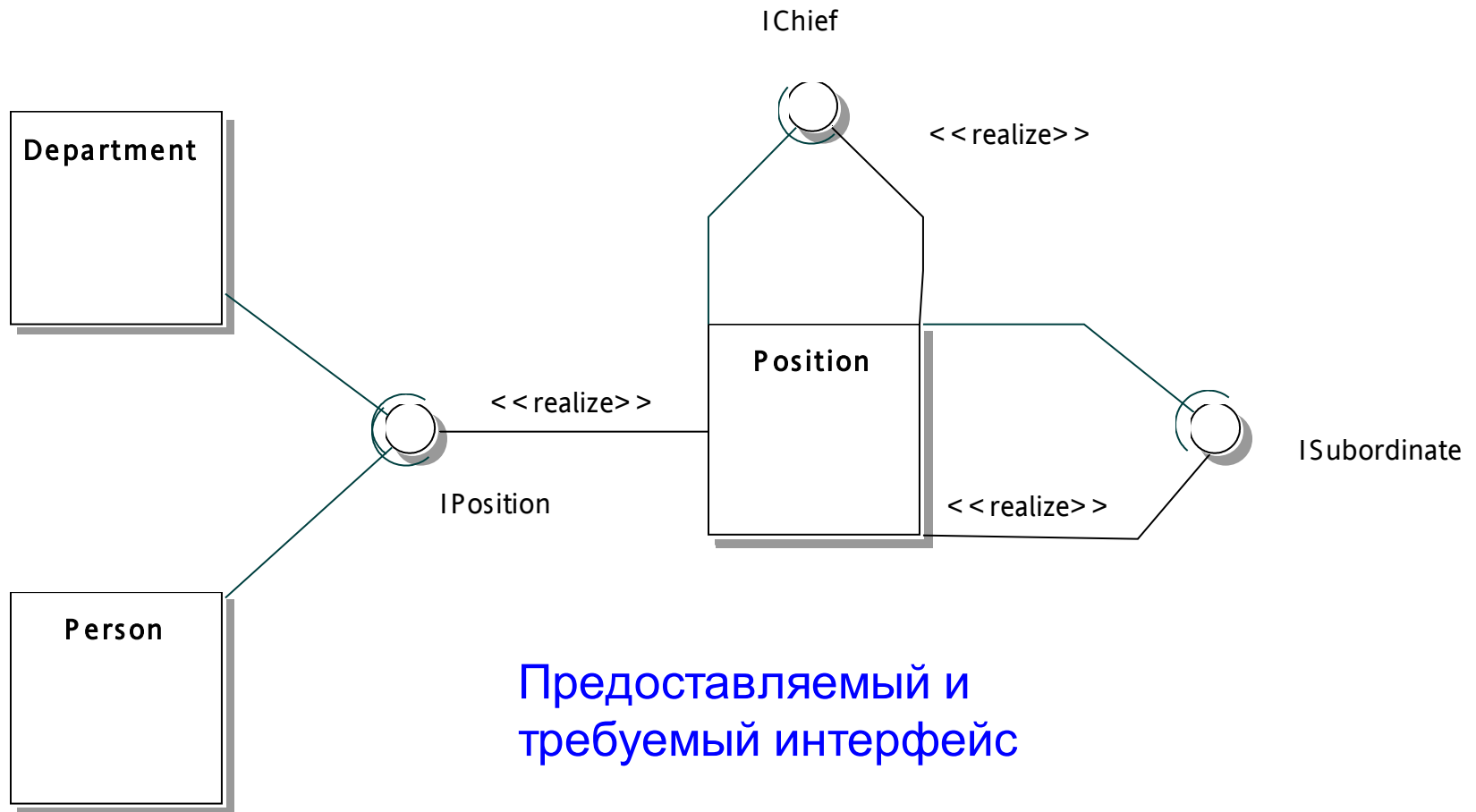
## 4.6. Интерфейсы, типы данных и роли

- Интерфейс = именованный набор абстрактных операций
- Класс реализует Интерфейс
- Класс использует Интерфейс
- Класс может реализовывать много интерфейсов (и наоборот)
- *Роль* = интерфейс, который предоставляет класс в данной ассоциации
- *Протокол* – автомат, описывающий допустимый порядок вызовов (2.0)

# Нотация реализации и использования интерфейсов (1.x)



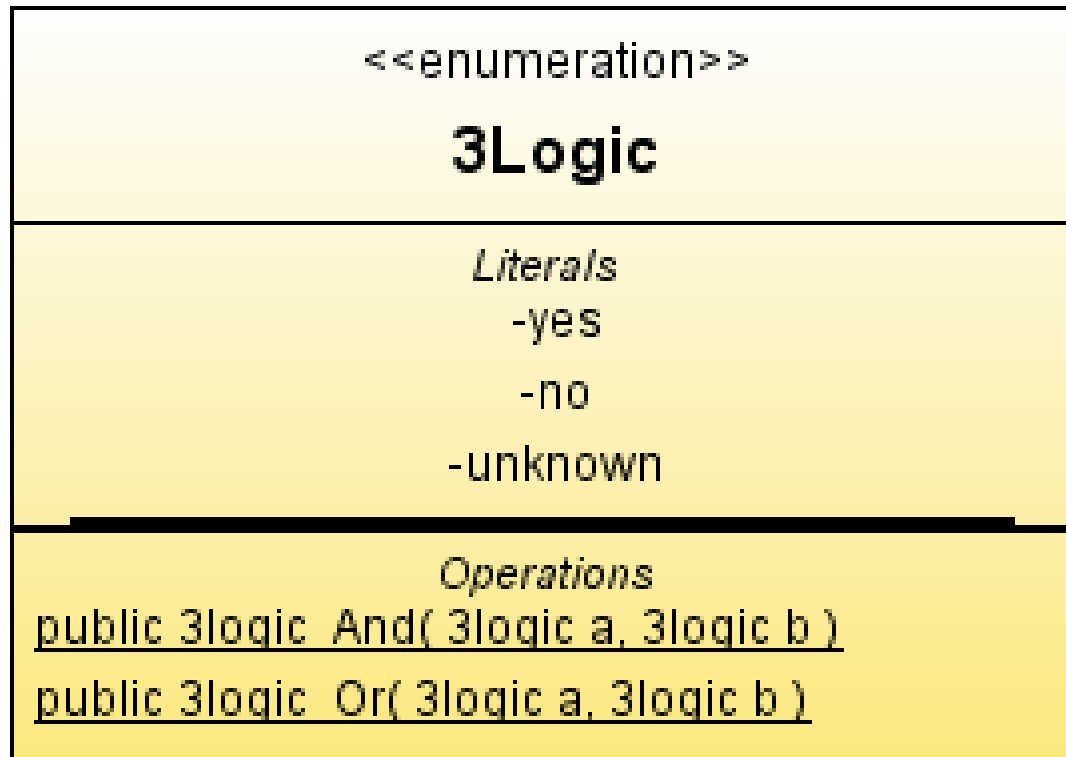
# Нотация «чупа-чупс» (2.0)



# Типы данных

- Тип (в UML) почти = интерфейс
- Тип – класс со стереотипом «datatype»
- Экземпляры (значения) типа не обладают индивидуальностью (identity)
- Примитивные типы (и типовые выражения) из целевого языка программирования
- Числа и строки есть всегда
- Перечисляемые типы «enumeration»

# Пример: трехзначная логика



# Шаблоны

- **Шаблон (template) – класс с параметрами**
- **Параметром может быть всё!**
- **ПАРАМЕТР : тип**
- **Явное связывание – зависимость со стереотипом «bind»**
- **Неявное связывание – имя\_шаблона <аргументы>**

# Пример: параметризованный класс (шаблон)

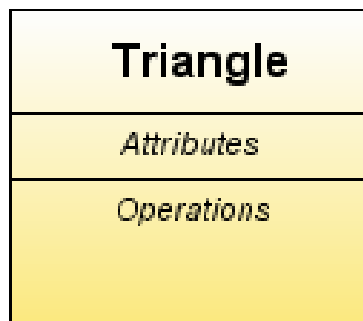
Неявное  
связывание

Array<n -> 3, T -> Point>

Тип данных

<<datatype>>  
Point

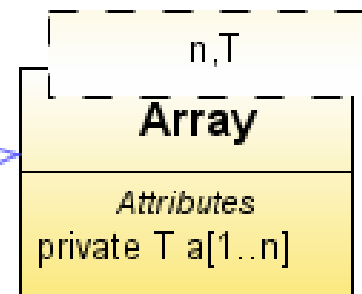
Параметры  
шаблона



Конкретный  
класс

—> <<T-> Point, n-> 3>  
<<bind>>

Явное связывание



Шаблон

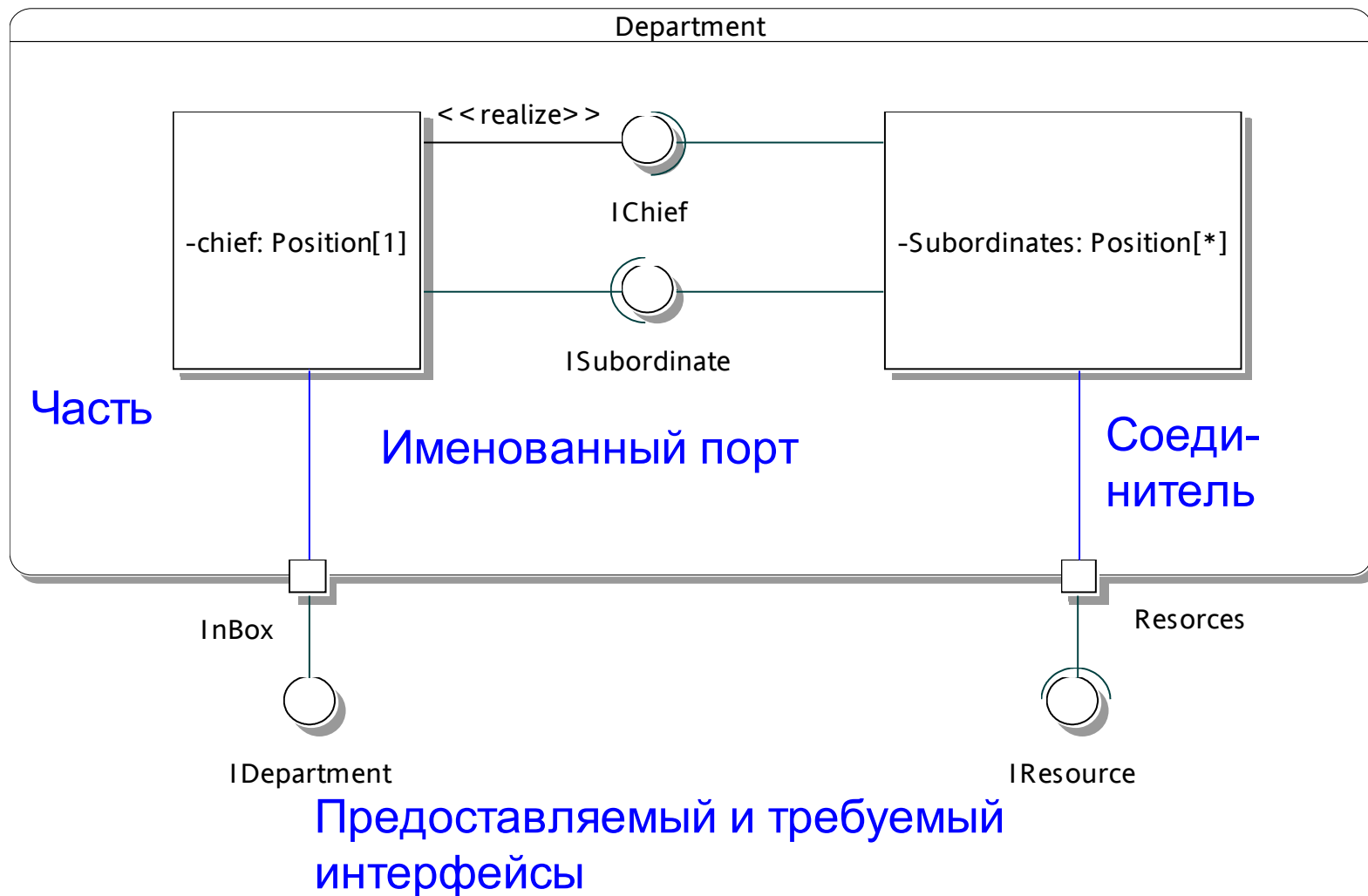




# 4.7. Внутренняя структура объектов

- **Важнейшее нововведение UML 2.0**
- Структурированный классификатор (structured classifier) состоит из частей (part), портов (port) и соединителей (connector)
- Части м.б. структурированными классификаторами → декомпозиция!
- *Часть* – структурная составляющая классификатора
- Часть имеет имя, тип и кратность
- *Порт* – индивидуальная точка взаимодействия экземпляра классификатора и внешней среды
- *Соединитель* – «контекстная» ассоциация между портами и частями, действующая только внутри классификатора
- НЕ только СТРУКТУРА, но и ПОВЕДЕНИЕ (кооперация частей)

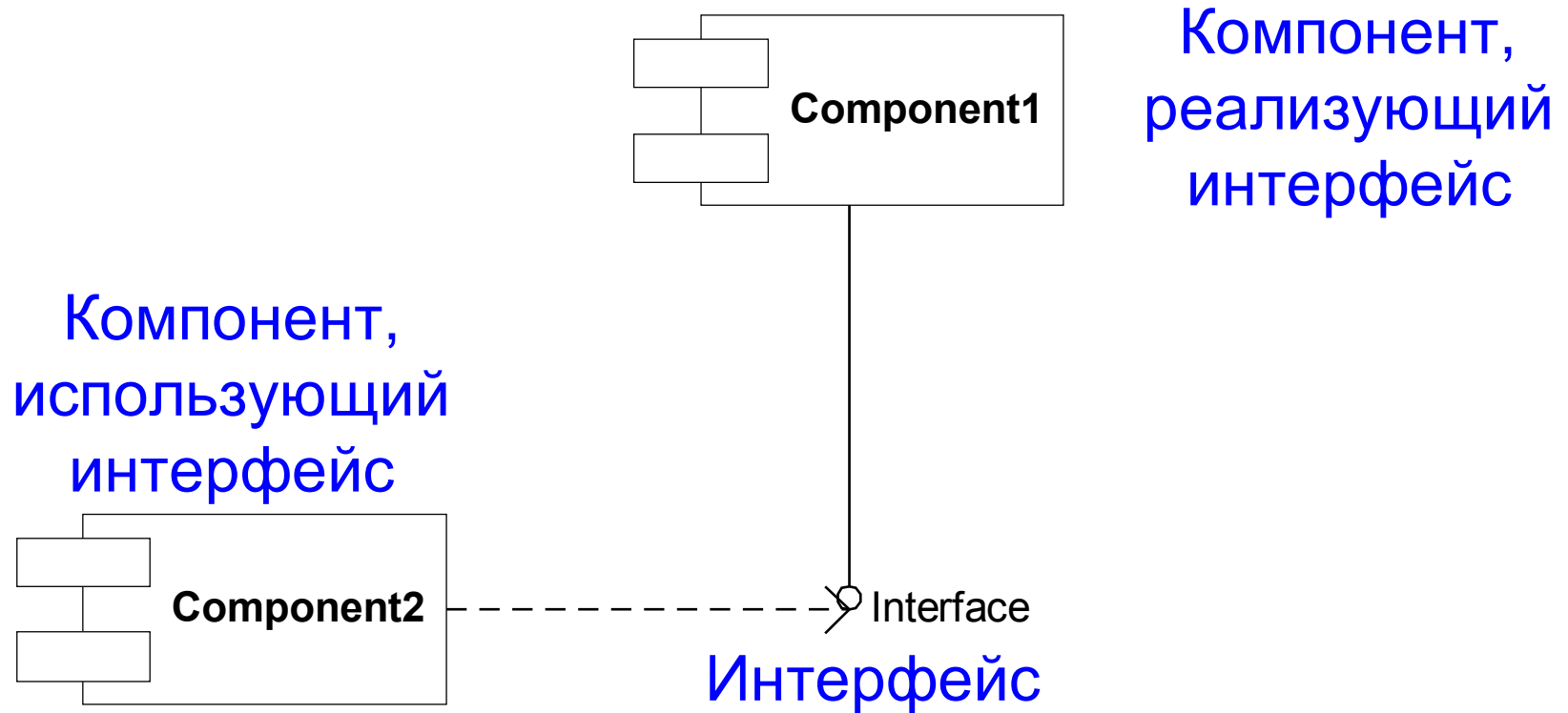
# Пример ИС ОК



# 4.8. Диаграммы компонентов

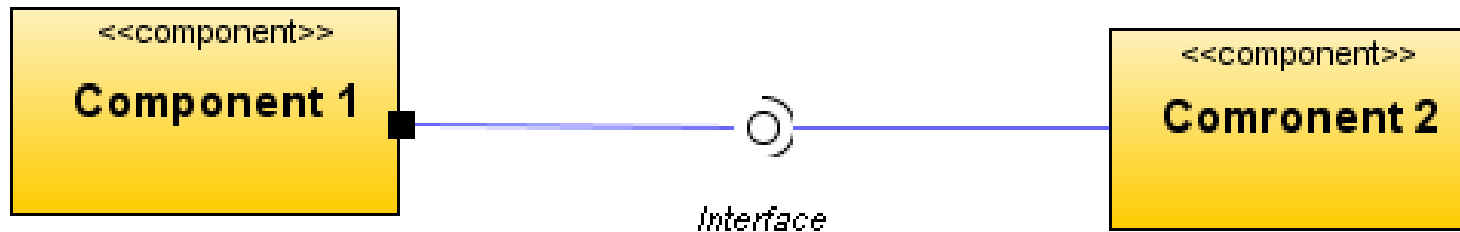
- **Назначение**
  - **Перечисление и взаимосвязи артефактов системы**
- **Сущности**
  - **Компоненты**
  - **Интерфейсы**
  - **Классы**
- **Отношения**
  - **Зависимость**
  - **Ассоциация**
  - **Реализация**

# Нотация 1.x



# Нотация 2.0

Интерфейс



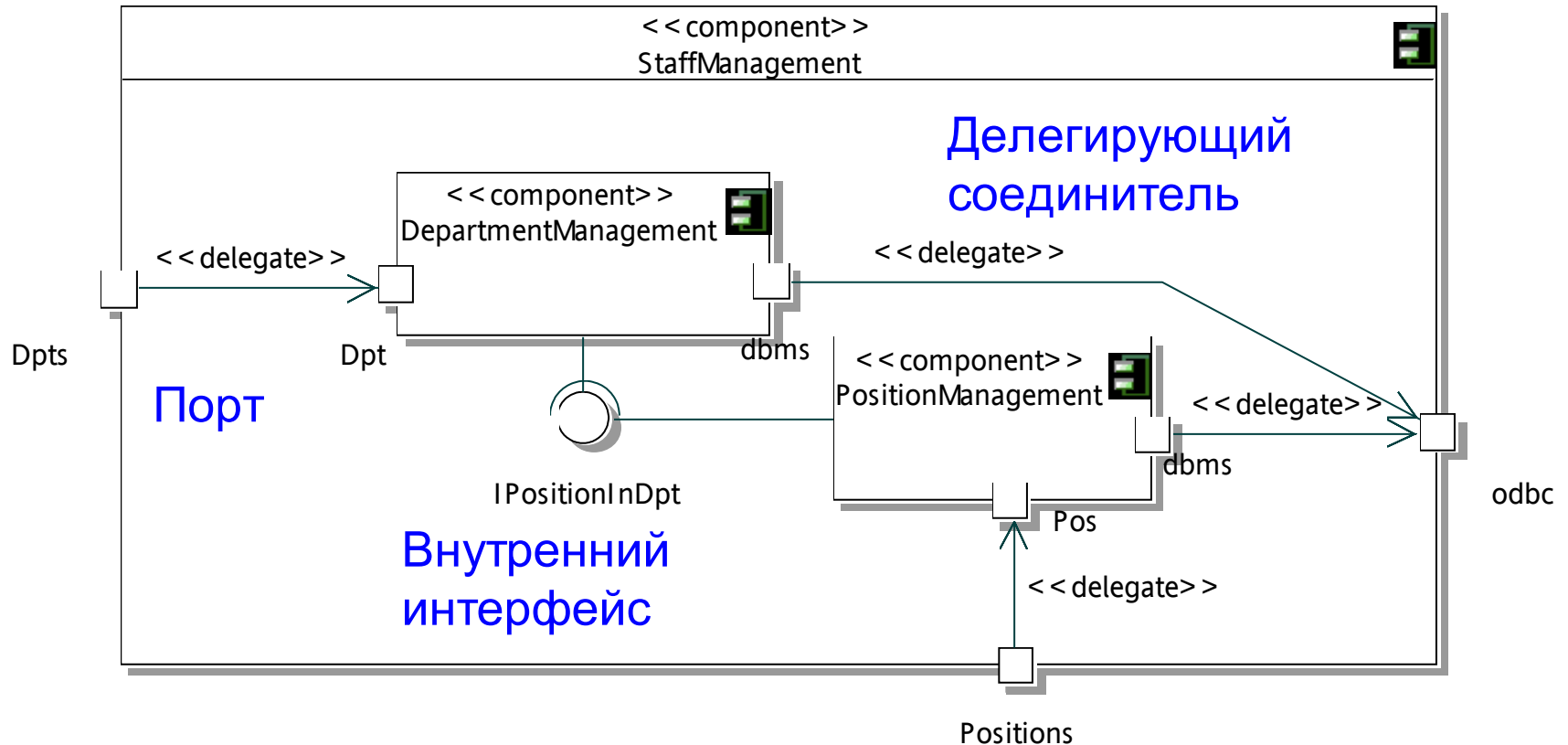
Компонент,  
предоставляющий  
интерфейс

Компонент,  
использующий  
интерфейс



# Внутренняя структура компонента

## Компонент с внутренней структурой



# Стереотипы компонентов 1.x

- «library»
- «table»
- «file»
- «document»
- «executable»
- Библиотека (DLL)
- Таблица (БД)
- Файл (.h, .cpp)
- Документ (help)
- По умолчанию

**В UML 2.0 введен новый стандартный стереотип «artifact»**

# Типичные применения

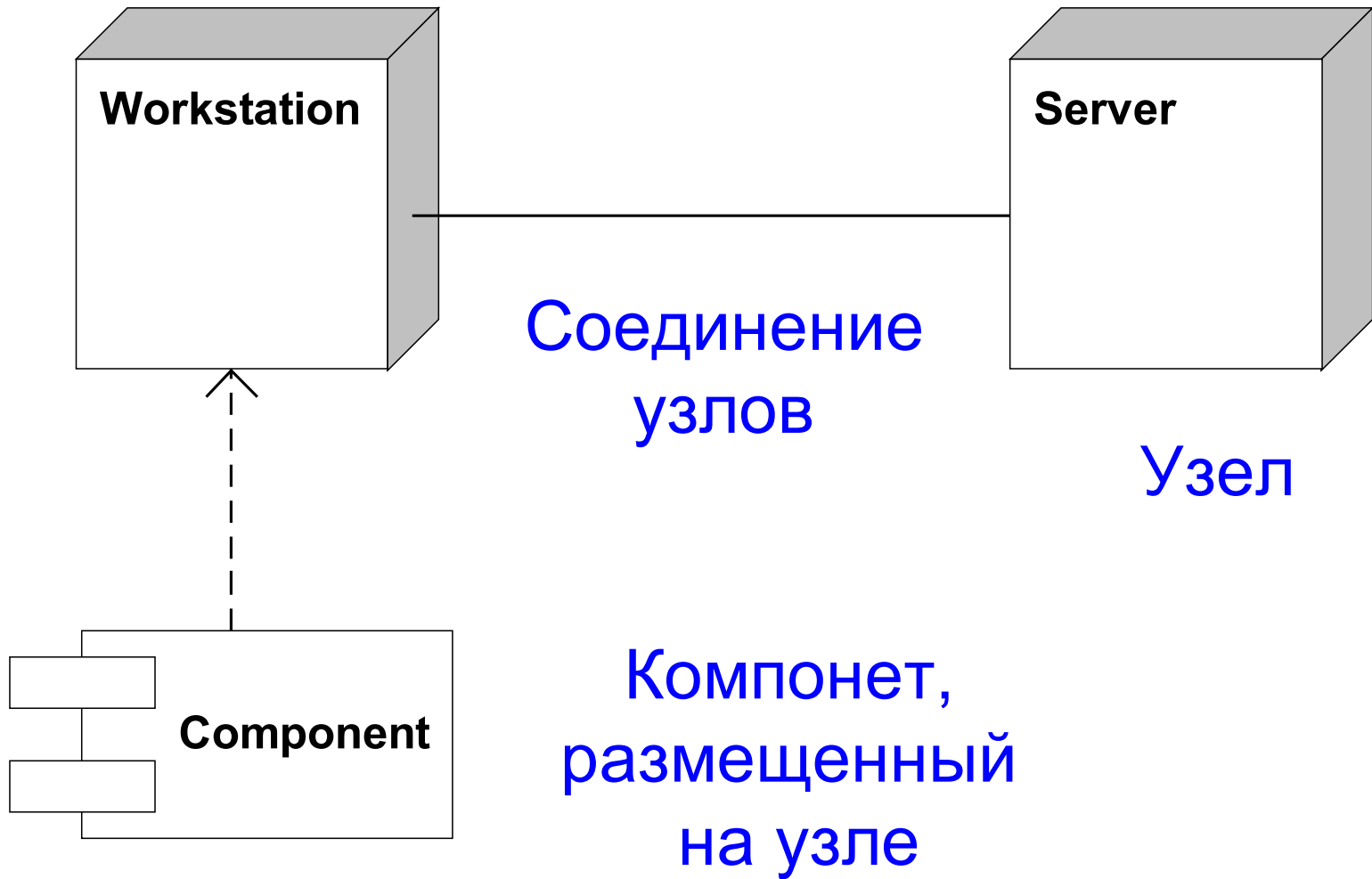
- **Монолитное приложение –  
диаграмма компонентов не нужна**
- **Управление конфигурацией**
- **Управление версиями**
- **Моделирование унаследованных  
баз данных**
- **Моделирование систем  
динамической архитектуры**



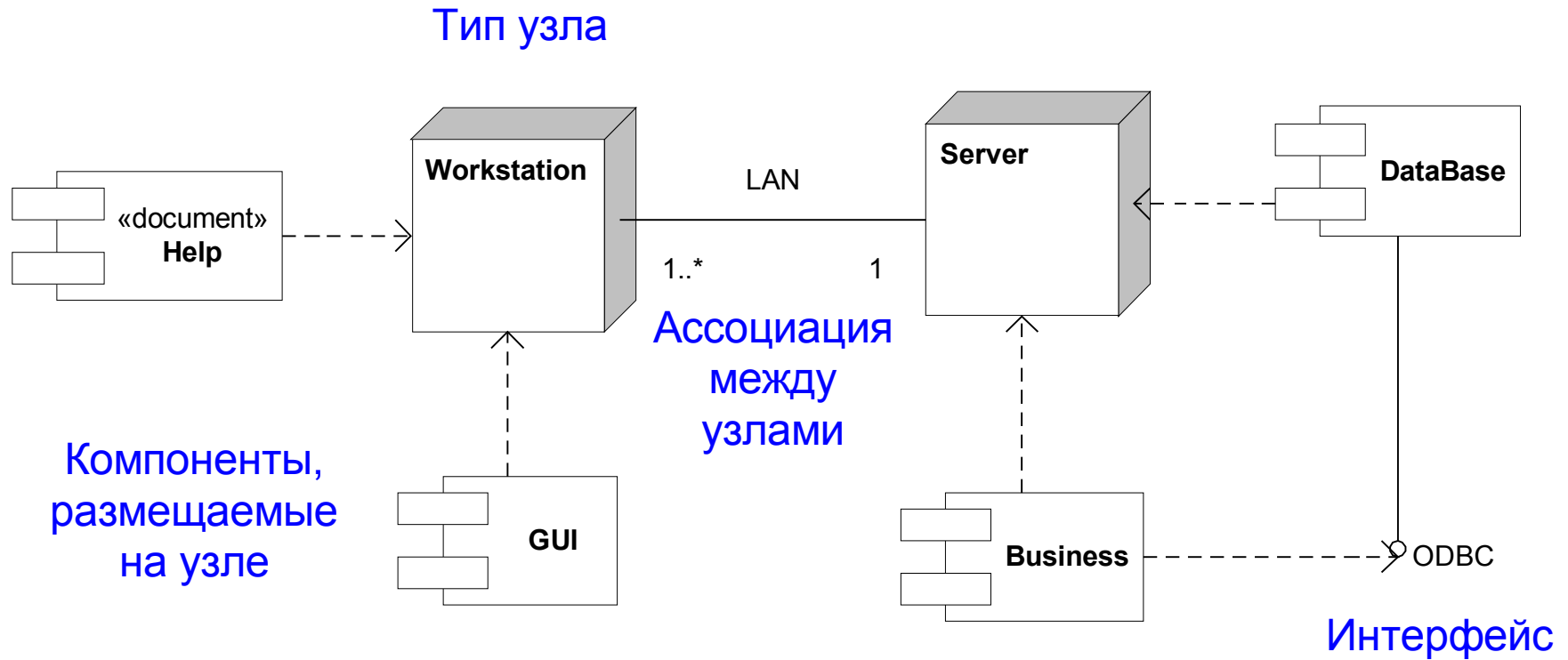
# Диаграммы размещения (1.x) → развертывания (2.0)

- **Назначение**
  - Описание топологии развернутой системы
  - Описание процесса установки программного продукта
- **Сущности**
  - + Узлы
- **Отношения**
  - Зависимость
  - Ассоциация

# Нотация 1.x

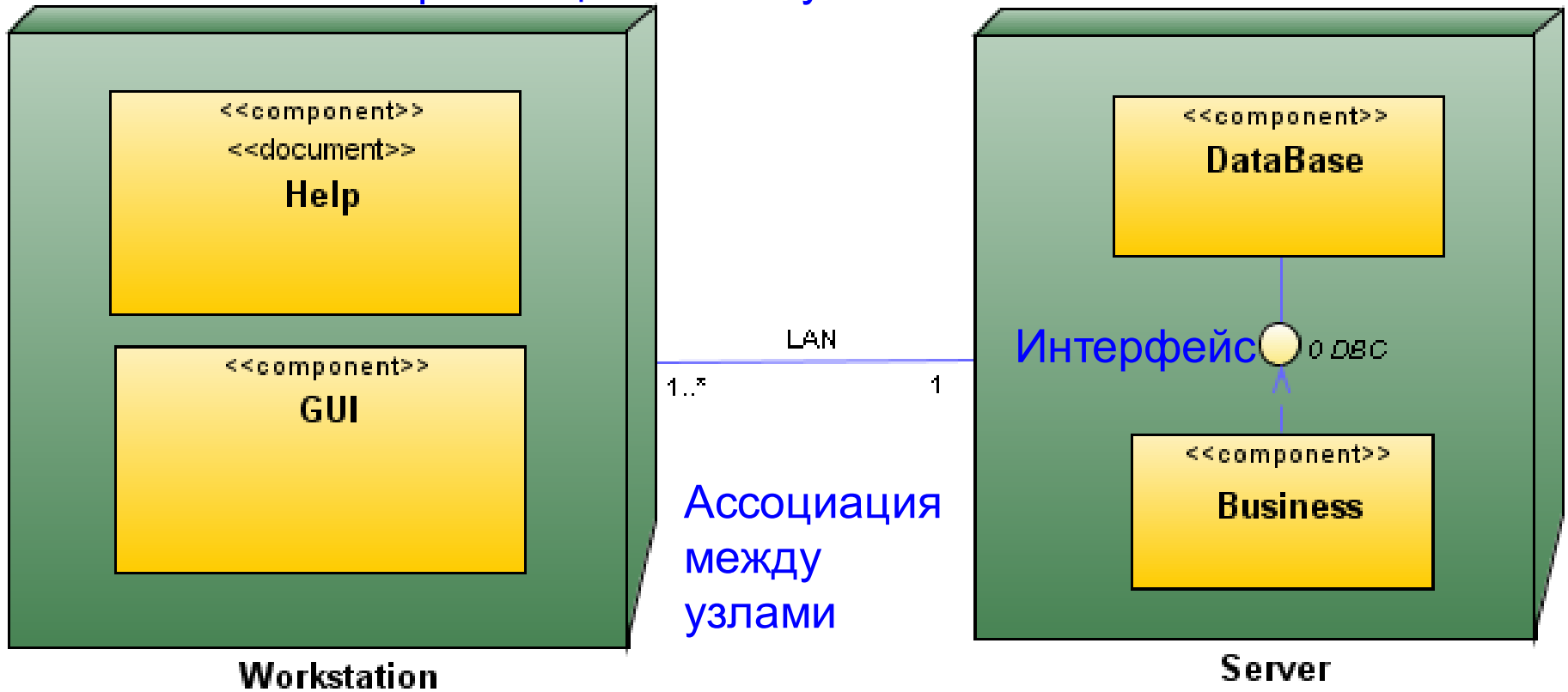


# Пример ИС ОК: тонкий клиент




# Пример ИС ОК: тонкий клиент (2.0)

Компоненты,  
размещаемые на узле



Тип узла

# Выводы

- Структура сложной системы описывается на уровне дескрипторов
- Диаграммы классов моделируют структуру объектов и связей между ними
- Диаграмма объектов = экземпляр диаграммы классов (в UML 2.0 )
- Диаграммы компонентов моделируют структуру компонентов (артефактов) и взаимосвязей между ними
- Диаграммы размещения моделируют структуру вычислительных ресурсов и размещение компонентов

# Советы

- **Словарь предметной области – основа для выбора классов**
- **Описывать структуру удобнее параллельно с описанием поведения**
- **Не обязательно включать в модель все классы сразу**
- **Не обязательно определять все свойства класса сразу**
- **Не обязательно показывать на диаграмме все свойства класса**
- **Не обязательно определять все отношения между классами сразу**