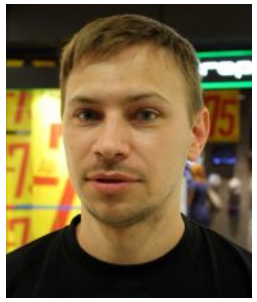
Моделирование на UML. Вторая ступень

Моделирование структуры



Иванов Д.Ю., Новиков Ф.А.

Структура курса

Об этом курсе

Часть 1. Введение в UML

Часть 2. Моделирование использования

✓ Часть 3. Моделирование структуры

Часть 4. Моделирование поведения

Часть 5. Дисциплина моделирования

Содержание

1. Моделирование структуры

1.1 Дескрипторы

1.2 Парадигмы программирования. Непроцедурные парадигмы

1.3 Объектно-ориентированная парадигма. Хронология развития. Принцип подстановки

1.4 Классификаторы. Свойства классификаторов

1.5 Идентификация классов

Природа моделирования

- Моделирование структуры: составные части системы (классы) и отношения между ними
- Число возможных вариантов наборов объектов и связей необозримо велико
- Представить их ВСЕ в модели бессмысленно
- Как строить компактные (полезные) модели необозримых систем?

Примеры моделей (i)

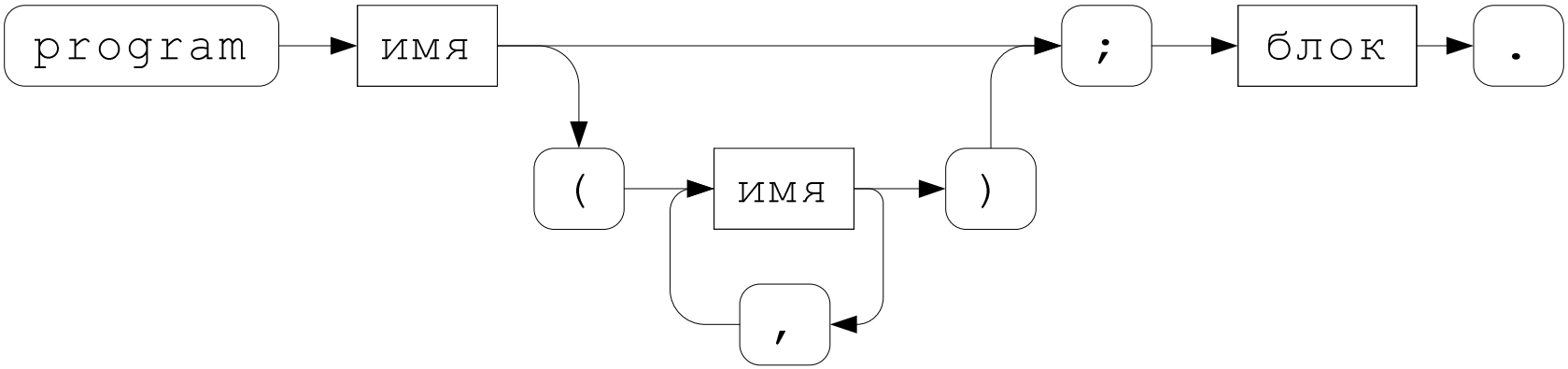
1. Наибольшее число Фибоначчи

```
function MaxFib : integer;
var f1, f2 : integer;
begin
        f1:=1; f2:=1;
        while maxint – f1 > f2 do begin
                f2 := f1 + f2;
                f1 := f2 – f1
        end;
        MaxFib := f2
end;
```

Примеры моделей (ii)

2. Описание формального языка синтаксическими диаграммами Вирта

- Синтаксическое понятие «программа»:



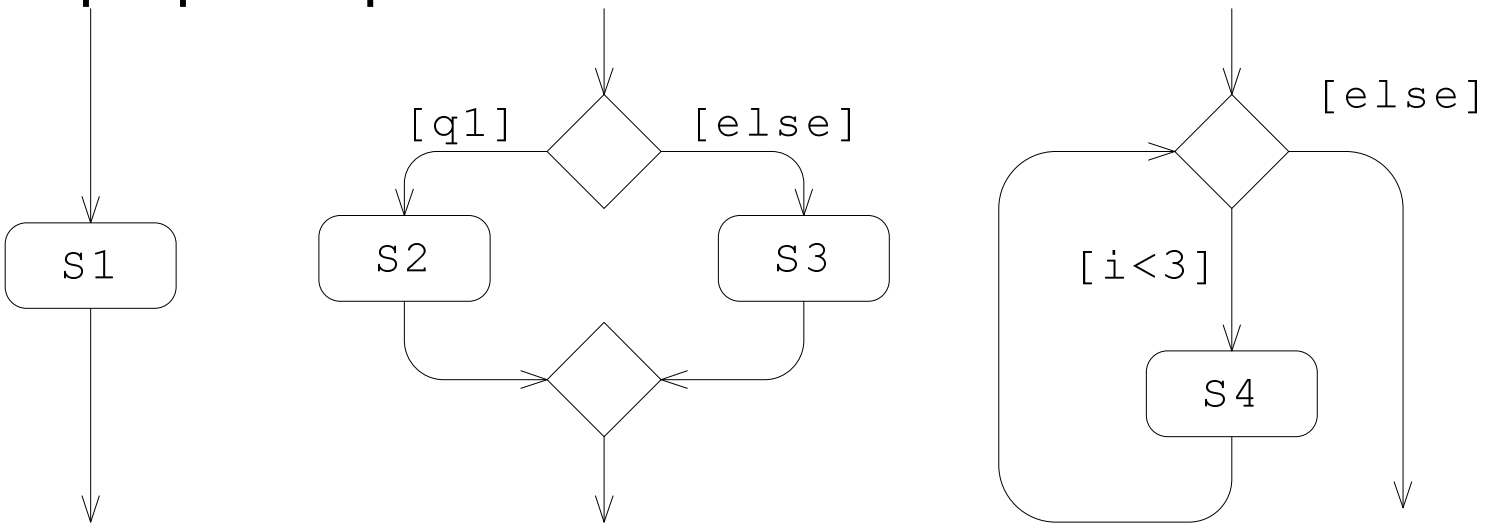
Дескрипторы

- Дескриптор (descriptor) – описание общих свойств множества объектов, включая их структуру, отношения, поведение и т. д.
- → Необходимы правила интерпретации описания
- Дескриптор – конечное описание (intent) бесконечного множества (extent)
- Литерал описывает сам себя
- Почти все элементы модели (классы, варианты использования, узлы, ассоциации …) – дескрипторы
- Примечания и пакеты – литералы

Парадигмы программирования (i)

- Парадигма программирования — это собрание основополагающих принципов, методической основы конкретных технологий и инструментальных средств программирования

1. Структурное программирование = «программирование без **Go To**»



Э.В. Дейкстра

Э́дсгер Ви́бе Де́йкстра (Edsger Wybe Dijkstra, 1930 – 2002) — нидерландский учёный

- применение математической логики при разработке компьютерных программ
- разработка языка программирования Алгол
- структурное программирования
- применение семафоров для синхронизации процессов в многозадачных системах
- алгоритм нахождения кратчайшего пути в графе

Лауреат премии Тьюринга 1972 года

Компьютерные архитектуры

Архитектура фон Неймана

- Исторически первая и до сих пор самая распространенная
- Пассивная (линейная) адресуемая память, состоящая из (одинаковых) ячеек
- Активный процессор, дискретно (и последовательно) выполняющий команды
- В памяти хранятся как обрабатываемые данные, так и выполняемые команды
- Команды оперируют с ячейками памяти (произвольными), указывая их адреса.

Парадигмы программирования (ii)

- Понятия:
  - **переменная** (имеет имя и может менять значение)
  - **управление** (определяемая разработчиком последовательность выполнения команд)
- Процедурное программирование — оба оба этих понятия
- Непроцедурное — одно или оба понятия не используется
- Программирование, в котором используется явное понятие управления, называется императивным
- Не используется явное управление — декларативное

Непроцедурные парадигмы (i)

Логическое программирование:

- Из конструктивного доказательства теоремы существования $\forall x(P(x) \to \exists y Q(x, y))$ может быть извлечена аннотированная программа

$$P(x)\{y := f(x)\}Q(x, y)$$

- где $P(x)$ — предусловие,
  $Q(x, y)$ — постусловие,
  f — сколемовская функция
  - Реализация — язык Пролог

Непроцедурные парадигмы (ii)

Продукционное программирование

- Связано с нормальными алгорифмами Маркова
- Продукционная программа состоит из набора правил вида **if** P(D) **then** D:=f(D)
  где D – глобальная база данных

Функциональное программирование

- Связано с λ-исчислением Чёрча
- Функциональная программа является композицией функционалов
- Нет управления, переменных и присваиваний

Объектно-ориентированная парадигма (i)

Основные характеристики:

- Индивидуальность (identity): все объекты отличаются друг от друга
- Классификация (classification): объекты с одинаковыми структурами данных и поведением группируются в абстракции = классы (class), описывающие свойства сходных объектов = экземпляров (instance)

Объектно-ориентированная парадигма (ii)

- Наследование (inheritance): позволяет выстроить иерархии, выделяя в суперклассах общие составляющие (feature), которые уточняются в подклассах
  - атрибуты (attribute) выглядят как переменные
  - операции (operation) выглядят как процедуры или функции
- Полиморфизм (polymorphism): одна и та же операция в разных подклассах – разное поведение и разные методы (method), идентификация метода не только по имени, но и классу, типам и количеству аргументов

Хронология развития

ВСЕ перечисленные понятия и идеи

- известны с 1968 года (СИМУЛА-67)
- были оформлены в теоретических работах к 1978 году (Б. Лисков и др.)
- получили общепризнанное языковое выражение к 1988 году (C++)
- стали основным и массовым инструментом для профессиональных программистов к 1998 году

Но и сейчас, после 2008 годуа наиболее массово используются гибридные языки (C++, Java, C#), а не чисто объектно-ориентированные (Smalltalk, Self, Eiffel)

Принцип подстановки

Наследование интерфейса: подкласс обеспечивает (по меньшей мере) тот же интерфейс, что и суперкласс →

Экземпляр подкласса может использоваться везде, где используется экземпляр суперкласс

- При открытом наследовании (= обобщение в UML)

Барбара Лисков

Барбара Лисков (Barbara Liskov, 1939) — первая женщина, получившая в Америке докторскую степень по информатике

- Известна как лидер многих значительных проектов, включая CLU, Argus, Thor
- Особенно часто ее имя связывается с принципом подстановки
- Лауреат медали имени Джона фон Неймана 2004 года
- Лауреат премии Тьюринга 2009

Моделирование структуры: Какой?

1. Структура связей между объектами во время выполнения
   - Ассоциации на диаграмме классов
2. Структура хранения данных
   - Ассоциации с указанием кратности полюсов
3. Структура программного кода
   - Структура классов и пакетов
4. Структура сложных объектов, состоящих из взаимодействующих частей
   - Диаграмма внутренней структуры классификатора
5. Структура артефактов в проекте
   - Диаграммы компонентов и размещения
6. Структура компонентов в приложении
   - Классы и интерфейсы на диаграмме компонентов
7. Структура используемых вычислительных ресурсов
   - Диаграммы размещения

Классификаторы

Классификатор — это дескриптор множества однотипных объектов

→ может иметь экземпляры

действующее лицо (actor), вариант использования (use case), артефакт (artifact), тип данных (data type), ассоциация (association), класс ассоциации (association class), интерфейс (interface), класс (class), кооперация (collaboration), компонент (component), узел (node)

применяются в процессе моделирования структуры

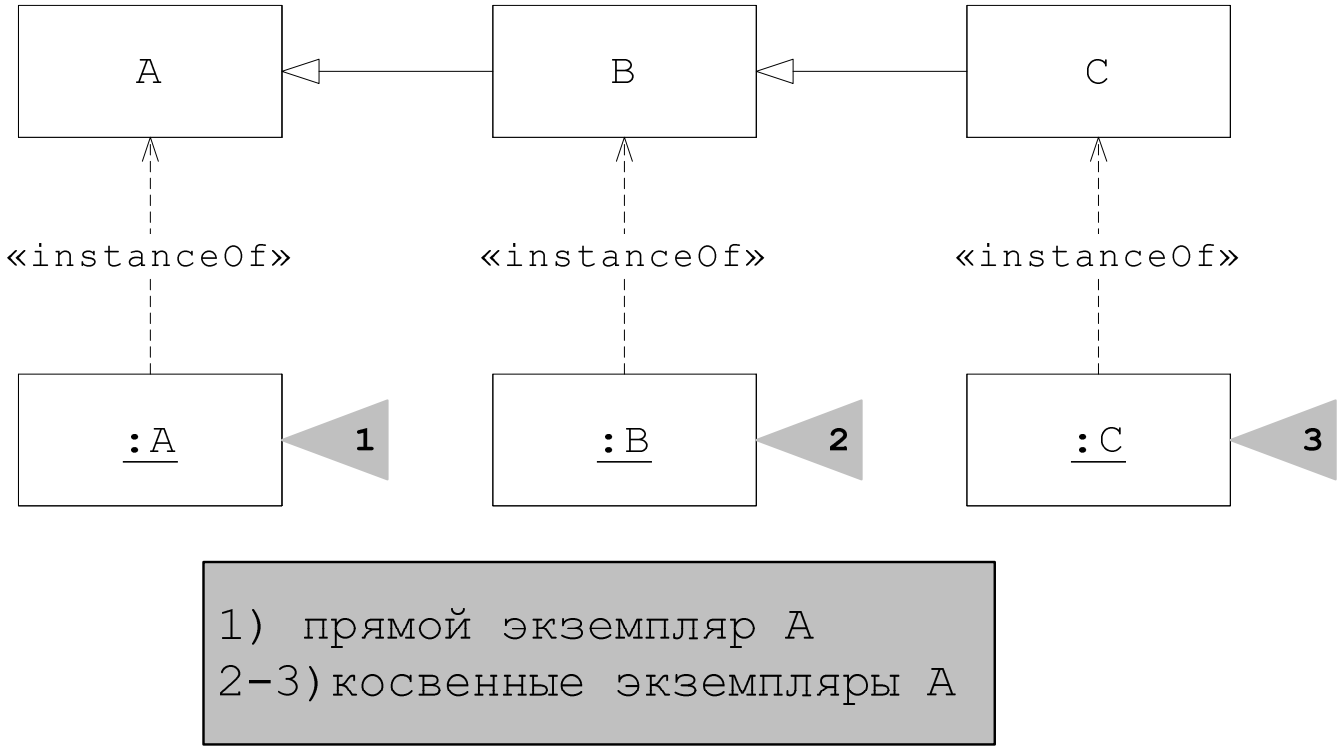
Свойства классификаторов (i)

1. Имя

- Уникально (в пространстве имен), служит для идентификации

2. Экземпляры

- Прямые – экземпляры *A*
- Косвенные – экземпляры потомков классификаторов *A*



Свойства классификаторов (ii)

3. Абстрактный *(abstract)*
   - не может иметь прямых экземпляров
   - имя выделяется курсивом

4. Видимость (visibility)
   - может ли составляющая одного классификатора использоваться в другом
   - *открытый* + public – виден везде
   - *защищенный* # protected – виден в контейнере
   - *закрытый* – private – виден в своем элементе
   - *пакетный* ~ package – виден в своем пакете

Свойства классификаторов (iii)

5. Область действия (scope)

   определяет, как проявляет себя составляющая классификатора в экземплярах: используется общее значение или у каждого экземпляра свое

   - экземпляр (instance) — по умолчанию
   - классификатор (classifier) — подчеркивается

Свойства классификаторов (iv)

6. Кратность (multiplicity)

- Все допустимые значения мощности

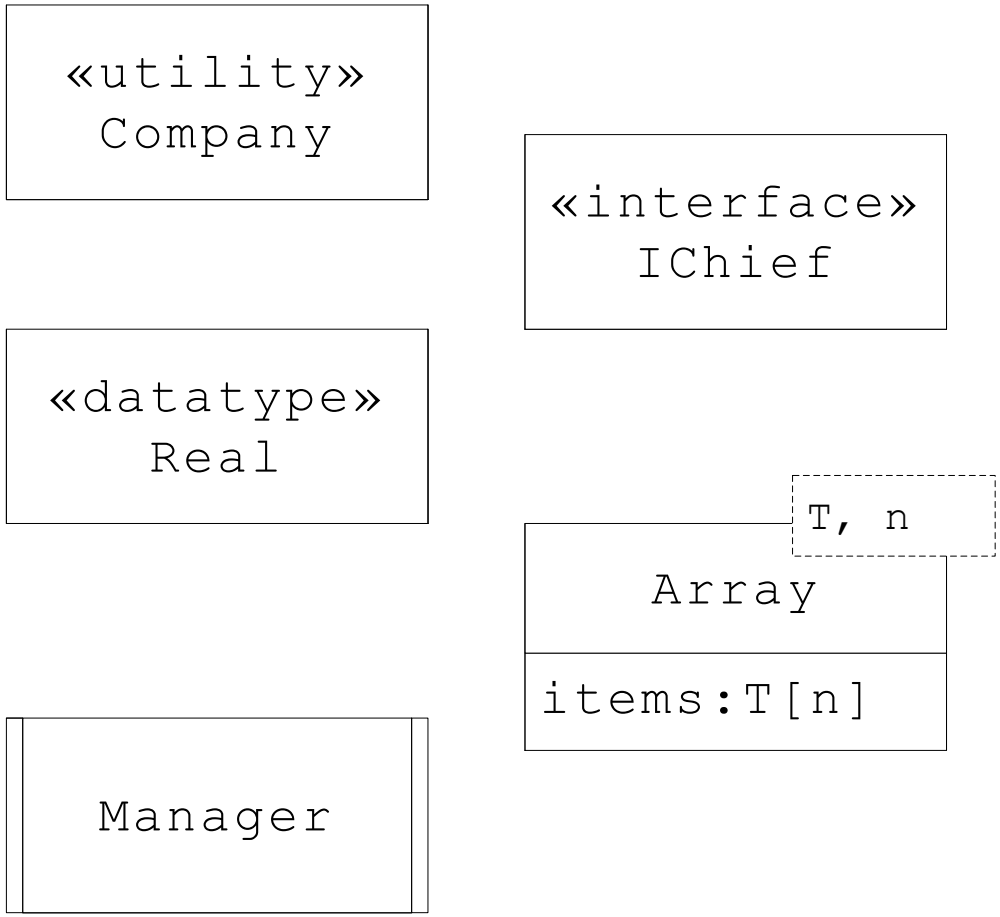
  Нижняя граница .. ВЕРХНЯЯ ГРАНИЦА

  - кратность 0 нет экземпляров *служба (utility)*
  - кратность 1 ровно один *одиночка (singleton)*
  - кратность 8 фиксированное число (e.g. концентратор)
  - кратность * произвольное число *по умолчанию*

7. Классификаторы (и только они) могут участвовать в отношении обобщения

2. Сущности на диаграмме классов

- Классы
  - стереотипы
  - интерфейсы
  - типы данных
  - шаблоны
  - активные классы
- Пакеты
- Примечания

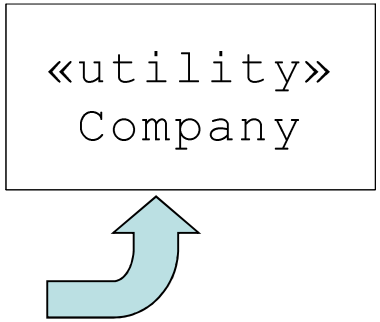


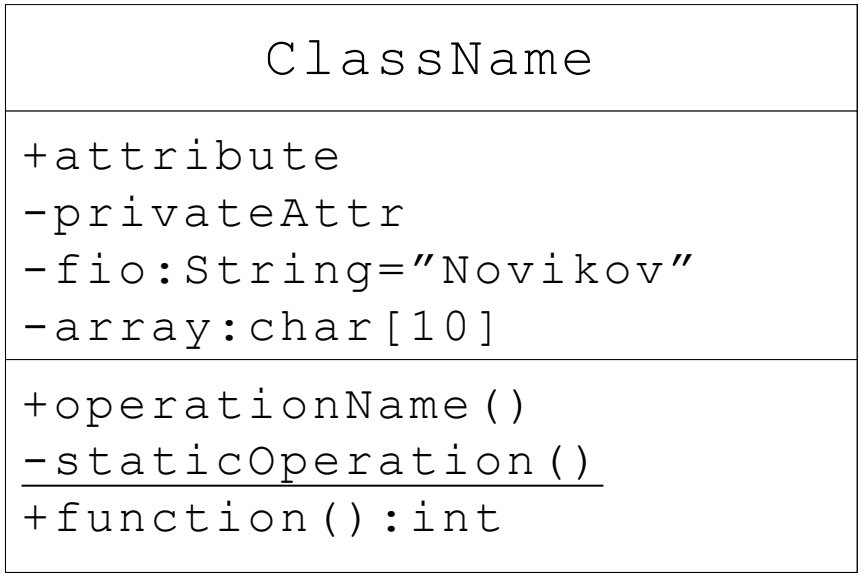
Класс

Секции (compartment):

- Секция имени
  - «стереотип» ИМЯ {свойства} кратность
- Секция атрибутов
  - список описаний атрибутов
- Секция операций
  - список описаний операций

+ доп. секции = примечания





Атрибуты

видимость ИМЯ [кратность] : тип = начальное_значение {свойства}

- Видимость: + – # ~
- ИМЯ: атрибут экземпляра, ИМЯ: атрибут класса
- Кратность: определяет атрибут как массив
- Тип: примитивный / определенный пользователем
- Начальное значение: при создании экземпляра атрибут получает указанное значение
- Свойства: ограничения и именованные значения, e.g. изменяемость (changeability)

Примеры описаний атрибутов

Пример	Пояснение
`name`	Только имя атрибута
`+name`	Имя и открытая видимость — непосредственные манипуляции с атрибутом по имени
`-name:String`	Имя, тип и закрытая видимость — манипуляции с атрибутом с помощью специальных операций
`-name[1..3]:String`	Кратность (для хранения фамилии, имени и отчества)
`-name:String = "Novikov"`	Дополнительно указано начальное значение
`+name:String {readOnly}`	Атрибут объявлен не меняющим своего значения после начального присваивания и открытым

Операции и методы

Операция – спецификация действия с объектом

Метод – выполняемый алгоритм

Сигнатура операции

видимость ИМЯ (параметры) : тип {свойства}

- Видимость: + – # ~
- ИМЯ : операция экземпляра, ИМЯ : операция класса, *ИМЯ* : абстрактная операция
- Параметры:

  направление ПАРАМЕТР : тип = значение

Параметры

- Терминология: параметры – формальные
  аргументы – фактические
- Способы передачи параметров:
  - по ссылке
  - по значению
- Направления передачи параметров в UML:

Ключевое слово	Назначение параметра
`in`	Входной параметр — аргумент д. б. значением, которое используется в операции, но не изменяется
`out`	Выходной параметр — аргумент д. б. хранилищем, в которое операция помещает значение
`inout`	Входной и выходной параметр — аргумент д. б. хранилищем, содержащим значение
`return`	Значение, возвращаемое операцией

Свойства операции

1. Параллелизм (`concurrency`) - когда имеется несколько потоков управления

Значение	Описание
`{sequential}`	Операция не допускает параллельного вызова (дальнейшее поведение системы не определено)
`{guarded}`	Выполняется только один из вызовов — остальные задерживается
`{concurrent}`	Допускается произвольное число параллельных вызовов

2. Побочные эффекты (`isQuery`)
   - True - не меняет состояния системы
   - False *(по умолчанию)* - операция меняет состояние системы: присваивает новые значения атрибутам, создает или уничтожает объекты и т. п.

Примеры описания операций

Пример	Пояснение
`move`	Только имя операции
`+move(in from, in to)`	Видимость операции, направления передачи и имена параметров
`+move(in from : Dpt, in to : Dpt)`	Подробное описание сигнатуры: указаны видимость операции, направления передачи, имена и типы параметров
`+getName():String {isQuery}`	Функция, возвращающая значение атрибута и не имеющая побочных эффектов
`+setPwd(in pwd:String = "password")`	Процедура, для которой указано значение аргумента по умолчанию

Интерфейсы и типы данных

- Интерфейс — это именованный набор абстрактных составляющих

- Тип данных — это:
  - множество значений (потенциально бесконечное)
  - конечное множество операций
  - значения не обладают индивидуальностью

Типы данных

Тип можно указать для:

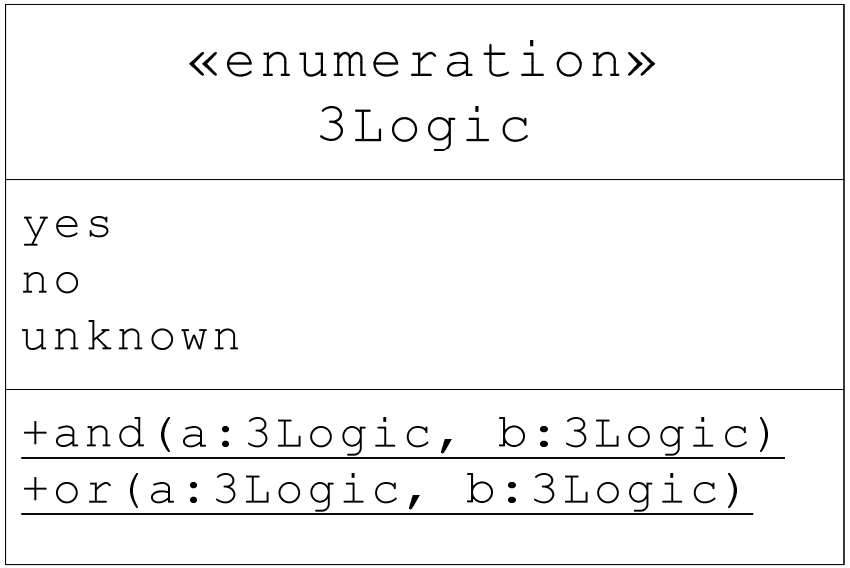
- атрибутов классов, параметров операций, ролей полюсов ассоциаций, квалификаторов полюсов ассоциаций, параметров шаблонов

В качестве типа используется классификатор → имеется набор базовых классификаторов = типы данных:

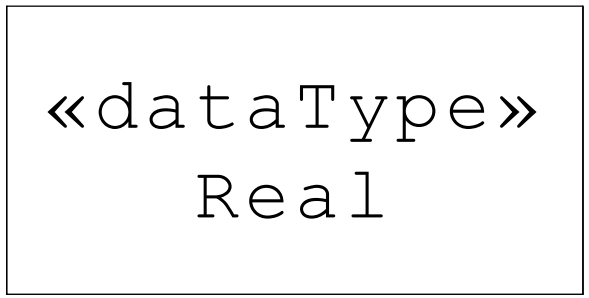
1. Примитивные типы: Integer, Boolean, String, UnlimitedNatural
2. Типы, которые определены в языке программирования
3. Типы, которые определены пользователем:
   - стереотип «enumeration» или «dataType»

Примеры типов данных

- Тип данных (в UML) — это классификатор, экземпляры которого не обладают индивидуальностью (identity)



Перечислимый тип данных «Трехзначная логика»



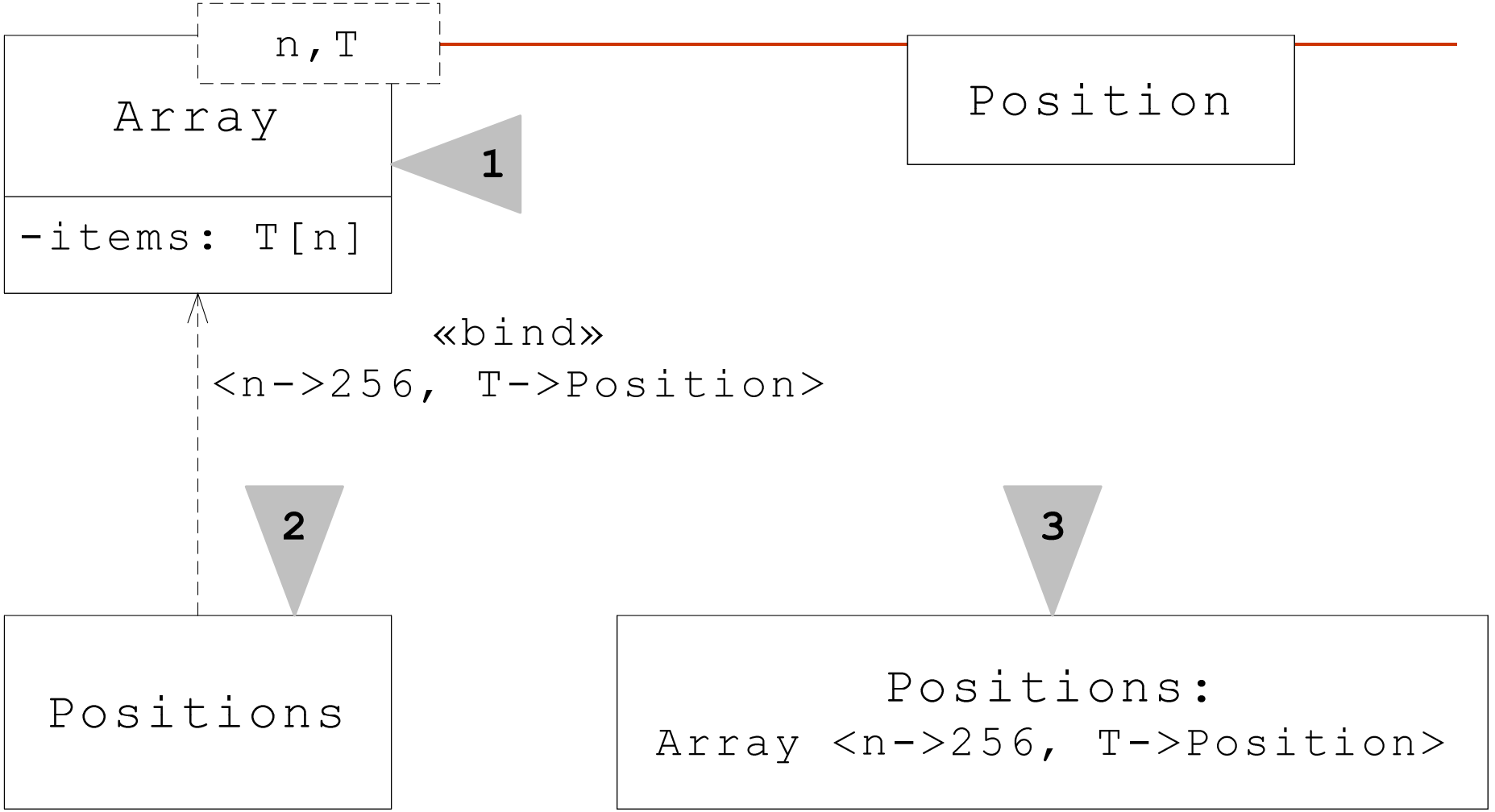
Тип данных «Действительное число»

Шаблоны

- Шаблон — это классификатор с параметрами
- Параметром может быть любой элемент описания классификатора ИМЯ
- Связывание (binding) — указывание явных значений аргументов:
  - явное связывание — зависимость со стереотипом «bind», в которой указаны значения аргументов
  - неявное связывание — определение класса, имя которого имеет формат

имя_классификатора : имя_шаблона < аргументы >

Связывание шаблона



1) шаблон
2) явное инстанцирование шаблона
3) неявное инстанцирование шаблона

Идентификация классов: словарь предметной области

Словарь предметной области:

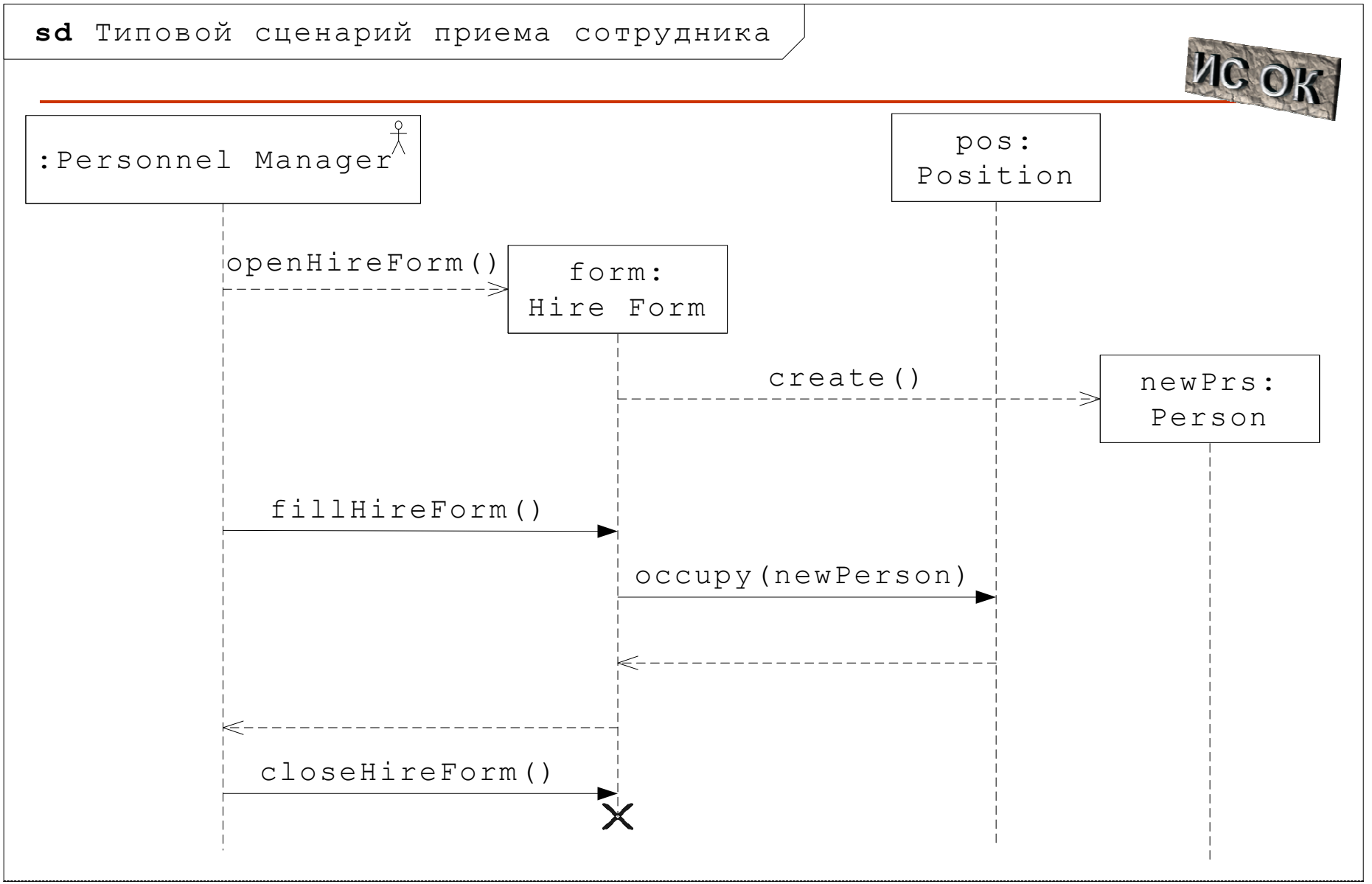
существительные в ТЗ

- прием
- перевод
- увольнение
- сотрудник
- создание
- ликвидация
- подразделение
- вакансия
- сокращение
- должность

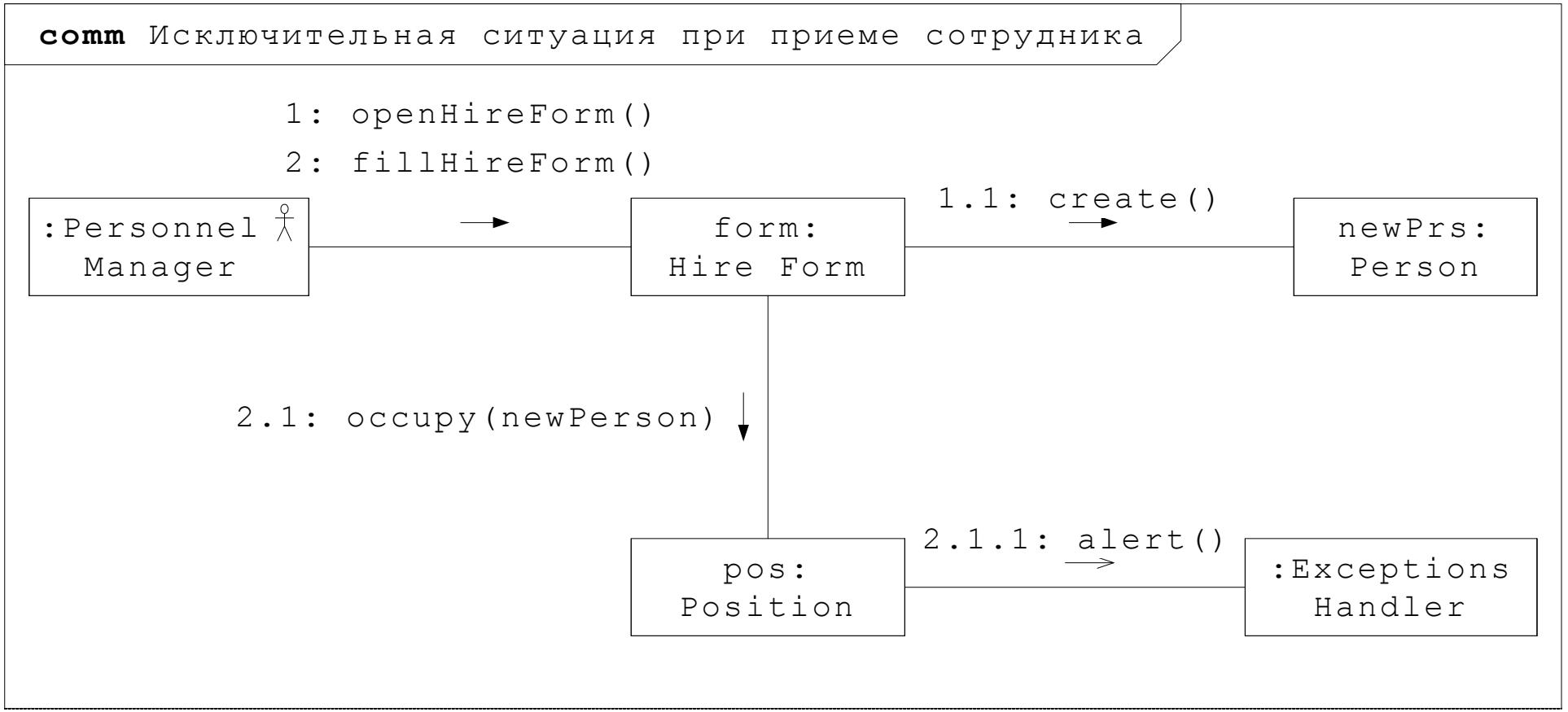
- сотрудник
- подразделение
- вакансия
- должность

- Сотрудник (Person)
- Подразделение (Department)
- Должность (Position)

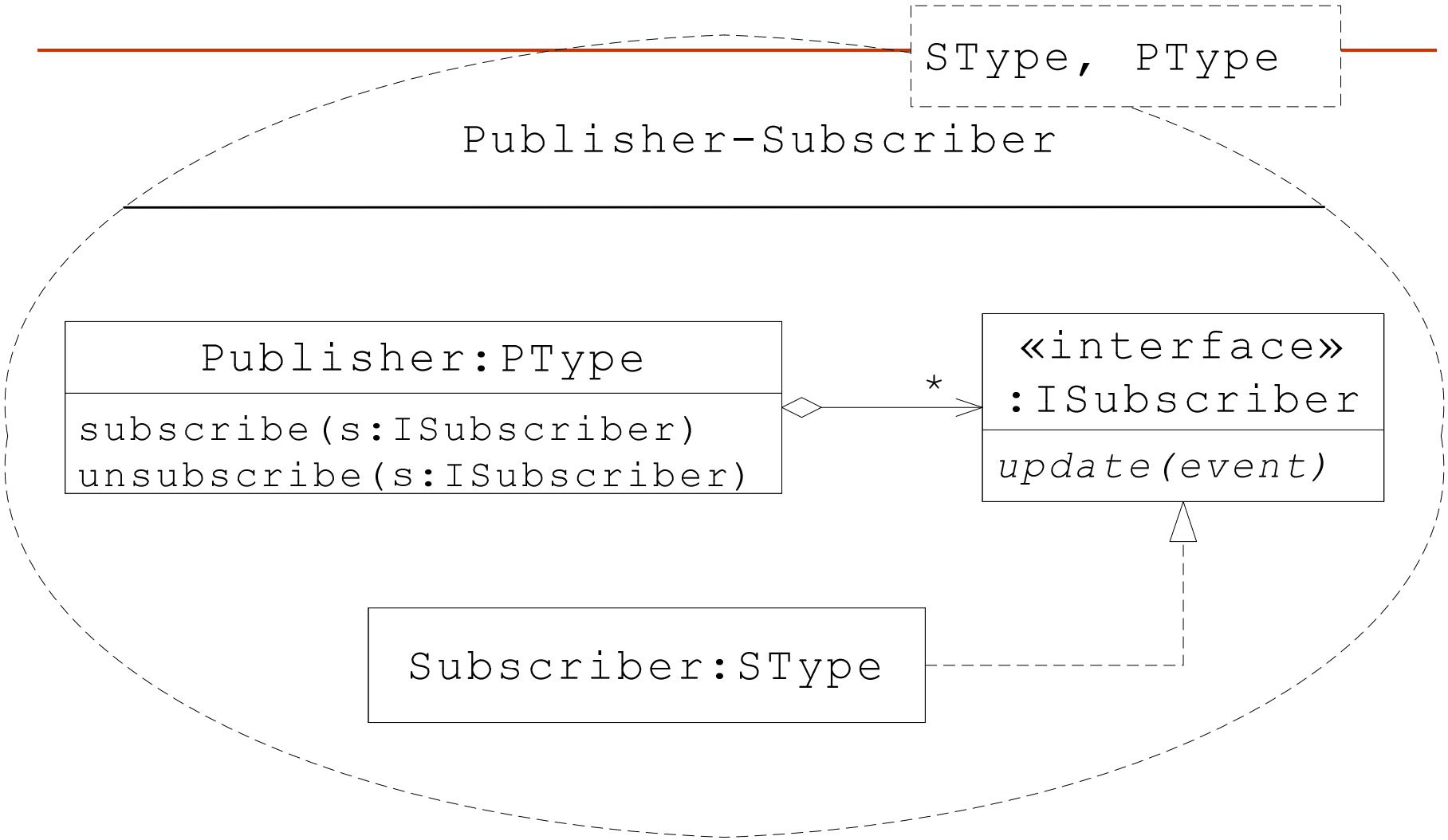
Идентификация классов: реализация вариантов использования (i)



Идентификация классов: реализация вариантов использования (ii)



Идентификация классов: образцы проектирования



3. Отношения на диаграмме классов

Сущности на диаграммах классов связываются отношениями:

- Ассоциации
  - в том числе агрегации и композиции
- Обобщения
- Зависимости
- Реализации

Отношения зависимости (i)

Стандартные стереотипы зависимости между классами и объектами

Стереотип	Применение
«bind»	Подстановка параметров в шаблон
«call»	Операция зависимого класса вызывает операцию независимого класса
«derive»	«Может быть вычислен по»
«friend»	Назначает специальные права видимости

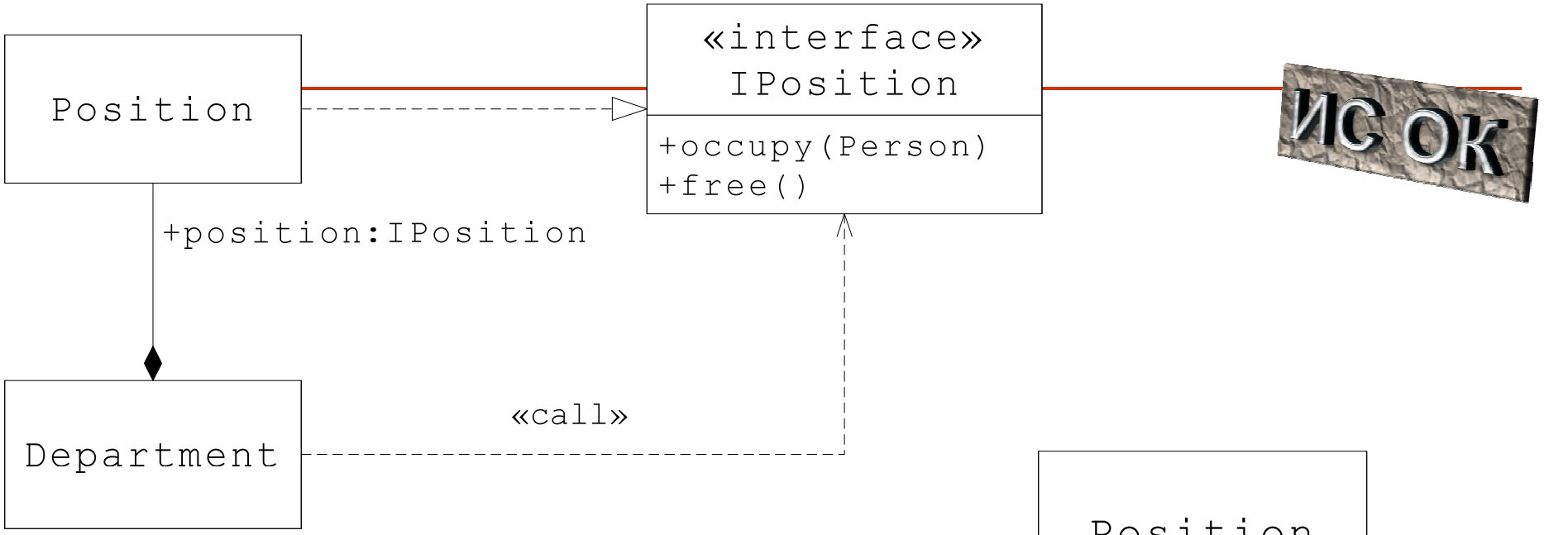
Отношения зависимости (ii)

Стереотип	Применение
«instanceOf»	Зависимый объект является экземпляром независимого класса
«instantiate»	Операции независимого класса создают экземпляры зависимого класса
«powertype» в UML 1	Экземплярами зависимого класса являются подклассы независимого класса
«refine»	Указывает, что зависимый класс уточняет независимый
«use»	Зависимый класс каким-либо образом использует независимый класс

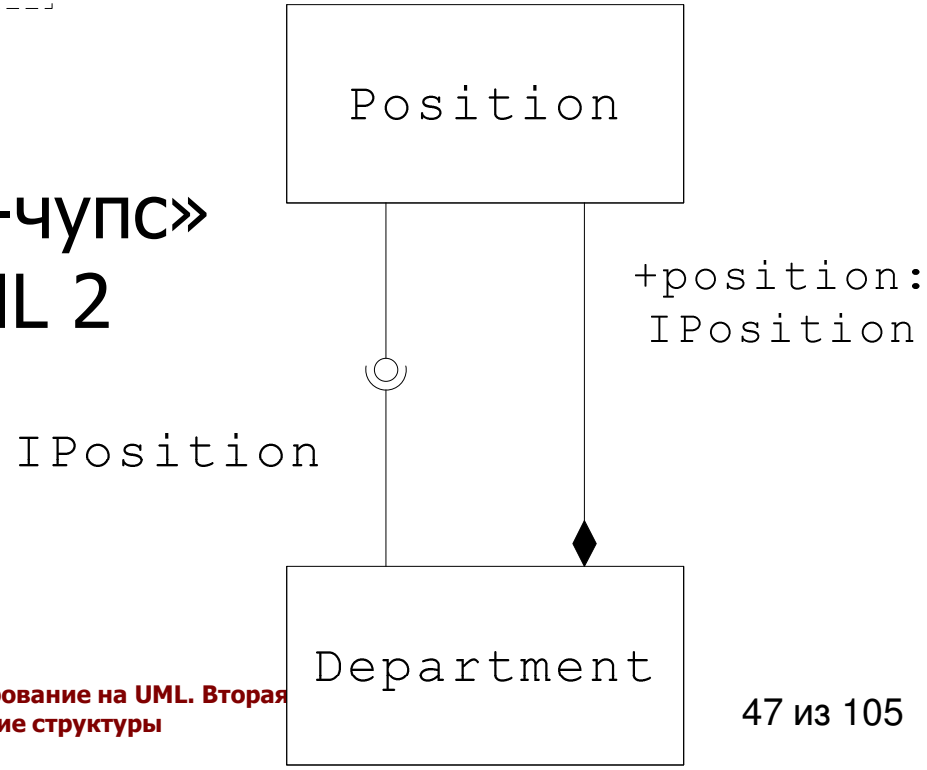
Отношение реализации

- Классификатор (в частности, класс) использует интерфейс — зависимость со стереотипом «call» (требуемый интерфейс)
  - указывает, что в операциях независимого класса вызываются операции зависимого класса
- Классификатор реализует интерфейс — реализация (обеспеченный интерфейс)
- Класс может реализовывать много интерфейсов (и наоборот)

Пример реализации и использования интерфейсов



Нотация «чупа-чупс» (lolly-pop) UML 2



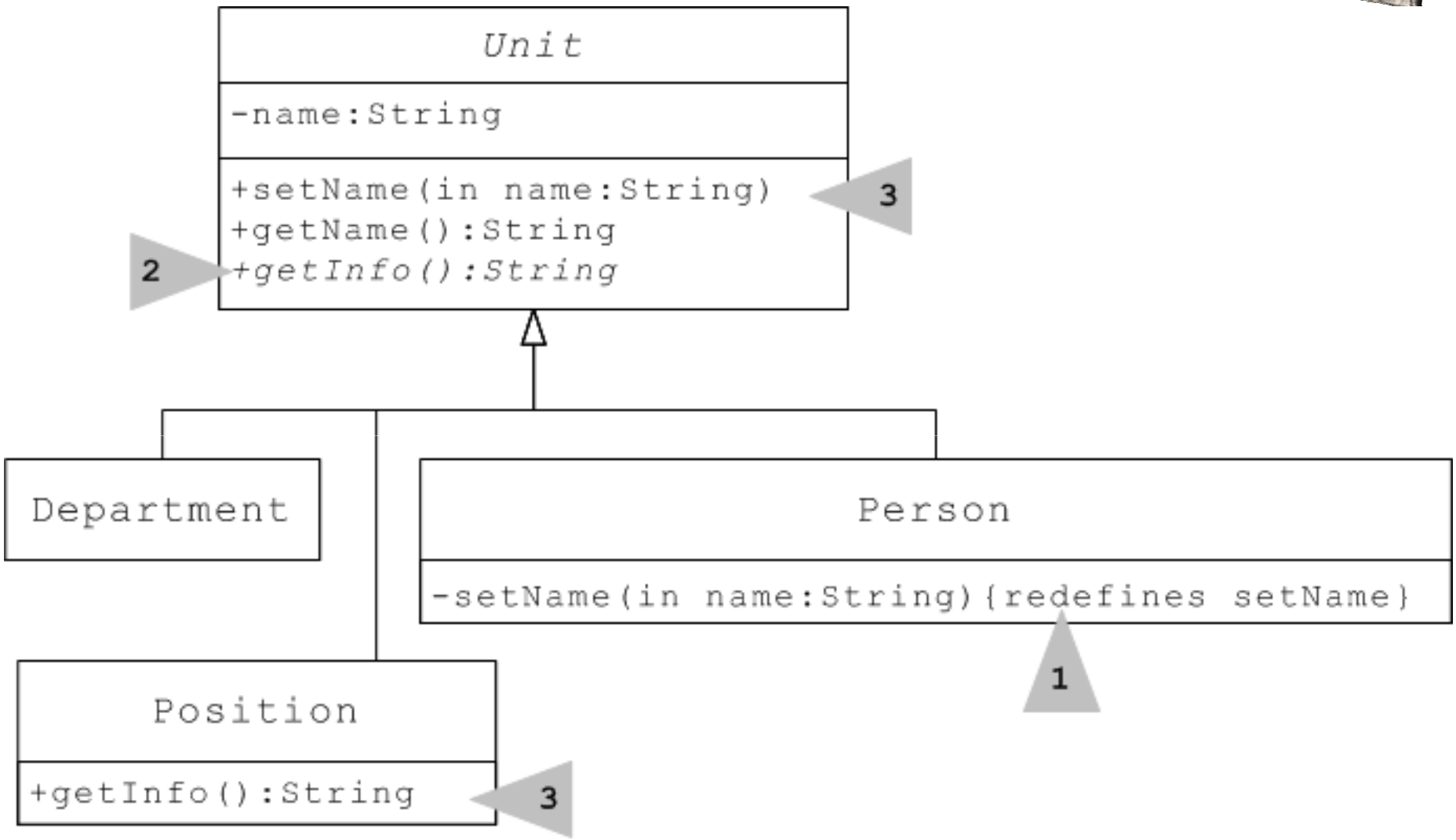
Отношение обобщения

Применение обобщений:

- Множественное наследование (multiple inheritance)
  - Класс является подклассом нескольких суперклассов
  - «Проблема множественного наследования находится в головах архитекторов» ☺
- Несколько иерархий обобщения
  - Не требуется, чтобы у базовых классов был общий суперкласс
- Ограничение: частичная упорядоченность
  - Отсутствие циклов в цепочках обобщений

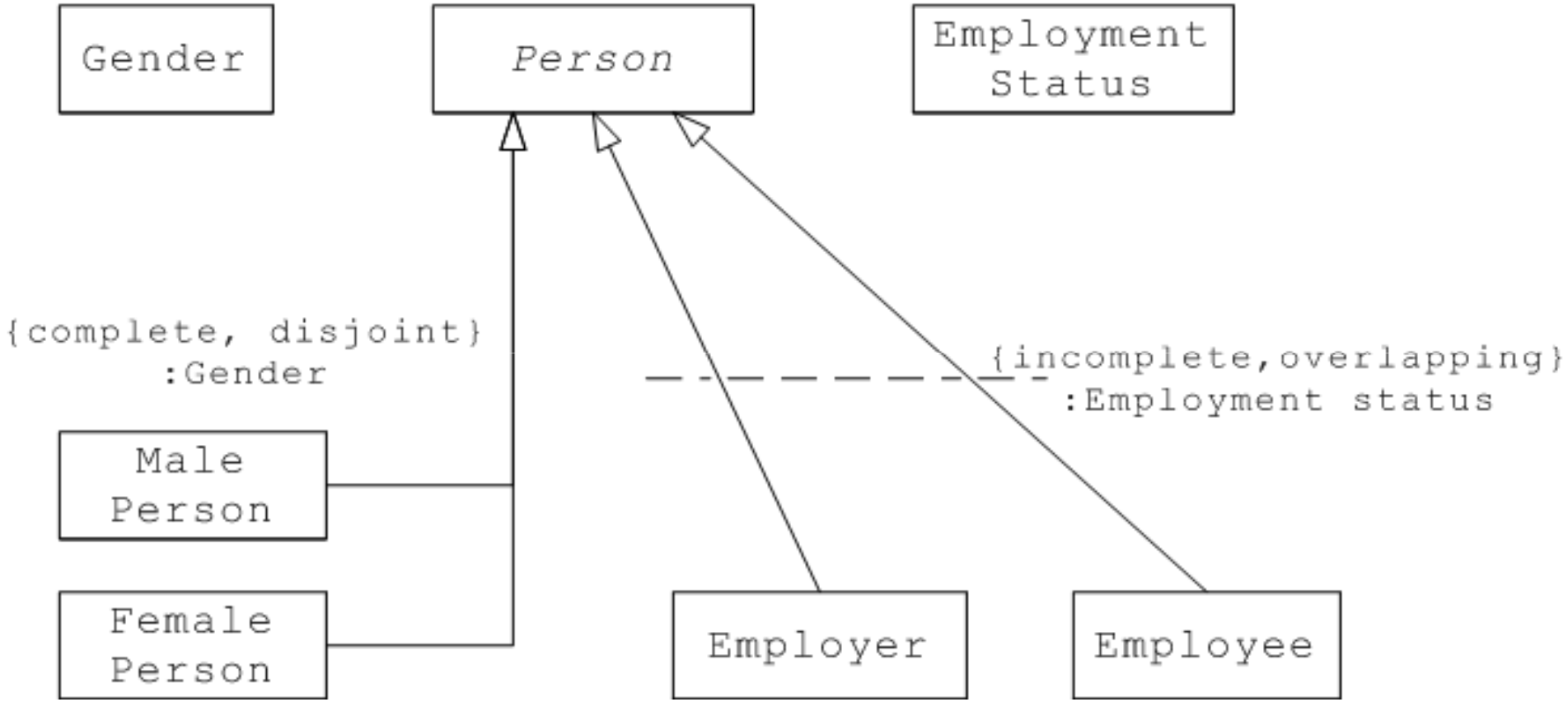
Обобщение

ИС ОК



1) переопределение операции
2) определение абстрактной операции
3) определение операции с реализацией (методом)

Подмножества обобщений



Отношение ассоциации

Ассоциация в UML — это классификатор, экземпляры которого называются связями (link)

- Минимум:

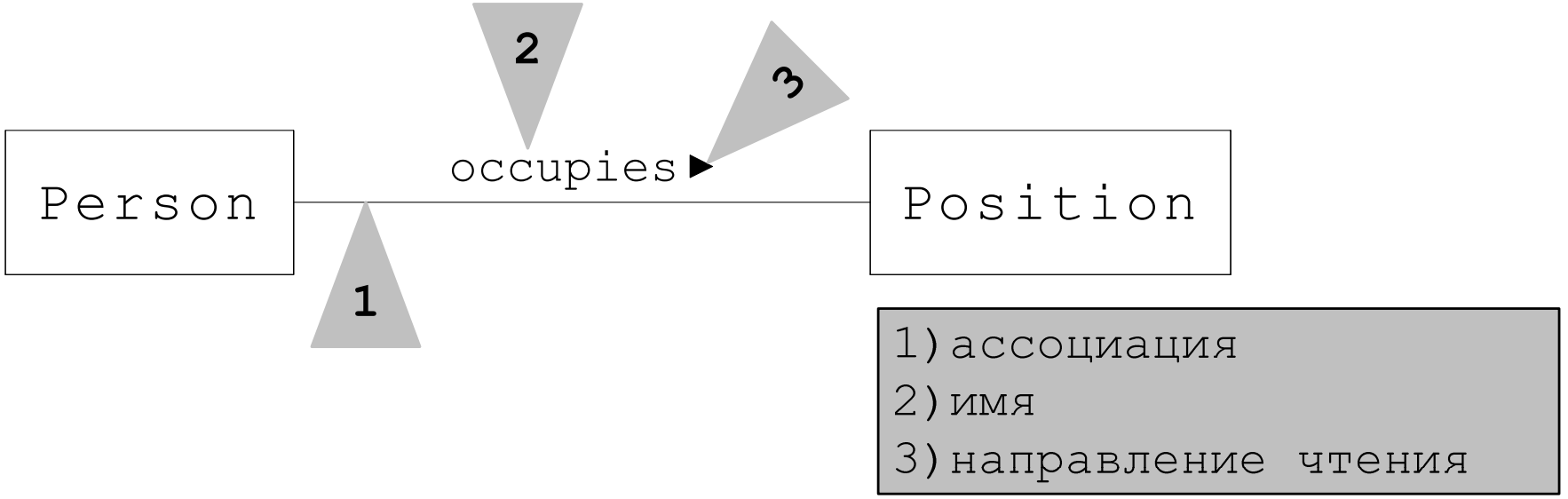
  Объекты ассоциированных классов могут взаимодействовать во время выполнения

Отношение ассоциации: дополнения

- Имя ассоциации (возможно, вместе с направлением чтения)
- Кратность полюса ассоциации
- Агрегации или композиция
- Возможность навигации для полюса ассоциации
- Роль полюса ассоциации
- Многополюсные ассоциации
- Класс ассоциации
- Квалификатор полюса ассоциации
- Видимость полюса ассоциации
- Упорядоченность и изменяемость множества объектов на полюсе ассоциации

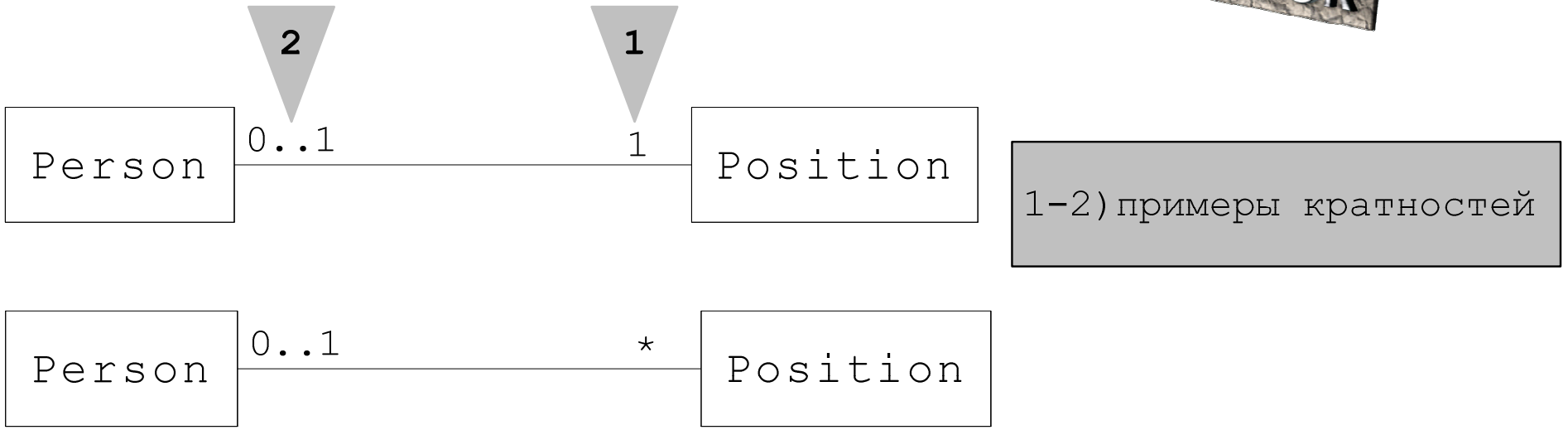
Имя ассоциации

- Позволяет различать ассоциации в модели
- Применяется если > 2 полюсов
- Направление чтения



Кратность полюса ассоциации

- Сколько объектов класса участвуют в связи со стороны данного полюса
- Может быть задана
  - как конкретное число
  - как диапазон возможных значений,
  - как неопределенное число *

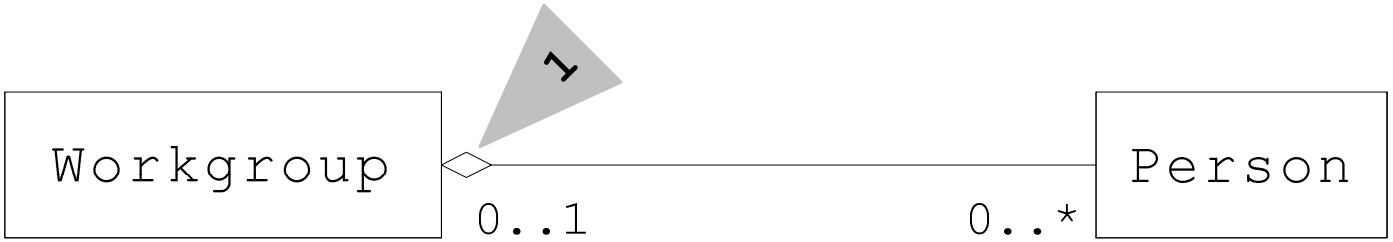


Агрегация и композиция

- Агрегация — это ассоциация между классом A (часть) и классом B (целое)
  - Не изменяет направления навигации и не накладывает ограничений на время жизни
- Композиция – более сильные ограничения:
  - Часть A может входить только в одно целое B
  - Время жизни частей совпадает с временем жизни целого

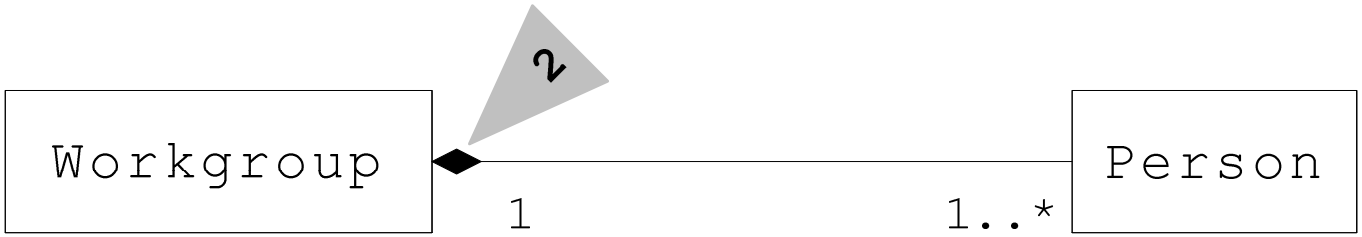
Примеры агрегации и композиции

"Мягкая" структура



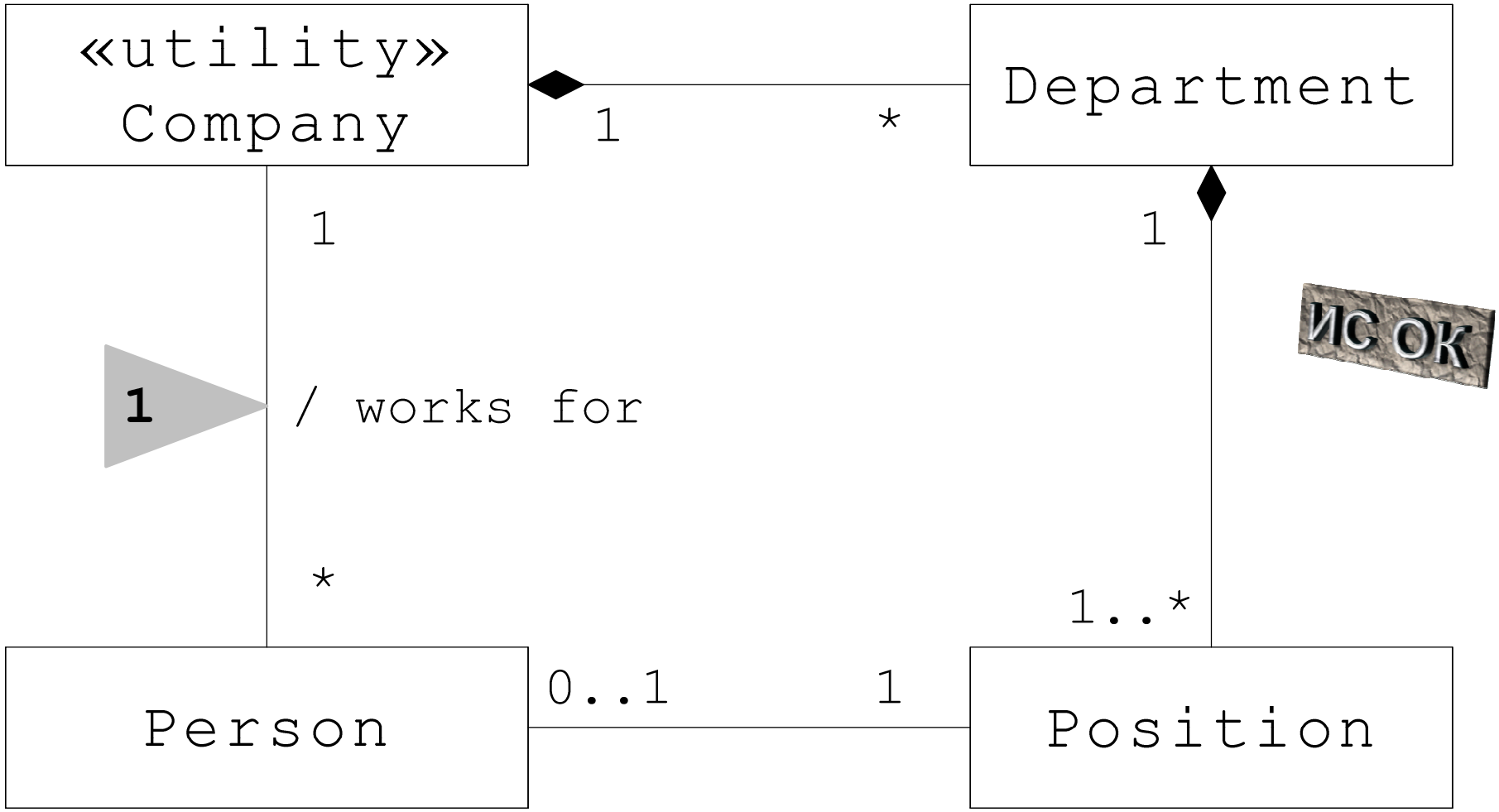
0..* = *

"Жесткая" структура



1) агрегация
2) композиция

Производные элементы

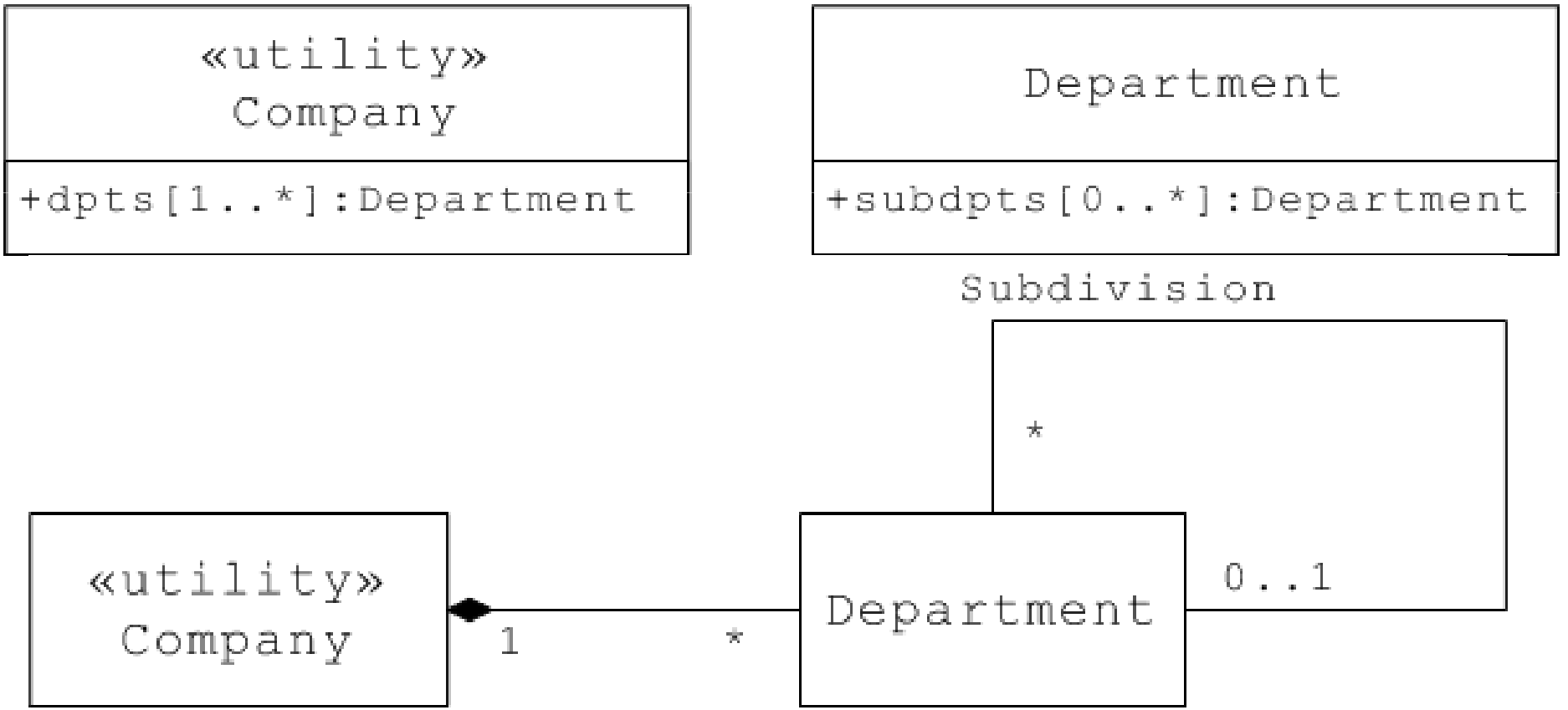


/ works for:

- Производный элемент (derived element) — это элемент, который можно вычислить или определить по другим элементам (для наглядности)

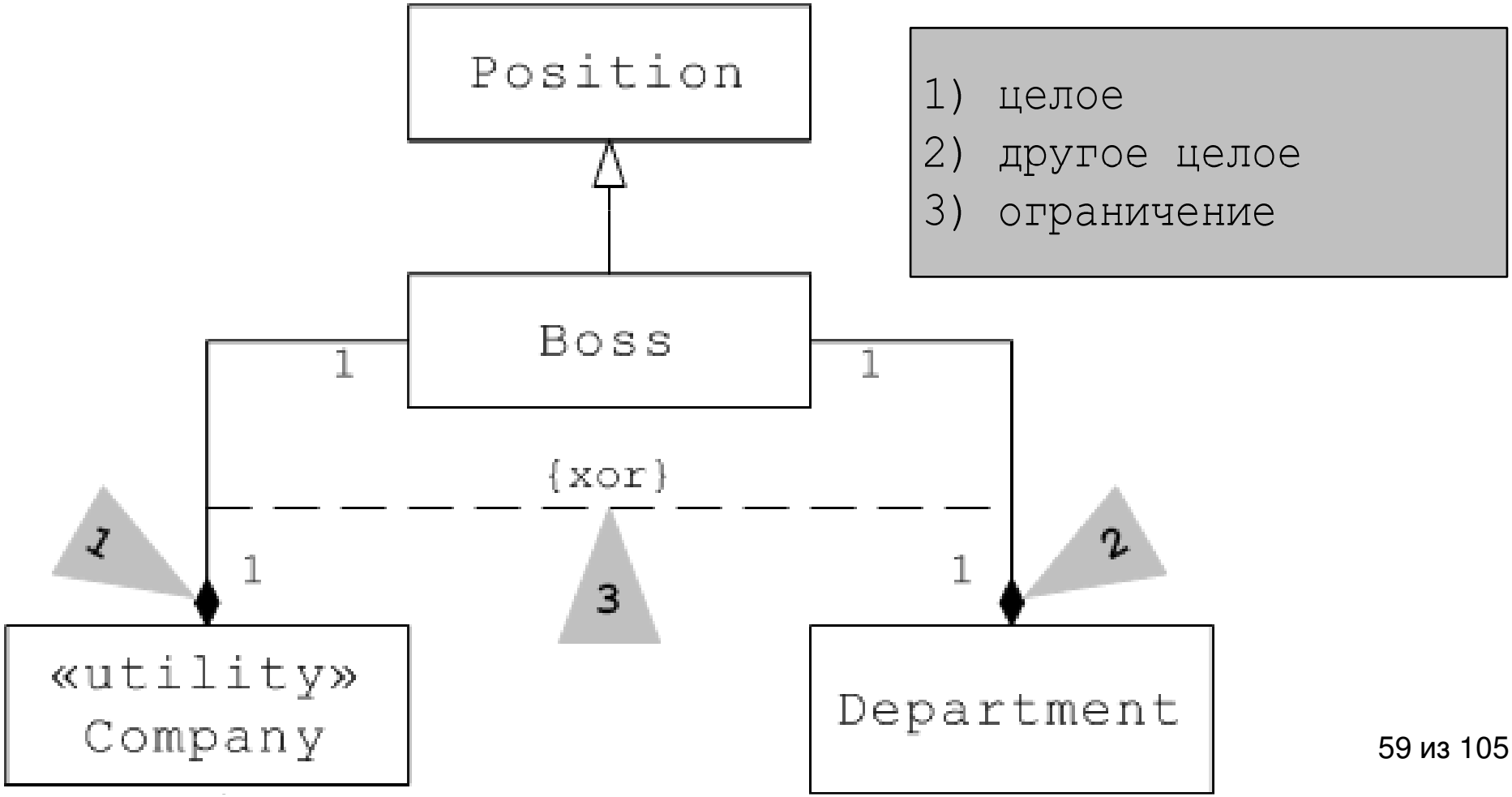
Композиция или атрибуты?

- Если примитивная часть, то атрибут
- Если есть взаимодействие, то композиция

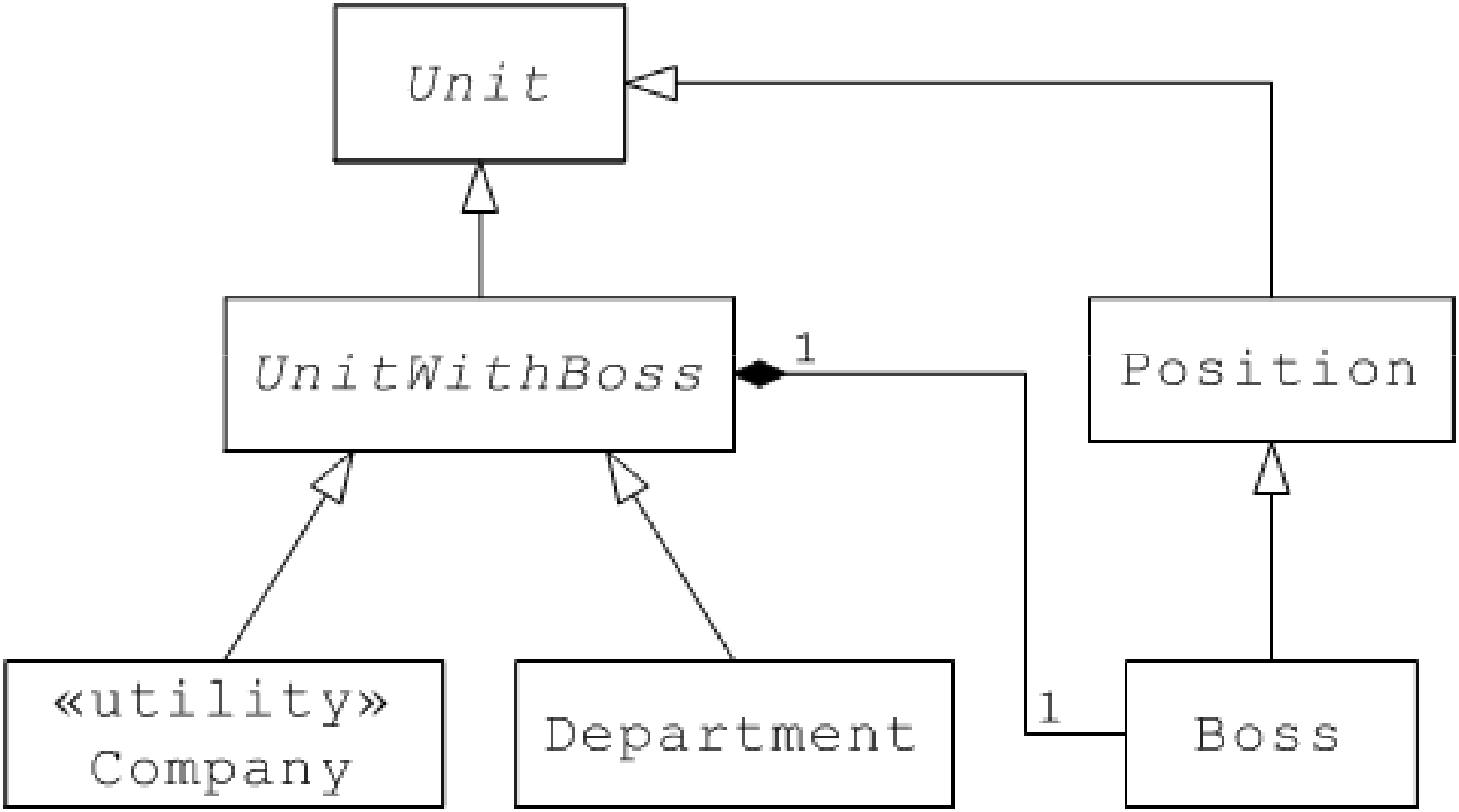


Время жизни частей (i)

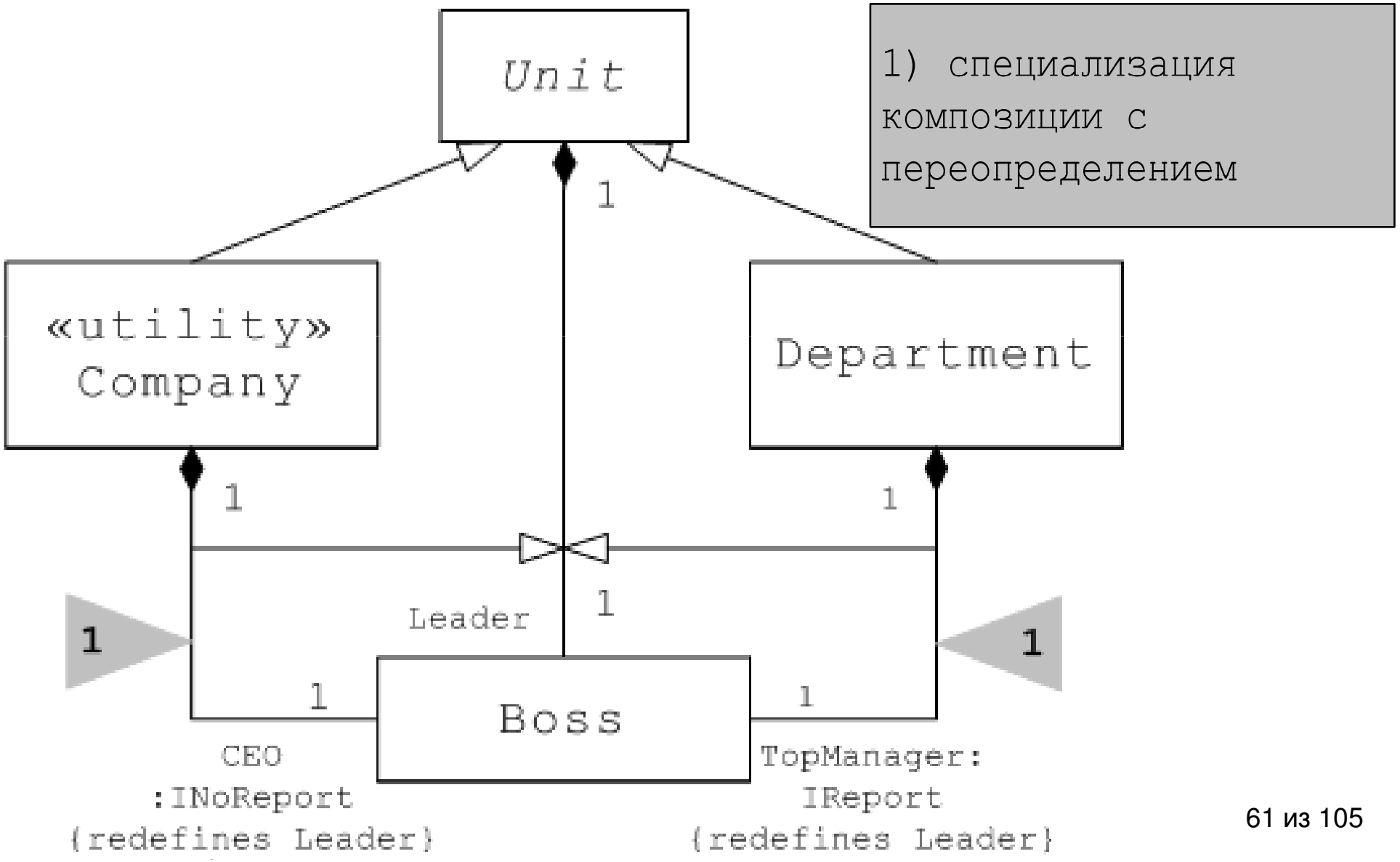
- Часть композита принадлежит только одному целому в каждый момент



Время жизни частей (ii)



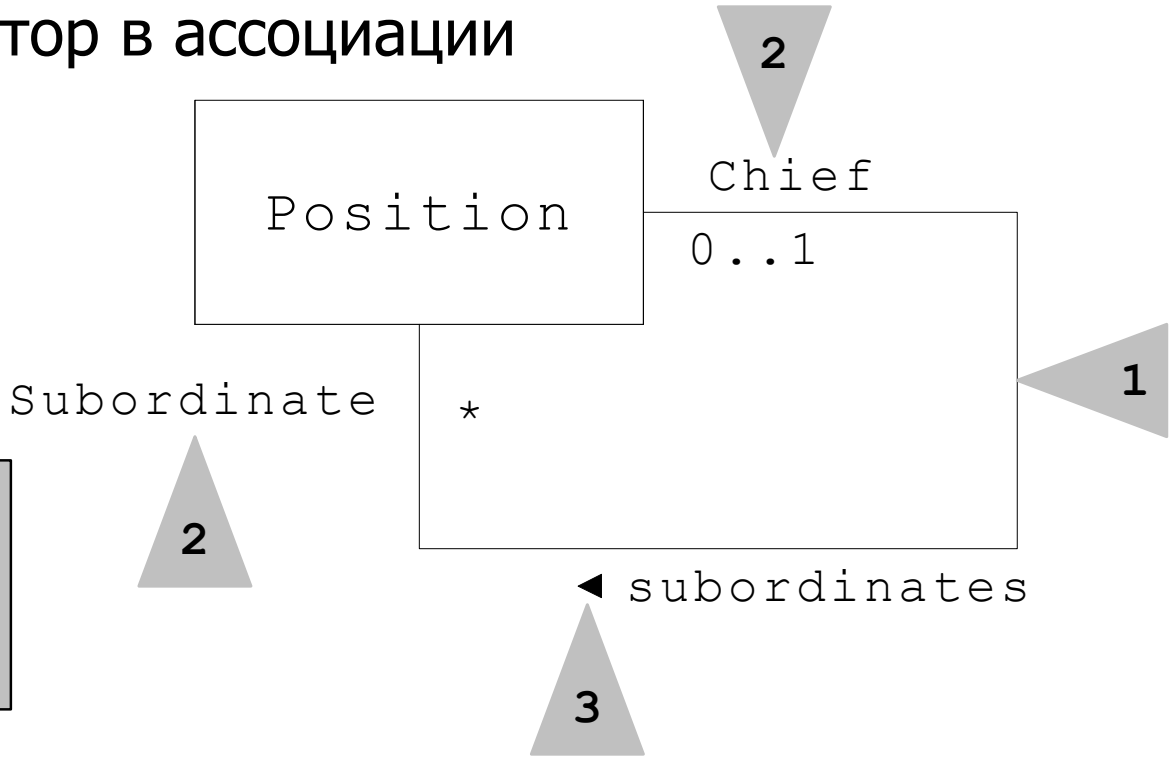
Время жизни частей (iii)



Роль полюса ассоциации

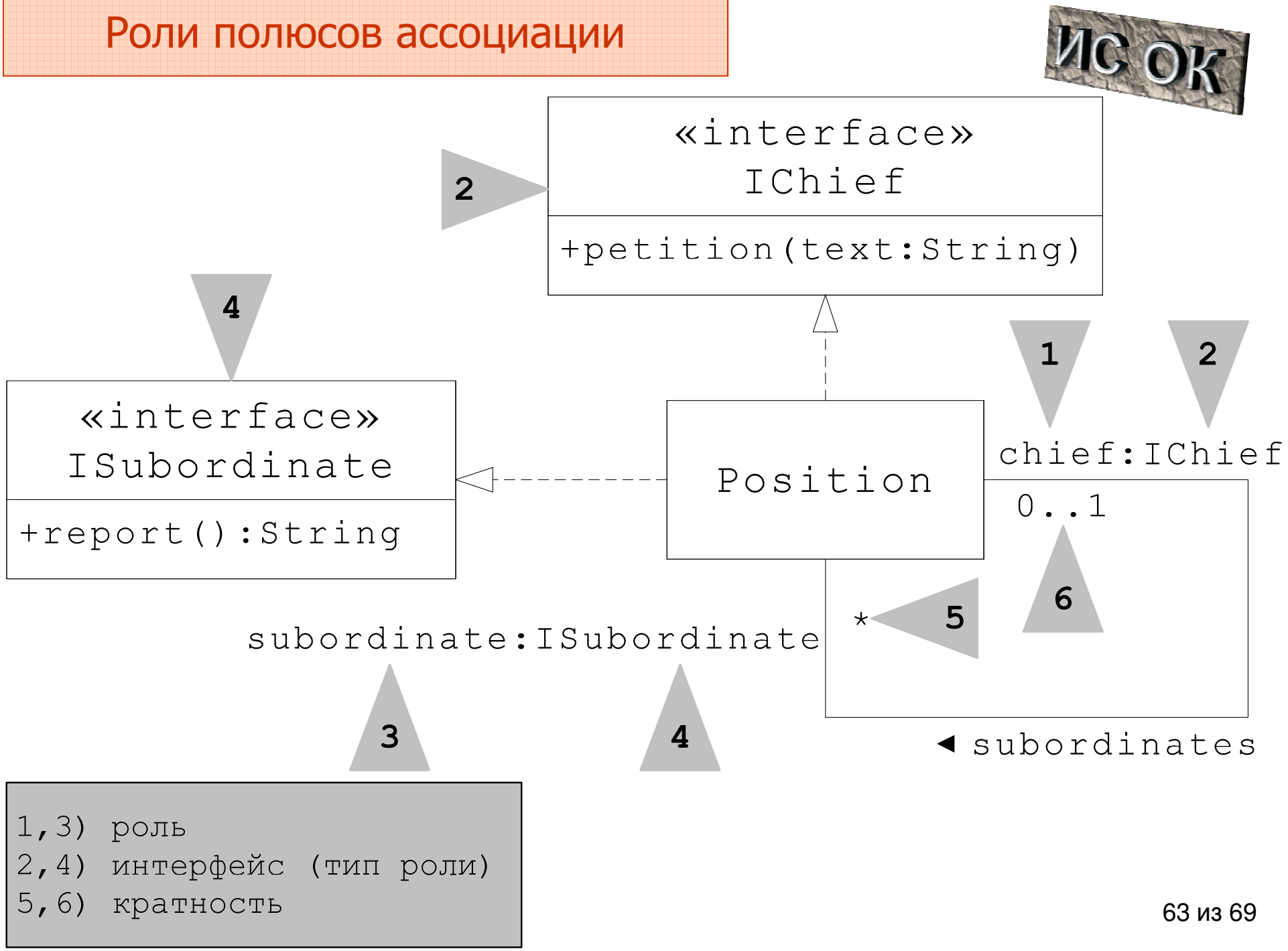
- Роль — это интерфейс, который предоставляет классификатор в данной ассоциации
- Роль полюса ассоциации = спецификатор интерфейса — это способ указать, как именно участвует классификатор в ассоциации

видимость ИМЯ : тип



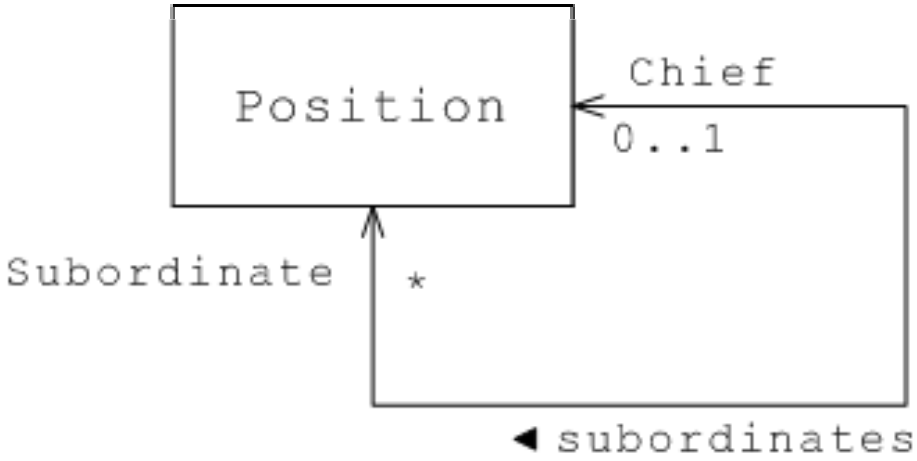
1) ассоциация
2) роль
3) направление чтения

Роли полюсов ассоциации



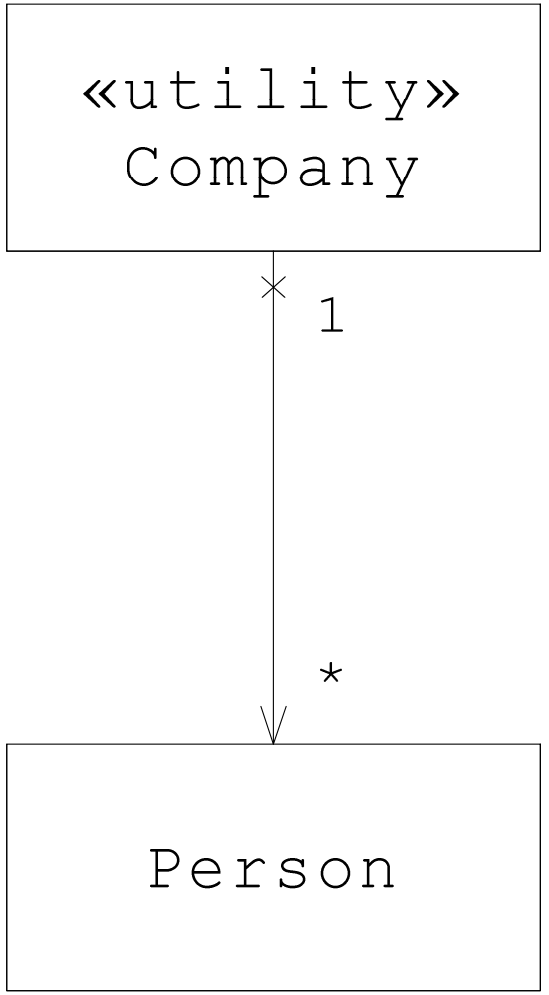
Направление навигации

- Возможность навигации (navigability) определяет, можно ли эффективно получить доступ к объектам класса



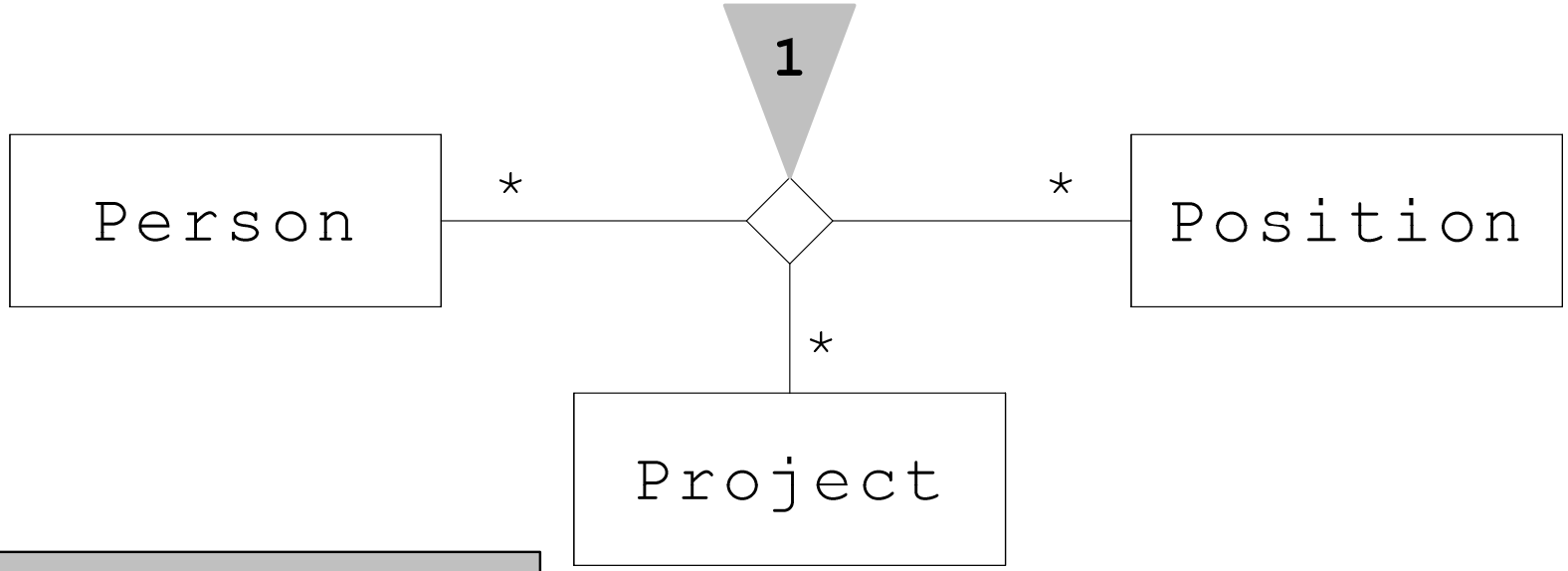
Навигация в обе стороны

Навигация возможна только в направлении от Company к Person, но не наоборот



Многополюсные ассоциации

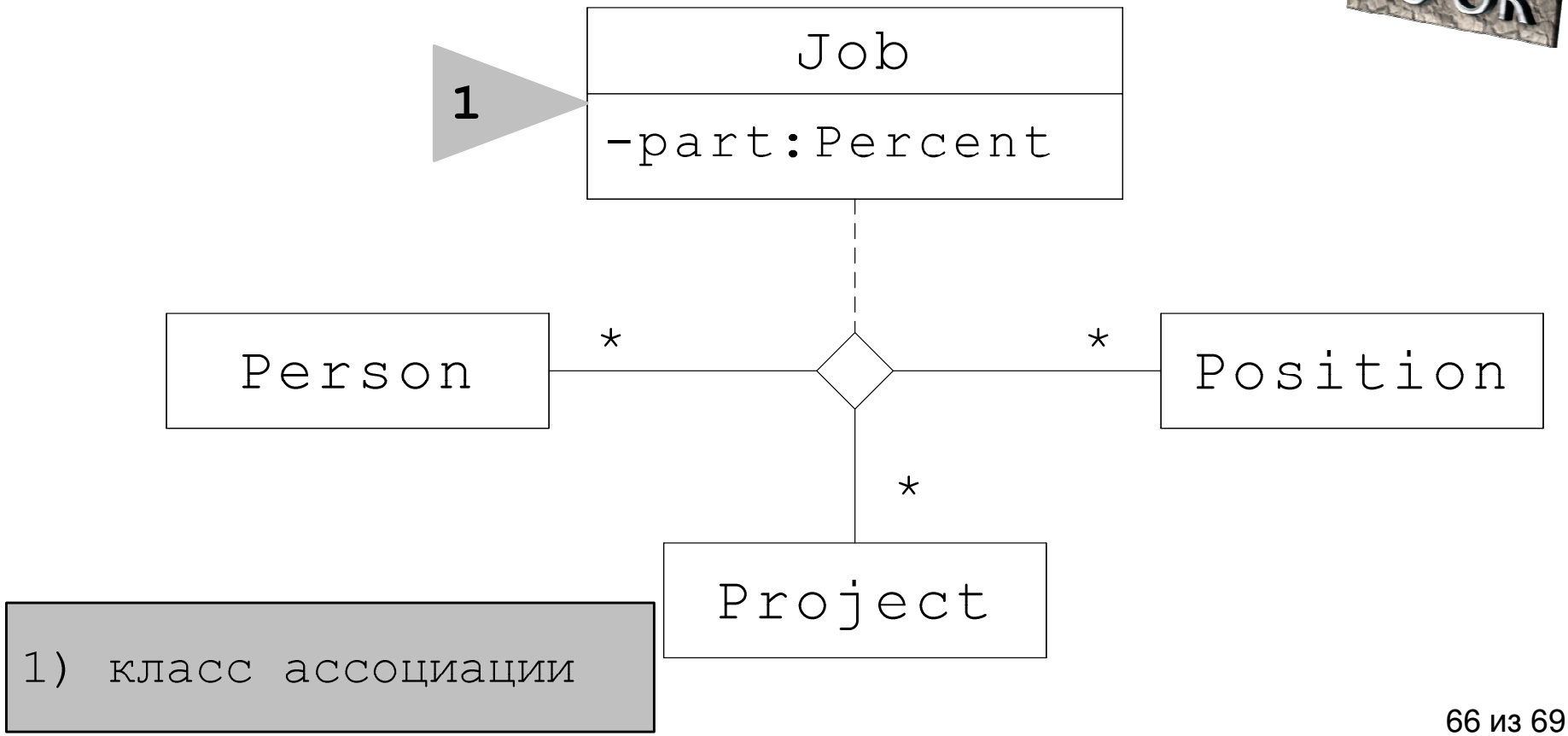
- Один и тот же сотрудник может участвовать во многих проектах и занимать различные должности



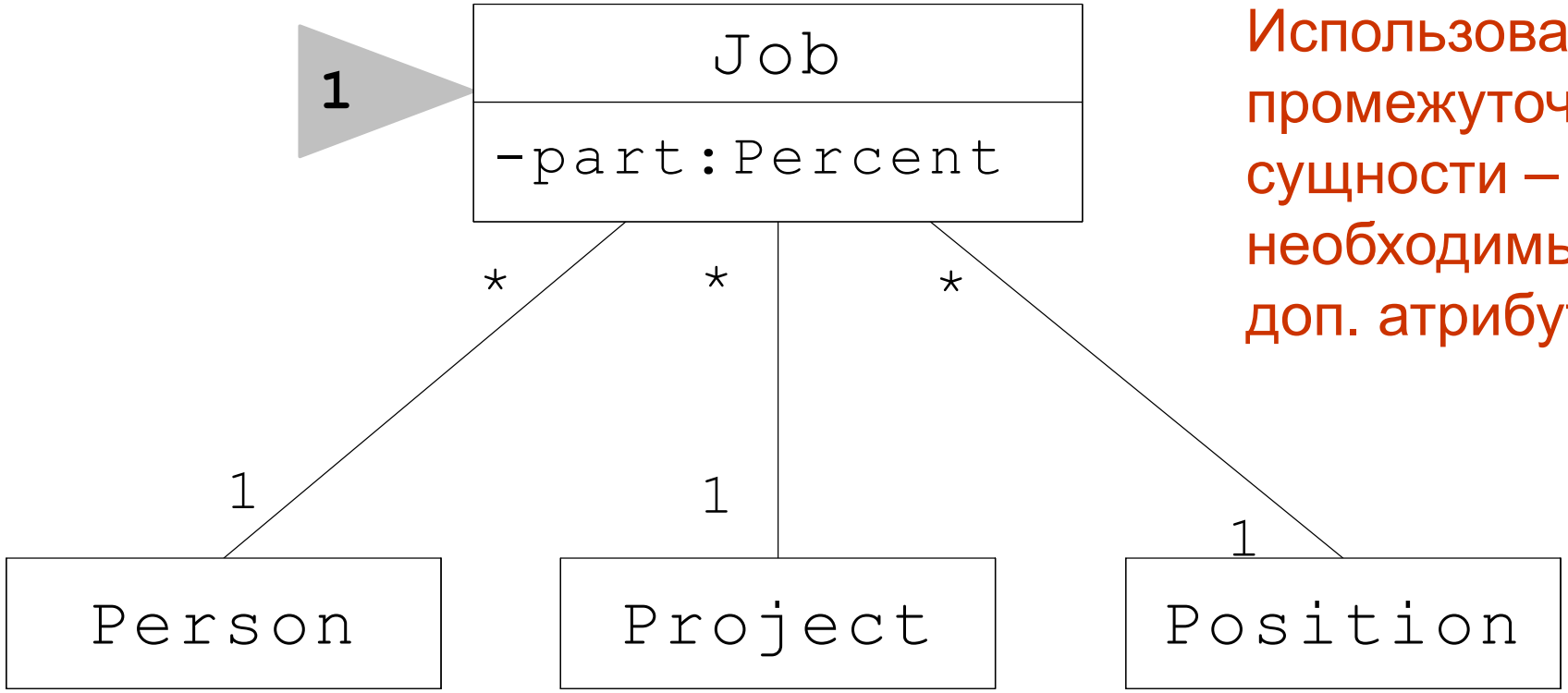
1) многополюсная ассоциация

Класс ассоциации

Класс ассоциации — ассоциация, имеющая в своем составе свойства и составляющие класса



Альтернатива классу ассоциации



Использование промежуточной сущности – необходимы доп. атрибуты

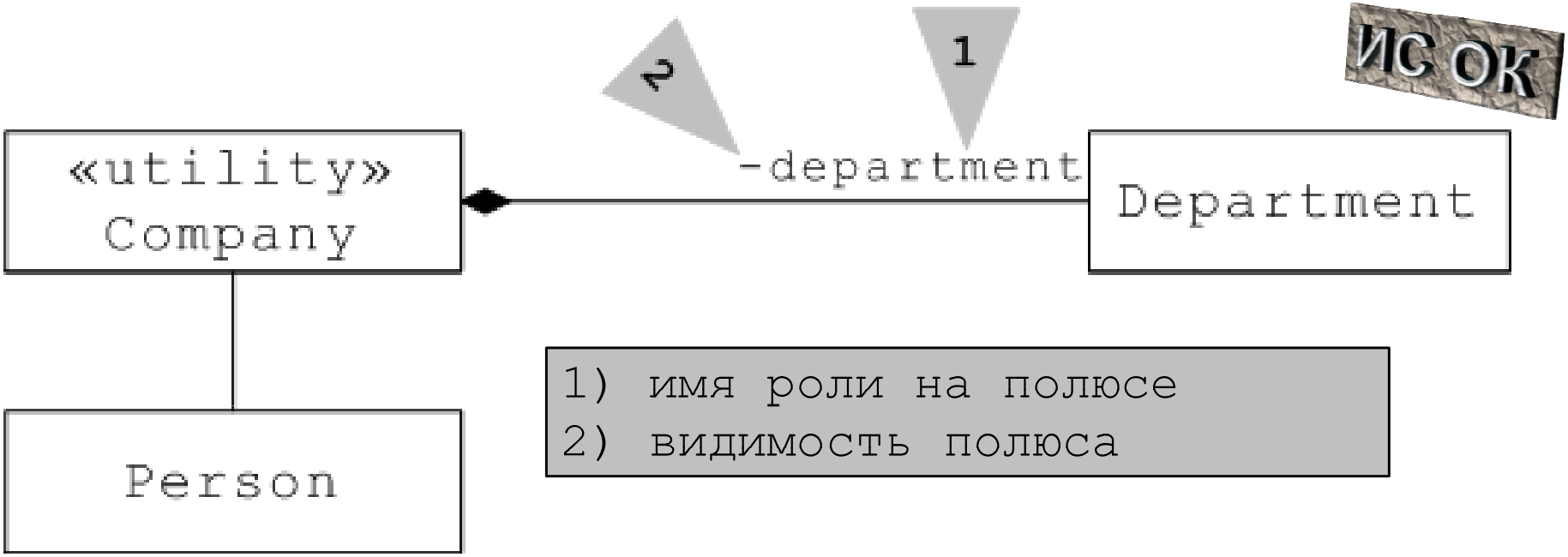
1) дополнительная сущность

Ограничения на полюсах ассоциации (i)

Упорядоченность объектов (ordering) / уникальность (uniqueness) – наличие / отсутствие одинаковых объектов

	Упорядоченное множество	Неупорядоченное множество
Есть одинаковые элементы	{sequence} Последовательность	{bag} Мультимножество
Нет одинаковых элементов	{ordered} Упорядоченное множество	{set} Множество

Видимость полюса ассоциации



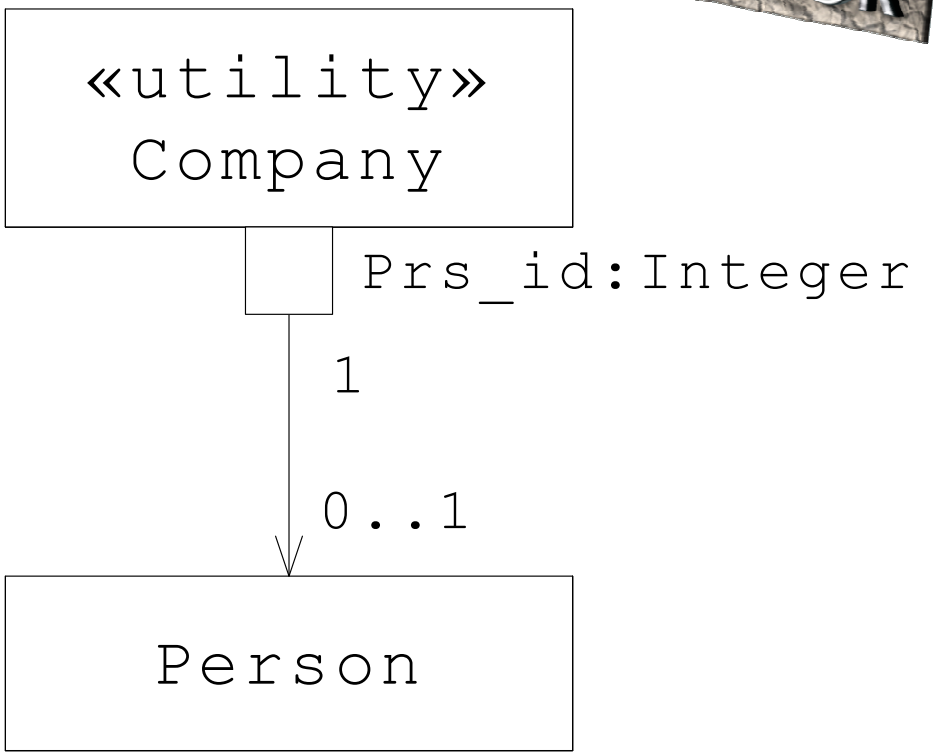
- Видимость полюса ассоциации – указание того, является ли классификатор присоединенный к данному полюсу видимым для классификаторов (кроме непосредственно свзязанных), маршруты из которых ведут к нему

Упорядоченность и изменяемость

- Многоугольник (Polygon) = упорядоченное множество точек (Point)
- Состав вершин, после того, как он определен, не может меняться
- Вершина может принадлежать нескольким многоугольникам

Квалификатор полюса ассоциации

Квалификатор полюса ассоциации — это атрибут полюса ассоциации, позволяющий выделить объект класса, присоединенного к другому полюсу ассоциации



Сводная диаграмма классов ИС ОК

Советы по проектированию

- Описывать структуру удобнее параллельно с описанием поведения
- Не обязательно включать в модель все классы сразу, на первых порах достаточно 10%
- Не обязательно определять все составляющие класса сразу
- В процессе работы диаграмма должна легко охватываться одним взглядом — не обязательно показывать на диаграмме все составляющие класса и их свойства
- Не обязательно определять все отношения между классами сразу — пусть класс на диаграмме "висит в воздухе", ничего с ним не случится

Мы действительно сами следуем своим советам ☺

4. Диаграммы реализации

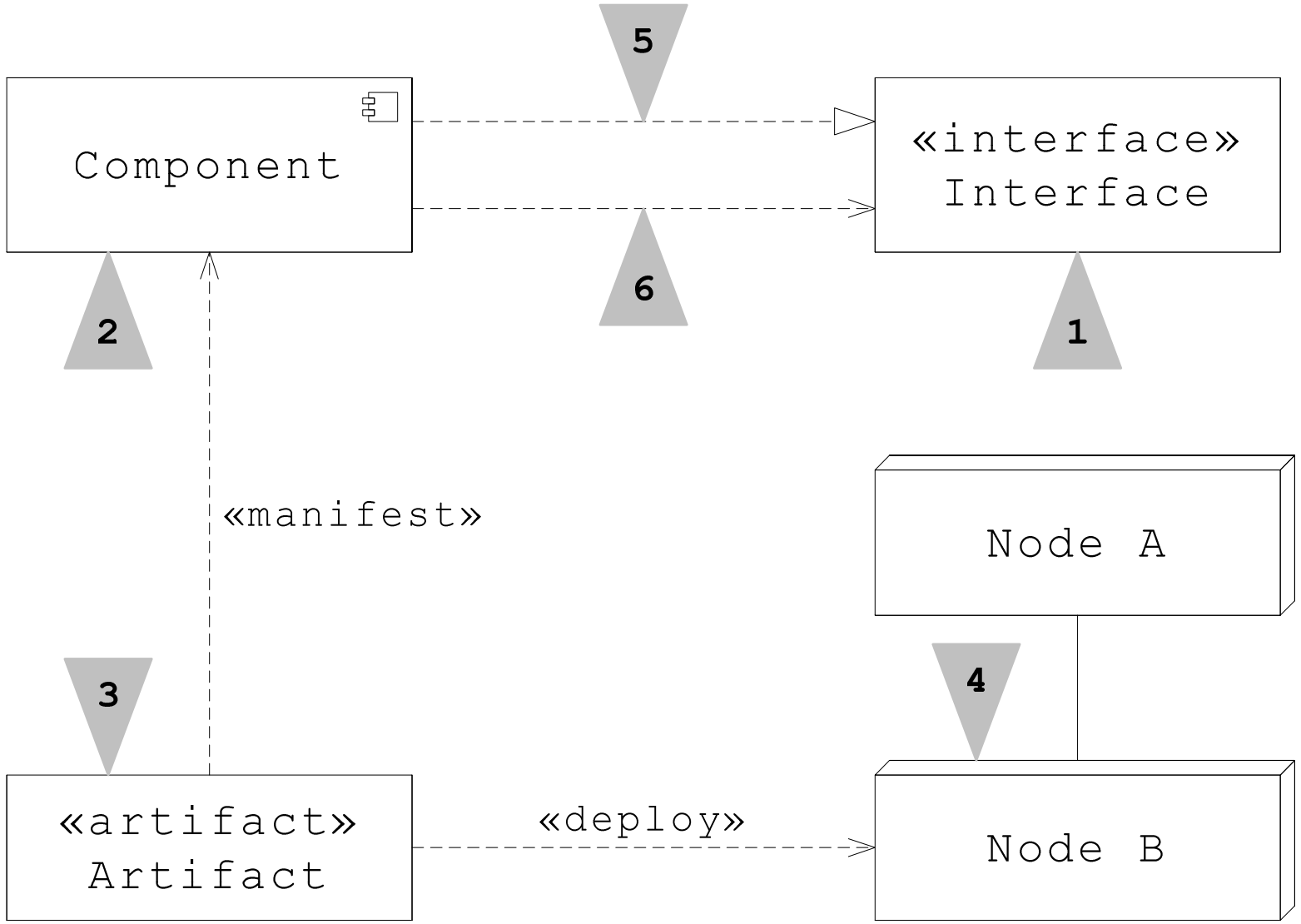
Реализация системы:

- Диаграммы компонентов – компоненты
- Диаграммы размещения – артефакты и их размещение по узлам

Объединяют:

- Структуры логических элементов – компонентов
- Отображения компонентов на физические элементы – артефакты
- Структуры используемых ресурсов (nodes) с распределенными по ним артефактами
- Применяются на позднейших фазах разработки

Отношения между сущностями



1) интерфейс
2) компонент
3) артефакт
4) узел
5) реализация интерфейса
6) использование интерфейса

Сущности: интерфейс

Интерфейс может быть (в зависимости от того, как используется):

- Если классификатор реализует интерфейс — обеспеченным (реализация)
- Если классификатор вызывает операции интерфейса — требуемым (зависимость)

Сущности: компонент

Компонент UML (≠ компонент в сборочном программировании) —

- является частью модели
- описывает логическую сущность, которая существует только на этапе проектирования (design time)
- в дальнейшем можно связать с физической реализацией (артефактом) времени исполнения (run time)
- взаимодействие описывается набором интерфейсов

Стандартные стереотипы компонентов

Стереотип	Описание
«buildComponent»	компонент, определяющий набор компонентов для разработки приложения
«entity»	постоянно хранимый информационный компонент, представляющий некоторое понятие предметной области
«service»	функциональный компонент без состояния, возвращающий запрашиваемые значения без побочных эффектов
«subsystem»	единица иерархической декомпозиции большой системы

Сущности: артефакт

Артефакт — это любой созданный искусственно элемент программной системы: исполняемые файлы, исходные тексты, веб-страницы, справочные файлы, сопроводительные документы ...

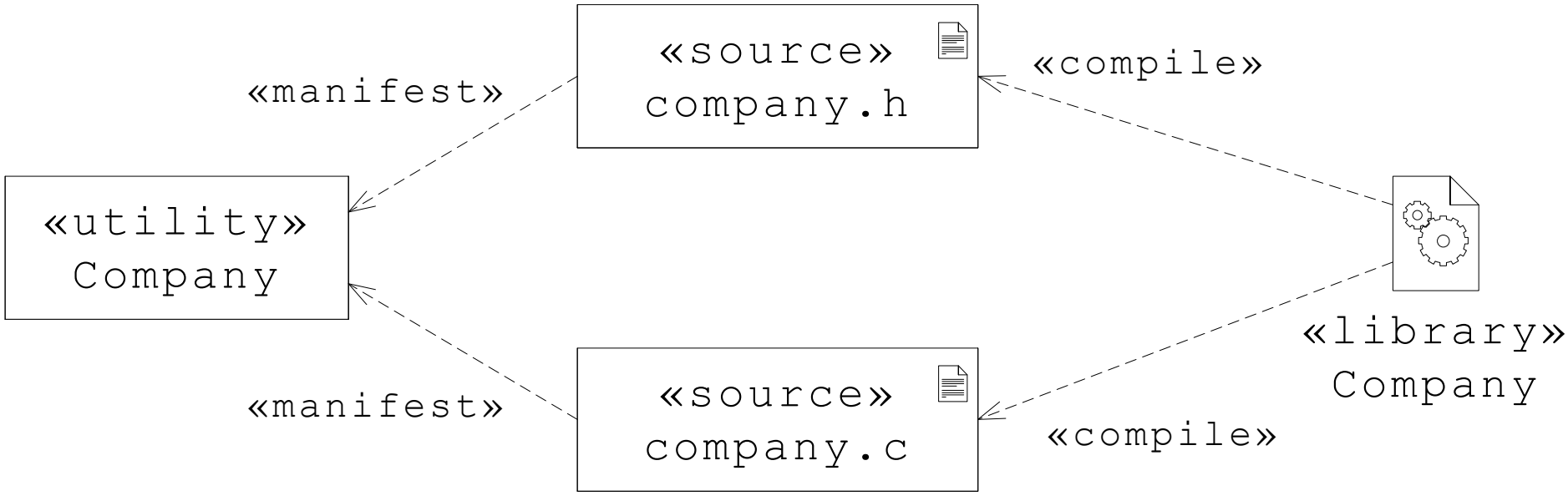
Стандартные стереотипы артефактов

Стереотип	Описание
«file»	физический файл
«document»	файл, который не является ни файлом исходных текстов, ни исполняемым файлом
«executable»*	выполнимая программа любого вида
«library»	статическая или динамическая библиотека
«script»	файл, содержащий текст, допускающий интерпретацию соответствующими программным средствами
«source»	файл с исходным кодом программы

* Подразумевается по умолчанию

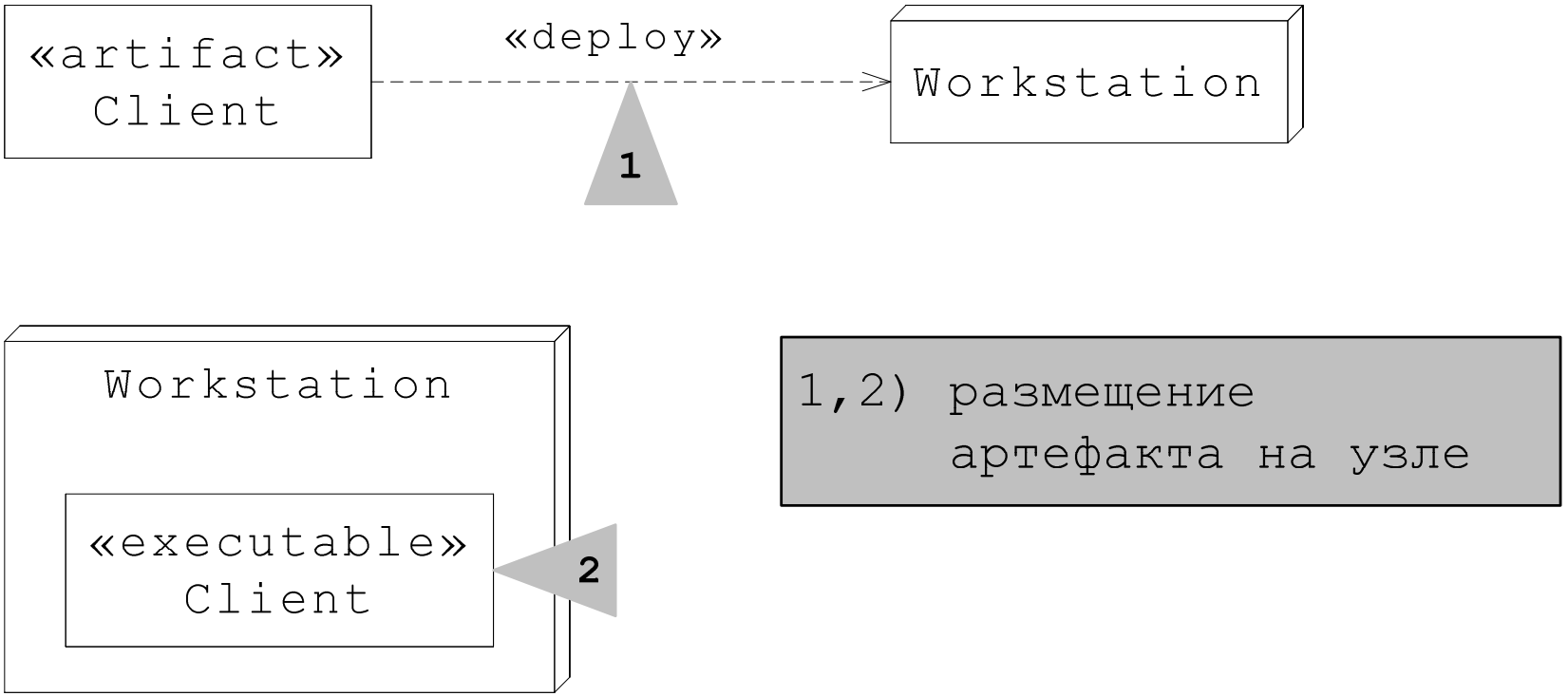
Отношение манифестации

- Манифестация — это отношение, связывающее элемент модели и его физическую реализацию в виде артефакта
- Отношение типа "многие ко многим"



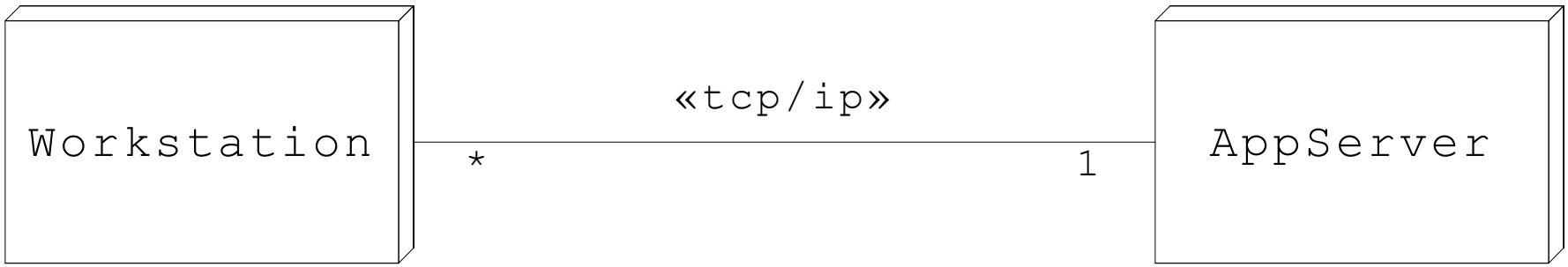
Сущности: узел

- Узел (node) — это физический вычислительный ресурс, участвующий в работе системы



Ассоциация между узлами

Ассоциация означает наличие канала связи



Доп. информация: стереотипы, ограничения и именованные значения

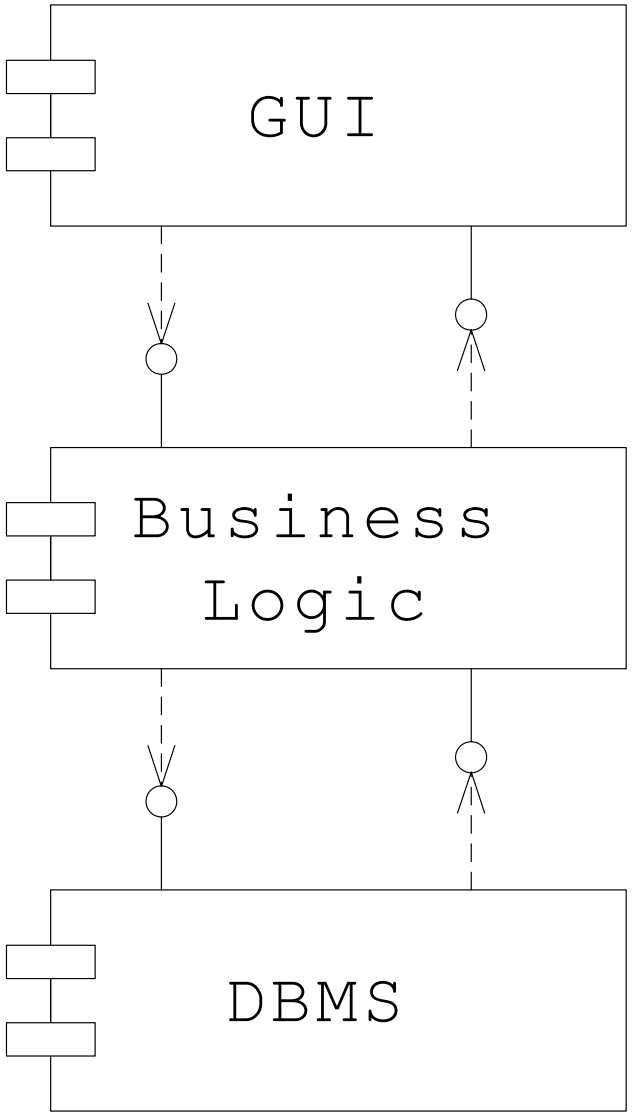
Применение диаграмм реализации

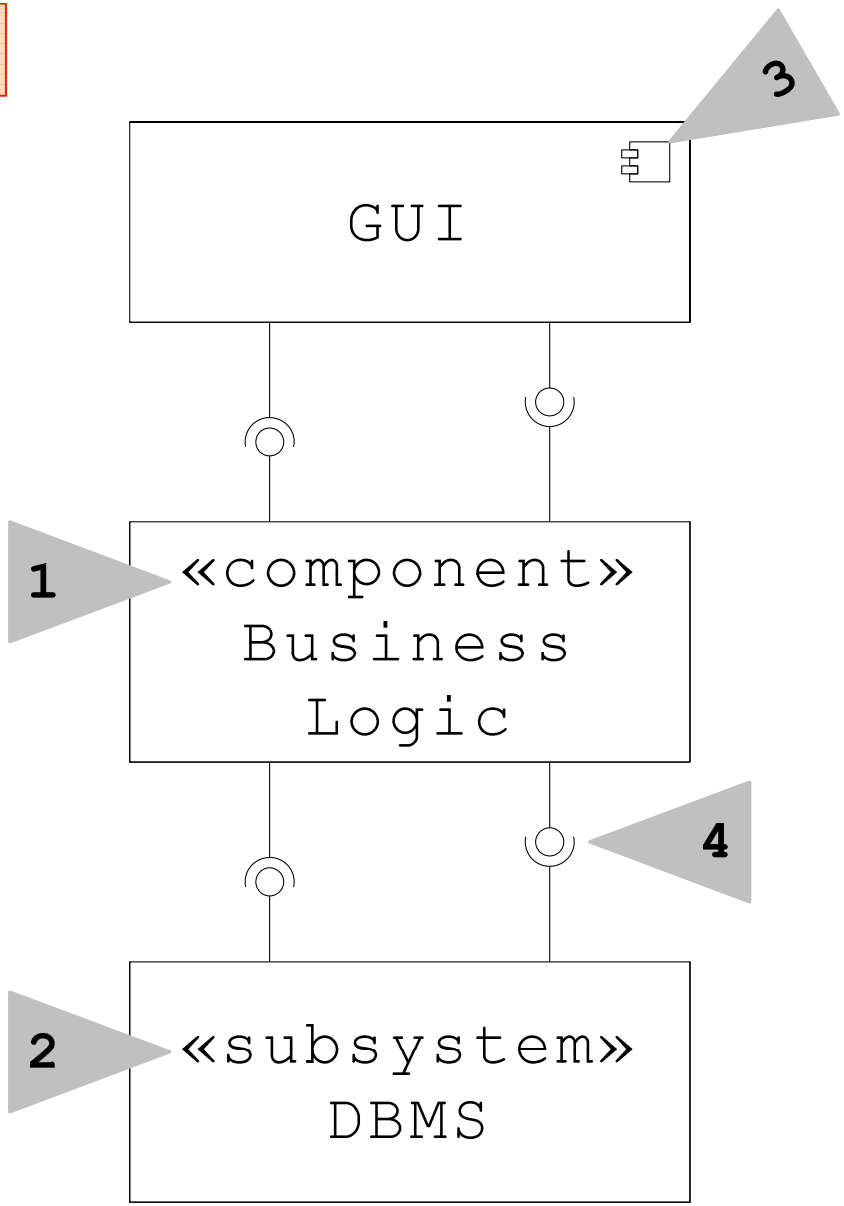
- «Монолитное» настольное приложение — диаграммы размещения не нужны

Диаграммы размещения необходимы:

- Управление конфигурацией – «конструктор»
- Управление версиями
- Моделирование унаследованных приложений и данных
- Моделирование систем динамической архитектуры

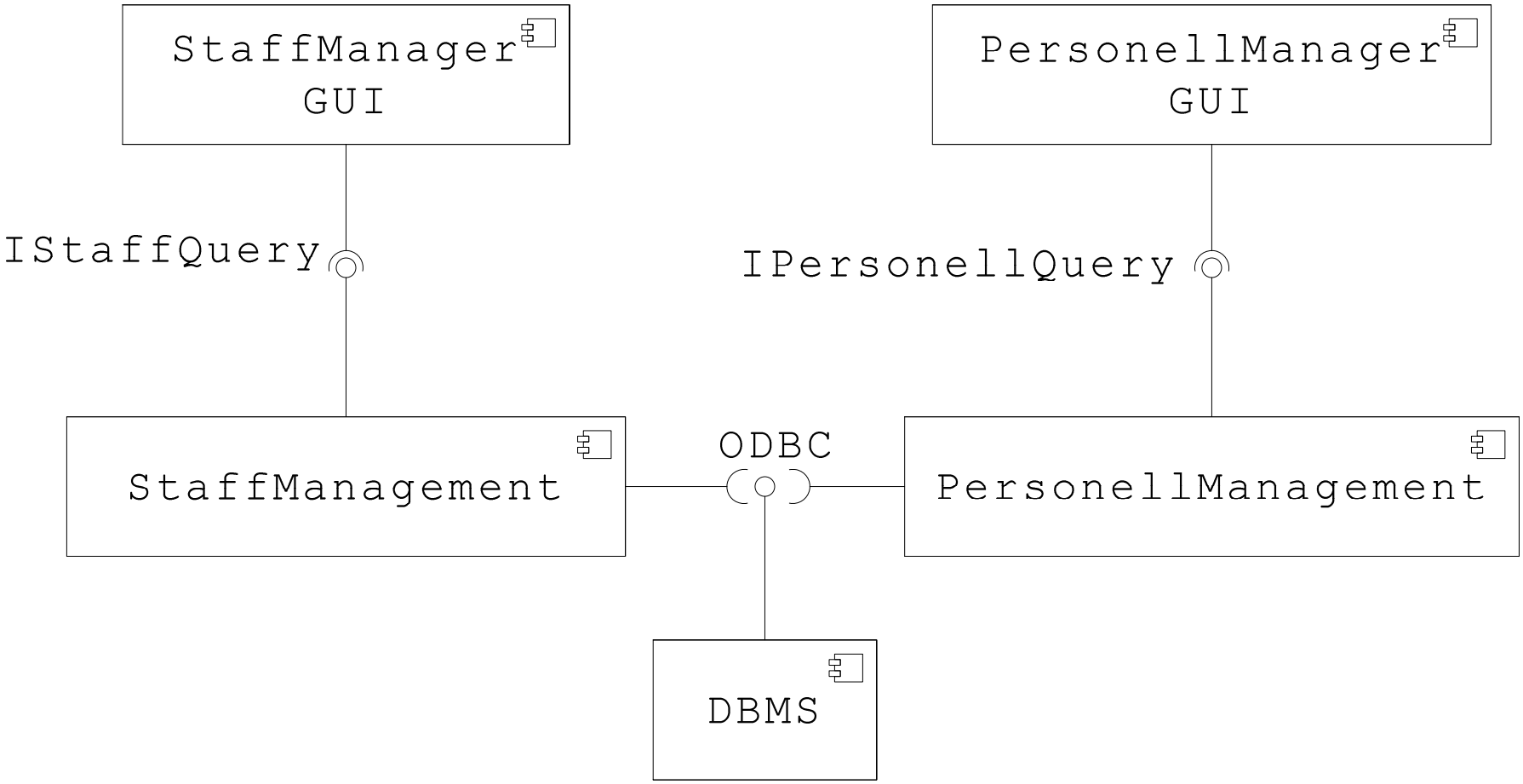
Трехуровневая архитектура





1,2,3) варианты нотации компонента
4) интерфейс (чупа-чупс)

Диаграмма компонентов



Разграничение доступа к данным по категориям пользователей

Диаграмма размещения (i)

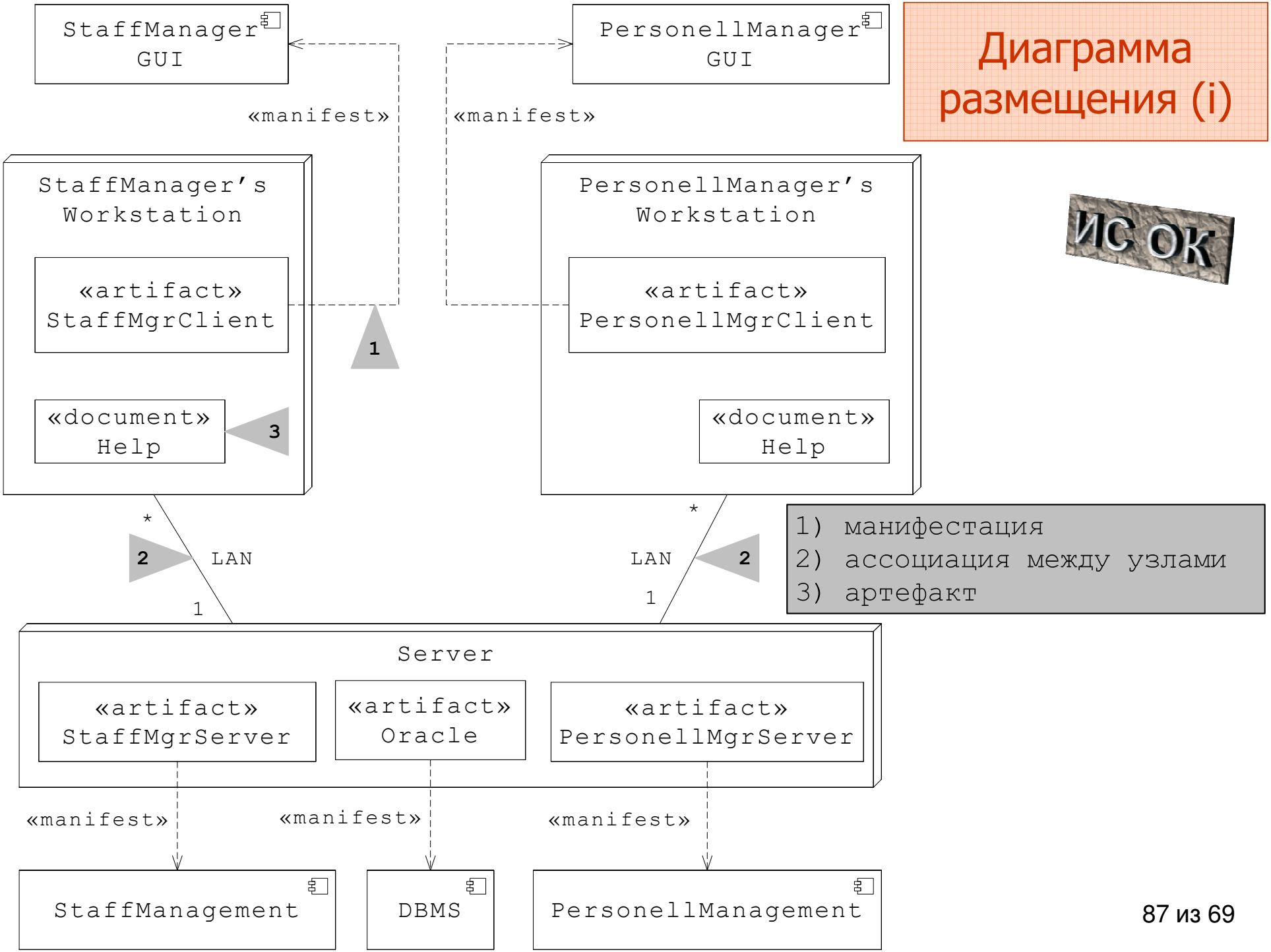
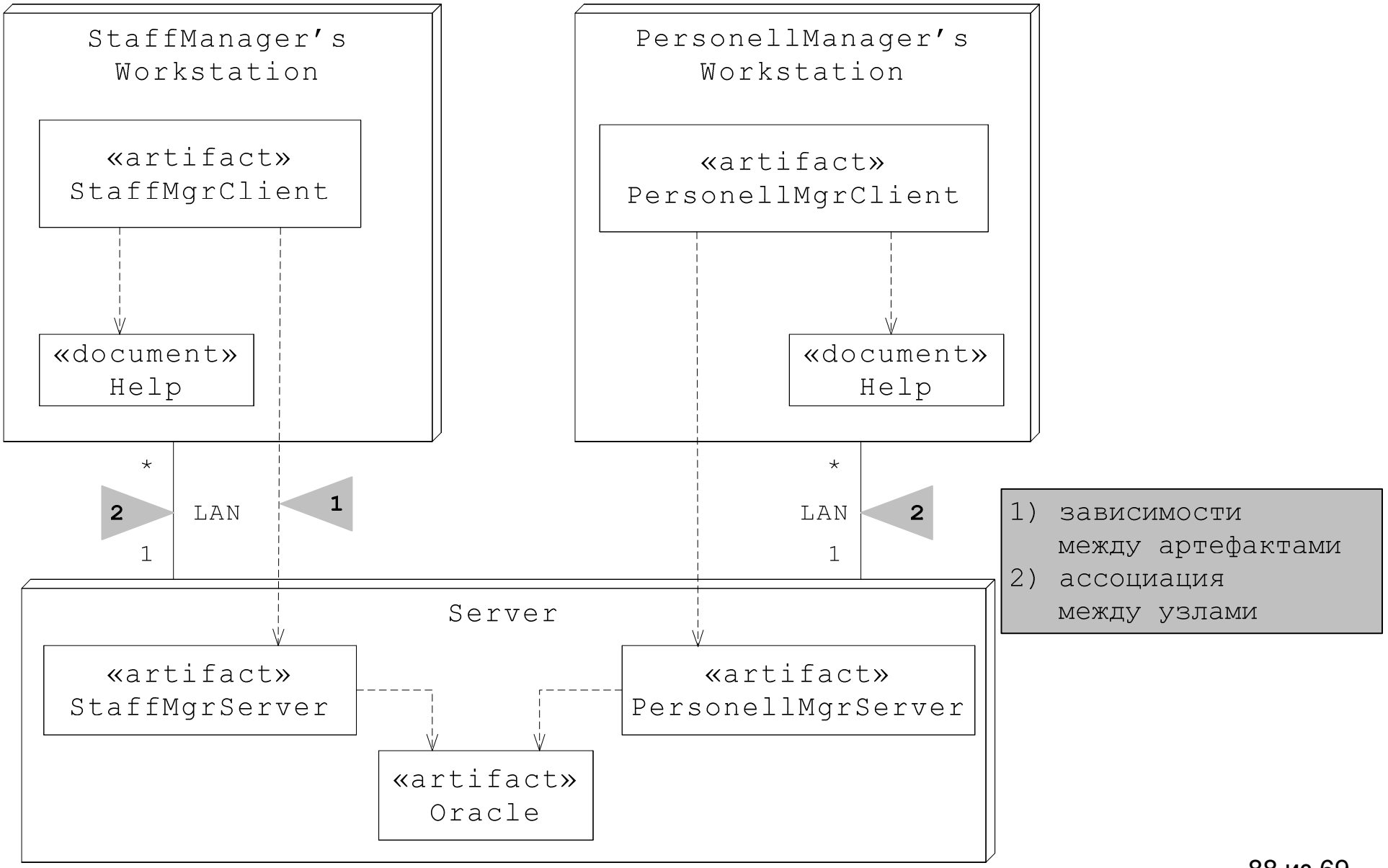


Диаграмма размещения (ii)



5. Моделирование на уровне ролей и экземпляров классификаторов

- Диаграммы внутренней структуры
- Кооперации
- Диаграммы объектов

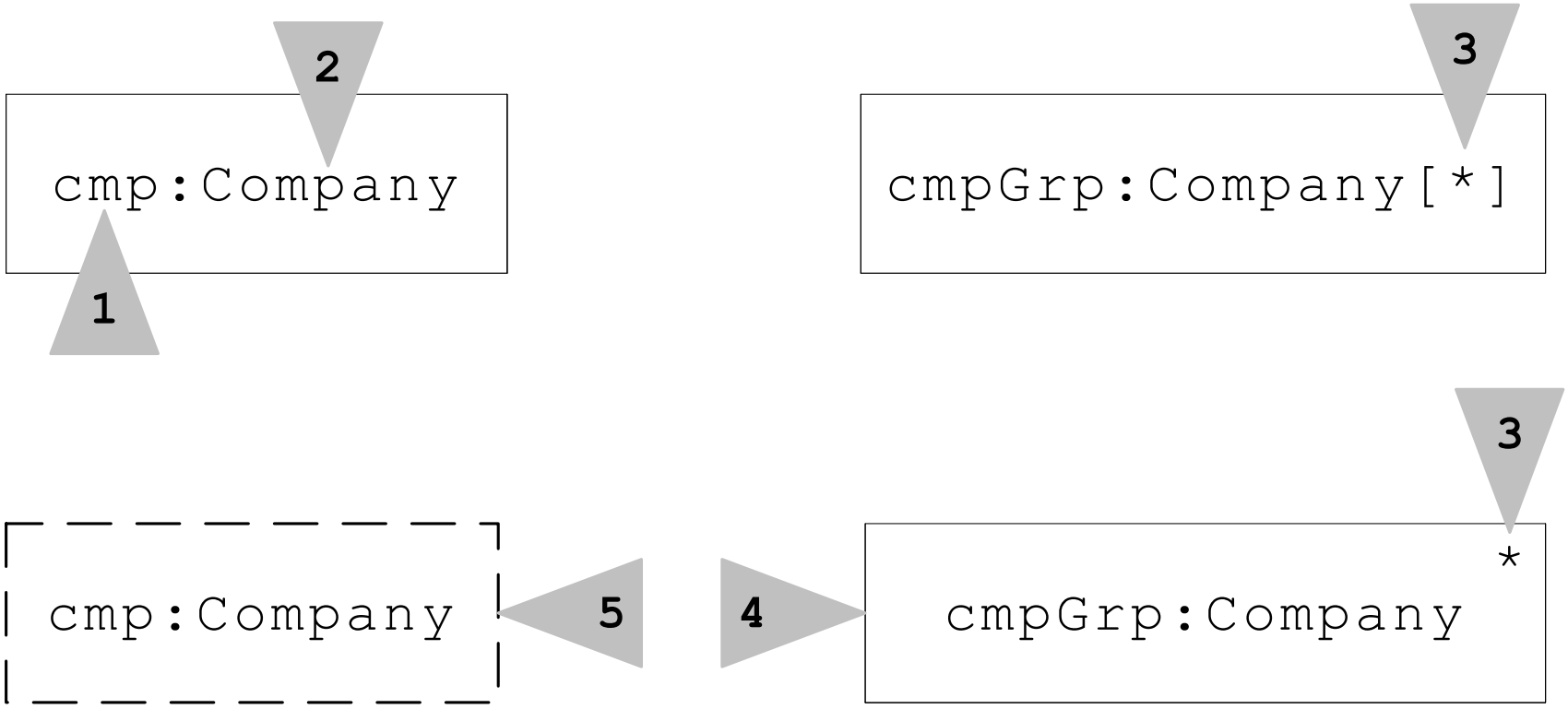
Диаграмма внутренней структуры

Диаграмма внутренней структуры — это структурная диаграмма, которая демонстрирует внутреннюю структуру классификатора и взаимодействие ее элементов (частей)

Сущности:

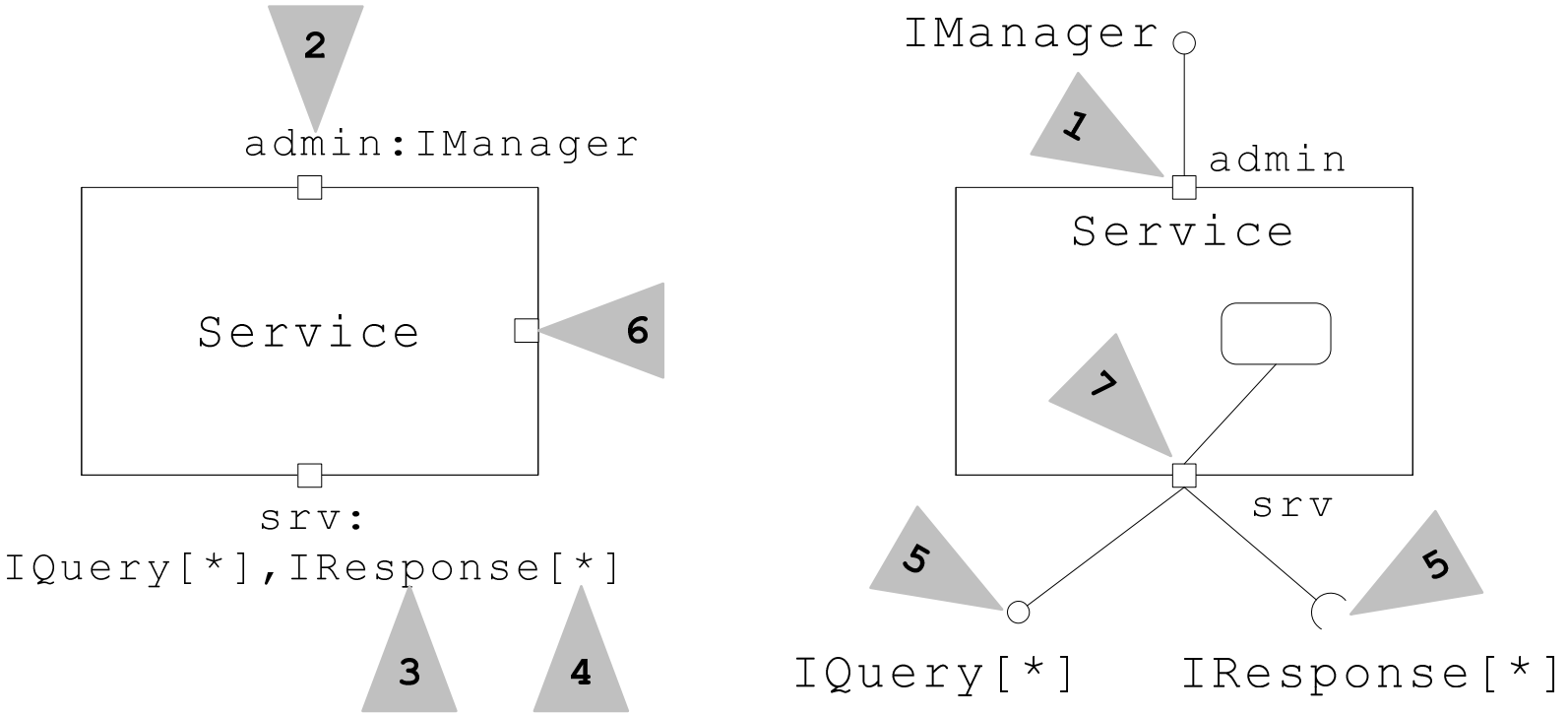
- структурный классификатор
- часть
- порт
- соединитель

Часть — описывает роль, которую ее экземпляр играет внутри экземпляра структурного классификатора



1) имя части
2) тип части
3) кратность части
4) композиция
5) не композиция

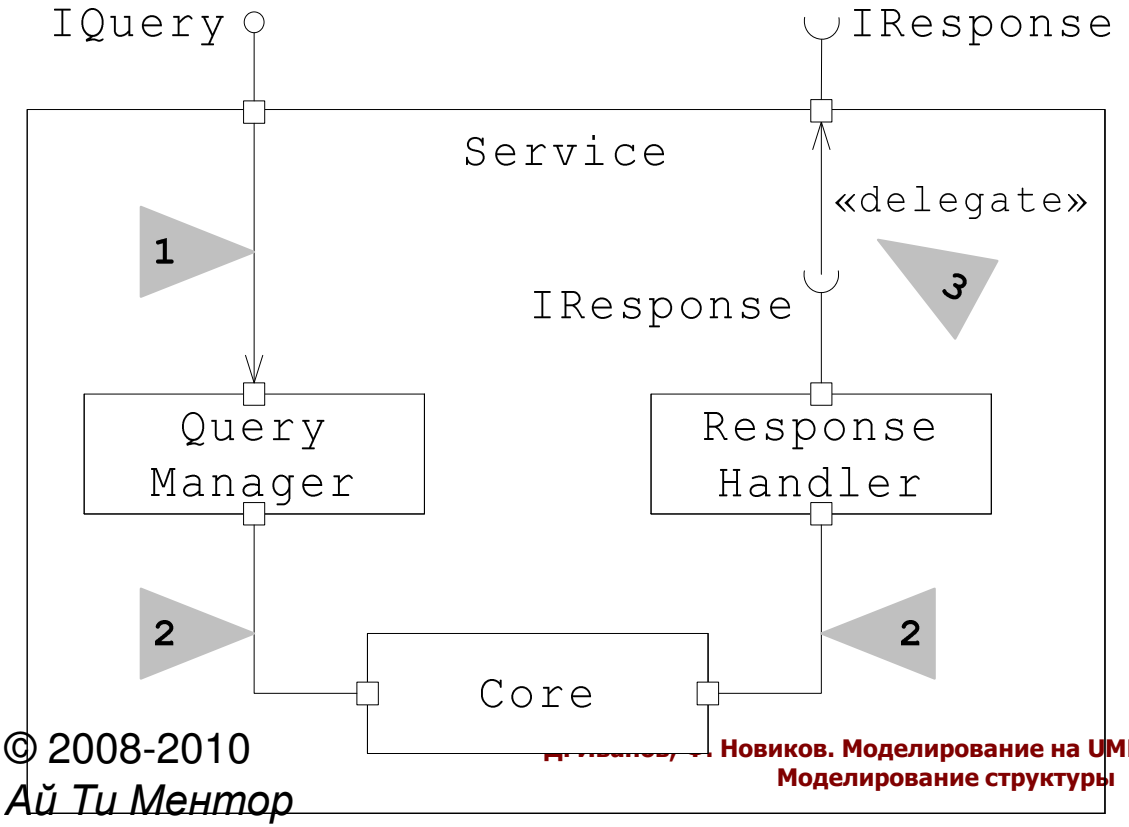
Порт — индивидуальная точка взаимодействия (interaction point) структурного классификатора с внешними сущностями



1) порт
2) имя порта
3) тип порта
4) кратность порта
5) интерфейсы порта (тип)
6) скрытый порт
7) порт поведения

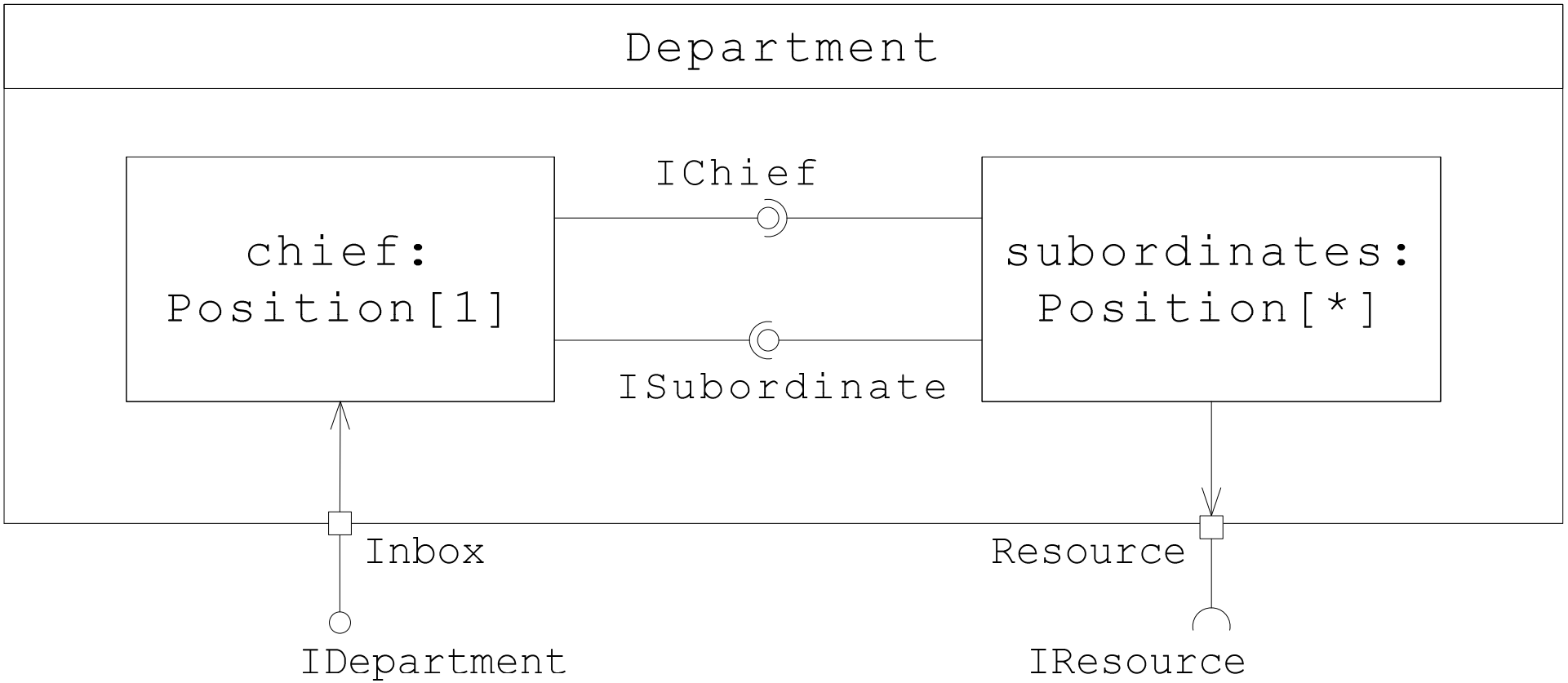
Соединитель служит для соединения частей структурного классификатора между собой

- Делегирующий — соединяет порт структурированного классификатора с его внутренней частью
- Сборочный (обычный) — соединяет две части структурированного классификатора



1,3) делегирующий соединитель
2) сборочный соединитель

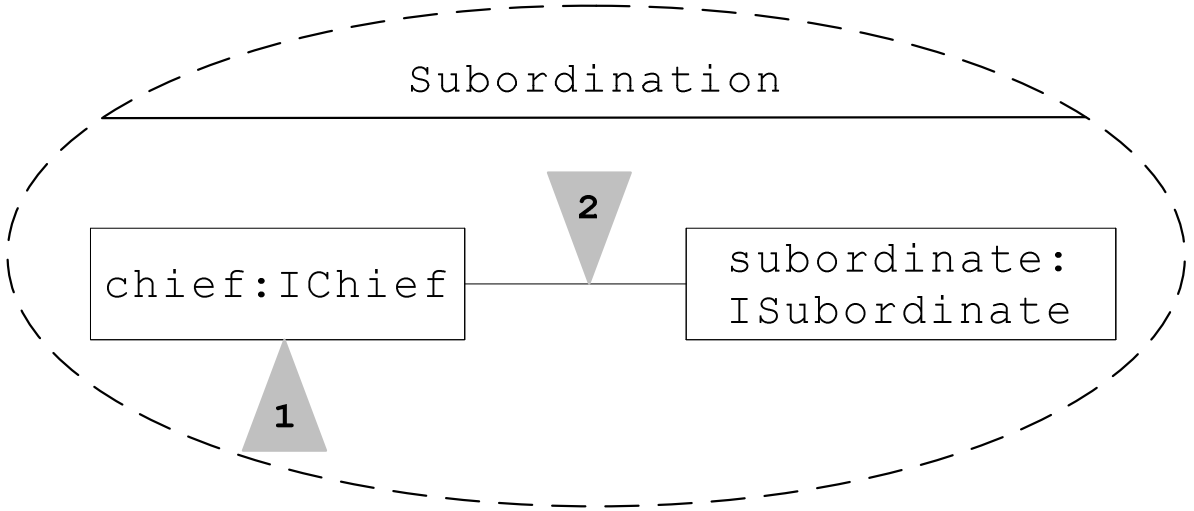
Диаграмма внутренней структуры классификатора



Кооперация

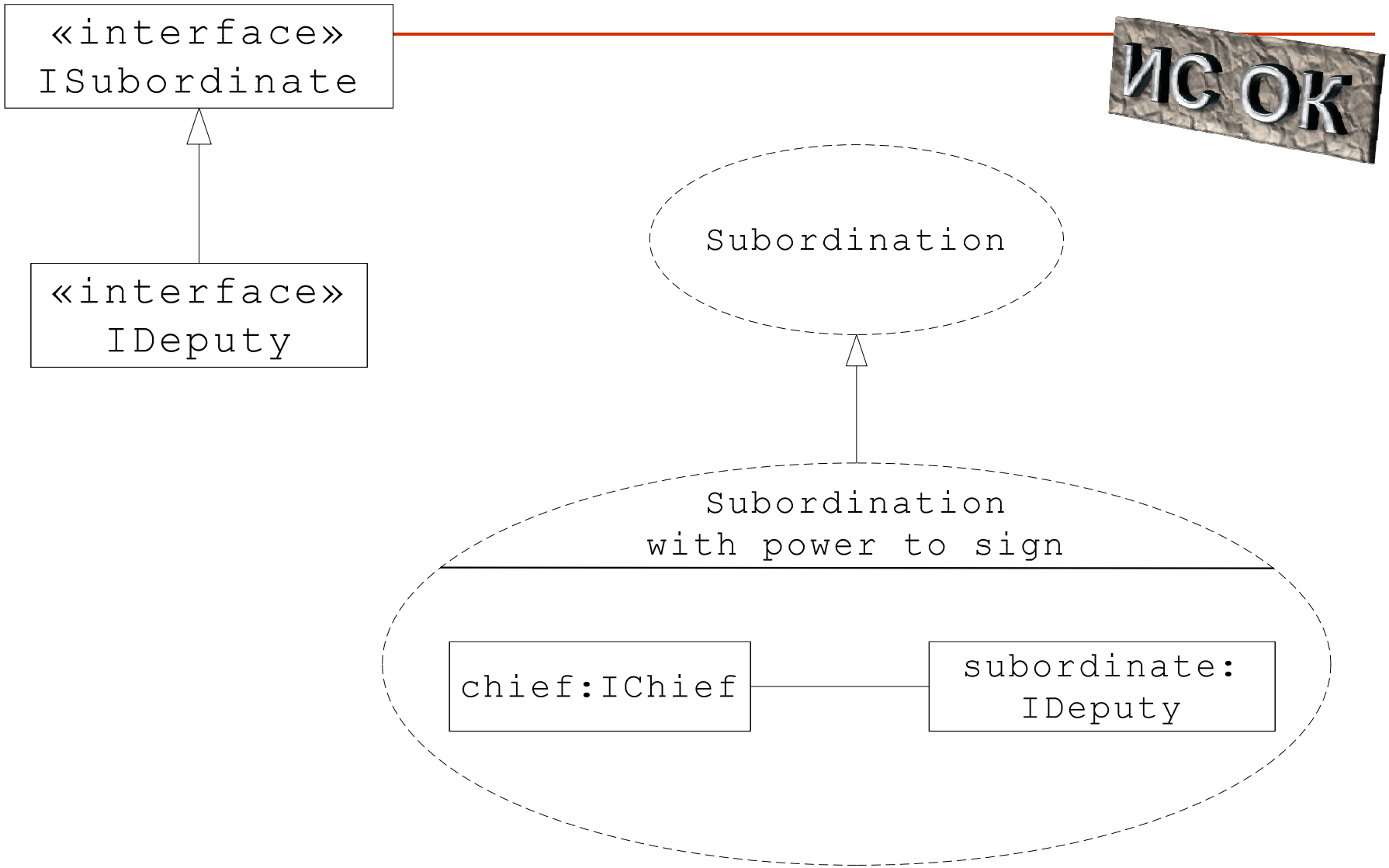
- Кооперация не владеет своими частями — они связаны участием в кооперации
- Кооперация имеет логическое имя
- Применение кооперацией – описание реализации вариантов использования и совместного поведения группы классов

Кооперация «Начальник-Подчиненные» (i)

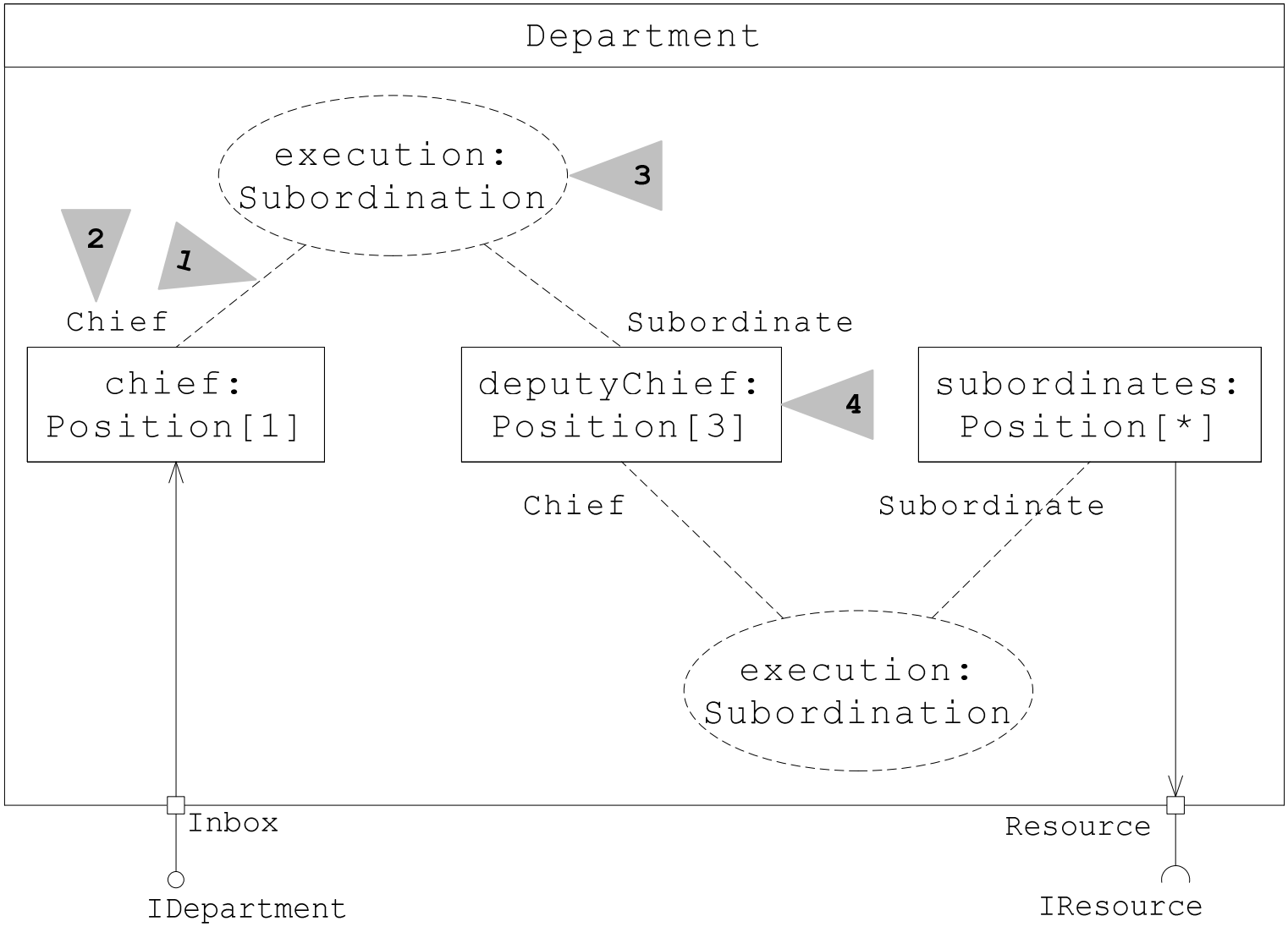


1) роль
2) соединитель

Кооперация «Начальник-Подчиненные» (ii)



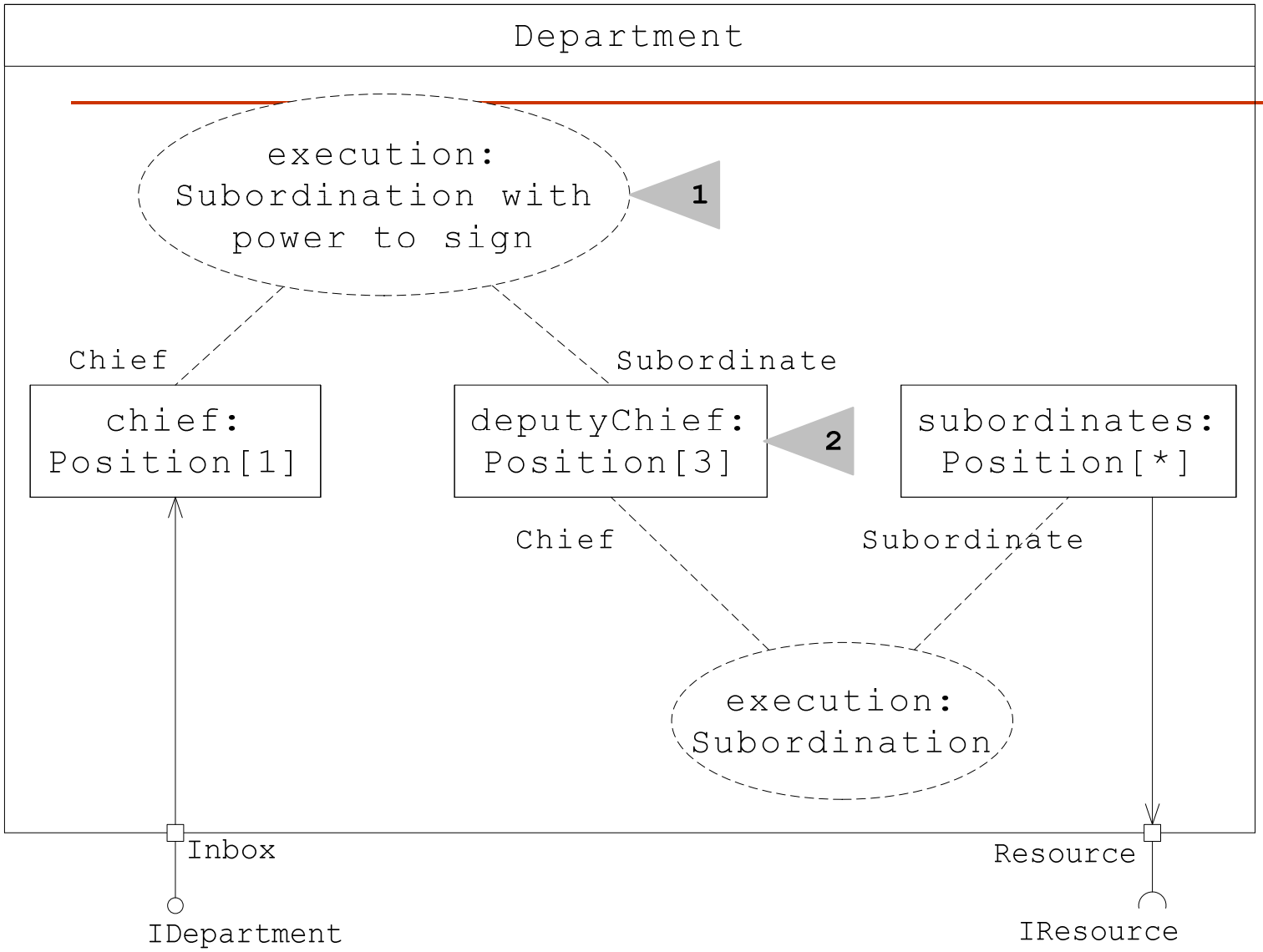
Использование кооперации (i)



ИС ОК

1) связь роль-классификатор
2) имя роли
3) экземпляр кооперации
4) классификатор, играющий роль

Использование кооперации (ii)



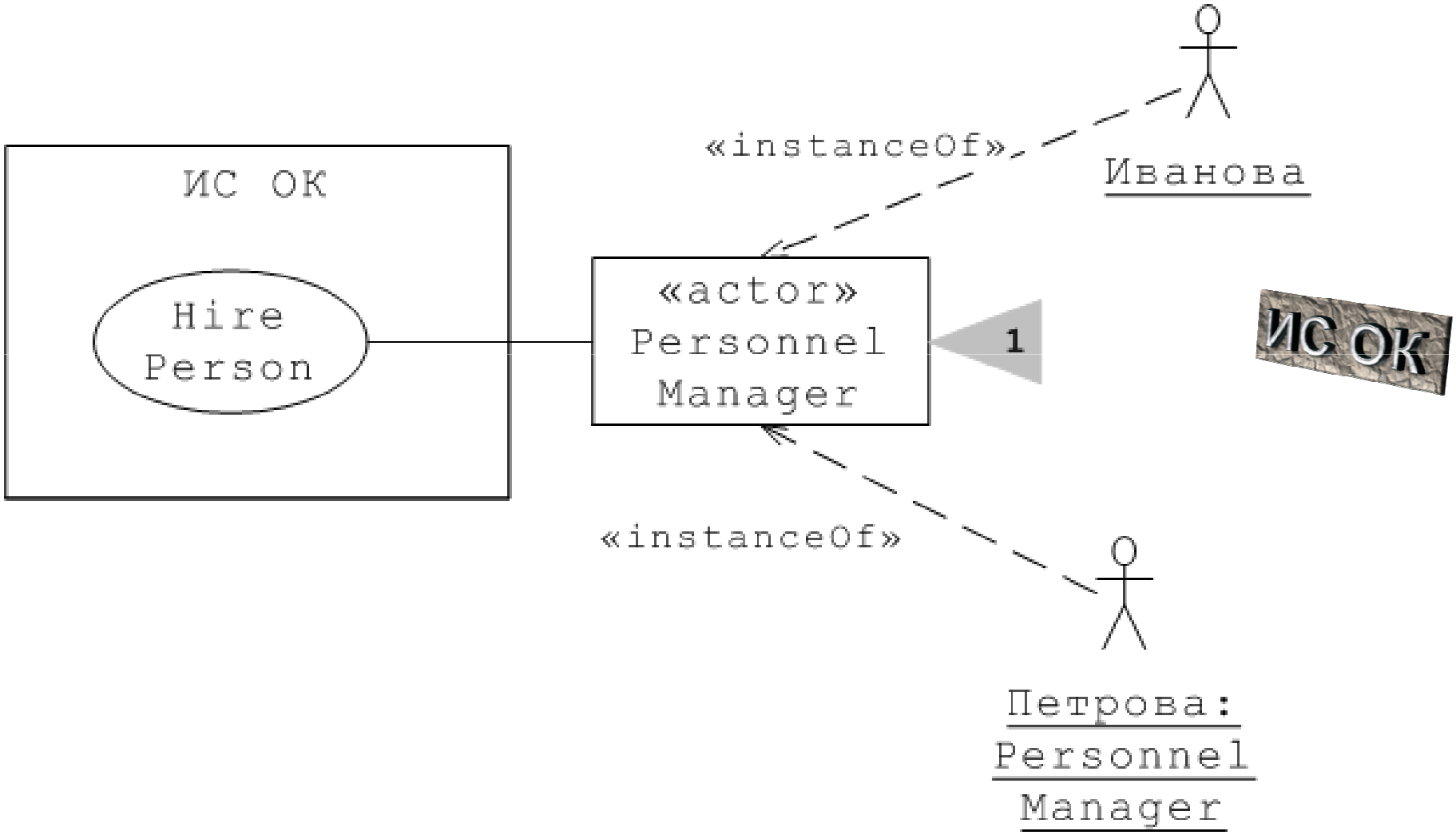
ИС ОК

1) экземпляр кооперации
2) классификатор, играющий роль

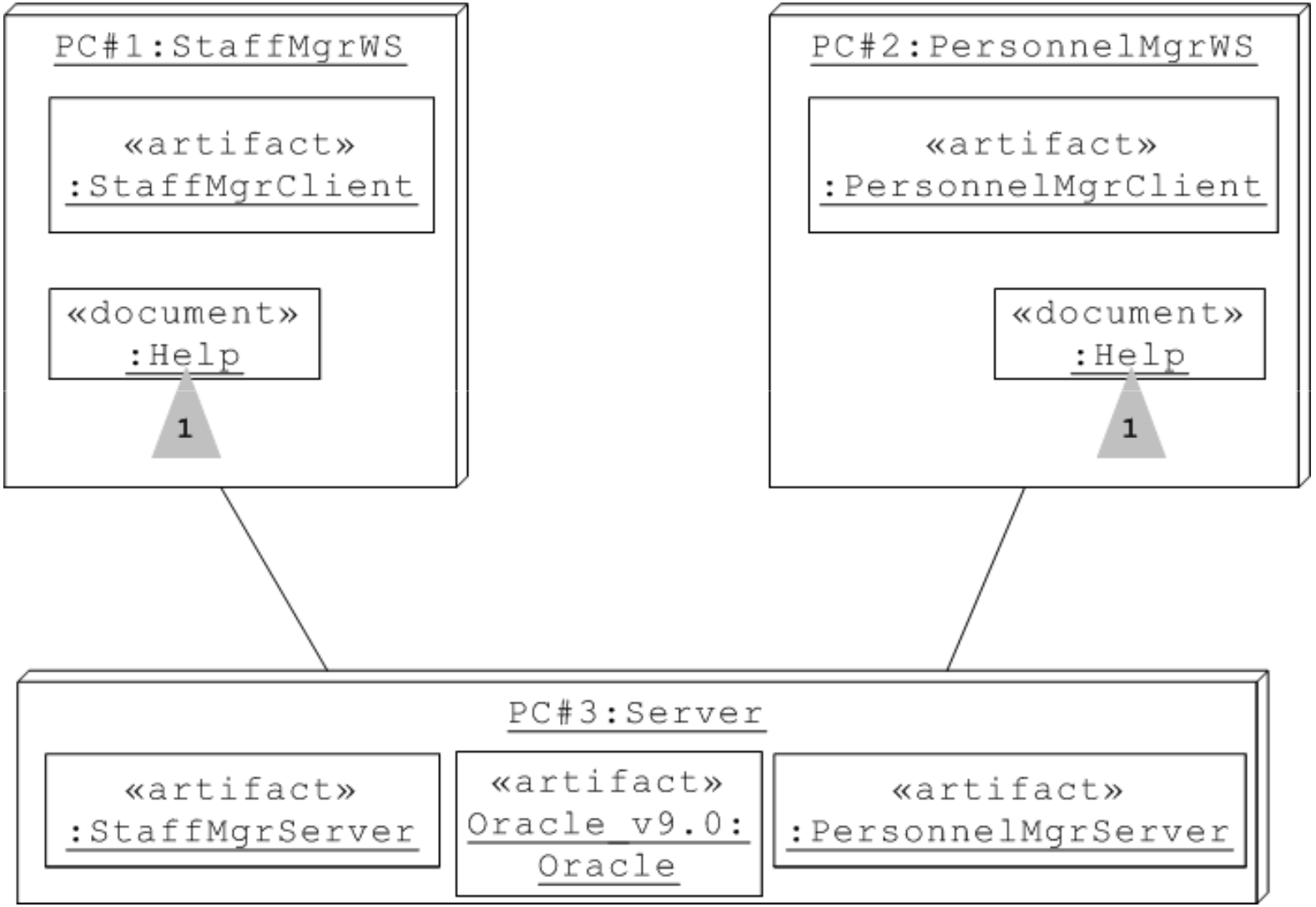
Экземпляры классификаторов

- Могут применяться на всех типах диаграмм, где есть классификаторы → везде
- Экземпляры действующих лиц
- Экземпляры узлов
- Экземпляры классов

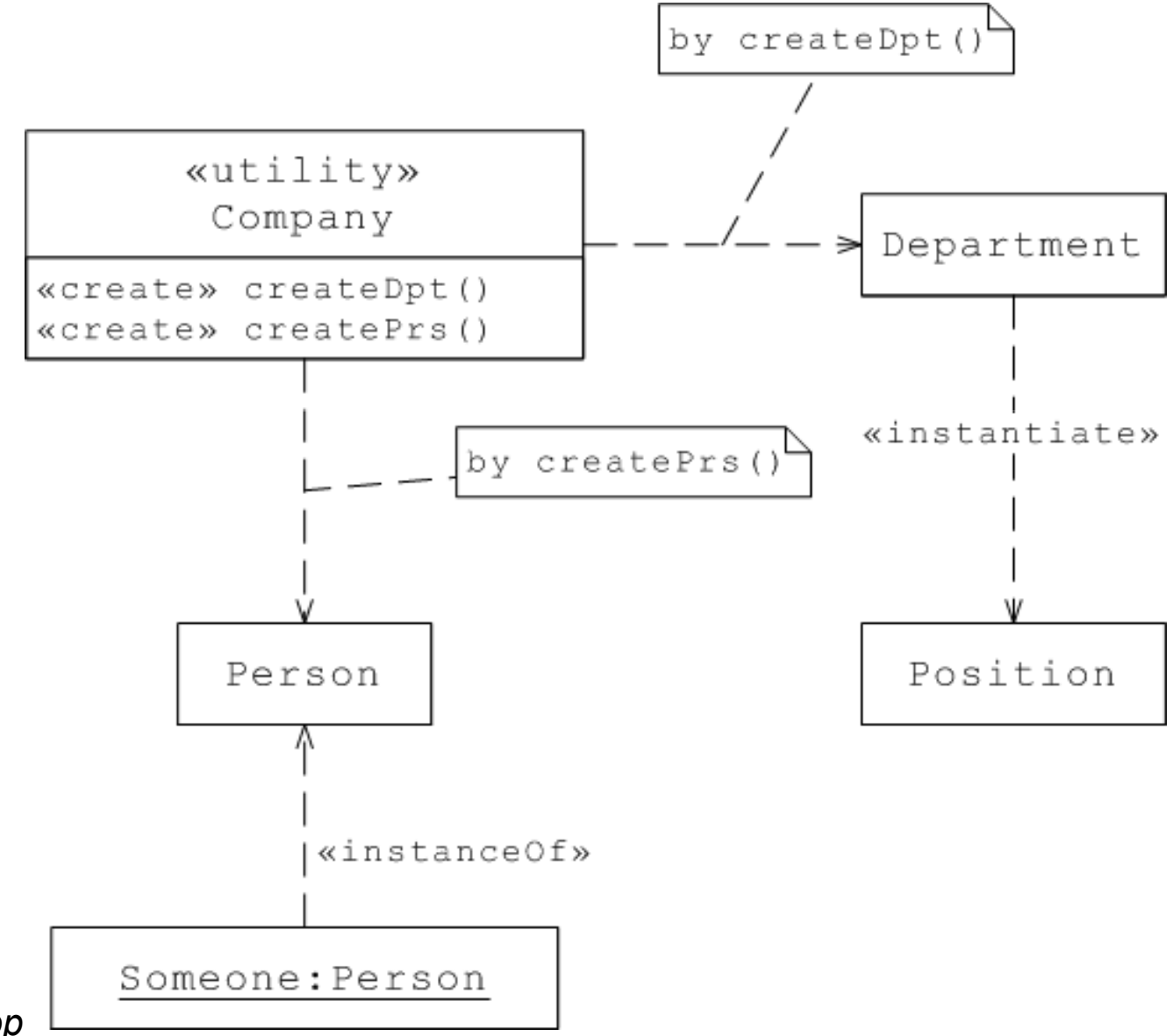
Экземпляры действующих лиц



Экземпляры узлов

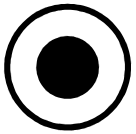


Экземпляры классов



6. Выводы (i)

- Структура сложной системы описывается дескрипторами
- Диаграммы классов моделируют структуру объектов и связей между ними
- Классы выбираются на основе анализа предметной области, взаимного согласования элементов и теоретических соображений
- Отношения между классами позволяют моделировать структуры любой сложности



6. Выводы (ii)

- Диаграммы компонентов моделируют структуру артефактов и взаимосвязей между ними
- Диаграммы размещения моделируют структуру вычислительных ресурсов и размещенных на них артефактов.
- Диаграммы внутренней структуры показывают контекст взаимодействия частей сложных классификаторов, причем части, в свою очередь, могут иметь внутреннюю структуру.
- Кооперация – это способ показать контекст взаимодействия нескольких классификаторов