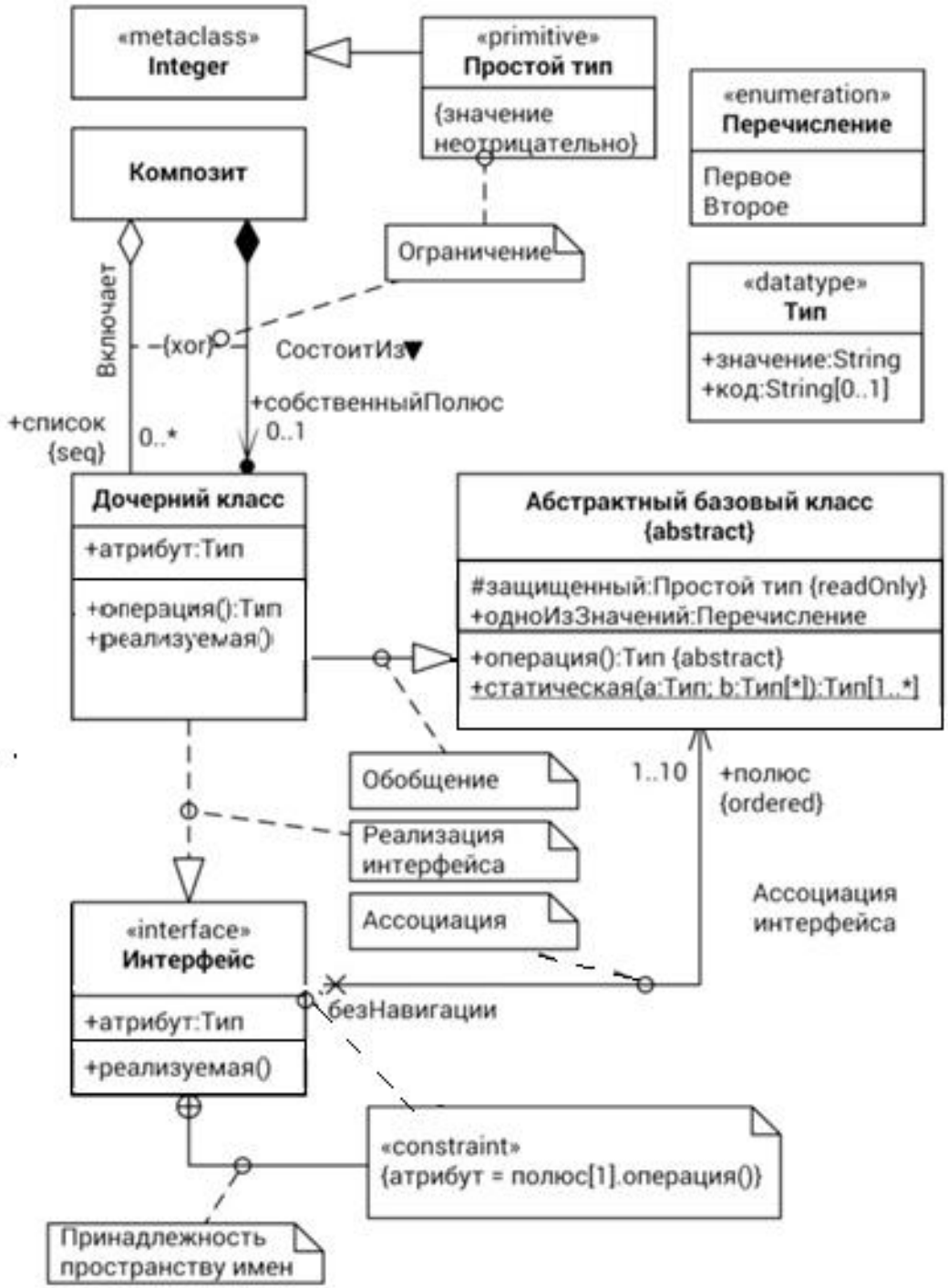
Диаграммы классов

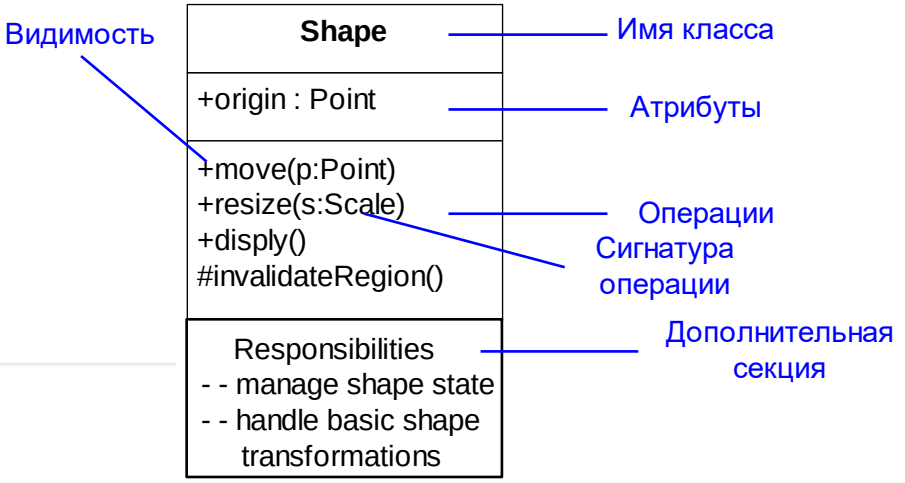
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

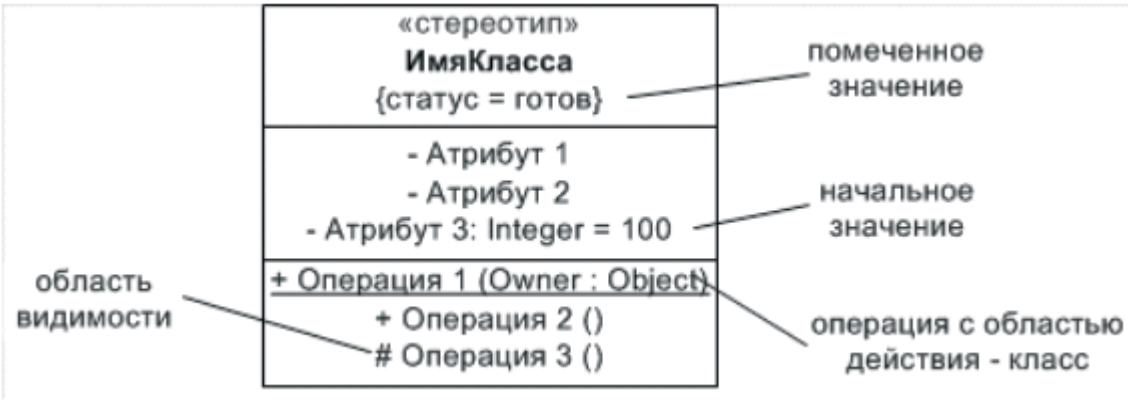
Словарь предметной области

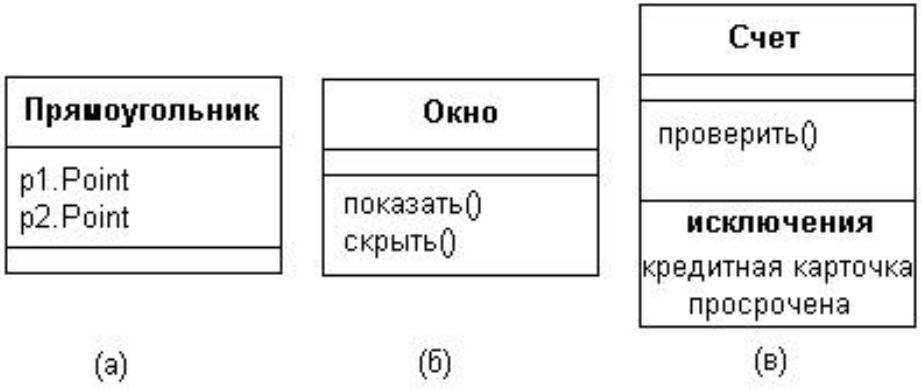
Пример информационной системы отдела кадров

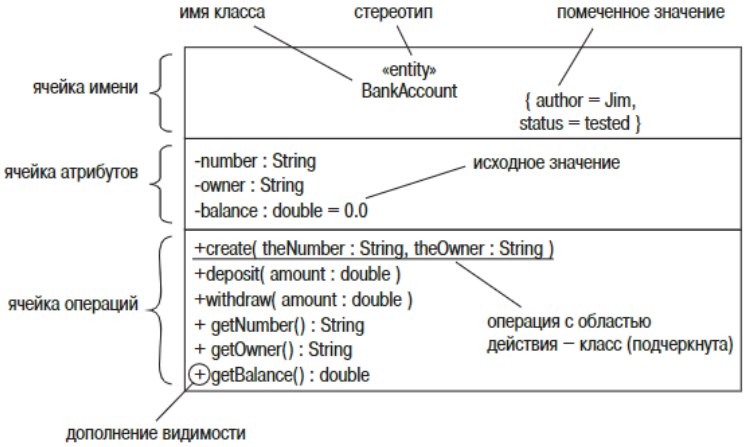
- прием;
- перевод;
- увольнение;
- сотрудник;
- создание;
- ликвидация;
- подразделение;
- вакансия;
- сокращение;
- должность.


«metaclass»
Integer
«primitive»
Простой тип
{значение неотрицательно}
«enumeration»
Перечисление
Первое
Второе
Композит
Ограничение
«datatype»
Тип
+значение:String
+код:String[0..1]
Включает
{xor}
СостоитИз
+собственныйПолюс
0..1
+список
{seq}
0..*
Дочерний класс
+атрибут:Тип
+операция():Тип
+реализуемая()
Абстрактный базовый класс
{abstract}
#защищенный:Простой тип {readOnly}
+одноИзЗначений:Перечисление
+операция():Тип {abstract}
+статическая(a:Тип; b:Тип[*]):Тип[1..*]
Обобщение
1..10
+полюс
{ordered}
Реализация интерфейса
Ассоциация
Ассоциация интерфейса
«interface»
Интерфейс
+атрибут:Тип
+реализуемая()
безНавигации
«constraint»
{атрибут = полюс[1].операция()}
Принадлежность пространству имен









<Имя_пакета>::<Имя_класса>

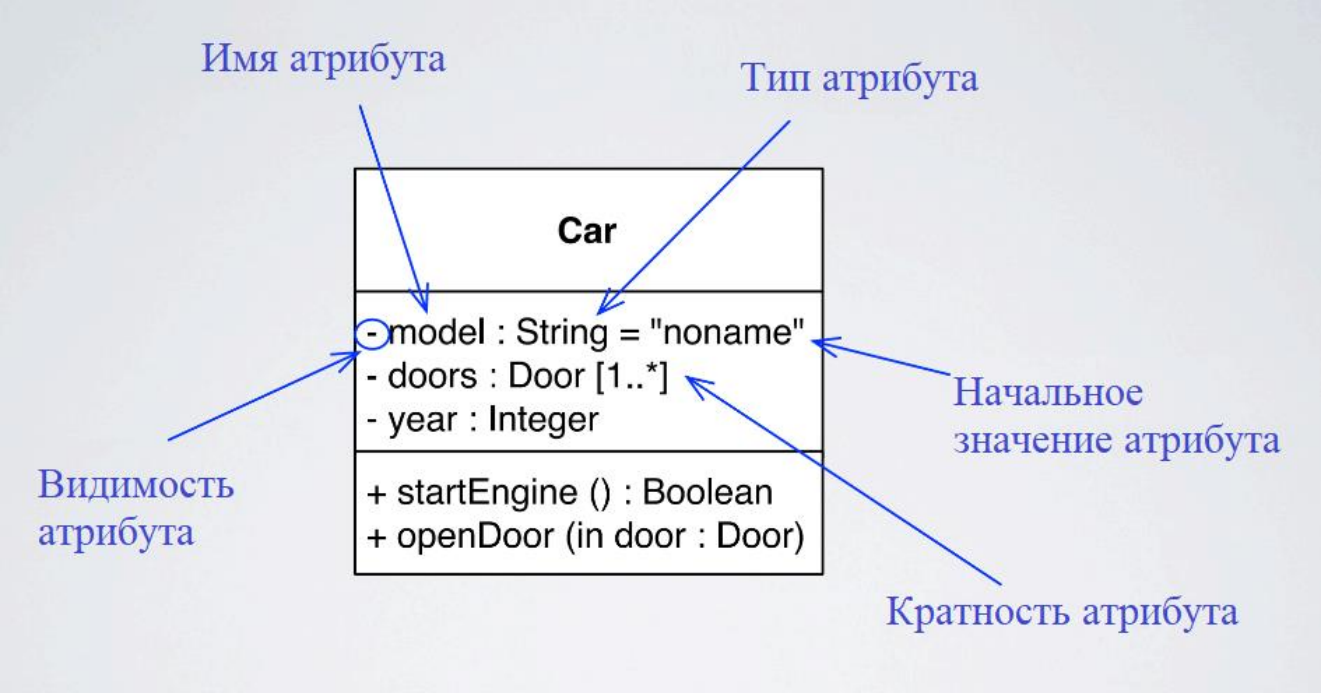
Банк::Счет

Табл. **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

Атрибуты

видимость имя кратность: тип = начальное значение {свойство}



+ имяСотрудника : String

форма : Многоугольник = «прямоугольник»

Атрибуты

Табл. **Выражения кратности**

Выражение кратности	Множество может иметь
`0..*` или `*`	Произвольное число элементов
`1..*`	Один или более элементов
`0..1`	Не более одного элемента
`1..10`	От одного до десяти элементов
`1..3, 5, 7..10`	Один, два, три, пять, семь, восемь, девять или десять элементов
`5..3`	Некорректная кратность. Нижняя граница больше верхней
`-1..3`	Некорректная кратность. Отрицательные числа недопустимы

Строка-свойство

readOnly – атрибут является только для чтения
union – атрибут является производным объединением его подмножеств
subsets <имя атрибута> – атрибут является собственным подмножеством атрибута с именем <имя атрибута>
redefines <имя атрибута> – атрибут переопределяет некоторый наследуемый атрибут с именем <имя атрибута>
ordered – значения атрибута являются упорядоченными. Этот порядок означает, что существует отображение из множества положительных целых чисел в элементы этой коллекции значений. Если атрибут не является многозначным, то это значение не имеет семантического эффекта. При отсутствии этого модификатора атрибут специфицируется как неупорядоченный.
unique – значения многозначного атрибута не могут иметь дубликатов, т.е. повторяться. Предполагается, что кратность соответствующего атрибута должна быть больше 1. Если атрибут не является многозначным, то значение unique не имеет семантического эффекта.

+ имяСотрудника [1..2] : String {readOnly}

~ датаРождения : Data {readOnly}

/возрастСотрудника : Integer

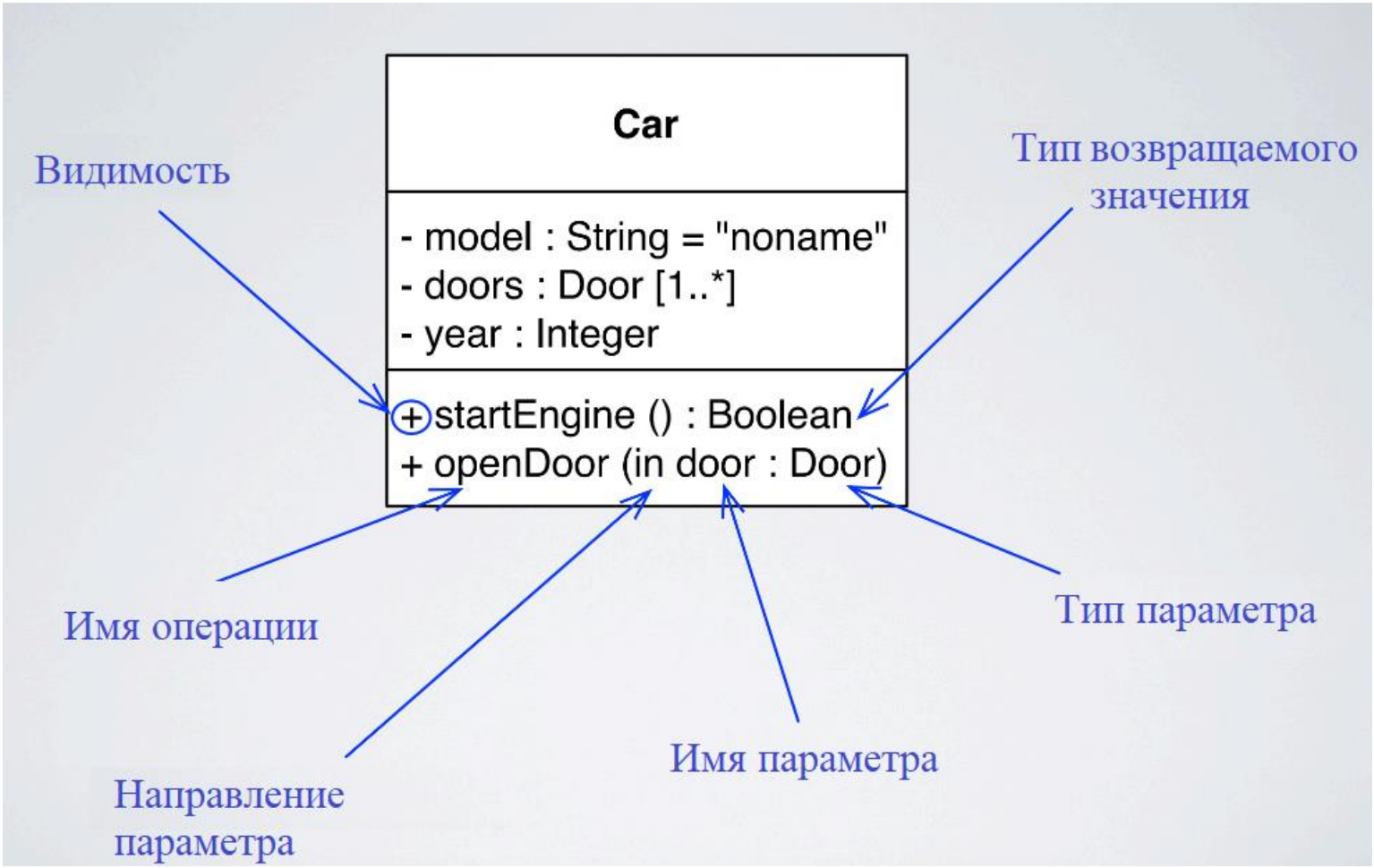
+ номерТелефона : Integer [1..*] {unique}

– заработнаяПлата : Currency = 500.00

Операции класса

видимость имя (параметры): тип {свойство}

направление ПАРАМЕТР : тип = значение



Операции класса

видимость имя (параметры): тип {свойство}

направление ПАРАМЕТР : тип = значение

Ключевое слово	Назначение параметра
in	Входной параметр – аргумент должен быть значением, которое используется в операции, но не изменяется
out	Выходной параметр – аргумент должен быть хранилищем, в которое операция помещает значение
inout	Входной и выходной параметр – аргумент должен быть хранилищем, содержащим значение. Операция использует переданное значение аргумента и помещает в хранилище результат
return	Значение, возвращаемое операцией. Такое значение направления передачи устанавливается автоматически для возвращаемого значения

Примеры записи операций

```
move()
```

```
+move(in from, in to)
```

```
+move(in from:Department,
in to:Department)
```

```
+getName():String{isQuery}
```

Табл. **Значения свойства** `concurrency`

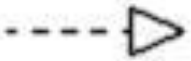
Значение	Описание
`{sequential}`	Операция не допускает параллельного вызова (не является повторно-входимой). Если параллельный вызов происходит, то дальнейшее поведение системы не определено.
`{quarded}`	Параллельные вызовы допускаются, но только один из них выполняется – остальные блокируются, и их выполнение задерживается до тех пор, пока не завершится выполнение данного вызова.
`{concurrent}`	Операция допускает произвольное число параллельных вызовов и гарантирует правильность своего выполнения. Такие операции называются повторно-входимыми (reenterable).

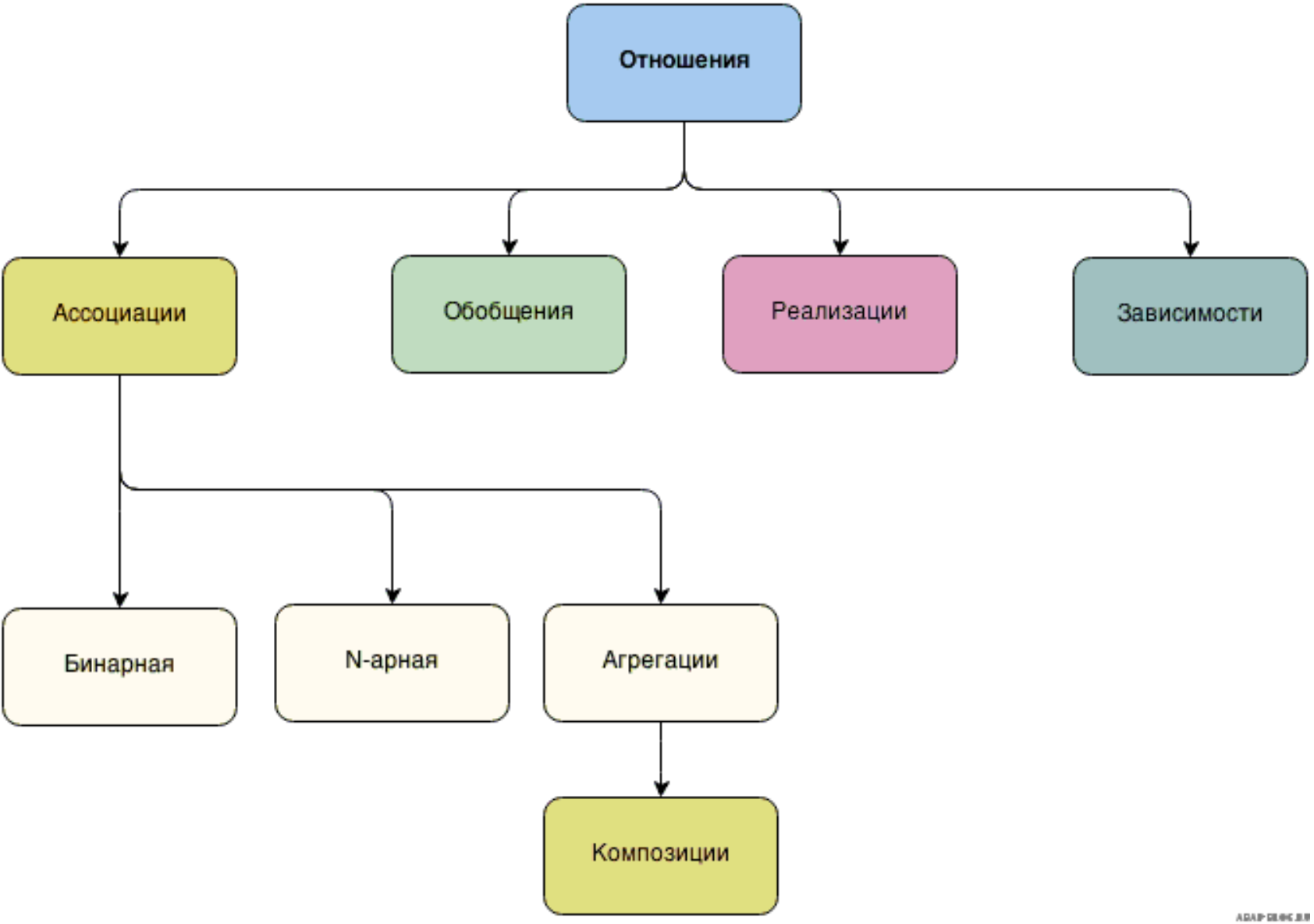
{concurrency = имя}

{isQuery}

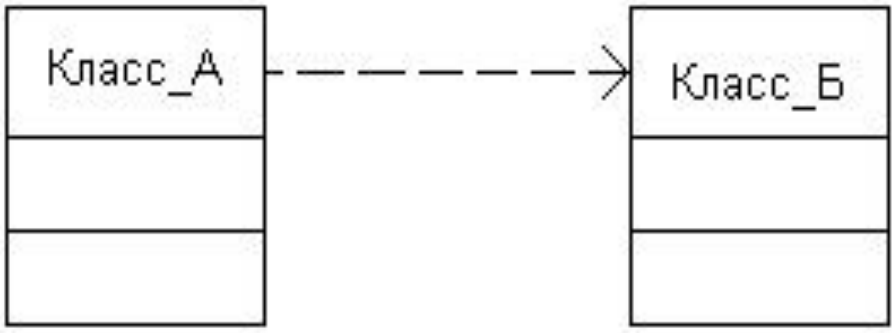
{leaf}

Отношения на диаграмме классов

Relationship	*Function*	*Notation*
association	Представление произвольного отношения между экземплярами классов	
generalization	Отношение типа "Общее-Частное", обладающая свойством наследования свойств	
aggregation	Отношение типа "Часть-Целое"	
composition	Более сильная форма отношения типа "Часть-Целое"	
realization	Отношение между спецификацией и ее выполнением	
dependency	Направленное отношение между двумя элементами модели с открытой семаникой	

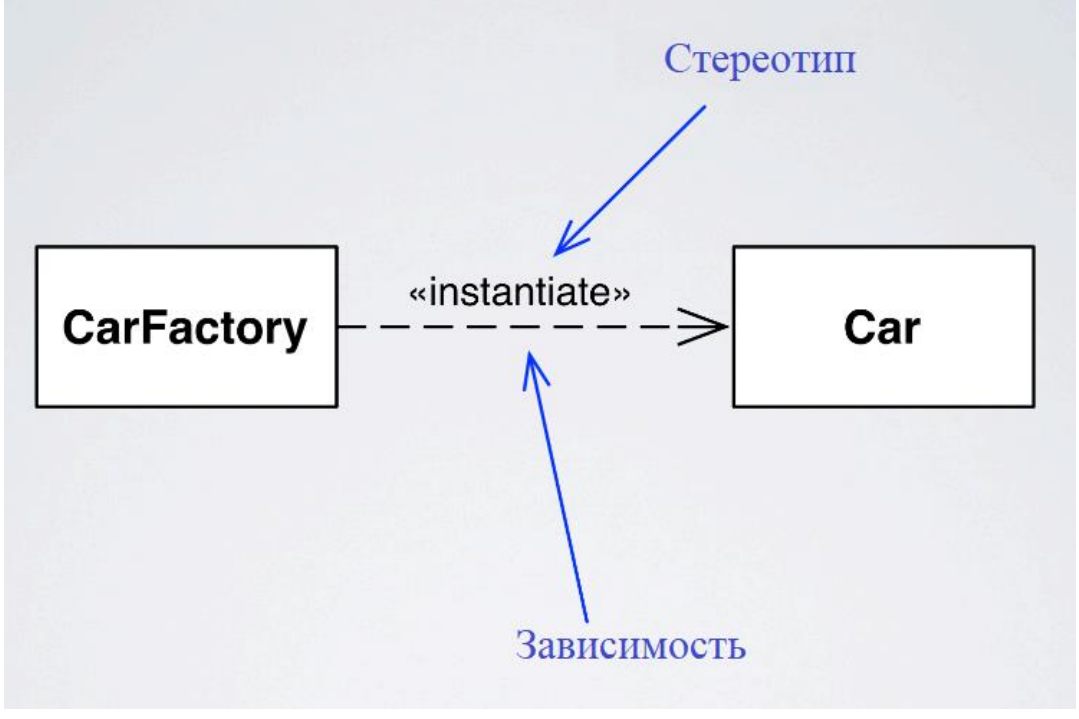

Отношения
Ассоциации
Обобщения
Реализации
Зависимости
Бинарная
N-арная
Агрегации
Композиции

Отношение зависимости



Сервер – объект предметной области

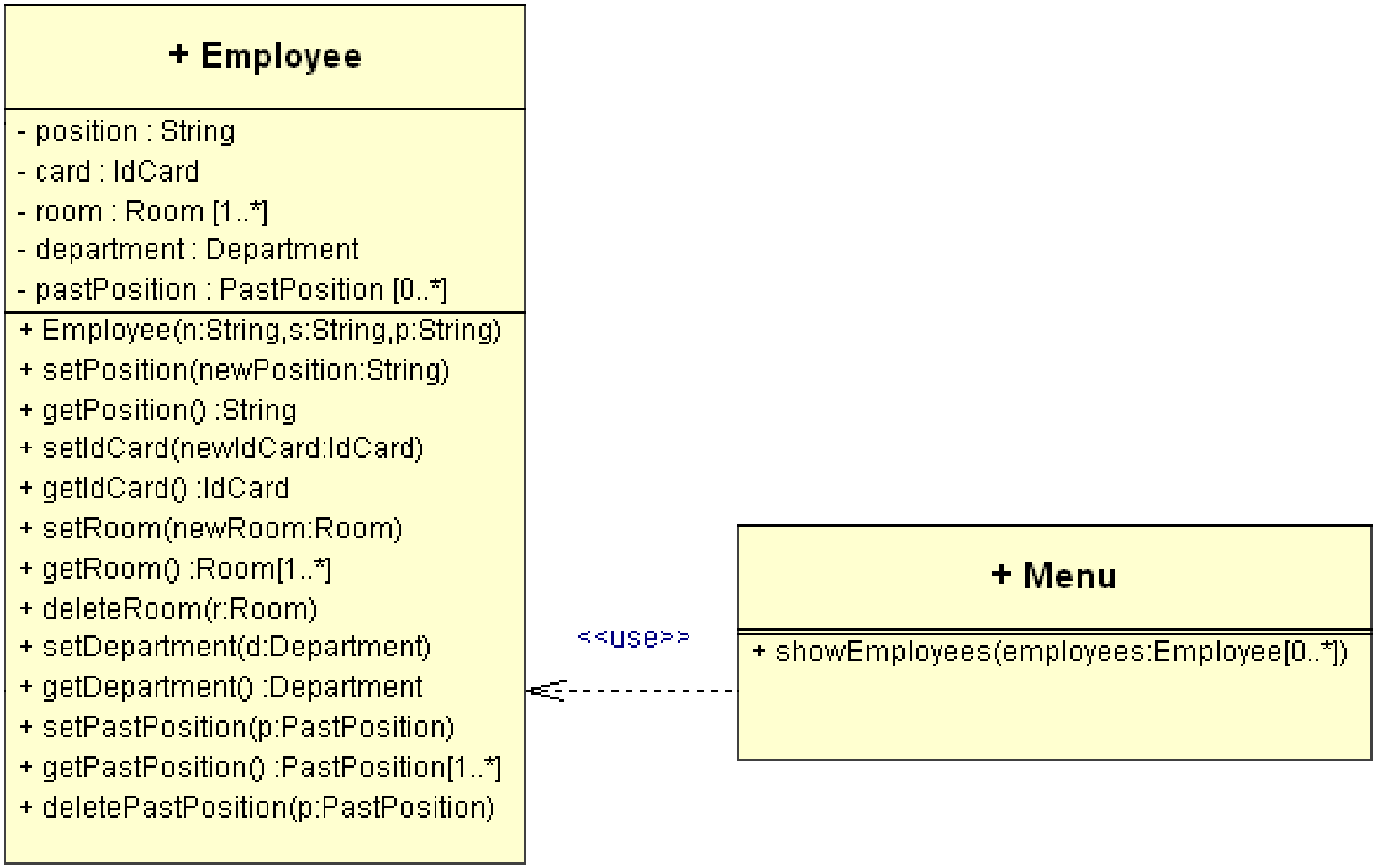
Клиент – Пользовательский интерфейс или класс представления



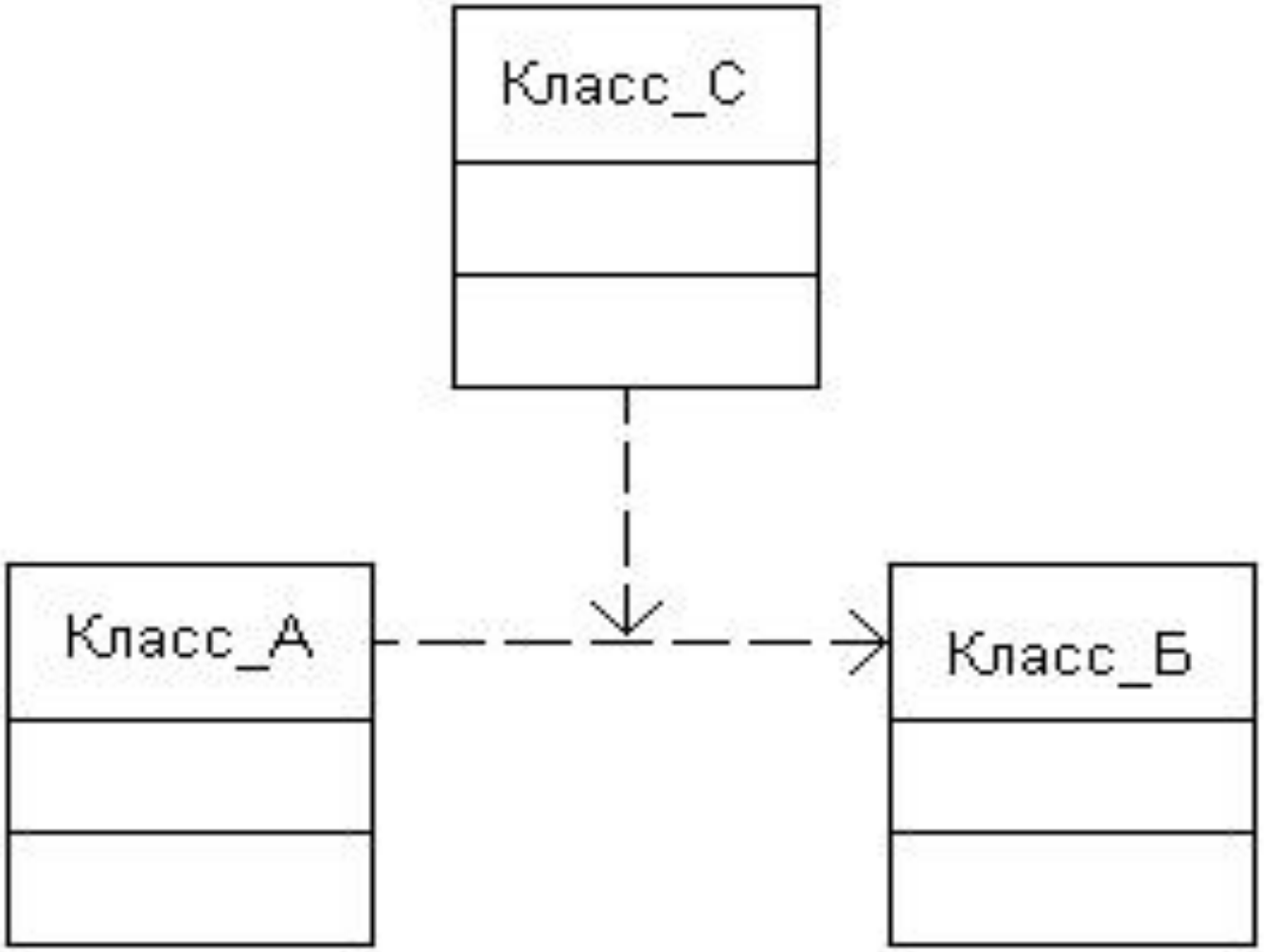
Стереотипы - Отношение зависимости

Стереотип	Описание
«bind»	Подстановка параметров в шаблон. Независимой сущностью является шаблон (класс с параметрами), а зависимой – класс, который получается из шаблона заданием аргументов.
«call»	Указывает зависимость между двумя операциями: операция зависимого класса вызывает операцию независимого класса.
«derive»	Буквально означает "может быть вычислен по". Зависимость с данным стереотипом применяется не только к классам, но и к другим элементам модели: атрибутам, ассоциациям и т.д. Суть состоит в том, зависимый элемент может быть восстановлен по информации, содержащейся в независимом элементе. Таким образом, данная зависимость показывает, что зависимый элемент, вообще говоря, излишен и введен в модель из соображений удобства, наглядности и т.д.
«friend»	Назначает специальные права видимости. Зависимый класс имеет доступ к составляющим независимого класса, даже если по общим правилам видимости такие права у него отсутствуют.
«instanceOf»	Указывает, что зависимый объект (или класс) является экземпляром независимого класса (метакласса).
«instantiate»	Указывает, что операции зависимого класса создают экземпляры независимого класса.
«powertype»	Показывает, что экземплярами зависимого класса являются подклассы независимого класса. Таким образом, в данном случае зависимый класс является метаклассом.
«refine»	Указывает, что зависимый класс уточняет (конкретизирует) независимый. Данная зависимость показывает, что связанные классы концептуально совпадают, но находятся на разных уровнях абстракции.
«use»	Зависимость самого общего вида, показывающая, что зависимый класс каким-либо образом использует независимый класс.

Отношение зависимости. Пример

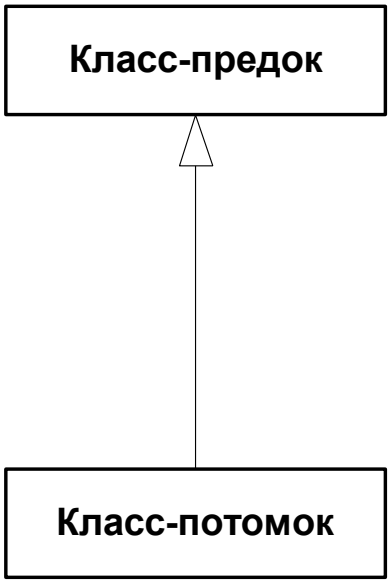


Отношение зависимости

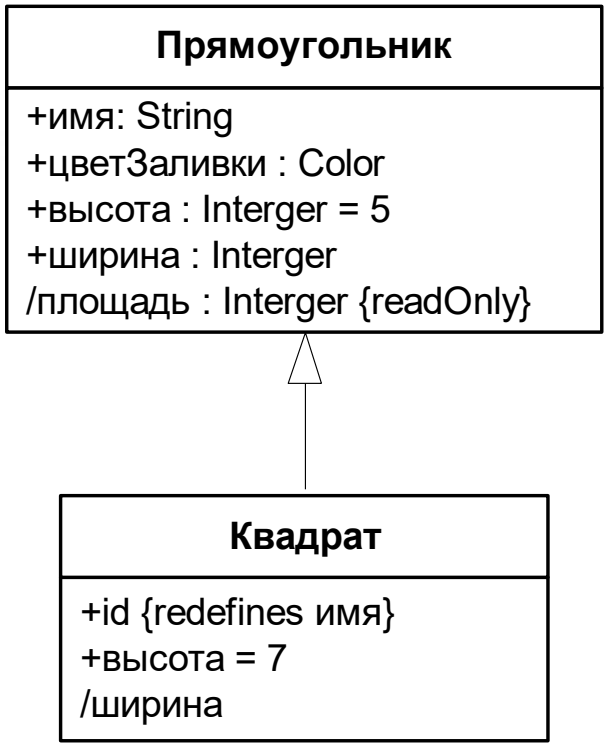


Обобщение (generalization)

- – таксономическое отношение между более общим классификатором (родителем или предком) и более специальным классификатором (дочерним или потомком)

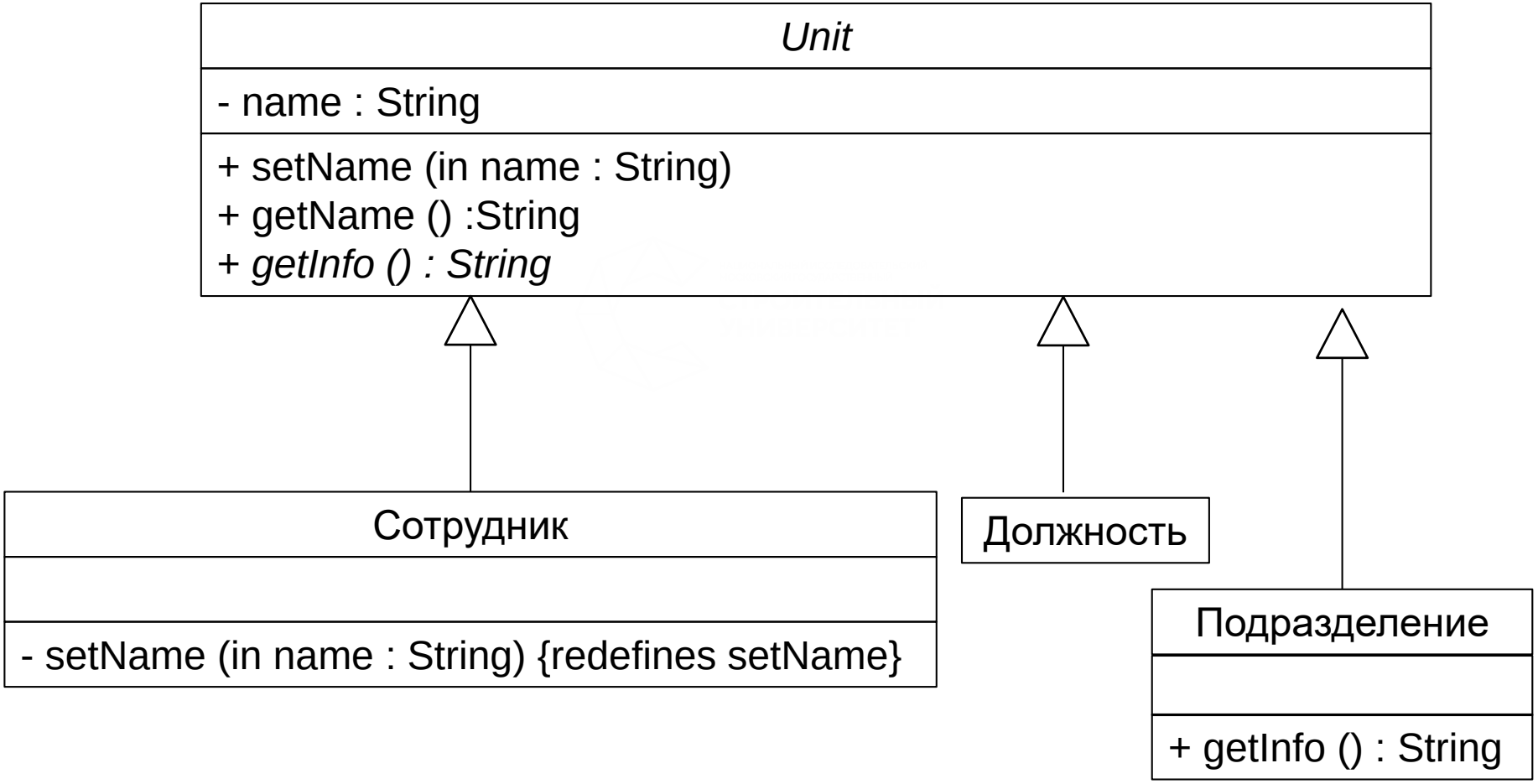


(а)



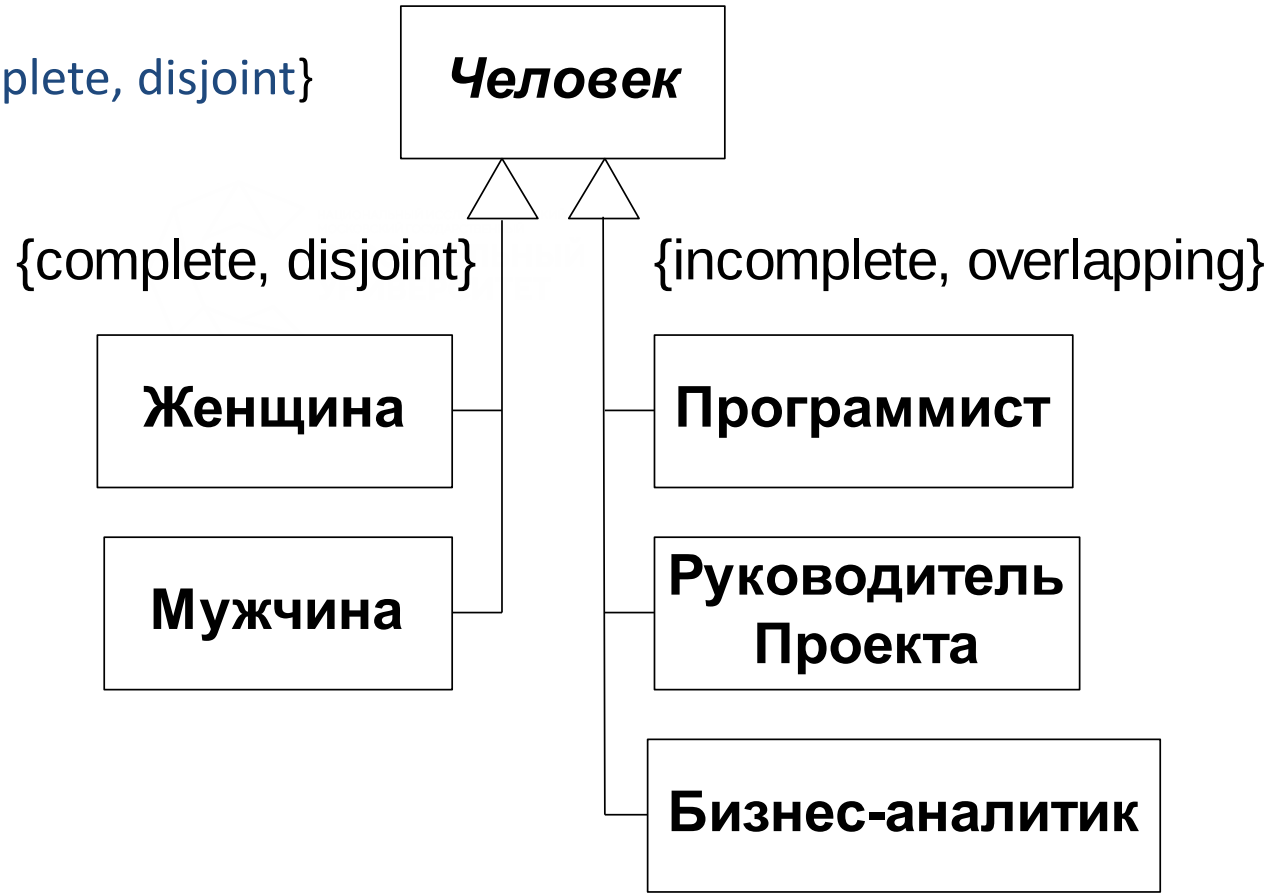
(б)

Примеры отношения обобщения

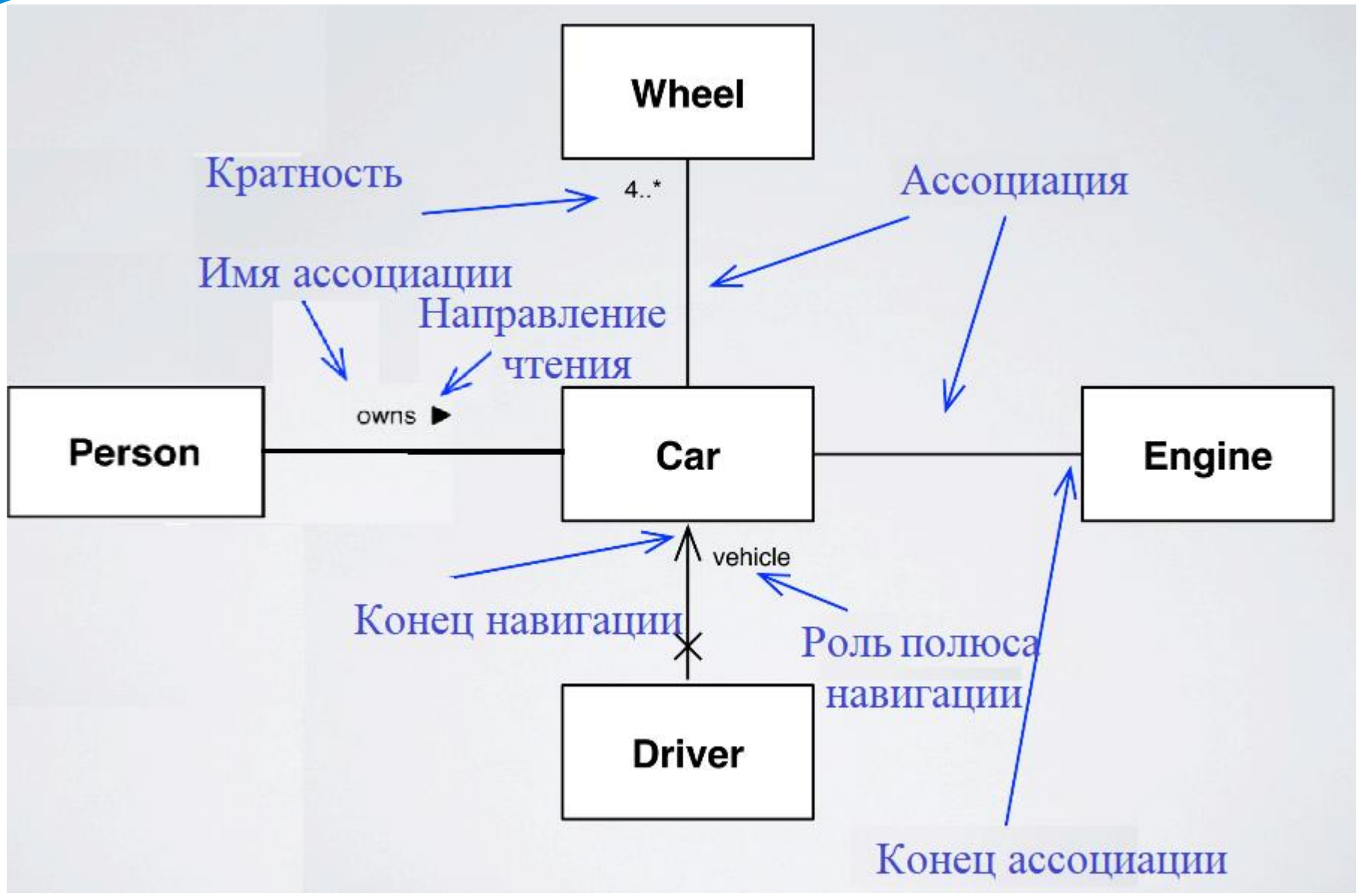


Ограничения на подмножество обобщений

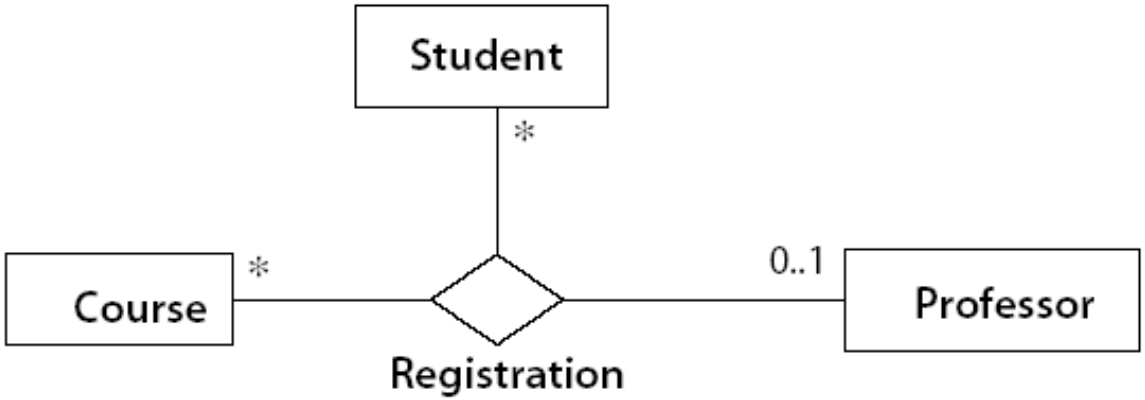
{complete}
{incomplete}
{disjoint}
{overlapping}
По умолчанию - {incomplete, disjoint}

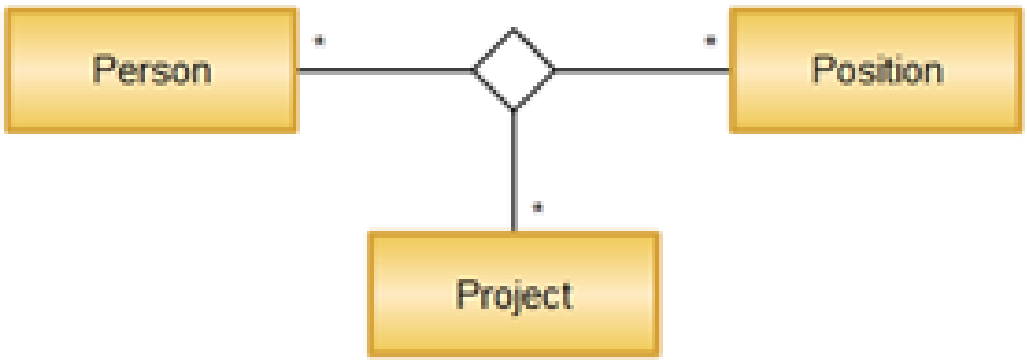


Ассоциация

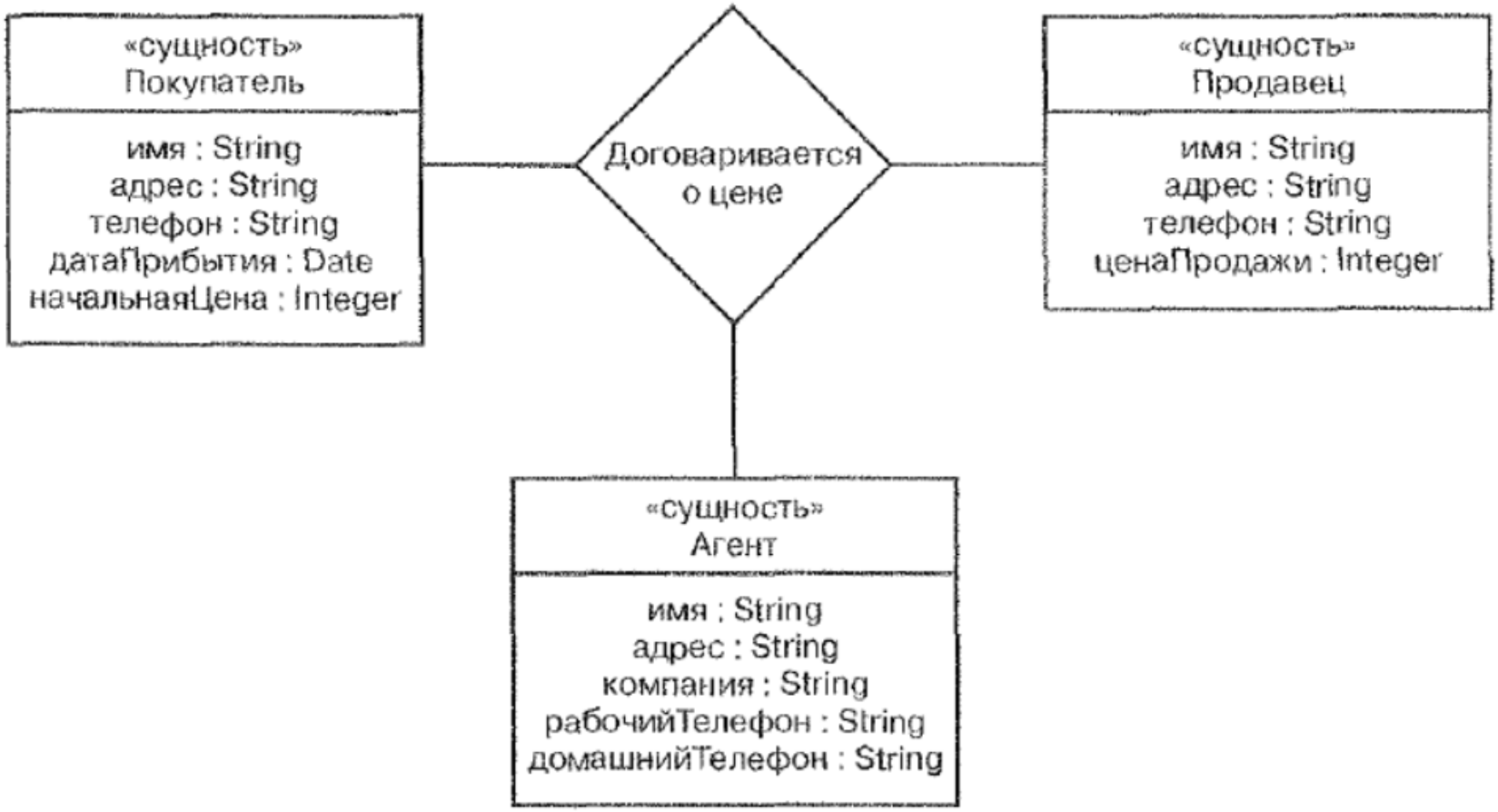

Wheel
Кратность
4..*
Ассоциация
Имя ассоциации
Направление чтения
owns
Person
Car
Engine
vehicle
Конец навигации
Роль полюса навигации
Driver
Конец ассоциации

Пример тернарной ассоциации

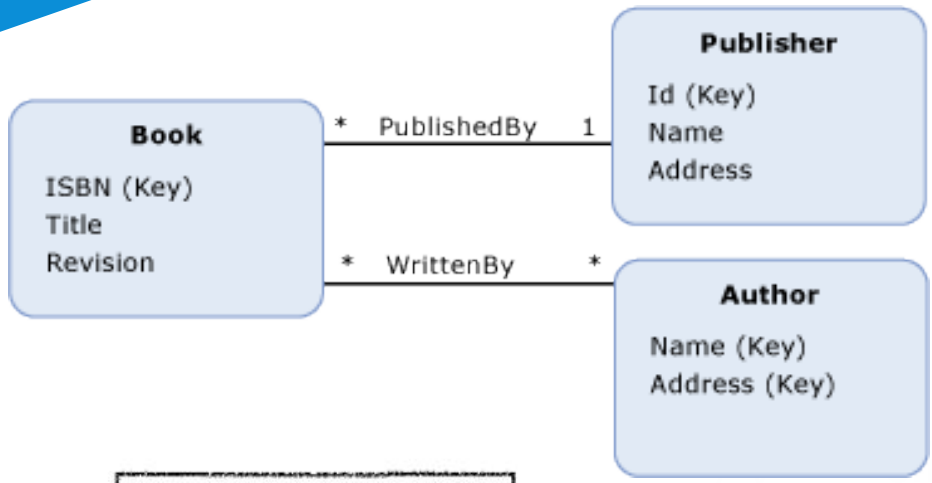


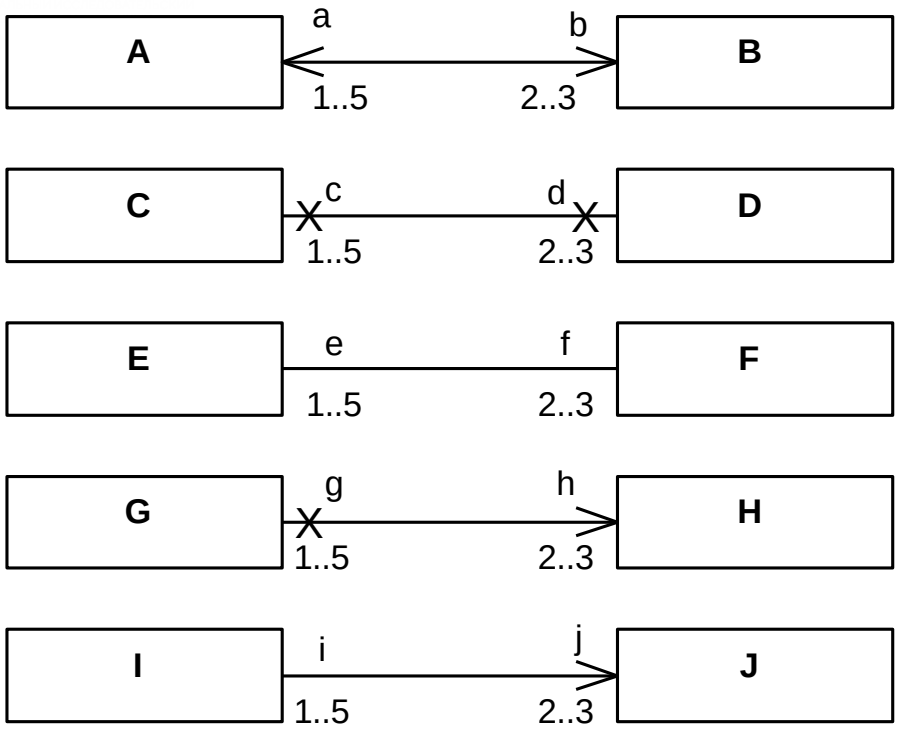


Пример тернарной ассоциации

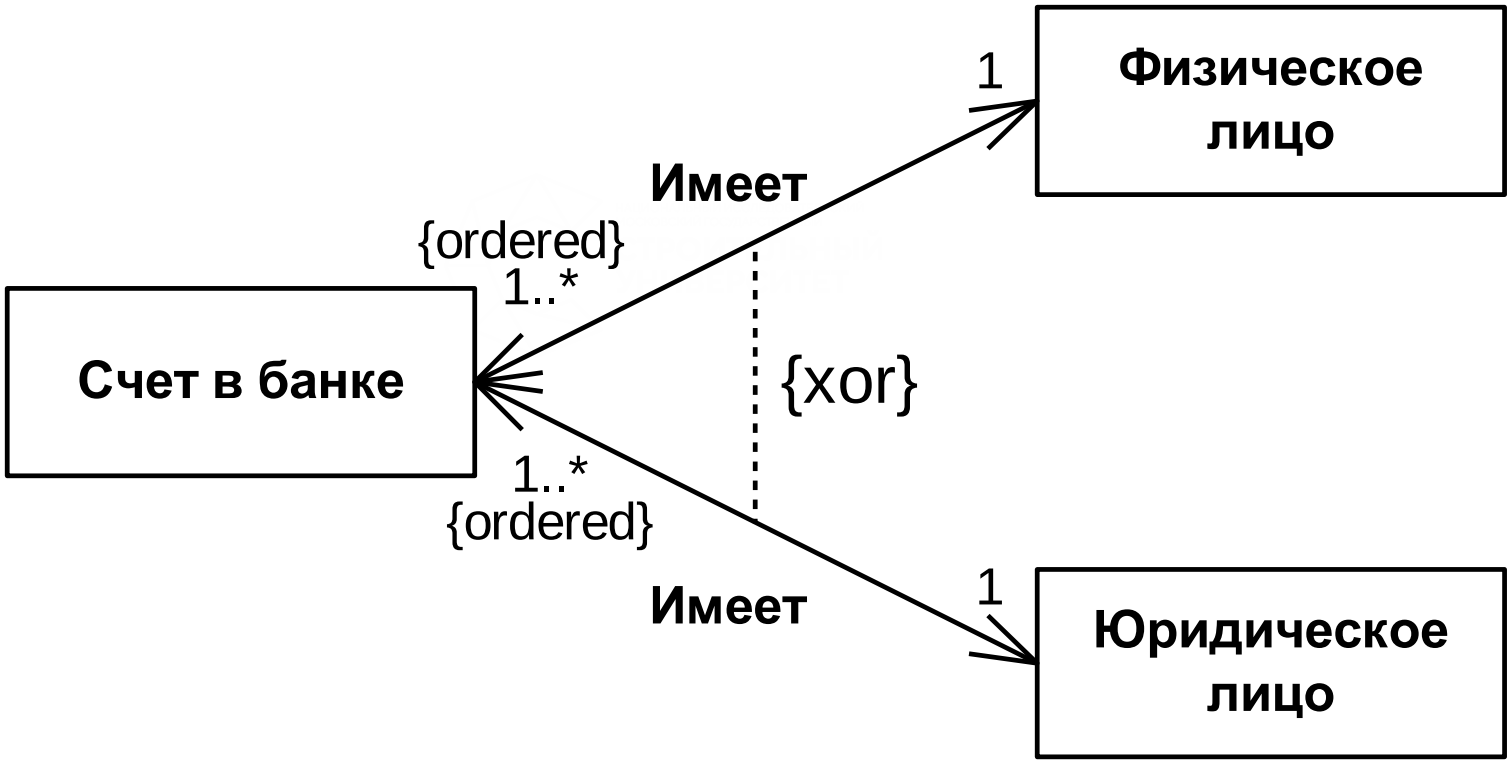


Варианты изображения навигации и кратности у концов ассоциации



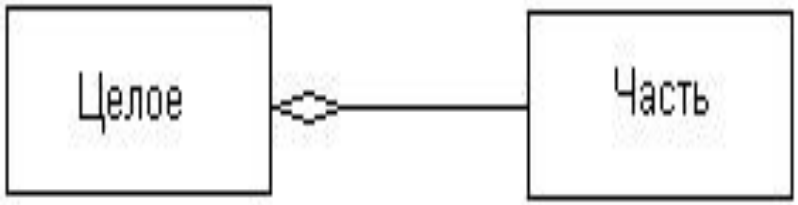


Исключающая ассоциация между тремя классами

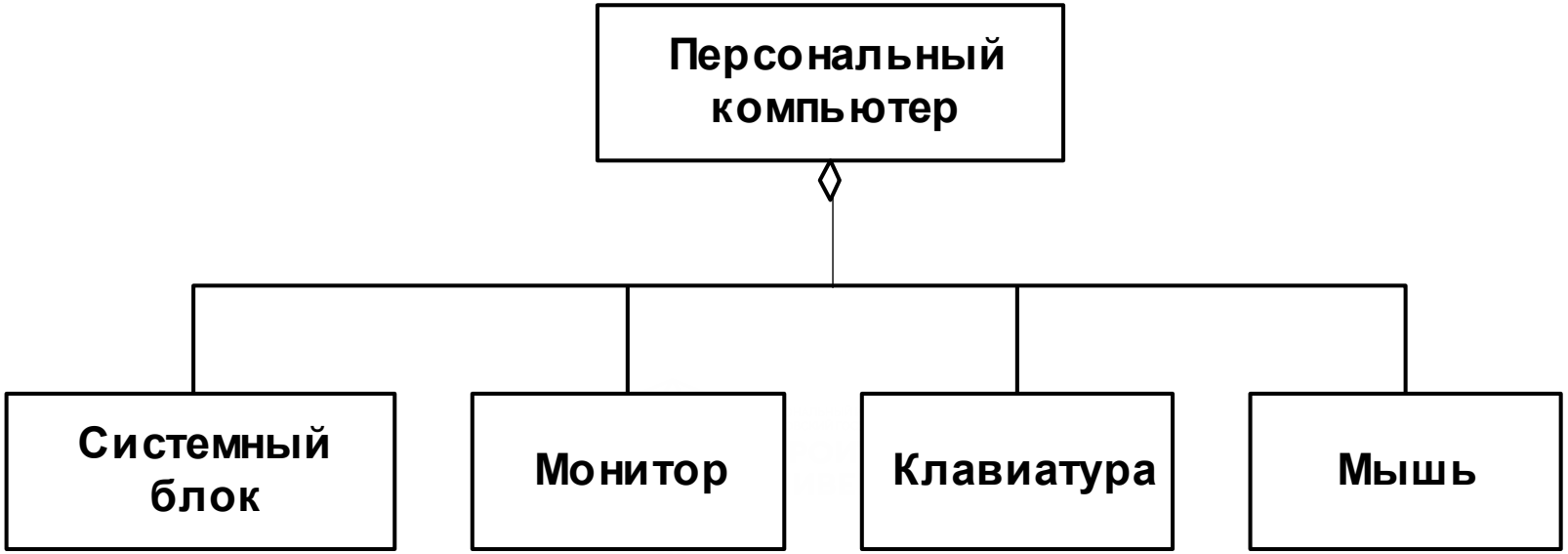


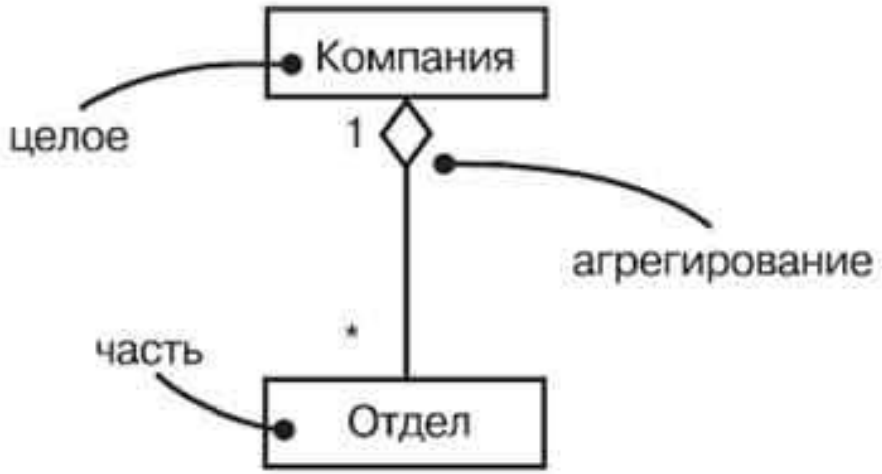
Агрегация (*aggregation*)

- – направленное отношение между двумя классами, предназначенное для представления ситуации, когда один из классов представляет собой некоторую сущность, которая включает в себя в качестве составных частей другие сущности



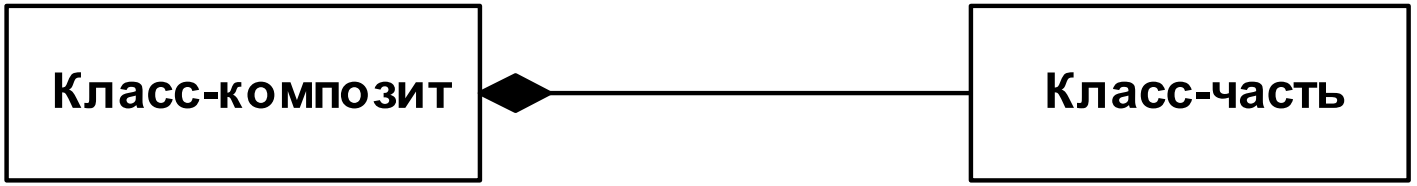
Пример отношения агрегации

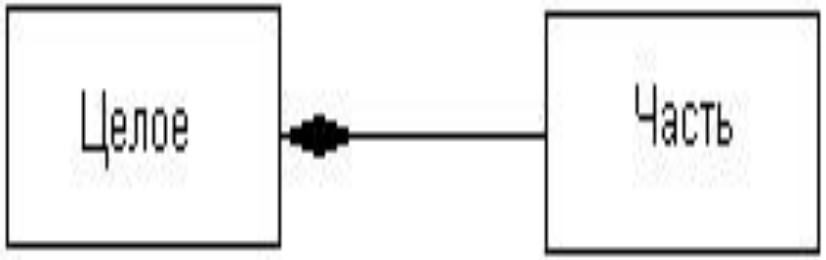




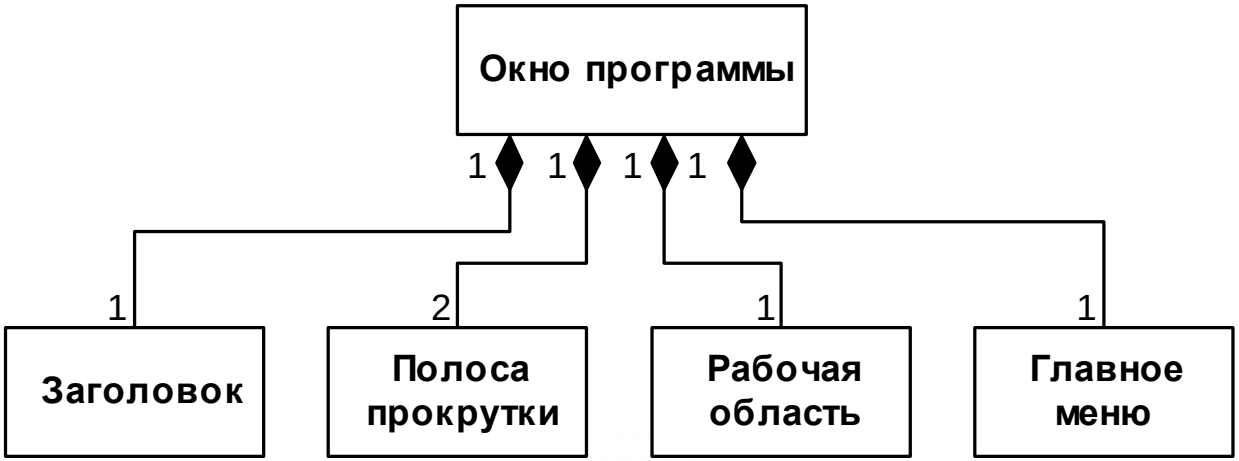
Композиция (composition)

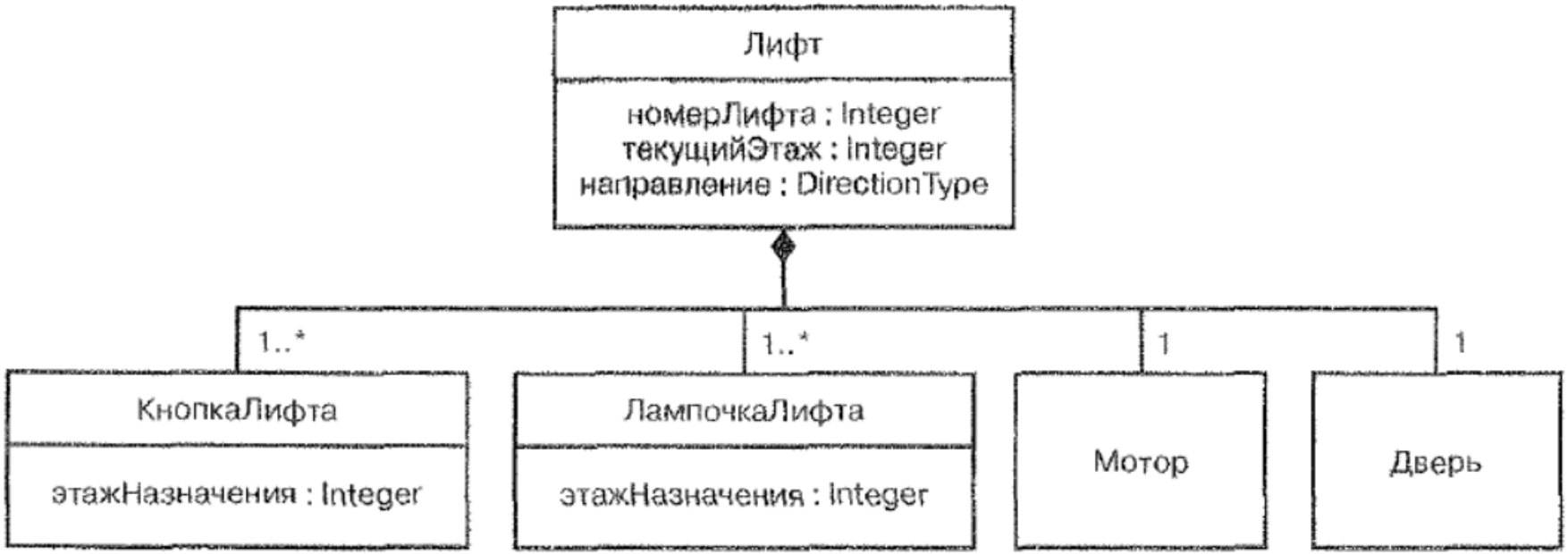
- или *композитная агрегация* предназначена для спецификации более сильной формы отношения "часть-целое", при которой с уничтожением объекта класса-контейнера уничтожаются и все объекты, являющимися его составными частями.



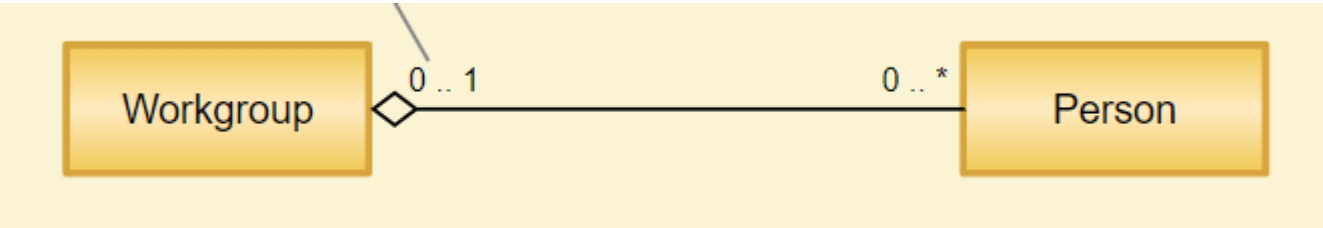


Пример отношения композиции





Программистская интерпретация



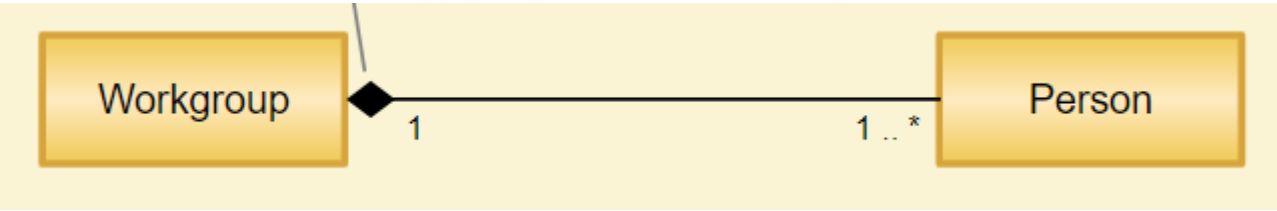


Диаграмма классов

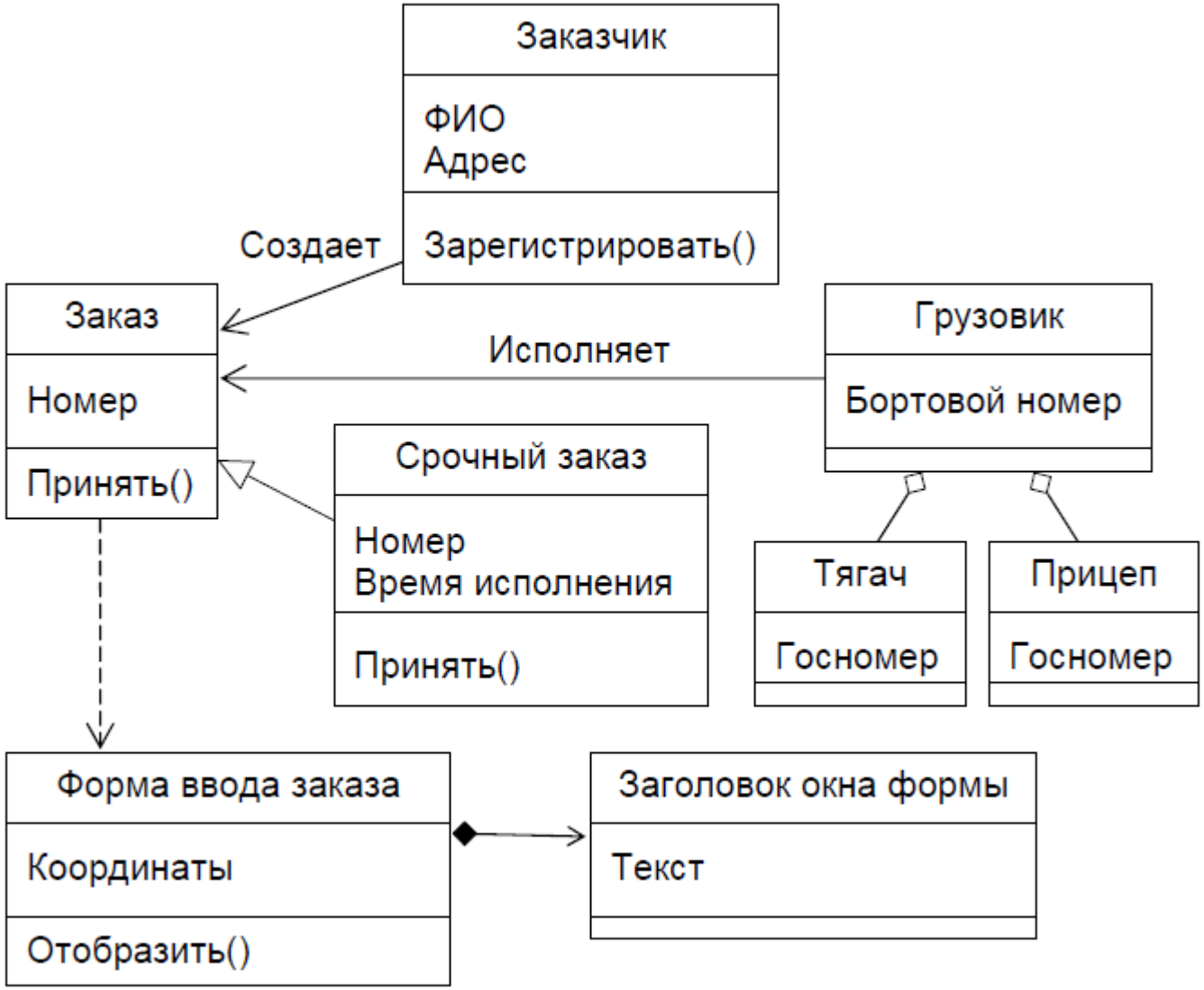
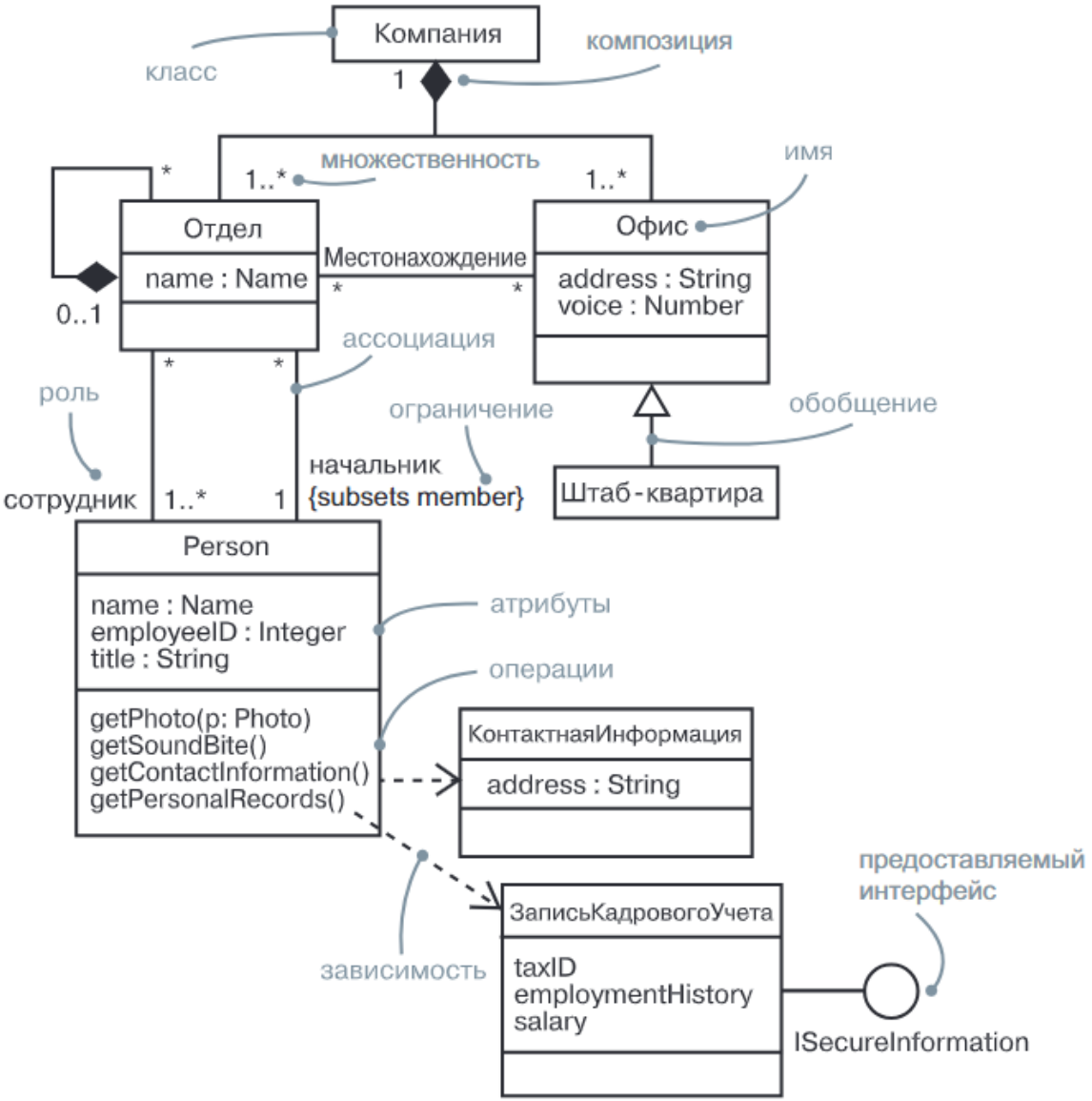
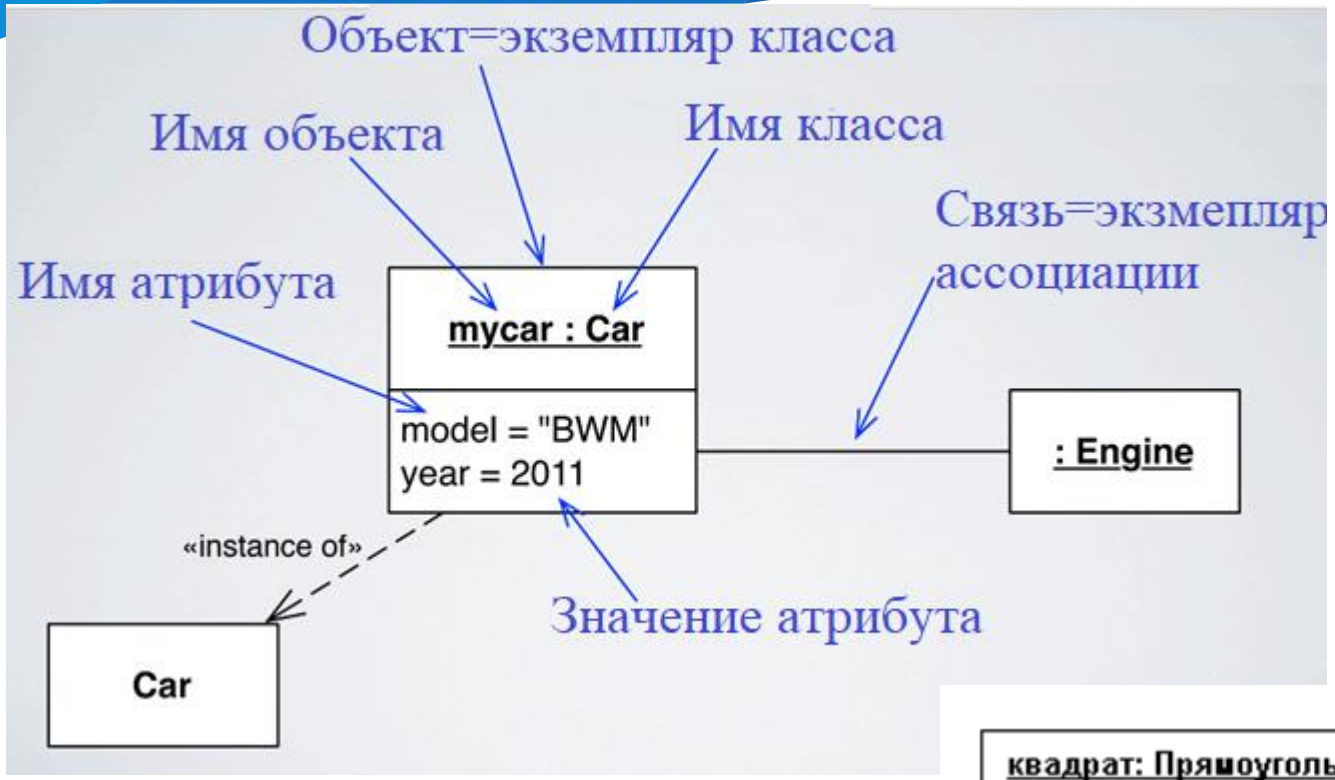


Диаграмма классов



Объекты



квадрат: Прямоугольник

(а)

квадрат:

(б)

квадрат: Прямоугольник

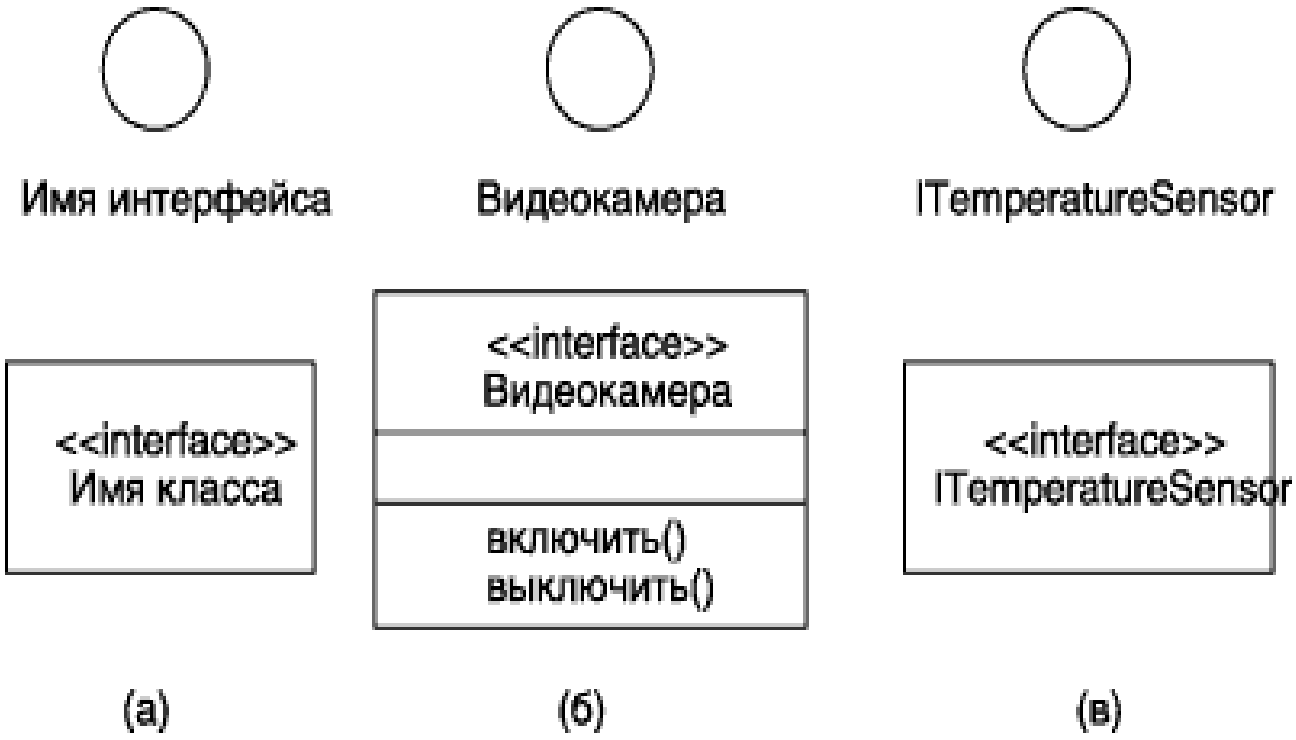
вершина = (1, 10)
сторона = 15
цвет_границы = черный
цвет_заливки = белый

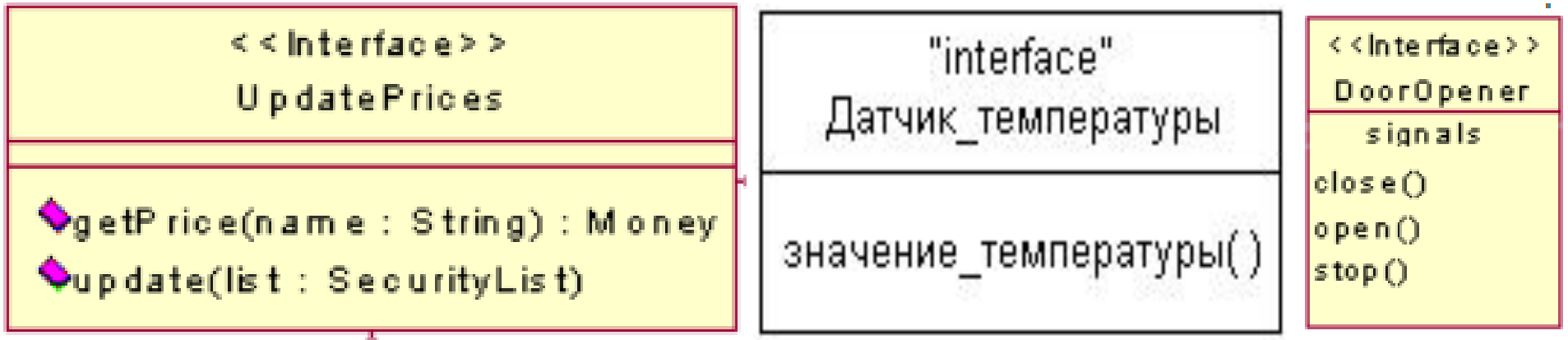
(в)

:Прямоугольник

(г)

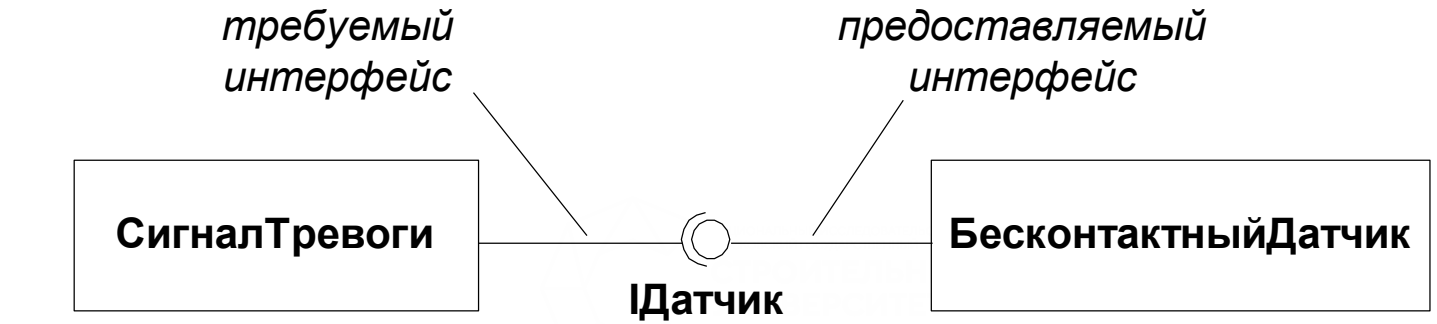
Интерфейсы

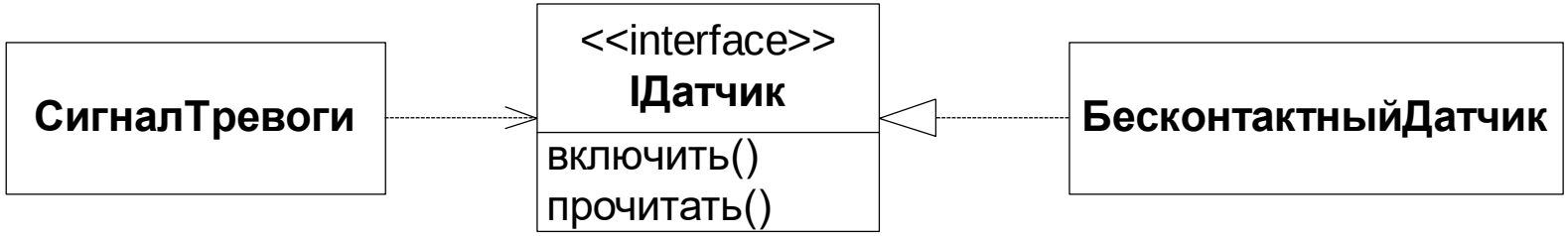

Имя интерфейса
Видеокамера
ITemperatureSensor
<<interface>>
Имя класса
<<interface>>
Видеокамера
включить()
выключить()
<<interface>>
ITemperatureSensor
(а)
(б)
(в)

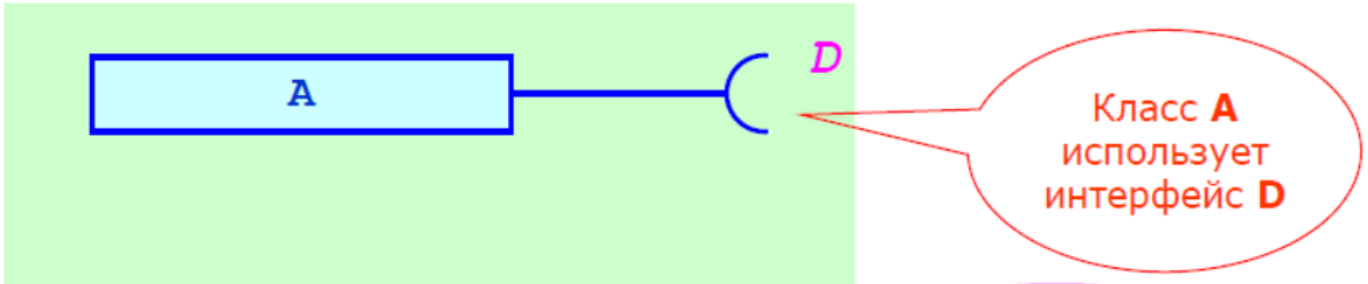

<<Interface>>
UpdatePrices
getPrice(name : String) : Money
update(list : SecurityList)
"interface"
Датчик_температуры
значение_температуры()
<<Interface>>
DoorOpener
signals
close()
open()
stop()

Интерфейс (interface)

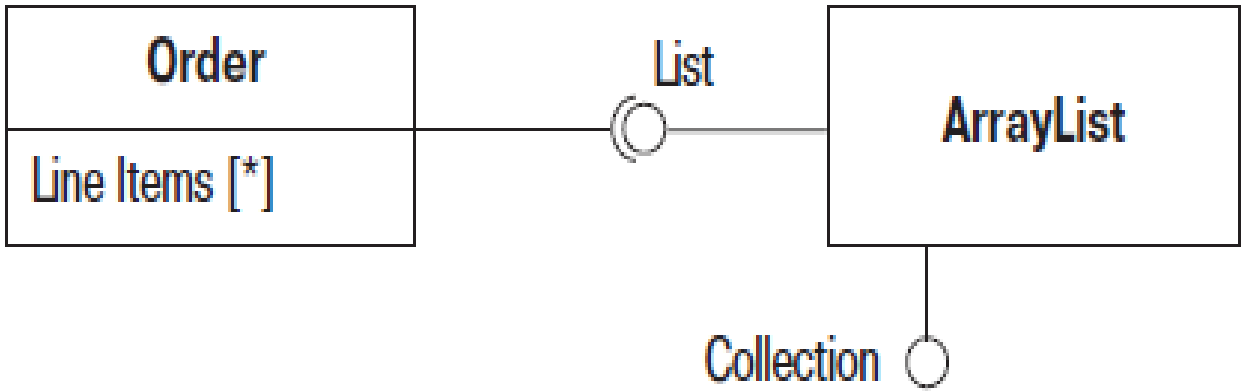
- – вид класса, который представляет собой объявление множества общедоступных характеристик и обязанностей.



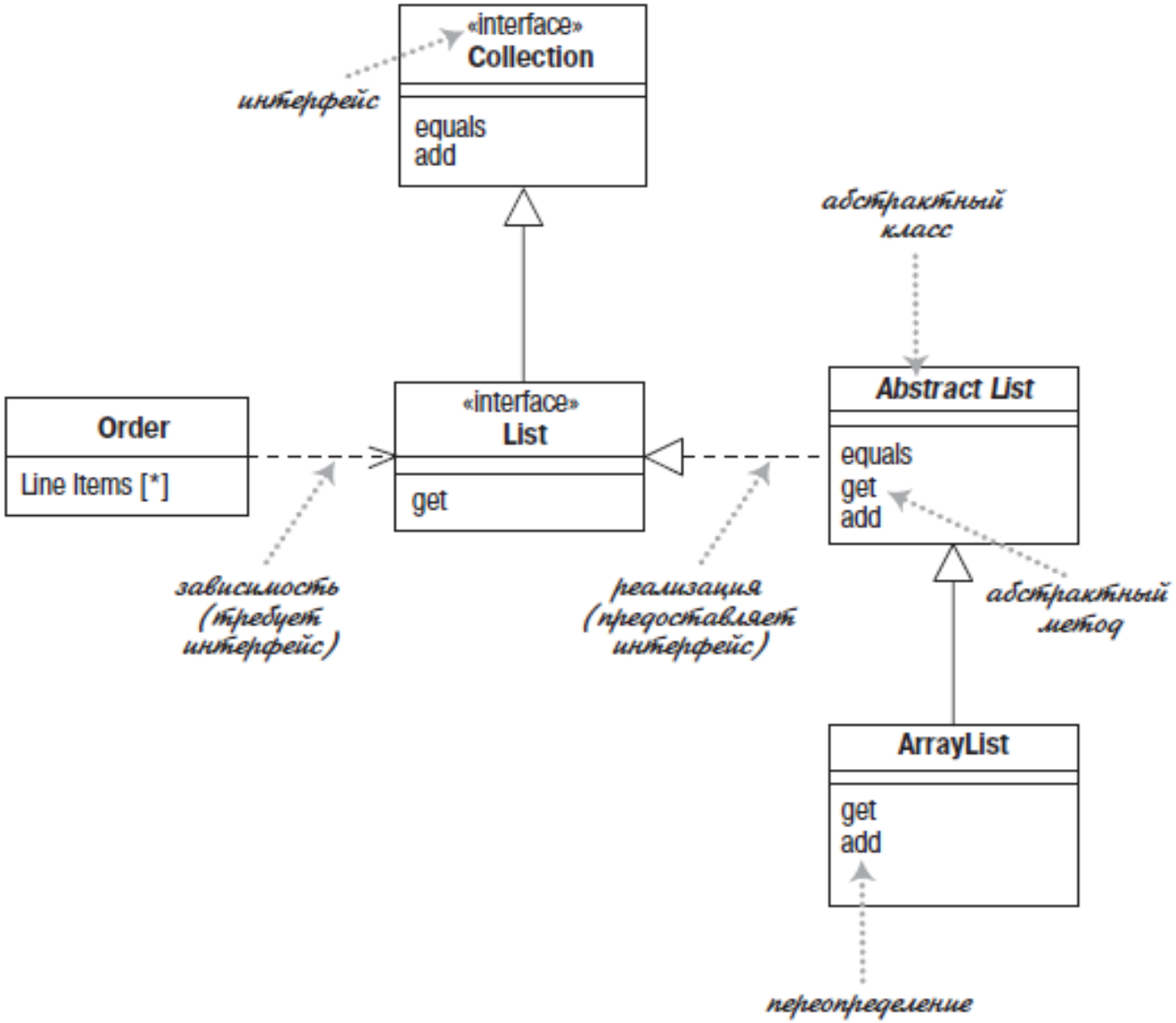




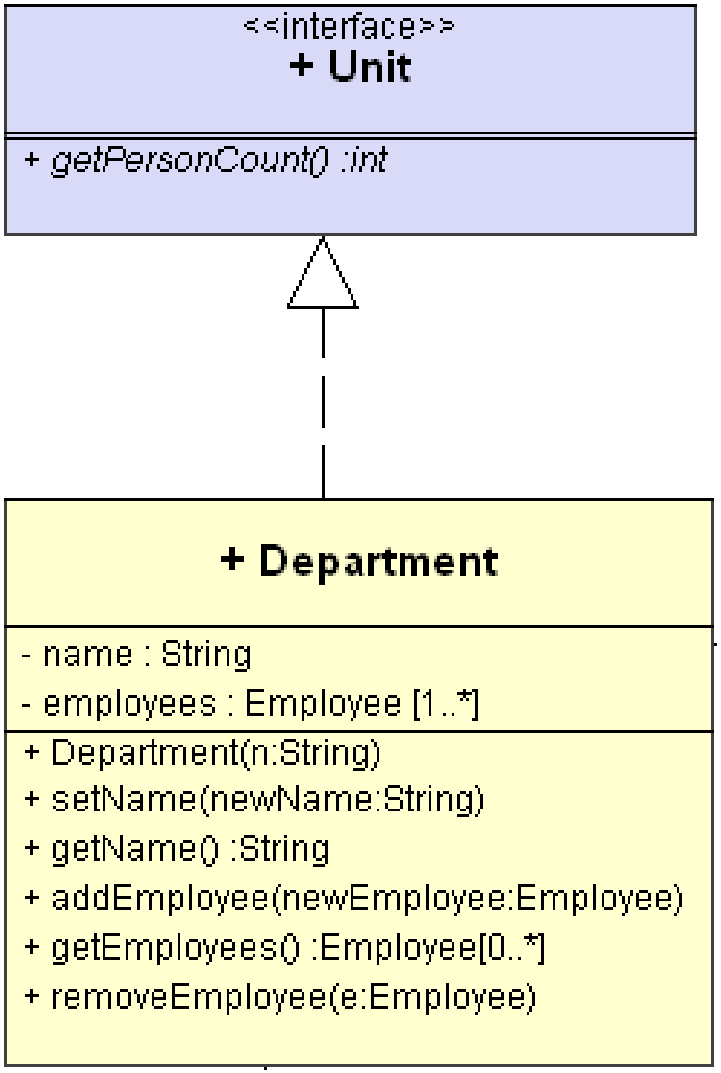
Интерфейс *(interface)*

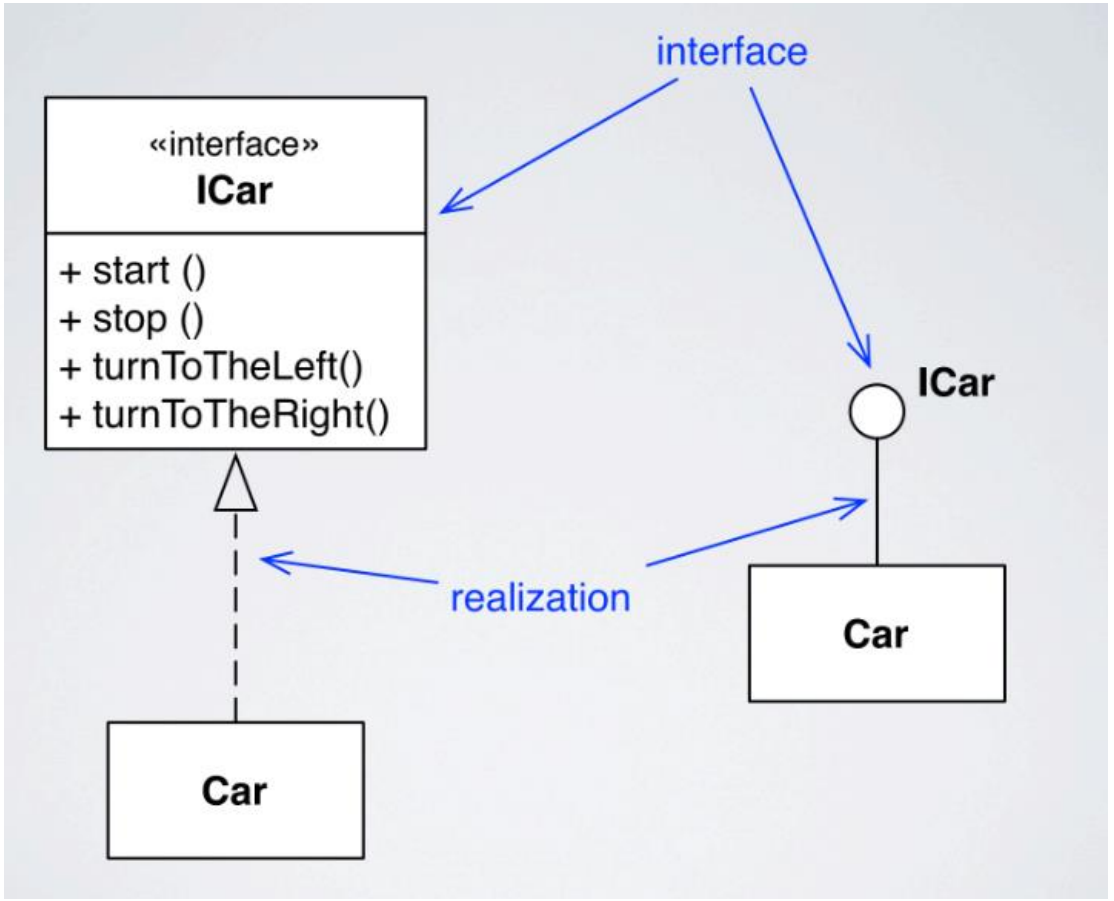


Интерфейс *(interface)*



Отношение реализации

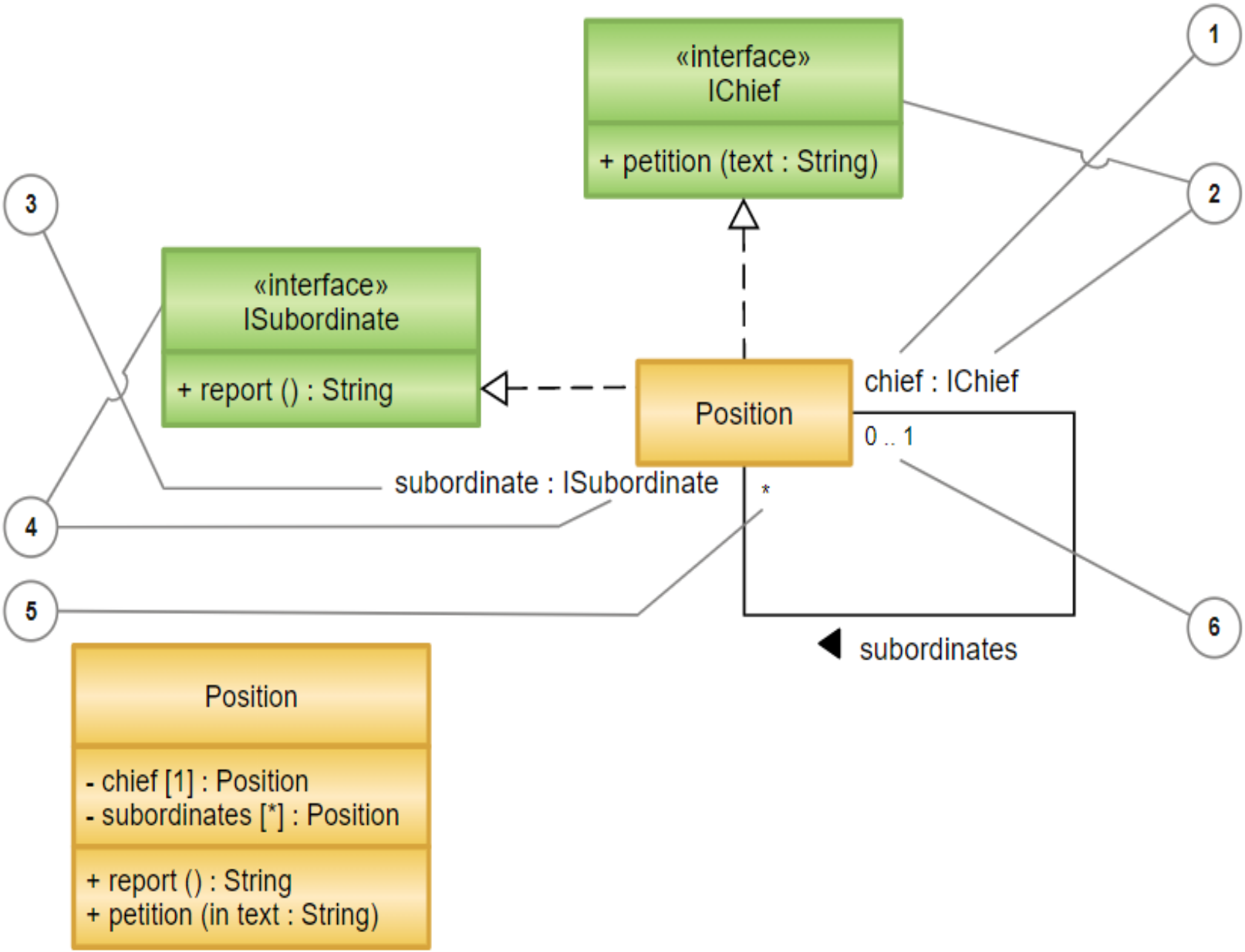




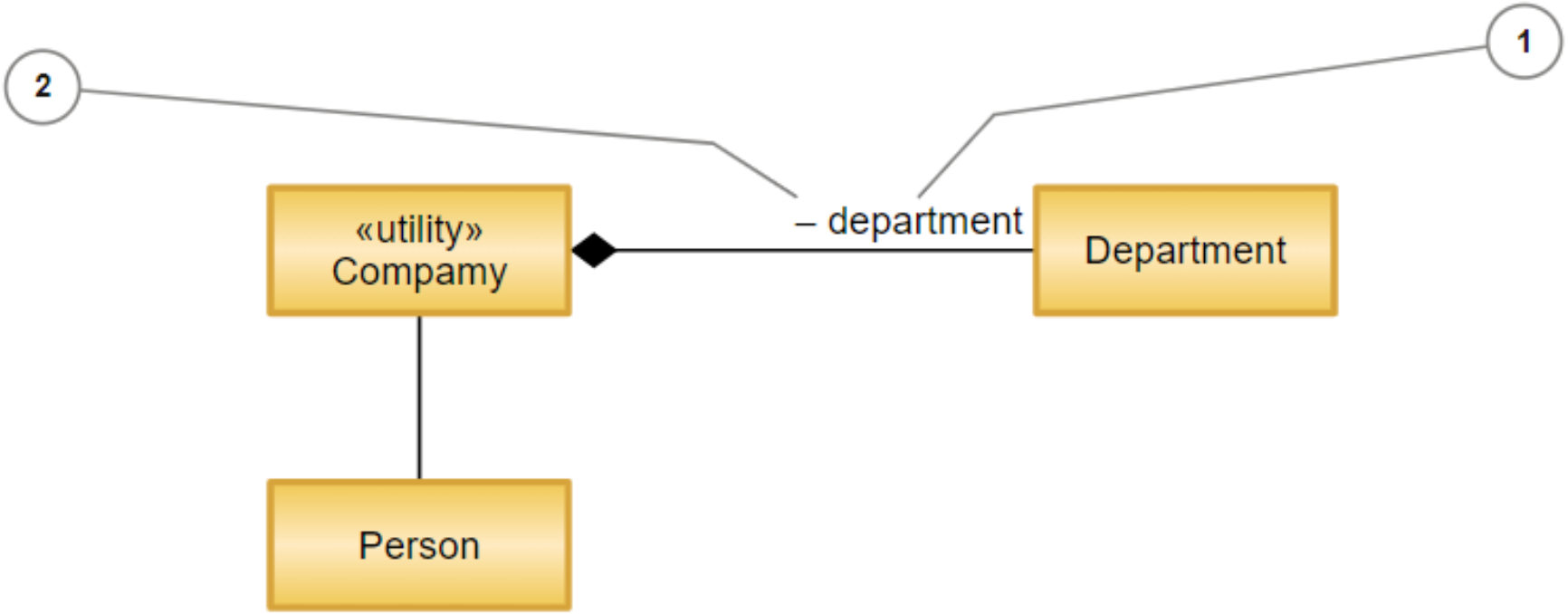
Имена полюсов

employee	employer
Joe Doe	Simplex
Mary Brown	Simplex
Jean Smith	United Widgets

Роль полюсов



Видимость



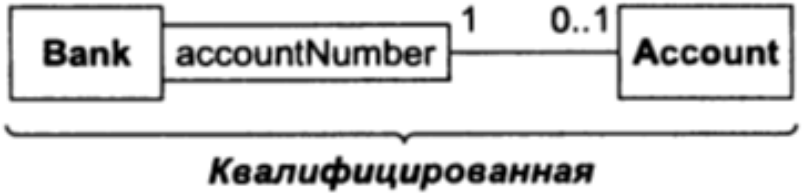
Упорядочение

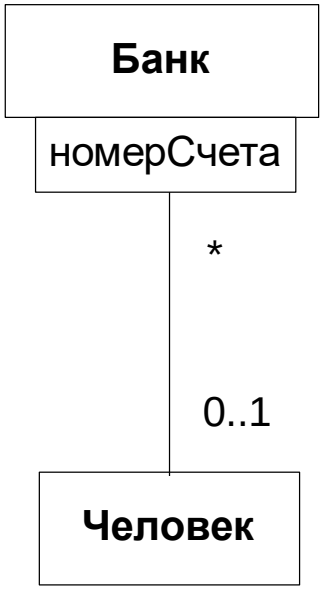
Табл. **Свойства упорядоченности и уникальности**

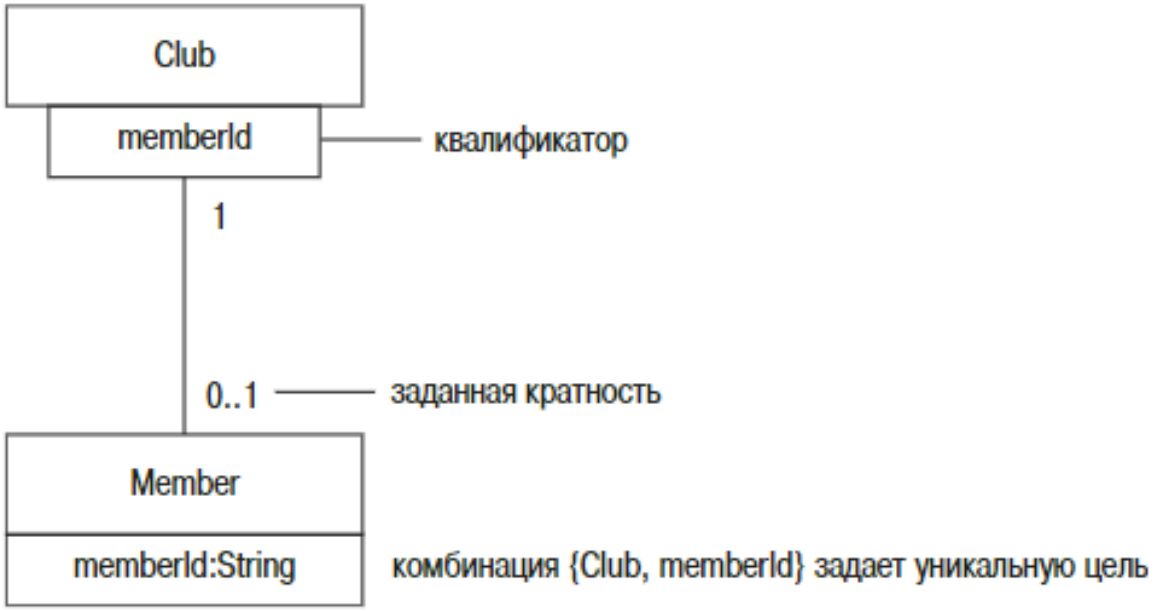
	Упорядоченное множество	Неупорядоченное множество
Есть одинаковые элементы	{sequence} последовательность	{bag} мультимножество
Нет одинаковых элементов	{ordered} упорядоченное множество	{set} множество

Квалификатор *(qualifier)*

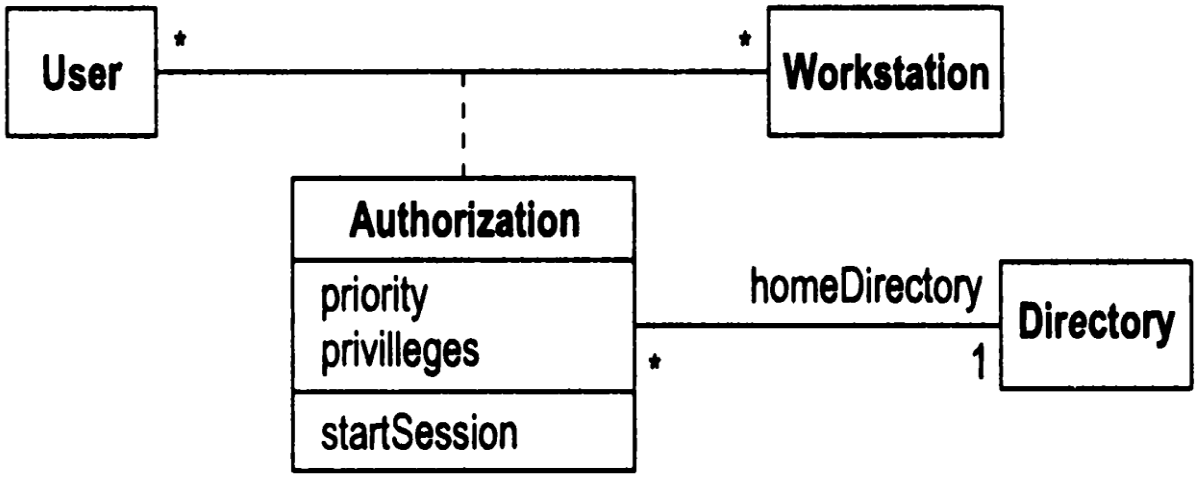
- *Квалификатор* (qualifier) объявляет разбиение множества ассоциированных экземпляров относительно экземпляра на л конце ассоциации





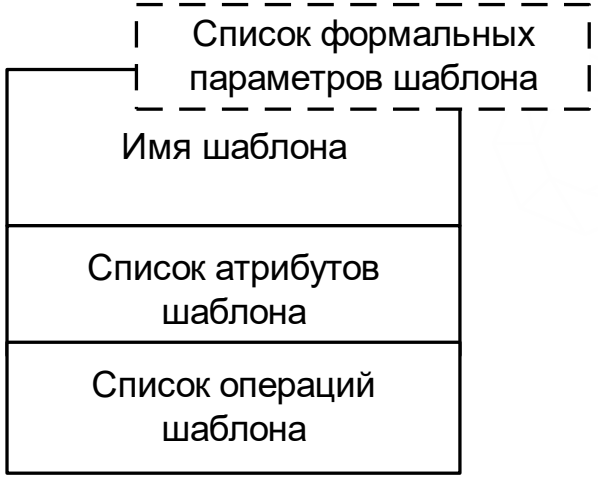


Ассоциация класс *(association class)*

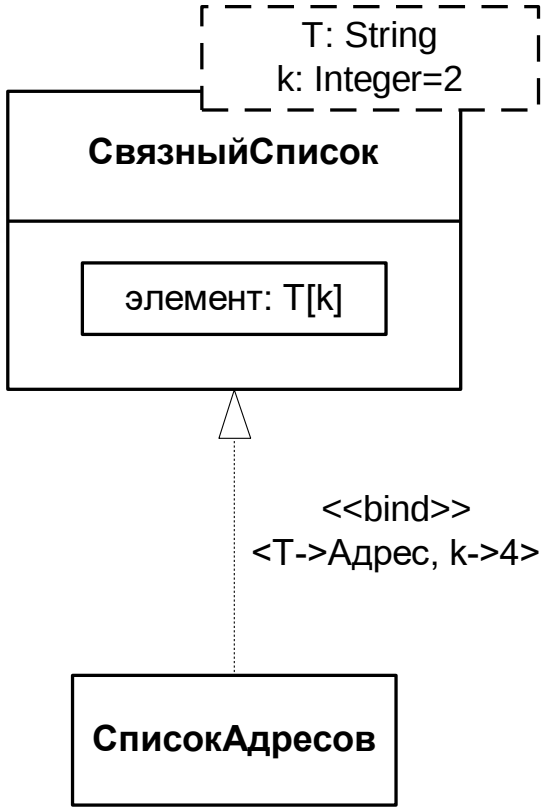


Шаблон *(template)*

- – классификатор, который в своем описании имеет несколько формальных параметров

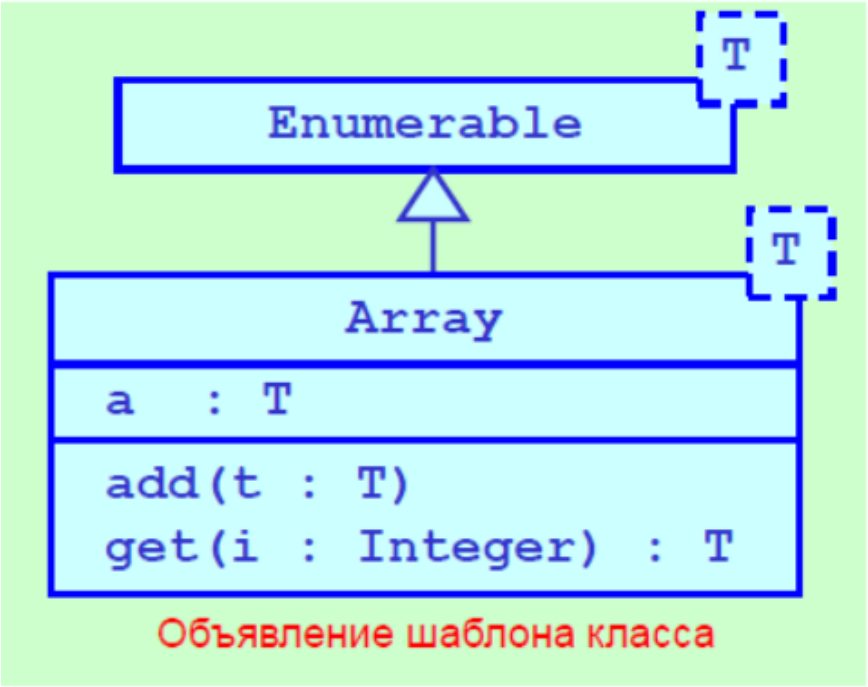


(а)

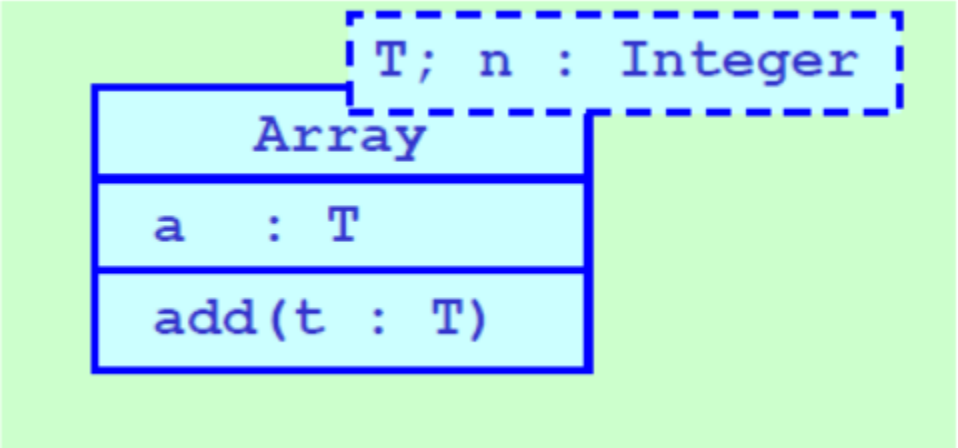


(б)

Шаблон *(template)*

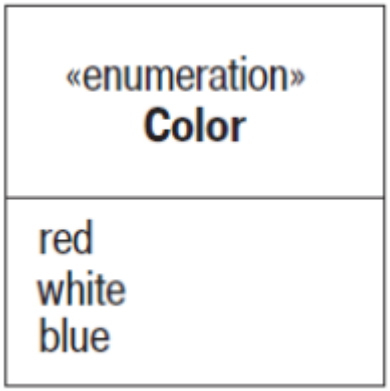


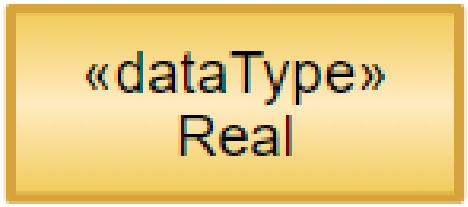
Объявление шаблона класса



Типы данных


«primitive»
Простой тип

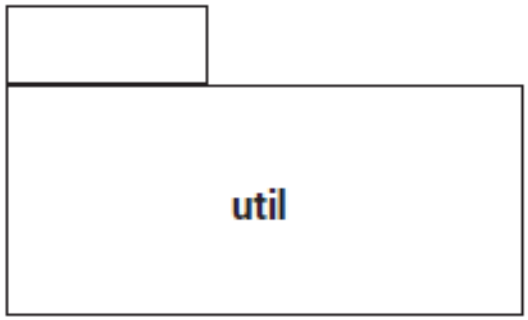

«enumeration»
Color
red
white
blue

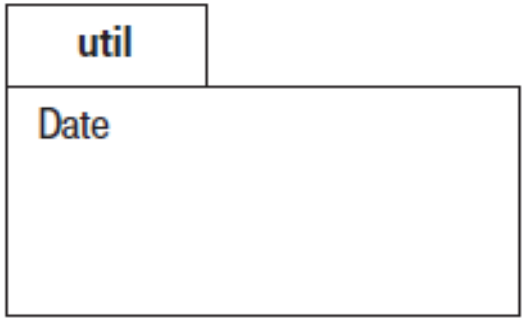

«dataType»
Real

Активный класс

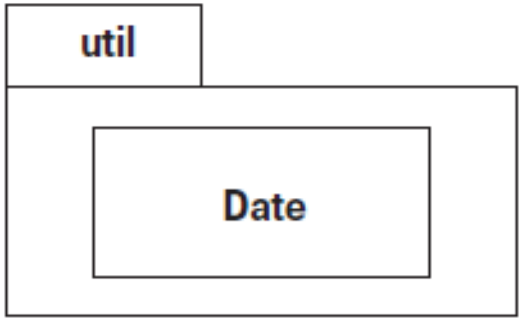
активный класс (UML 2)

Пакеты

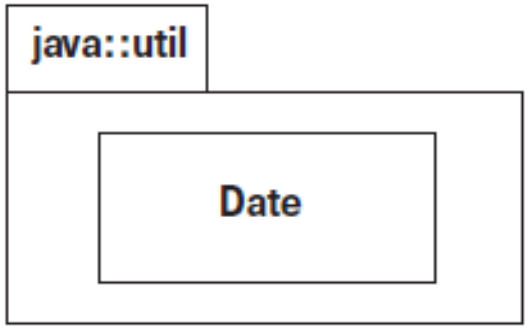




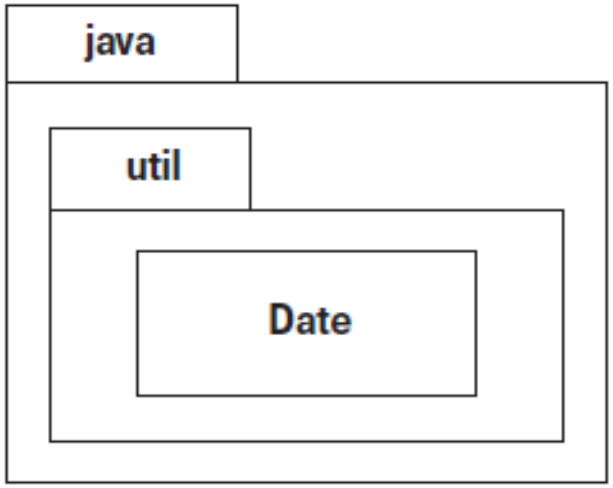
Содержимое, перечисленное в прямоугольнике



Содержимое в виде диаграммы в прямоугольнике



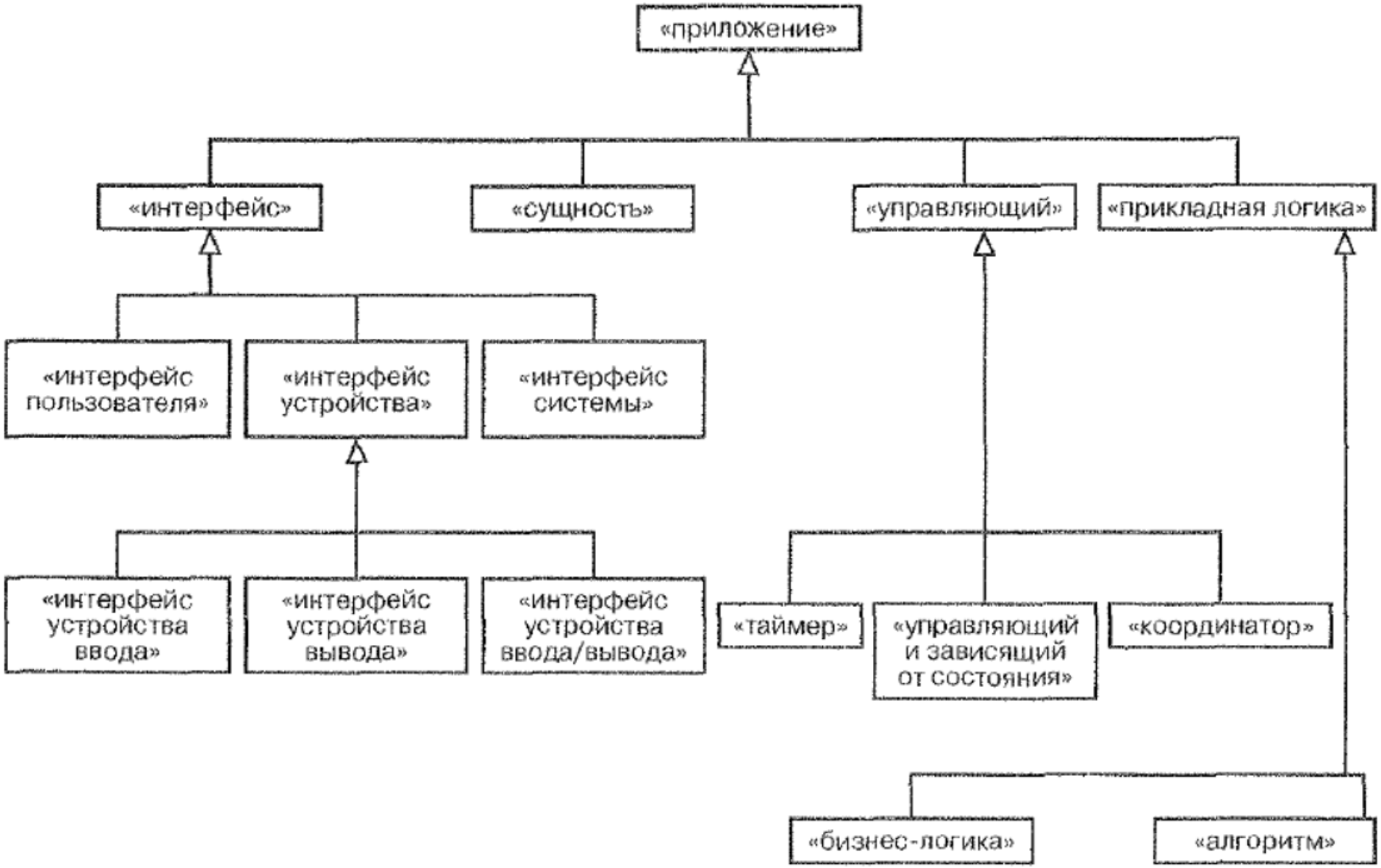
Полностью определенное имя пакета



Вложенные пакеты

Полностью определенное имя класса

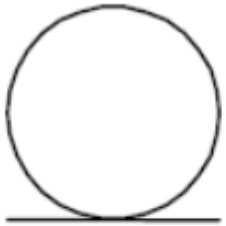
Классы приложений



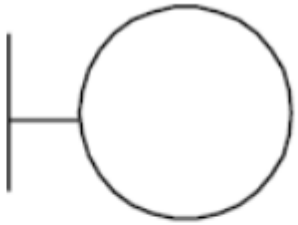
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

Классы анализа

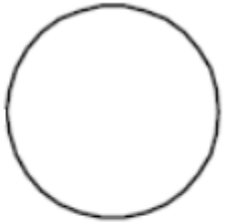
управляющий класс



класс-сущность



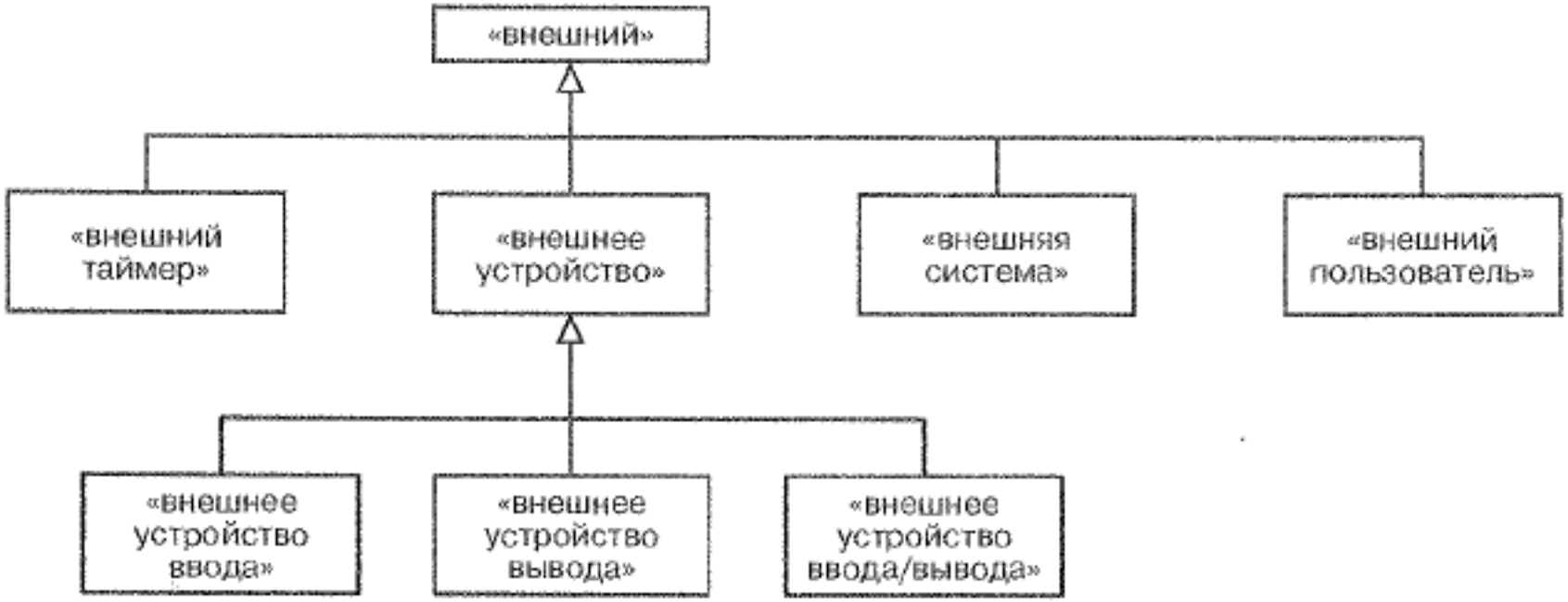
граничный класс



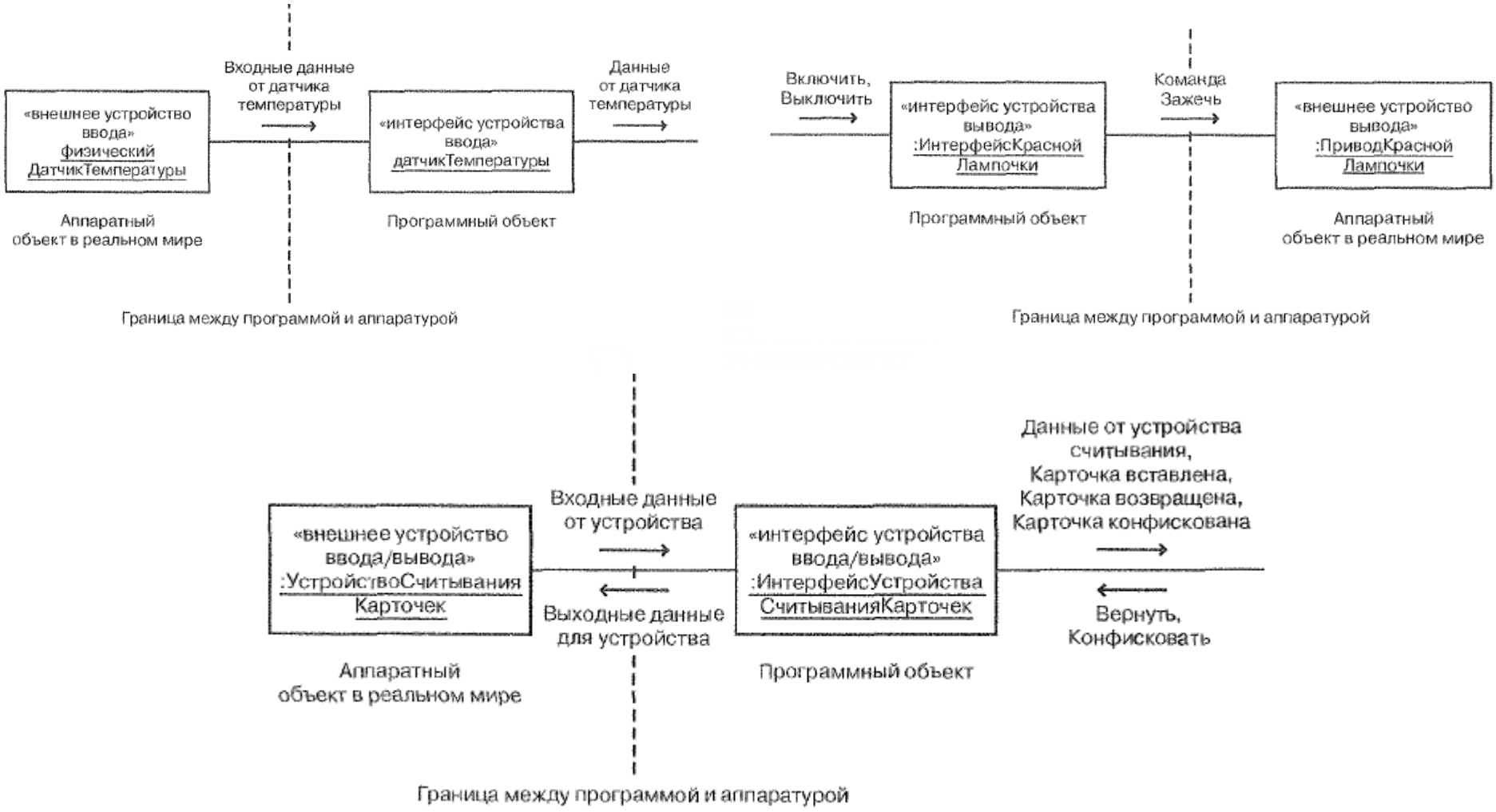
интерфейс

<<control>> Имя класса	<<entity>> Имя класса	<<boundary>> Имя класса	<<interface>> Имя класса

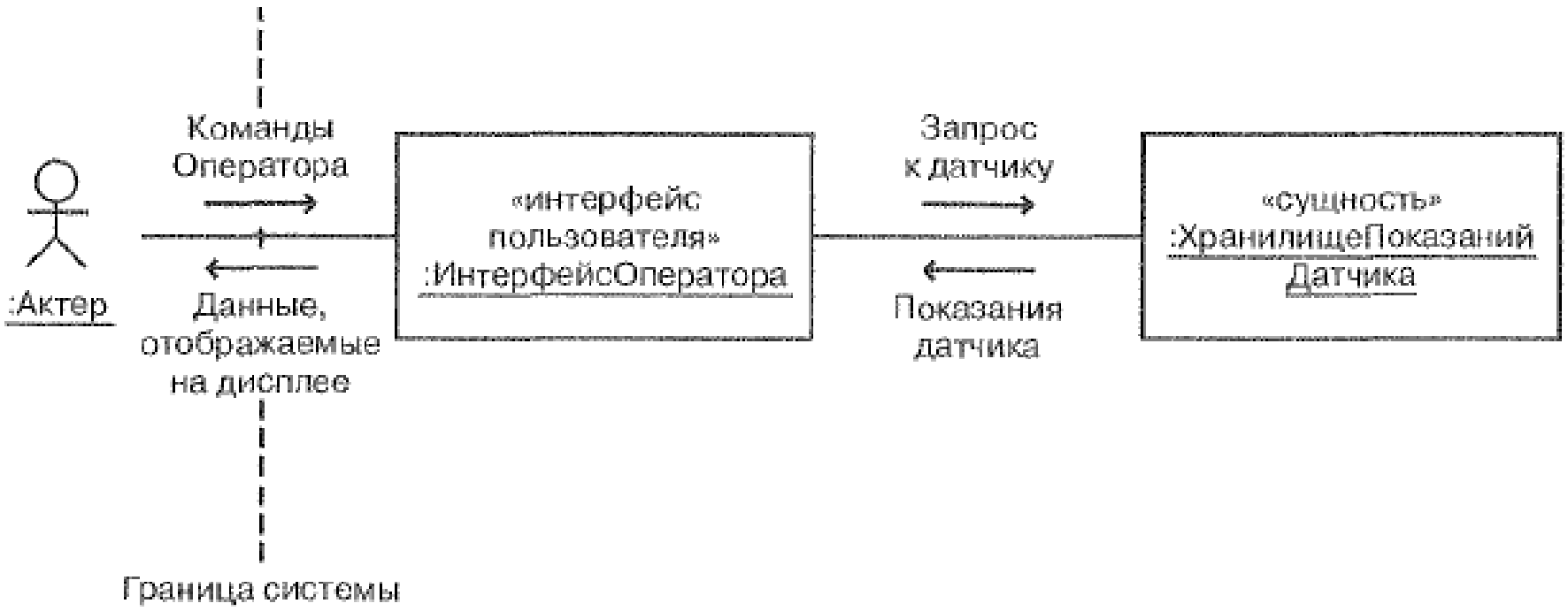
Внешние классы



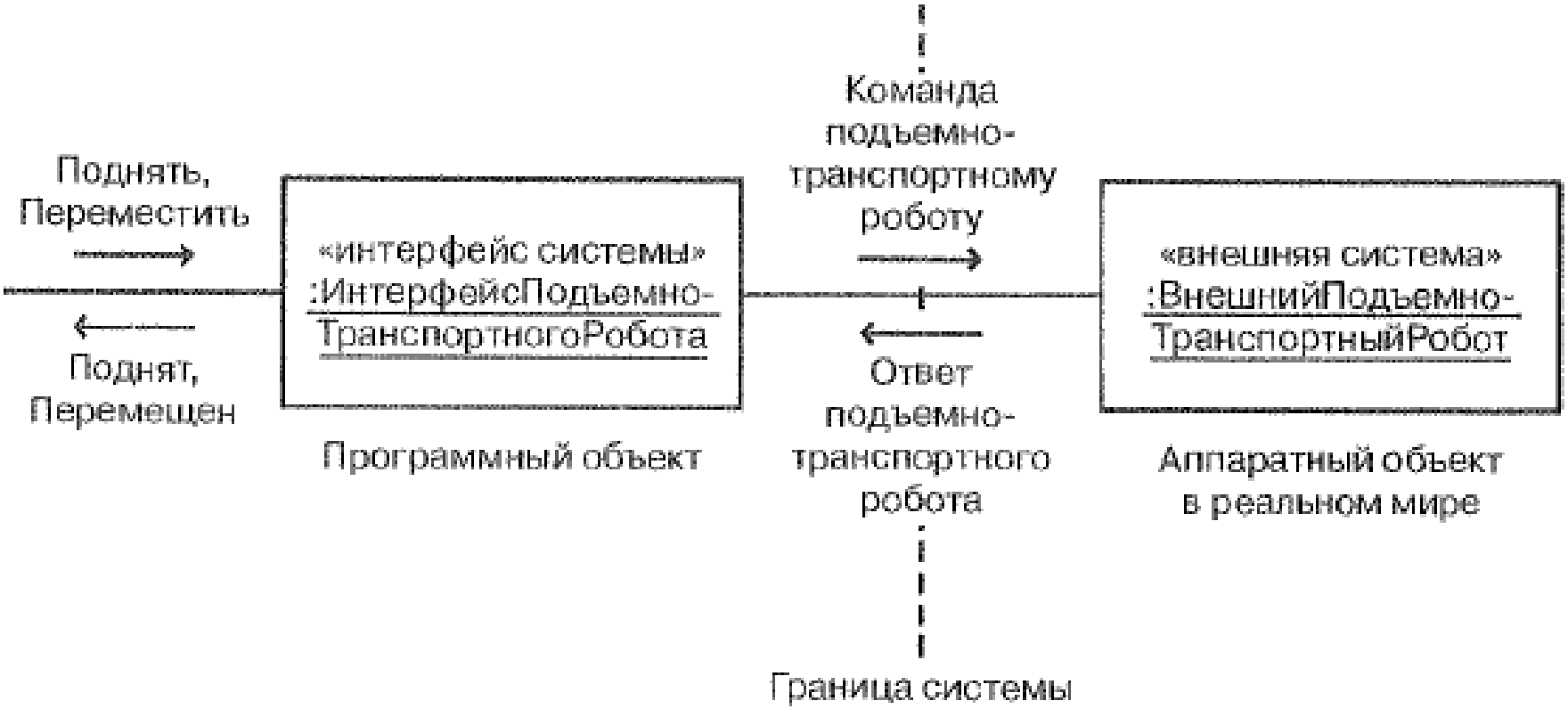
Объект интерфейса устройства



Объект интерфейса пользователя

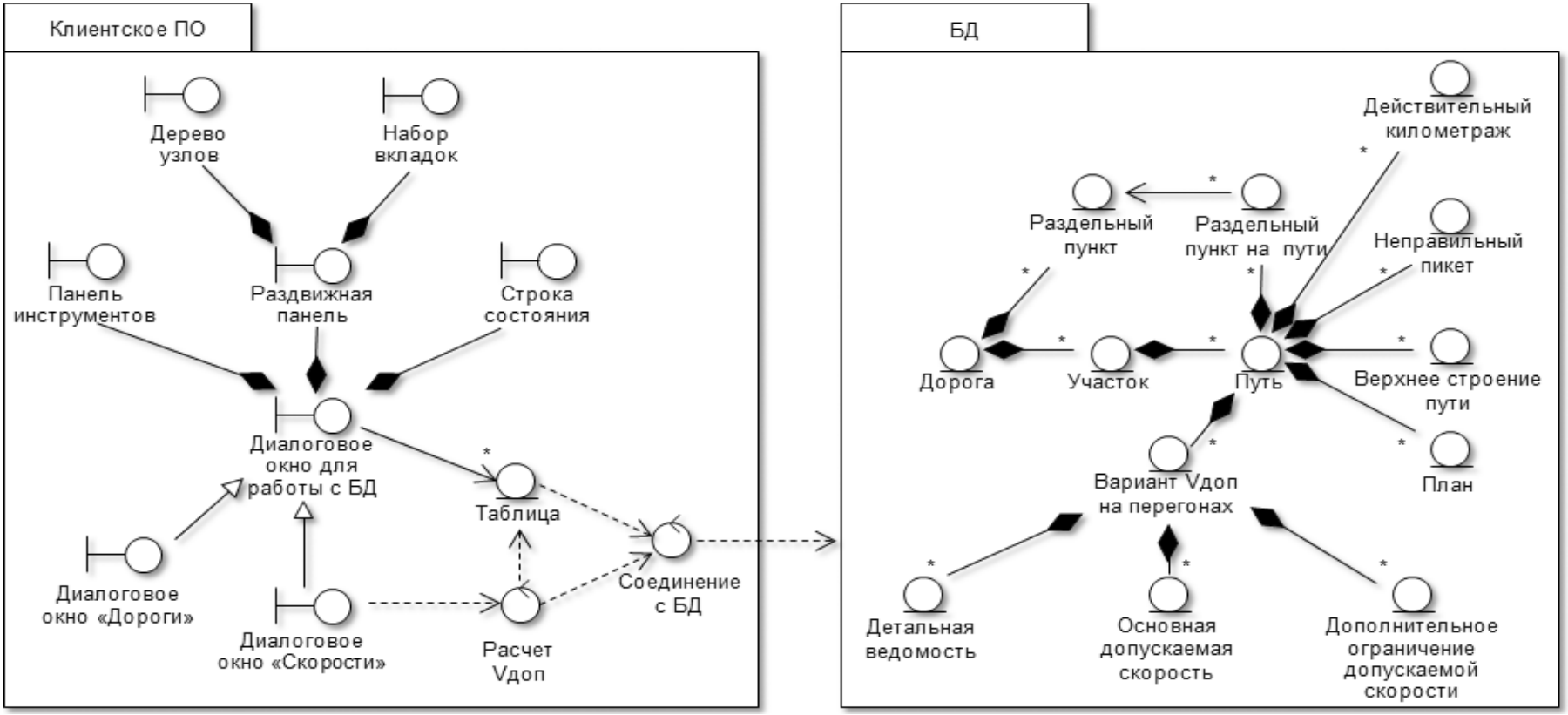


Объект интерфейса системы

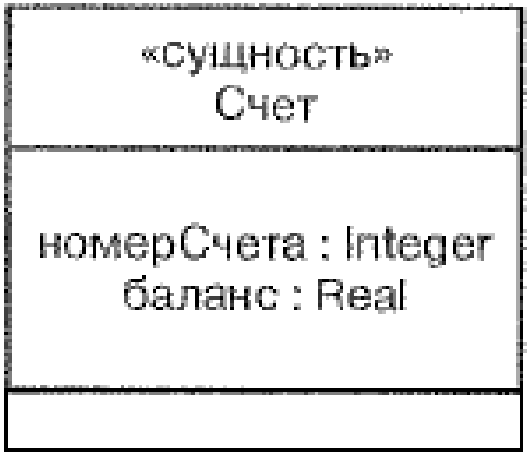


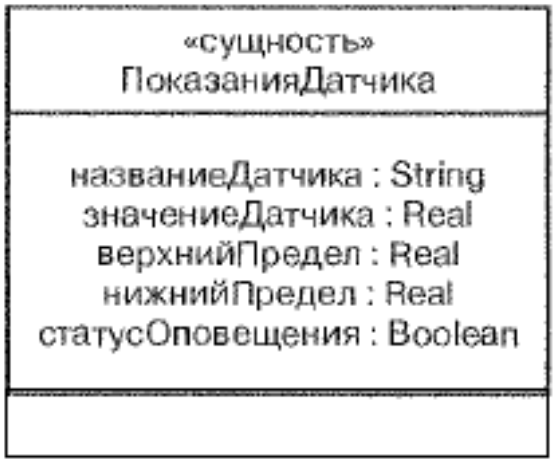
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ

Классы анализа

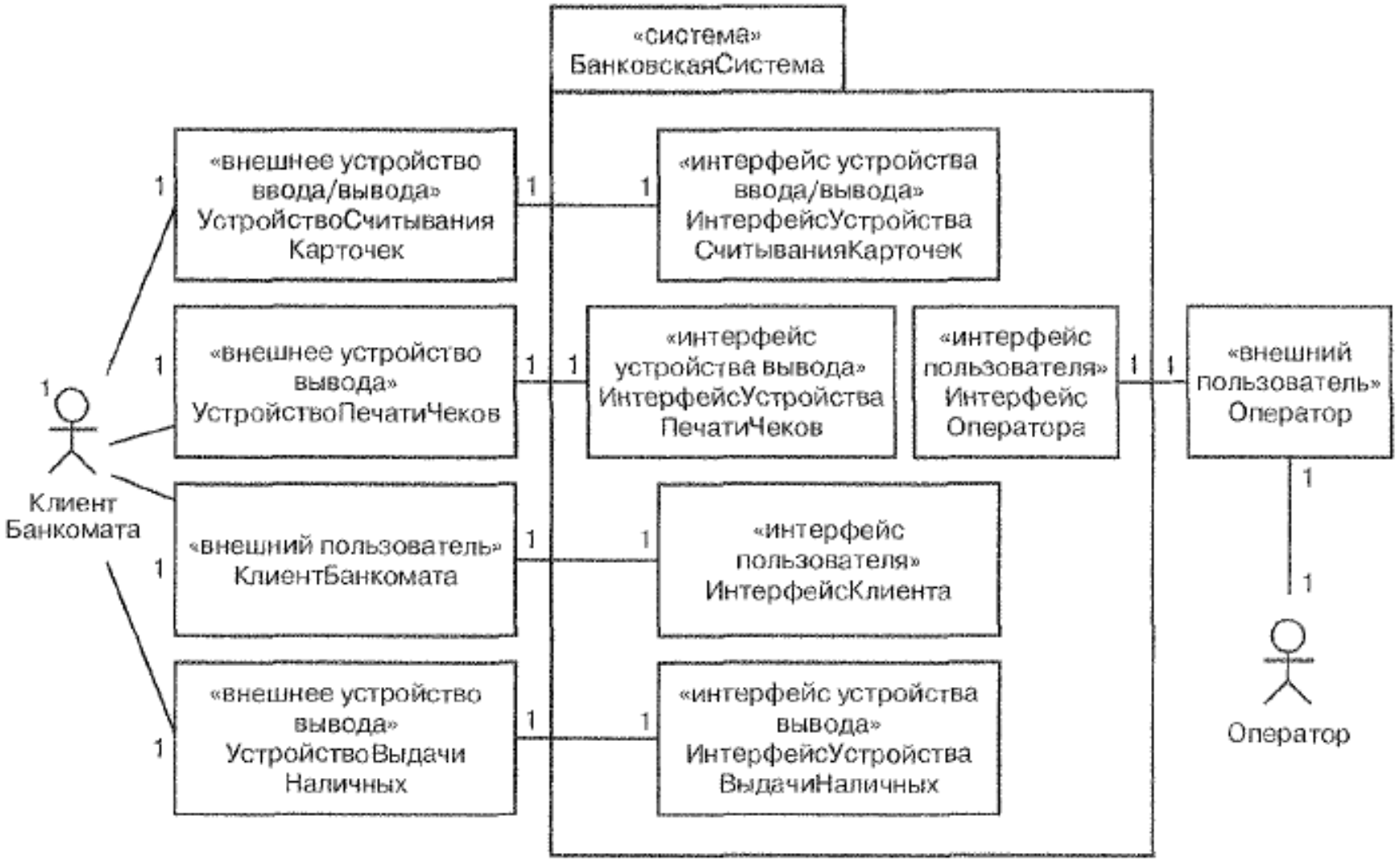


Класс-сущность

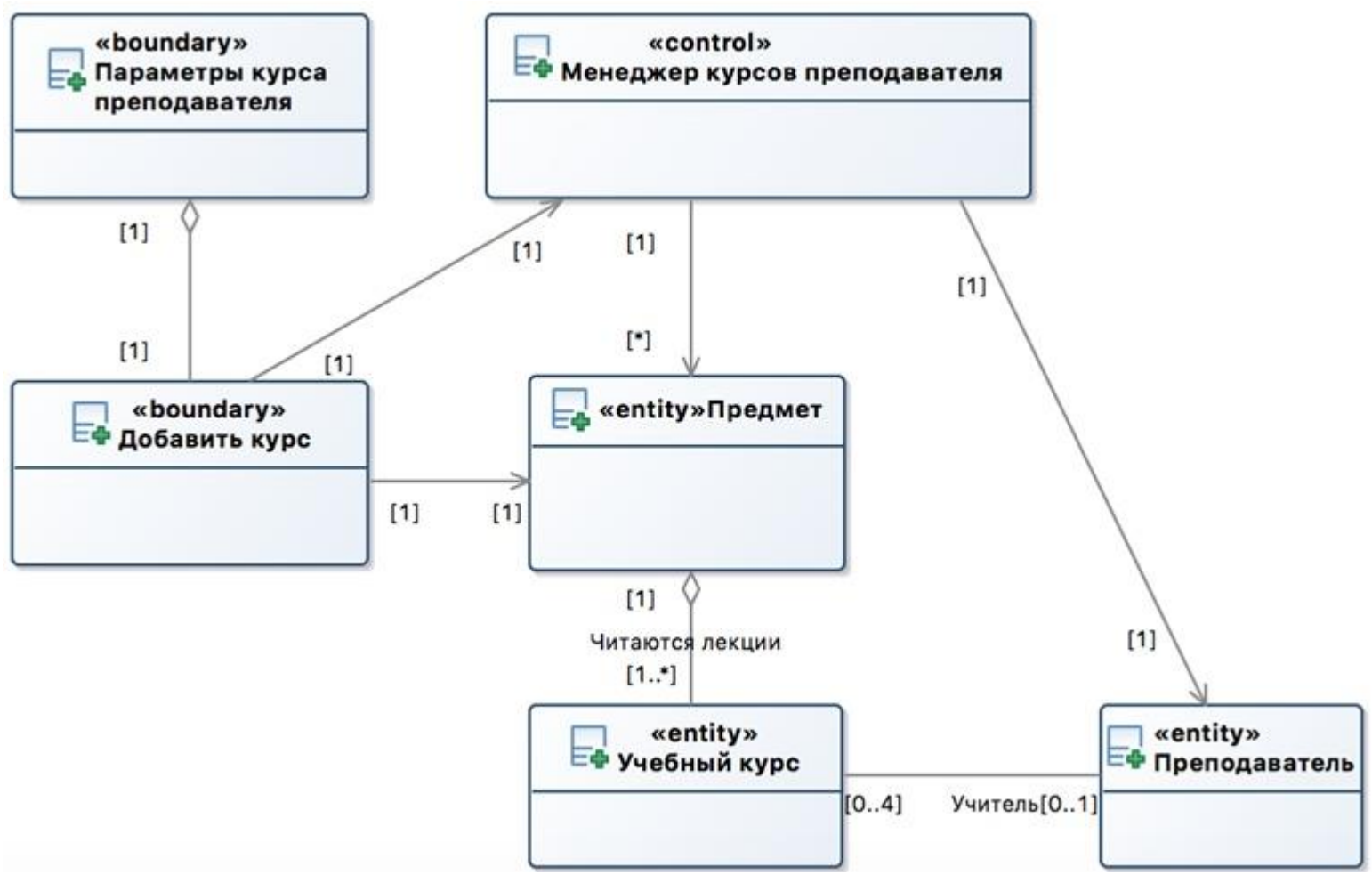




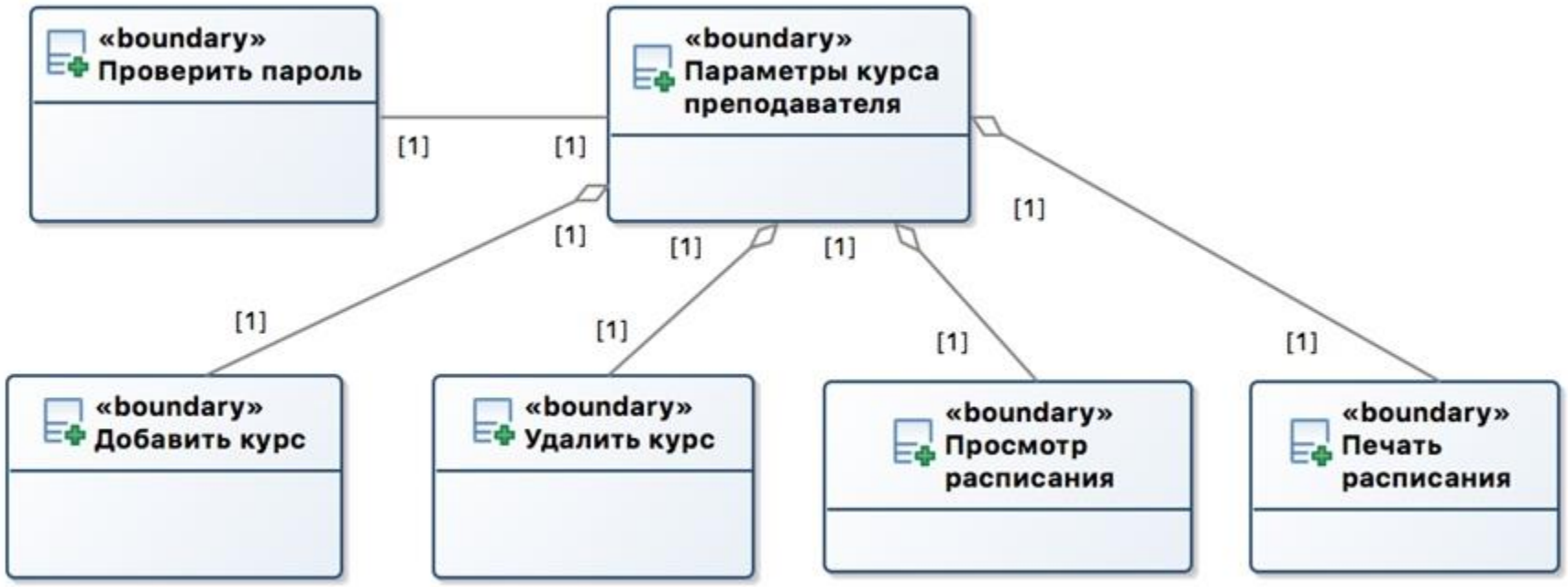
Классы анализа

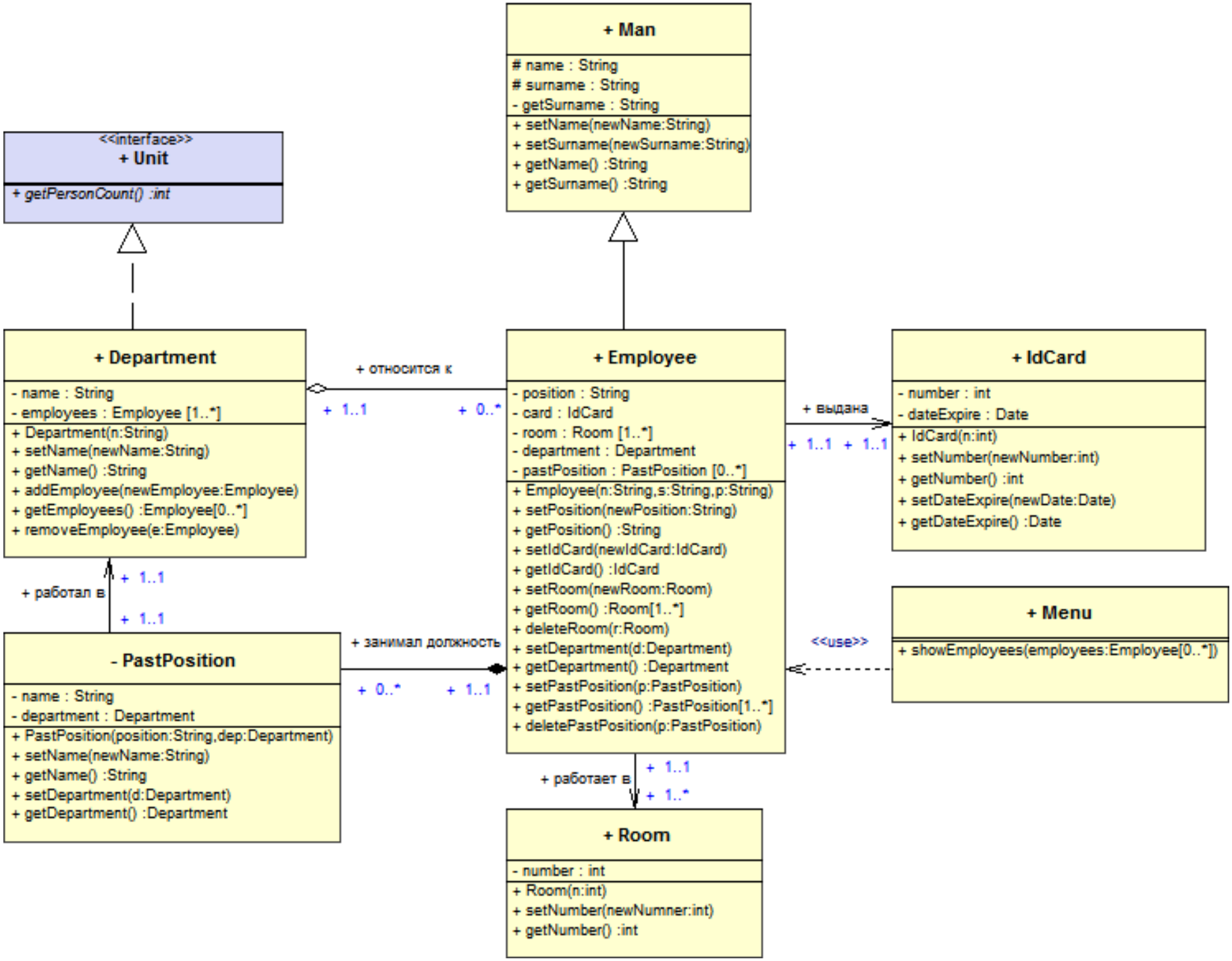


Анализ


«boundary»
Параметры курса преподавателя
«control»
Менеджер курсов преподавателя
[1]
[1]
[1]
[1]
[1]
[*]
«boundary»
Добавить курс
«entity»Предмет
[1]
[1]
[1]
Читаются лекции
[1..*]
[1]
«entity»
Учебный курс
«entity»
Преподаватель
[0..4]
Учитель[0..1]

Анализ


«boundary»
Проверить пароль
«boundary»
Параметры курса преподавателя
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
«boundary»
Добавить курс
«boundary»
Удалить курс
«boundary»
Просмотр расписания
«boundary»
Печать расписания


+ Man
# name : String
# surname : String
- getSurname : String
+ setName(newName:String)
+ setSurname(newSurname:String)
+ getName() :String
+ getSurname() :String
<<interface>>
+ Unit
+ getPersonCount() :int
+ Department
- name : String
- employees : Employee [1..*]
+ Department(n:String)
+ setName(newName:String)
+ getName() :String
+ addEmployee(newEmployee:Employee)
+ getEmployees() :Employee[0..*]
+ removeEmployee(e:Employee)
+ относится к
+ 1..1
+ 0..*
+ 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 : Date
+ IdCard(n:int)
+ setNumber(newNumber:int)
+ getNumber() :int
+ setDateExpire(newDate:Date)
+ getDateExpire() :Date
+ работал в
+ 1..1
+ 1..1
- PastPosition
- name : String
- department : Department
+ PastPosition(position:String,dep:Department)
+ setName(newName:String)
+ getName() :String
+ setDepartment(d:Department)
+ getDepartment() :Department
+ занимал должность
+ 0..*
+ 1..1
<<use>>
+ Menu
+ showEmployees(employees:Employee[0..*])
+ работает в
+ 1..1
+ 1..*
+ Room
- number : int
+ Room(n:int)
+ setNumber(newNumner:int)
+ getNumber() :int