Классы

- Классы - это самые важные строительные блоки любой объектно-ориентированной системы. Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс реализует один или несколько интерфейсов.

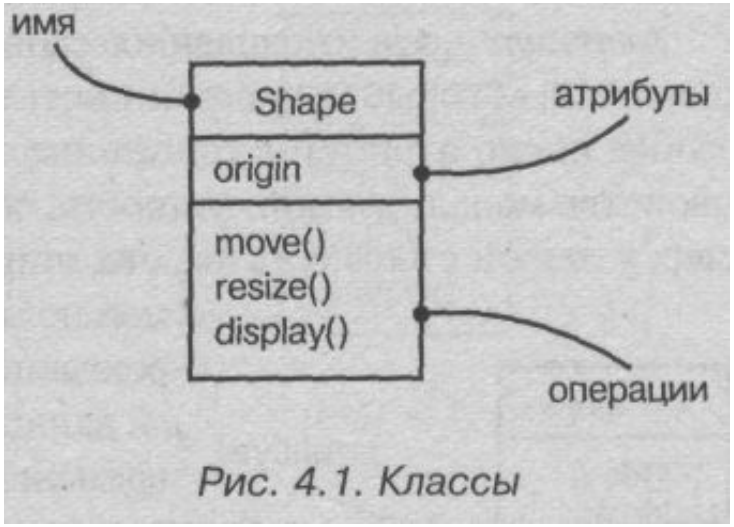
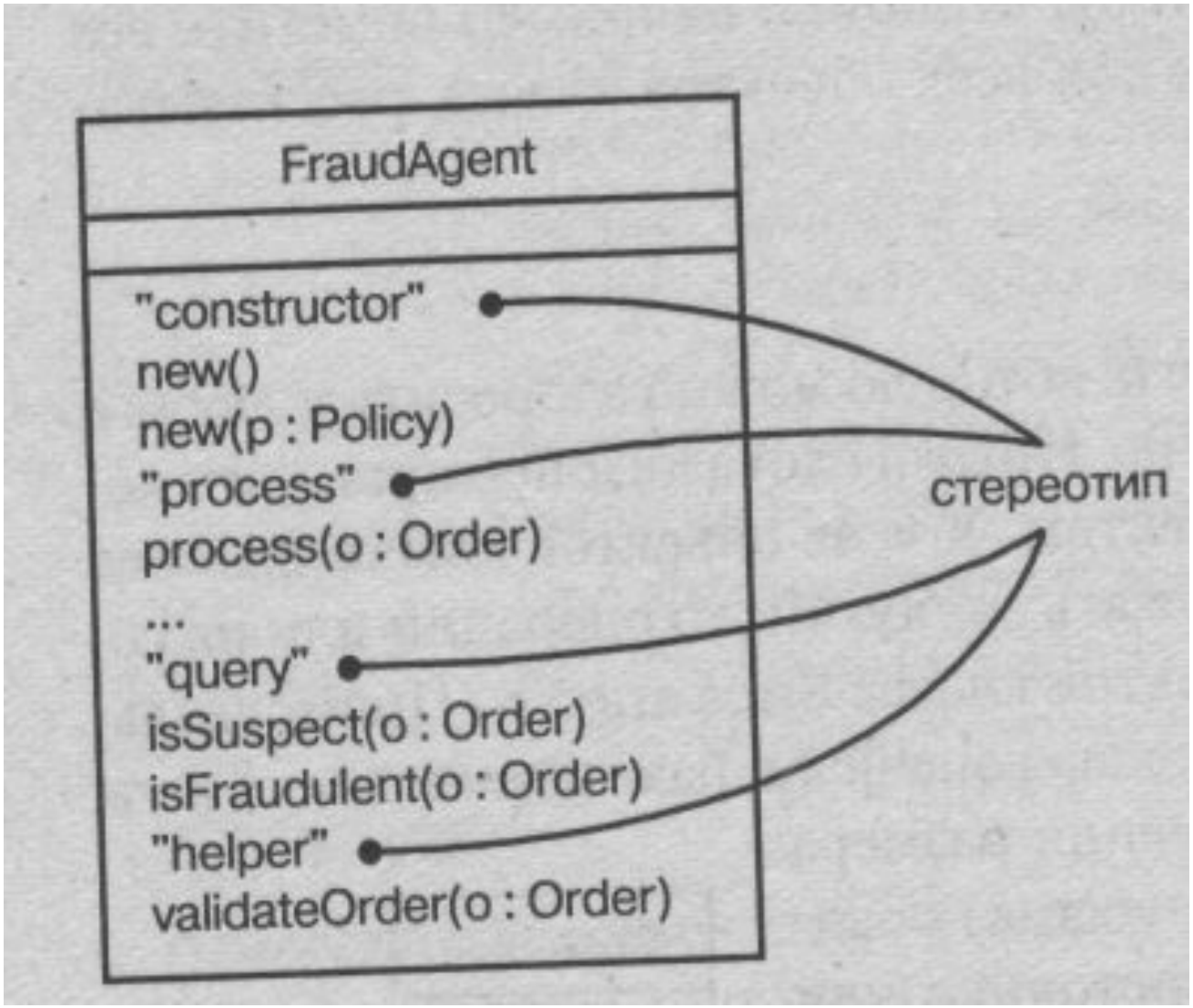


Рис. 4.1. Классы

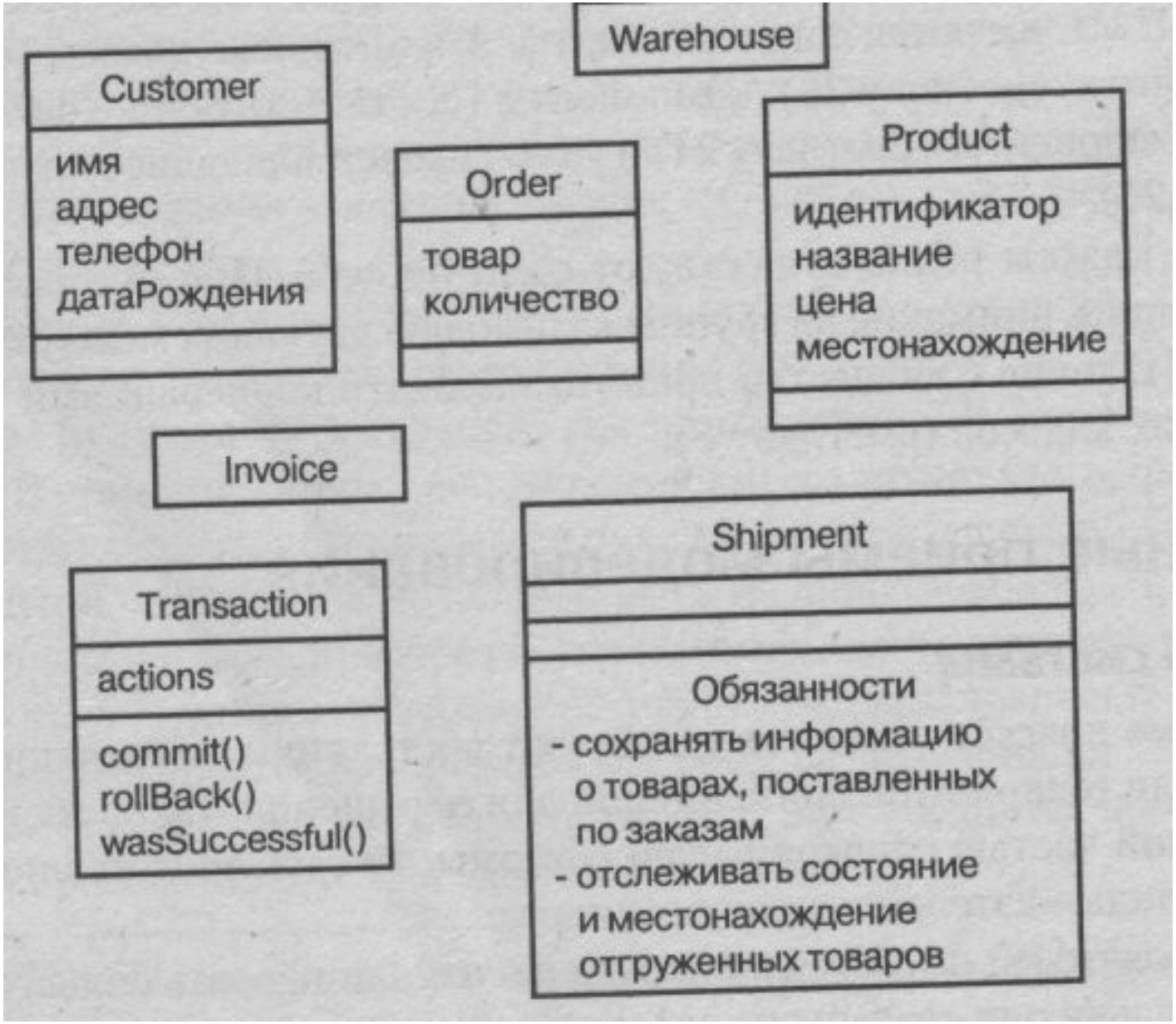
Организация атрибутов и операций

- При изображении класса необязательно сразу показывать все его атрибуты и операции. Как правило, это попросту невозможно - их чересчур много для одного рисунка, - да и не требуется (поскольку для данного представления системы лишь небольшое подмножество атрибутов и операций имеет значение).
- Для лучшей организации списков атрибутов и операций можно снабдить каждую группу дополнительным описанием, воспользовавшись стереотипами


FraudAgent
"constructor"
new()
new(p : Policy)
"process"
process(o : Order)
...
"query"
isSuspect(o : Order)
isFraudulent(o : Order)
"helper"
validateOrder(o : Order)
стереотип

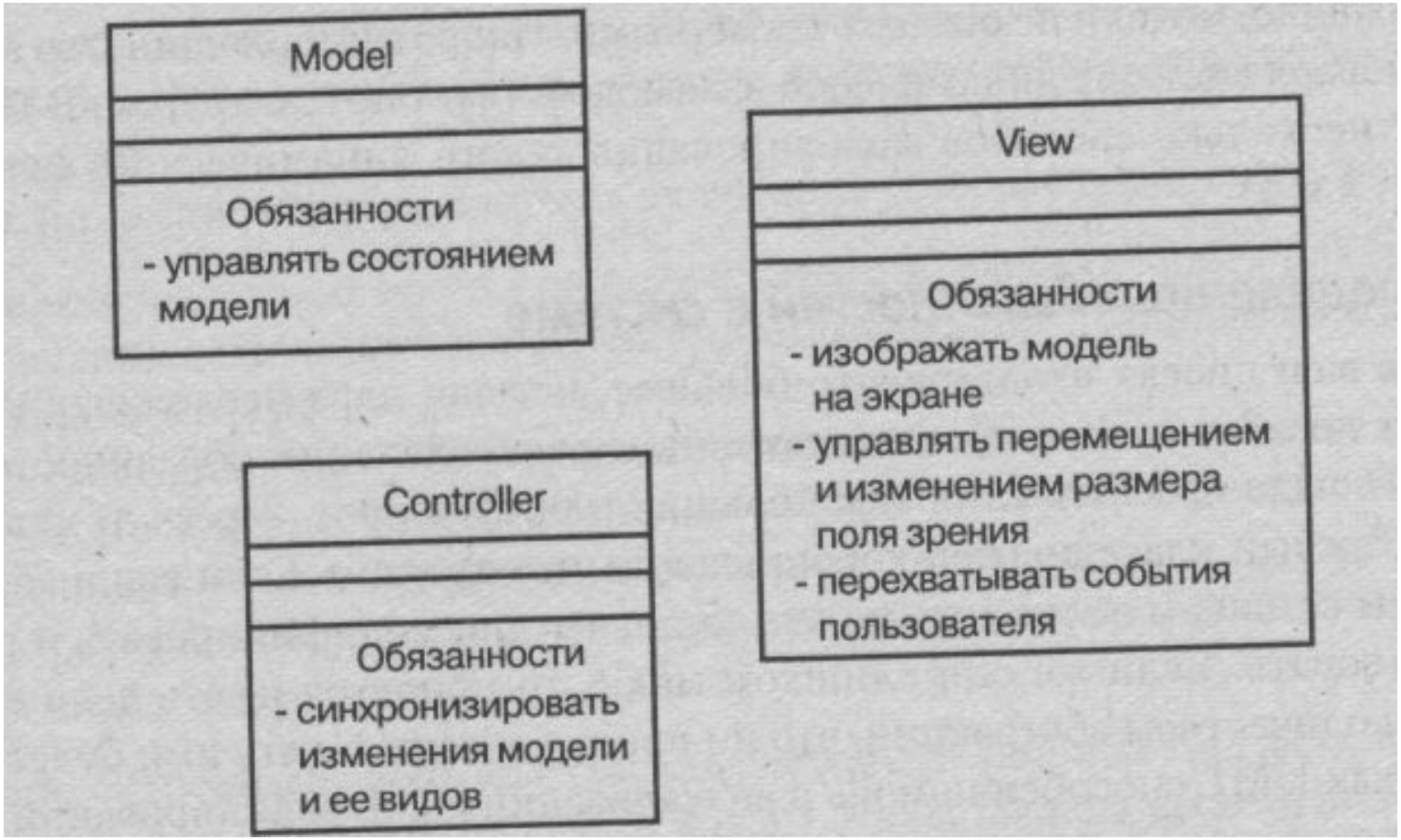
Обязанности

- Обязанности (Responsibilities) класса - это своего рода контракт, которому он должен подчиняться. Определяя класс, вы постулируете, что все его объекты имеют однотипное состояние и ведут себя одинаково.
- Обязанности оформляются в виде произвольно составленного текста. Как правило, каждая обязанность излагается в одном предложении или, на крайний случай, в коротком абзаце.


Warehouse
Customer
имя
адрес
телефон
датаРождения
Order
товар
количество
Product
идентификатор
название
цена
местонахождение
Invoice
Shipment
Transaction
actions
commit()
rollBack()
wasSuccessful()
Обязанности
- сохранять информацию о товарах, поставленных по заказам
- отслеживать состояние и местонахождение отгруженных товаров

Моделирование распределения обязанностей в системе включает в себя следующие этапы:

- 1. Идентифицируйте совокупность классов, совместно отвечающих за некоторое поведение.
- 2.Определите обязанности каждого класса.
- 3. Взгляните на полученные классы как на единое целое и разбейте те из них, у которых слишком много обязанностей, на меньшие - и наоборот, крошечные классы с элементарными обязанностями объедините в более крупные.
- 4. Перераспределите обязанности так, чтобы каждая абстракция стала в разумной степени автономной.
- 5. Рассмотрите, как классы кооперируются друг с другом и перераспределите обязанности с таким расчетом, чтобы ни один класс в рамках кооперации не делал слишком много или слишком мало.


Model
Обязанности
- управлять состоянием модели
View
Обязанности
- изображать модель на экране
- управлять перемещением и изменением размера поля зрения
- перехватывать события пользователя
Controller
Обязанности
- синхронизировать изменения модели и ее видов

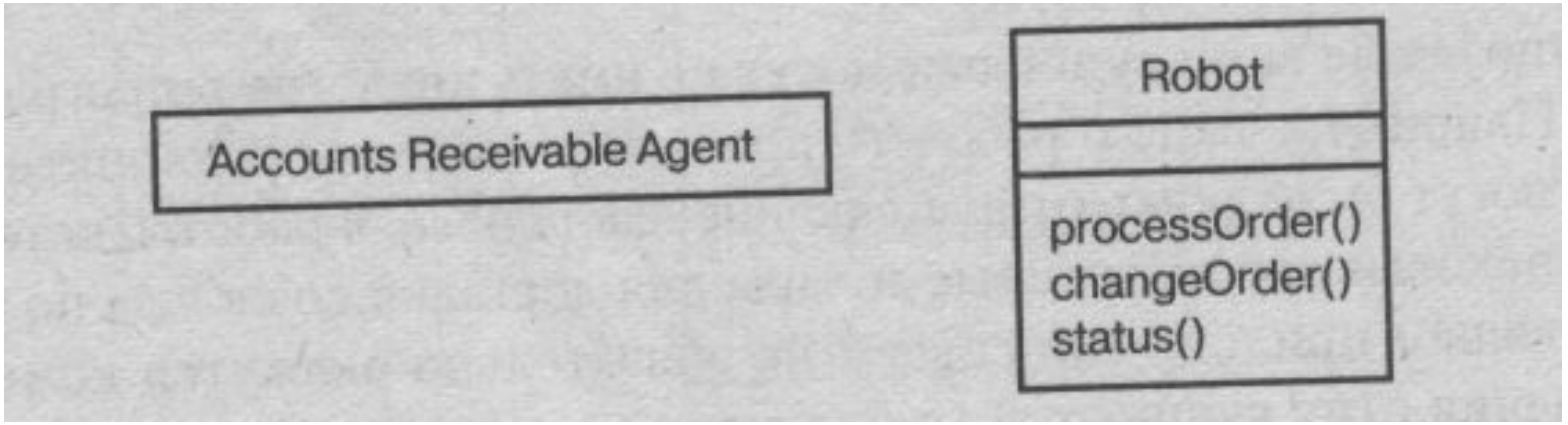
- В качестве примера на рис. показано распределение обязанностей между классами Model (Модель), View (Вид) и Controller (Контроллер) из совокупности классов языка Smalltalk. Как видите, их совместная работа организована так, что ни одному из классов не приходится делать слишком мало или слишком много. (Это множество классов формирует образец проектирования.)

Непрограммные сущности

- Моделируемые вами сущности могут не иметь аналогов в программном обеспечении. Например, частью рабочего процесса в модели предприятия розничной торговли могут быть люди, отправляющие накладные, и роботы, которые автоматически упаковывают заказанные товары...
- В вашем приложении совсем не обязательно окажутся компоненты для представления этих сущностей (в отличие от сущности «клиент», для хранения информации о которой, собственно, и создается система).

- UML предназначен в первую очередь для моделирования программных систем, его вполне допустимо использовать и для моделирования аппаратных систем.
- Абстрагирование людей (AccountsReceivableAgent -Агент По Дебиторской Задолженности) и аппаратуры (Robot) в виде классов вполне естественно, поскольку они представляют собой множества объектов с общей структурой и поведением. Сущности, внешние по отношению к системе, часто моделируются как актеры.

Непрограммные сущности

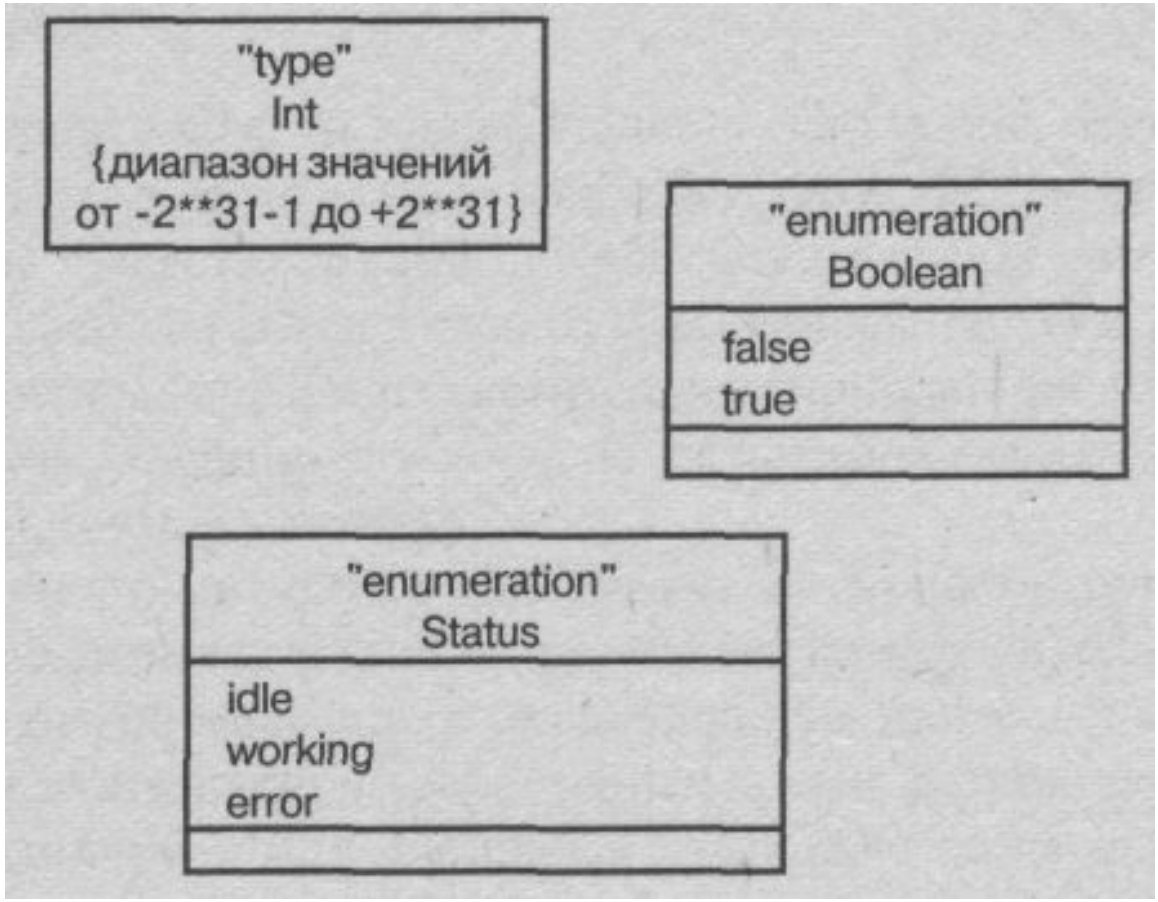


Моделирование непрограммных сущностей производится следующим образом:

- 1. Смоделируйте сущности, абстрагируемые в виде классов.
- 2. Если вы хотите отличить эти сущности от предопределенных строительных блоков UML, создайте с помощью стереотипов новый строительный блок, опишите его семантику и сопоставьте с ним ясный визуальный образ.
- 3. Если моделируемый элемент является аппаратным средством с собственным программным обеспечением, рассмотрите возможность смоделировать его в виде узла.

Примитивные типы

- Другой крайностью являются сущности, взятые непосредственно из языка программирования, используемого при решении задачи. Обычно к таким абстракциям относят примитивные типы, например целые, символы, строки и даже перечислимые типы, которые вы создаете сами.


"type"
Int
{диапазон значений
от -2**31-1 до +2**31}
"enumeration"
Boolean
false
true
"enumeration"
Status
idle
working
error

Моделирование примитивных типов производится следующим образом:

- 1. Смоделируйте сущность, абстрагируемую вами в виде типа или перечисления. Она изображается с помощью нотации класса с подходящим стереотипом.
- 2. Если требуется задать связанный с типом диапазон значений, воспользуйтесь ограничениями ({…}).

Хорошо структурированный класс обладает следующими свойствами:

- - является четко очерченной абстракцией некоторого понятия из словаря проблемной области или области решения;
- - содержит небольшой, точно определенный набор обязанностей и выполняет каждую из них;
- - поддерживает четкое разделение спецификаций абстракции и ее реализации;
- - понятен и прост, но в то же время допускает расширение и адаптацию к новым задачам

Изображая класс в UML, придерживайтесь следующих правил:

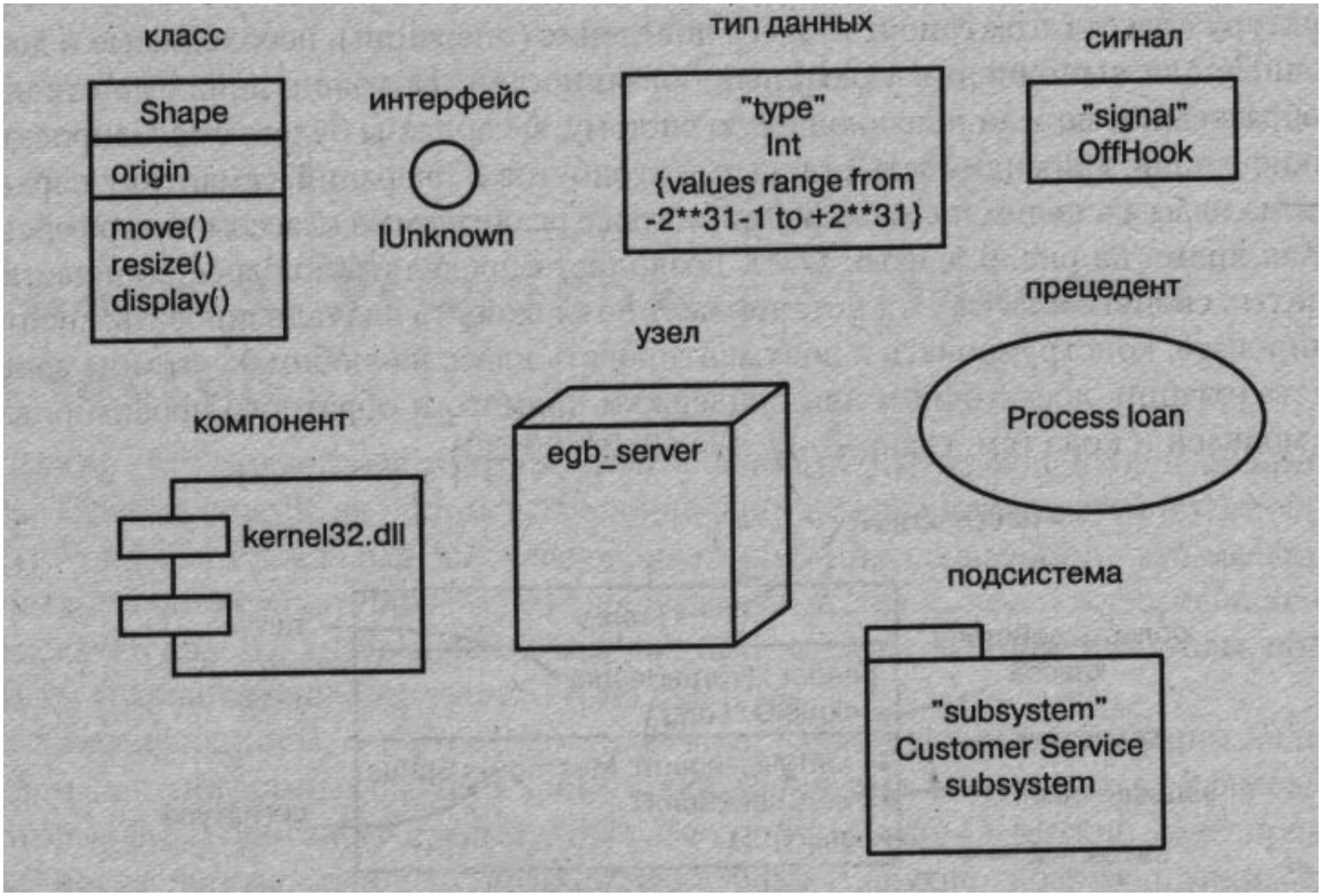
- - показывайте только те его свойства, которые важны для понимания абстракции в данном контексте;
- - разделяйте длинные списки атрибутов и операций на группы в соответствии с их категориями;
- - показывайте взаимосвязанные классы на одной и той же диаграмме.

Углубленное изучение классов

Классификатор

- Классификатор - это механизм, описывающий структурные и поведенческие свойства.
- К числу классификаторов относятся классы, интерфейсы, типы данных, сигналы, компоненты, узлы, прецеденты (варианты использования) и подсистемы.

- О сигнал - спецификация асинхронного стимула, используемого для связи между экземплярами.
- Подсистема — совокупность элементов, из которых отдельные составляют спецификацию поведения других элементов .


класс
Shape
origin
move()
resize()
display()
интерфейс
IUnknown
тип данных
"type"
Int
{values range from
-2**31-1 to +2**31}
сигнал
"signal"
OffHook
прецедент
Process loan
узел
egb_server
компонент
kernel32.dll
подсистема
"subsystem"
Customer Service
subsystem

Видимость (Visibility)

- + public (открытый) - любой внешний классификатор, который «видит» данный, может пользоваться его открытыми свойствами.
- # protected (защищенный) - любой потомок данного классификатора может пользоваться его защищенными свойствами.
- - private (закрытый) - только данный классификатор может пользоваться закрытыми свойствами.

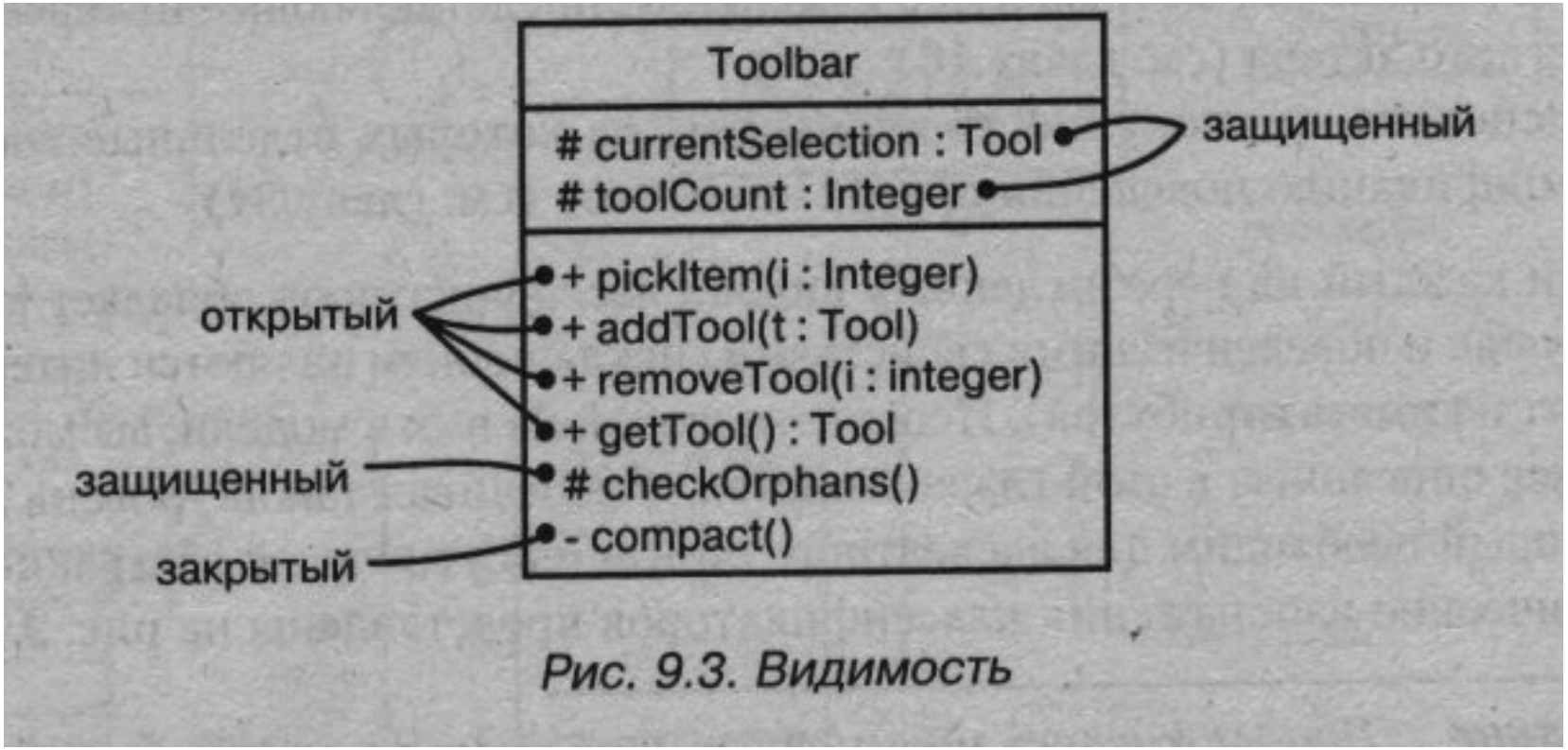
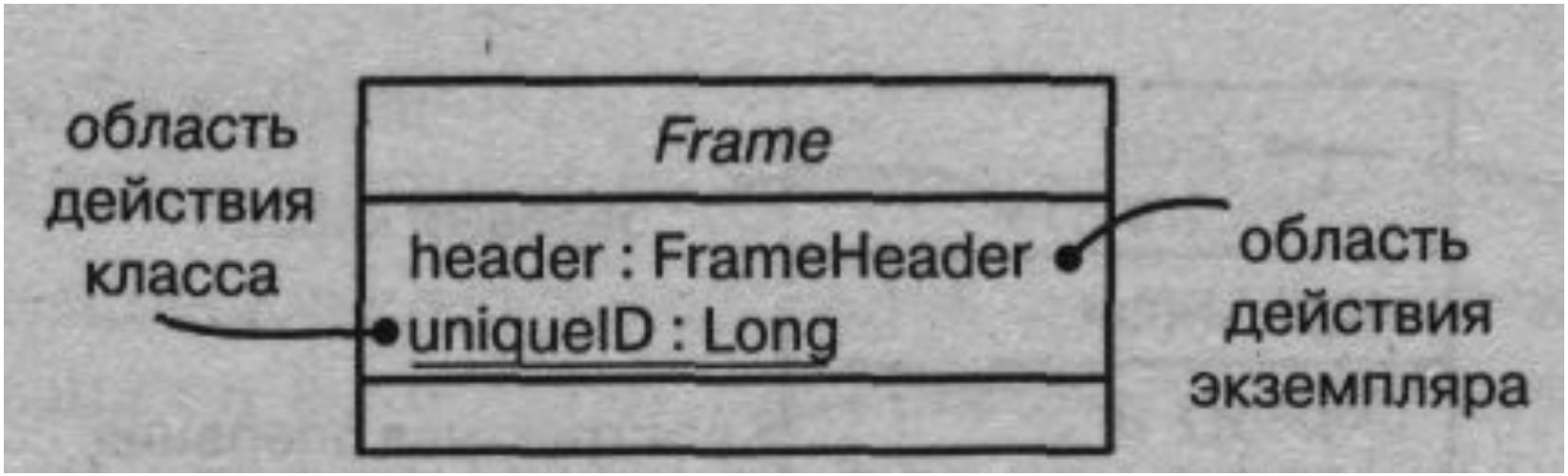


Рис. 9.3. Видимость

Область действия (Scope)

- **instance** (экземпляр) - у каждого экземпляра классификатора есть собственное значение данного свойства;
- **classifier** (классификатор) - все экземпляры классификатора совместно используют общее значение данного свойства.


область
действия
класса
Frame
header : FrameHeader
uniqueID : Long
область
действия
экземпляра

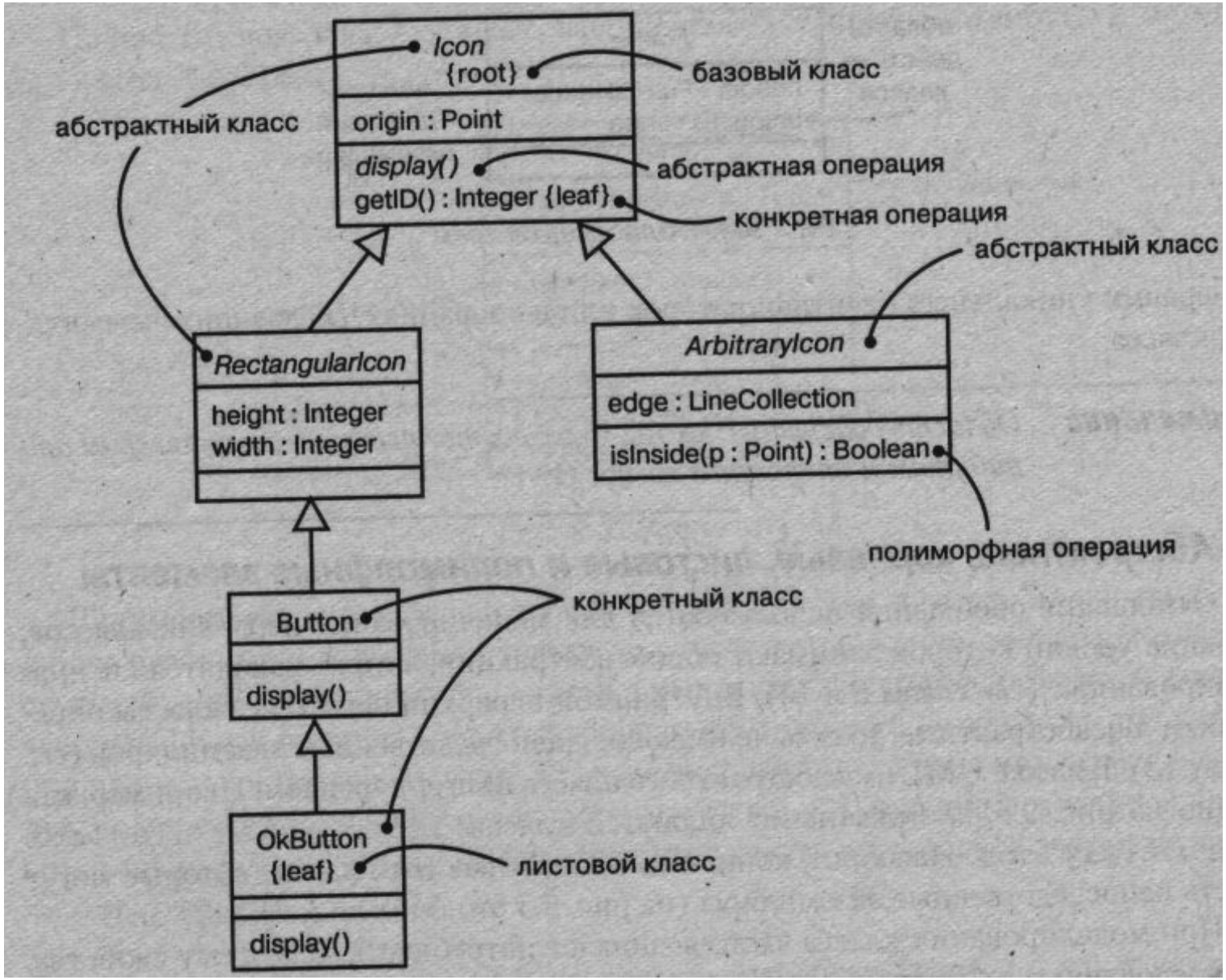
Абстрактные, корневые, листовые и полиморфные элементы

- Отношения обобщения используются для моделирования иерархии классов, верхние уровни которой занимают общие абстракции, а ниже находятся специализированные
- Внутри этой иерархии некоторые классы определяют как абстрактные, то есть не имеющие непосредственных экземпляров (*имя пишется курсивом*).

- Можно определить и такие классы, у которых нет потомков. Они называются листовыми и задаются в UML с помощью свойства {leaf}, написанного под именем класса.
- Можно явно обозначить вершину иерархии с помощью ограничения {root}

Операции могут иметь сходные свойства

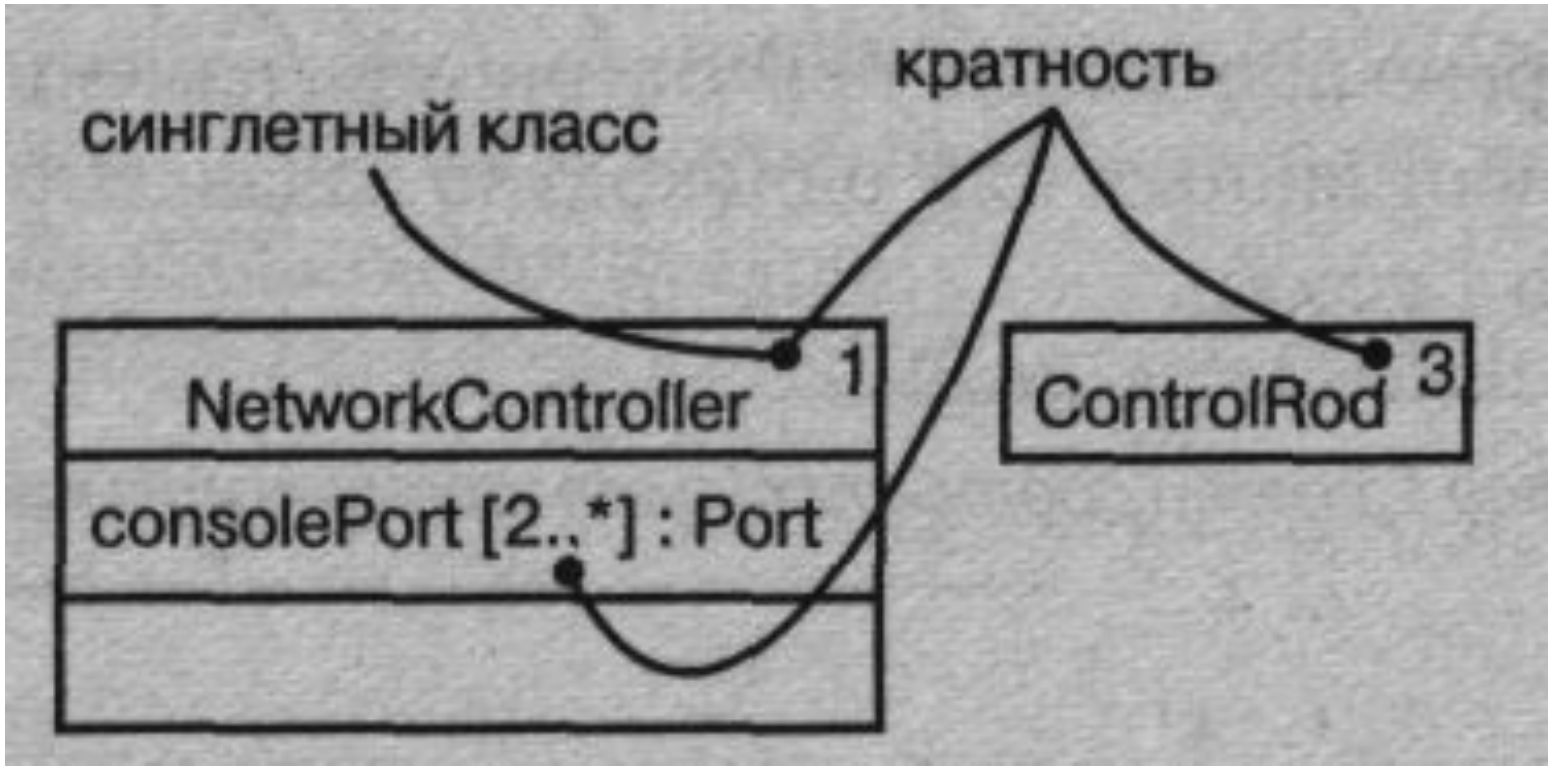
- Как правило, операции являются полиморфными, то есть могут переопределяться в потомках;
- листовые не могут переопределяться в потомках, помечаются ограничением {leaf};
- абстрактные операции неполны и определяются в потомках (*имя пишется курсивом*), виртуальные операции C++.


Icon
{root}
базовый класс
абстрактный класс
origin : Point
display()
абстрактная операция
getID() : Integer {leaf}
конкретная операция
абстрактный класс
RectangularIcon
height : Integer
width : Integer
ArbitraryIcon
edge : LineCollection
isInside(p : Point) : Boolean
полиморфная операция
Button
конкретный класс
display()
OkButton
{leaf}
листовой класс
display()

Кратность (Multiplicity)

- нет ни одного экземпляра - тогда класс становится служебным (Utility), содержащим только атрибуты и операции с областью действия класса;
- ровно один экземпляр - такой класс называют синглетным (Singleton);
- заданное число экземпляров;
- произвольное число экземпляров - вариант по умолчанию.

Кратность


кратность
синглетный класс
NetworkController
1
consolePort [2..*] : Port
ControlRod
3

Атрибуты

Полная форма синтаксиса атрибута в языке UML следующая:

[visibility] name [multiplicity] [: type]

[= initial_value] [{property_string}]

Наряду с {leaf} с атрибутами можно использовать три свойства:

{changeable} (изменяемый) - ограничений на изменение значений атрибута не установлено;

{addOnly} (только добавляемый) - разрешается добавлять новые значения для атрибутов с кратностью больше единицы, но созданное значение не может быть изменено или удалено;

{frozen} (замороженный) - после инициализации объекта нельзя изменять значения его атрибутов (соотв. **const**)

Ниже приводятся примеры допустимых объявлений атрибутов:

- origin - только имя;
- + origin - видимость и имя;
- origin : Point - имя и тип;
- head : *Item - имя и сложный тип;
- name [0..1] : String - имя, кратность и тип;
- origin : Point = (0,0)- имя, тип и начальное значение;
- id : Integer {frozen} - имя и свойство.

Операции

- Полный синтаксис операции в языке UML таков:
- [visibility] name [(parameter-list)] [: return-type] [{property_string}]

Сигнатура операции может содержать ноль или более параметров, каждый из которых имеет следующий синтаксис:

- [direction] name : type [= default-value]
- Параметр direct ion может принимать любое из ниже перечисленных значений:
- in - входящий параметр, который не может быть модифицирован;
- out - выходящий параметр, который может быть изменен, чтобы передать информацию вызвавшей процедуре;
- inout - входящий параметр, который может быть изменен.

Помимо описанного ранее свойства leaf для операций определены еще четыре свойства:

- {isQuery} (запрос) - выполнение операции не изменяет состояния системы. Другими словами, операция является просто функцией без побочных эффектов;
- {sequential} (последовательная) - при вызове операции необходимо гарантировать, что в любой момент объект выполняет только один поток;
- {guarded} (охраняемая) - семантика и целостность объекта при наличии нескольких потоков управления гарантируются упорядочением всех обращений к его охраняемым операциям;
- {concurrent} (параллельная) - семантика и целостность объекта при наличии нескольких потоков управления гарантируются благодаря тому, что операция рассматривается как атомарная.

Примеры:

- display - только имя;
- + display - видимость и имя;
- set (n: Name, s: String) - имя и параметры;
- getID () : Integer - имя и возвращаемое значение;
- restart () {guarded} - имя и свойство.

Шаблоны классов

- Шаблоном называется параметризованный элемент. В таких языках программирования, как C++ или Ada, предусмотрена возможность создавать шаблоны классов, определяющие семейства классов (можно задавать также шаблоны функций, определяющие семейства функций).
- Параметрами шаблона могут быть классы, объекты или значения.

Шаблон (Template)

```
Template <class Item, class Value, int Buckets>
    class Map {
public:
virtual Boolean bind (const Item&, const Value&) ;
virtual Boolean isBound (const Item&);
};
```

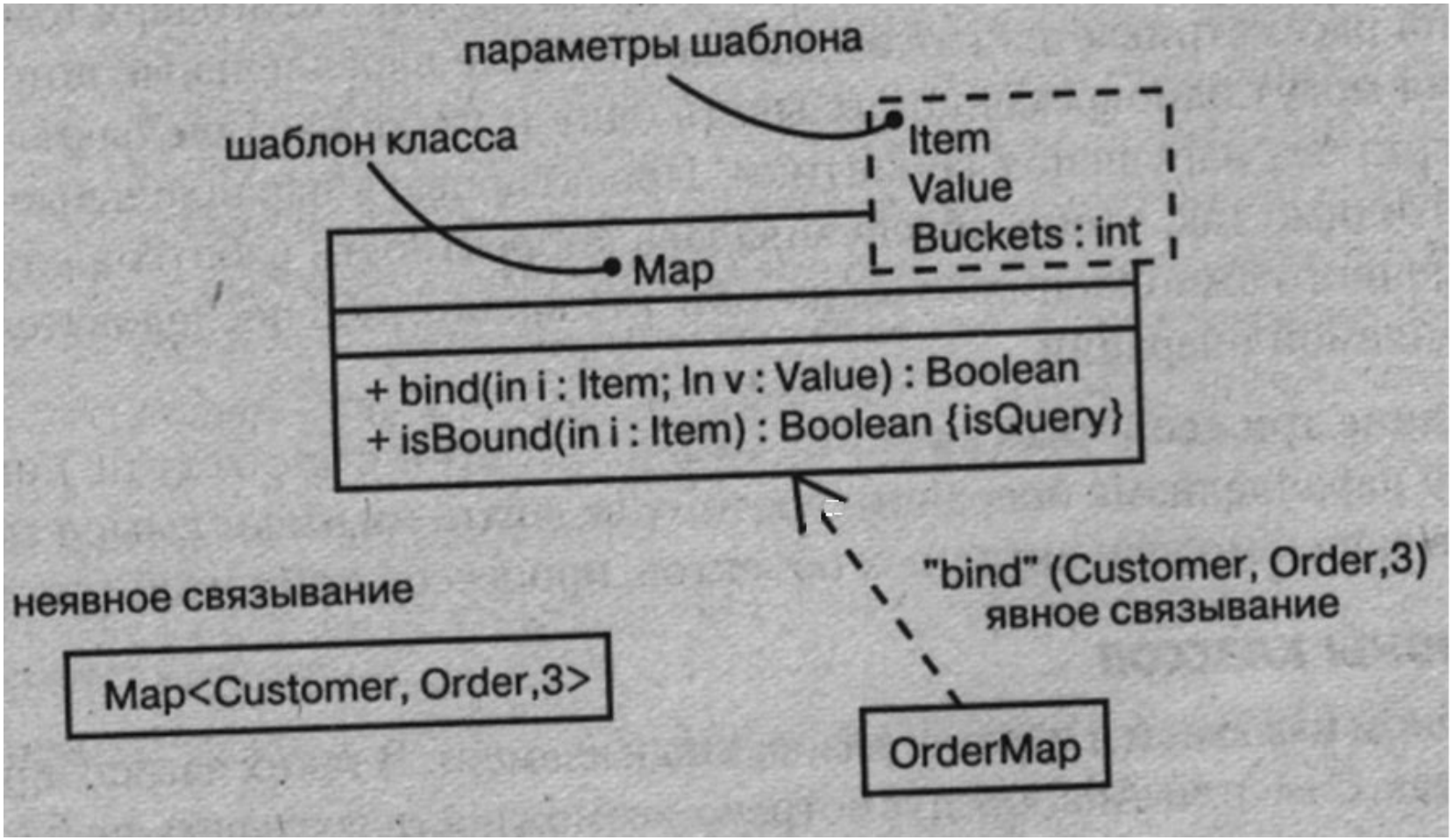
в вышеприведенном фрагменте кода на языке C++ объявляется параметризованный класс Map (Карта).

- Шаблон нельзя использовать непосредственно; сначала его нужно инстанцировать, то есть конкретизировать.
- Процесс инстанцирования - это связывание формальных параметров шаблона с фактическими. В результате из шаблона класса получается конкретный класс, с которым можно работать как с любым другим.

- можно инстанцировать этот шаблон Map для отображения объектов класса Customer (Клиент) на объекты класса Order (Заказ):

Map<Customer, Order, 3>;

- Язык UML также позволяет моделировать шаблоны классов, такой класс изображается в точности как обычный, но в верхнем правом углу его пиктограммы находится дополнительная ячейка, нарисованная пунктиром; в ней перечислены параметры шаблона.
- На данном рисунке показано также, что моделировать инстанцирование шаблона класса можно двумя способами.
  - Во-первых - неявно, для чего требуется объявить класс, имя которого обеспечивает связывание.
  - Во-вторых, можно явным образом определить зависимость со стереотипом bind, показывающую, что источник инстанцирует целевой шаблон с заданными фактическими параметрами.


параметры шаблона
шаблон класса
Item
Value
Buckets : int
Map
+ bind(in i : Item; In v : Value) : Boolean
+ isBound(in i : Item) : Boolean {isQuery}
"bind" (Customer, Order,3)
явное связывание
неявное связывание
Map<Customer, Order,3>
OrderMap

Стандартные элементы

В UML определены четыре стандартных стереотипа, применимые к классам:

- «metaclass» - определяет классификатор, все объекты которого являются классами;
- «powertype» - определяет классификатор, все объекты которого являются потомками данного родителя;
- «stereotype» - определяет, что данный классификатор является стереотипом, который можно применить к другим элементам;
- «utility» - определяет класс, атрибуты и операции которого находятся в области'действия всех классов.

Для моделирования семантики класса можно воспользоваться любыми из перечисленных ниже возможностей (они расположены в порядке возрастания формализации):

1. Определите обязанности класса. Обязанностью называется контракт или обязательство, «подписываемое» типом или классом.
2. Опишите семантику класса в целом с помощью структурированного текста, который изображается в виде примечания (со стереотипом semantics), присоединенного к классу.
3. Определите тело каждого метода с помощью структурированного текста или языка программирования и поместите определение в примечание, присоединенное к операции отношением зависимости.
4. Специфицируйте пред- и постусловия каждой операции а также инварианты класса в целом с помощью структурированного текста. Эти элементы изображаются в виде примечаний (со стереотипами precondition, postcondition и invariant), присоединенных к операции или классу отношениями зависимости.
5. Определите автомат класса.
6. Специфицируйте представляющие класс кооперации. Поскольку у кооперации имеются структурная и динамическая составляющие, вы можете с их помощью специфицировать все аспекты семантики класса.
7. Определите пред- и постусловия каждой операции и инварианты класса в целом с помощью такого формализованного языка, как OCL (язык OCL обсуждается в книге "The Unified Modeling Language Reference Manual").

Советы

Моделируя классификаторы в языке UML, помните, что в вашем распоряжении имеется множество строительных блоков - интерфейсы, классы, компоненты и т.д. Из них вы должны выбрать тот, который наилучшим образом соответствует вашей абстракции. Хорошо структурированный классификатор обладает следующими свойствами:

- наделен как структурными, так и поведенческими аспектами;
- а внутренне согласован и слабо связан с другими классификаторами;
- раскрывает только те особенности, которые необходимы для использующих класс клиентов, и скрывает остальные;
- его семантика и назначение не допускают неоднозначного толкования;
- не настолько формализован, чтобы лишить всякой свободы тех, кто будет его реализовывать;
- специфицирован в достаточной степени, чтобы исключить неоднозначное толкование его назначения.

Изображая классификатор в UML, примите во внимание следующие рекомендации:

- показывайте только те его свойства, которые необходимы для понимания абстракции в контексте класса;
- используйте такие стереотипы, которые наилучшим образом отражают назначение классификатора.