# Курс
# «Средства и технологии анализа и разработки информационных систем (СиТАиРИС)»

**1. Объектно-ориентированный анализ и моделирование, разработка статических моделей**

**2. Объектно-ориентированное моделирование, разработка динамических моделей**

**3. Шаблоны проектирования**

**4. Эффективная передача данных между программными системами**

**5. Средства совместной разработки программных проектов**

**6. Автоматизация сборки проектов**

**7. Создание качественного программного кода, JUNIT**

**1. Объектно-ориентированный анализ и моделирование, разработка статических моделей**

**Язык UML и инструментальные средства разработки моделей программных систем. Общая характеристика особенностей и применения UML-моделей для разработки архитектуры программного продукта. Разработка:**

**- диаграммы вариантов использования (Use cases);**

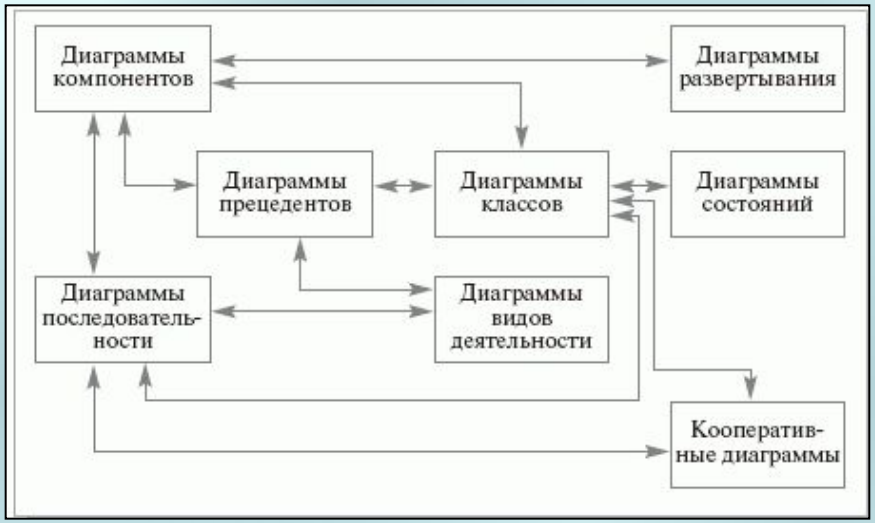
**- диаграмма классов;**

**- диаграмма последовательности.**

**Программирование диаграммы классов на языке Java**


# Объектно-ориентированный подход к проектированию информационных систем. Диаграммы UML

## Диаграмма классов (Class Diagram)

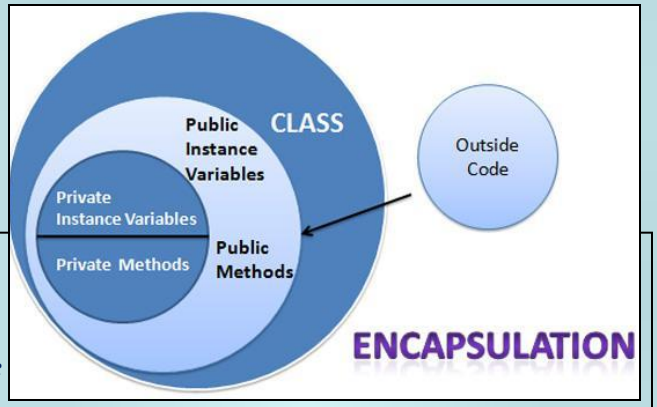



# ООП в проектировании ИС

В ООП центральным понятием является объект. Объект – это экземпляр класса.

Объекты и классы организуются с использованием следующих принципов:

- Инкапсуляция (скрытие информации) декларирует запрет любого доступа к атрибутам объекта, кроме как через его открытые методы. В соответствии с этим внутренняя структура объекта скрыта от пользователя, а любое его действие инициируется внешним сообщением, вызывающим выполнение соответствующей операции (опции).




**C++**

```
class A {
public: int a, b;//данные открытого интерфейса
int ReturnSomething();//метод открытого интерфейса
private: int Aa, Ab;//скрытые данные
void DoSomething();//скрытый метод
};
```

**Java**

```
class A {
private int a;
private int b;
private void doSomething()//скрытый метод
{ //actions }
public int returnSomething(){}//открытый интерфейс
 }
```

# ООП в проектировании ИС

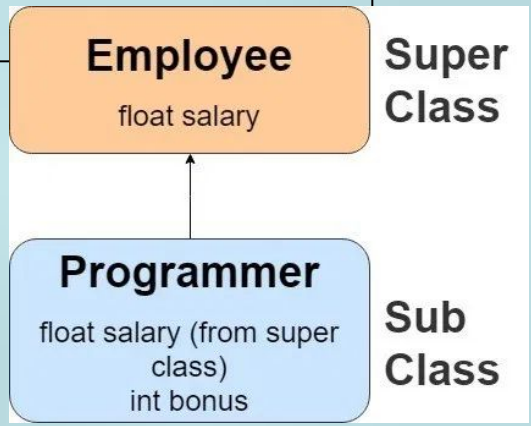
- Наследование декларирует создание новых классов от общего к частному. Такие новые классы сохраняют все свойства классов-родителей и при этом содержат дополнительные атрибуты и операции, характеризующие их специфику.

**C++ :**

```
class A
{ //базовый класс };
class B : public A
{ //public наследование };
class C : protected A
{ //protected наследование };
class Z : private A
{ //private наследование };
```

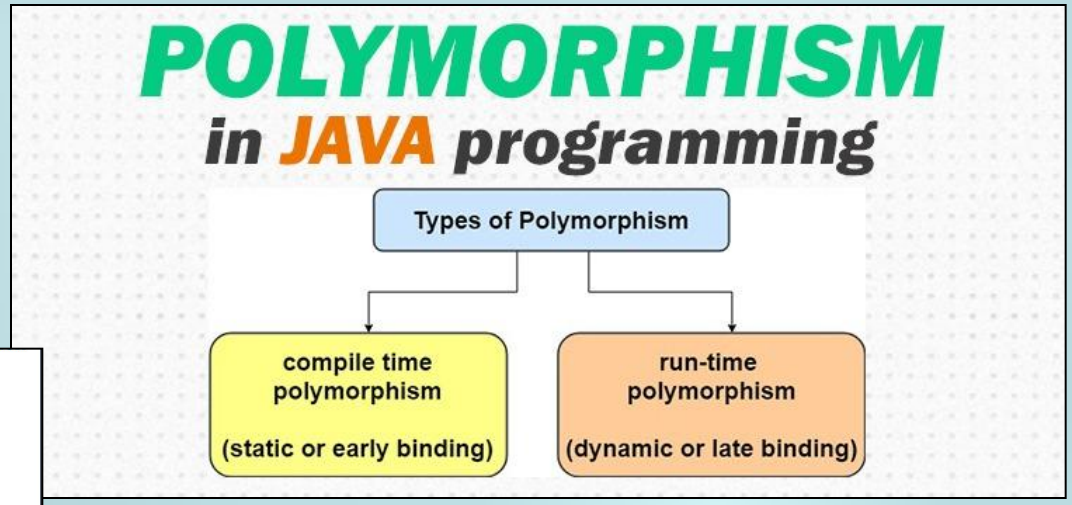
**Java :**

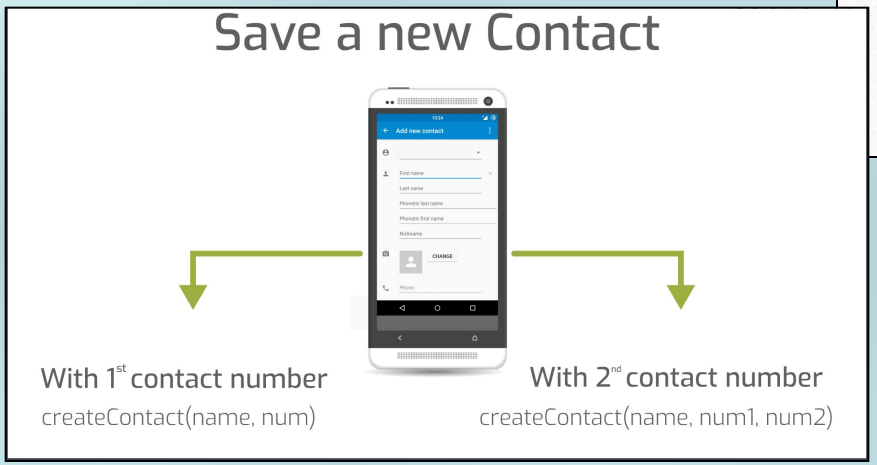
```
class A
{ //базовый класс };
class B extends A
{ //производный класс B };
class C  extends A
{ //производный класс C };
class Z extends A
{ //производный класс Z };
```



# ООП в проектировании ИС

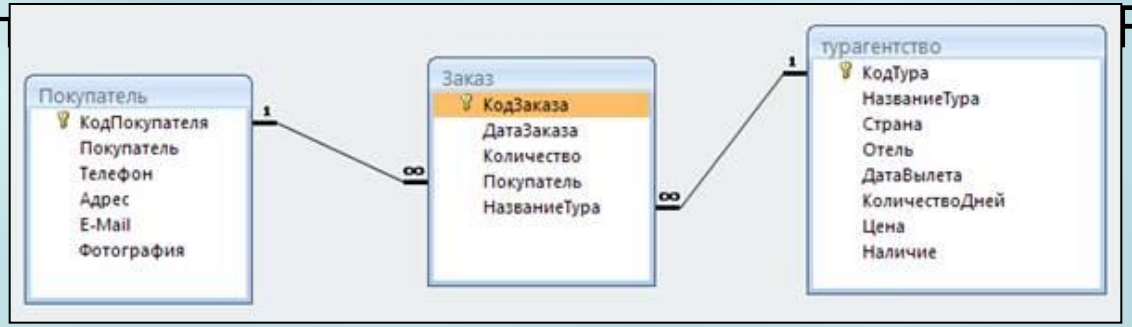
- Полиморфизм декларирует возможность объектов с одинаковой спецификацией иметь различную реализацию (или возможность работы с объектом без информации о конкретном классе, экземпляром которого он является. Каждый объект может выбирать операцию на основании типов данных, принимаемых в сообщении, т.е. реагировать индивидуально на это (одно и то же для различных объектов) сообщение).



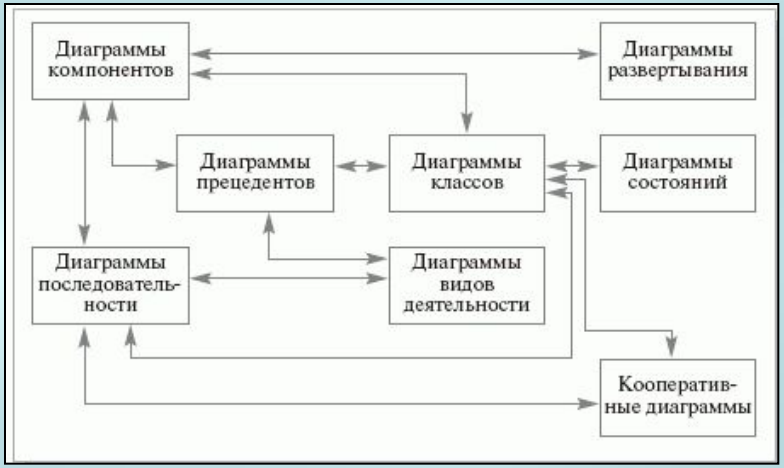
# Методология объектного проектирования средствами UML

- Унифицированный язык моделирования (Unified Modeling Language, UML) - язык для специфицирования, визуализации, конструирования и документирования на основе объектно-ориентированного подхода разных видов систем: программных, аппаратных, программно-аппаратных, смешанных и т. д.
- Помимо прочего, язык UML применяется для проектирования реляционных БД. Для этого используется небольшая часть языка (диаграммы классов), да и то не в полном объеме. С точки зрения проектирования реляционных БД модельные возможности не сл R-диаграмм



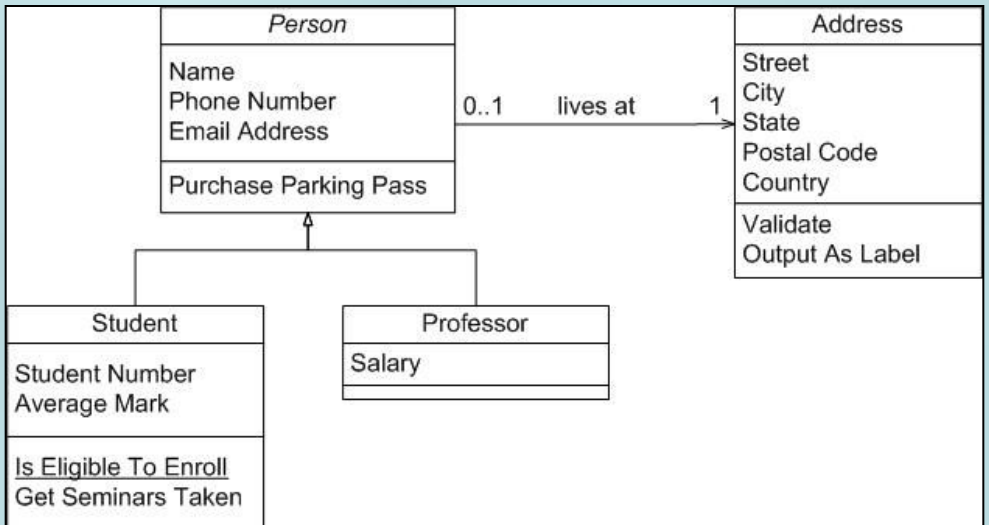
# Диаграммы UML

- компонентов,
- вариантов использования,
- развертывания,
- взаимодействия,
- состояний,
- последовательности действий.



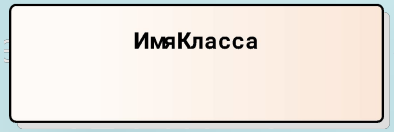
# Диаграмма классов

- Является центральным звеном объектно-ориентированного подхода
- Содержит информацию об объектах системы и статических связях между объектами
- Отражает *декларативные знания* о предметной области
- Оперирует понятиями *класса*, *объекта*, *отношения, пакета*

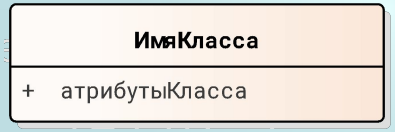


# Класс

- ***Класс*** – это множество объектов, которые обладают *одинаковой* структурой, поведением и отношениями с объектами из других классов.

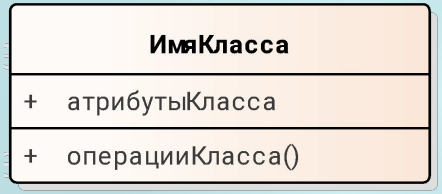



Простейший вид класса состоит только из секции имени



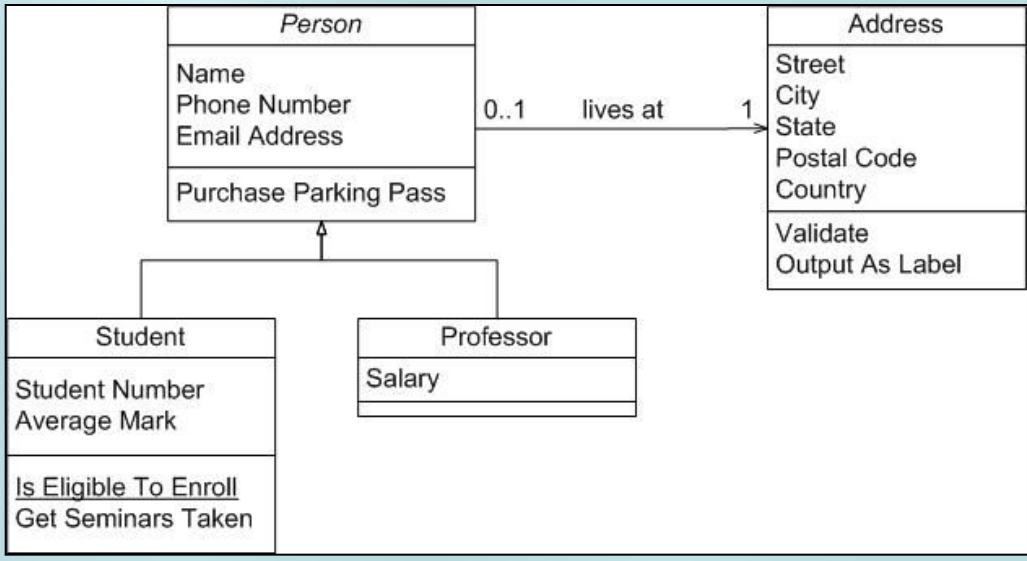

Класс с указанием атрибутов (переменных)

Полное описание класса, состоящее из 3 разделов – секции имени, секции атрибутов, секции операций



# Класс

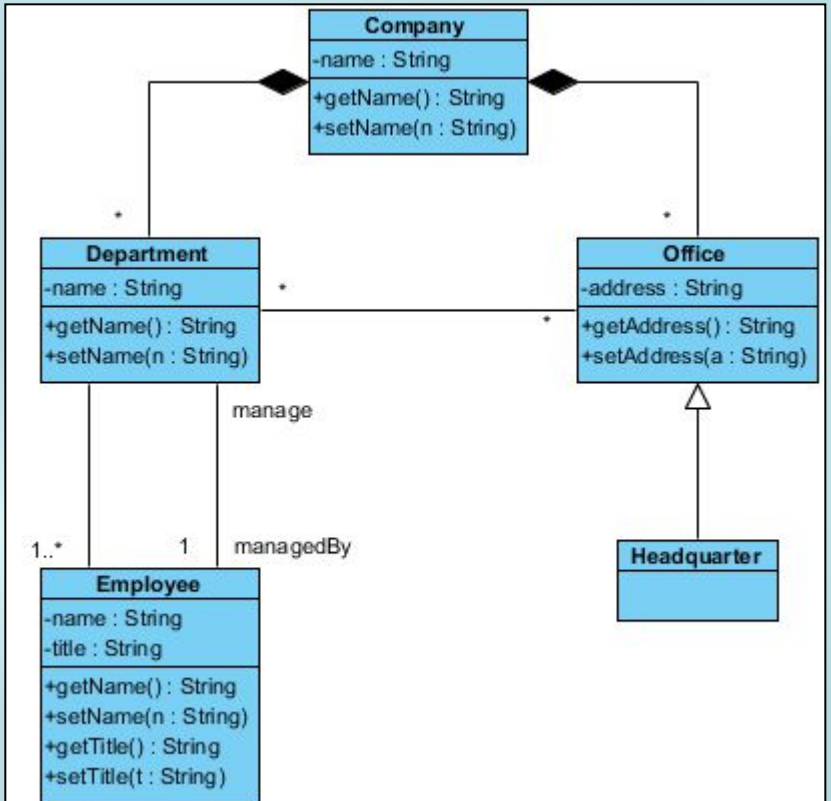
- ***Имя класса*** должно быть уникально и начинаться с заглавной буквы.
- Класс может не иметь экземпляров или объектов. В этом случае он называется **абстрактным классом**, а для обозначения его имени используется *курсив.*



# Атрибуты класса

- **Атрибут** = **свойство**, которое является общим для всех объектов данного класса
- Общий формат записи атрибутов:

*<квантор видимости> <имя атрибута> [кратность]: <тип атрибута> = <исходное значение> {строка-свойство}*



# Атрибуты класса. Квантор видимости атрибута

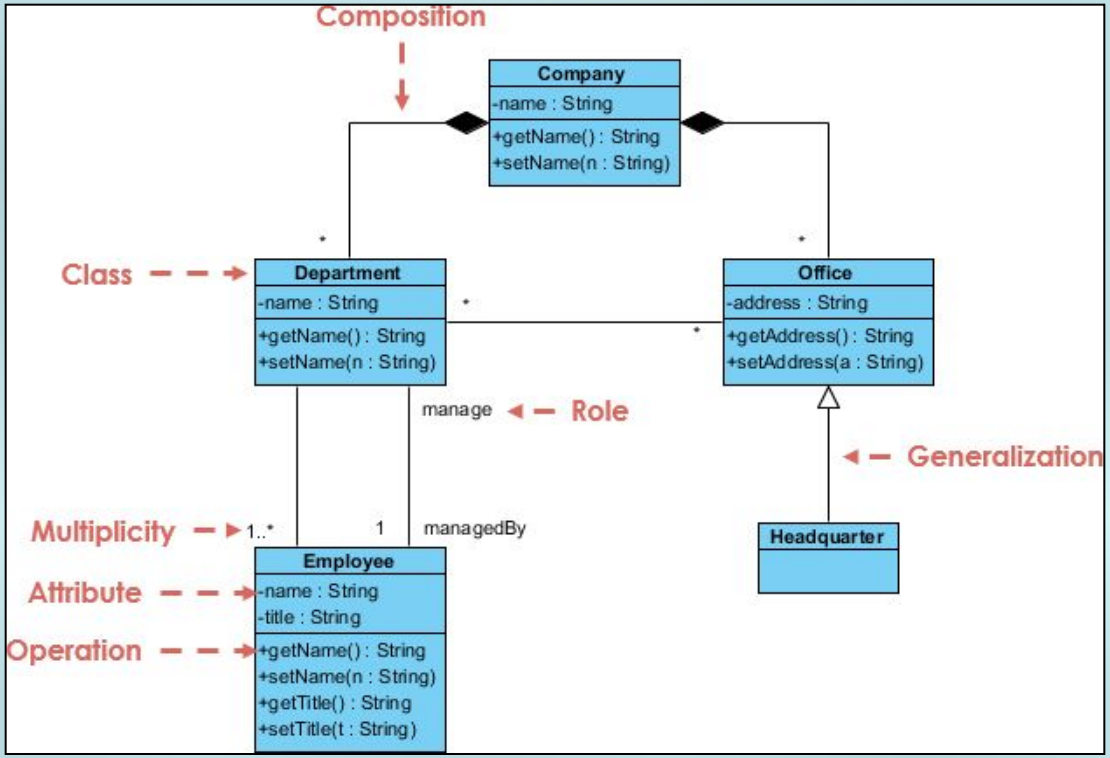
***<квантор видимости>*** *<имя атрибута> [кратность]: <тип атрибута> = <исходное значение> {строка-свойство}*

- Квантор видимости может принимать одно из следующих значений: **+**, **#**, **-** , **~**.
- «**+**» – атрибут с областью видимости типа ***общедоступный*** (public).
- «**#**» – атрибут с областью видимости типа ***защищенный*** (protected).
- «**-**» – атрибут с областью видимости типа ***закрытый*** (private).
- «**~**» – атрибут с областью видимости типа ***пакетный*** (package).

# Атрибуты класса. Имя атрибута

*<квантор видимости>* ***<имя атрибута>*** *[кратность]: <тип атрибута> = <исходное значение> {строка-свойство}*

- Представлено в виде *уникальной* строки текста
- Имя атрибута является единственным обязательным элементом в синтаксическом обозначении атрибута
- Должно начинаться со строчной буквы
- Не должно содержать пробелов



# Атрибуты класса. Кратность атрибута

*<квантор видимости> <имя атрибута> **[кратность]**: <тип атрибута> = <исходное значение> {строка-свойство}*

- ***Кратность атрибута*** характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса.
- Формат:*[нижняя граница . . верхняя граница]*

  ***Например: [0..1], [0..*], [1..3,5..7]***

[0..1] – кратность атрибута может принимать значение или 0 или 1
[0..*] – кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа – [*].
[1..*] – кратность атрибута может принимать любое положительное целое значение большее или равное 1
[1..5] – кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.
[1..3,5,7] – кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.
[1..3,7.. 10] – кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.
[1..3,7..*] – кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.
Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т. е. 1

# Атрибуты класса. Тип атрибута

*<квантор видимости> <имя атрибута> [кратность]:* ***<тип атрибута>*** *= <исходное значение> {строка-свойство}*

- Выражение, определяемое некоторым **типом данных** (в зависимости от языка программирования)
- В простейшем случае – *осмысленная строка текста*.
- Пример:

  *цвет: Color*

  *имяСотрудника[1..2]: String;*

  *видимость: Boolean*

# Атрибуты класса. Исходное значение

*<квантор видимости> <имя атрибута> [кратность]: <тип атрибута> =* ***<исходное значение>*** *{строка-свойство}*

- Служит для задания некоторого начального значения в момент *создания* отдельного экземпляра класса
- Пример:

  *цвет: Color = (255, 0, 0)*

  *имяСотрудника[1..2]: String = ‘Иван Иванов’;*

  *видимость: Boolean = истина*

# Атрибуты класса. Строка-свойство

*<квантор видимости> <имя атрибута> [кратность]: <тип атрибута> = <исходное значение>* ***{строка-свойство}***

- Служит для указания ***дополнительных свойств атрибута***, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения программы.
- Это значение принимается за ***исходное значение атрибута***, которое не может быть изменено в дальнейшем.
- Пример:

  **заработнаяПлата: Currency = $500 {frozen}**

## Операции класса

- Представляют собой некоторый сервис, интерфейс, который предоставляет каждый экземпляр класса или объект по требованию своих клиентов.
- Правила записи операций:

*<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения> {строка-свойство}*

# Операции класса. Список параметров

*<квантор видимости> <имя операции> **(список параметров)**: <выражение типа возвращаемого значения> {строка-свойство}*

- *Список параметров* является перечнем разделенных запятой *формальных параметров*, каждый из которых, в свою очередь, может быть представлен в следующем виде:

*<вид параметра> <имя параметра>: <выражение типа> = <значение параметра по умолчанию>*

# Операции класса. Строка-свойство

*<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения>* ***{строка-свойство}***

- ***Строка-свойство*** служит для указания значений свойств, которые могут быть применены к данной операции.
- Например, для указания последовательности действий будет использована строка-свойство вида:

**{concurrency = имя}** , где *имя* может принимать одно из следующих значений:

- **sequential** (последовательная),
- **concurrent** (параллельная),
- **guarded** (охраняемая)

# Операции класса. Строка-свойство

*<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения>* ***{строка-свойство}***

| 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. Такая семантика может породить состояние взаимной блокировки (или тупика), в которой два или более процессов взаимно блокируют друг друга и ни один не может продолжить выполнение. |
| {concurrent} | Операция допускает произвольное число параллельных вызовов и гарантирует правильность своего выполнения. Такие операции называются повторно входимыми (reenterable). |

# Операции класса. Примеры

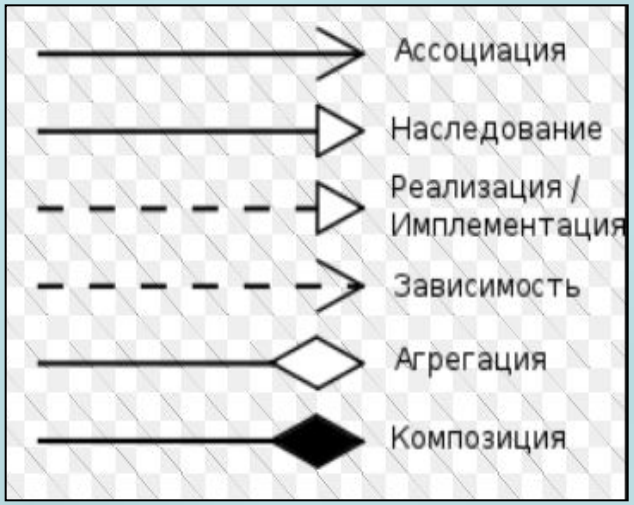
*<квантор видимости> <имя операции> (список параметров): <выражение типа возвращаемого значения> {строка-свойство}*

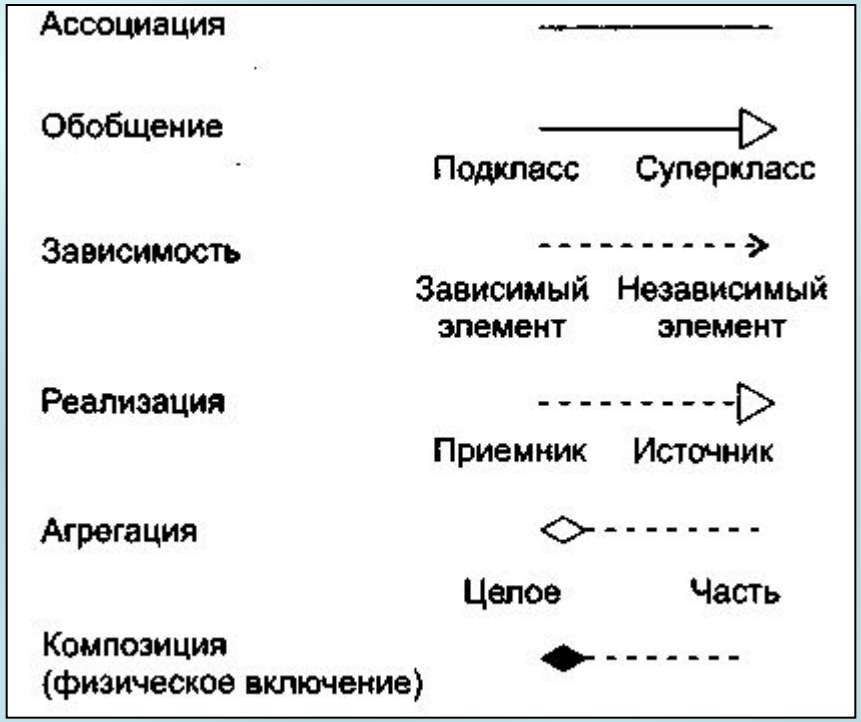
- +нарисовать (форма : Многоугольник = прямоугольник, цветЗаливки : Color = (0, 0, 255));
- -изменитьСчетКлиента (номерСчета : Integer) : Currency;
- #выдатьСообщение() : (‘Ошибка деления на ноль’).

# Отношения между классами

Базовыми отношениями на диаграмме классов являются:

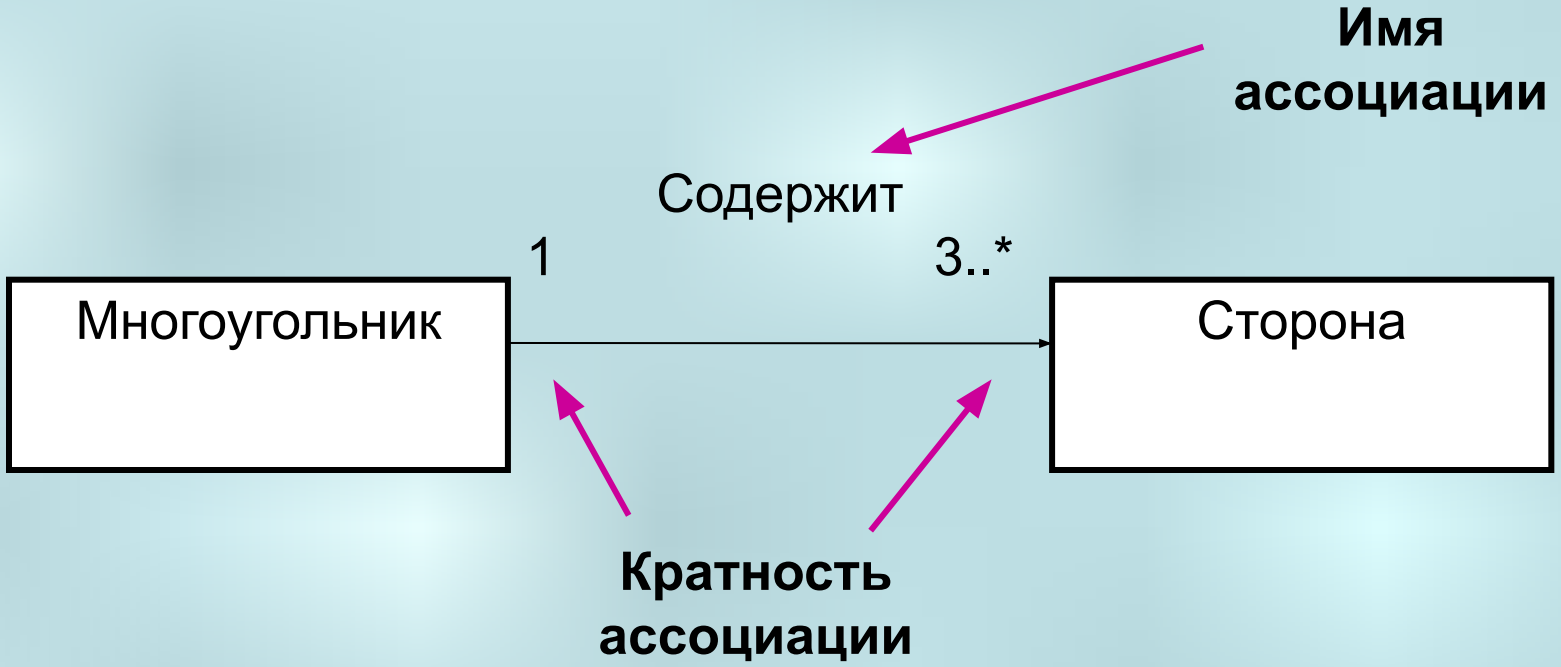
- отношения **зависимости** (*dependency*);
- отношения **обобщения** (*generalization*);
- отношения **ассоциации** (*association*);
- отношения **агрегации** (*aggregation*);
- отношения **композиции** (*composition*).



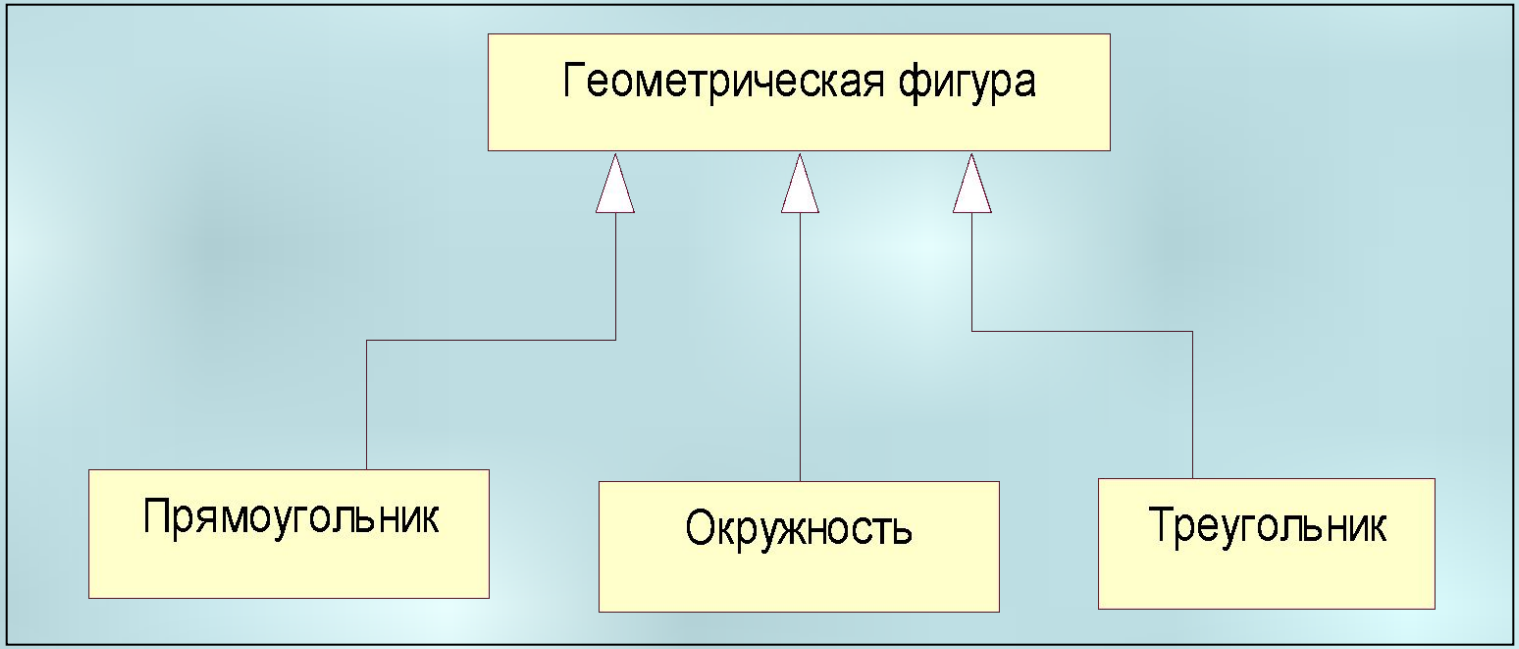

# Отношение ассоциации

- **Отношение ассоциации** свидетельствует о наличии **произвольного** отношения между классами.



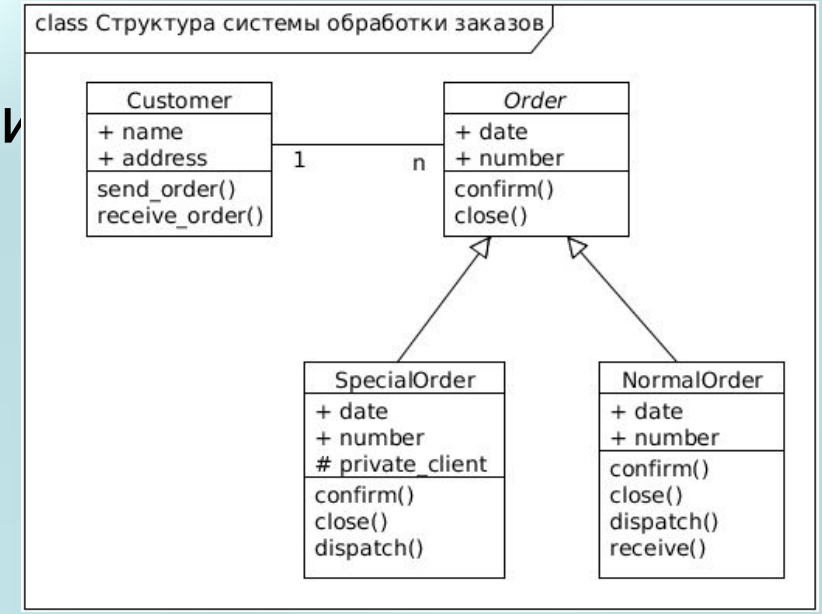
# Отношение обобщения

- Является отношением *классификации* между **более общим** элементом (родителем или предком) и **более частным** или специальным элементом (дочерним или потомком)



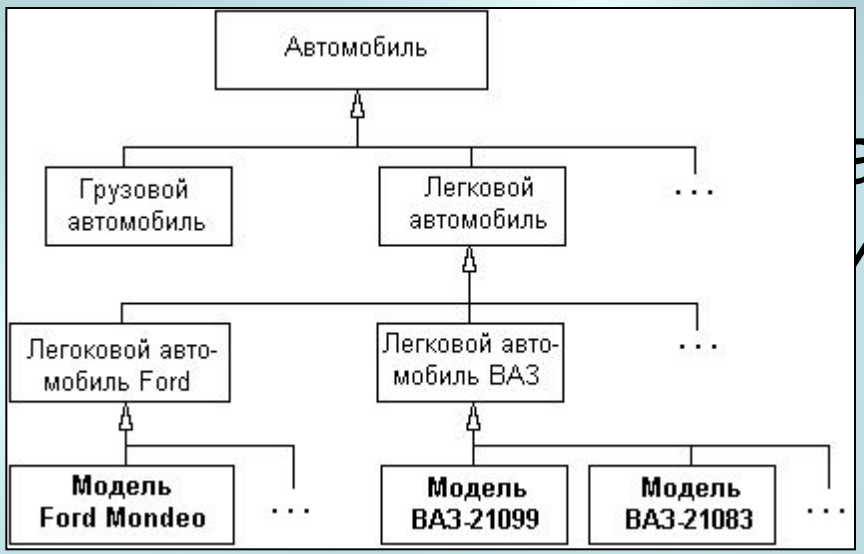
# Отношение обобщения

- Обобщение (generalization) показывает, что один из двух связанных классов (*подтип*) является частной формой другого (*надтипа*), который называется **обобщением** первого. Любой экземпляр подтипа является также экземпляром надтипа. Например: животные – это супертип млекопитающих, которые, в свою очередь, являются супертипом приматов. Эта взаимосвязь может быть описана фразой «А – это Б» (приматы – это млекопитающие, млекопитающи – это животные).



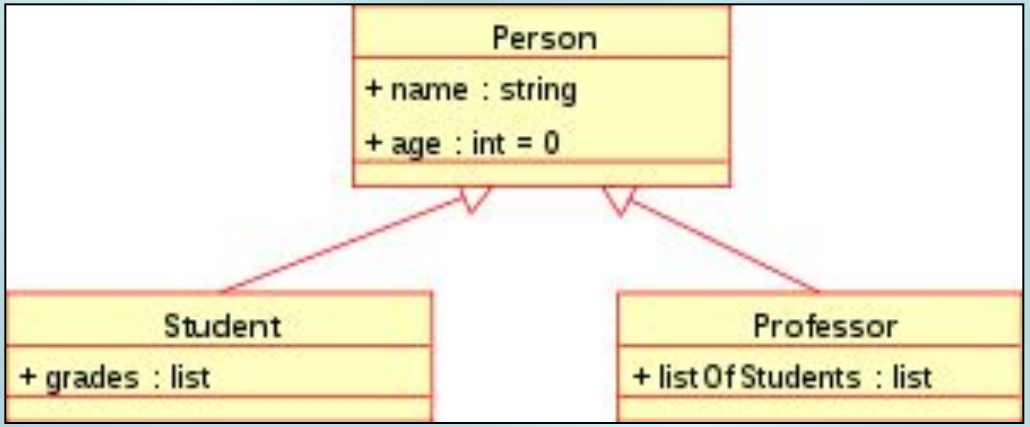
# Отношение обобщения

- Графически *обобщение* представляется линией с пустым треугольником у



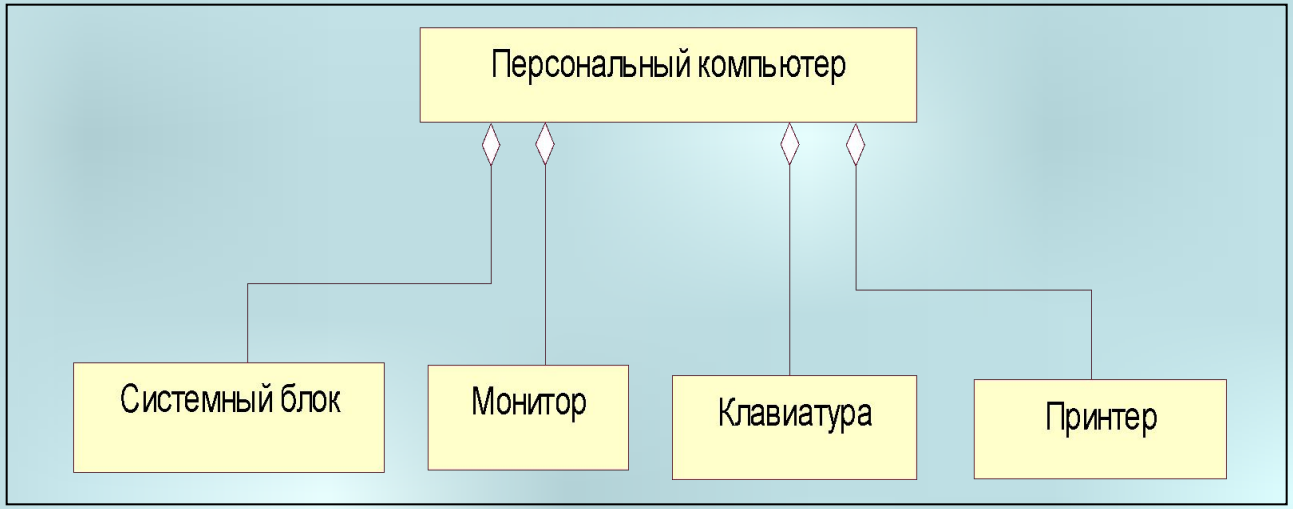

акже известно как

и «is a» взаимосвязь.



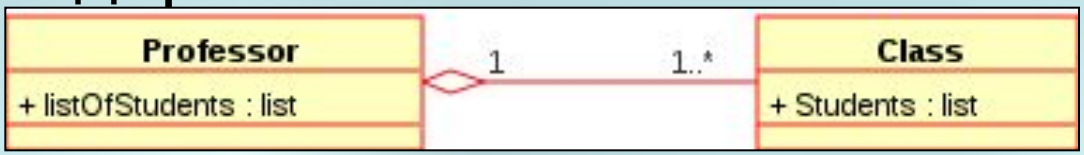
# Отношение агрегации

- Один из классов представляет собой некоторую сущность, которая **включает** в себя в качестве составных частей другие сущности.
- Применяется для представления системных взаимосвязей типа «*часть-целое*».



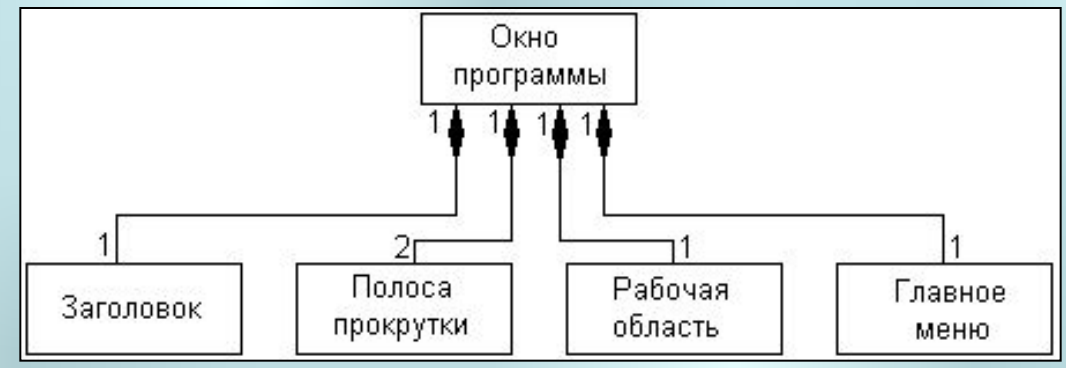
# Отношение агрегации

- *Агрегация* встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют *агрегацию по ссылке*, то есть когда время существования содержащихся классов (составных частей) не зависит от времени существования содержащего их класса (контейнера). Если контейнер будет уничтожен, то его содержимое — нет.



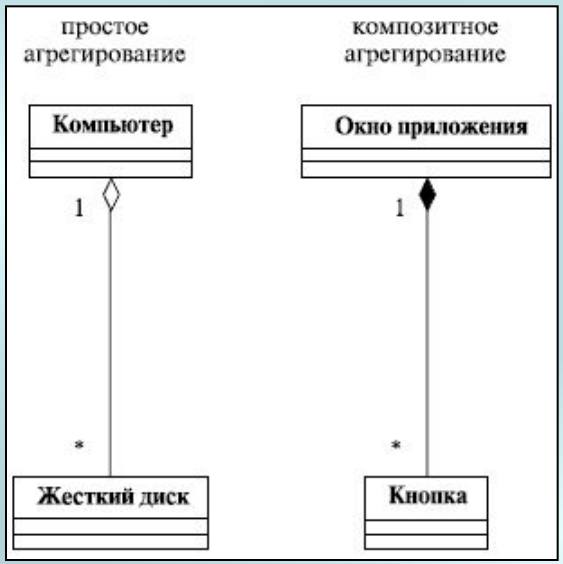
# Отношение композиции

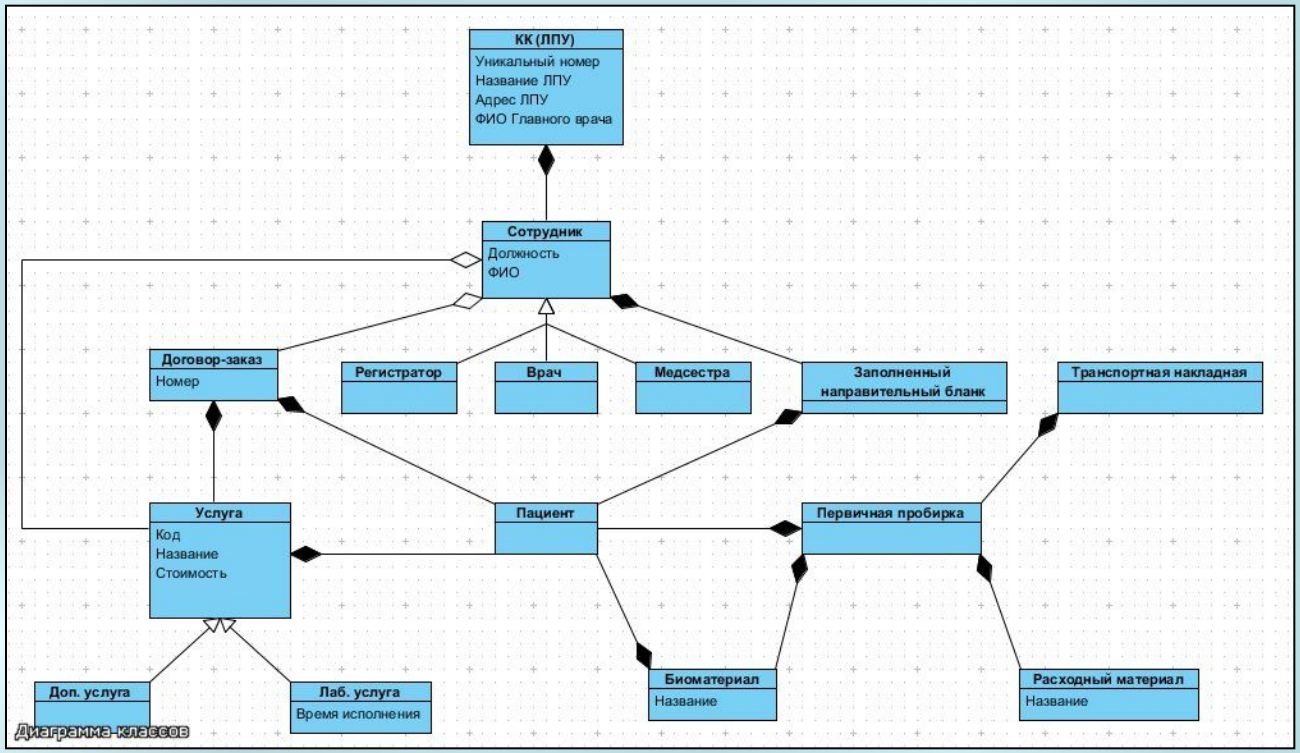
- Является частным случаем отношения агрегации.
- Части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются составные части.




# Отношение композиции

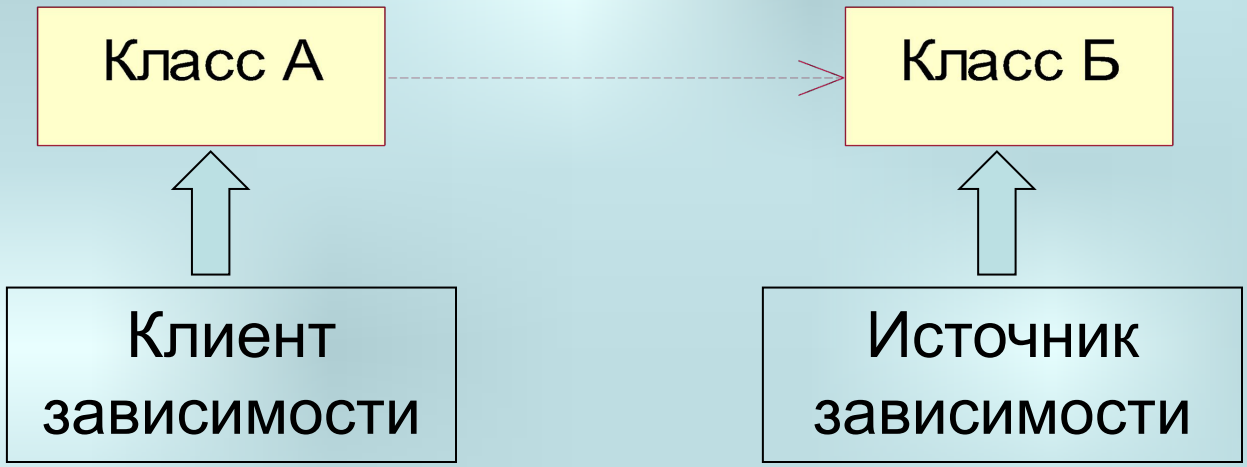
- *Композиция* имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

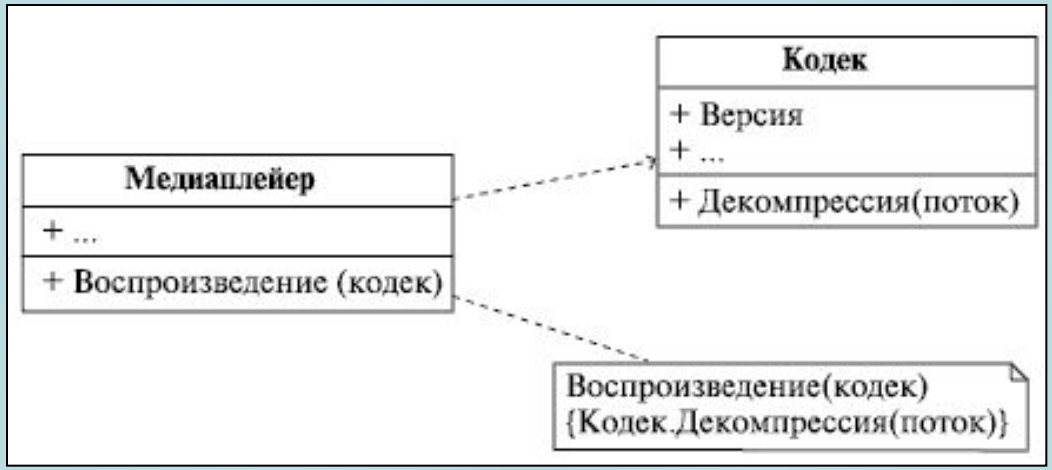


# Отношение зависимости

- Используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого элемента.



# Отношение зависимости

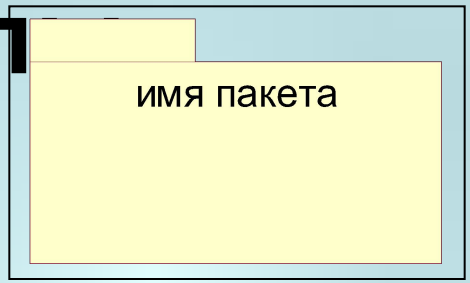
- Зависимость — это слабая форма отношения использования, при котором изменение в спецификации одного влечёт за собой изменение другого, причем обратное не обязательно. Возникает когда объект выступает например в форме параметра или локальной переменной.
- Графически представляется пунктирной стрелкой, **идущей от зависимого элемента к тому, от которого он зависит**.
- Зависимость может быть между экземплярами, классами или экземпляром и классом.

# Мощность отношений (кратность)

Мощность отношения (мултипликатор) означает число связей между каждым экземпляром класса (объектом) в начале линии с экземпляром класса в ее конце. Различают следующие типичные случаи:

| ***нотация*** | ***объяснение*** | ***пример*** |
|---|---|---|
| **0..1** | Ноль или один экземпляр | кошка имеет или не имеет хозяина |
| **1** | Обязательно один экземпляр | у кошки одна мать |
| **0..*** или * | Ноль или более экземпляров | у кошки может быть, а может и не быть котят |
| **1..*** | Один или более экземпляров | у кошки есть хотя бы одно место, где она спит |

# Пакеты и стереотип

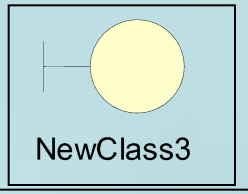



- служат для **группировки** элементов модели
- любой пакет владеет своими элементами
- любой элемент может принадлежать *только одному пакету*

Стереотипы классов – механизм, позволяющий разделять классы на категории. В языке UML определены три основных стереотипа классов:

- boundary (граница);
- entity (сущность);
- control (управление).

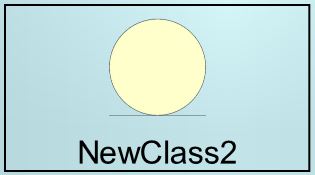
# Стереотипы



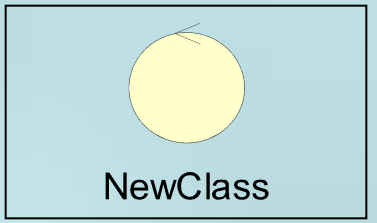

**Граничными классами (boundary classes)** называются такие классы, которые расположены на границе системы и всей окружающей среды и является составной частью системы. Это экранные формы, отчеты, интерфейсы с внешней аппаратурой (например, принтеры или сканеры) и интерфейсы с другими системами.

Чтобы найти граничные классы, надо исследовать диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать, по крайней мере, один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

**Классы-сущности (entity classes)** информацию, которая должна храниться **постоянно** и не уничтожаться с уничтожением объектов данного класса или прекращением работы моделируемой системы. Они имеют наибольшее значение для пользователя, и потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.



# Стереотипы

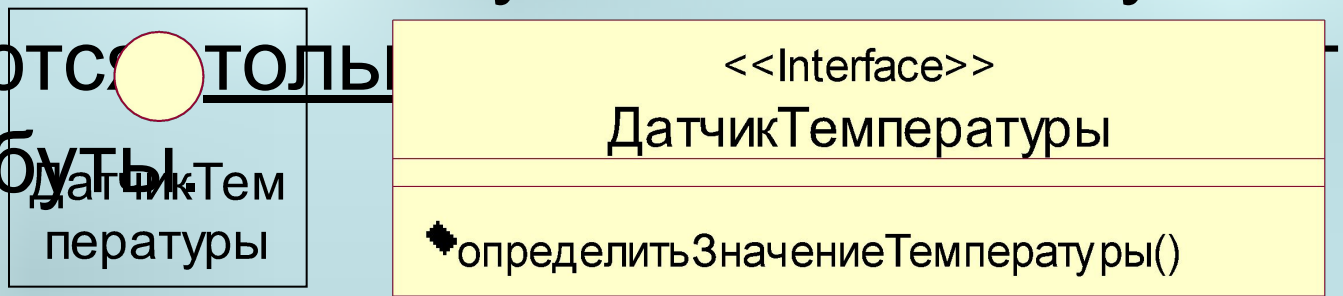



**Управляющие классы (control classes**) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

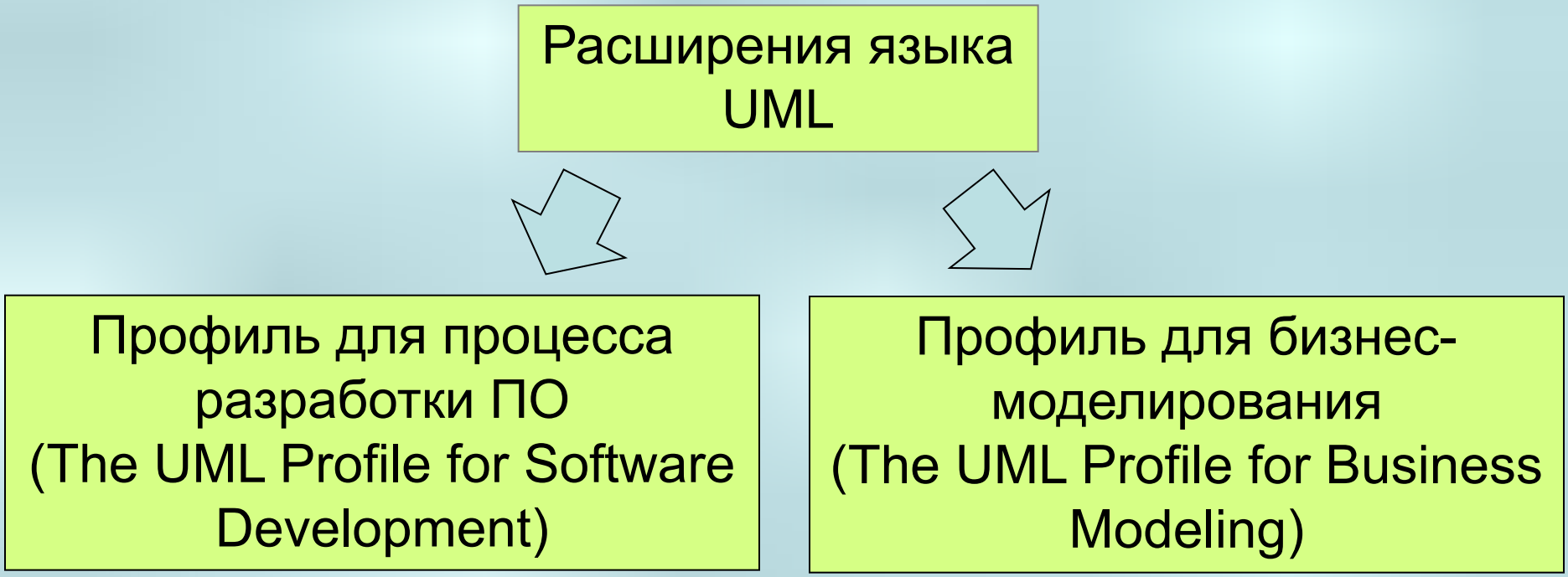
В системе могут быть управляющие классы, общие для нескольких вариантов использования. Например, может быть класс SecurityManager (менеджер безопасности), отвечающий за контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими как разделение ресурсов, распределенная обработка данных или обработка ошибок.

# Интерфейс (interface)

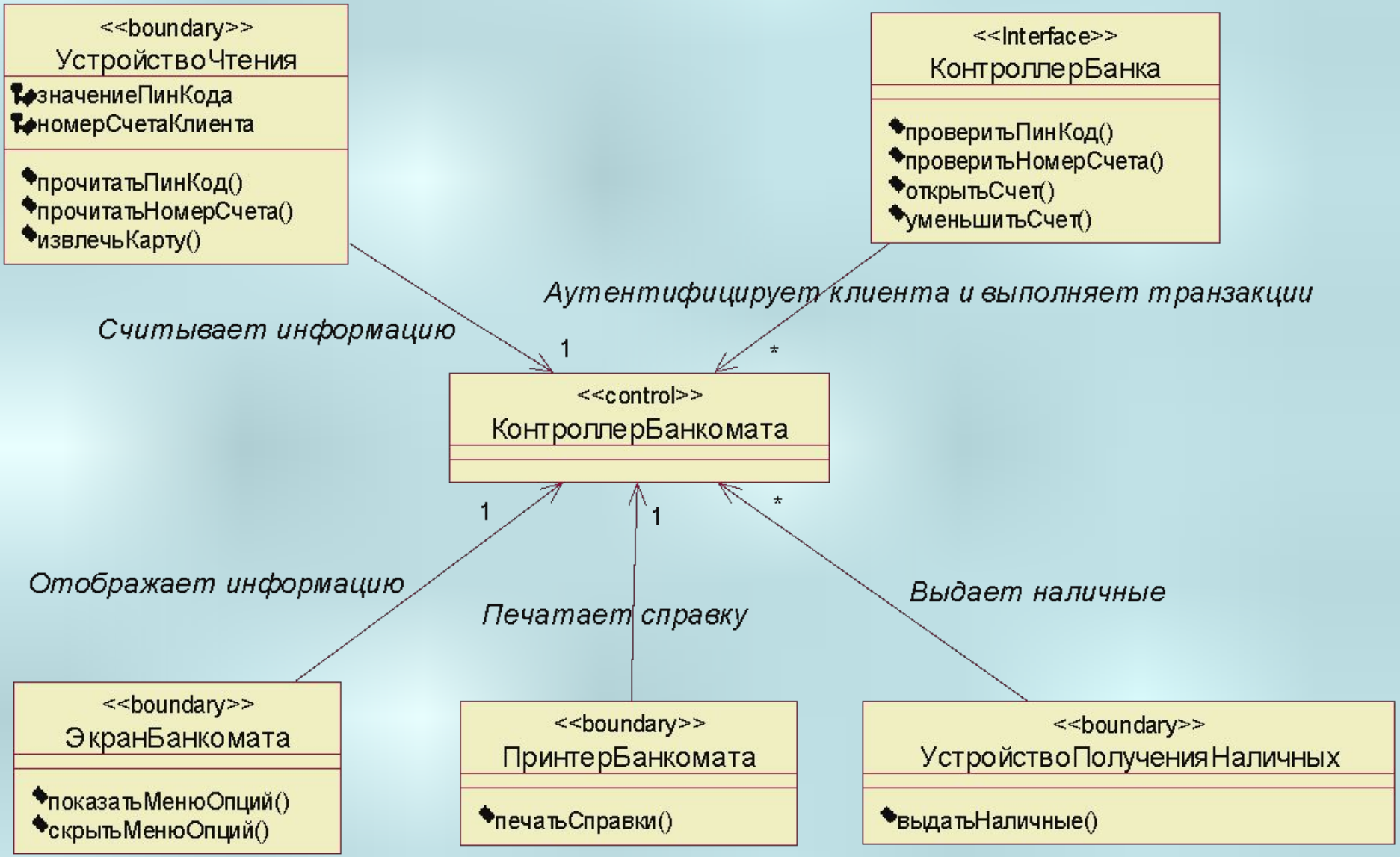
- в контексте языка UML является специальным случаем класса, у которого имеются только ствуют атрибуты



# Расширения языка UML



# Пример диаграммы классов



# Литература


1. Ларман К. - Применение UML 2.0 и шаблонов проектирования 2016
2. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование Д. Арлоу, Айви Нейштадт, 2016