

### 3. ОО Программирование

- ОО парадигма программирования
- Основные понятия ООП
  - Классы
  - Объекты
  - Экземпляры
- Принципы ООП

# ОО программирование

- Реши, какие требуются классы
- Обеспечь полный набор операций для каждого класса
- Явно вырази общность через наследование

Б.Страуструп

Квинтэссенция ОО- парадигмы программирования

# ОО декомпозиция

- Программная система состоит из объектов, которые обмениваются сообщениями
- Каждый объект обладает:
  - Поведением
  - Состоянием
  - Идентичностью (Индивидуальностью)
- Схожие объекты объединяются в классы

# **Основные понятия ООП**

**Класс** – абстракция данных и поведения некоторого «вида» объектов

**Объект** – экземпляр класса



*Инкапсуляция* - это защита отдельных элементов объекта, не затрагивающих существенных характеристик его как целого

| Символ | Значение                                                                            |
|--------|-------------------------------------------------------------------------------------|
| +      | public - открытый доступ                                                            |
| -      | private - только из операций того же класса                                         |
| #      | protected - только из операций этого же класса и классов, создаваемых на его основе |

| Телевизор                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------|
| + Язык экранного меню<br>— Частота каналов<br>+ Порядок и именование каналов<br>+ ...                                                         |
| — Самодиагностика()<br>+ Включить()<br>+ Выключить()<br>+ Поиск каналов()<br>— Декодирование сигнала()<br>+ Переключение каналов()<br>+ ...() |

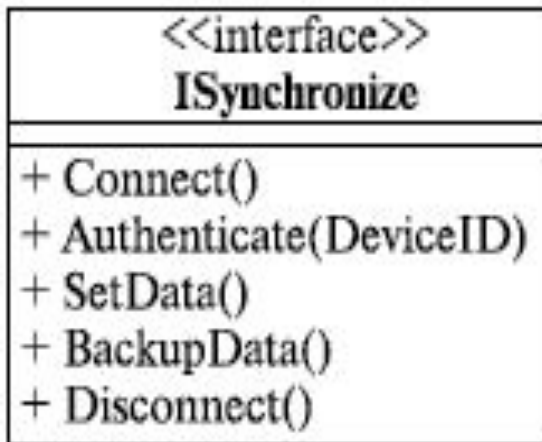
# Как использовать объекты класса?

*Интерфейс* - это логическая группа открытых ( public) операций объекта. Один и тот же объект может иметь несколько интерфейсов.

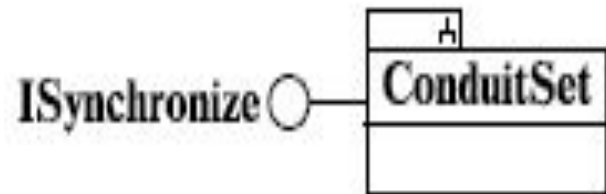
- Однако *интерфейс* - это не только и не столько группа операций объекта.
- *Интерфейс* всегда реализуется некоторым классом, который в таком случае называют классом, поддерживающим интерфейс.



# Первый и самый простой из них - это *класс* со стереотипом <<interface>>



1



2

- *1 - способ хорош, если нужно показать, какие именно операции предоставляет интерфейс. Если же такие подробности в данный момент не важны, предоставляемый интерфейс изображают в виде (2-рисунок) кружочка или, как говорят, "леденца" (lollipop)*

# Всегда ли нужно создавать новые классы?



В дополнение можно назвать несколько причин, почему стоит использовать уже существующие классы:

- Во-первых: если когда-то мы уже решили некоторую проблему, зачем начинать все "с нуля", повторяя уже однажды сделанные действия?
- Во-вторых, таким образом мы делаем решение мобильным и расширяемым.
- В-третьих, существующие классы, как правило, хорошо отлажены и показали себя в работе. Разработчику не надо тратить время на *кодирование*, отладку, тестирование и т. д., - мы работаем с хорошо отлаженным и проверенным временем кодом, который зарекомендовал себя в других проектах и в котором уже выявлено и исправлено большинство ошибок.

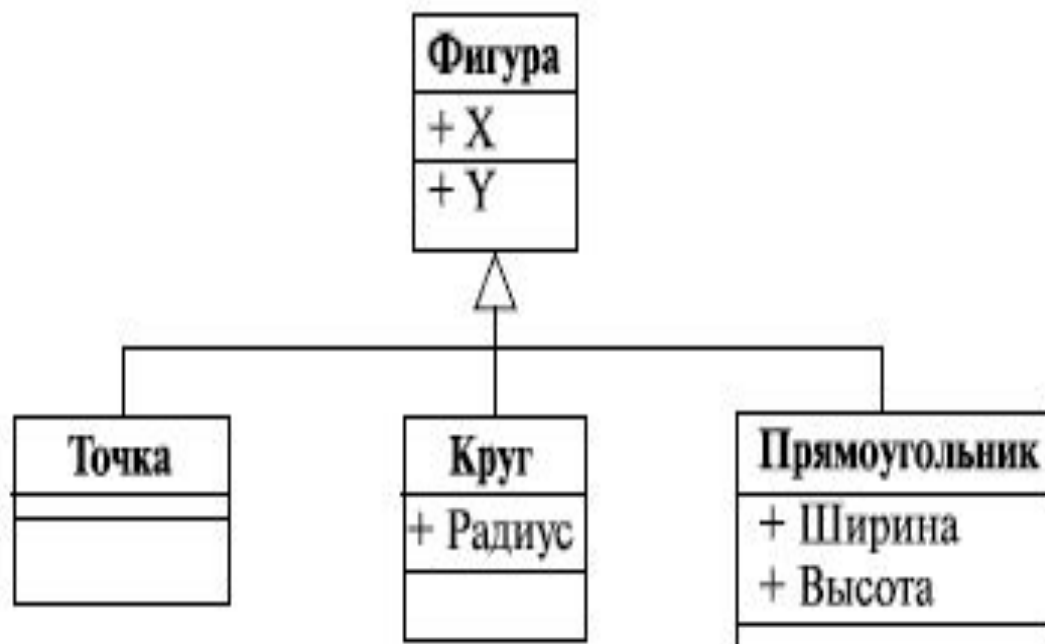
**Нужно создавать классы на основе уже существующих, но как?**

**Обобщение (наследование)** – это *отношение* между более общей сущностью, называемой *суперклассом*, и ее конкретным воплощением, называемым *подклассом*.

Для того чтобы научиться эффективно моделировать *наследование*, обратимся к классикам, а именно к Г. Бучу. Он советует проводить эту процедуру в такой последовательности:

- Найдите атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности. Это позволит избежать ненужного дублирования структуры и функциональности объектов.
- Вынесите эти элементы в некоторый общий суперкласс, а если такого не существует, то создайте новый класс.
- Отметьте в модели, что подклассы наследуются от суперкласса, установив между ними отношение обобщения.







***Полиморфизм*** является основой для реализации *механизма интерфейсов* в языках программирования.

Какого класса *объект*: как только *пользователь* обращается к некоторой *операции* через *интерфейс*, определяется фактический *класс* объекта и вызывается соответствующая *операция класса*.

# Основные принципы ООП

- Абстракция
  - Рассмотрение только существенных для решаемой задачи характеристик объекта
  - Граница между существенными и НЕсущественными характеристиками объекта называется **барьером абстракции**
- Инкапсуляция
  - Соккрытие особенностей реализации, отделение контрактных обязательств абстракции от их реализации
- Иерархия
  - Упорядочение абстракций, расположение их по уровням
- Модульность
  - Разделение системы на внутренне связанные модули, которые слабо связаны между собой

# Это и есть ОО метод

- Абстракция - оставляет нам только существенные детали
- Инкапсуляция - убирает из поля зрения реализацию, оставляя для рассмотрения только поведение объектов
- Модульность - объединяет абстракции в группы
- Иерархия – распределяет абстракции по уровням, позволяя вести рассуждения на абстрактном уровне и применять результаты к частным случаям

Это и есть ОО-метод декомпозиции программной системы, ОО-способ ведения рассуждений, ОО-средство борьбы со сложностью

# Методы ООП

## - Типизация

Способ защититься от использования объектов одного типа вместо другого

## - Полиморфизм

способ поставить в соответствие некой грамматической конструкции контекстно-зависимую семантику

## - Параллелизм

способность объекта обрабатывать несколько сообщений одновременно

## - Сохраняемость

способность объекта сохранять состояние между сеансами работы приложения

# Классы

- Class

Абстракция данных с общей структурой и поведением

- Interface

базовый класс, задающий только поведение, в UML имеет стереотип <<interface>>

- Abstract class

базовый класс, не имеющий экземпляров

# Атрибуты классов

- Attribute

атрибут (поле)

- Class attribute

атрибут класса (static)

- Derived attribute

производный (вычисляемый) атрибут

- Export control

доступ (public:+, protected:#, private:-, package:~)

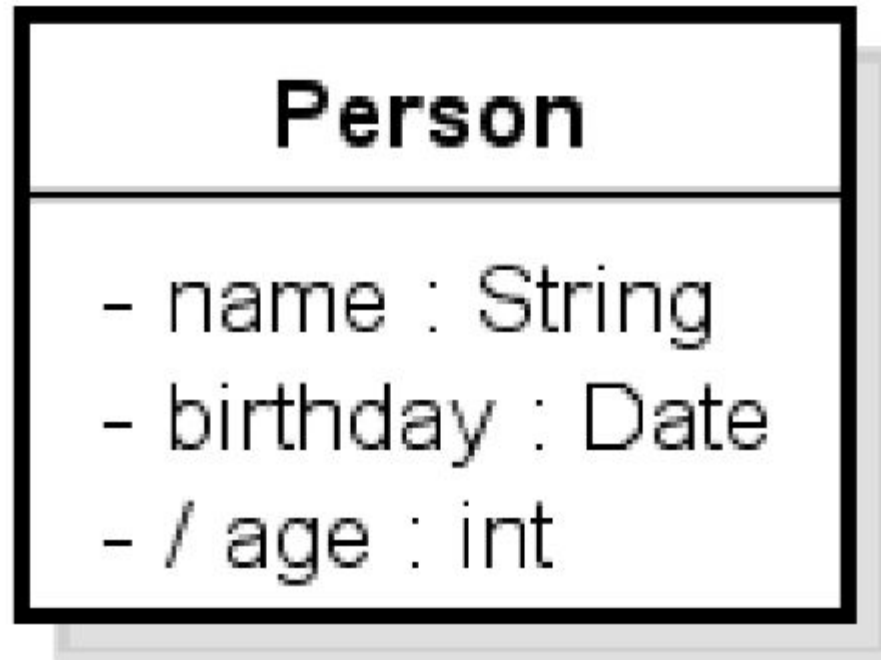
- Aggregation

способ включения (none, composite, aggregate)

- Может иметь стереотип

Syntax: <role\_name>:<class\_name><=default\_value>

# Атрибуты классов



name, birthday – атрибуты age – производный атрибут  
(вычисляется через birthday)

# Атрибуты классов

| <b>DataLabels</b>                             |
|-----------------------------------------------|
| <u>+ NAME LABEL : String = "Your Name"</u>    |
| <u>+ BIRTHDAY LABEL : String = "Birthday"</u> |
| <u>+ AGE LABEL : String = "Age"</u>           |
|                                               |

NAME\_LABEL, BIRTHDAY\_LABEL, AGE\_LABEL - атрибуты класса (static)



# Методы(операции)

- Method (operation) – метод
- Могут быть static, final, abstract
- Видимость: public, protected, private, package
- Syntax:
- <<stereotype>> name(<parameters>) : <return type>
- Parameter: parameter\_name : type

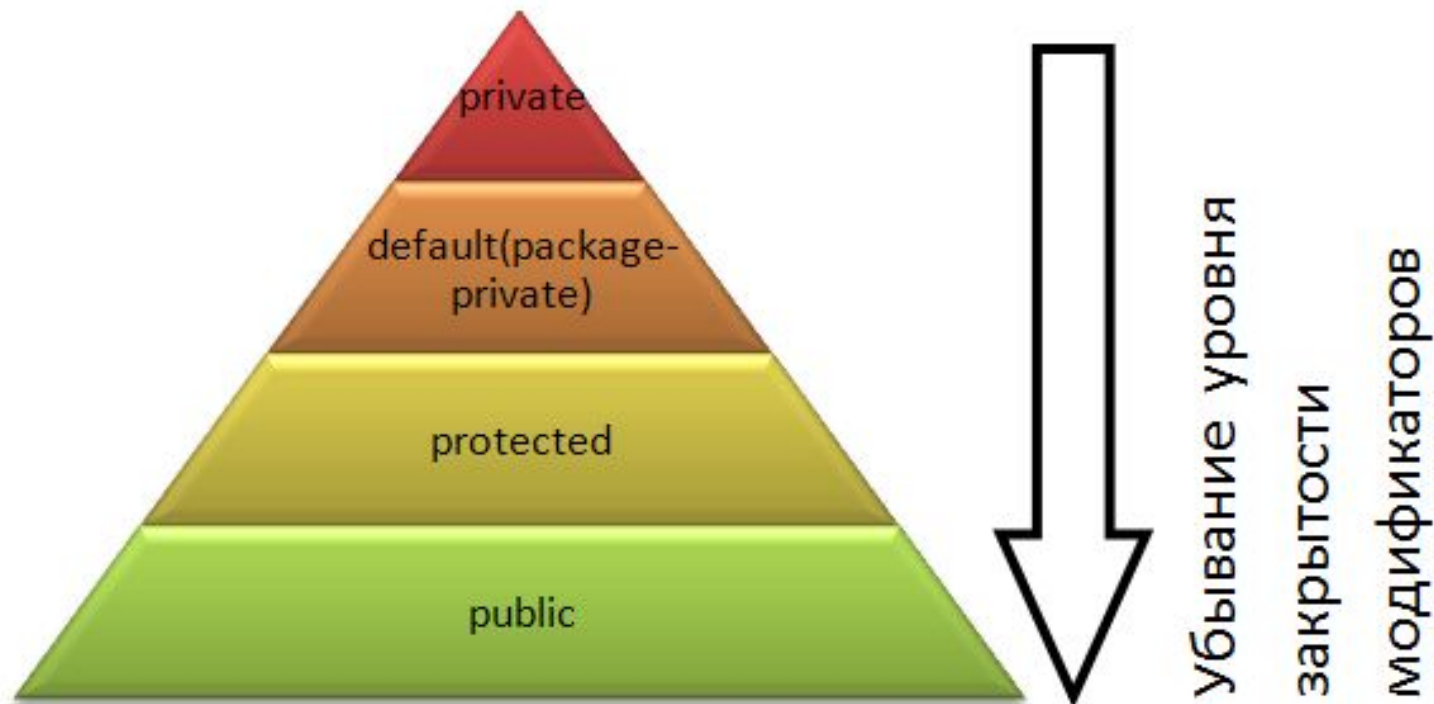
| Date                                                                              |
|-----------------------------------------------------------------------------------|
| - date : long                                                                     |
| + Date(date : long) : void<br>+ setDate(date : long) : void<br>+ getDate() : long |

- • **final** – нельзя изменять значение переменной, нельзя переопределять метод родителя, нельзя наследовать класс
- **abstract** – первая окончательная реализация класса должна переопределять все абстрактные методы. Не должно быть фигурных скобок.
- **static** – можно помечать методы, переменные класса, вложенный класс, блоки инициализации

## Дополнение по поводу интерфейса:

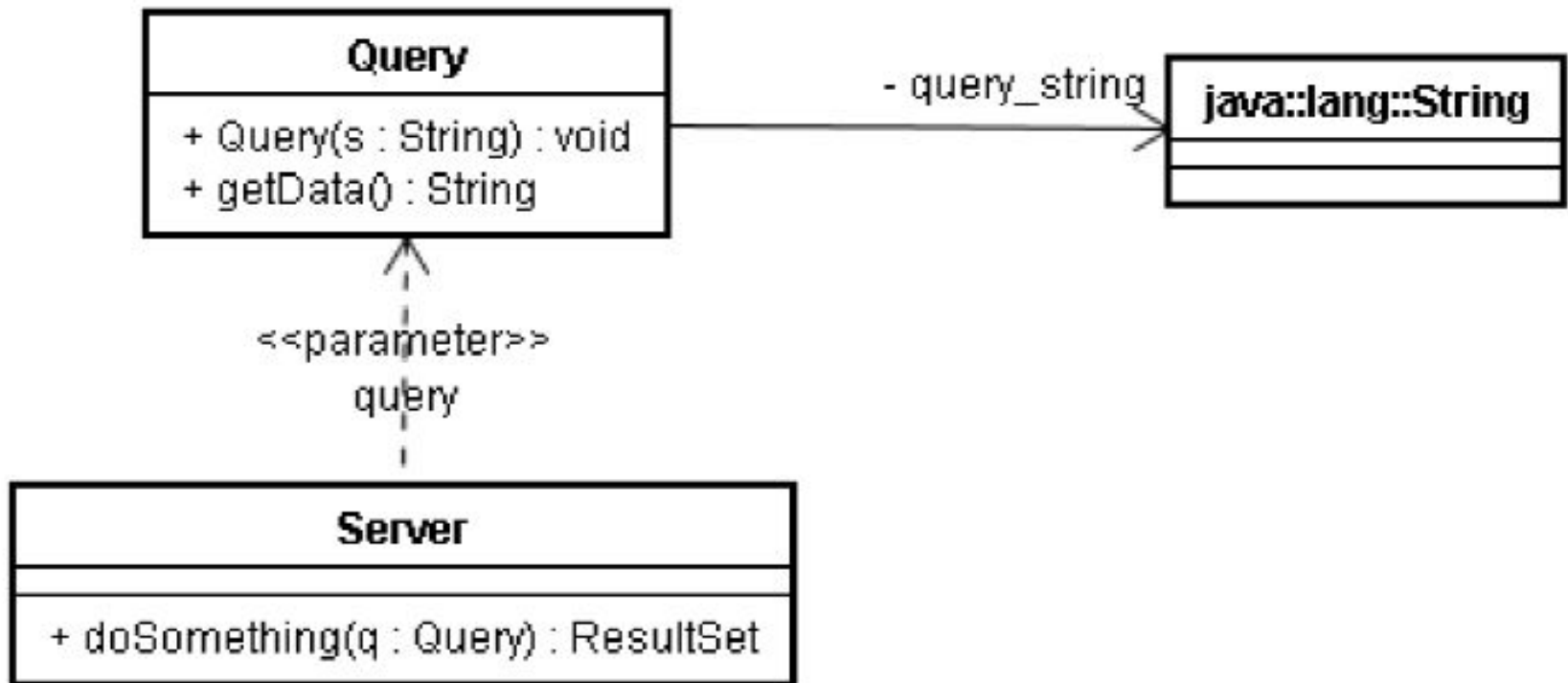
- Методы всегда PUBLIC и ABSTRACT, даже если это не объявлено.
- Методы НЕ могут быть STATIC, FINAL, STRICTFP, NATIVE, PRIVATE, PROTECTED
- Переменные только PUBLIC STATIC FINAL, даже если это не объявлено.
- Переменные НЕ могут быть STRICTFP, NATIVE, PRIVATE, PROTECTED
- Может только наследовать (extend) другой интерфейс, но не реализовывать интерфейс или класс (implement).

- **private** члены класса доступны только внутри класса
- **package-private** или **default (по умолчанию)** члены класса видны внутри пакета
- **protected** члены класса доступны внутри пакета и в классах-наследниках
- **public** члены класса доступны всем



# Диаграмма классов

- определяет типы (классы) объектов системы и статические связи между ними



# Связи между классами

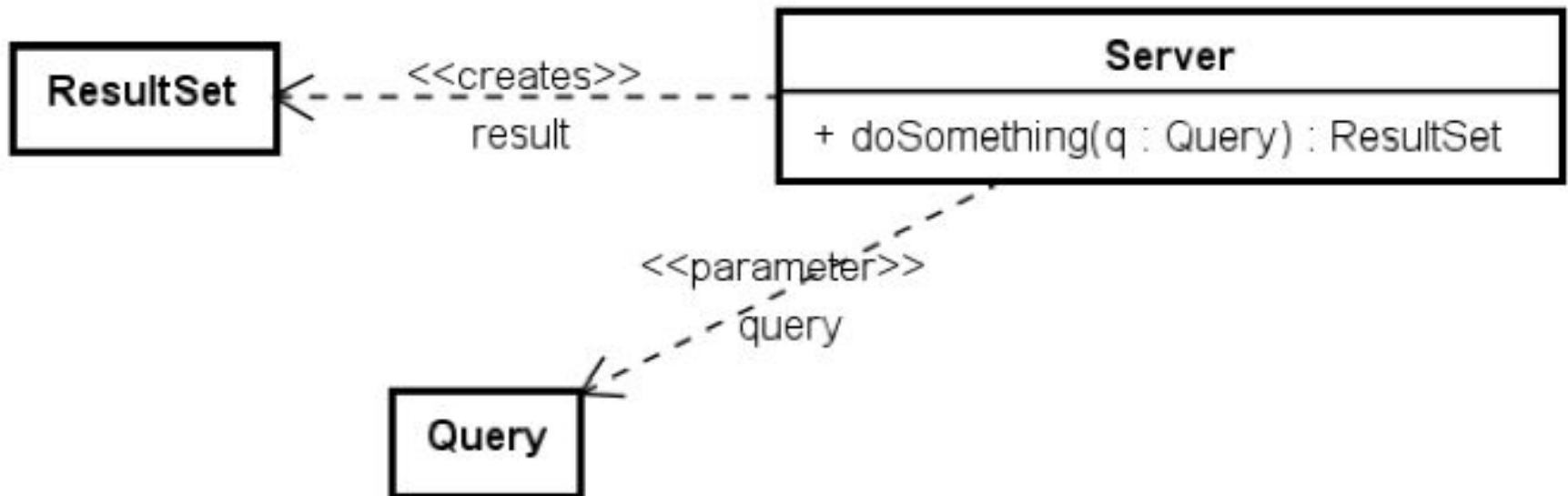
- Зависимость - Dependency
- Ассоциация - Association
- Агрегация - Aggregation
- Композиция - Composition
- Генерализация- Generalization
- Реализация - Realization

# Dependency

- Определяет отношение зависимости (осведомленности)
- Имеет выделенное направление
- Обладает ролью

Server зависит от Query, так как использует этот класс в качестве параметра метода

Server также зависит от ResultSet, поскольку возвращает значение этого типа



# Association

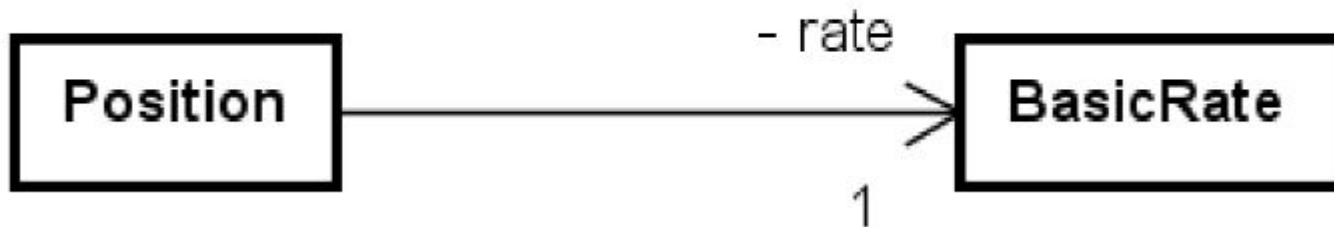
- Ассоциация - отношение связанности
- Подразумевает наличие зависимости
- Обладает 2-мя ролями
- Роль обладает множественностью (1, n, \*, 0..n, 1..n, 1..\*)
- Пример: сотрудник может занимать несколько должностей, но на одной должности находится не более одного сотрудника





# Association

- Ассоциация может иметь выделенное направление
  - Должность связана с базовым тарифом оплаты
  - Тариф оплаты никак не связан с конкретной должностью



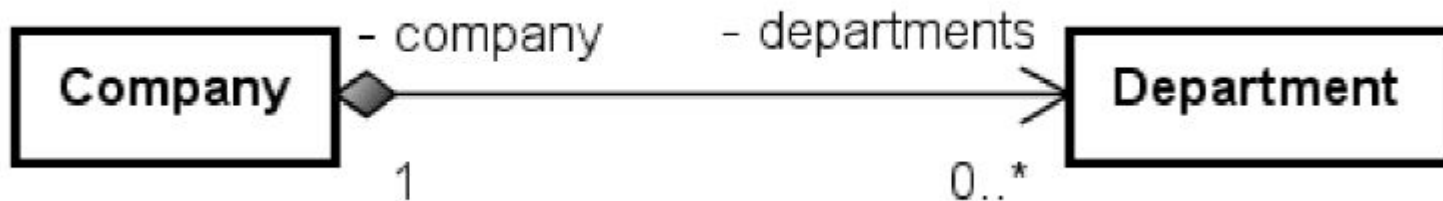
# Aggregation

- Агрегация – определяет отношение часть-целое
- Частный случай ассоциации
- Часть может принадлежать различным целым
  - Журнал состоит из одной и более статей; статья может быть опубликована в нескольких журналах



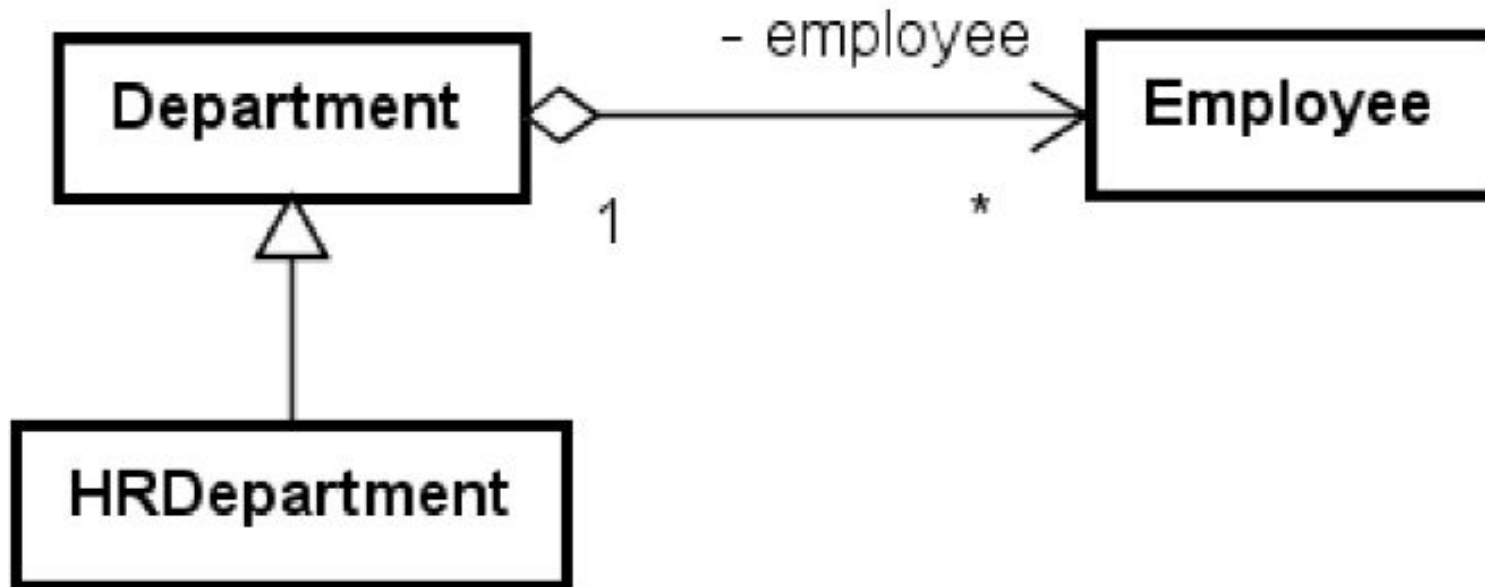
# Composition

- Композиция – частный случай агрегации
- Отношение «часть - целое»
- Целое отвечает за жизненный цикл своих частей
  - Отделы не существуют без компании
- Часть принадлежит только одному целому



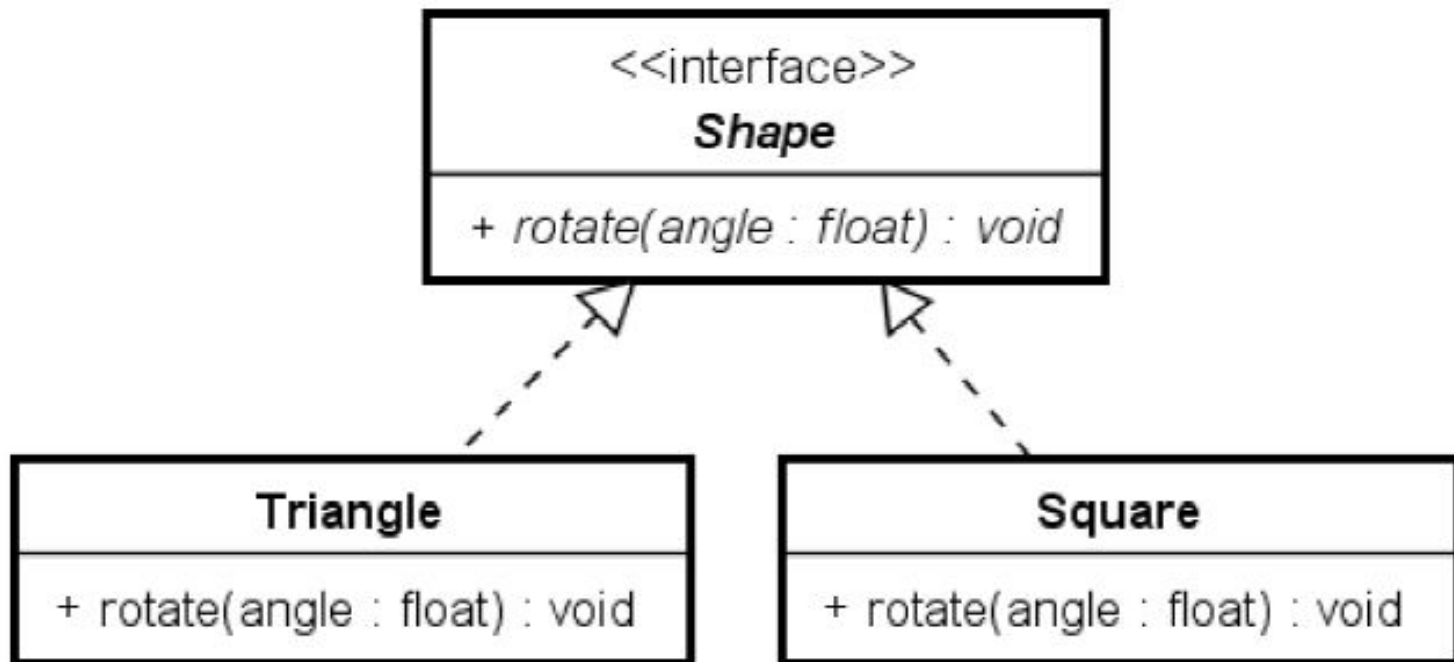
# Generalization

- Генерализация - обобщение
- Отношение «частное-общее»
  - Отдел кадров – частный случай отдела



# Realization

- Реализация – отношение выполнения соглашения (реализация интерфейса)
  - Треугольник и квадрат реализуют алгоритм вращения, специфицированный интерфейсом «Фигура»



- **Контрольные вопросы**
- Какие три принципа лежат в основе ООП?
- Что такое интерфейс? На каком из базовых принципов ООП основан *механизм интерфейсов*?
- Что такое n-арная ассоциация?
- В чем разница между агрегацией и композицией?
- Что такое класс ассоциации?