

# Основы объектно-ориентированного программирования

Наумов Д.А., доц. каф. КТ

Основы программной инженерии, 2020

# Содержание лекции

- 1 Элементы объектной модели
- 2 Структура и организация определения класса

Объектную модель составляют четыре главных элемента:

- абстрагирование (abstraction) - выделение существенных характеристик некоторого объекта, отличающие его от всех других видов объектов;
- инкапсуляция (encapsulation) - отделение друг от друга элементов объекта, определяющих его устройство и поведение; представление системы в виде совокупности обособленных сегментов (модульность);
- иерархия (hierarchy) - упорядочение абстракций, средство классификации объектов, систематизация связей между объектами.
- полиморфизм (polymorphism) - механизм, позволяющий заменять поведение объектов производных классов.

# Абстракция

## Абстракция

выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.

Два вида абстракций в ООП:

- **тип данных объектной природы (класс)**— определяемое программистом расширение исходных типов языка;
- **экземпляр класса (объект)** — переменная класса. Объект обладает состоянием, поведением и идентичностью.

Состояние объекта:

- характеризуется набором его свойств (атрибутов) и текущими значениями каждого из этих свойств;
- результат его поведения.

# Инкапсуляция

Инкапсуляция предполагает возможность ограничения доступа к данным класса из других классов (python: namespaces).

- это позволяет упростить интерфейс класса, показав наиболее существенные для внешнего пользователя данные и методы;
- скрытие реализации обеспечивает возможность внесения изменений в реализацию класса без изменения других классов.

# Типы структурных иерархий

Иерархическая декомпозиция и иерархическая организация ПО образуют один из основных систематических методов преодоления сложности программного обеспечения.

- структурная иерархия "is-part-of" (агрегирование как разновидность ассоциации)
- Структурные иерархии "is-a" и "is-like-a"

## Обобщение

означает, что объекты классапотомка могут использоваться везде, где допустимы объекты родительского класса.

- Создавая базовый тип (базовый класс), программист выражает наиболее общие идеи относительно объектов, из которых конструируется программа.
- В производных классах программист уточняет различия в реализации конструкций базового класса.
- Производный класс полностью дублирует интерфейс базового класса, т. е. дублирует наблюдаемое поведение объекта базового класса.
- Поведение объекта производного класса может отличаться от поведения объекта базового класса.

# Полиморфизм

Поведение объектов, на которых вызываются видоизмененные методы, интерфейс к которым заявлен в базовом классе, называют **полиморфным**, а механизм, обеспечивающий такое поведение, называют **полиморфизмом**.

Мы имеем возможность:

- работая с объектами разных типов, представляющих различные абстракции, мы можем использовать методы, которые имеют одинаковый интерфейс, что, в конечном счете, упрощает и реализацию программы, и ее восприятие;
- разрабатывать общие процедуры обработки объектов.



**Задача классов** - предоставить программисту инструмент для создания новых типов данных с тем, чтобы их можно было без ограничений использовать в программе наряду со встроенными типами.

**Тип данных** является воплощением некоторой концепции (пример: целые числа), которое определяется рядом характеристик:

- областью применения типа;
- способом представления типа в памяти;
- множеством операций, разрешенных для этого типа;
- множеством совместимых типов.

Новые типы создаются для воплощения концепций, которые *не выражаются непосредственно* (или адекватно) встроенными типами.

# Класс - элемент абстракции

- Определяя класс, мы создаем программную сущность, позволяющую выделить существенные характеристики некоторого объекта, отличающие его от других видов объектов.
- Мы отделяем друг от друга элементы объекта, определяющие его устройство (**элементы-данные**), от элементов, определяющих его поведение (**элементы-действия**).
- Данные, объявленные внутри определения класса, могут, в частности, быть **переменными-членами класса** и **константами-членами класса**.
- Функции, объявленные внутри определения класса, называются **функциями-членами класса**, или **методами класса**.

# Диаграмма классов

## Диаграмма классов — основная логическая модель проектируемой системы

- **Диаграмма классов (class diagram)** — диаграмма, предназначенная для представления модели статической структуры программной системы в терминологии классов объектно-ориентированного программирования.
- Диаграмма классов представляет собой **граф, вершинами или узлами которого являются элементы типа “классификатор”**, которые **связаны различными типами структурных отношений**.
- **Классификатор (classifier)** — специальное понятие, предназначенное для классификации экземпляров, которые имеют общие характеристики.

# Диаграмма классов

## Характеристика (feature)

Понятие, предназначенное для спецификации особенностей структуры и поведения экземпляров классификаторов.

## Структурная характеристика (structural feature)

Типизированная характеристика классификатора, которая специфицирует структуру его экземпляров.

## Характеристика поведения (behavioral feature)

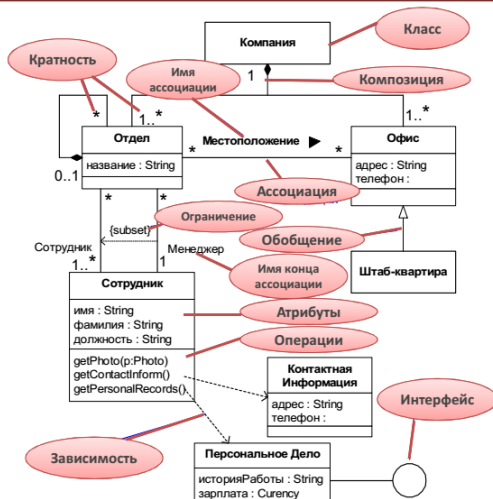
Характеристика классификатора, которая специфицирует некоторый аспект поведения его экземпляров.

## Класс (class)

Элемент модели, который описывает множество объектов, имеющих одинаковые спецификации характеристик, ограничений и семантики.

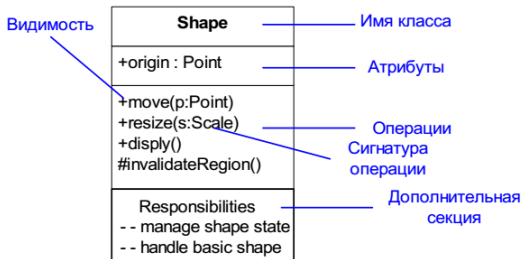
# Диаграмма классов

## Основные обозначения на диаграмме



# Диаграмма классов

## Варианты графического изображения класса на диаграмме классов



# Диаграмма классов

## Разновидности классов

### Абстрактный (abstract)

класс **не имеет экземпляров или объектов**, для обозначения его имени используется наклонный шрифт

### Активный класс (active class)

класс, каждый экземпляр которого **имеет свою собственную нить управления**

### Пассивный класс (passive class)

класс, каждый экземпляр которого **выполняется в контексте некоторого другого объекта**

**Квалифицированное имя (qualified name)** используется для того, чтобы явно указать, к какому пакету относится тот или иной класс. Для этого применяется специальный символ в качестве разделителя имени – двойное двоеточие “::”.

Имя класса без символа разделителя называется **простым именем** класса.

# Диаграмма классов

## Атрибут (attribute) класса

- служит для представления отдельной структурной характеристики или свойства, которое является общим для всех объектов данного класса.
- $\langle \text{атрибут} \rangle ::= [\langle \text{видимость} \rangle] [\text{'/'}] \langle \text{имя} \rangle [\text{'.'}]$   
 $\langle \text{тип атрибута} \rangle [\text{'['} \langle \text{кратность} \rangle \text{'}] [\text{'='} \langle \text{значение по умолчанию} \rangle]$   
 $[\text{'{'} \langle \text{модификатор атрибута} \rangle [\text{' '}, \langle \text{модификатор атрибута} \rangle]^* \text{'}}]$   
 где:

$\langle \text{видимость} \rangle ::= \text{'+'} \mid \text{'-' } \mid \text{'\#'} \mid \text{'\sim'}$ .

- **видимость (visibility)** может принимать одно из 4-х возможных значений и отображаться либо посредством специального символа, либо соответствующего ключевого слова.



# Диаграмма классов

## Вид видимости

**+ public**  
(общедоступный)

Общедоступный элемент является видимым всеми элементами, который имеют доступ к содержимому пространства имен, который им владеет.

**- private**  
(закрытый)

Закрытый элемент является видимым только внутри пространства имен, который им владеет.

**# protected**  
(защищенный)







Защищенный элемент является видимым для элементов, которые имеют отношение обобщения с пространством имен, который им владеет.

**~ package**  
(пакет)

Элемент, помеченный как имеющий пакетную видимость, является видимым всеми элементами в ближайшем охватывающем пакете в предположении. За пределами ближайшего охватывающего пакета элемент, помеченный как имеющий пакетную видимость, не является видимым.

# 

### 

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 семантикой	

# Диаграмма классов

**Ассоциация (*association*)**  
произвольное отношение или  
взаимосвязь между классами

**Имя конца ассоциации**  
специфицирует *роль* (role), которую играет класс, расположенный на соответствующем конце рассматриваемой ассоциации.

**Символ отсутствия навигации**  
**(non navigable)**  
изображается с помощью буквы «X» на линии у конца ассоциации

**Кратность конца ассоциации**  
специфицирует возможное количество экземпляров соответствующего класса, которое может соотноситься с одним экземпляром класса на другом конце этой ассоциации

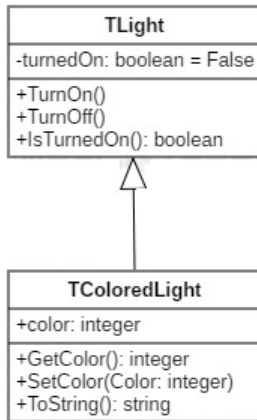
**Видимость конца ассоциации**  
специфицирует возможность доступа к соответствующему концу ассоциации с других ее концов

**Символ наличия навигации**  
**(navigable)**  
изображается с помощью простой стрелки в форме буквы «V» на конце ассоциации

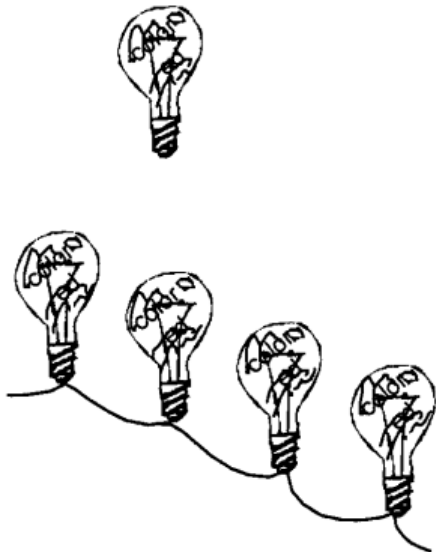
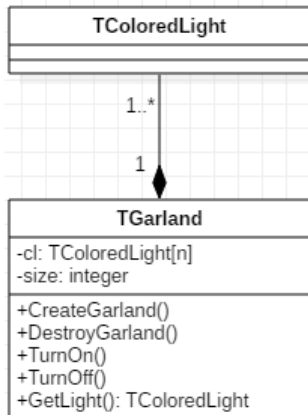
# Отношения классов

- **Отношение обобщения** (generalization) описывает отношение между общей сущностью и ее конкретным воплощением (один класс является специализацией другого класса);
- **Отношение ассоциации** (association) описывает структурное отношение, показывающее, что объекты одного типа некоторым образом связаны с объектами другого типа (находятся в отношении типа "часть/целое").
- **Отношение зависимости** (dependency) описывает отношение использования, при котором изменение в спецификации одного класса может повлиять на класс, его использующий (объекты некоторого класса передаются в качестве аргументов функциям-членам другого класса).
- **Отношение реализации** (realization) - семантическое отношение, при котором класс гарантирует выполнение контракта, определяемого некоторым интерфейсом.

# Отношение обобщения



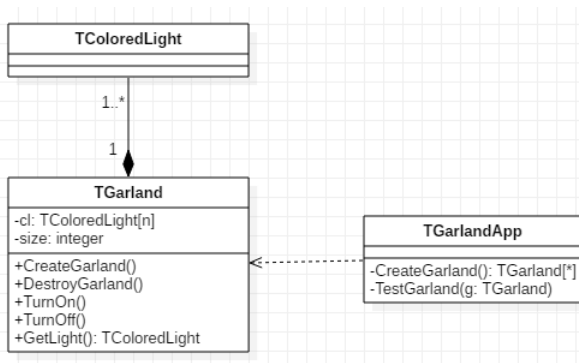
# Отношение ассоциации



# Отношение зависимости

## Отношение зависимости

является таким типом отношений между классами, когда изменение в спецификации или реализации одного класса влияет на спецификацию или реализацию другого класса.



# Отношение реализации

## Отношение реализации

используется для определения отношения между интерфейсом и классом, реализующим интерфейс.

Интерфейс определяет набор элементов (как правило, действий), характерных для объектов, обладающих определенными свойствами.

