# UML

## Class

Сверху всегда пишется имя класса

Под ним идут атрибуты класса (свойства)

Последние — операции (методы)

- закрытые

+ публичные

# защищенные (может использоваться самим классом, всеми суб-классами и всеми классами внутри package

Затем следуют операции (методы класса)

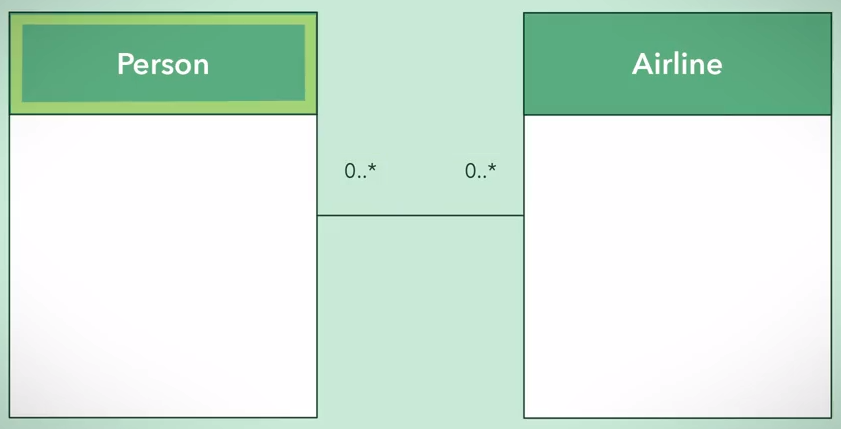
|  |
| --- |
| ClassName |
| Attributes:  -privateAttr  -publicAttr |
| Operations:  +setPublicAttr(String): void  -setPrivateAttr(int):void  +getPublicAttr():String |

# Decomposition

## Acquaintance

*использует, но не является*

Осведомленность — один обьект получает ссылку на другого, но не управляет временем его жизни. Loose partnership — свободное партнерство

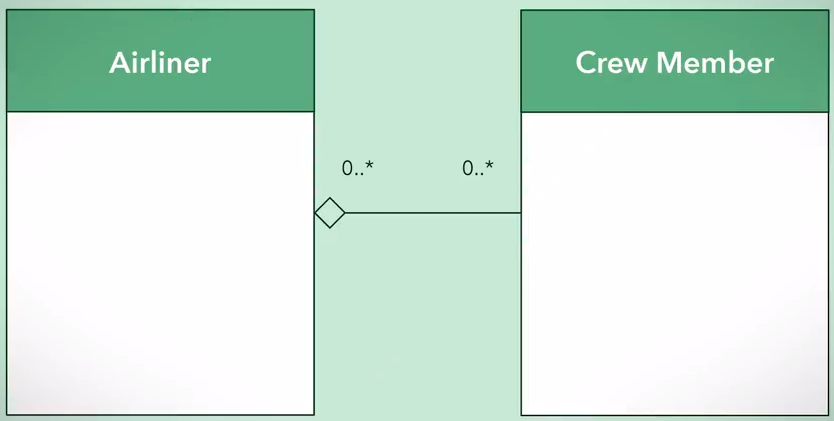


## Delegation

*не является*, *но содержит*.

Частью первого обьекта является второй, но они могут существовать и раздельно. Обычно владелец поручает выполнение операции специалисту — включенному обьекту.

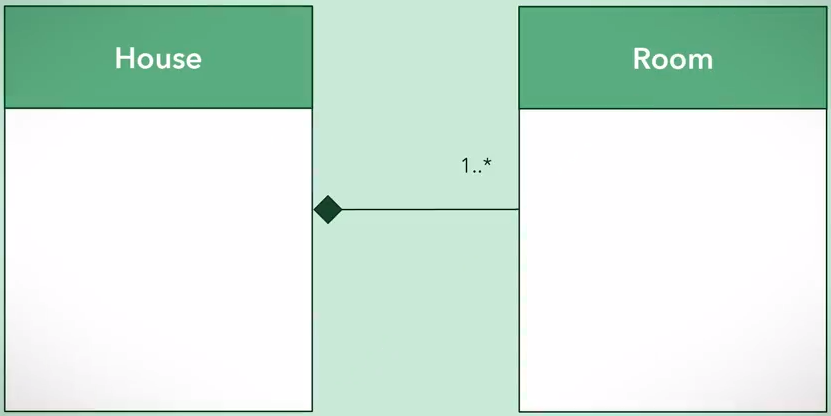
Пустой ромб указывает на обьект-владалец. Другой конец указывает на обьект, который включен во владельца



## Aggregation

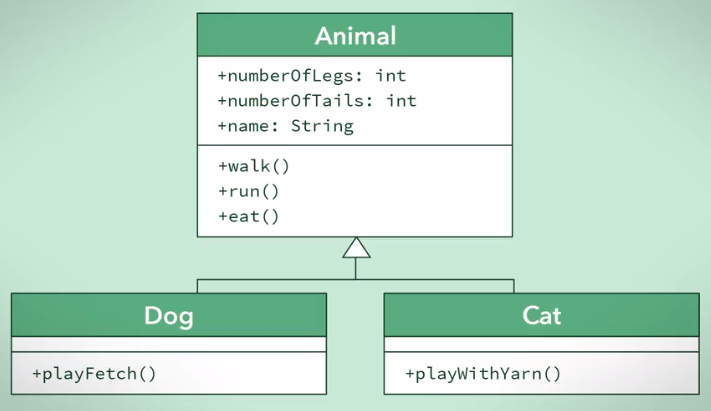
Один обьект является частью другого. Когда внешний обьект, умирает и внутренний.

Закрашенный ромб указывает на обьект-владалец. Другой конец указывает на обьект, который включен во владельца и который будет уничтожен после уничтожения владельца.



## **Generalization**

Вынос повторяющихся кусков отдельно. Вычленение родительского общего класса. На диаграмме указываетяся в форме треугльника, направленного на родительский класс со сплошной линией. Подклассы всегда находятся ниже родительского



**Интерфейс класса**

В Java можно определить класс как интерфейс, который описывает только сигнатуры методов (название метода, параметры на вход и выход), но не включает атрибуты (свойства) или тело методов

Если класс определен только как абстрактный интерфейс, перед его названием ставят «I». Классы, которые наследуют интерфейс, обязаны наследовать его полностью, при этом самостоятельно определить поведение каждого из методов. Один класс может наследовать несколько интрефейсов

На диаграмме указывается в форме треугльника, направленного на родительский класс с пунктирной линией

public interface IPublicSpeaking {

public void givePresentation();

public void speak();

}

public interface IPrivateConversation {

public void lowerVoiceVolume();

public void speak();

}

public class Person implements IPublicSpeaking, IPrivateConversation {

public void speak(){

System.out.println("hello");

}

}

**Sequence diagrams (схемы последовательности)**

Фигура человечка — актер

: Object — указание обьекта, пишется через двоеточие

Вертикальная пунктирная линия — время жизни обьекта

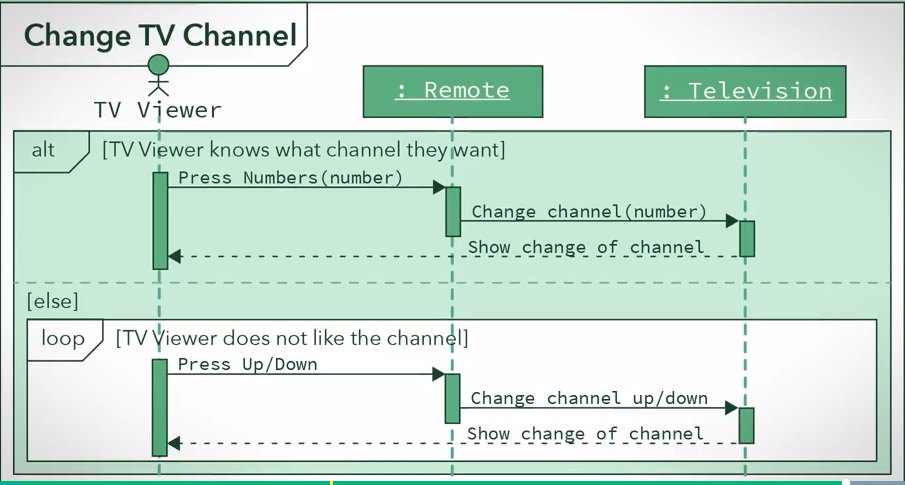
Вертикальный прямоугольник — обьект активен

Горизонтальная линия — взаимодействие обьекта с другими обьектами

На примере условие с циклом:

- юзер знает какой канал хочет смотреть. Он нажимает на пульт, пульт перенаправляет сигнал телевизору, телевизор отображает картинку для юзера

- юзер переключает каналы. Схема та же самая, но указан цикл



**State diagrams**

Может описывать одиночный обьект и иллюстрировать, как обьект реагирует на серию событий в системе

Точка состояния отображается кружком, описание состояния пишется рядом с прямоугольнике с закругленными краями. Прямоугольник поделен на 3 строки:

* название состояния — пишется сверху
* переменные состояния — пишутся посредние
* процессы проходящие в данный момент — в самом конце

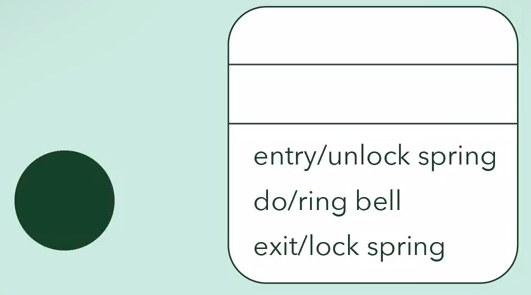
в свою очередь, процессы делятся на 3 вида:

- начальное состояние — процессы которые начинаются, когда текущее состояние стартует

- состояние во время процесса — процессы которые происходят единожды или множество раз, когда обьект в этом состоянии

- состояние на выходе — процессы которые происходят, когда текущее состояние переходит в следующее состояние

*Пример будильника:* когда будильник готовится звенеть, он разблокирует колокольчик, затем звенит, затем блокирует колокольчик



Пример 2: автомат по выдаче шоколадок. Юзер вставляет монетку и получает шоколадку. Так же он может нажать «отмена»

