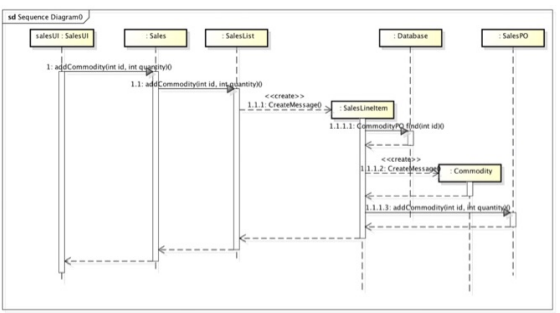
11 面向对象编程五：协作.

一. 类之间的动态协作.

1. 协作

2. 顺序图

- 可以用顺序图表示对象之间的协作。顺序图是交互图的一种,它表达了对象之间如何通过消息的传递来完成比较大的职责。



3. 可协作对象

1) 该对象自身

2) 作为以参数形式传入的对象

3) 被该对象直接创建的对象.

4) 其所持有的对象引用.

4. 协作时数据传递方式.

1) ● B拥有实现职责的所有数据

- 1. A计算（A先去拿B的数据，再计算） 错
- 2. B计算（A直接调用B的计算方法） 对

Example：智能热水器

智能控制水温：周末水温高，夜晚水温低，生病等特殊情况，度假水温低

判断当前是否为特殊时期：

- Controller自己保存特殊时间并计算（比较当前时间和特殊时间）
  - Bad：多个职责。
- 由SpecialTime类保存特殊时间；Controller调用getSpecialTime（）得到特殊时间，再计算
  - Bad：数据职责与行为职责的分离
- 由SpecialTime类保存特殊时间，并提供isSpecialTime（）；Controller调用方法
  - Good：单一职责

(2)

- A和B各拥有实现职责的一部分数据
  - 1. A计算（A先去拿B的数据，再计算）
  - 2. B计算（ A调用B的计算方法，通过参数将数据传给B）

评判标准：

- 看A、B拥有数据的多少 A:B=9:1 vs A:B=1:9
- 看A、B谁拥有职责更合适
- 结构化编程范式偏向1、面向对象编程范式偏向2

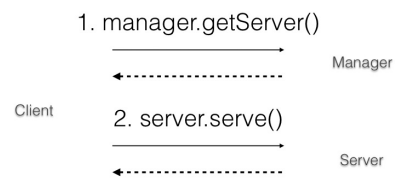
5. 协作对象的角色：

- 从消息传递的角度
  - 客户（Client）
  - 服务器（Server）
  - 经理（Manager）
  - 代理（Delegate）

Client - Server 模式：

Client知道Server

Client - Manager - Server 模式：



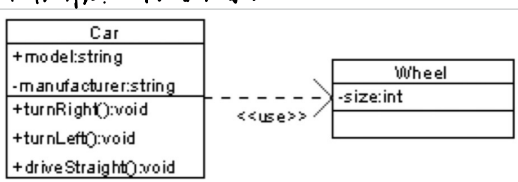
Client必须知道所有的秘密

Client - Delegate - Server 模式：

Client只知道Delgate的存在

二. 类之间的静态关系.

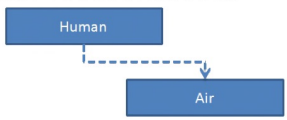
1. 一般关系：依赖.（物理关系）



- Dependencies can exist between other elements than classes. (Requirements, use cases, objects, packages, etc.)

- 关系："... uses a ..."

- 所谓依赖就是某个对象的功能依赖于另外的某个对象，而被依赖的对象只是作为一种工具在使用，而并不持有对它的引用。



Human uses air.

```
class Human
{
   public void breath()
   {
      Air freshAir = new Air();
      freshAir.releasePower();
   }
   public static void main()
   {
      Human me = new Human();
      while(true)
      {
         me.breath();
      }
   }
}
```

```
class Air
{
   public void releasePower()
   {
      //do sth.
   }
}
```

2. 实例层次关系：连接、关联。（逻辑关系）

连接是关联的一个实例。

关联的分类：

(1) 普通关联。

(2) 可导航关联。

(3) 聚合。

(4) 组合。

强弱关系：依赖 < 普通关联 < 聚合 < 组合。

- 关系："... has a ..."

- 所谓关联就是某个对象会长期的持有另一个对象的引用，而二者的关联往往也是相互的。关联的两个对象彼此间没有任何强制性的约束，只要二者同意，可以随时解除关系或是进行关联，它们在生命期问题上没有任何约定。被关联的对象还可以再被别的对象关联，所以关联是可以共享的。

Human has a friend (human).

```
class Human
{
   ArrayList friends = new ArrayList();
   public void makeFriend(Human human)
   {
      friends.add(human);
   }
   public static void main()
   {
      Human me = new Human();
      while(true)
      {
         me.makeFriend(mySchool.getStudent());
      }
   }
}
```

- 普通关联

- 可导航关联

  - 单向

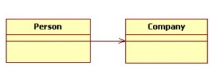
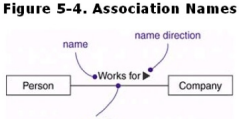
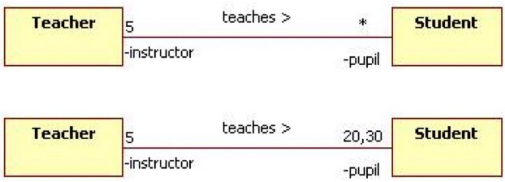


Figure 5-4. Association Names



  - 双向

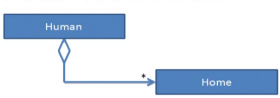
- 多重性（Multiplicity）
  - the number of objects that participate in the association



- 0..1No instances, or one instance (optional, may)
- 1　Exactly one instance
- 0..* or *　Zero or more instances
- 1..* One or more instances (at least one)

聚合：

- 关系："... owns a ..."
- 聚合是强版本的关联。它暗含着一种所属关系以及生命期关系。被聚合的对象还可以再被别的对象关联，所以被聚合对象是可以共享的。虽然是共享的，聚合代表的是一种更亲密的关系。




```
class Human
{
    Home myHome;
    public void goHome()
    {
        //在回家的路上
        myHome.openDoor();
        //看电视
    }
    public static void main()
    {
        Human me = new Human();
        while(true)
        {
            //上学
            //吃饭
            me.goHome();
        }
    }
```

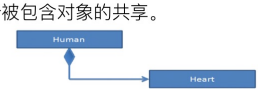
聚合和普通关联不能从代码上区分.

聚合 = 普通关联 + 所属关系

组合 = 聚合 + 生命周期关系.

组合：

- 关系："... is a part of ..."
- 组合是关系当中的最强版本，它直接要求包含对象对被包含对象的拥有以及包含对象与被包含对象生命期的关系。被包含的对象还可以再被别的对象关联，所以被包含对象是可以共享的，然而绝不存在两个包含对象对同一个被包含对象的共享。




```
class Human
{
    Heart myHeart = new Heart();
    public static void main()
    {
        Human me = new Human();
        while(true)
        {
            myHeart.beat();
        }
    }
```

3. 类层次关系：继承，实现.

三. 面向对象的职责与协作.

1. 明确各个对象的职责.

2. 明确各个对象之间的静态关系.

3. 明确动态相互的对象、方法、时机.