继承

实现

依赖：形参的方式，局部变量的方式，静态方法的方式

我只是个司机，别人给我什么车我就开什么车，我使用这个车

关联：车是我自己的车，我“拥有”这个车

聚合：has-a ：雁群与大雁类

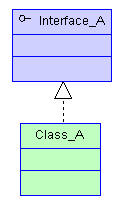
组合：contain-a ：大雁与翅膀类。

## 一、继承关系

      继承指的是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力。在Java中继承关系通过关键字extends明确标识，在设计时一般没有争议性。在UML类图设计中，继承用一条带空心三角箭头的实线表示，从子类指向父类，或者子接口指向父接口。

## http://images.cnitblog.com/blog/420264/201305/03112812-ae0eb58c0ba943fba7eaabc064510e58.jpg 二、实现关系

      实现指的是一个class类实现interface接口（可以是多个）的功能，实现是类与接口之间最常见的关系。在Java中此类关系通过关键字implements明确标识，在设计时一般没有争议性。在UML类图设计中，实现用一条带空心三角箭头的虚线表示，从类指向实现的接口。



## 三、依赖关系

      简单的理解，依赖就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常弱的，但是类B的变化会影响到类A。比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖。表现在代码层面，为类B作为参数被类A在某个method方法中使用。在UML类图设计中，依赖关系用由类A指向类B的带箭头虚线表示。

## http://images.cnitblog.com/blog/420264/201305/03112853-be5da3e87f0b4ce8a466a60ca2df1477.jpg 四、关联关系

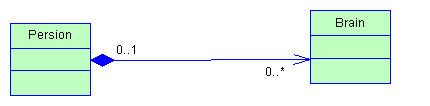
关联体现的是两个类之间语义级别的一种强依赖关系，比如我和我的朋友，这种关系比依赖更强、不存在依赖关系的偶然性、关系也不是临时性的，一般是长期性的，而且双方的关系一般是平等的。关联可以是单向、双向的。表现在代码层面，为被关联类B以类的属性形式出现在关联类A中，也可能是关联类A引用了一个类型为被关联类B的全局变量。在UML类图设计中，关联关系用由关联类A指向被关联类B的带箭头实线表示，在关联的两端可以标注关联双方的角色和多重性标记。

## http://images.cnitblog.com/blog/420264/201305/03112902-e31a5ae618f445918ccb14402ad63dac.jpg 五、聚合关系

      聚合是关联关系的一种特例，它体现的是整体与部分的关系，即has-a的关系。此时整体与部分之间是可分离的，它们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享。比如计算机与CPU、公司与员工的关系等，比如一个航母编队包括海空母舰、驱护舰艇、舰载飞机及核动力攻击潜艇等。表现在代码层面，和关联关系是一致的，只能从语义级别来区分。在UML类图设计中，聚合关系以空心菱形加实线箭头表示。

## http://images.cnitblog.com/blog/420264/201305/03112913-f3b9083cd039432081bc7574d16c035a.jpg 六、组合关系

     组合也是关联关系的一种特例，它体现的是一种contains-a的关系，这种关系比聚合更强，也称为强聚合。它同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束，比如人和人的大脑。表现在代码层面，和关联关系是一致的，只能从语义级别来区分。在UML类图设计中，组合关系以实心菱形加实线箭头表示。

  
七、总结     对于继承、实现这两种关系没多少疑问，它们体现的是一种类和类、或者类与接口间的纵向关系。其他的四种关系体现的是类和类、或者类与接口间的引用、横向关系，是比较难区分的，有很多事物间的关系要想准确定位是很难的。前面也提到，这四种关系都是语义级别的，所以从代码层面并不能完全区分各种关系，但总的来说，后几种关系所表现的强弱程度依次为：组合>聚合>关联>依赖。

在学习面向对象设计对象关系时，依赖、关联、聚合和组合这四种关系之间区别比较容易混淆。特别是后三种，仅仅是在语义上有所区别，所谓语义就是指上下文环境、特定情景等。   
  
  
**依赖(Dependency)**关系是类与类之间的联接。依赖关系表示一个类依赖于另一个类的定义。例如，一个人(Person)可以买车(car)和房子(House)，Person类依赖于Car类和House类的定义，因为Person类引用了Car和House。与关联不同的是，Person类里并没有Car和House类型的属性，Car和House的实例是以参量的方式传入到buy()方法中去的。一般而言，依赖关系在Java语言中体现为局域变量、方法的形参，或者对静态方法的调用。   
  
**关联(Association）**关系是类与类之间的联接，它使一个类知道另一个类的属性和方法。关联可以是双向的，也可以是单向的。在Java语言中，关联关系一般使用成员变量来实现。   
  
**聚合(Aggregation)**关系是关联关系的一种，是强的关联关系。聚合是整体和个体之间的关系。例如，汽车类与引擎类、轮胎类，以及其它的零件类之间的关系便整体和个体的关系。与关联关系一样，聚合关系也是通过实例变量实现的。但是关联关系所涉及的两个类是处在同一层次上的，而在聚合关系中，两个类是处在不平等层次上的，一个代表整体，另一个代表部分。   
  
**组合(Composition)**关系是关联关系的一种，是比聚合关系强的关系。它要求普通的聚合关系中代表整体的对象负责代表部分对象的生命周期，组合关系是不能共享的。代表整体的对象需要负责保持部分对象和存活，在一些情况下将负责代表部分的对象湮灭掉。代表整体的对象可以将代表部分的对象传递给另一个对象，由后者负责此对象的生命周期。换言之，代表部分的对象在每一个时刻只能与一个对象发生组合关系，由后者排他地负责生命周期。部分和整体的生命周期一样。   
  
以上关系的耦合度依次增强（关于耦合度的概念将在以后具体讨论，这里可以暂时理解为当一个类发生变更时，对其他类造成的影响程度，影响越小则耦合度越弱，影响越大耦合度越强）。由定义我们已经知道，依赖关系实际上是一种比较弱的关联，聚合是一种比较强的关联，而组合则是一种更强的关联，所以笼统的来区分的话，实际上这四种关系、都是关联关系。 

* 依赖关系比较好区分，它是耦合度最弱的一种，在java中表现为**局域变量**、方法的**形参**，或者对**静态方法的调用**，如下面的例子：Driver类依赖于Car类，Driver的三个方法分别演示了依赖关系的三种不同形式。

Java代码

1. **class Car {**
2. **public static void run(){**
3. System.out.println("汽车在奔跑");
4. }
5. }
7. **class Driver {**
8. //使用形参方式发生依赖关系
9. **public void drive1(Car car){**
10. car.run();
11. }
12. //使用局部变量发生依赖关系
13. **public void drive2(){**
14. Car car = **new Car();**
15. car.run();
16. }
17. //使用静态变量发生依赖关系
18. **public void drive3(){**
19. Car.run();
20. }
21. }

* **关联关系**在java中一般使用成员变量来实现，有时也用方法形参的形式实现。依然使用Driver和Car的例子，使用方法参数形式可以表示依赖关系，也可以表示关联关系，毕竟我们无法在程序中太准确的表达语义。在本例中，使用成员变量表达这个意思：车是我自己的车，我“拥有”这个车。使用方法参数表达：车不是我的，我只是个司机，别人给我什么车我就开什么车，我使用这个车。

Java代码

1. **class Driver {**
2. //使用成员变量形式实现关联
3. Car mycar;
4. **public void drive(){**
5. mycar.run();
6. }
7. ...
8. //使用方法参数形式实现关联
9. **public void drive(Car car){**
10. car.run();
11. }
12. }

聚合关系是是一种比较强的关联关系，java中一般使用成员变量形式实现。对象之间存在着整体与部分的关系。例如上例中

Java代码

1. **class Driver {**
2. //使用成员变量形式实现聚合关系
3. Car mycar;
4. **public void drive(){**
5. mycar.run();
6. }
7. }

假如给上面代码赋予如下语义：车是一辆私家车，是司机财产的一部分。则相同的代码即表示聚合关系了。**聚合关系一般使用setter方法给成员变量赋值**。   
  
假如赋予如下语义：车是司机的必须有的财产，要想成为一个司机必须要先有辆车，车要是没了，司机也不想活了。而且司机要是不干司机了，这个车就砸了，别人谁也别想用。那就表示组合关系了。一般来说，为了表示组合关系，常常会使用构造方法来达到初始化的目的，例如上例中，加上一个以Car为参数的构造方法

Java代码

1. **public Driver(Car car){**
2. mycar = car;
3. }

所以，关联、聚合、组合只能配合语义，结合上下文才能够判断出来，而只给出一段代码让我们判断是关联，聚合，还是组合关系，则是无法判断的。

雁群类：

**[csharp]** [view plain](http://blog.csdn.net/liushuijinger/article/details/6994265) [copy](http://blog.csdn.net/liushuijinger/article/details/6994265)

1. **public**  **class** GooseGroup
2. {
3. **public** Goose goose;
5. **public** GooseGroup(Goose goose)
6. {
7. **this**.goose = goose;
8. }
9. }

大雁类：

**[csharp]** [view plain](http://blog.csdn.net/liushuijinger/article/details/6994265) [copy](http://blog.csdn.net/liushuijinger/article/details/6994265)

1. **public** **class** Goose
2. {
3. **public** Wings wings;
5. **public** Goose()
6. {
7. wings=**new** Wings();
8. }
9. }