UML 基础: 类图

关于UML 2中结构图的介绍
来自Rational Edge：在 UML 2 中，作为新结构图类型的最重要实例，类图可以在整个软件开发生命周期中，被分析师，业务建模人员，开发者和测试者使用的。本文提供了全面的介绍。

Donald Bell, IBM 全球服务, IBM

2005 年 2 月 15 日

内容

这是关于统一建模语言、即UML 里采用的基本图的一系列文章的一部分。在我 先前关于序列图的文章里，我把重点从 UML 1.4 版，转移到 OMG的采用UML 2.0版草案规范（又称为UML 2）。在这篇文章中，我将会讨论结构图，这是已经在 UML 2 中提出的一种新图种类。由于本系列文章的目的是使人们了解记号元素及它们的含意，该文主要关注类图。你很快就会知道这样做的理由。随后的文章将会覆盖结构范畴中包含的其它图。我也想提醒读者，这一系列文章是关于 UML 记号元素的，所以这些文章并不意味着为建模的最好方式提供指导方针，或是该如何决定哪些内容应该首先被建模。相反的，该文及本系列文章的目的主要是帮助大家对于记号元素 -- 语法和含义有一个基本的理解。借由这些知识，你应该可以阅读图，并使用正确的记号元素创建你自己的图。
*这篇文章假定你对面向对象的设计已经有了基本的理解。你们当中如果有人需要一些面向对象概念的帮助，那么可以访问*http://java.sun.com/docs/books/tutorial/java/concepts/，来获得Sun公司关于面向对象编程的简短指导。*阅读 “什么是类？”和 什么是继承？” 章节，将提供给你足够的理解，并对该文的阅读会有所帮助。另外，David Taylor的书《*Object-Oriented Technologies: A Manager's Guide*》提供了面向对象设计的优秀，高水平的说明，而无需对计算机编程有高深的理解。*

UML 2 中的阴和阳

参考 UML 基础系列的其他文章和教程

在 UML 2 中有二种基本的图范畴：结构图和行为图。每个 UML 图都属于这二个图范畴。结构图的目的是显示建模系统的静态结构。它们包括类，组件和（或）对象图。另一方面，行为图显示系统中的对象的动态行为，包括如对象的方法，协作和活动之类的内容。行为图的实例是活动图，用例图和序列图。

大体上的结构图

如同我所说的，结构图显示建模系统的静态结构。关注系统的元件，无需考虑时间。在系统内，静态结

构通过显示类型和它们的实例进行传播。除了显示系统类型和它们的实例，结构图至少也显示了这些元素间的一些关系，可能的话，甚至也显示它们的内部结构。

贯穿整个软件生命周期，结构图对于各种团队成员都是有用的。一般而言，这些图支持设计验证，和个体与团队间的设计交流。举例来说，业务分析师可以使用类或对象图，来为当前的资产和资源建模，例如分类账，产品或地理层次。架构师可以使用组件和部署图，来测试/确认他们的设计是否充分。开发者可以使用类图，来设计并为系统的代码（或即将成为代码的）类写文档。

特殊的类图

UML 2 把结构图看成一个分类；这里并不存在称为“结构图”的图。然而，类图提供结构图类型的一个主要实例，并为我们提供一组记号元素的初始集，供所有其它结构图使用。由于类图是如此基本，本文的剩余部分将会把重点集中在类图记号集。在本文的结尾，你将对于如何画UML 2类图有所了解，而且对于理解在后面文章中将涉及的其他结构图有一个稳固的基础。

基础

如先前所提到的，类图的目的是显示建模系统的类型。在大多数的 UML 模型中这些类型包括：

- 类
- 接口
- 数据类型
- 组件

UML 为这些类型起了一个特别的名字：“分类器”。通常地，你可以把分类器当做类，但在技术上，分类器是更为普遍的术语，它还是引用上面的其它三种类型为好。

类名

类的 UML 表示是一个长方形，垂直地分为三个区，如图 1 所示。顶部区域显示类的名字。中间的区域列出类的属性。底部的区域列出类的操作。当在一个类图上画一个类元素时，你必须要有顶端的区域，下面的二个区域是可选择的（当图描述仅仅用于显示分类器间关系的高层细节时，下面的两个区域是不必要的）。图 1 显示一个航线班机如何作为 UML 类建模。正如我们所能见到的，名字是 *Flight*，我们可以在中间区域看到Flight类的3个属性：flightNumber，departureTime 和 flightDuration。在底部区域中我们可以看到Flight类有两个操作：delayFlight 和 getArrivalTime。

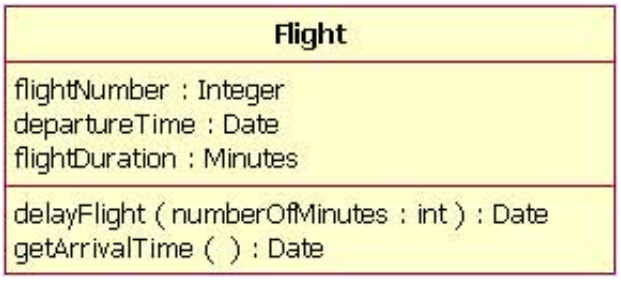

图 1: Flight类的类图类属性列表

类的属性节（中部区域）在分隔线上列出每一个类的属性。属性节是可选择的，要是一用它，就包含类的列表显示的每个属性。该线用如下格式：

> name： attribute type

> flightNumber： Integer

继续我们的Flight类的例子，我们可以使用属性类型信息来描述类的属性，如表 1 所示。

表 1：具有关联类型的Flight类的属性名字

属性名称	属性类型
flightNumber	Integer
departureTime	Date
flightDuration	Minutes

在业务类图中，属性类型通常与单位相符，这对于图的可能读者是有意义的（例如，分钟，美元，等等）。然而，用于生成代码的类图，要求类的属性类型必须限制在由程序语言提供的类型之中，或包含于在系统中实现的、模型的类型之中。

在类图上显示具有默认值的特定属性，有时是有用的（例如，在银行账户应用程序中，一个新的银行账户会以零为初始值）。UML 规范允许在属性列表节中，通过使用如下的记号作为默认值的标识：

> name： attribute type = default value

举例来说：

> balance： Dollars = 0

显示属性默认值是可选择的；图 2 显示一个银行账户类具有一个名为 *balance*的类型，它的默认值为0。

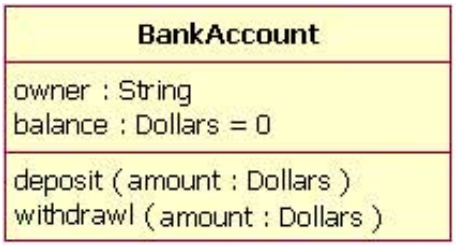

图 2：显示默认为0美元的balance属性值的银行账户类图。类操作列表

类操作记录在类图长方形的第三个（最低的）区域中，它也是可选择的。和属性一样，类的操作以列表格式显示，每个操作在它自己线上。操作使用下列记号表现：

> name(parameter list) : type of value returned

下面的表 2 中Flight类操作的映射。

表 2：从图 2 映射的Flight类的操作操作名称返回参数值类型getArrivalTime N/A Date

图3显示，delayFlight 操作有一个Minutes类型的输入参数 -- numberOfMinutes。然而，delayFlight 操作没有返回值。[1] 当一个操作有参数时，参数被放在操作的括号内；每个参数都使用这样的格式：“参数

名：参数类型”。

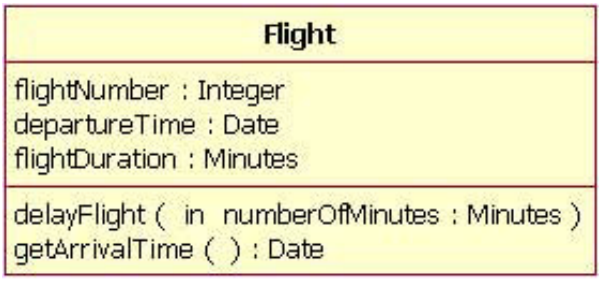

图 3：Flight类操作参数，包括可选择的“in”标识。

当文档化操作参数时，你可能使用一个可选择的指示器，以显示参数到操作的输入参数、或输出参数。这个可选择的指示器以“in”或“out”出现，如图3中的操作区域所示。一般来说，除非将使用一种早期的程序编程语言，如Fortran ，这些指示器可能会有所帮助，否则它们是不必要的。然而，在 C++和Java 中，所有的参数是“in”参数，而且按照UML规范，既然“in”是参数的默认类型，大多数人将会遗漏输入/输出指示器。

继承

在面向对象的设计中一个非常重要的概念，*继承*，指的是一个类（子类）*继承*另外的一个类（超类）的同一功能，并增加它自己的新功能（一个非技术性的比喻，想象我继承了我母亲的一般的音乐能力，但是在我的家里，我是唯一一个玩电吉他的人）的能力。为了在一个类图上建模继承，从子类（要继承行为的类）拉出一条闭合的，单键头（或三角形）的实线指向超类。考虑银行账户的类型：图 4 显示 CheckingAccount 和 SavingsAccount 类如何从 BankAccount 类继承而来。

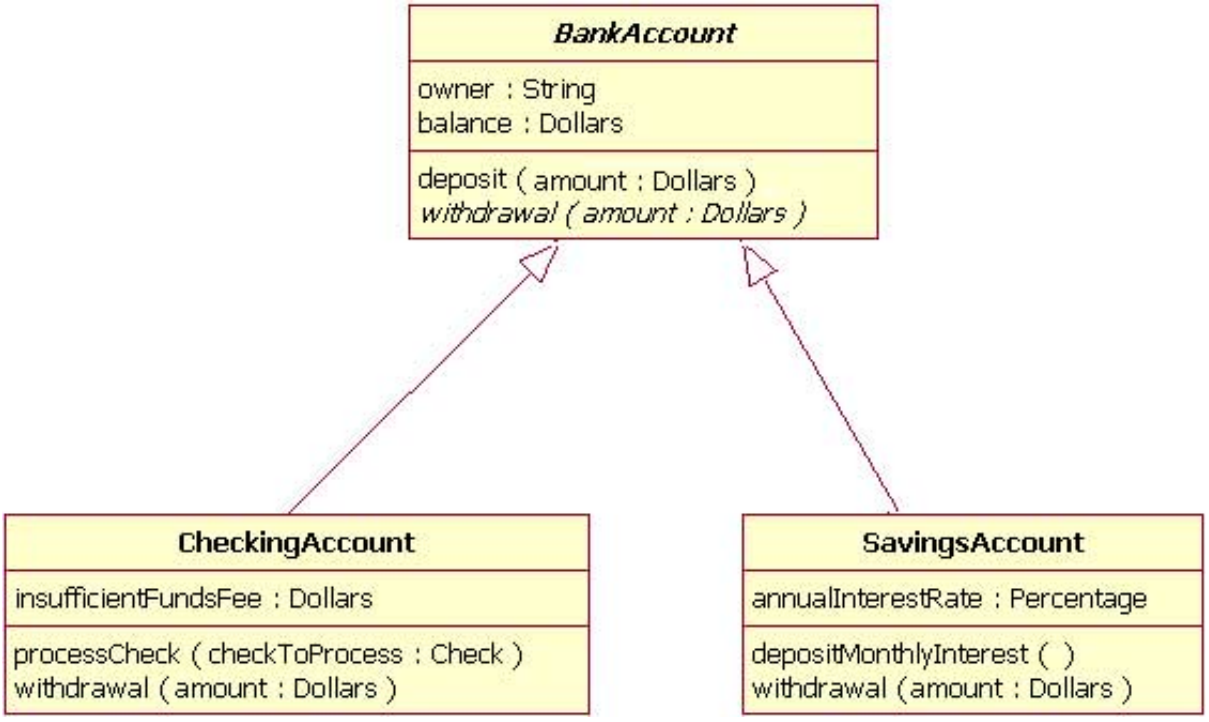

图 4: 继承通过指向超类的一条闭合的，单箭头的实线表示。

在图 4 中，继承关系由每个超类的单独的线画出，这是在IBM Rational Rose和IBM Rational XDE中使用的方法。然而，有一种称为 *树标记*的备选方法可以画出继承关系。当存在两个或更多子类时，如图 4 中所示，除了继承线象树枝一样混在一起外，你可以使用树形记号。图 5 是重绘的与图 4 一样的继承，但是这次使用了树形记号。

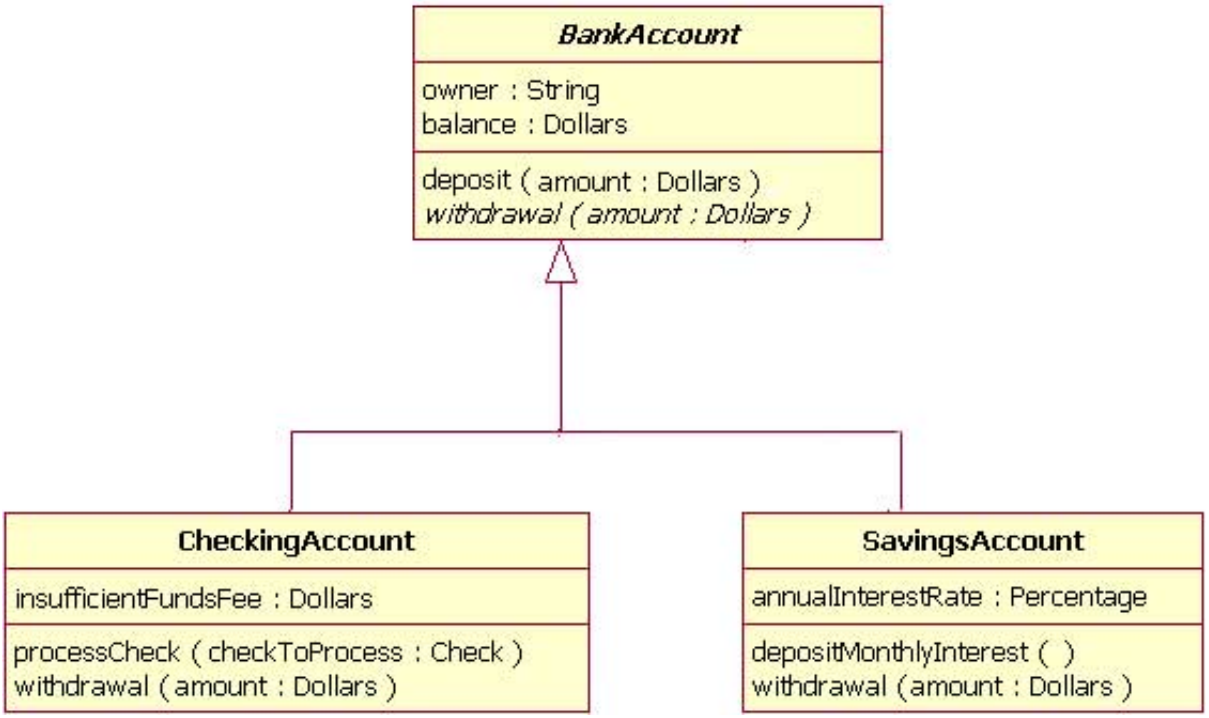

图 5: 一个使用树形记号的继承实例

抽象类及操作

细心的读者会注意到，在图 4 和 图5 中的图中,类名BankAccount和withdrawal操作使用斜体。这表示，BankAccount 类是一个抽象类，而withdrawal方法是抽象的操作。换句话说，BankAccount 类使用 withdrawal规定抽象操作，并且CheckingAccount 和 SavingsAccount 两个子类都分别地执行它们各自版本的操作。

然而，超类（父类）不一定要是抽象类。标准类作为超类是正常的。

关联

当你系统建模时，特定的对象间将会彼此关联，而且这些关联本身需要被清晰地建模。有五种关联。在这一部分中，我将会讨论它们中的两个 -- 双向的关联和单向的关联，而且我将会在*Beyond the basics*部分讨论剩下的三种关联类型。请注意，关于何时该使用每种类型关联的详细讨论，不属于本文的范围。相反的，我将会把重点集中在每种关联的用途，并说明如何在类图上画出关联。

双向（标准）的关联

关联是两个类间的联接。关联总是被假定是双向的；这意味着，两个类彼此知道它们间的联系，除非你限定一些其它类型的关联。回顾一下Flight 的例子，图 6 显示了在Flight类和Plane类之间的一个标准类型的关联。

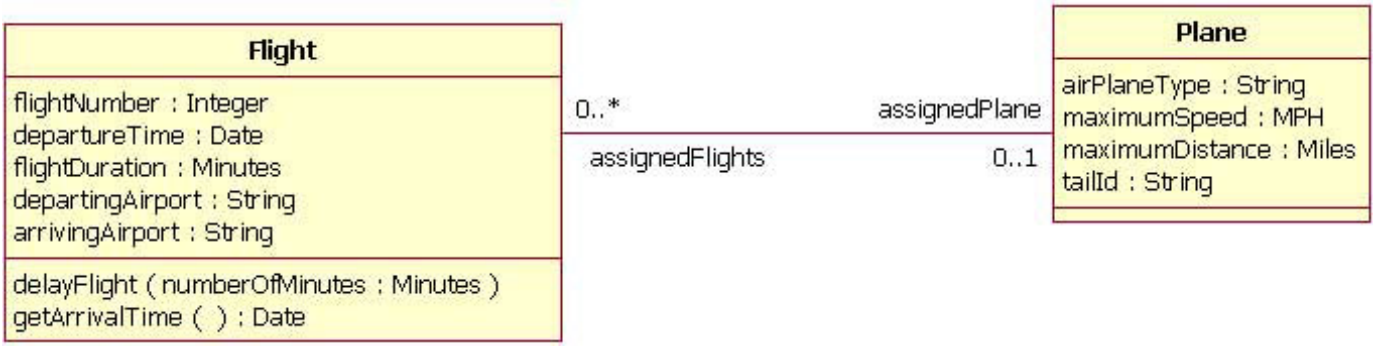

图 6：在一个Flight类和Plane类之间的双向关联的实例

一个双向关联用两个类间的实线表示。在线的任一端，你放置一个角色名和多重值。图 6 显示Flight与一个特定的Plane相关联，而且Flight类知道这个关联。因为角色名以Plane类表示，所以Plane承担关联中的“assignedPlane”角色。紧接于Plane类后面的多重值描述0...1表示，当一个Flight实体存在时，可以有一个或没有Plane与之关联（也就是，Plane可能还没有被分配）。图 6 也显示Plane知道它与Flight类的关联。在这个关联中，Flight承担“assignedFlights”角色；图 6 的图告诉我们，Plane实体可以不与flight关联（例如，它是一架全新的飞机）或与没有上限的flight（例如，一架已经服役5年的飞机）关联。

由于对那些在关联尾部可能出现的多重值描述感到疑惑，下面的表3列出了一些多重值及它们含义的例子。

表 3: 多重值和它们的表示

可能的多重值描述

表示	含义
0..1	0个或1个
1	只能1个
0..*	0个或多个
*	0个或多个
1..*	1个或我个
3	只能3个
0..5	0到5个
5..15	5到15个

单向关联

在一个单向关联中，两个类是相关的，但是只有一个类知道这种联系的存在。图 7 显示单向关联的透支财务报告的一个实例。

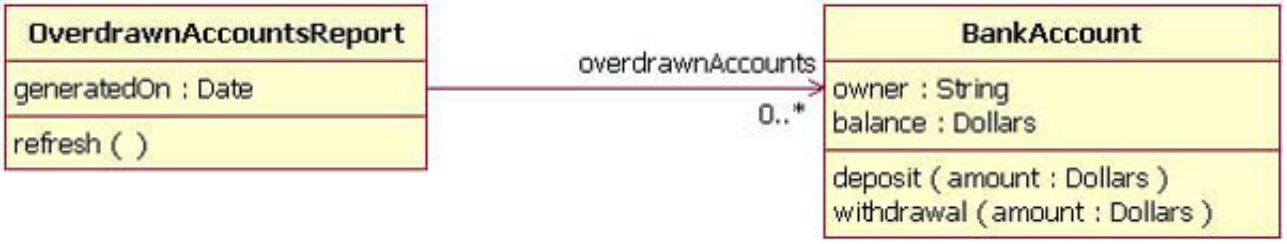

图 7: 单向关联一个实例：OverdrawnAccountsReport 类 BankAccount 类，而 BankAccount 类则对关联一无所知。

一个单向的关联，表示为一条带有指向已知类的开放箭头（不关闭的箭头或三角形，用于标志继承）的实线。如同标准关联，单向关联包括一个角色名和一个多重值描述，但是与标准的双向关联不同的时，单向关联只包含已知类的角色名和多重值描述。在图 7 中的例子中，OverdrawnAccountsReport 知道 BankAccount 类，而且知道 BankAccount 类扮演“overdrawnAccounts”的角色。然而，和标准关联不

同，BankAccount 类并不知道它与 OverdrawnAccountsReport 相关联。[2]

软件包

不可避免，如果你正在为一个大的系统或大的业务领域建模，在你的模型中将会有许多不同的分类器。管理所有的类将是一件令人生畏的任务；所以，UML 提供一个称为 *软件包*的组织元素。软件包使建模者能够组织模型分类器到名字空间中，这有些象文件系统中的文件夹。把一个系统分为多个软件包使系统变成容易理解，尤其是在每个软件包都表现系统的一个特定部分时。[3]

在图中存在两种方法表示软件包。并没有规则要求使用哪种标记，除了用你个人的判断：哪种更便于阅读你画的类图。两种方法都是由一个较小的长方形（用于定位）嵌套在一个大的长方形中开始的，如图 8 所示。但是建模者必须决定包的成员如何表示，如下：

- 如果建模者决定在大长方形中显示软件包的成员，则所有的那些成员[4] 需要被放置在长方形里面。另外，所有软件包的名字需要放在软件包的较小长方形之内（如图 8 的显示）。
- 如果建模者决定在大的长方形之外显示软件包成员，则所有将会在图上显示的成员都需要被置于长方形之外。为了显示属于软件包的分类器属于，从每个分类器画一条线到里面有加号的圆周，这些圆周粘附在软件包之上（图9）。

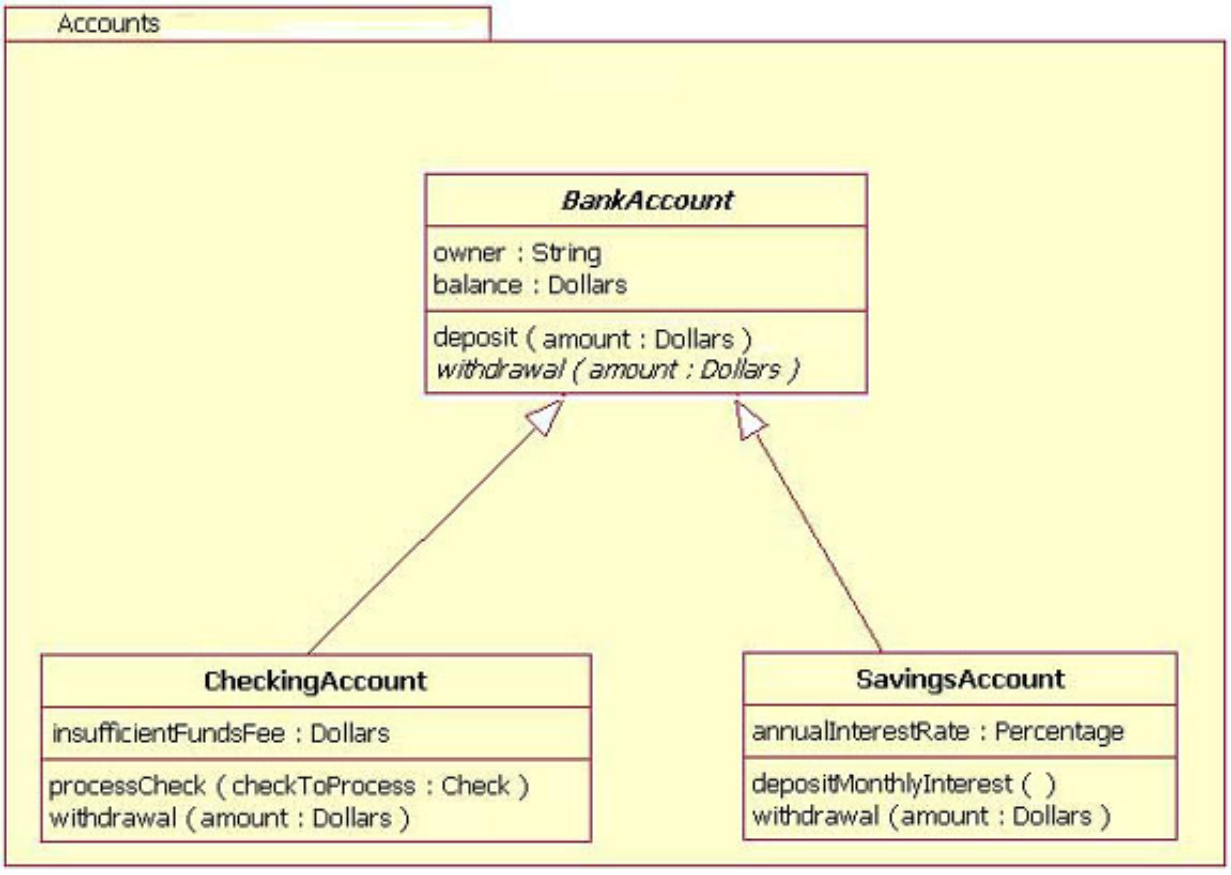

图 8：在软件包的长方形内显示软件包成员的软件包元素例子

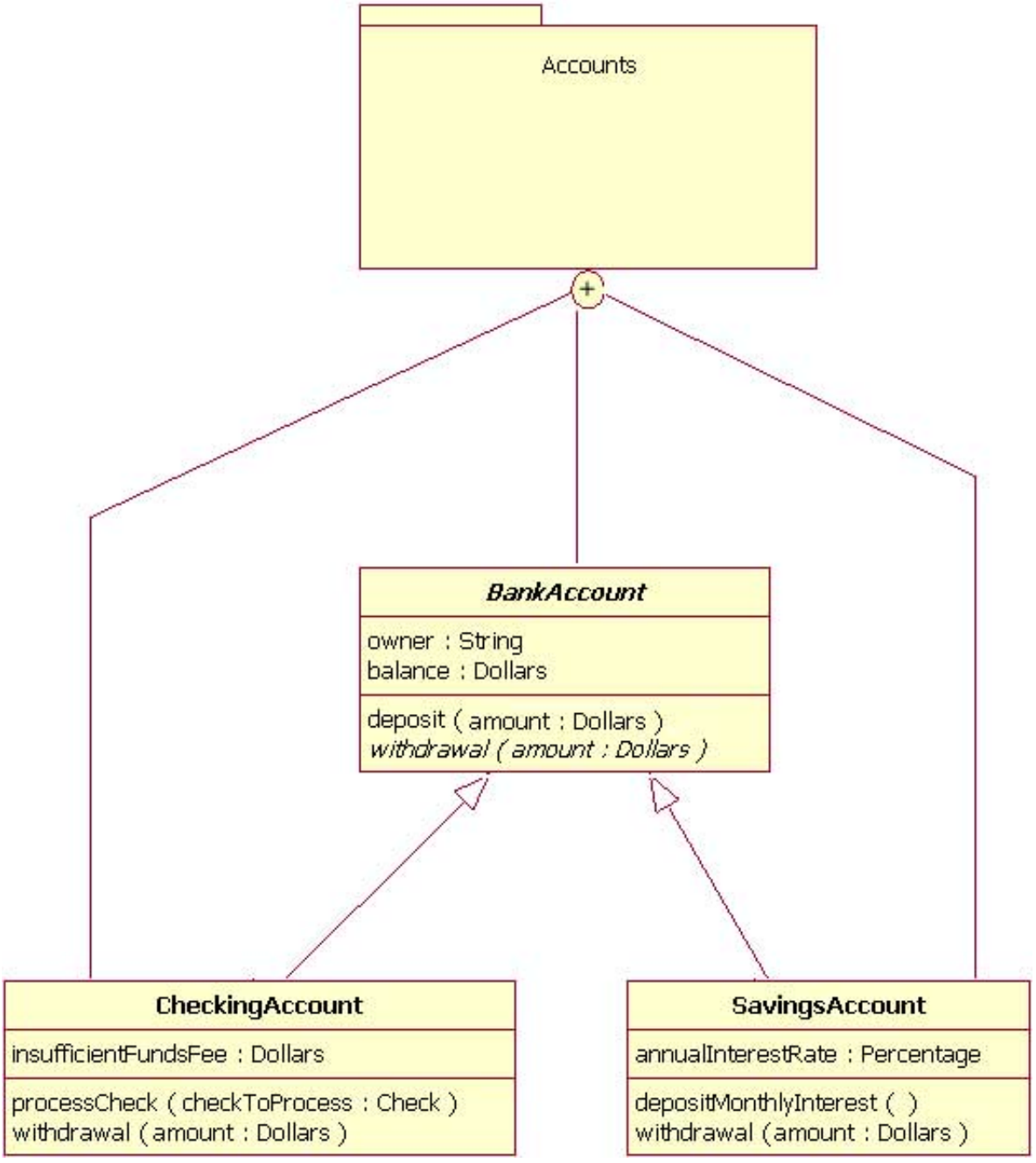

图 9：一个通过连接线表现软件包成员的软件包例子了解基础重要性

在 UML 2 中，了解类图的基础更为重要。这是因为类图为所有的其他结构图提供基本的构建块。如组件或对象图（仅仅是举了些例子）。

超过基础

到此为止，我已经介绍了类图的基础，但是请继续往下读！在下面的部分中，我将会引导你到你会使用的类图的更重要的方面。这些包括UML 2 规范中的接口，其它的三种关联类型，可见性和其他补充。

接口

在本文的前面，我建议你以类来考虑*分类器*。事实上，分类器是一个更为一般的概念，它包括数据类型和接口。

关于何时、以及如何高效地在系统结构图中使用数据类型和接口的完整讨论，不在本文的讨论范围之内。既然这样，我为什么要在这里提及数据类型和接口呢？你可能想在结构图上模仿这些分类器类型，

在这个时候，使用正确的记号来表示，或者至少知道这些分类器类型是重要的。不正确地绘制这些分类器，很有可能将使你的结构图读者感到混乱，以后的系统将不能适应需求。

一个类和一个接口不同：一个类可以有它形态的真实实例，然而一个接口必须至少有一个类来实现它。在 UML 2 中，一个接口被认为是类建模元素的特殊化。因此，接口就象类那样绘制，但是长方形的顶部区域也有文本“interface”，如图 10 所示。[5]

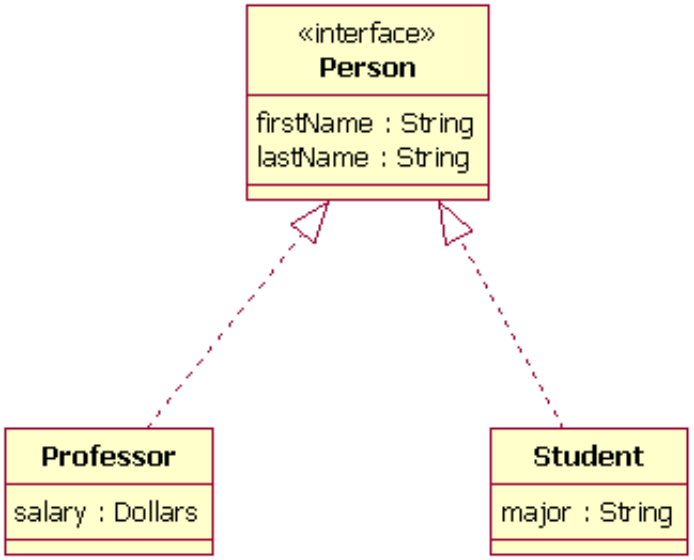

图 10：Professor类和Student类实现Person接口的类图实例

在图 10 中显示的图中，Professor和Student类都实现了Person的接口，但并不从它继承。我们知道这一点是由于下面两个原因：1) Person对象作为接口被定义 -- 它在对象的名字区域中有“interface”文本，而且我们看到由于Professor和Student对象根据画类对象的规则（在它们的名字区域中没有额外的分类器文本）标示，所以它们是 *类*对象。 2) 我们知道继承在这里没有被显示，因为与带箭头的线是点线而不是实线。如图 10 所示，一条带有闭合的单向箭头的*点* 线意味着实现（或实施）；正如我们在图 4 中所见到的，一条带有闭合单向箭头的*实*线表示继承。

更多的关联

在上面，我讨论了双向关联和单向关联。现在，我将会介绍剩下的三种类型的关联。

关联类

在关联建模中，存在一些情况下，你需要包括其它类，因为它包含了关于关联的有价值的信息。对于这种情况，你会使用 *关联类* 来绑定你的基本关联。关联类和一般类一样表示。不同的是，主类和关联类之间用一条相交的点线连接。图 11 显示一个航空工业实例的关联类。

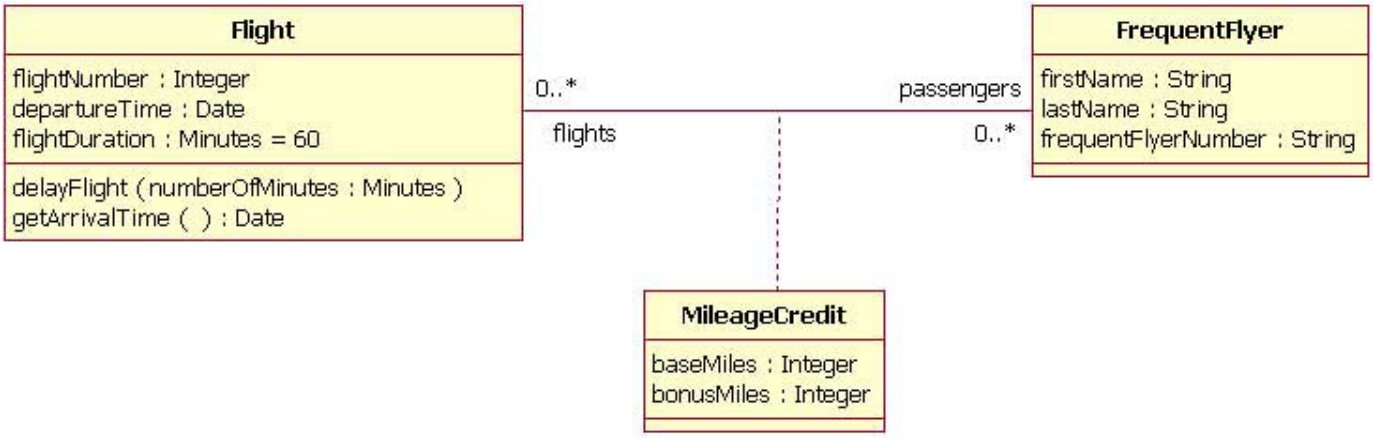

图 11：增加关联类 MileageCredit

在图 11 中显示的类图中，在Flight类和 FrequentFlyer 类之间的关联，产生了称为 MileageCredit的关联类。这意味当Flight类的一个实例关联到 FrequentFlyer 类的一个实例时，将会产生 MileageCredit 类的一个实例。

聚合

聚合是一种特别类型的关联，用于描述“总体到局部”的关系。在基本的聚合关系中， *部分类* 的生命周期独立于 *整体类* 的生命周期。

举例来说，我们可以想象，*车* 是一个整体实体，而 *车轮* 轮胎是整辆车的一部分。轮胎可以在安置到车时的前几个星期被制造，并放置于仓库中。在这个实例中，Wheel类实例清楚地独立地Car类实例而存在。然而，有些情况下， *部分* 类的生命周期并 *不* 独立于 *整体* 类的生命周期 -- 这称为合成聚合。举例来说，考虑公司与部门的关系。 *公司和部门* 都建模成类，在公司存在之前，部门不能存在。这里Department类的实例依赖于Company类的实例而存在。

让我们更进一步探讨基本聚合和组合聚合。

基本聚合

有聚合关系的关联指出，某个类是另外某个类的一部分。在一个聚合关系中，子类实例可以比父类存在更长的时间。为了表现一个聚合关系，你画一条从父类到部分类的实线，并在父类的关联末端画一个未填充棱形。图 12 显示车和轮胎间的聚合关系的例子。

图 12: 一个聚合关联的例子

组合聚合

组合聚合关系是聚合关系的另一种形式，但是子类实例的生命周期依赖于父类实例的生命周期。在图13中，显示了Company类和Department类之间的组合关系，注意组合关系如聚合关系一样绘制，不过这次菱形是被填充的。

图 13: 一个组合关系的例子

在图 13 中的关系建模中，一个Company类实例至少总有一个Department类实例。因为关系是组合关系，当Company实例被移除/销毁时，Department实例也将自动地被移除/销毁。组合聚合的另一个重要功能是部分类只能与父类的实例相关（举例来说，我们例子中的Company类）。

反射关联

现在我们已经讨论了所有的关联类型。就如你可能注意到的，我们的所有例子已经显示了两个不同类之间的关系。然而，类也可以使用反射关联与它本身相关联。起先，这可能没有意义，但是记住，类是抽象的。图 14 显示一个Employee类如何通过manager / manages角色与它本身相关。当一个类关联到它本身时，这并不意味着类的实例与它本身相关，而是类的一个实例与类的另一个实例相关。

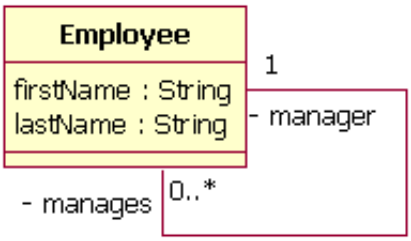

图 14：一个反射关联关系的实例

图 14 描绘的关系说明一个Employee实例可能是另外一个Employee实例的经理。然而，因为“manages”的关系角色有 0..*的多重性描述；一个雇员可能不受任何其他雇员管理。

可见性

在面向对象的设计中，存在属性及操作可见性的记号。UML 识别四种类型的可见性：public，protected，private及package。

UML 规范并不要求属性及操作可见性必须显示在类图上，但是它要求为每个属性及操作定义可见性。为了在类图上的显示可见性，放置可见性标志于属性或操作的名字之前。虽然 UML 指定四种可见性类型，但是实际的编程语言可能增加额外的可见性，或不支持 UML 定义的可见性。表4显示了 UML 支持的可见性类型的不同标志。

表 4：UML 支持的可见性类型的标志

标志	可见性类型
+	Public
#	Protected
-	Private
~	Package

现在，让我们看一个类，以说明属性及操作的可见性类型。在图 15 中，所有的属性及操作都是public，除了 updateBalance 操作。updateBalance 操作是protected。

图 15：一个 BankAccount 类说明它的属性及操作的可见性

UML 2 补充

既然我们已经覆盖了基础和高级主题，我们将覆盖一些由UML 1. x增加的类图的新记号。

实例

当一个系统结构建模时，显示例子类实例有时候是有用的。为了这种结构建模，UML 2 提供 *实例规范* 元素，它显示在系统中使用例子（或现实）实例的值得注意的信息。

实例的记号和类一样，但是取代顶端区域中仅有的类名，它的名字是经过拼接的:

> Instance Name : Class Name

举例来说：

> Donald : Person

因为显示实例的目的是显示值得注意的或相关的信息，没必要在你的模型中包含整个实体属性及操作。相反地，仅仅显示感兴趣的属性及其值是完全恰当的。如图16所描述。

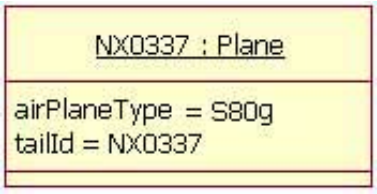

图 16：Plane类的一个实例例子（只显示感兴趣的属性值）

然而，仅仅表现一些实例而没有它们的关系不太实用；因此，UML 2 也允许在实体层的关系/关联建模。绘制关联与一般的类关系的规则一样，除了在建模关联时有一个附加的要求。附加的限制是，关联关系必须与类图的关系相一致，而且关联的角色名字也必须与类图相一致。它的一个例子显示于图 17 中。在这个例子中，实例是图 6 中类图的例子实例。

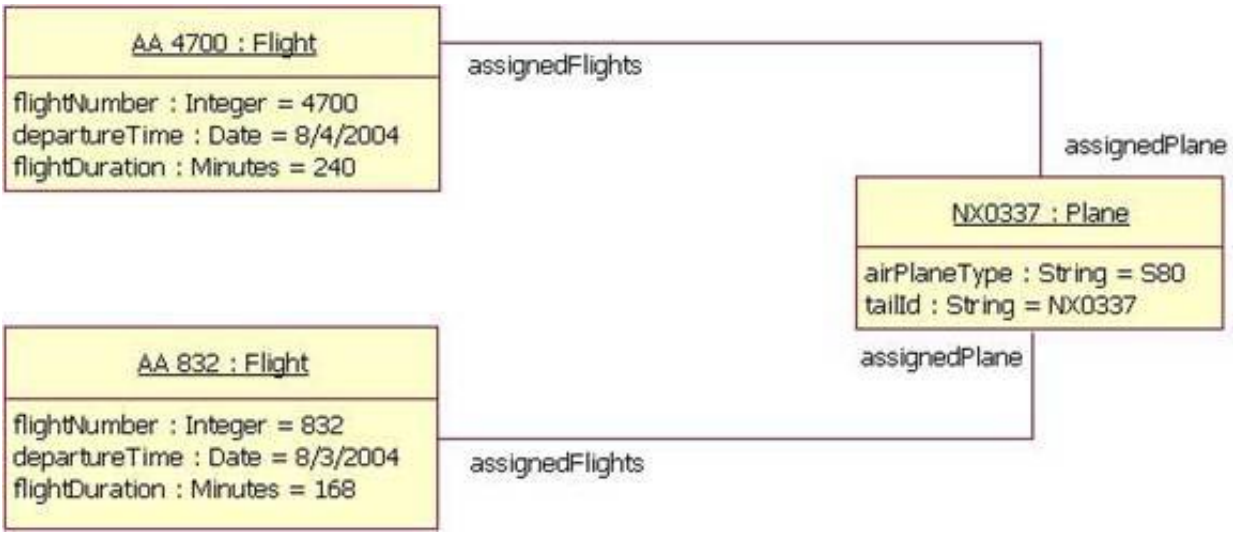

图 17：图 6 中用实例代替类的例子

图 17 有Flight类的二个实例，因为类图指出了在Plane类和Flight类之间的关系是 *0或多*。因此，我们的例子给出了两个与NX0337 Plane实例相关的Flight实例。

角色

建模类的实例有时比期望的更为详细。有时，你可能仅仅想要在一个较多的一般层次做类关系的模型。在这种情况下，你应该使用 *角色* 记号。角色记号类似于实例记号。为了建立类的角色模型，你画一个方格，并在内部放置类的角色名及类名，作为实体记号，但是在这情况你不能加下划线。图 18 显示一个由图 14 中图描述的雇员类扮演的角色实例。在图 18 中，我们可以认为，即使雇员类与它本身相关，关系

确实是关于雇员之间扮演经理及团队成员的角色。

图 18：一个类图显示图14中扮演不同角色的类

注意，你不能在纯粹类图中做类角色的建模，即使图 18显示你可以这么做。为了使用角色记号，你将会需要使用下面讨论的内部结构记号。

内部的结构

UML 2 结构图的更有用的功能之一是新的内部结构记号。它允许你显示一个类或另外的一个分类器如何在内部构成。这在 UML 1. x 中是不可能的，因为记号限制你只能显示一个类所拥有的聚合关系。现在，在 UML 2 中，内部的结构记号让你更清楚地显示类的各个部分如何保持关系。

让我们看一个实例。在图 18 中我们有一个类图以表现一个Plane类如何由四个引擎和两个控制软件对象组成。从这个图中省略的东西是显示关于飞机部件如何被装配的一些信息。从图 18 的图，你无法说明，是每个控制软件对象控制两个引擎，还是一个控制软件对象控制三个引擎，而另一个控制一个引擎。

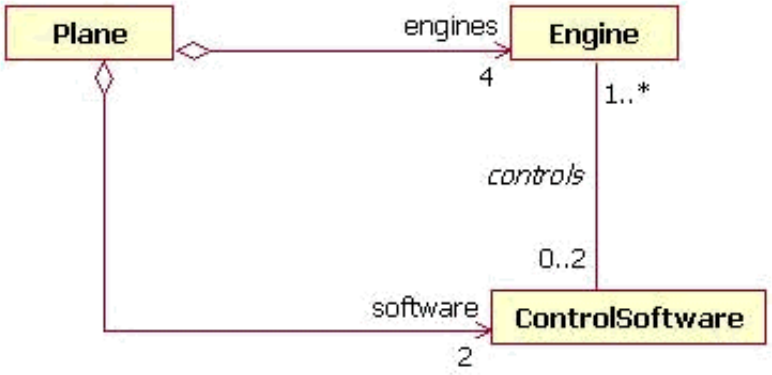

图 19: 只显示对象之间关系的类图

绘制类的内在结构将会改善这种状态。开始时，你通过用二个区域画一个方格。最顶端的区域包含类名字，而较低的区域包含类的内部结构，显示在它们父类中承担不同角色的部分类，角色中的每个部分类也关系到其它类。图 19 显示了Plane类的内部结构；注意内部结构如何澄清混乱性。

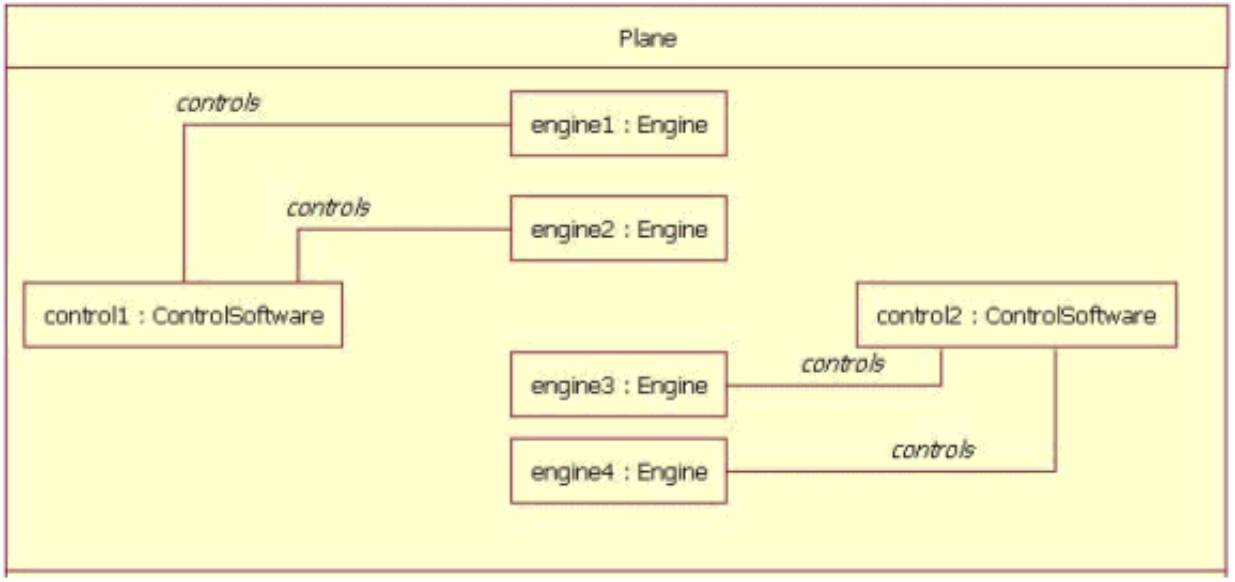

图 20：Plane类的内部结构例子。

在图 20 中Plane有两个 ControlSoftware 对象，而且每个控制二个引擎。在图左边上的 ControlSoftware（control1）控制引擎 1 和 2 。在图右边的 ControlSoftware（control2）控制引擎 3 和 4 。

结论

至少存在两个了解类图的重要理由。第一个是它显示系统分类器的静态结构；第二个理由是图为UML描述的其他结构图提供了基本记号。开发者将会认为类图是为他们特别建立的；但是其他的团队成员将发现它们也是有用的。业务分析师可以用类图，为系统的业务远景建模。正如我们将会在本系列关于 UML 基础的文章中见到的，其他的图 -- 包括活动图，序列图和状态图——参考类图中的类建模和文档化。

关于“UML 基础”的本系列的后面的元件图。

脚注

[1] delayFlight没有返回值，因为我作出了设计决定，不要返回值。有一点可以争论的是，延迟操作应该返回新的到达时间，而且，如果是这种情形，操作属性将显示为 `delayFlight( numberOfMinutes : Minutes) : Date。`

[2]可能看起来很奇怪， BankAccount 类不知道 OverdrawnAccountsReport 类。这个建模使报表类可以知道它们报告的业务类，但是业务类不知道它们正在被报告。这解开两个对象的耦合，并因此使系统变得更能适应变化。

[3] 软件包对于组织你的模型类是庞大的，但是记住重要的一点是，你的类图应该是关于建模系统的容易交流的信息。在你的软件包有许多类的情况下，最好使用多个主题类图，而不是仅仅产生一个大的类图。

[4] 要理解重要一点，当我说“所有的那些成员”时，我仅仅意味着在当前图中的类将显示出来。显示一个有内容的软件包的图，不需要显示它的所有内容。它可以依照一些准则，显示包含元素的子集，这个准则就是并非所有的软件包分类器都是必需的。

[5] 当画一个类图时，在 UML 规范中，全部要做的只是把类放入长方形的顶部区域，而你同理处理接口；然而，UML 规范认为，在这个区域放置“class”文本是可选的，如果类没有显示，那么它应该被假设。