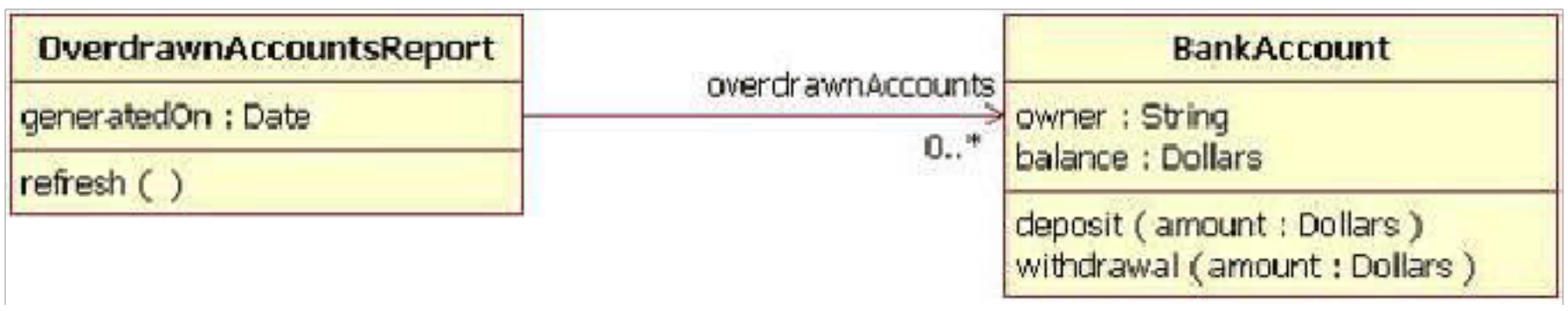
- **单向关联**

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。



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。

简单的说就是 OverdrawAccountReport 中包含了 BankAccount 属性，而 BankAccount 中不需要包含 OverdrawnAccountsReport 对象

6. 聚合的表示：

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 类实例清楚地独立于 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。

注意：聚合与普通的关联的区别在于：普通的关联可能只是一个简单的“包含、引用”关系，关联和被关联类之间在逻辑概念上不一定有紧密的联系，而聚合则不同，它表示的是一种内在关系紧密，相互依存，相互包含的概念，其中的一部分是构成另外一部分的不可或缺的成分。

- **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。

图中清楚的表明了类 Car 对象包含了另一类 Wheel 的 4 个实例，这两者在概念上是密不可分的，其中的一个类是另一个类的构成成分。菱形表示“包含”，箭头表示被包含的对象，数字 4 表示包含的数目。

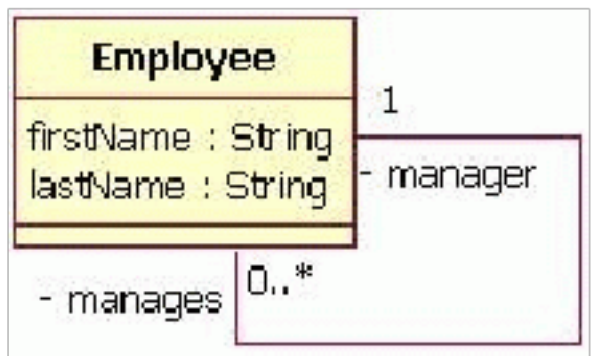
- **组合聚合**

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。

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。

7. 反射关联的表示：

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。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。



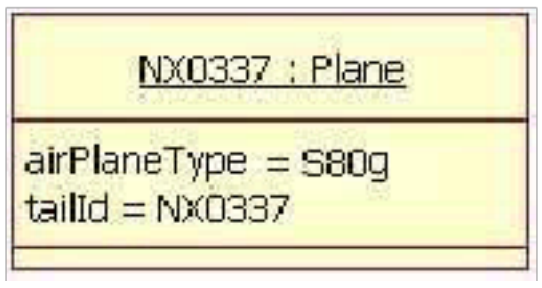
图描绘的关系说明一个 Employee 实例可能是另外一个 Employee 实例的经理。然而，因为“manages”的关系角色有 0..*的多重性描述；一个雇员可能不受任何其他雇员管理。

三、UML 中的对象图：

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 如 Donald ： Person

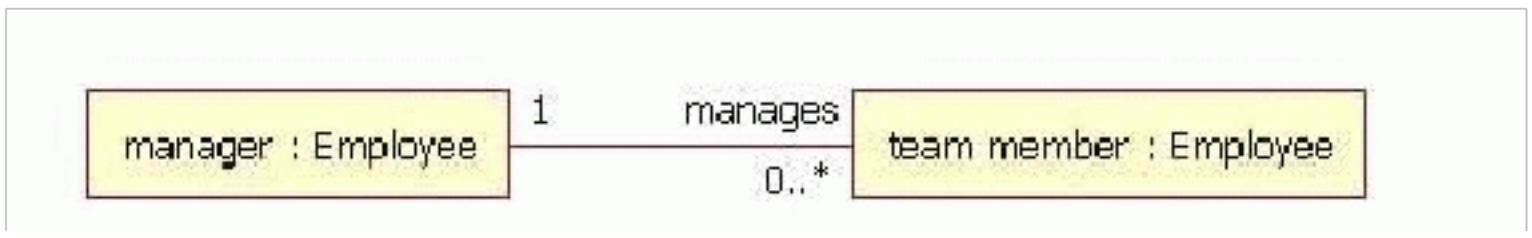
因为显示实例的目的是显示值得注意的或相关的信息，没必要在你的模型中包含整个实体属性及操作。相反地，仅仅显示感兴趣的属性及其值是完全恰当的。



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。

四、UML 中的角色图：

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。



注意：角色图和对象图的一个明显区别就是：对象图每个对象名称下面都加了下划线，而角色图没有

以下是：序列图

序列图主要用于按照交互发生的一系列顺序，显示对象之间的这些交互。很象类图，开发者一般认为序列图只对他们有意义。然而，一个组织的业务人员会发现，序列图显示不同的业务对象如何交互，对于交流当前业务如何进行很有用。除记录组织的当前事件外，一个业务级的序列图能被当作一个需求文件使用，为实现

一个未来系统传递需求。在项目的需求阶段，分析师能通过提供一个更加正式层次的表达，把用例带入下一层次。那种情况下，用例常常被细化为一个或者更多的序列图。

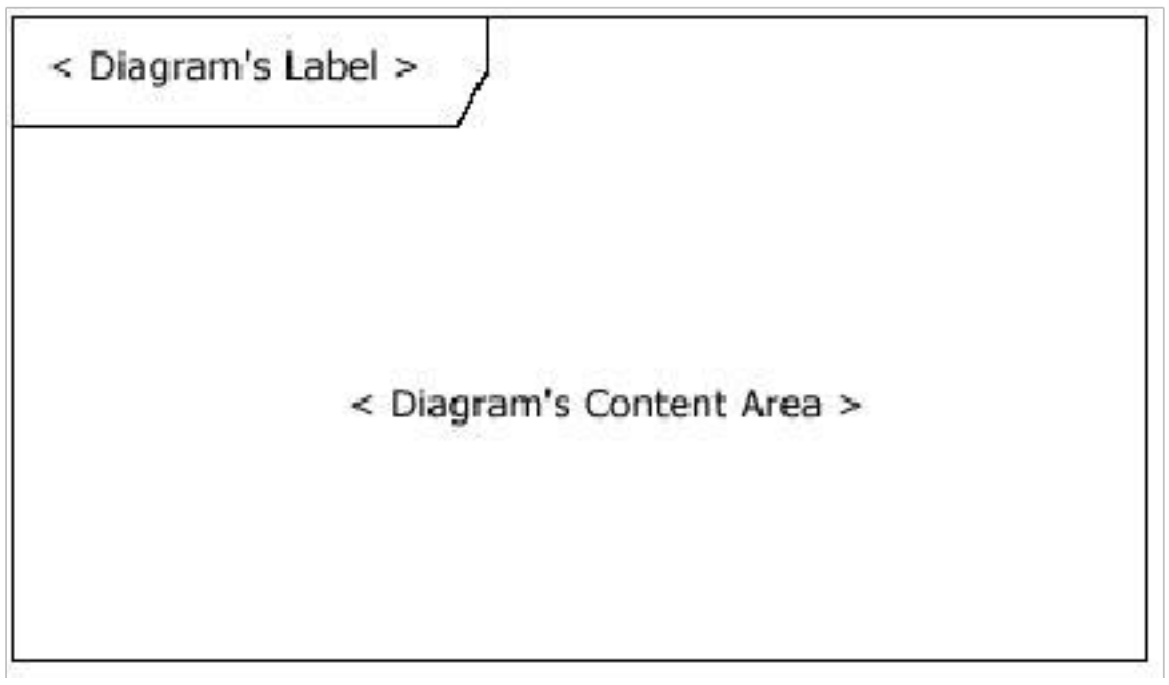
组织的技术人员能发现，序列图在记录一个未来系统的行为应该如何表现中，非常有用。在设计阶段，架构师和开发者能使用图，挖掘出系统对象间的交互，这样充实整个系统设计。

序列图的主要用途之一，是把用例表达的需求，转化为进一步、更加正式层次的精细表达。用例常常被细化为一个或者更多的序列图。序列图除了在设计新系统方面的用途外，它们还能用来记录一个存在系统（称它为“遗产”）的对象现在如何交互。当把这个系统移交给另一个人或组织时，这个文档很有用。

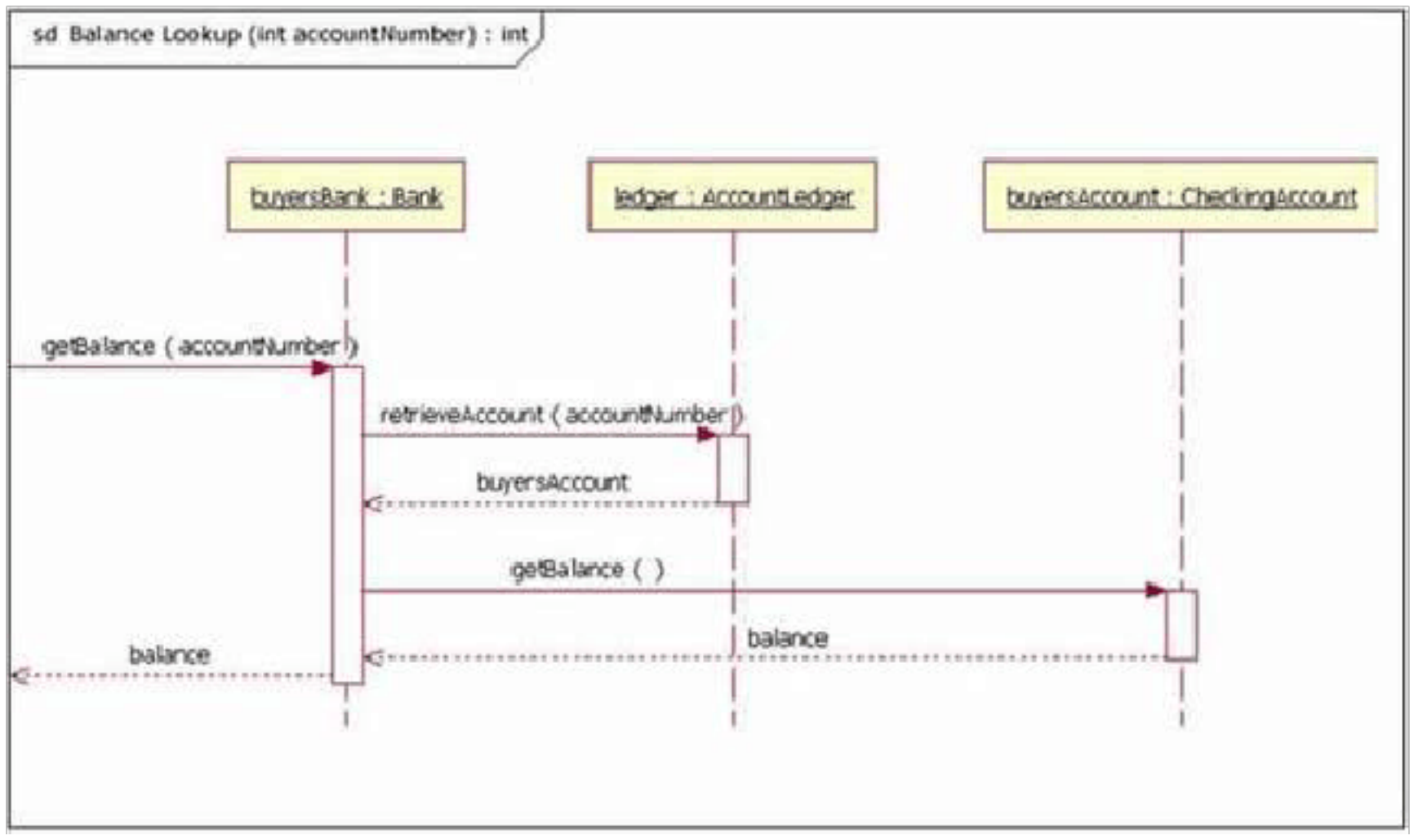
Java 应用程序由许多类所构成，是 Java 实现面向对象应用程序的核心。类图主要描述 Java 应用程序中各种类之间的相互静态关系，如类的继承、抽象、接口以及各种关联。要利用 UML 设计 Java 应用程序，仅仅使用类图来描述这些静态关系，利用可视化工具，要实现 Java 应用程序的代码自动生成，是远远不够的。我们还必须描述各种类相互之间的协作关系、动态关系，如时间序列上的交互行为。其中 UML 序列图就是用来描述类与类之间的方法调用过程（或消息发送）是如何实现的。

一、UML　中的新元素－框架：

在 UML 2 中，框架元件用于作为许多其他的图元件的一个基础，但是大多数人第一次接触框架元件的情况，是作为图的图形化边界。当为图提供图形化边界时，一个框架元件为图的标签提供一致的位置。在 UML 图中框架元件是可选择的。



除了提供一个图形化边框之外，用于图中的框架元件也有描述交互的重要的功能，例如序列图。在序列图上一个序列接收和发送消息（又称交互），能通过连接消息和框架元件边界，建立模型（如图 2 所见到）。



对于序列图，图的标签由文字“sd”开始。当使用一个框架元件封闭一个图时，图的标签需要按照以下的格式：图类型 图名称。

UML 规范给图类型提供特定的文本值。（举例来说，sd 代表序列图，activity 代表活动图，use case 代表用例图）。

二、UML 中的序列图：

序列图主要用于按照交互发生的一系列顺序，显示对象之间的这些交互。

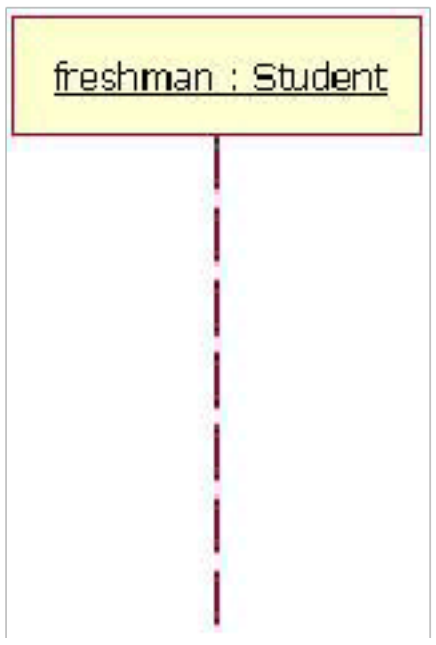
在项目的需求阶段，分析师能通过提供一个更加正式层次的表达，把用例带入下一层次。那种情况下，用例常常被细化为一个或者更多的序列图。

序列图的主要用途之一，是把用例表达的需求，转化为进一步、更加正式层次的精细表达。用例常常被细化为一个或者更多的序列图。序列图除了在设计新系统方面的用途外，它们还能用来记录一个存在系统（称它为“遗产”）的对象现在如何交互。

序列图的主要目的是定义事件序列，产生一些希望的输出。重点不是消息本身，而是消息产生的顺序；不过，大多数序列图会表示一个系统的对象之间传递的什么消息，以及它们发生的顺序。图按照水平和垂直的维度传递信息：垂直维度从上而下表示消息/调用发生的时间序列，而且水平维度从左到右表示消息发送到的对象实例。

1.生命线：

生命线画作一个方格，一条虚线从上而下，通过底部边界的中心（图 3）。生命线名字放置在方格里。



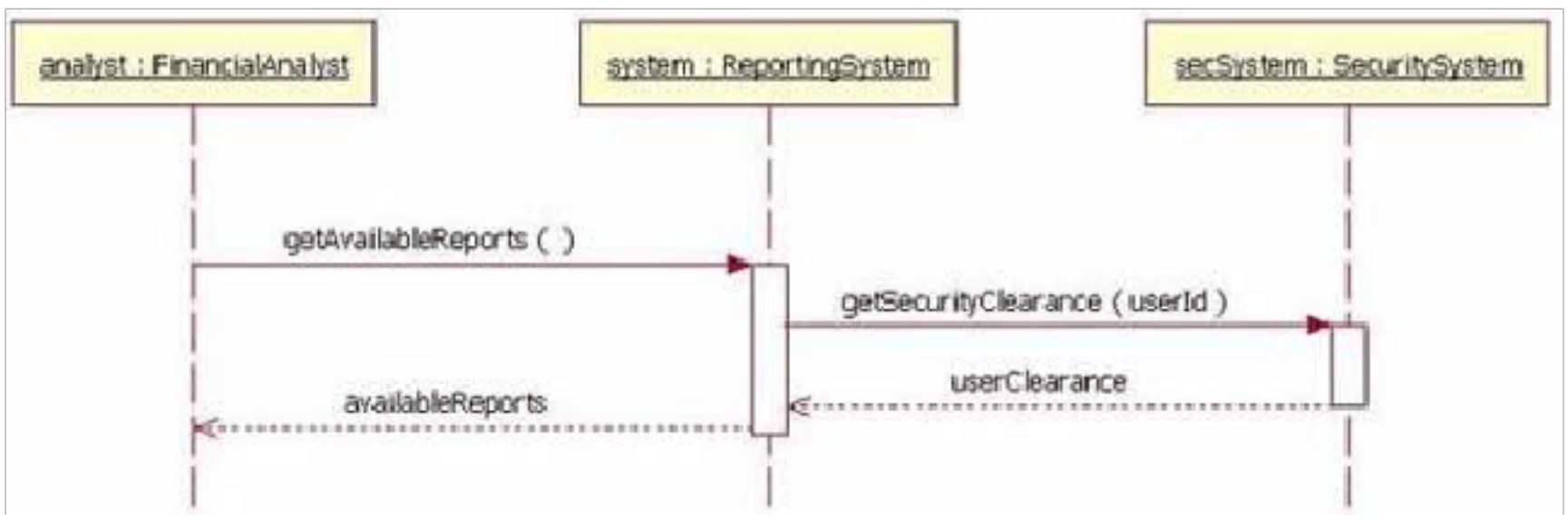
UML 的生命线命名标准按照如下格式：实体名:类名

生命线名称带下划线。当使用下划线时，意味着序列图中的生命线代表一个类的特定实体，不是特定种类的实体（例如，角色）。序列图的实例名称有下划线，而角色名称没有。

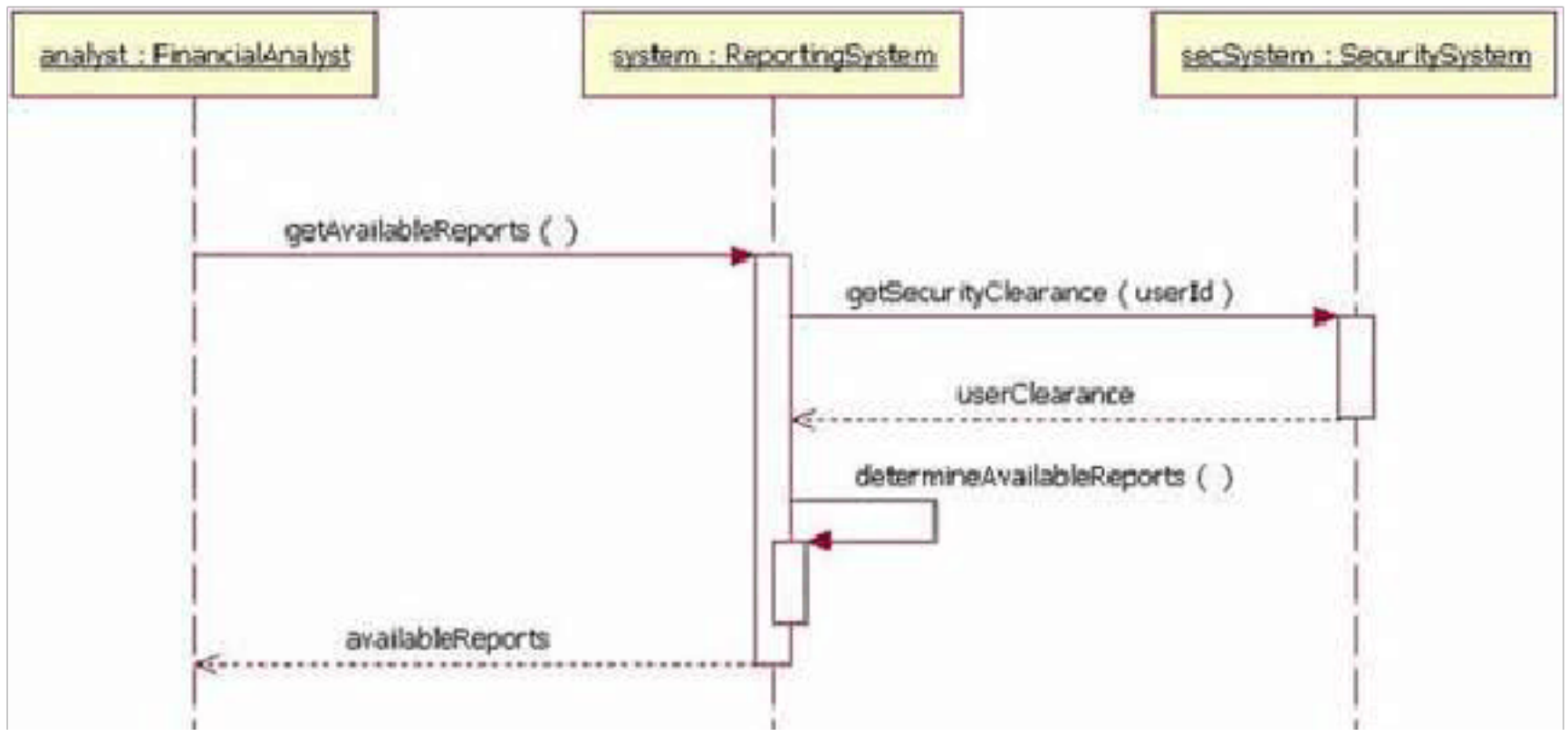
一个生命线能用来表现一个匿名的或未命名的实体。当在一个序列图上，为一个未命名的实例建模时，生命线的名字采用和一个命名实例相同的模式；但是生命线名字的位置留下空白，而不是提供一个例图名字。

2. 消息体：

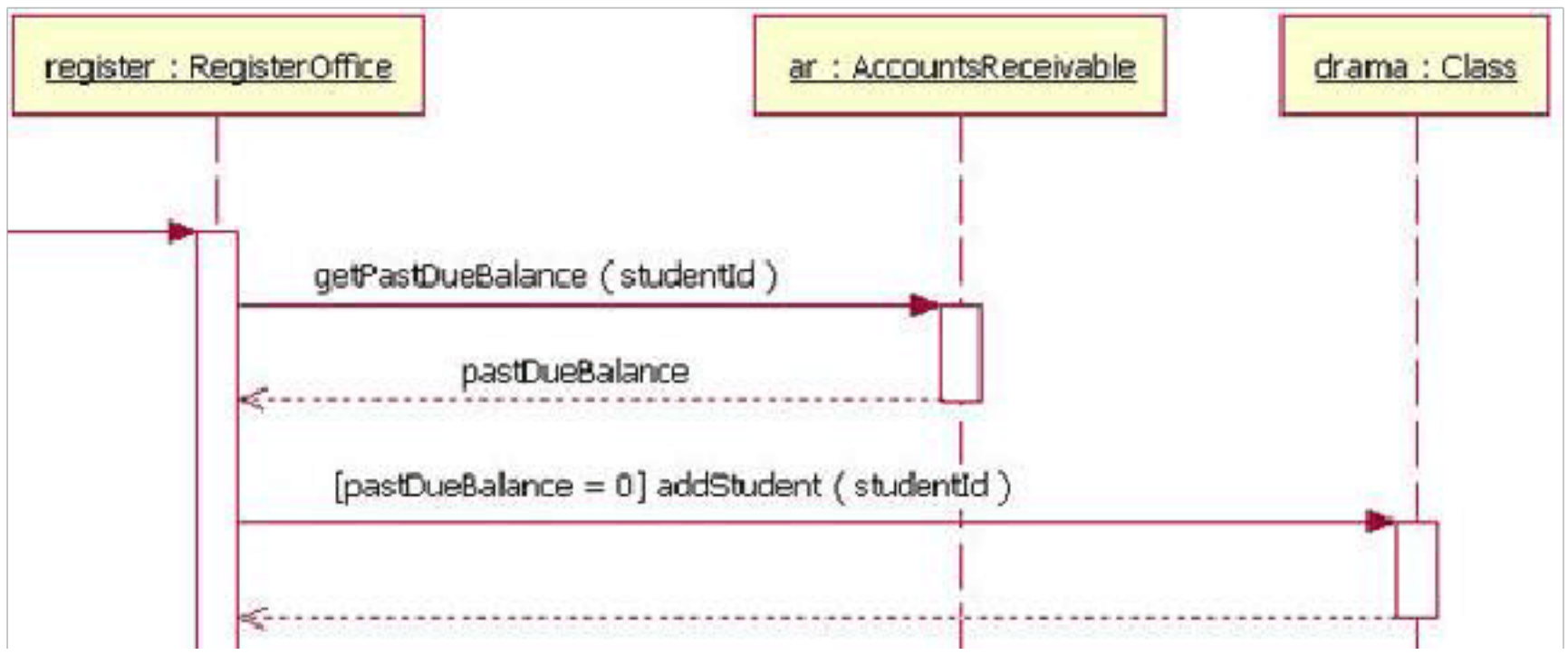
为了显示一个对象（例如，生命线）传递一个消息给另外一个对象，你画一条线指向接收对象，包括一个实心箭头（如果是一个同步调用操作）或一个棍形箭头（如果是一个异步讯号）。消息/方法名字放置在带箭头的线上面。正在被传递给接收对象的消息，表示接收对象的类实现的一个操作/方法。



返回消息是可选择的；一个返回消息画作一个带开放箭头的虚线，向后指向来源的生命线，在这条虚线上面，你放置操作的返回值。为了要画一个调用本身的对象，如你平时所作的，画一条消息，但是不是连接它到另外的一个对象，而是你把消息连接回对象本身。



三、UML 中的约束：



约束的符号很简单；格式是：【Boolean Test】

四、UML 中的新元素－组合碎片（变体方案、选择项、循环）：

一个组合碎片用来把一套消息组合在一起，在一个序列图中显示条件分支。

1. 变体：试卷，计算机走向，据去年进行高中语文，语文试卷，计算机一项调查显示，%高

变体用来指明在两个或更多的消息序列之间的、互斥的选择。一个变体的组合碎片元件使用框架来画。单词“alt”放置在框架的 namebox 里。然后较大的长方形分为 UML 2 所称的操作元。操作元被虚线分开。每个操作元有一个约束进行测试，而这个约束被放置在生命线顶端的操作元的左上部。如果操作元的约束等于“true”，然后那个操作元是要执行的操作元。

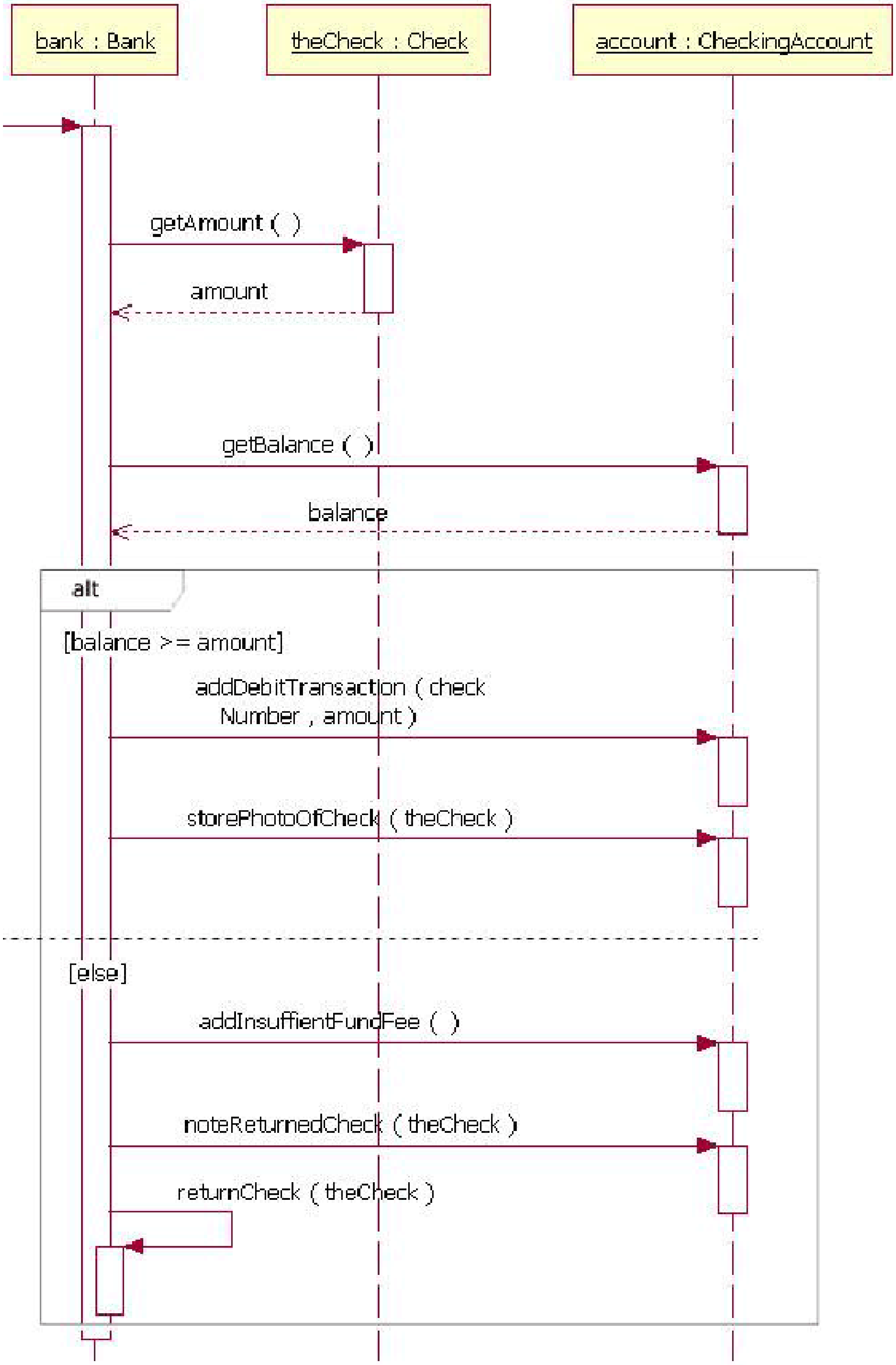


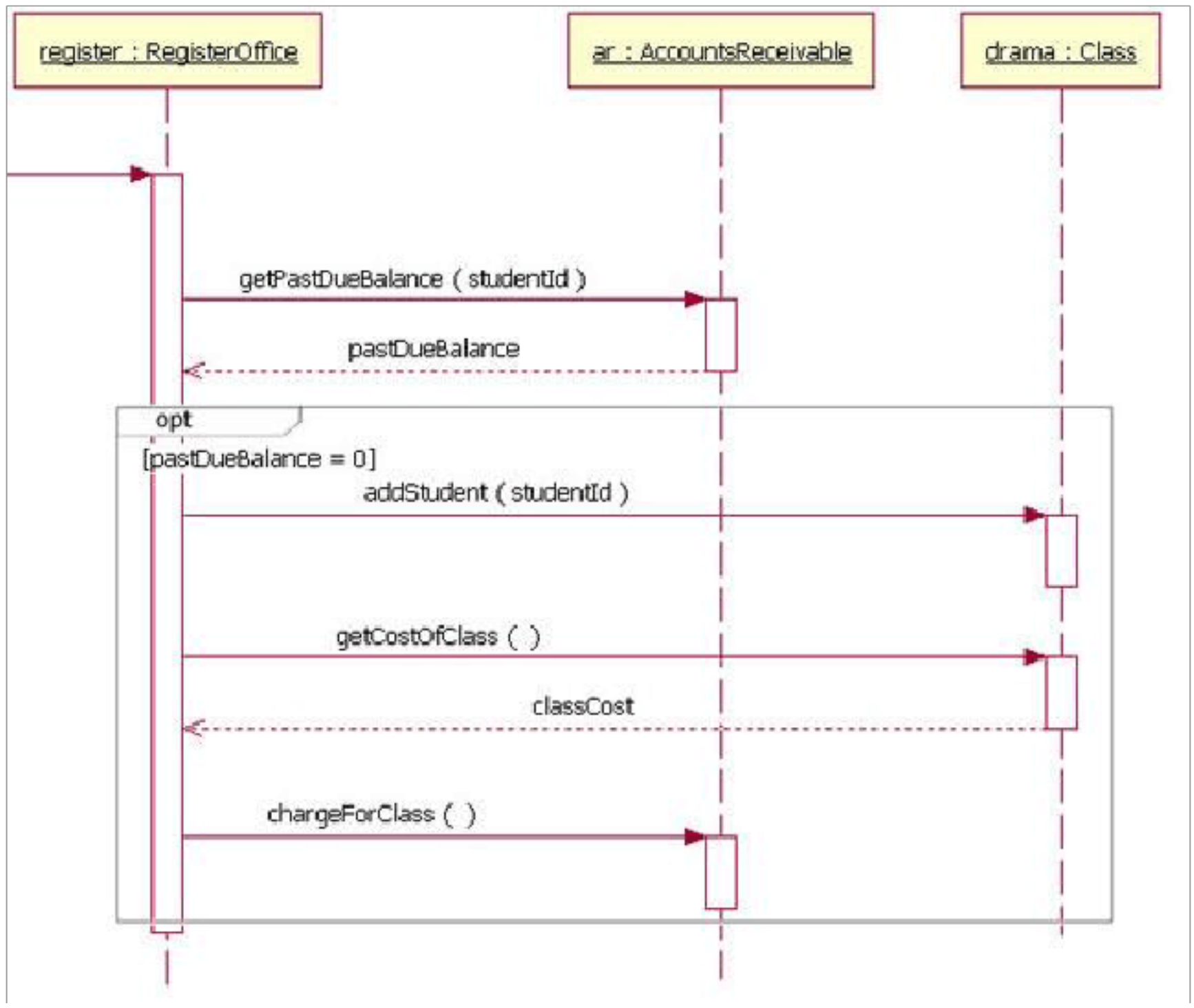
图 8 作为一个变体的组合碎片如何阅读的例子，显示序列从顶部开始，即bank 对象获取支票金额和帐户结余。此时，序列图中的变体组合碎片接管。因为约束“[balance >= amount]”，如果余额超过或等于金额，然后顺序进行 bank 对象传递 addDebitTransaction 和 storePhotoOfCheck 消息给 account 对象。然而，如果余额不是超过或等于金额，然后顺序的过程就是 bank 传递 addInsuffientFundFee 和 noteReturnedCheck 消息给 account 对象，

returnCheck 消息给它自身。因为“else”约束，当余额不大于或者等于金额时，第二个序列被调用。在变体的组合碎片中，不需要“else”约束；而如果一个操作元，在它上面没有一个明确的约束，那么将假定“else”约束。

2. 选择项：

一个选择项用来为简单的“if then”表达式建模。（例如，如果架上的圈饼少于五个，那么另外做两打圈饼）。

选择项组合碎片符号与变体组合碎片类似，除了它只有一个操作元并且永不能有“else”约束以外（它就是如此，没有理由）。要画选择项组合，你画一个框架。文字“opt”是被放置在框架的 namebox 里的文本，在框架的内容区，选择项的约束被放置在生命线顶端上的左上角。 然后选择项的消息序列被放在框架的内容区的其余位置内。



注意：变体用于为 if then else 建模，选择项用于为 if then 建模，因为只有一个分支，所以不能出现[else]

以下是：用例图：

用例图主要用来图示化系统的主事件流程，它主要用来描述客户的需求，即用户希望系统具备的完成一定功能的动作，通俗地理解用例就是软件的功能模块，所以是设计系统分析阶段的起点，设计人员根据客户的需求来创建和解释用例图，用来描述软件应具备哪些功能模块以及这些模块之间的调用关系，用例图包含了用例和参与者，用例之间用关联来连接以求把系统的整个结构和功能反映给非技术人员（通常是软件的用户），对应的是软件的结构和功能分解。

用例是从系统外部可见的行为，是系统为某一个或几个参与者（Actor）提供的一段完整的服务。从原则上来讲，用例之间都是独立、并列的，它们之间并不存在着包含从属关系。但是为了体现一些用例之间的业务关系，提高可维护性和一致性，用例之间可以抽象出包含(include)、扩展(extend)和泛(generalization)几种关系。

共性：都是从现有的用例中抽取出公共的那部分信息，作为一个单独的用例，然后通后过不同的方法来重用这个公共的用例，以减少模型维护的工作量。

1、包含(include)

包含关系：使用包含（Inclusion）用例来封装一组跨越多个用例的相似动作（行为片断），以便多个基（Base）用例复用。基用例控制与包含用例的关系，以及被包含用例的事件流是否会插入到基用例的事件流中。基用例可以依赖包含用例执行的结果，但是双方都不能访问对方的属性。

包含关系对典型的应用就是复用，也就是定义中说的情景。但是有时当某用例的事件流过于复杂时，为了简化用例的描述，我们也可以把某一段事件流抽象成为一个被包含的用例；相反，用例划分太细时，也可以抽象出一个基用例，来包含这些细颗粒的用例。这种情况类似于在过程设计语言中，将程序的某一段算法封装成一个子过程，然后再从主程序中调用这一子过程。

例如：业务中，总是存在着维护某某信息的功能，如果将它作为一个用例，那新建、编辑以及修改都要在用例详述中描述，过于复杂；如果分成新建用例、编辑用例和删除用例，则划分太细。这时包含关系可以用来理清关系。

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/557054044041006063