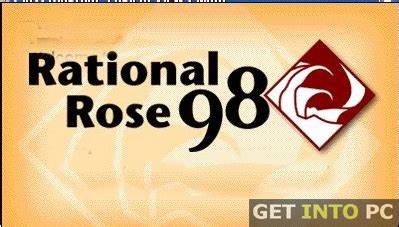
**一、简介**

1. Rational简介：Rational Rose是IBM开发的软件建模工具，以UML图形化表示为核心。本教程将带你全面了解Rational Rose的基础知识和使用方法，包括界面组成、创建项目与模型、绘制UML图表、链接模型与代码、模型验证与分析等。



2、UML的概念与重要性

概念:

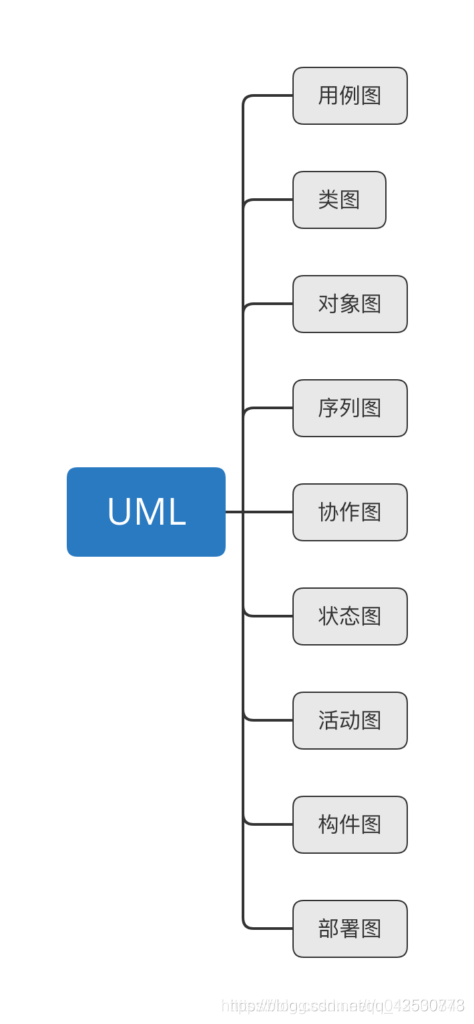
统一建模语言（Unified Modeling Language，UML）是一种标准的图表表示方法，用于分析、设计和实现软件系统。UML 提供了一套图形化的建模语言，可以帮助开发人员和系统架构师可视化系统设计的各个方面。

UML不是编程语言，而是一种用来描述软件蓝图的工具。它将需求分析和系统设计从代码中分离出来，允许设计师构建清晰、准确的模型来描述软件系统的结构和行为。UML的目的是提供一种通用的、标准化的建模方式，确保设计者与开发者之间的沟通不受各自使用工具或编程语言的限制。

重要性（UML在软件工程中的作用）：

UML在软件工程中起到了至关重要的作用。首先，它为软件设计提供了一个共同的语言，使得团队成员，无论专业背景如何，都能围绕模型进行沟通。其次，UML有助于及早发现设计中的问题，因为通过可视化模型，开发人员和利益相关者更容易理解系统的设计意图，并提出改进建议。

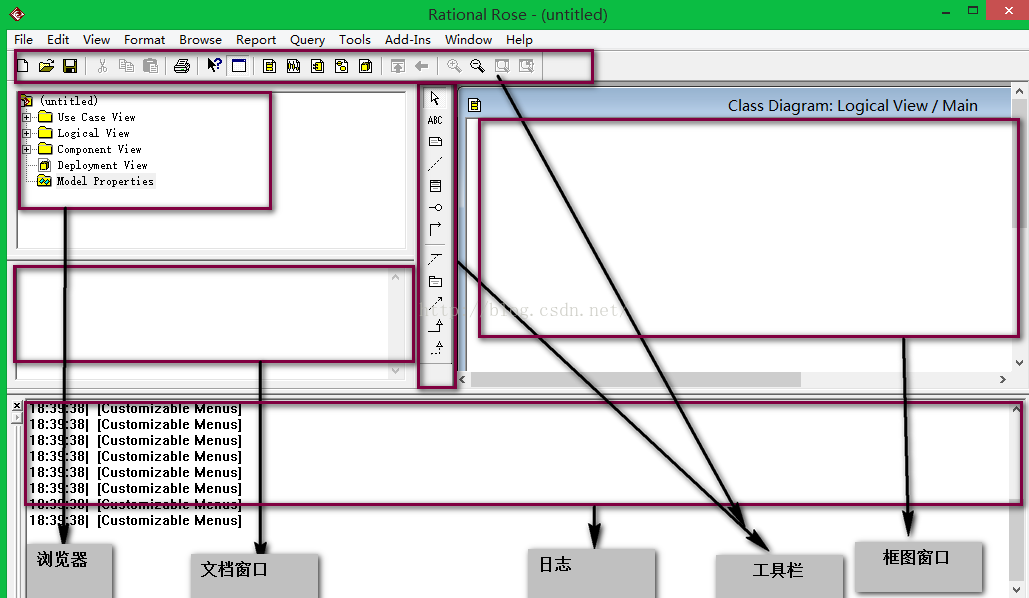
此外，UML也支持软件的迭代和增量开发过程，允许设计者在系统演进过程中逐步完善模型。UML 的这种灵活性使其成为了敏捷开发和传统瀑布模型的共同选择。通过各种图表的辅助，UML 还有助于提高软件的可维护性和可扩展性，从而在产品的整个生命周期内提供价值。



**二.Rational Rose环境简介**

1.1 Rational Rose可视化环境组成

Rose界面的五大部分是浏览器、文档工具、工具栏、框图窗口和日志。



1、浏览器：用于在模型中迅速漫游。

2、文档工具：用于查看或更新模型元素的文档。

3、工具栏：用于迅速访问常用命令。

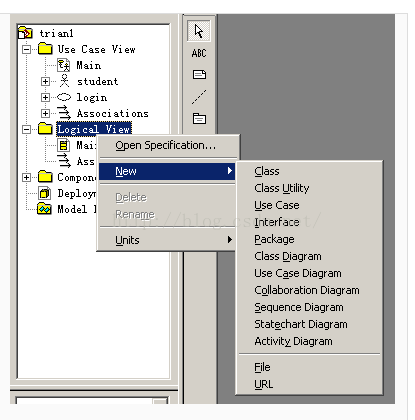
4、框图窗口：用于显示和编辑一个或几个UML框图。

5、日志：用于查看错误信息和报告各个命令的结果。

1.2浏览器和视图

浏览器是层次结构，用于在Rose模型中迅速漫游。在浏览器中显示了模型中增加的一切，如参与者、用例、类、组件等等。

浏览器中包含四个视图：Use Case视图、Logical视图、Component视图和Deployment视图。点击每个视图的右键，选择new就可以看到这个视图所包含的一些模型元素。



1.3框图窗口

我们可以浏览模型中的一个或几个UML框图。改变框图中的元素时，Rose自动更新浏览器。同样用浏览器改变元素时，Rose自动更新相应框图。这样，Rose就可以保证模型的一致性。

**三.UML各类框图的建立（以用例图、活动图和类图为例）**

1.建立用例图use case diagram

从用例图中我们可以看到系统干什么，与谁交互。用例是系统提供的功能，参与者是系统与谁交互，参与者可以是人、系统或其他实体。一个系统可以创建一个或多个用例图。

用例图由参与者(Actor)、用例(Use Case)、系统边界、箭头组成，用画图的方法来完成。

(1）、参与者:

参与者不是特指人，是指系统以外的，在使用系统或与系统交互中所扮演的角色。

因此参与者可以是人，可以是事物，也可以是时间或其他系统等等。

还有一点要注意的是，参与者不是指人或事物本身，而是表示人或事物当时所扮演的角色。

比如小明是图书馆的管理员，他参与图书馆管理系统的交互，这时他既可以作为管理员这个角色参与管理，也可以作为借书者向图书馆借书，在这里小明扮演了两个角色，是两个不同的参与者。

参与者在画图中用简笔人物画来表示，人物下面附上参与者的名称。

参与者的画法：



(2）、用例:

是对包括变量在内的一组动作序列的描述，系统执行这些动作，并产生传递特定参与者的价值的可观察结果。这是UML对用例的正式定义。

我们可以简单的理解为：用例是参与者想要系统做的事情。



(3）、系统边界

系统边界是用来表示正在建模系统的边界。边界内表示系统的组成部分，边界外表示系统外部。系统边界在画图中方框来表示，同时附上系统的名称，参与者画在边界的外面，用例画在边界里面。因为系统边界的作用有时候不是很明显，所以我个人理解，在画图时可省略。

(4）、箭头

箭头用来表示参与者和系统通过相互发送信号或消息进行交互的关联关系。

箭头尾部用来表示启动交互的一方，箭头头部用来表示被启动的一方，其中用例总是要由参与者来启动。

2.USE CASE图的作用：

主要的作用有三个:(1)获取需求;(2)指导测试;(3)还可在整个过程中的其它工作流起到指导作用。

use case图中的关系：

用例图中包含的元素除了系统边界、角色和用例，另外就是关系。

关系包括用例之间的关系，角色之间的关系，用例和角色之间的关系。

(1）、角色之间的关系

由于角色实质上也是类，所以它拥有与类相同的关系描述，即角色之间存在泛化关系，

泛化关系的含义是把某些角色的共同行为提取出来表示为通用的行为。

(2）、用例之间的关系:

包含关系:基本用例图的行为包含了另一个用例的行为。

基本用例描述在多个用例中都有的公共行为。

包含关系本质上是比较特殊的依赖关系。它比一般的依赖关系多了一些语义。在包含关系中箭头的方向是从基本用例到包含用例。在UML1.1中用例之间是使用和扩展这两种关系，这两种关系都是泛化关系的版型。在UML1.3以后的版本中用例之间是包含和扩展这两种关系

(3）、泛化关系:

代表一般与特殊的关系。

它的意思和面向对象程序设计中的继承的概念是类似的。

不同的是继承使用在实施阶段，泛化使用在分析、设计阶段。在泛化关系中子用例继承了父用例的行为和含义，子用例也可以增加新的行为和含义或者覆盖父用例中的行为和含义。

(4）、扩展关系：

扩展关系的基本含义和泛化关系类似，但在扩展USE CASE图关系中，对于扩展用例有更多的规则限制，基本用例必须声明扩展点，而扩展用例只能在扩展点上增加新的行为和含义。与包含关系一样，扩展关系也是依赖关系的版型。在扩展关系中，箭头的方向是从扩展用例到基本用例，这与包含关系是不同的。

(5）、用例的泛化、包含、扩展关系的比较：

一般来说可以使用"is a"和"has a"来判断使用那种关系。

泛化和扩展关系表示用例之间是"is a"关系，包含关系表示用例之间是"has a"关系。

扩展与泛化相比多了扩展点，扩展用例只能在基本用例的扩展点上进行扩展。

在扩展关系中基本用例是独立存在。

在包含关系中在执行基本用例的时候一定会执行包含用例。如果需要重复处理两个或多个用例时可以考虑使用包含关系，实现一个基本用例对另一个的引用。当处理正常行为的变形是偶尔描述时可以考虑只用泛化关系。当描述正常行为的变形希望采用更多的控制方式时，可以USE CASE图在基本用例中设置扩展点，使用扩展关系。

扩展关系比较难理解，如果把扩展关系看作是带有更多规则限制的泛化关系，可以帮助理解。

通常先获得基本用例，针对这个用例中的每一个行为提问:该步骤会出什么差错?该步骤有不同的情况工作怎样以不同的方式进行等，把所有的变化情况都标识为扩展。通常基本用例很容易构造，

而扩展用例需要反复分析、验证。当我们发现已经存在的两个用例间具有某种相似性时，可以把相似的部分从两个用例中抽象出来单独作为一个用例，该用例被这两个用例同时使用，这个抽象出的用例和另外两个用例形成包含关系。

(6）、用例之间的关系举例

1、包含: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。

2、扩展:系统中允许用户对查询的结果进行导出、打印。对于查询而言，能不能导出、打印查询都是一样的，导出、打印是不可见的。导出、打印和查询相对独立，而且为查询添加了新行为。

3、泛化:子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。

(7）、关于用例图的总结

用例图展示了用例之间以及同用例参与者之间是怎样相互联系的。

用例图用于对系统、子系统或类的行为进行可视化，使用户能够理解如何使用这些元素，并使开发者能够实现这些元素。

将每个系统中的用户分出工作状态的属性和工作内容，方便建模，防止功能重复和多余的类。

用例图定义了系统的功能需求，它是从系统的外部看系统功能，并不描述系统内部对功能的具体实现。

3.创建用例图

3.1

1、在浏览器内的Use Case视图中，双击Main，让新的用例图显示在框图窗口中。

2、新建一个包（右击Use Case视图，选择new→package，并命名），然后右击这个新建包的，选择new→use case diagram。

3、对系统总的用例一般画在Use Case视图中的Main里，如果一个系统可以创建多个用例图，则可以用包的形式来组织。

3.2创建参与者

(1) 在工具栏中选择“Actor”，光标的形状变成加号。

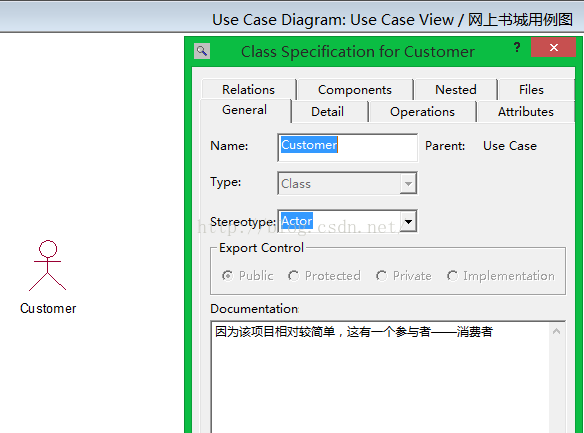
(2) 在用例图中要放置参与者符号的地方单击鼠标左键，键入新参与者的名称，如“客户”。

若要简要的说明参与者，可以执行以下步骤：

(1) 在用例图或浏览器中双击参与者符号，打开对话框，而且已将原型(stereotype)设置定义为“Actor”。

(2) 打开“General”选项卡，在documentation字段中写入该参与者的简要说明。

(3) 单击OK按钮，即可接受输入的简要说明并关闭对话框。



3.3创建用例

(1) 在工具栏中选择“Use Case”，光标的形状变成加号。

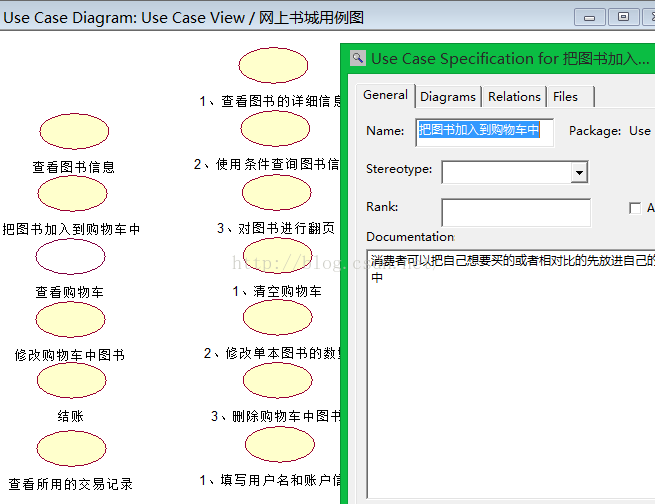
(2) 在用例图中要放置用例符号的地方单击鼠标左键，键入新用例的名称，如“存款”。

若要简要的说明用例，可以执行以下步骤：

(1) 在用例图或浏览器中双击用例符号，打开对话框，接着打开“General”选项卡。

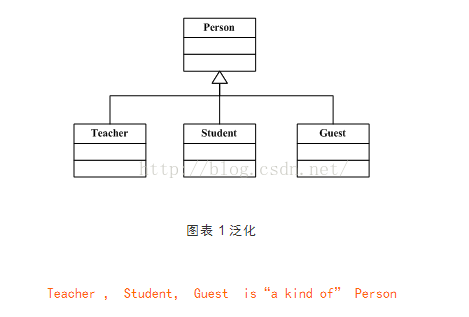
(2) 在documentation字段中写入该用例的简要说明。

(3) 单击OK按钮，即可接受输入的简要说明并关闭对话框。



**4.概念讲解**

4.1、泛化（Generalization）



在上图中，空心的三角表示继承关系（类继承），在UML的术语中，这种关系被称为泛化（Generalization）。Person(人)是基类，Teacher(教师)、Student(学生)、Guest(来宾)是子类。

若在逻辑上B是A的“一种”，并且A的所有功能和属性对B而言都有意义，则允许B继承A的功能和属性。

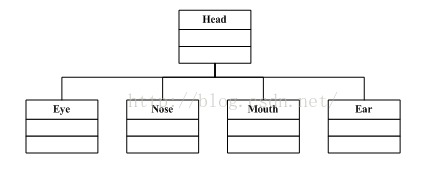
例如，教师是人，Teacher 是Person的“一种”（a kind of ）。那么类Teacher可以从类Person派生（继承）。

如果A是基类，B是A的派生类，那么B将继承A的数据和函数。

如果类A和类B毫不相关，不可以为了使B的功能更多些而让B继承A的功能和属性。

若在逻辑上B是A的“一种”（a kind of ），则允许B继承A的功能和属性。

4.2、聚合（组合）

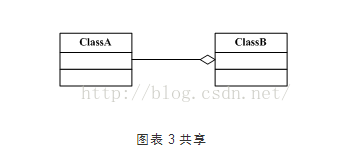


若在逻辑上A是B的“一部分”（a part of），则不允许B从A派生，而是要用A和其它东西组合出B。

例如，眼（Eye）、鼻（Nose）、口（Mouth）、耳（Ear）是头（Head）的一部分，所以类Head应该由类Eye、Nose、Mouth、Ear组合而成，不是派生（继承）而成。

聚合的类型分为无、共享(聚合)、复合(组合)三类。

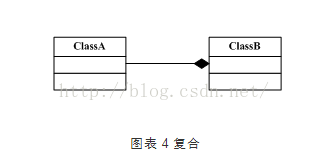
4.4、聚合（aggregation）



上面图中，有一个菱形（空心）表示聚合（aggregation）（聚合类型为共享），

聚合的意义表示has-a关系。聚合是一种相对松散的关系，聚合类B不需要对被聚合的类A负责。

4.5组合（composition）

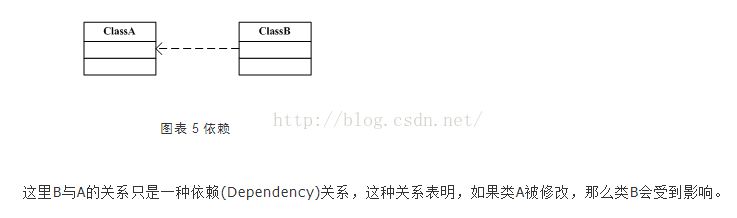


这幅图与上面的唯一区别是菱形为实心的，

它代表了一种更为坚固的关系——组合（composition）（聚合类型为复合）。

组合表示的关系也是has-a，不过在这里，A的生命期受B控制。即A会随着B的创建而创建，随B的消亡而消亡。

4.6、依赖(Dependency)

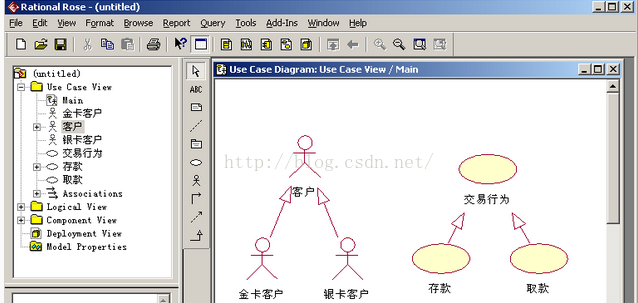


4.7.增加泛化关系

(1) 从工具栏中选择泛化关系箭头。

(2) 从子用例拖向父用例，也可从子参与者拖向父参与者。

简要说明关系执行的步骤同上类似。

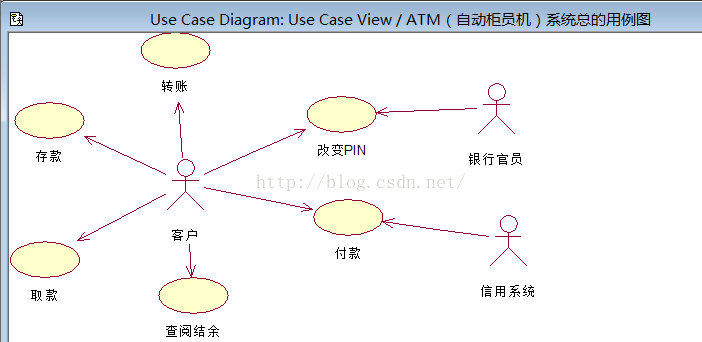


★练习：画ATM（自动柜员机）系统总的用例图

理解：

对于银行的客户来说:可以通过ATM机启动几个用例：存款、取款、查阅结余、付款、转帐和改变PIN（密码）。

银行官员也可以启动改变PIN这个用例。参与者可能是一个系统，这里信用系统就是一个参与者，因为它是在ATM系统之外的。箭头从用例到参与者表示用例产生一些参与者要使用的信息。



5建立活动图 activity diagram

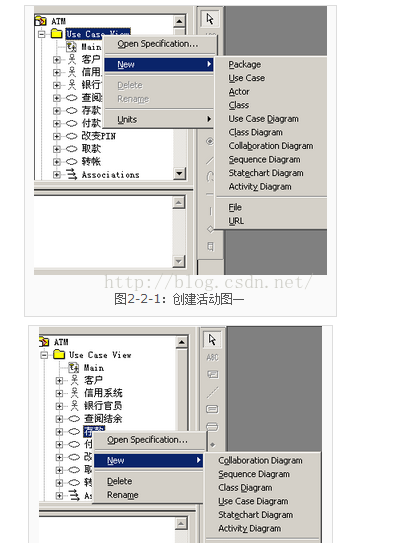
活动图显示了从活动到活动的流。

活动图可以在分析系统业务时用来演示业务流，也可以在收集系统需求的时候显示一个用例中的事件流。活动图显示了系统中某个业务或者某个用例中，要经历哪些活动，这些活动按什么顺序发生。

5.1创建活动图（图2-2-1）

(1) 用于分析系统业务：在浏览器中右击Use Case视图，选择new→activity diagram。

(2) 用于显示用例中的事件流：在浏览器中选中某个用例，然后右击这个用例，选择new→activity diagram。

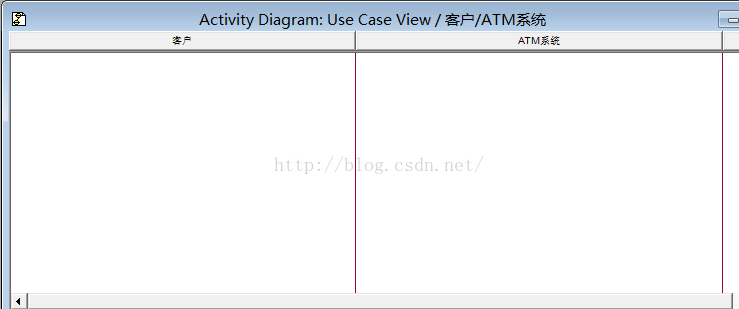


5.2增加泳道 (图2-2-2)

泳道是框图里的竖段，包含特定人员或组织要进行的所有活动。

可以把框图分为多个泳道，每个泳道对应每个人员或组织。

在工具栏选择swimlane按钮，然后单击框图增加泳道，最后用人员或组织给泳道命名。

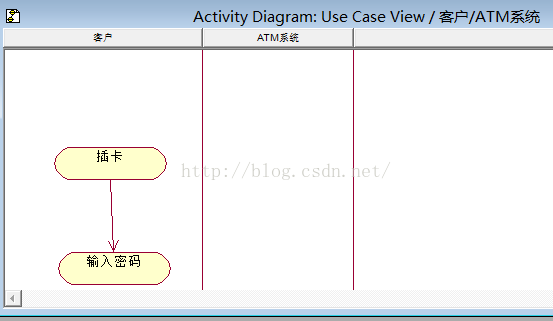


注意:先创建一个活动图，才会有swinlane

5.3增加活动并设置活动的顺序（图2-2-3）

(1) 在工具栏中选择Activity 按钮，单击活动图增加活动，命名活动。

(2) 在工具栏中选择Transition按钮，把箭头从一个活动拖向另一个活动。



5.4增加同步（图2-2-4）

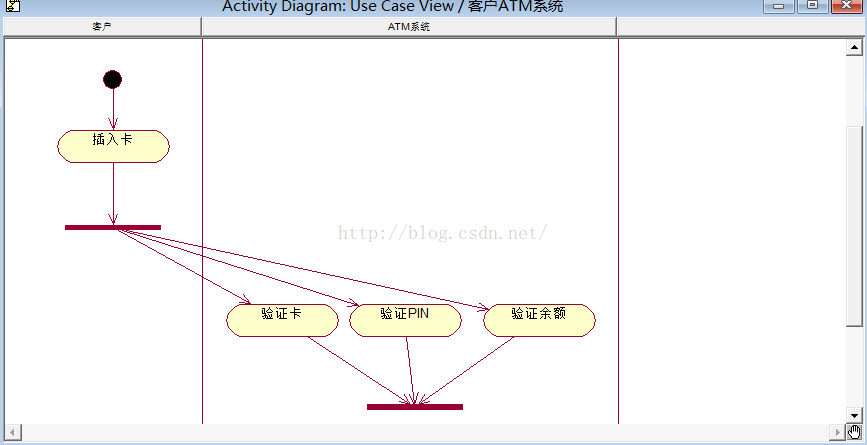
(1)选择synchronization工具栏按钮，单击框图来增加同步棒。

(2)画出从活动到同步棒的交接箭头，表示在这个活动之后开始并行处理。

(3) 画出从同步棒到可以并行发生的活动之间的交接箭头。

(4) 创建另一同步棒，表示并行处理结束。

(5) 画出从同步活动到最后同步棒之间的交接箭头，表示完成所有这些活动之后，停止并行处理。



5.5增加决策点(图2-2-5)

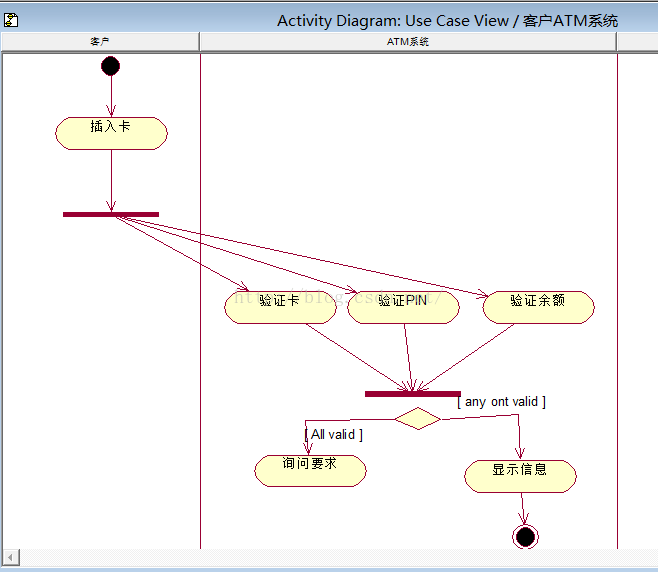
决策点表示可以采取两个或多个不同的路径。从决策到活动的交接箭头要给出保证条件，控制在决策之后采取什么路径。保证条件应该是互斥的。

(1) 选择decision工具栏按钮，单击框图增加决策点。

(2) 拖动从决策到决策之后可能发生的活动之间的交接,双击交接，打开“detail”选项卡，在Guard Condition字段中写入保证条件。

★练习：画ATM系统中“客户插入卡”的活动图

理解：客户插入信用卡之后，可以看到ATM系统运行了三个并发的活动：验证卡、验证PIN(密码)和验证余额。这三个验证都结束之后，ATM系统根据这三个验证的结果来执行下一步的活动。如果卡正常、密码正确且通过余额验证，则ATM系统接下来询问客户有哪些要求也就是要执行什么操作。如果验证卡、验证PIN(密码)和验证余额这三个验证有任何一个通不过的话，ATM系统就把相应的出错信息在ATM屏幕上显示给客户。

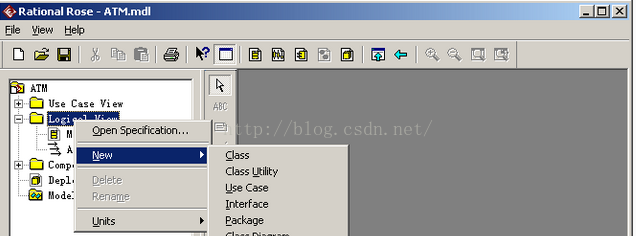


6建立类图class diagram

类图显示系统之中类和类之间的交互。

6.1创建类(图2-3-1)

在Rational Rose中可以通过几种途径来创建类。最简单的方法是利用模型的Logic视图中的类图标和绘图工具，在图中创建一个类。或者，在浏览器中选择一个包并使用快捷菜单的new→class。一旦创建了一个类，就可以通过双击打开它的对话框并在Documentation字段中添加文本来对这个类进行说明。

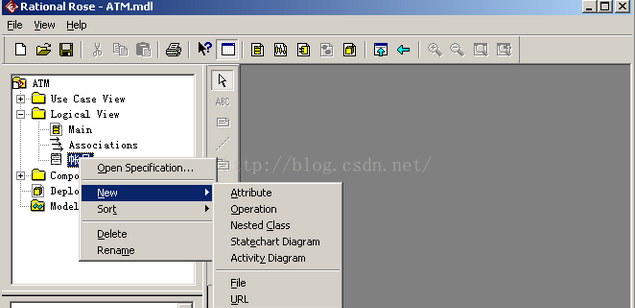


6.2创建方法(图2-3-2)

(1) 选择浏览器中或类图上的类。

(2) 使用快捷菜单的new→Operation

(3) 输入方法的名字，可在Documentation字段中为该方法输入描述其目的的简要说明。



6.3创建属性(图2-3-2)

(1) 选择浏览器中或类图上的类。

(2) 使用快捷菜单的new→Attribute。

(3) 输入属性的名字，可在Documentation字段中为该属性输入描述其目的的简要说明。

6.4创建类图(图2-3-3)

右击浏览器内的Logical视图，选择new→class diagram。把浏览器内的类拉到类图中即可。

6.5创建类之间的关系

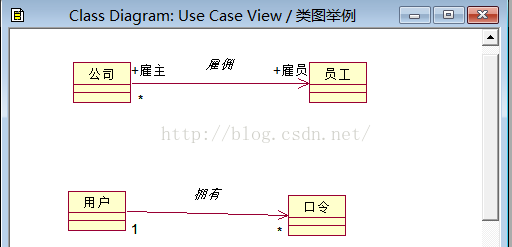
类之间的关系：

继承、关联、聚合、依赖

(1) 类之间的关系在工具栏中显示。

(2) 对于关联关系来说，双击关联关系，就可以在弹出的对话框中对关联的名称和角色进行编辑(图2-3-4)。

(3) 编辑关联关系的多重性：右单击所要编辑的关联的一端，从弹出的菜单中选择Multiplicity，然后选择所要的基数



★练习：画ATM系统中取款这个用例的类图

理解：类图显示了取款这个用例中各个类之间的关系，由四个类完成：读卡机、账目、ATM屏幕和取钱机。

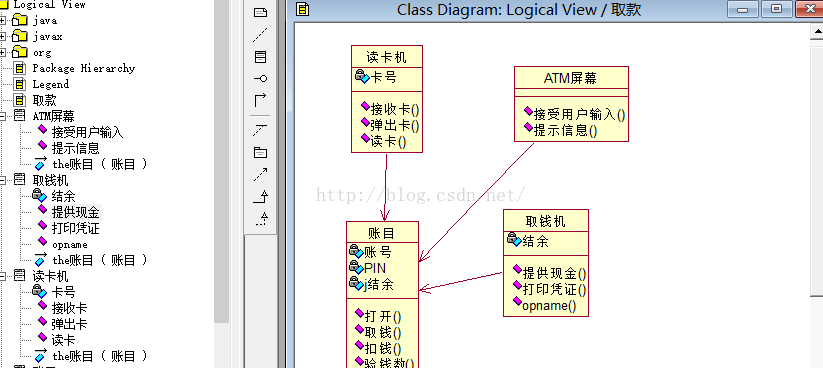
类图中每个类都是用方框表示的，分成三个部分。第一部分是类名；第二部分是类包含的属性，属性是类和相关的一些信息，如账目类包含了三个属性：账号、PIN(密码)和结余；最后一部分包含类的方法，方法是类提供的一些功能，例如帐目类包含了四个方法：打开、取钱、扣钱和验钱数。

类之间的连线表示了类之间的通信关系。例如，账目类连接了ATM屏幕，因为两者之间要直接相互通信；取钱机和读卡机不相连，因为两者之间不进行通信。

有些属性和方法的左边有一个小锁的图标，表示这个属性和方法是private的(UML中用’－’表示)，该属性和方法只在本类中可访问。

没有小锁的，表示public(UML中用’+’表示)，即该属性和方法在所有类中可访问。

若是一个钥匙图标，表示protected(UML中用’＃’表示)，即属性和方法在该类及其子类中可访问。



1. **使用心得**

使用 Rational Rose 这款 UML 建模工具，有不少值得分享的心得。它功能强大且全面，在可视化建模方面优势显著，能通过各类 UML 图，像用例图、类图、顺序图等，将复杂软件系统的结构与行为以图形化呈现，让开发团队能直观把握系统设计，还促进了不同角色间的沟通，减少理解偏差。其代码生成和逆向工程功能也很实用，正向工程能依据模型自动生成代码框架，提升开发效率、降低手动编码错误；逆向工程可从现有代码提取模型信息，便于理解和维护已有系统 。

不过，Rational Rose 也存在一些不足。它的操作较为复杂，新用户入门学习成本较高，需要花时间熟悉各种功能和操作技巧。而且在如今敏捷开发流行的趋势下，它灵活性欠佳，重量级的特性使得在快速迭代的项目中难以灵活调整。同时，在处理复杂模型时，对硬件性能要求较高，否则可能出现运行卡顿的情况。

总的来说，在大型、对架构设计要求高且需求相对稳定的软件项目中，Rational Rose 是出色的建模帮手；但在小型、快速迭代的项目里，可能需要搭配更轻量灵活的工具使用。