

面向对象的问题的处理的关键是建模问题。建模可以把在复杂世界的许多重要的细节给抽象出。许多建模工具封装了 UML（也就是 Unified Modeling Language™），这篇课程的目的是展示出 UML 的精彩之处。

UML 中有九种建模的图标，即：

- 用例图
- 类图
- 对象图
- 顺序图
- 协作图
- 状态图
- 活动图
- 组件图
- 配置图

本课程中的某些部分包含了这些图的细节信息的页面链接。而且每个部分都有一个小问题，测试一下你对这个部分的理解。

为什么 UML 很重要？

为了回答这个问题，我们看看建筑行业。设计师设计出房子。施工人员使用这个设计来建造房子。建筑越复杂，设计师和施工人员之间的交流就越重要。蓝图就成为了这个行业中的设计师和施工人员的必修课。

写软件就好像建造建筑物一样。系统越复杂，参与编写与配置软件的人员之间的交流也就越重要。在过去十年里 UML 就成为分析师，设计师和程序员之间的“建筑蓝图”。现在它已经成为了软件行业的一部分了。UML 提供了分析师，设计师和程序员之间在软件设计时的通用语言。

UML 被应用到面向对象的问题的解决上。想要学习 UML 必须熟悉面向对象解决问题的根本原则——都是从模型的建造开始的。一个模型 **model** 就是根本问题的抽象。域 **domain** 就是问题所处的真实世界。

模型是由对象 **objects** 组成的，它们之间通过相互发送消息 **messages** 来相互作用的。记住把一个对象想象成“活着的”。对象有他们知道的事（属性 **attributes**）和他们可以做的事（行为或操作 **behaviors or operations**）。对象的属性的值决定了它的状态 **state**。

类 **Classes** 是对象的“蓝图”。一个类在一个单独的实体中封装了属性（数据）和行为（方法或函数）。对象是类的实例 **instances**。

用例图

用例图 **Use case diagrams** 描述了作为一个外部的观察者的视角对系统的印象。强调这个系统是什么而不是这个系统怎么工作。

用例图与情节紧紧相关的。情节 **scenario** 是指当某个人与系统进行互动时发生的情况。下面是一个医院门诊部的情节。

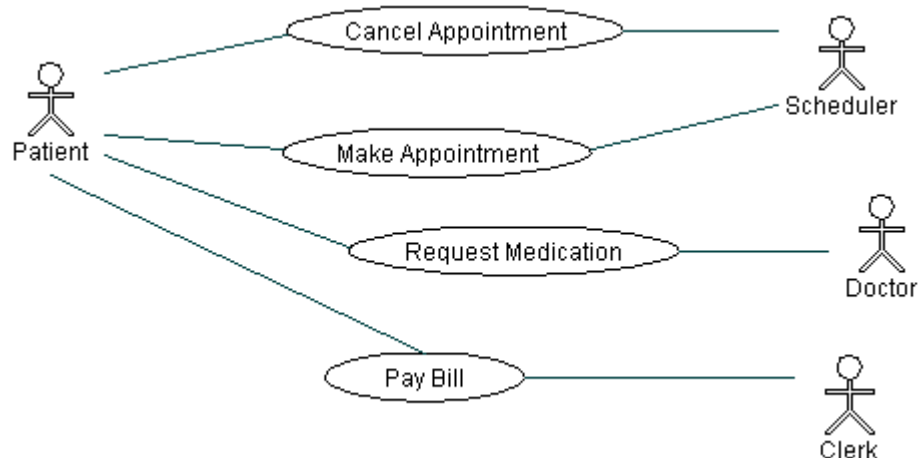
“一个病人打电话给门诊部预约一年一次的身体检查。接待员找出在预约记录本上找出最近的没有预约过的时间，并记上那个时间的预约记录。”

用例 **Use case** 是为了完成一个工作或者达到一个目的的一系列情节的总和。角色 **actor** 是发动与这个工作有关的事件的人或者事情。角色简单的扮演着人或者对象的作用。下面的图是一个门诊部 **Make Appointment** 用例。角色是病人。角色与用例的联系是通讯联系 **communication association**（或简称通讯 **communication**）



角色是人状的图标，用例是一个椭圆，通讯是连接角色和用例的线。

一个用例图是角色，用例，和它们之间的联系的集合。我们已经把 **Make Appointment** 作为一个含有四个角色和四个用例的图的一部分。注意一个单独的用例可以有多个角色。



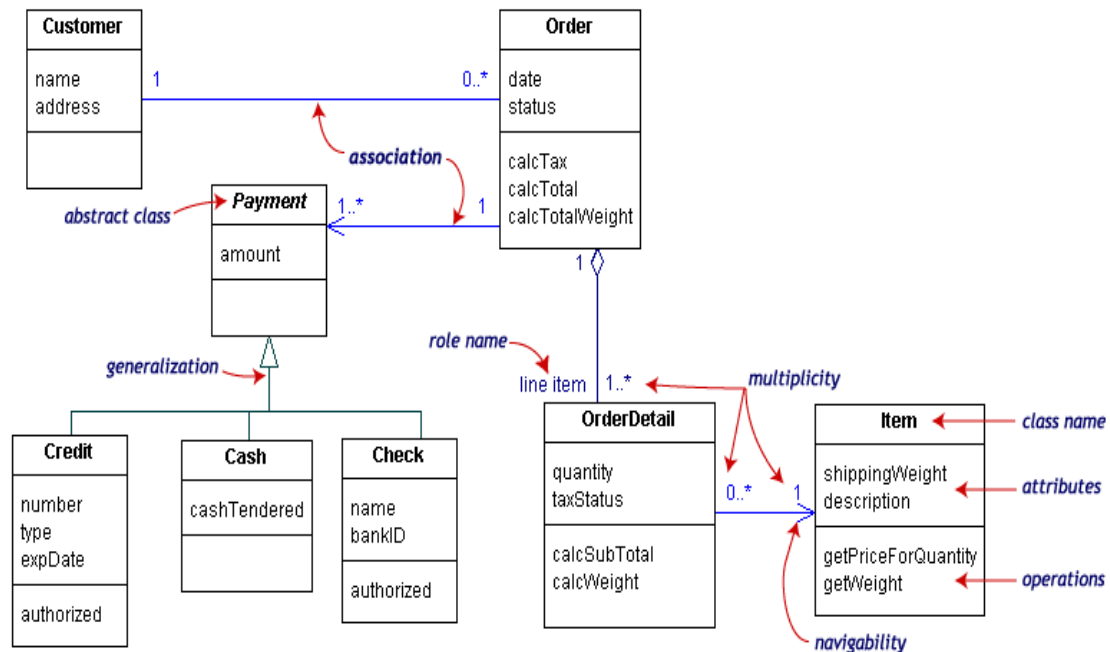
用例图在三个领域很有作用。

- 决定特征（需求）。当系统已经分析好并且设计成型时，新的用例产生新的需求
- 客户通讯。使用用例图很容易表示开发者与客户之间的联系。
- 产生测试用例。一个用例的情节可能产生这些情节的一批测试用例。

类图

类图 **Class diagram** 通过显示出系统的类以及这些类之间的关系来表示系统。类图是静态的——它们显示出什么可以产生影响但不会告诉你什么时候产生影响。

下面是一个顾客从零售商处预定商品的模型的类图。中心的类是 **Order**。连接它的是购买货物的 **Customer** 和 **Payment**。**Payment** 有三种形式：**Cash**，**Check**，或者 **Credit**。订单包括 **OrderDetails**（line item），每个这种类都连着 **Item**。



UML 类的符号是一个被划分成三块的方框：类名，属性，和操作。抽象类的名字，像 **Payment** 是斜体的。类之间的关系是连接线。

类图有三种关系。

- 关联 **association**—表示两种类的实例间的关系。如果一个类的实例必须要用另一个类的实例才能完成工作时就要用关联。在图中，关联用两个类之间的连线表示。
- 聚合 **aggregation**—当一个类属于一个容器是的一种特殊关系。聚合用一个带菱形的连线，菱形指向具有整体性质的类。在我们的图里，**Order** 是 **OrderDetails** 的容器。
- 泛化 **generalization**—一个指向以其他类作为超类的继承连线。泛化关系用一个三角形指向超类。**Payment** 是 **Cash**，**Check** 和 **Credit** 的超类。

一个关联有两个尾端。每个尾端可以有一个角色名 **role name** 来说明关联的作用。比如，一个 **OrderDetail** 实例是一个 **Order** 实例的项目。

关联上的方向性 **navigability** 箭头表示该关联传递或查询的方向。**OrderDetail** 类可以查询他的 **Item**，但不可以反过来查询。箭头方向同样可以告诉你哪个类拥有这个关联的实现；也就是，**OrderDetail** 拥有 **Item**。没有方向性的箭头的关联是双向。

关联尾端的数字表示该关联另一边的一个实例可以对应的数字端的实例的格数，通过这种方式表达关联的多样性 **multiplicity**。多样性的数字可以是一个单独的数字或者是一个数字的范围。在例子中，每个 **Order** 只有一个 **Customer**，但一个 **Customer** 可以有任意多个 **Order**。

下面这个表给出了最普遍的多样性示例。

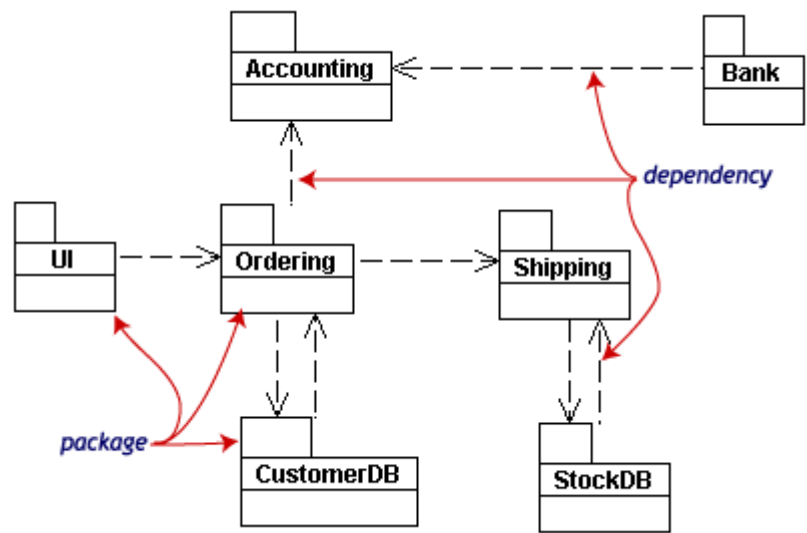
多样性	意义
0..1	0 或 1 个实例. n..m 符号表示有 n 到 m 个实例
0..* or *	没有实例格数的限制（包括没有）
1	只有一个实例
1..*	最少一个实例

每个类图包括类，关联和多样性表示。方向性和角色是为了使图示得更清楚时可选的项目。

包和对象图

为了简单地表示出复杂的类图，可以把类组合成包 **packages**。一个包是 UML 上有逻辑关系的元件的集合。下面这个图是是一个把类组合成包的一个商业模型。

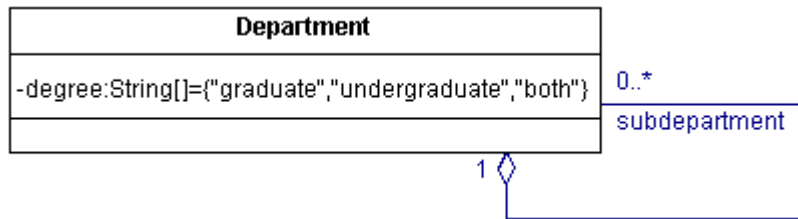
dependencies 关系。如果另一个的包 **B** 改变可能会导致一个包 **A** 改变，则包 **A** 依赖包 **B**。



包是用一个在上方带有小标签的矩形表示的。包名写在标签上或者在矩形里面。点化线箭头表示依赖

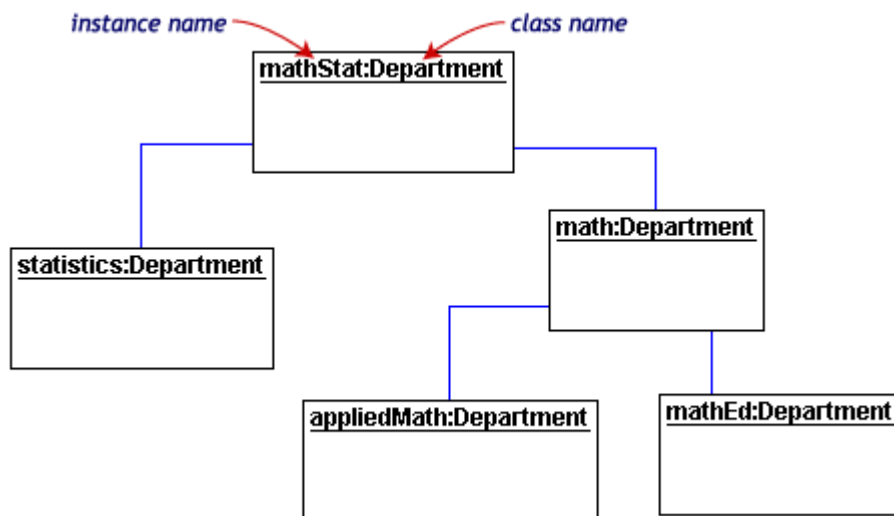
对象图 **Object diagrams** 用来表示类的实例。他们在解释复杂关系的细小问题时（特别是递归关系时）很有用。

这个类图示一个大学的 **Department** 可以包括其他很多的 **Departments**。



这个对象图示上面类图的实例。用了很多具体的例子。

UML 中实例名带有下划线。只要意思清楚，类或实例名可以在对象图中被省略。



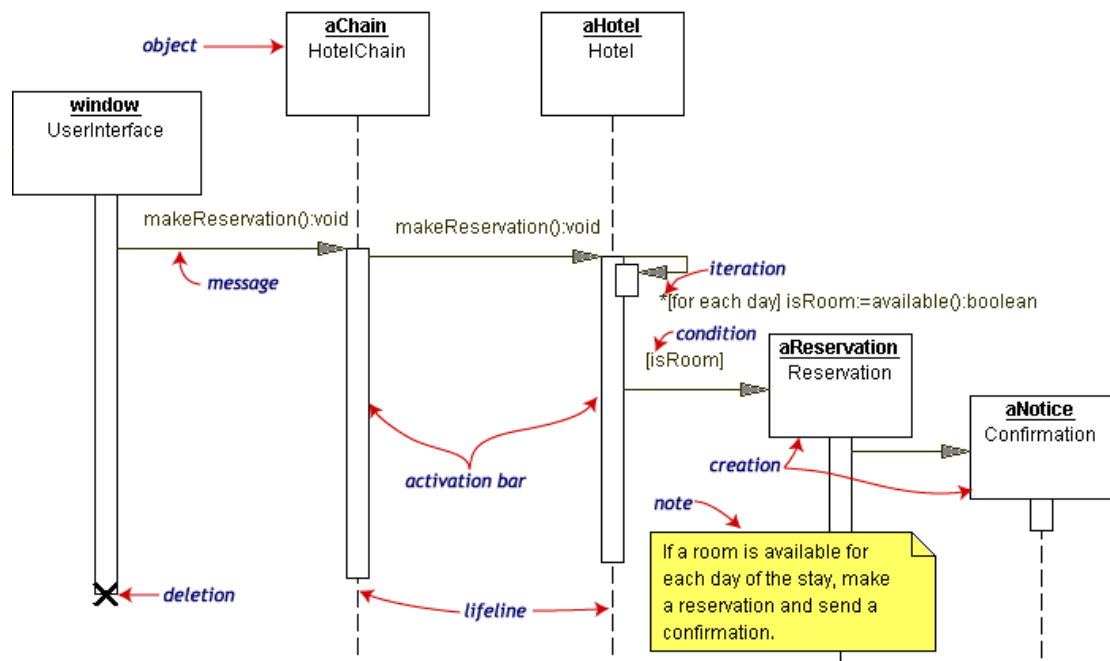
每个类图的矩形对应了一个单独的实例。实例名称中所强调的 UML 图表。类或实例的名称可能是省略对象图表只要图的意义仍然是明确的。

顺序图

类图和对象图是静态模型的视图。交互图是动态的。他们描述了对象间的交互作用。

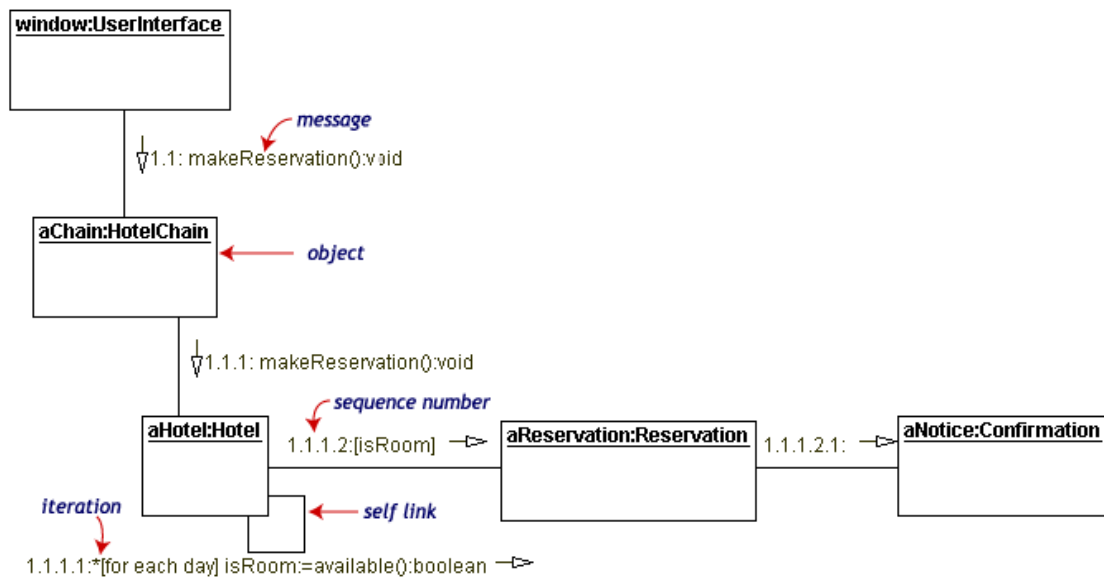
顺序图将交互关系表示为一个二维图。纵向是时间轴，时间沿竖线向下延伸。横向轴代表了在协作中各独立对象的类元角色。类元角色用生命线表示。当对象存在时，角色用一条虚线表示，当对象的过程处于激活状态时，生命线是一个双道线。

消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头以时间顺序在图中从上到下排列。



协作图

协作图也是互动的图表。他们像序列图一样也传递相同的信息，但他们不关心什么时候消息被传递，只关心对象的角色。在序列图中，对象的角色放在上面而消息则是连接线。



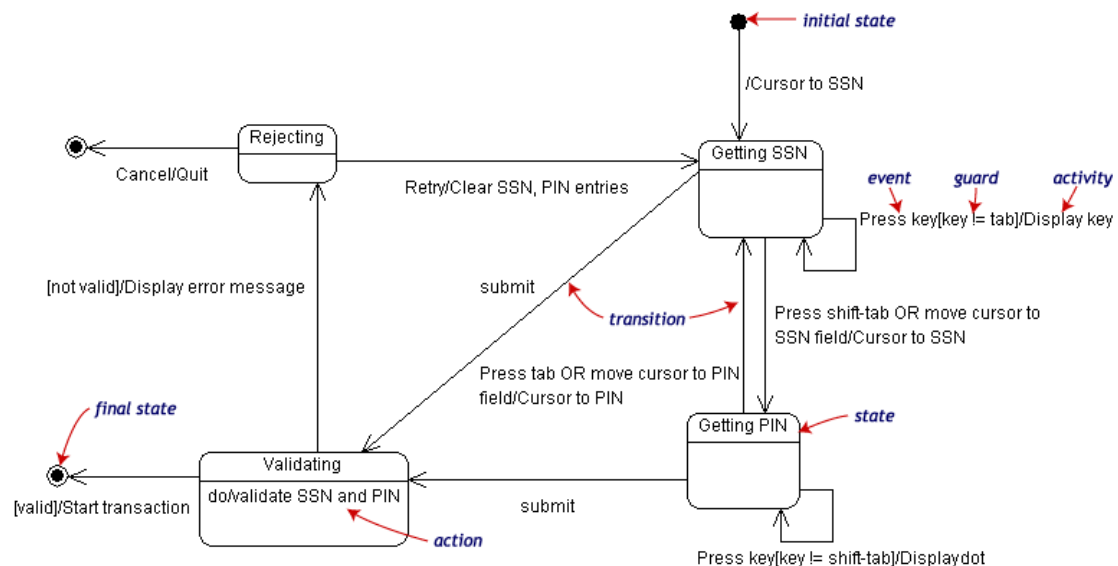
对象角色矩形上标有类或对象名（或者都有）。类名前面有个冒号（:）。

协作图的每个消息都有一个序列号。顶层消息的数字是 1。同一个等级的消息（也就是同一个调用中的消息）有同样的数字前缀，再根据他们出现的顺序增加一个后缀 1, 2 等等。

状态图

对象拥有行为和状态。对象的状态是由对象当前的行动和条件决定的。状态图 **statechart diagram** 显示出了对象可能的状态以及由状态改变而导致的转移。我们的模型例图建立了一个银行的在线登录系统。登录过程包括输入合法的密码和个人账号，再提交给系统验证信息。

登录系统可以被划分为四种不重叠的状态：**Getting SSN**, **Getting PIN**, **Validating**, 以及 **Rejecting**。每个状态都有一套完整的转移 **transitions** 来决定状态的顺序。



状态是用圆角矩形来表示的。转移则是使用带箭头的连线表示。触发转移的事件或者条件写在箭头的旁边。我们的图上有两个自转移。一个是在 **Getting SSN**, 另一个则在上 **Getting PIN**。

初始状态（黑色圆圈）是开始动作的虚拟开始。结束状态也是动作的虚拟结束。

事件或条件触发动作时用（/动作）表示。当进入 **Validating** 状态时，对象并不等外部事件触发转移。取而代之，它产生一个动作。动作的结果决定了下一步的状态。

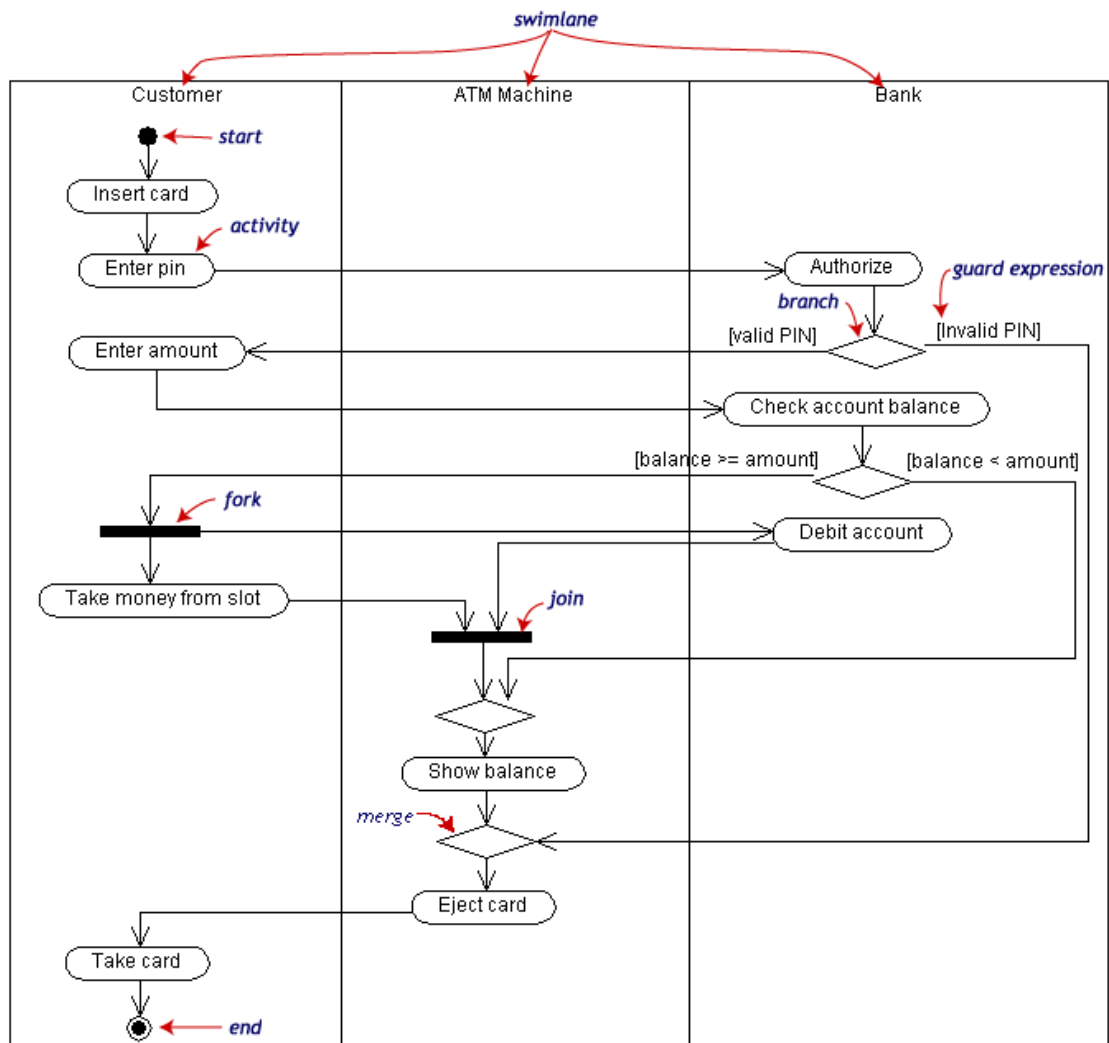
活动图

活动图 **activity diagram** 是一个很特别的流程图。活动图和状态图之间是有关系的。状态图把焦点集中在过程中的对象身上，而活动图则集中在一个单独过程动作流程。活动图告诉了我们活动之间的依赖关系。

对我们的例子来说，我们使用如下的过程。

“通过 ATM 来取钱。”

这个活动有三个类 **Customer**, **ATM** 和 **Bank**。整个过程从黑色圆圈开始到黑白的同心圆结束。活动用圆角矩形表示。



活动图可以被分解成许多对象泳道 **swimlanes**，可以决定哪些对象负责那些活动。每个活动都有一个单独的转移 **transition** 连接这其他的活动。

转移可能分支 **branch** 成两个以上的互斥的转移。保护表达式（在[]中）表示转移是从一个分支中引出的。分支以及分支结束时的合并 **merge** 在图中用菱形表示。

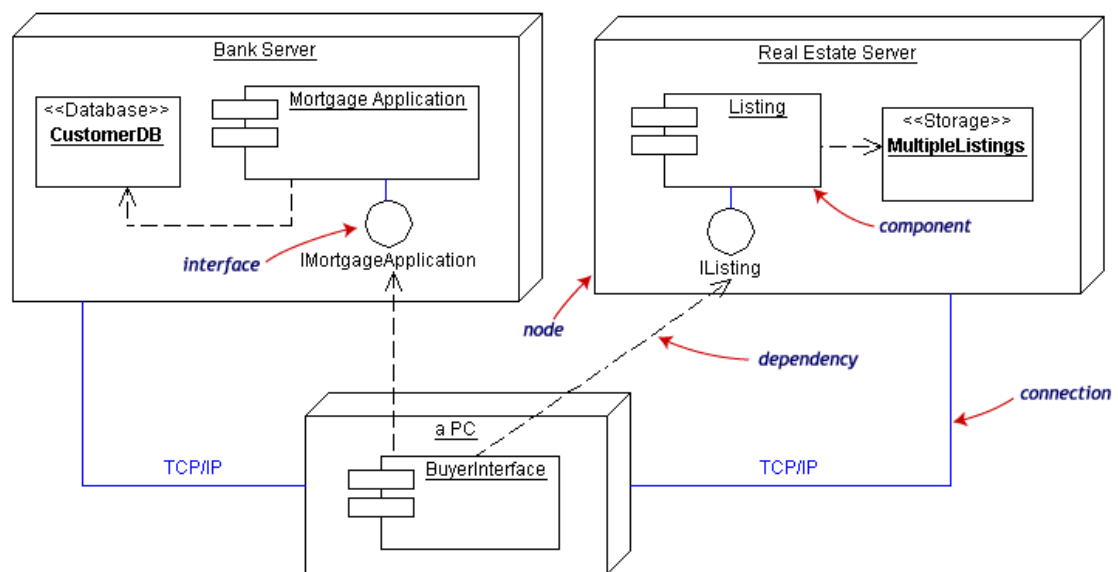
转移也可以分解 **fork** 成两个以上的并行活动。分解以及分解结束时的线程结合 **join** 在图中用粗黑线表示

组件与配置图

组件 **component** 是代码模块。组件图是是类图的物理实现。

配置图 **Deployment diagrams** 则是显示软件及硬件的配置。

下面的配置图说明了与房地产事务有关的软件及硬件组件的关系。



物理上的硬件使用节点 **nodes** 表示。每个组件属于一个节点。组件用左上角带有两个小矩形的矩形表示。