软件 = 一组相互作用的对象
对象 = 一个或多个角色的实现
状态
行为
责任 = 执行一项任务或掌握某种信息的义务
角色 = 一组相关的责任
协作 = 对象或角色（或两者）之间的互动•

对象角色（Role）是对象责任（Responsibility）的体现，责任是指对象持有、维护特定数据并基于该数据进行操作的能力。
要求对象有明确的角色就是要求对象在应用中维护一定的数据并对其进行操作，简单的说就是要拥有状态和行为状态是对象的特征描述，一般可以认为是类的成员变量
对象的行为通常是针对状态的操作，一般表示为类的方法。
对象是对现实世界事物的抽象，在应用中履行特定的职责。对象具有标识、状态和行为。
协作是对象之间的相互请求，一般表现为对象之间的方法调用。
场景（scenario）是用来描述一个用户和系统之间交互的一系列步骤。
假定我们有一个虚拟图书馆系统（借书行为在系统中模拟），有一个借书的场景：
用户根据图书名查询图书，确认需要借阅的图书是否有库存，确定借阅。系统检查用户的借阅配额以及借阅权限，并立刻确认借阅。这个场景是借阅时发生的一种成功的情况，但也许用户已经借阅的图书已经达到可借阅的上限，无法再借阅新的图书；或者需要借阅的图书已经没有库存了。这些都是新的场景。
用例是具有共同用户目标的一系列场景的集合
一个用例是所有和用户某一目标相关的成功和失败场景的集合，用例用来记录系统的功能需求。
UML中使用用例来获取系统的功能需求，而不是所有的系统需求，尤其是非功能性需求，比如性能目标、安全性设计等等。
用例是从用户的角度来描述系统要做的事情，一般不应该涉及到系统怎样实现该功能，也不应该包含系统与用户之间的交互界面描述，比如屏幕的布局、文字显示、按钮设置等内容。用例：
名称
参与者
正常流程
扩展流程
特殊需求

基本问题求解的原则
• 分解与抽象
• 面向对象方法的原则
• 职责与协作
• 面向对象
方法三要素
• 封装

• 继承
• 多态
个对象维护其自身的状态需要对外公开一些方法，行使其职能也要对外公开一些方法，这些方法组合起来定义了该对象允许外界访问的方法，或者说限定了外界可以期望的表现，它们是对象需要对外界履行的协议（Protocol）。
一个对象的整体协议可能会分为多个内聚的逻辑行为组，例如，一个学生对象的有些行为是在学习时发生的，而另外一些可能是在购物时发生的，这样，学生对象的行为就可以分为两组。划分后的每一个逻辑行为组就描述了对象的一个独立职责，体现了对象的一个独立角色。
如果一个对象拥有多个行为组，就意味着该对象拥有多个不同的职责，需要扮演多个不同的角色。例如，上例的学生对象就需要同时扮演学生和顾客两个角色。每一个角色都是对象一个职责的体现，所有的角色是对象所有职责的体现。所以，理想的单一职责对象应该仅仅扮演一个角色。
协作模型描述的是一些关于“如何做”，“何时做”和“与谁工作”的动态行为
当将具有共同责任的对象组织成“邻域”时，需要细致地安排邻里里之间
的协作关系以履行行其更大大的责任。同样也要指定邻域之外的对象如何
与邻域对象提供的服务进行行交互。
这样做的好处之一是，当我们对系统部分作修改时不会涉及整个系统。
一个设计良好的面向对象应用软
件应包含一定量的变化，而不用做大幅度的改动。
• 软件实现了一个责任系统。不同的角色通过协作履行不同的责任，良
好的软件结构可以有效地执行这些责任。我们的设计工作从创建对象
开始，将特定的责任分配给对象，使其了解某些信息或完成某些工
作，而这些对象的集体行为将履行更大的责任。
• 一个对象所提供的服务和所持有的信息定义了该对象的行为方式，并
由此和其他对象区分开来。在早期设计中，只需要将具体责任分配给
对象，这就已经足够了。最重要的是，保证责任为他人服务。设计模
型负责在对象之间分配责任。
当将具有共同责任的对象组织成“邻域”时，需要细致地安排邻里之间的协作关系以履行其更大的责任。同样也要指定邻域之外的对象如何与邻域对象提供的服务进行交互。这样做的好处之一是，当我们对系统部分作修改时不会涉及整个系统。一个设计良好的面向对象应用软件应包含一定量的变化，而不用做大幅度的改动。
软件实现了一个责任系统。不同的角色通过协作履行不同的责任，良好的软件结构可以有效地执行这些责任。我们的设计工作从创建对象开始，将特定的责任分配给对象，使其了解某些信息或完成某些工作，而这些对象的集体行为将履行更大的责任。
一个对象所提供的服务和所持有的信息定义了该对象的行为方式，并由此和其他对象区分开来。在早期设计中，只需要将具体责任分配给对象，这就已经足够了。最重要的是，保证责任为他人服务。设计模型负责在对象之间分配责任。
智能控制水水温
• 周末水温高
• 夜晚水温低
• 生病等特殊情况水温高
• 度假水温低
• Class:

• WaterHeaterController
• mode
• lowTemp
• highTemp
• weekendDays
• Special Time
• Interface:
• ThemostatDevice
由SpecialTime类保存特殊时间，并提供
isSpecialTime（ ）；Controller调用方法
• Good：单一职责

寻找参与者
• 谁对系统有着明确的目标和要求并且主动
发出动作？
• 系统是为谁服务的？
• 用用例是相对独立的
• 取钱？填写取款单？
• 用用例的执行行结果对参与者来说是可观测的和有意义的
• 登陆系统？后台进程监控？
• 这件事必须有一一个参与者发起。
• ATM吐钞票？
• 用用例必须是以动宾短语形式出现的
• 统计？报表
面向对象编程的概念与原则
• 类的协作
• 类之间的关系
• General Relationship
• Instance Level Relationship
• Class Level Relationship
General Relationship
• 依赖
• Instance Level Relationship
• 连接
• 关联
• Class Level Relationship
• 继承
• 实现
• 依赖（Dependency)
• 物理关系(model-time relationship between definitions)

• 连接（External Links）
• Relationship among objects
• A link is an instance of an association
• 关联（Association）
• 逻辑关系(run-time relationship between instances of classifiers)
• 关联的分类
• 普通关联
• 可导航关联
• 聚合（Aggregation)
• 组合(Composition)
• 它们的强弱关系是没有异议的：依赖 < 普通关联 < 聚合 < 组合
继承（extends）
实现（implements）
依赖
• 关系：" ... uses a ..."
• 所谓依赖就是某个对象的功能依赖于另外
的某个对象，而而被依赖的对象只是作为一
种工具在使用用，而而并不持有对它的引用
表现为局部变量，方法参数或者静态方法的调用
关联：
关系：" ... has a ..."
• 所谓关联就是某个对象会长期的持有另一
个对象的引用，二者的关联往往也是相
互的。关联的两个对象彼此间没有任何强
制性的约束，只要二者同意，可以随时解
除关系或是进行关联，它们在生命期问题
上没有任何约定。被关联的对象还可以再
被别的对象关联，所以关联是可以共享
的
普通关联：
可导航关联：
多重性（Multiplicity)
the number of objects that participate in the association
0..1
No instances, or one instance (optional, may)
• 1
Exactly one instance
• 0..* or *
Zero or more instances
• 1..*
One or more instances (at least one)
聚合：

关系：" ... owns a ..."
聚合是强版本的关联。它暗含着一种所属关系以
及生命期关系。被聚合的对象还可以再被别的对象关联，
所以被聚合对象是可以共享的。虽然是共享的，聚合代
表的是一种更亲密的关系。
组合：
关系：" ... is a part of ..."
组合是关系当中的最强版本，它直接要求包含对象对被包含对象的拥有以及包含对象与被包含对象生命期的关系。被包含的对象还可以再被别的对象关联，所以被包含对象是可以共享的，然而绝不存在两个包含对象对同一个被包含对象的共享。
组合关系就是整体与部分的关系，部分属于整体，整体不存在，部分一定不存在，然而部分不存在整体是可以存在的，说的更明确一些就是部分必须创生于整体创生之后，而销毁于整体销毁之前。
部分在这个生命期内可以被其它对象关联甚至聚合，但有一点必须注意，一旦部分所属于的整体销毁了，那么与之关联的对象中的引用就会成为空引用，这一点可以利用程序来保障。
心脏的生命期与人的生命期是一致的，如果换个部分就不那么一定，比如阑尾，很多人在创生后的某个时间对其厌倦便提前销毁了它，可它和人类的关系不可辩驳的属于组合。