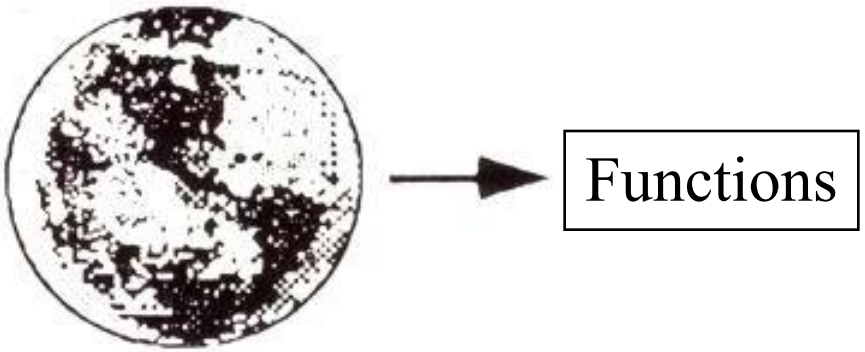
第6章 UML概要

面向对象(Object-oriented)技术

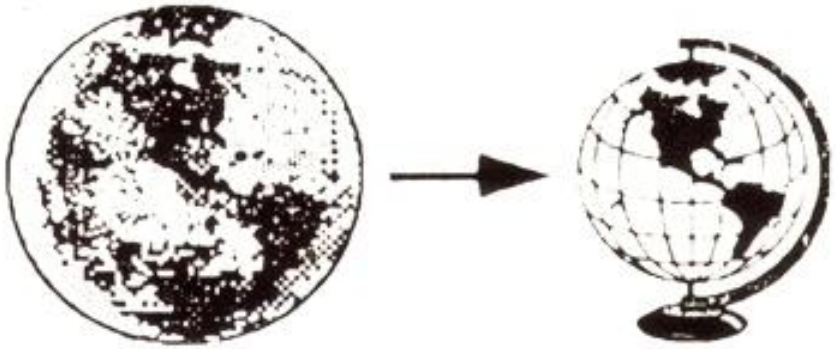
- 面向对象技术充分体现了分解，抽象，模块化，信息隐蔽等思想，可以有效地
  - 提高软件生产率
  - 缩短软件开发时间
  - 提高质量

面向对象技术的优势

- **稳定**: 较小的需求变化不会导致系统结构大的改变。
- **易于理解**：面向对象的模型更加贴切地反映了现实世界，尤其对于使用者。



功能分解法



面向对象的方法

- **更好的适应性**，能更好地适应用户需求地变化，有助于开发**大型软件系统**。
- 具有更高的**可靠性**。

……

问题：和传统方法相比，OO方法为什么会具有这些优势？

面向对象领域中的一些常用术语

Object 对象
(Instance 实例)
Class 类
Attribute 属性
Operation 操作
(Method 方法)

Encapsulation 封装
Inheritance 继承
Polymorphism 多态
Message 消息
OOA 面向对象分析
OOD 面向对象设计

对象（Object）

- **对象**是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组方法组成。

说明：

- 对象只描述客观事物本质的、与系统目标有关的特征，而不考虑那些非本质的、与系统目标无关的特征。
- 在软件生命周期的不同阶段，对象可以有不同的表现形式。
  - 如在OOA/OOD阶段对象是用某种表示法给出的比较粗略的定义，而在OOP阶段对象是比较详细的源程序代码。
- 对象之间通过消息通信。
  - 一个对象通过向另一个对象发送消息激活某一个功能。
- 在不是纯面向对象语言中允许有不属于任何对象的成分存在，例如C++程序中的main函数。
- “**实例**”(instance)和“**对象**”的关系

类(Class)

- **类**是具有相同属性和方法的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和方法两个主要部分。

说明

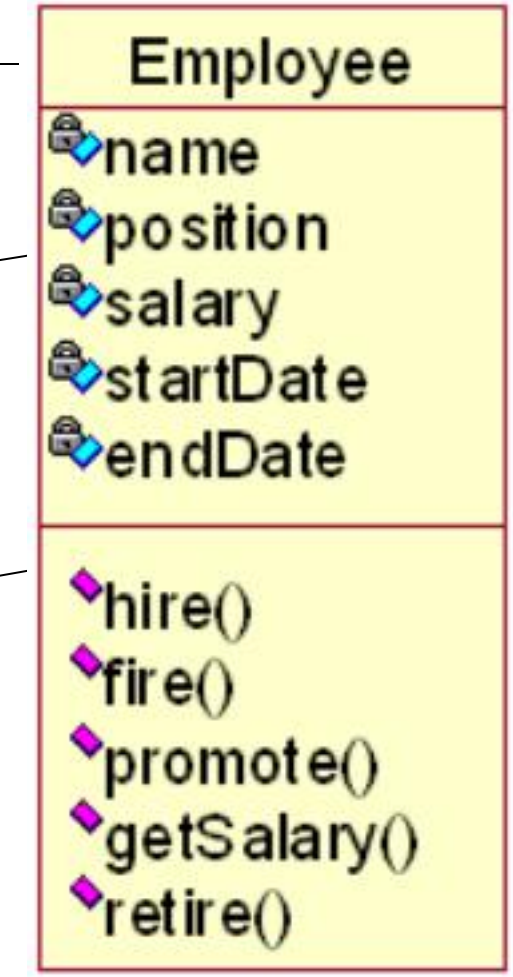
- 同类对象具有相同的属性和方法，是指它们的定义形式相同，而不是说每个对象的属性值都相同。
- 类是**静态**的；类的存在、语义和关系在程序执行前就已经定义好了。
- 对象是**动态**的; 对象在程序执行时可以被创建和删除。

类的示例

类名

属性

方法



类: Employee

封装（Encapsulation）

- **封装**就是把对象的属性和方法结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。

说明：

- 封装就是使一个对象形成两个部分: **接口**和**实现**，对于用户来说，接口是可见的，实现是不可见的。
- 封装提供两种保护

1. **保护对象**：防止被用户误用。
2. **保护客户端**：封装能减小实现过程改变的副作用，即实现过程的的改变不会影响到相应客户端的改变。

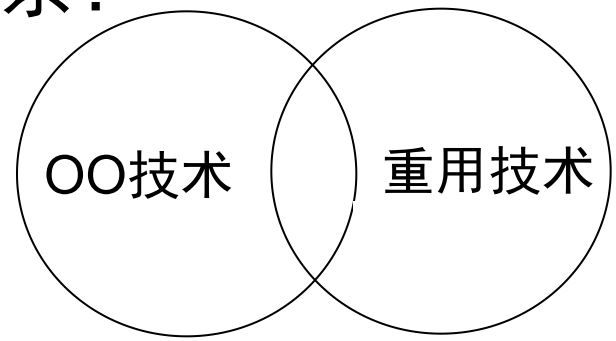
继承（Inheritance）

- 特殊类的对象拥有其一般类的全部属性与方法，称作特殊类对一般类的**继承**。

（一般类/特殊类；父类/子类；超类/子类；基类/派生类等是相同的概念。）

说明：

- 继承保证类之间的一致性
  - 父类可以为所有子类定制规则；（许多OOP语言提供了这种实现机制，如C++中的虚函数，Java中的接口等）。
- 利用继承可以开发更贴近现实的模型
- 继承增加软件重用的机会
  - 降低开发和维护费用
- 问题：
  - 面向对象技术和软件重用的关系？
  - 继承的目的？是否只为重用？



- 子类可以继承父类的属性，也可以增加或重新定义继承的属性。
- 子类可以继承父类的操作，也可以增加或重新定义继承的操作。（重新定义称为override)
- override(覆盖)和overload(重载)的区别。

override的例子

```
public class A {
   String name;
   public String getValues () {
      return “Value is: ” + name;
   }
}

public class B extends A {
    String address;
    public String getValues ()  {
       return “Value is: ” + address;
    }
}
```

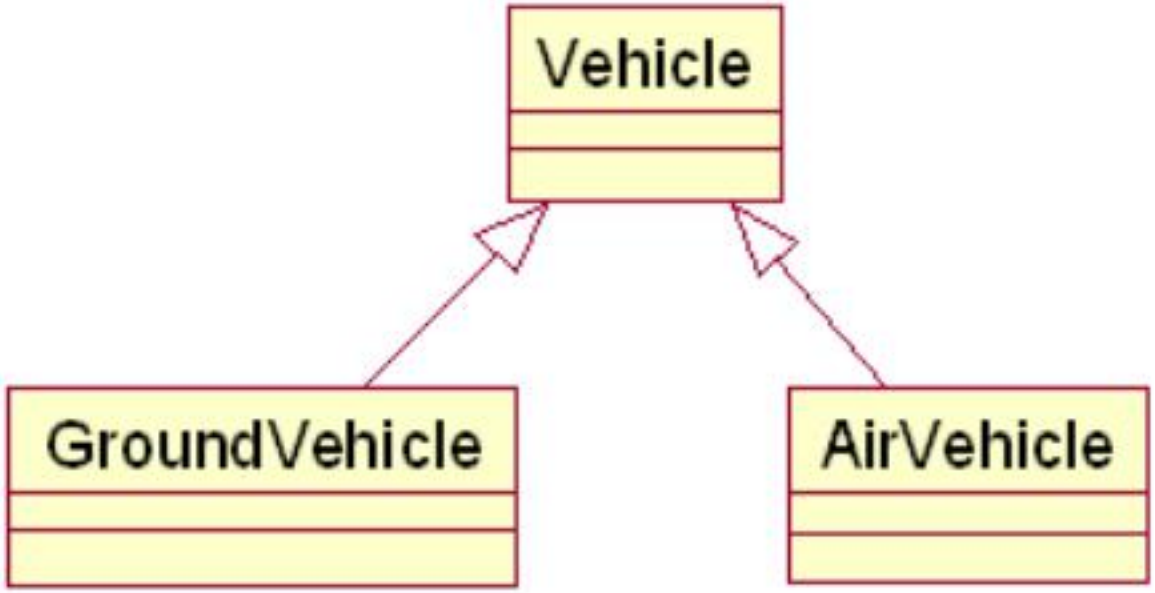
overload的例子

```
public class A {
    int age;
    String name;
    public void  setValue (int i) {
        age = i;
    }
    public void  setValue (String s) {
        name = s;
}
```

- 继承可分为**单继承**和**多继承**
  - 单继承：子类只从一个父类继承
  - 多继承：子类从多于一个的父类继承

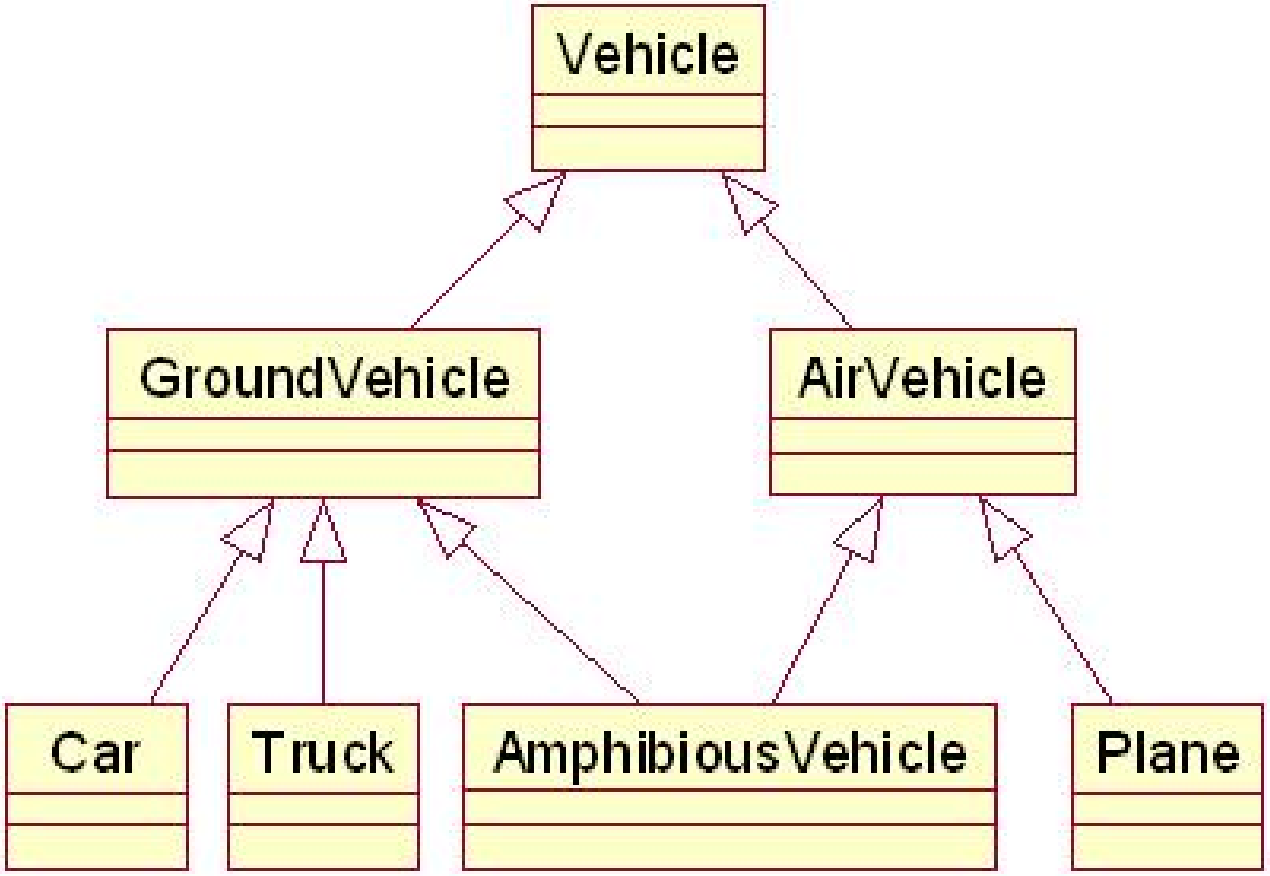
单继承例子

- 例：交通工具是一个父类，地面交通工具和空中交通工具是子类



多继承例子

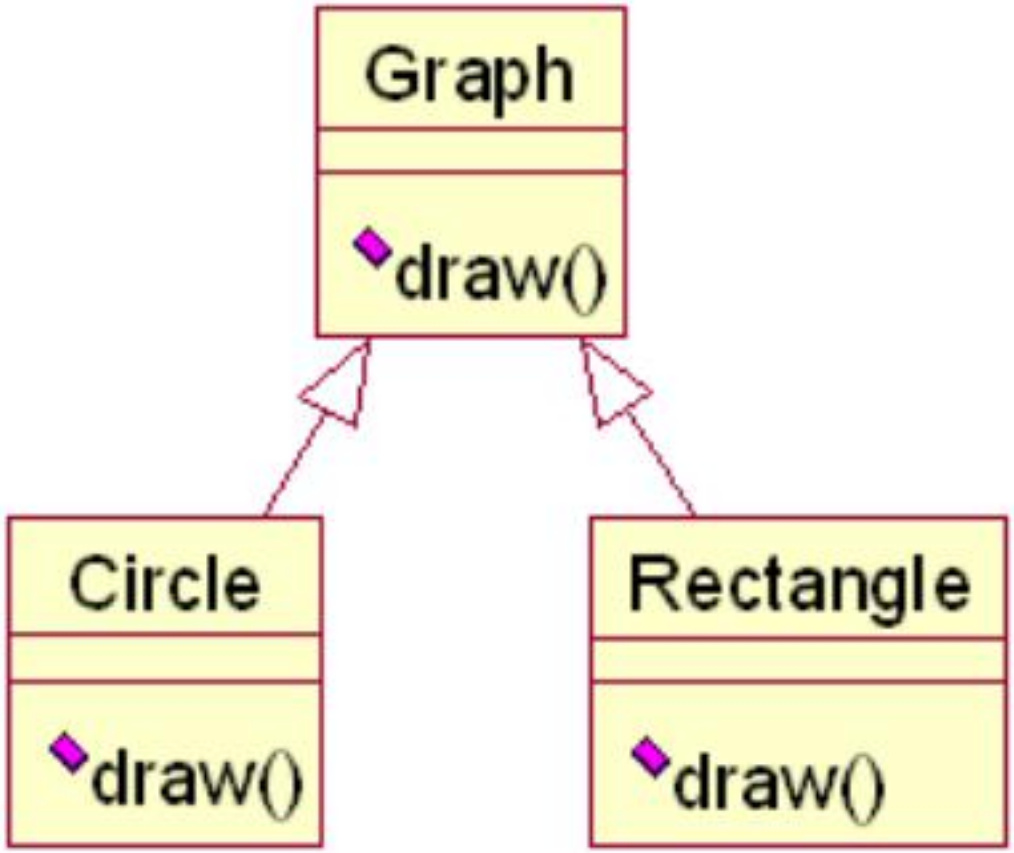
- 一个两栖交通工具同时继承地面交通工具和水上交通工具。



多态(Polymorphism)

- 从字面上理解，**多态**就是有多种形态的意思。
- 在面向对象技术中，**多态**指的是使一个实体在不同的上下文条件下具有不同意义或用法的能力。
- **多态**是保证系统有较好的适应性的一个重要手段，也是用OO技术所表现出来的一个重要特征。

多态的例子



说明：

- 跟多态相关的概念：
  - 覆盖(override)，动态绑定(dynamic binding)
- 多态属于运行时问题，而重载(overload)是编译时问题。

消息(Message)

- **消息**就是向对象发出的服务请求，它包含下述信息：提供服务的对象标识、服务（方法）标识、输入信息和回答信息。

说明：

- 通过消息进行对象之间的通信，是OO方法的一个原则。
- 采用**消息**这个术语的好处（而不是**函数调用**）：
  - 第一，更接近人们日常思维所采用的术语；
  - 第二，其涵义更具有一般性，不限制采用何种实现技术。
    - 如在分布式环境中，对象可以在不同的网络结点上实现并且相互提供服务，在这种情况下，消息术语具有更强的适应性。

面向对象分析OOA

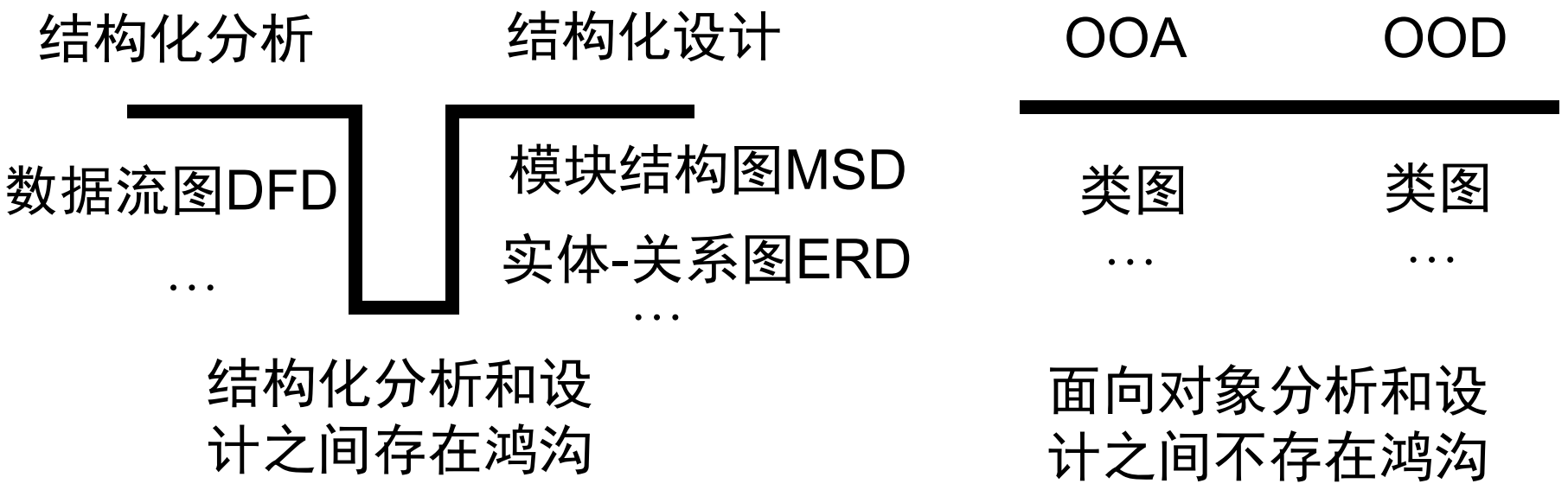
- 简单地说，**OOA** (Object-Oriented Analysis)
  —— 面向对象的分析，就是运用面向对象的方法进行需求分析。

面向对象设计OOD

- 定义：**OOD** (Object-Oriented Design)——面向对象的设计，就是在OOA 模型的基础上运用面向对象方法进行系统设计，目标是产生一个符合具体实现条件的OOD 模型。

OOA和OOD的关系

- **OOA** 和 **OOD** 采用一致的概念和表示法，二者之间不存在鸿沟。



OOA/OOD和结构化分析/设计

- OOA/OOD中的思维过程，发现过程，以及用户、分析员、设计人员之间的交流与结构化分析/设计是本质不同的。
- 一些适合于早期方法的概念，如内聚、耦合、抽象、有意识地推迟设计决策等同样适用于OOA/OOD。
- OOA/OOD对结构化方法是“revolution”还是“evolution”?

什么是UML?

- 什么是UML？
  - 统一建模语言（Unified Modelling Language）的简称。
  - 是面向对象系统分析设计时常用的一种标准标记方法。
  - 由OMG(Object Management Group)软件标准化组织进行统一标准化的。
  - 仅仅只是一种描述语言，规定了一些表示方法，和具体的开发语言以及开发过程没有任何关系。

UML的特点

- UML的主要特点可归纳为以下几点：
  - 统一的标准：已成为面向对象的标准化的统一的建模语言。
  - 面向对象
  - 可视化、表示能力强大
  - 独立于过程
  - 概念明确，建模表示法简洁，图形结构清晰，容易掌握使用。

UML和程序设计语言的关系

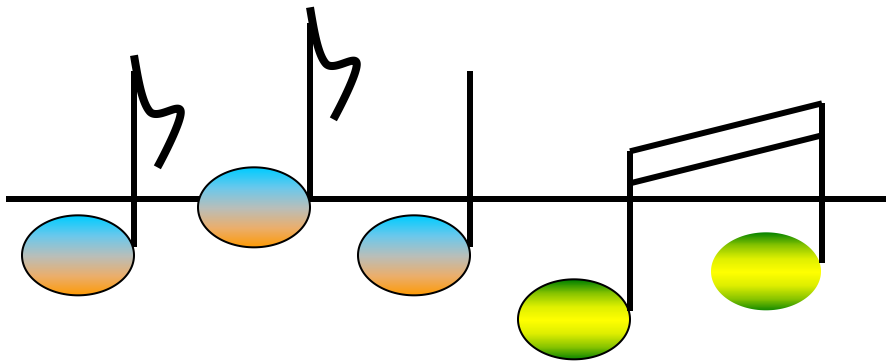
- 用Java，C++ 等 programming language是用编码实现一个系统。
- 用UML是对一个系统建立模型。
- 一些CASE工具可以根据 UML所建立的系统模型来产生Java，C++ 或其它程序设计语言代码框架。

关于UML的一些认识

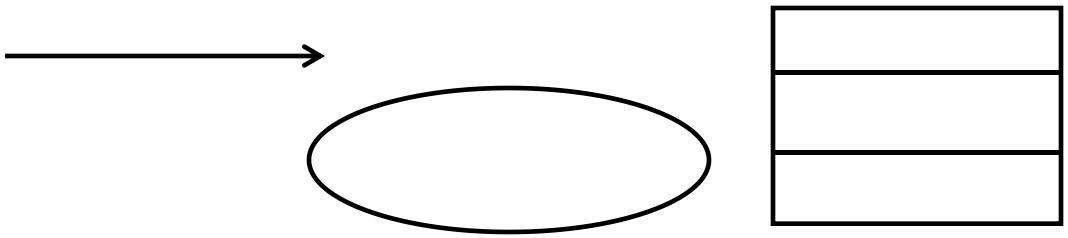
- UML不是一个独立的软件工程方法，而是面向对象软件工程方法中的一个部分。
- 一个比喻：UML中所提供标准的图符，相当于音乐五线谱里的乐符，学会看乐符才能看得懂乐谱，才能进一步创造音乐。同样，懂得UML中的图符才能进行系统分析和设计。
- OOA/OOD教您活用UML的图符，以及活用时所必须遵循的原则及步骤。

一个比喻

贝多芬作曲时使用五线谱



您设计软件时使用UML



说明：对于同一个系统，不同人员所关心的内容并不一样：

- 分析人员和测试人员关心的是系统的行为，因此会侧重于用例视图；
- 最终用户关心的是系统的功能，因此会侧重于逻辑视图；
- 程序员关心的是系统的配置、装配等问题，因此会侧重于实现视图；
- 系统集成人员关心的是系统的性能、可伸缩性、吞吐率等问题，因此会侧重于进程视图；
- 系统工程师关心的是系统的发布、安装、拓扑结构等问题，因此会侧重于部署视图。

UML在系统开发各阶段的应用

- 在**分析阶段**，用户的需求用UML模型来描述。
- 在**设计阶段**，引入定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。
- 在**实现阶段**，用面向对象程序设计语言将来自设计阶段的类转换成实际的代码。
- UML模型还是**测试阶段**的依据。
  - 单元测试使用类图和类规格说明
  - 集成测试使用构件图和协作图
  - 系统测试使用用例图来验证系统的行为

UML的概要

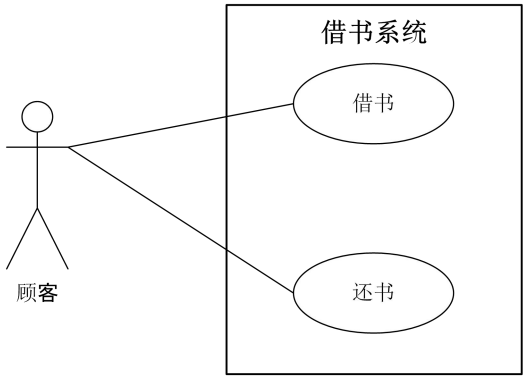
◆ UML1.5中定义了9种图形，UML2.0中又追加定义了4种图形，共13种图形。
◆ 每种图形在不同的工程阶段，从不同的侧面对系统进行表示。
◆ 并不一定要使用所有的图形，根据需求选择合适的图形进行绘制。
◆各种图形的特征如下：

类型	图名		概要说明
静态结构图	类图		表示分析设计领域的静态结构。
	对象图		表示系统中的多个对象在某一时刻的状态。
	包图		对UML的模型元素进行分组整理，还可表示多个包之间的关系。
	构件图		表示构成软件的各个可重复再利用的部件的内部结构以及部件之间的相互关系。
	组合结构图		表示具有复杂内部结构的类或者构件的内部结构。
	部署图		表示系统执行时的硬件构成以及硬件节点之间的关系。
动态行为图	用例图		表示从系统外部看到的系统所提供给用户可使用的功能。
	活动图		表示某个业务的具体处理流程，或者程序的某个算法。
	交互图	顺序图	按照时间顺序从上到下来表示生命线之间的消息交互。可结合用例图用来表示某个用例的脚本。
		通信图	以参与交互的生命线之间的连接为着眼点来表示生命线之间的消息交互。
		交互概览图	用活动图的形式来表示多个交互之间的控制关系。
		时序图	表示多个生命线的状态变化和时间之间的关系。
	状态图		表示某个对象在其生命周期内，各个状态之间的迁移变化以及引起状态迁移的触发事件之间的关系。

用例图(usecase diagram)

- 表示的是从外部看到的系统所具有的功能。
- 主要在需求分析阶段使用。根据从用户收集来的需求，明确开发对象时使用。
- 组成要素：
  - 参与者（actor）
  - 用例（usecase）
  - 系统边界（可选）

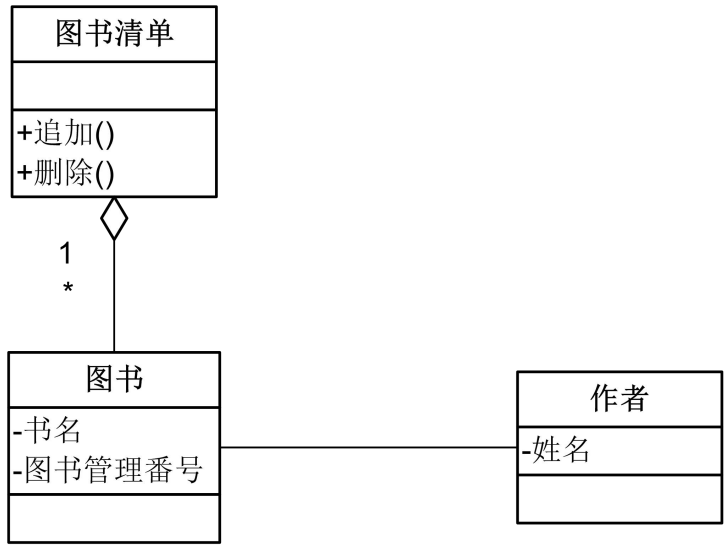
顾客利用系统借书的用例图：



类图(Class Diagram)

- 类就是问题域的静态概念。
- 类由类名，属性和操作三部分组成。
- 类图用来描述系统的静态结构。

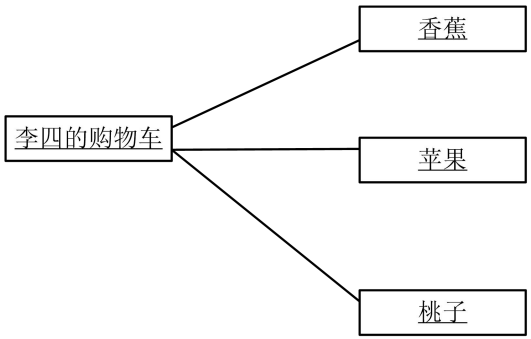
图书管理系统的类图：



对象图(Object Diagram)

- 对象图描述在某一时刻系统内对象的状态。
- 可用来辅助决定类图中类和类之间的多重度关系。

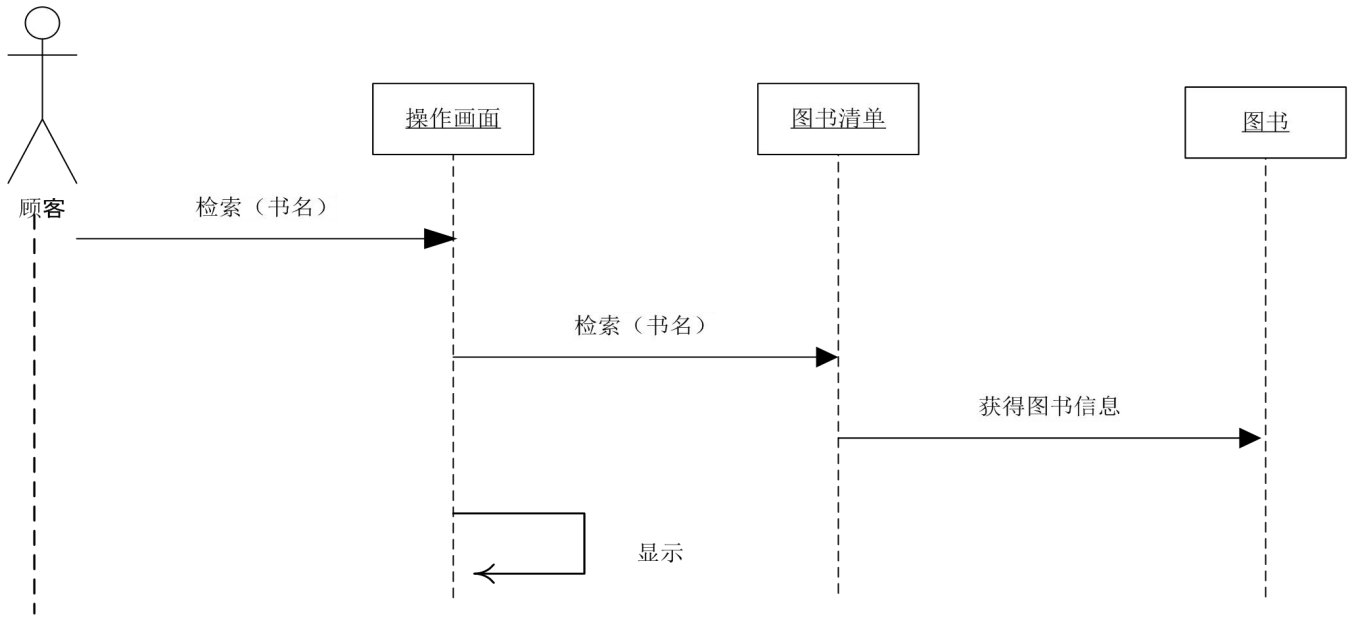
购物系统的对象图：



顺序图(Sequence Diagram)

- 交互图的一种，描述系统的动态行为
  - 按照时间顺序来描述生命线（对象，组件实例）之间的消息交换
- 顺序图的组成部分
  - 生命线
  - 消息交互

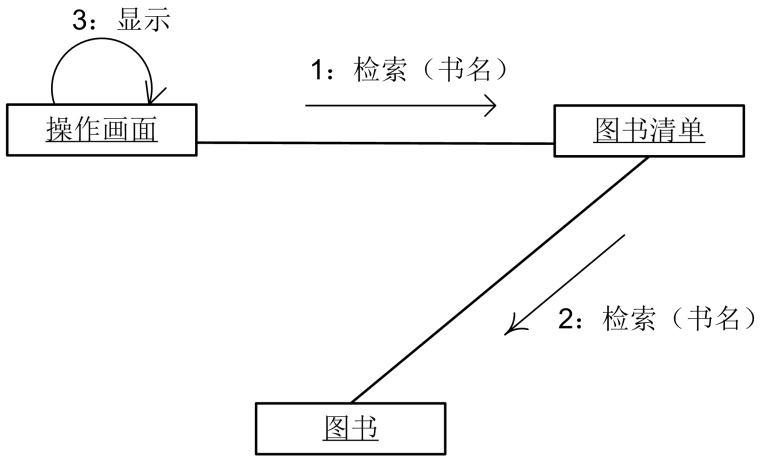
网上书店系统中“检索书籍”用例的顺序图：



通信图(Communication Diagram)

- 交互图的一种，用来描述系统的动态行为
  - 以生命线之间的连接关系为中心来描述消息交换的。
- 通信图的组成部分
  - 生命线
  - 消息

网上书店系统中“检索书籍”用例的通信图：



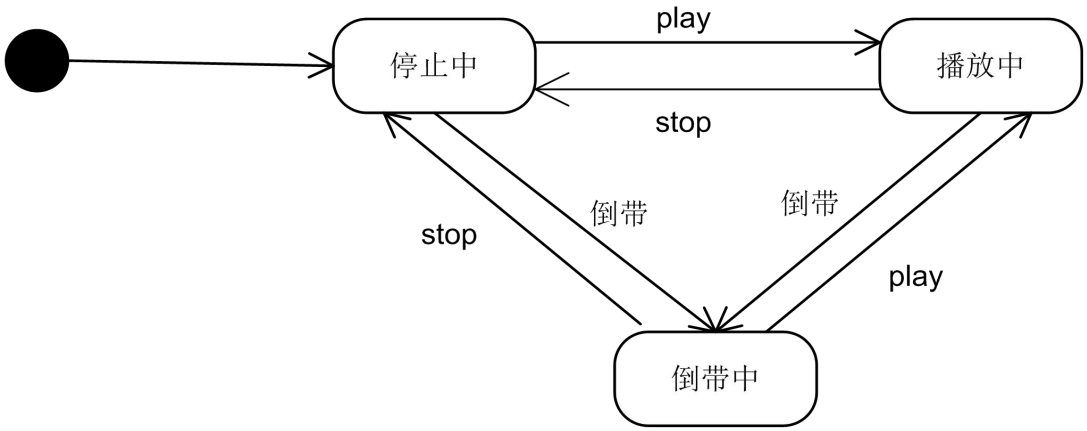
状态图(Statemachine Diagram)

◆ 描述某个对象在生命周期内随着外部事件的触发而进行的状态变化的图。

- 着眼于一个对象，描述该对象在生命周期内的状态迁移变化情况。

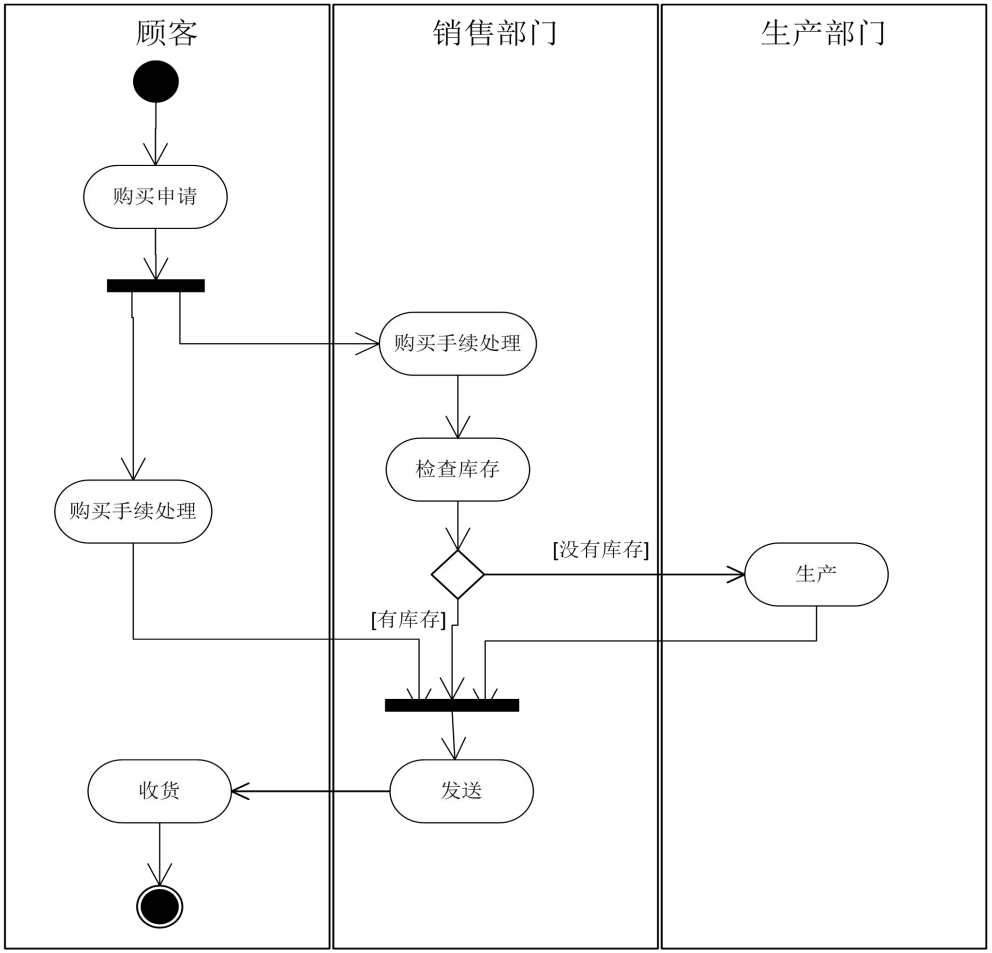
◆ 状态图的组成部分

- 状态（初始状态，终了状态）
- 事件



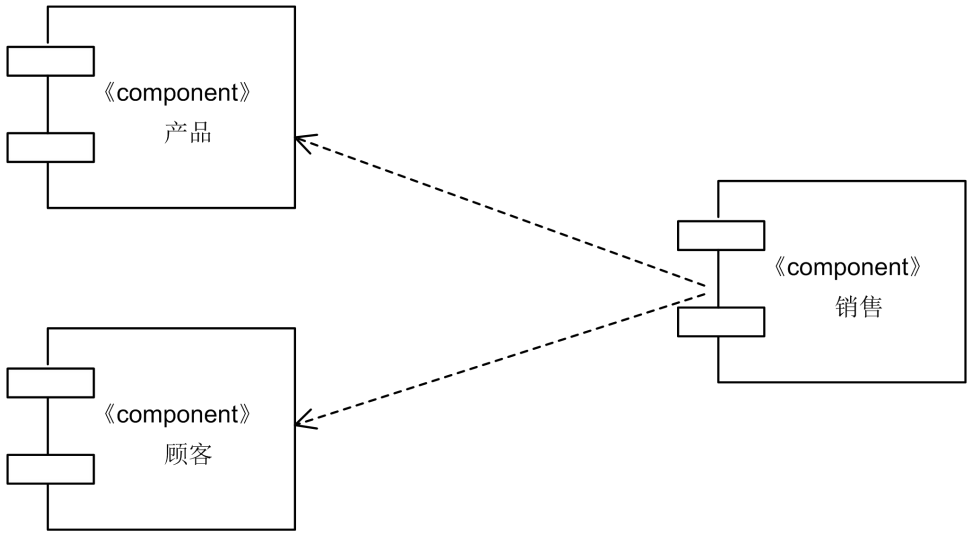
活动图(Activity Diagram)

- 描述系统业务流程，用例流程的图
- 活动图的组成部分
  - 开始节点，终了节点
  - 活动
  - 判断节点和监护条件
  - fork节点
  - join节点
  - 参与者



构件图(Component Diagram)

- 描述构成系统的软件构件的结构及其相互关系的图
  - 构件为构成软件系统的可以再利用的部品。
  - 在设计阶段使用
- 构件图的组成部分：
  - 构件
  - 构件之间的关系



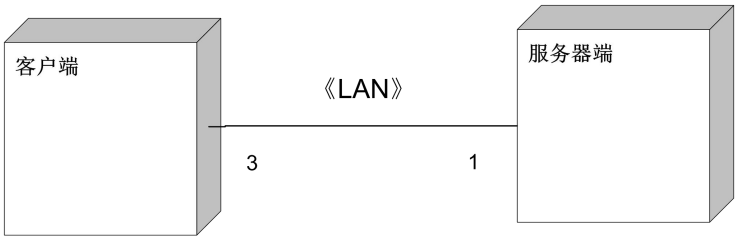
部署图(Deployment Diagram)

◆ 描述软件运行时的硬件，配置在硬件上的软件和硬件节点之间关系的图。

◆ 部署图的组成部分：

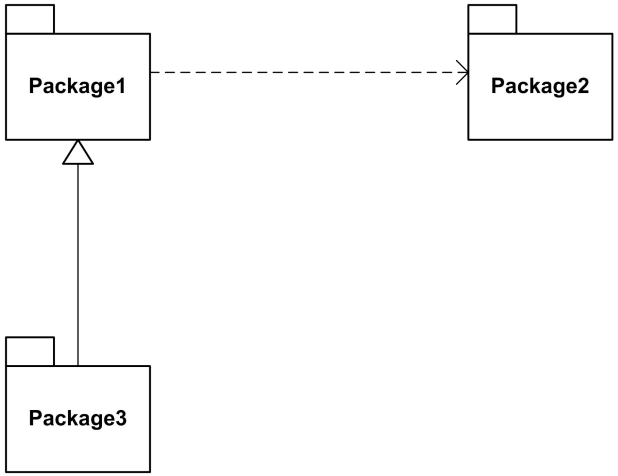
–节点

– 节点之间的关系



包图(Package Diagram)

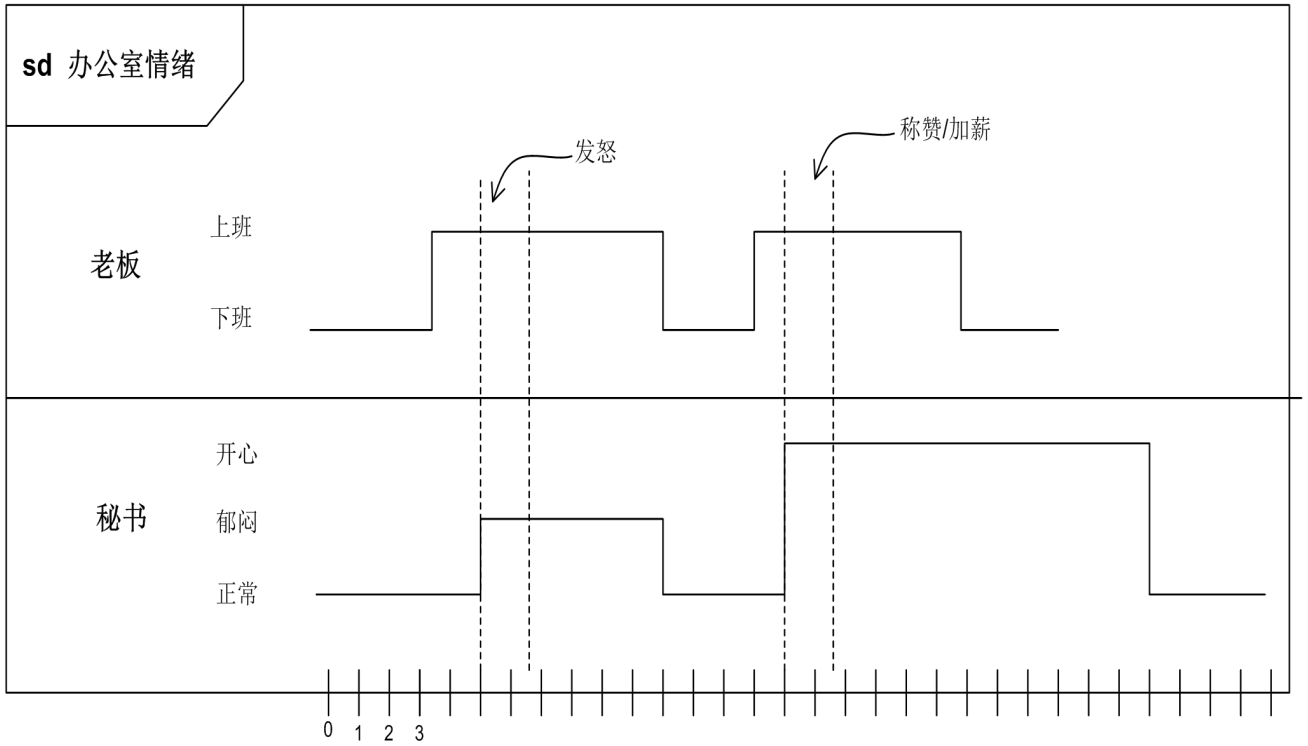
- 对构成系统的模型元素进行分组整理的图
- 包图的组成部分：
  - 包（package）
  - 包之间的关系



注：UML2.0新追加图形

时序图(Timming Diagram)

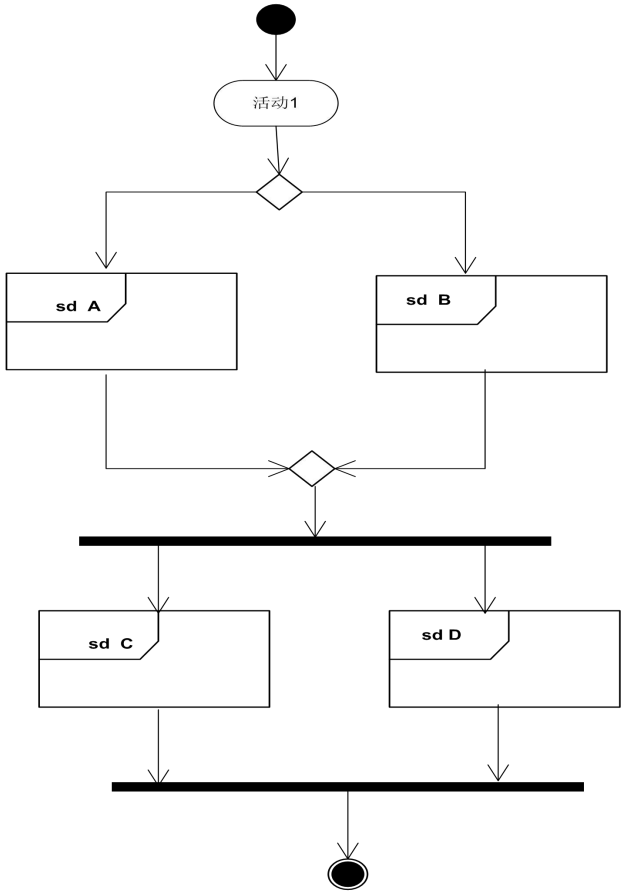
- 从左至右按时间顺序表示生命线状态变化的图
  - 也可像顺序图那样表示多个生命线之间的消息交互。
- 时序图的组成部分：
  - 生命线
  - 状态
  - 事件



注：UML2.0新追加图形

交互概览图(Interaction Overview Diagram)

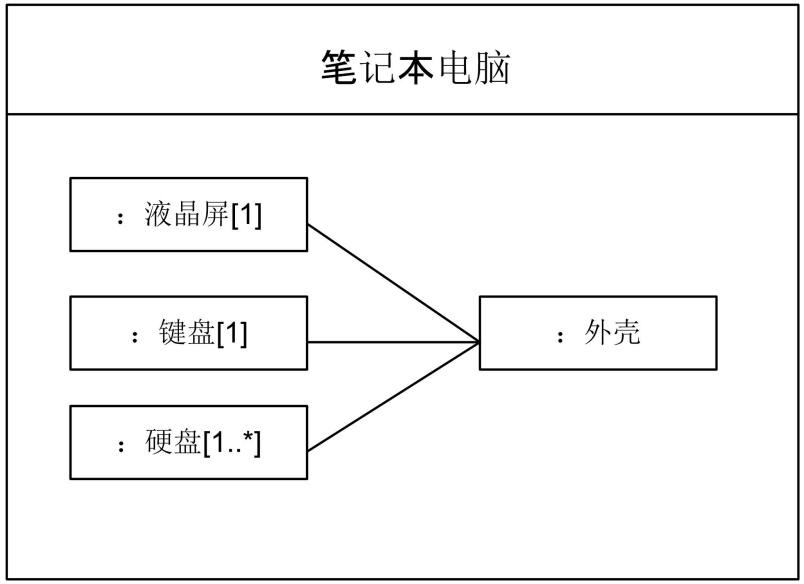
- 用活动图来表示多个交互之间的控制关系的图
  - 交互概览图中的每一个交互可以是一个顺序图或者是通信图。
- 交互概览图的组成部分：
  - 交互
  - 交互调用
  - 条件分支



注：UML2.0新追加图形

组合结构图(Composite Structure Diagram)

- 用来表示类或者组件内部结构的图
- 组合结构图的组成部分：
  - 组合结构类别
  - 组成部分（part）
  - 连接(connect)



注：UML2.0新追加图形

UML应用领域

- 最常用的是为软件系统建模，但不限于软件系统建模。UML还可用来描述其他非软件系统，如一个机构的组成或机构中的工作流程等。

UML应用---需求分析阶段

- UML 的用例视图可以表示客户的需求，通过用例建模可以对外部的角色以及它们所需要的系统功能建模。

UML应用---分析阶段

- 分析阶段主要考虑所要解决的问题。可用UML 的逻辑视图和动态视图来描述，类图描述系统的静态结构；协作图、状态图、顺序图和活动图描述系统的动态特征。

- 在分析阶段，只为问题领域的类建模，不定义软件系统的解决方案的细节（如用户接口的类数据库等）。

UML应用---设计阶段

- 在设计阶段把分析阶段的结果扩展成技术解决方案，加入新的类来提供技术基础结构---用户接口，数据库操作等。

- 分析阶段的领域问题类被嵌入在这个技术基础结构中，设计阶段的结果是构造阶段的详细的规格说明。

UML应用---构造阶段

- 在构造（或程序设计）阶段把设计阶段的类转换成某种面向对象程序设计语言的代码。

UML应用---测试阶段

- 对系统的测试通常分为单元测试、集成测试、系统测试和接受测试几个不同级别。

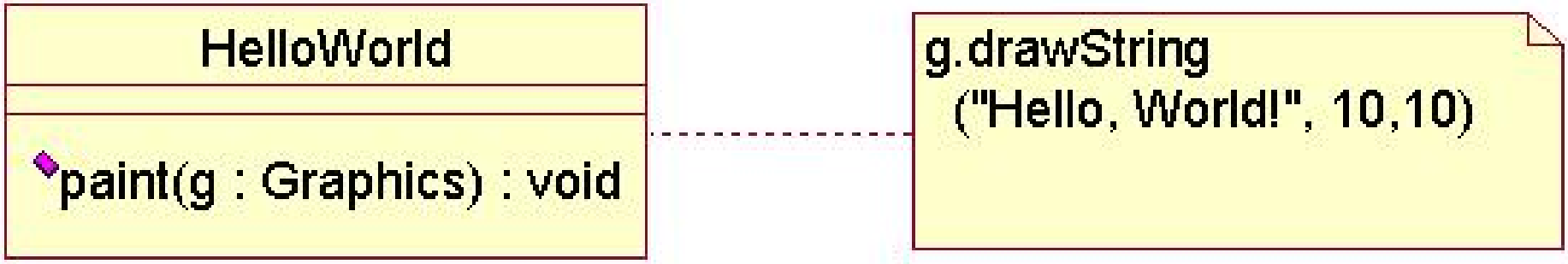
- 不同的测试小组使用不同的UML 图作为他们工作的基础。单元测试使用类图和类的规格说明；集成测试典型地使用组件图和协作图；而系统测试实现用例图来确认系统的行为是否符合这些图中的定义。

一个UML的例子 (HelloWorld)

在Web浏览器中，显示“Hello, World!”的Java applet程序：

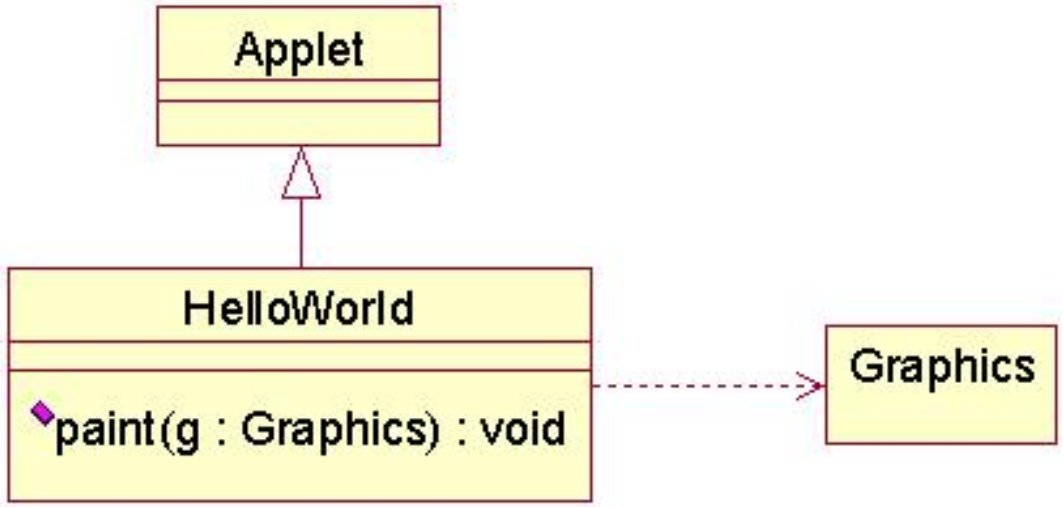
```
import java.awt.Graphics;

public class HelloWorld extends java.applet.Applet {
    public void paint (Graphics g) {
        g.drawString("Hello, World!", 10,10);
    }
}
```

在UML中，对这个applet的建模如下图所示，类HelloWorld 用一个矩形表示。类HelloWorld 中给出了paint操作，在一个附属的note中说明了该操作的实现。

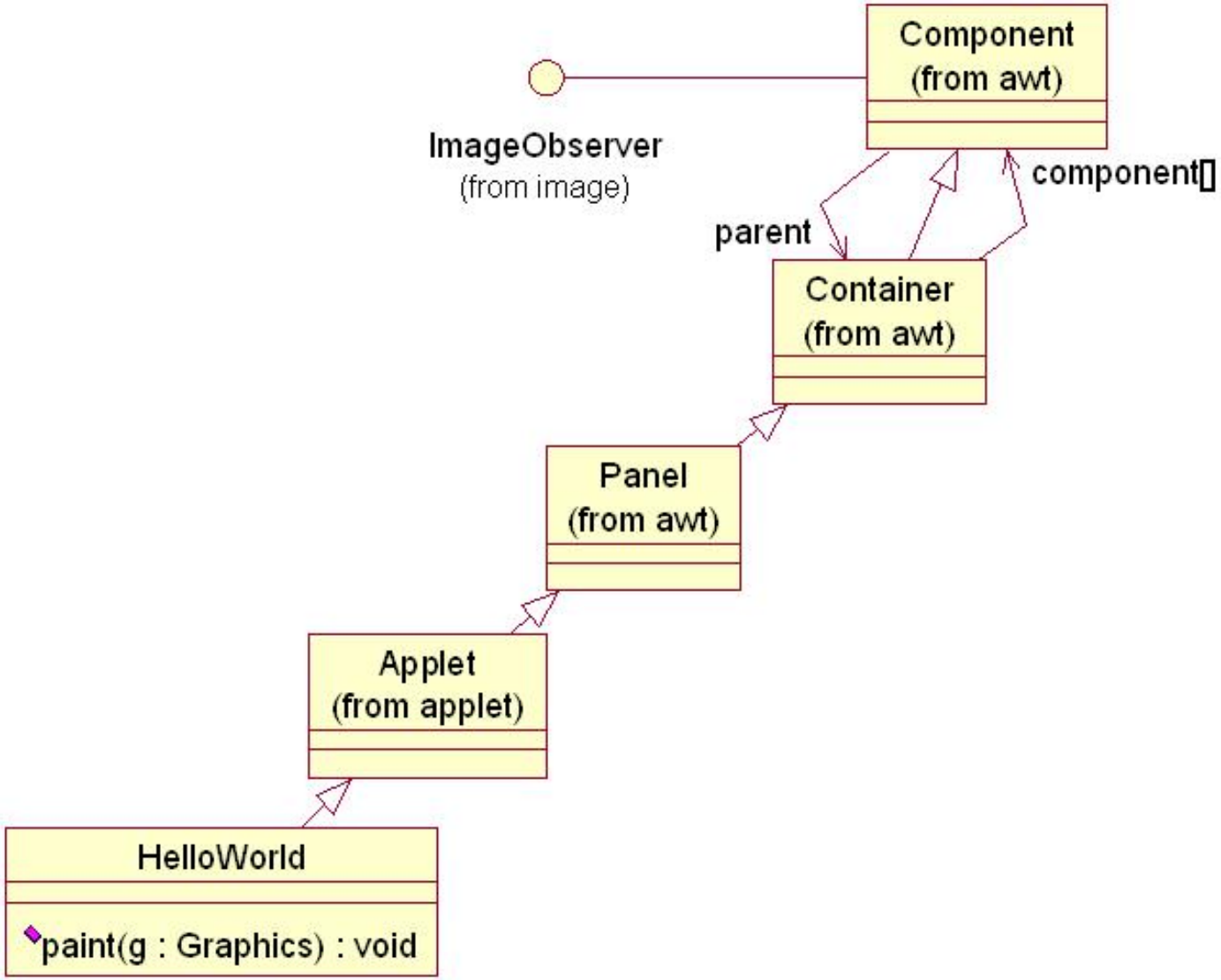


前面这个类图反映出了“Hello World!”这个applet基本部分，并没考虑别的事物。根据代码，这个applet还涉及另外两个类，即Applet类和Graphics类。

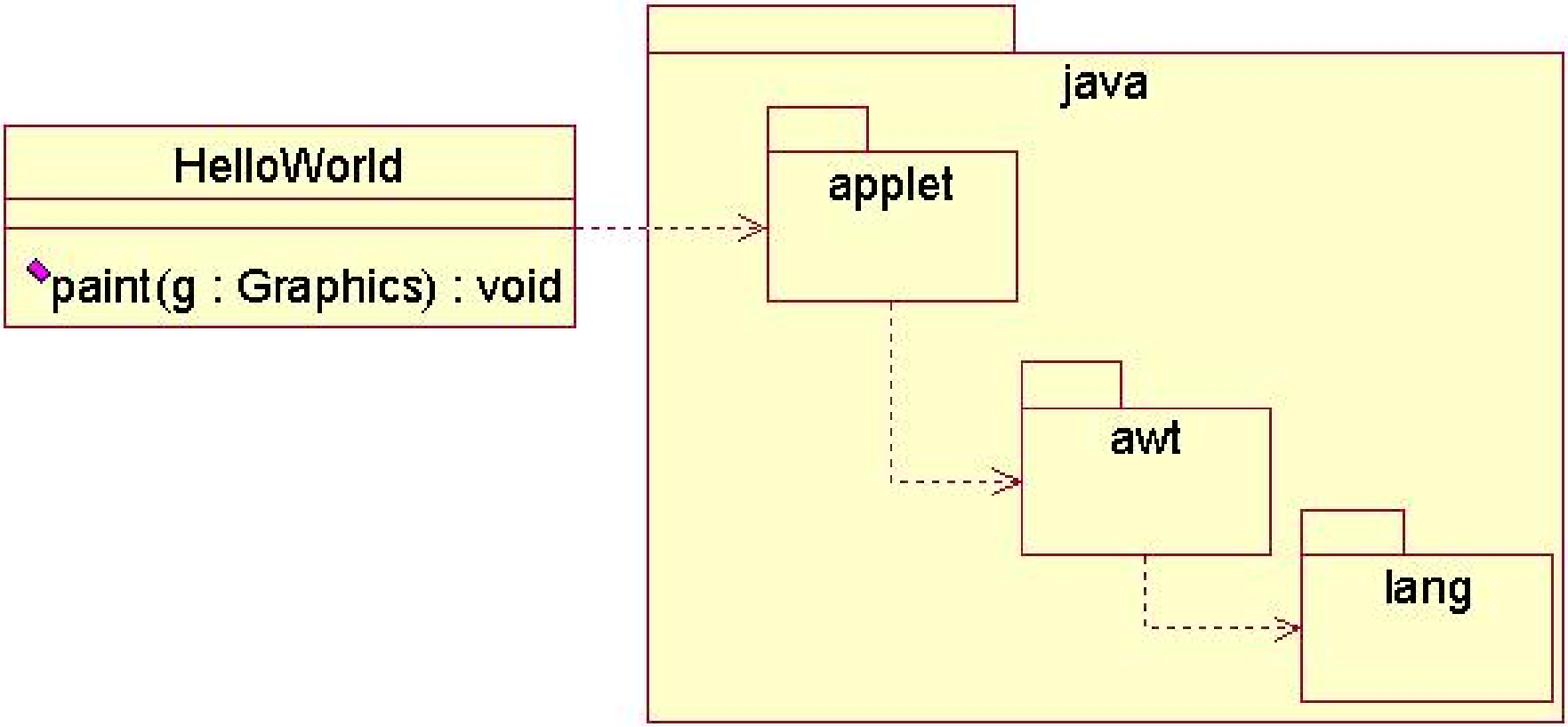
Applet类是HelloWorld类的父类，Graphics类则在HelloWorld类的paint操作的特征标记(signature)和实现中被使用。在UML中，可用下图表示这两个类及与HelloWorld类的关系。



如果考虑类库及Applet上的继承关系，可以得到另一个类图，如下图所示：

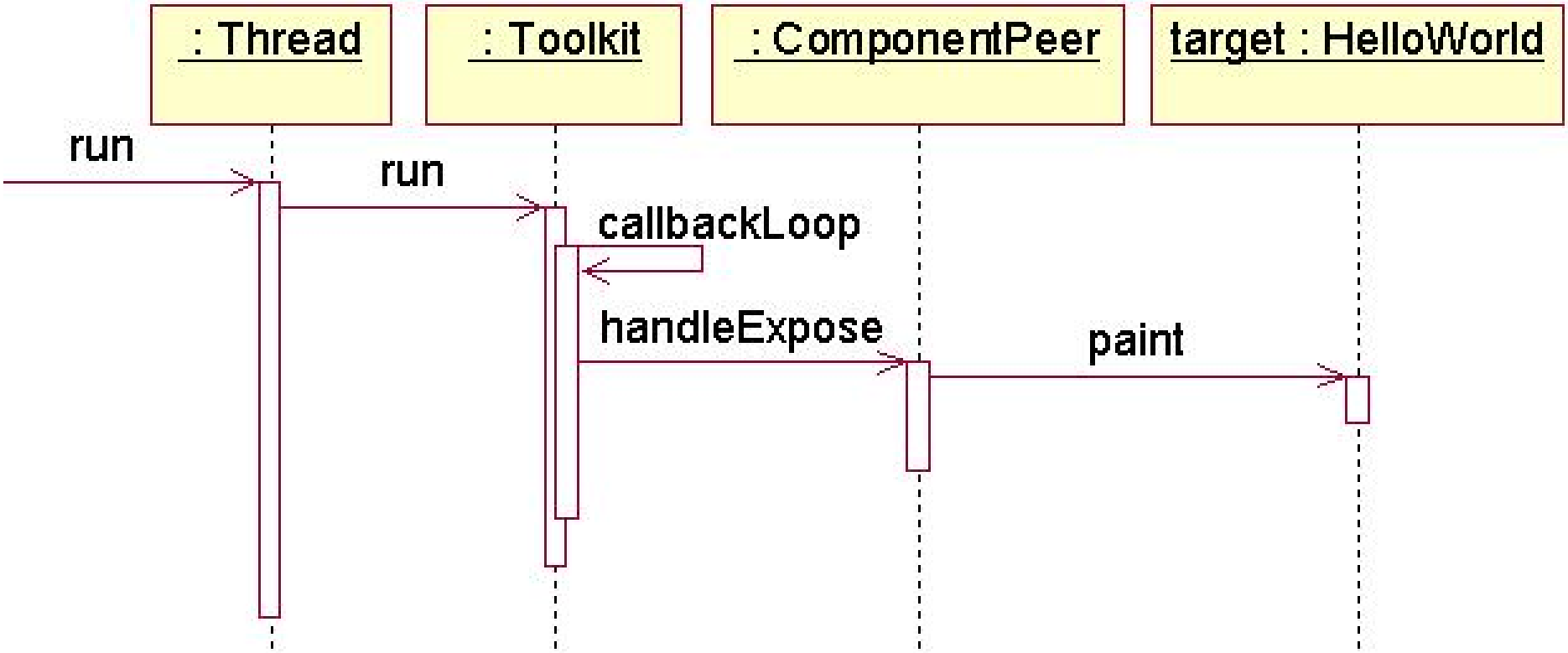


为了管理大规模的类层次图，可以用包来组织类，如下图所示：



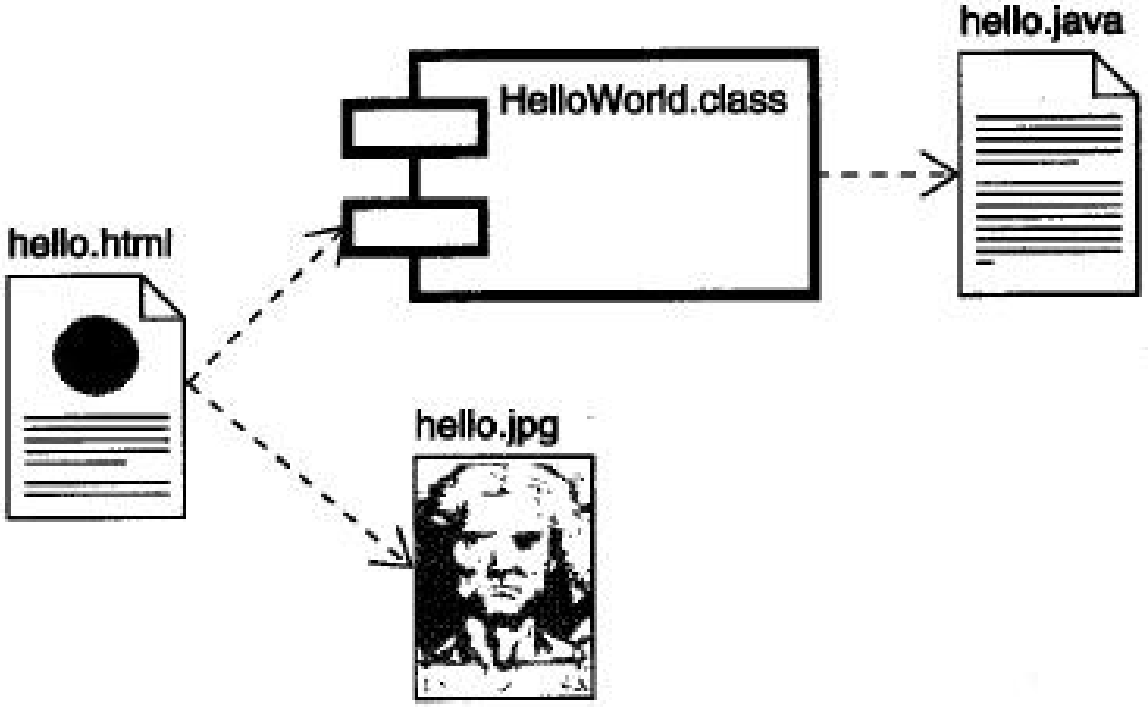
对象间的协同工作

HelloWorld的一个Sequence图



HelloWorld的构件图

“HelloWorld!”是一个applet，不能单独运行，通常是嵌入在Web页中。下面是HelloWorld的构件图：



问题：如何在Rose中画上述图？

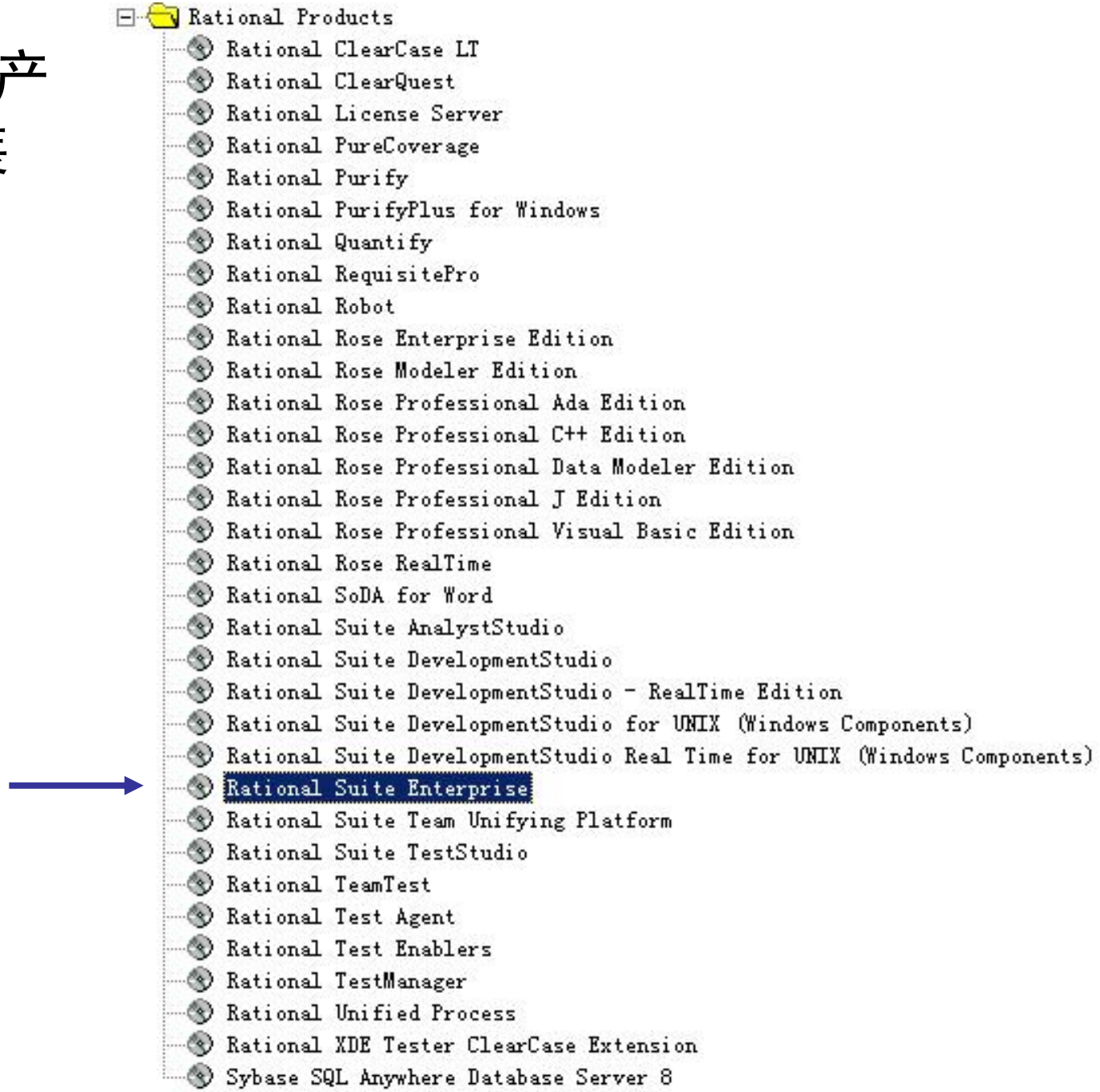
支持UML的工具

- Rational Rose 2003
- Together 6.1
- ArgoUML v0.16.1, 可获得源代码
- MagicDraw UML 8.0
- Visual UML 4.01
- Poseidon for UML 2.5
- ……

Rational Rose开发工具

- **Rose** 是Rational公司开发的支持UML的，用于分析和设计面向对象软件系统的工具。
  - IBM.Rational.Suite.V2003.06.00.Multilanguage

- Rational产品的安装



总结

- 面向对象技术的基本概念
- UML的产生背景，特点，构成，对UML的一些认识等
- Rose开发工具