第二部分：如何运用UML建模

第三章 统一建模语言UML简介

提纲

- 什么是UML?
- UML发展历史
- UML应用目标
- UML建模工具—**Rational Rose**
- UML与RUP

什么是UML?

- UML (Unified Modeling Language)
  - UML是一种对软件系统进行可视化表示、描述、构造和文档化的标准建模语言
  - UML 并不是一种面向对象的开发方法，而是一种可视化的面向对象建模语言
  - UML 把建模语言与开发过程明确进行了分离，专注于建模语言

什么是UML?

- 统一语言
  - 统一：标准的沟通方式
  - 基本词汇：元模型
  - 语法：视图
    - 规则：定义元模型间的关系
    - 图形：如何运用元素与规则绘制图形

什么是UML?

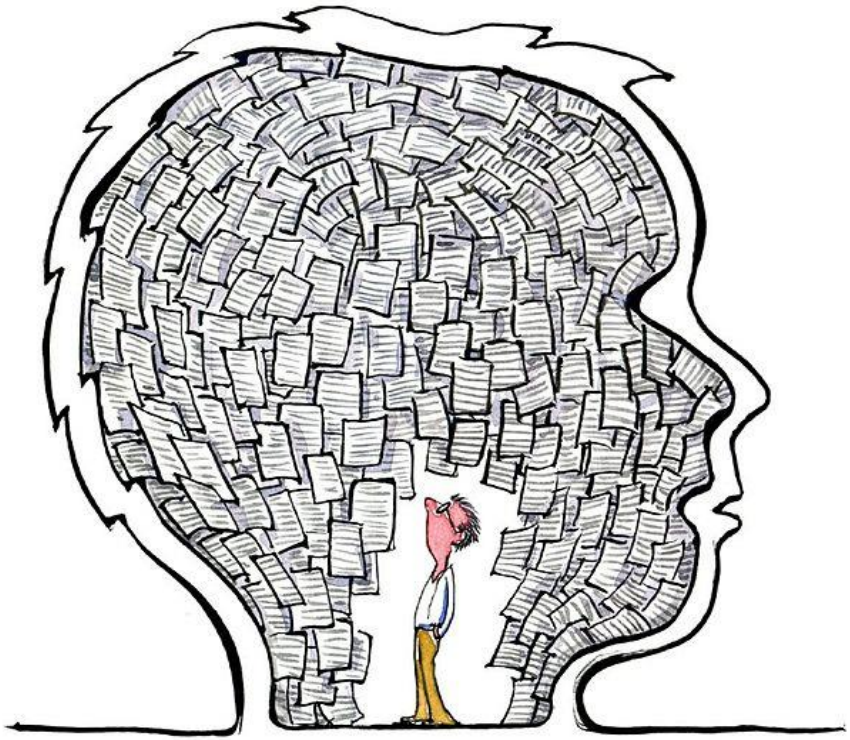
- The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

the artifacts of a software-intensive system.

什么是UML?

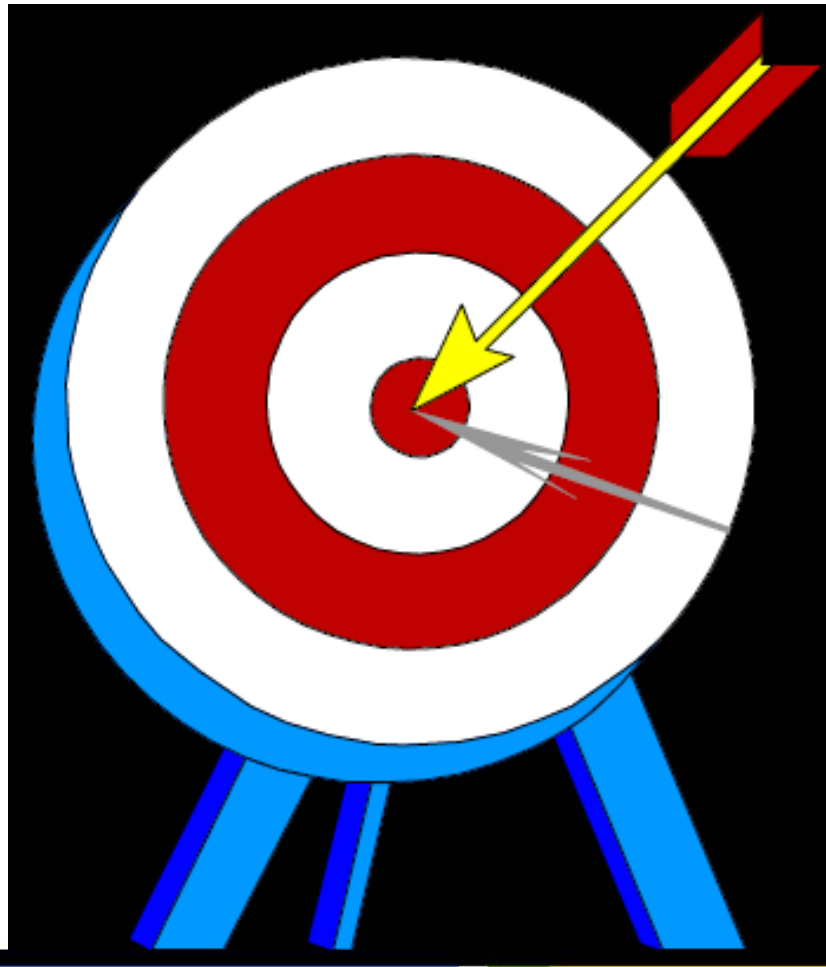
- 可视化建模（Visualizing）
  - 使用标准的图形符号进行建模
  - 清晰的模型有利于交流

“A picture is worth a thousand words.”



什么是UML?

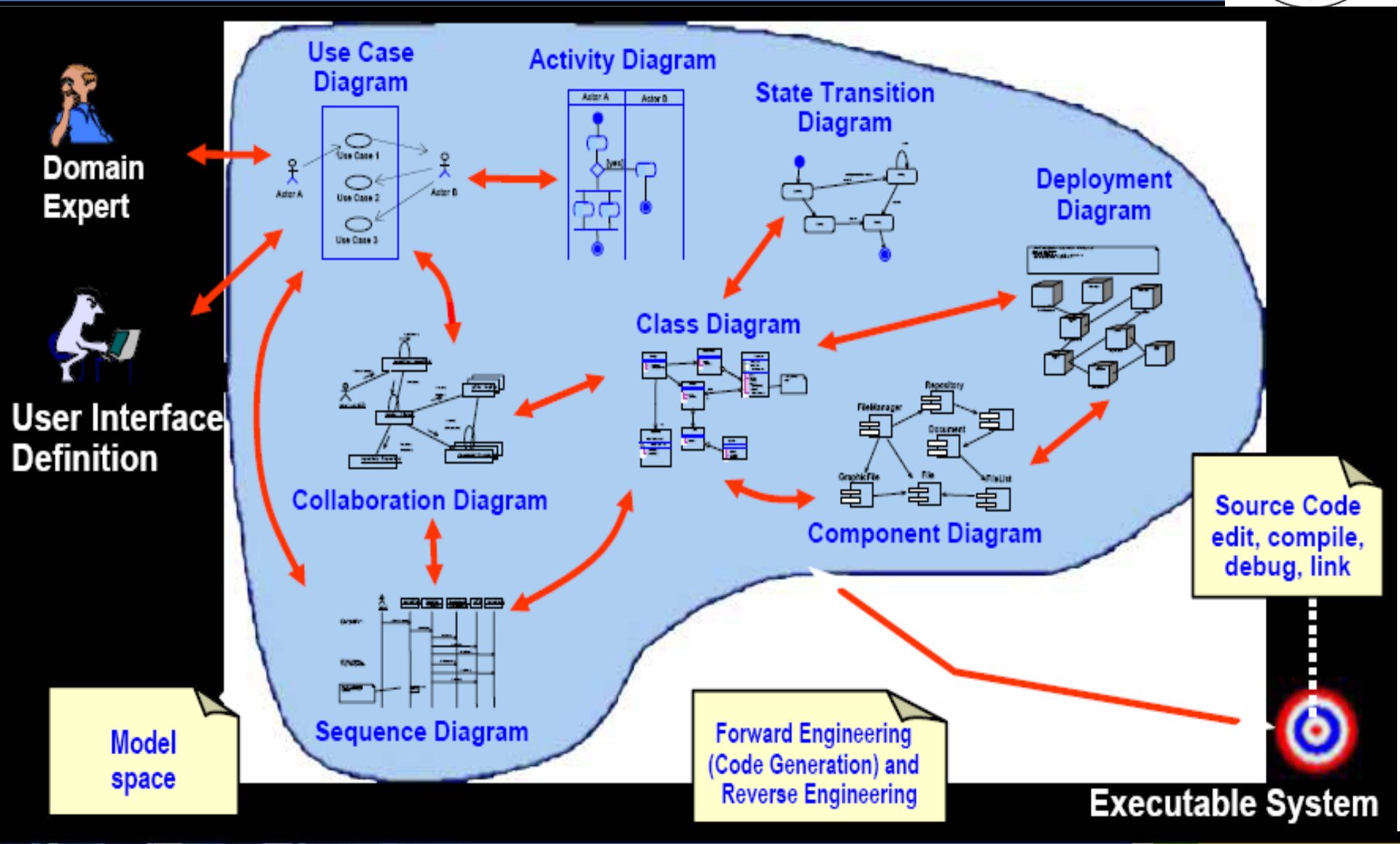
- 可用于详细描述的语言(Specifying)
  - 所建模型是精确、无歧义和完整的



什么是UML?

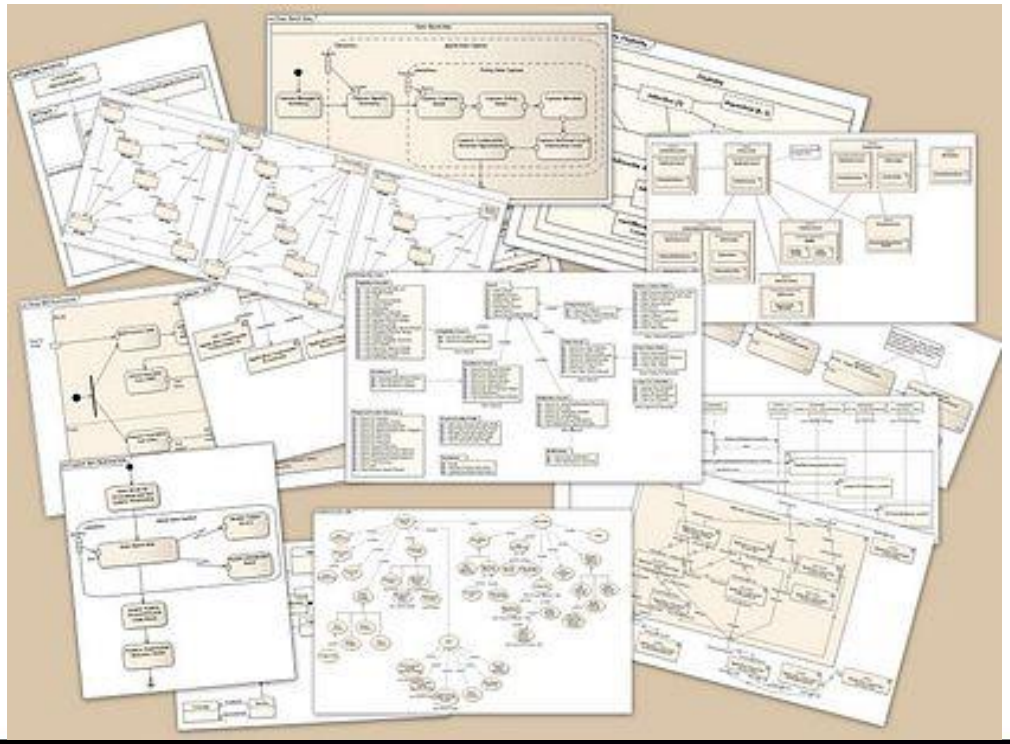
- 构造语言(Constructing)
  - UML描述的模型可与各种编程语言直接相连
  - UML描述的模型可映射到
    - Java, C++, Visual Basic, and so on
    - Tables in a RDBMS
  - 允许正向代码生成
  - 允许逆向工程

什么是UML?


Domain Expert
User Interface Definition
Use Case Diagram
Activity Diagram
State Transition Diagram
Deployment Diagram
Class Diagram
Collaboration Diagram
Component Diagram
Sequence Diagram
Source Code edit, compile, debug, link
Model space
Forward Engineering (Code Generation) and Reverse Engineering
Executable System

什么是UML?

- 文档化语言(Documenting)
  - UML 可用于建立系统体系结构、需求、测试、项目计划及发布管理活动等的所有细节文档



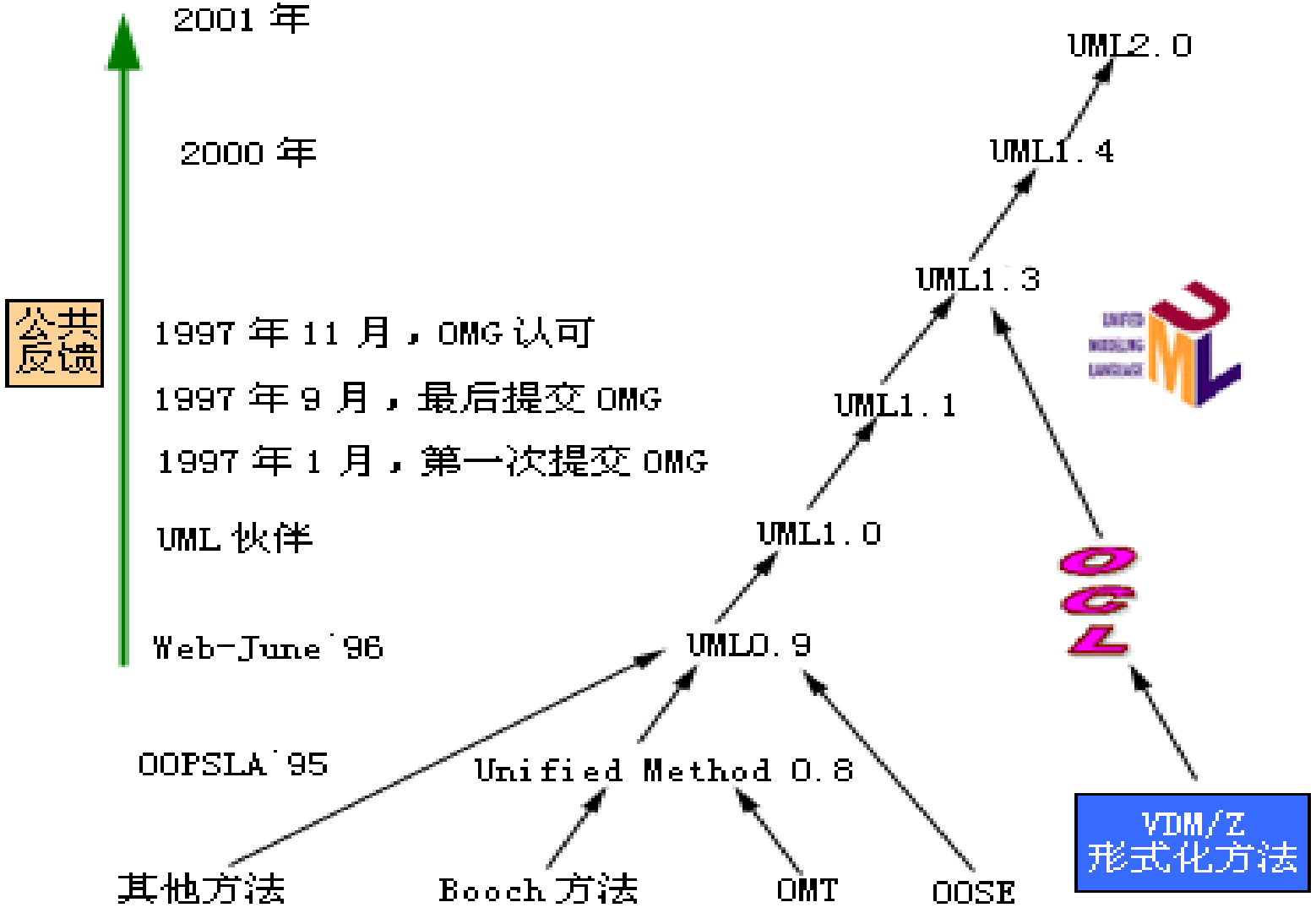
UML发展历史

- 60年代后期：Simula67，基本思想
- 70年代后期：Smalltalk80，实用化
- 80年代：理论基础，C++等，商业化
- 90年代：面向对象分析与设计方法学
  - B.H.Sellers等提出喷泉模型
  - G.Booch提出面向对象开发方法等
  - P.Coad和E.Yourdon提出OOA和OOD
  - J.Rumbaugh提出OMT
  - Jacobson提出OOSE
- 1997年：被OMG组织采纳为国际标准

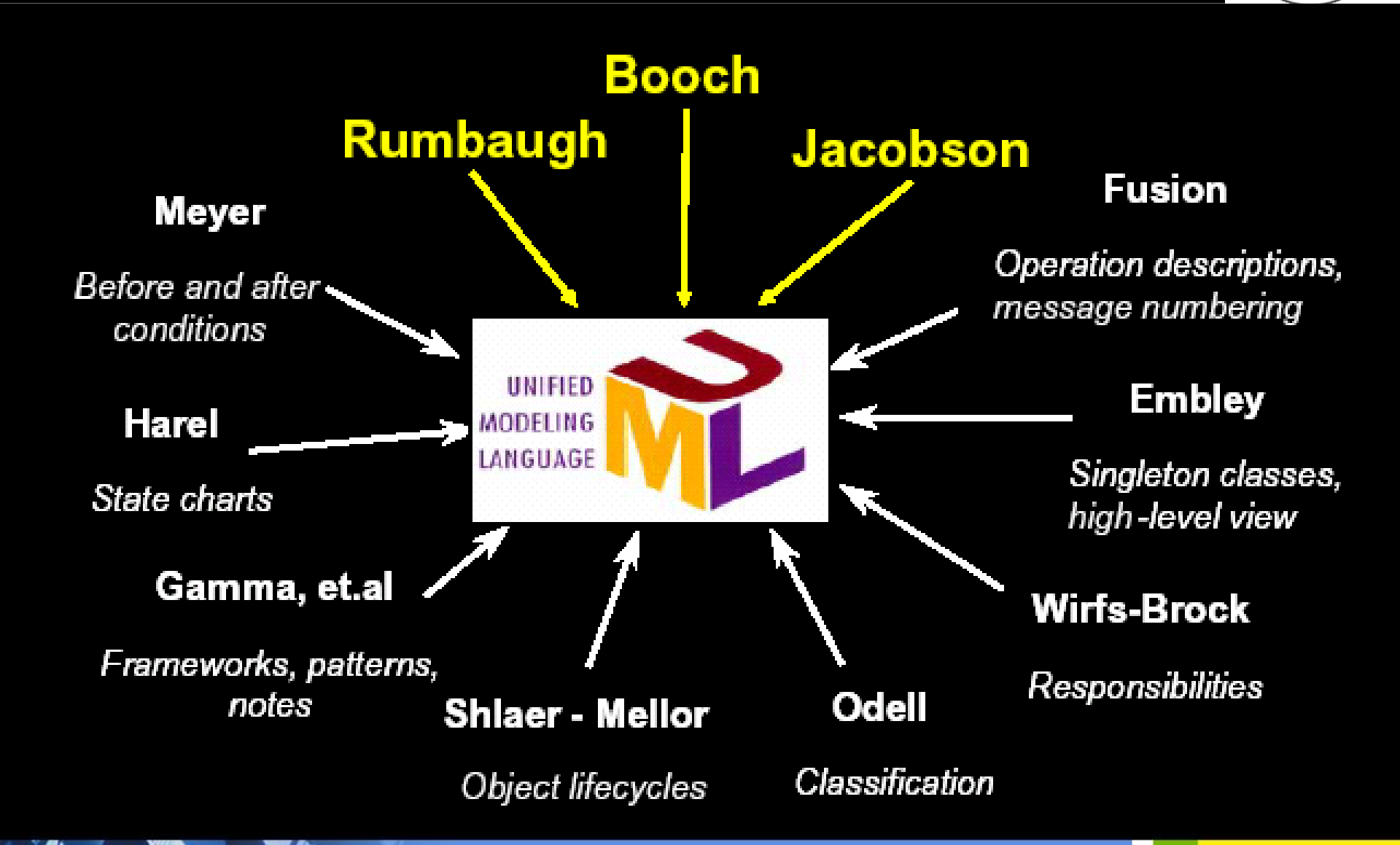
面向对象程序设计语言

“方法大战”

UML发展历史


2001 年
2000 年
公共反馈
1997 年 11 月，OMG 认可
1997 年 9 月，最后提交 OMG
1997 年 1 月，第一次提交 OMG
UML 伙伴
Web-June'96
OOPSLA'95
其他方法
Booch 方法
Unified Method 0.8
OMT
OOSE
UML0.9
UML1.0
UML1.1
UML1.3
UML1.4
UML2.0
OCL
VDM/Z
形式化方法

UML发展历史


Booch
Rumbaugh
Jacobson
Meyer
Before and after conditions
Fusion
Operation descriptions, message numbering
UNIFIED MODELING LANGUAGE
Harel
State charts
Embley
Singleton classes, high-level view
Gamma, et.al
Frameworks, patterns, notes
Wirfs-Brock
Responsibilities
Shlaer - Mellor
Object lifecycles
Odell
Classification

UML应用目标

- 提供易用的、表现力强的可视化建模语言
- 提供可扩展、可定制的核心扩充机制
- 不依赖于特定的程序设计语言和开发过程
- 提供形式化基础以利于理解建模语言
- 促进面向对象工具的市场拓展
- 支持高层开发概念（如协同、构架、模式等）
- 集成最好的实践经验

UML建模工具—Rational Rose

- 一种工具，可在UML建模中提供建立、视图、修改和操作组件的能力
- Rose 运行环境
  - Windows NT，Windows 95
  - UNIX（Solaris，HP/UX，AIX，DEC Unix）
- Rose支持Unified、Booch、OMT标记法

UML建模工具—Rational Rose

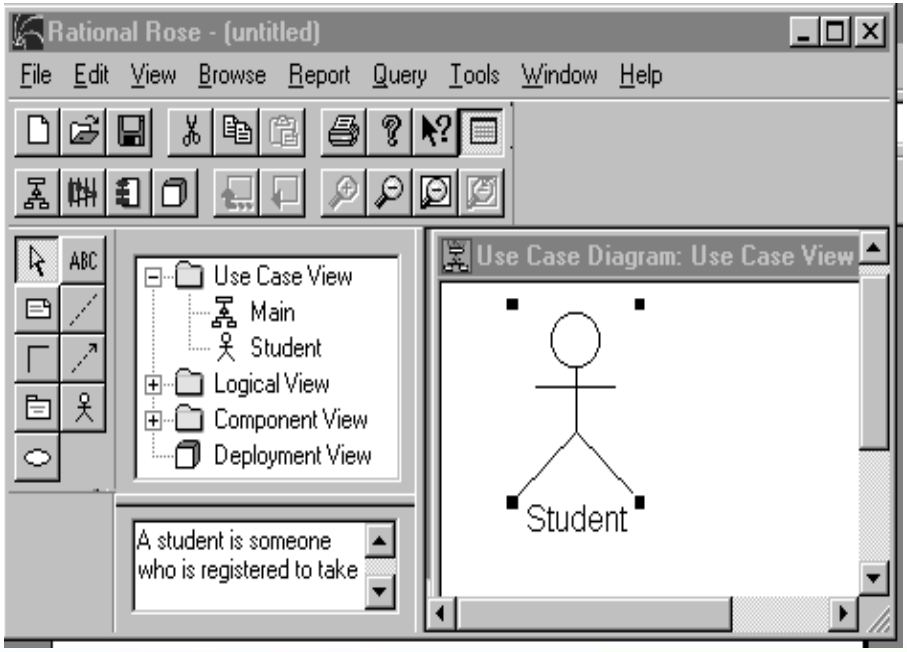
- 在Rose中有四种视图
  - Use Case 视图：包、Actor、Use Case、对象、消息和关系
  - 逻辑视图：包、类、状态和关系
  - 组件视图：包、组件和依附关系
  - 拓扑视图
    - 一个系统在物理设计阶段进程处理的分配情况
    - 节点和关系

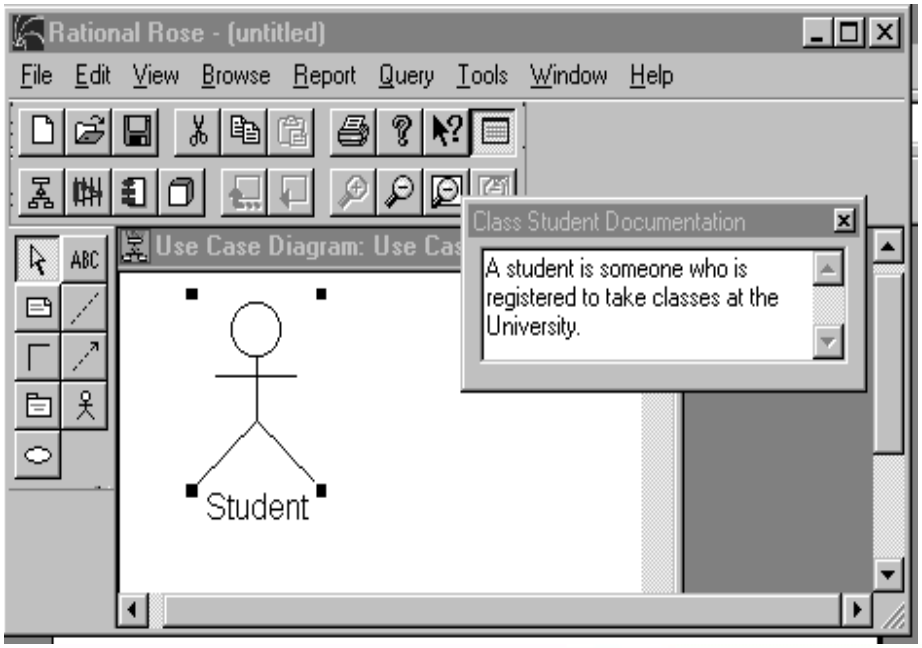
UML建模工具—Rational Rose

- Rose的界面组成

UML建模工具—Rational Rose

- 文档窗口
  - 为所选择的项和图形提供建立、浏览或修改文档的能力组件
  - 可视或被隐藏；固定或浮动

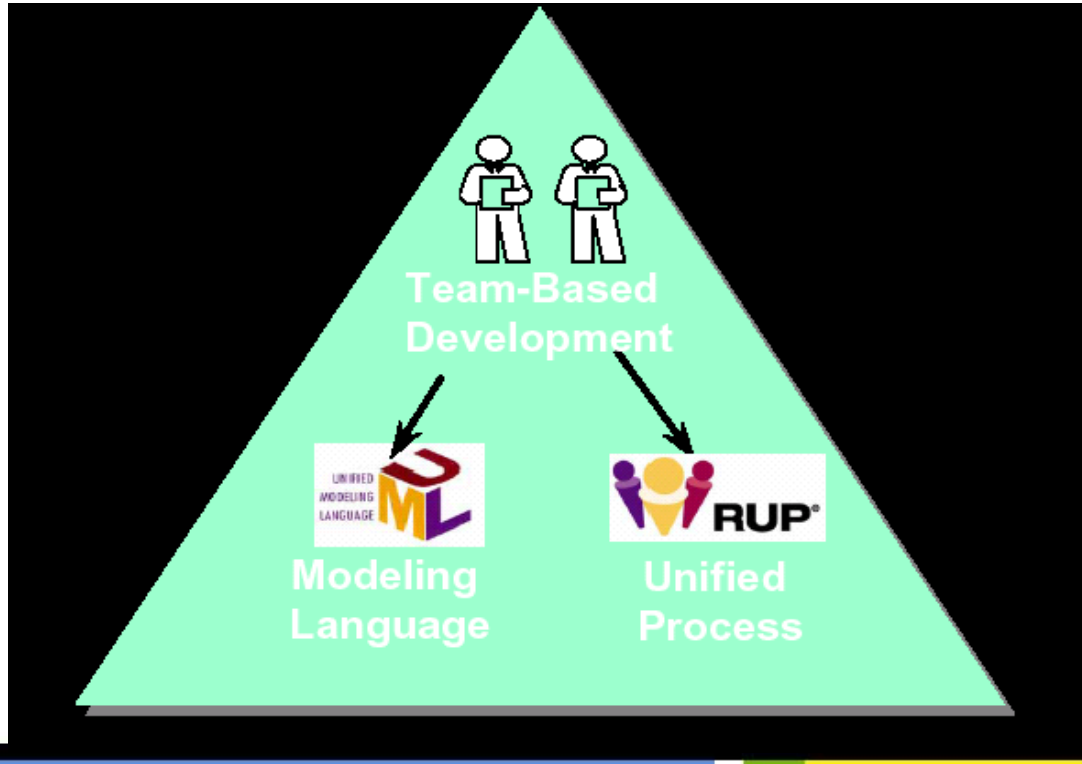




UML与RUP

- RUP（Rational Unified Process）
  - 统一(软件开发)过程
  - 与UML集成和应用最好的软件开发方法

“A language is not enough to build a system.”

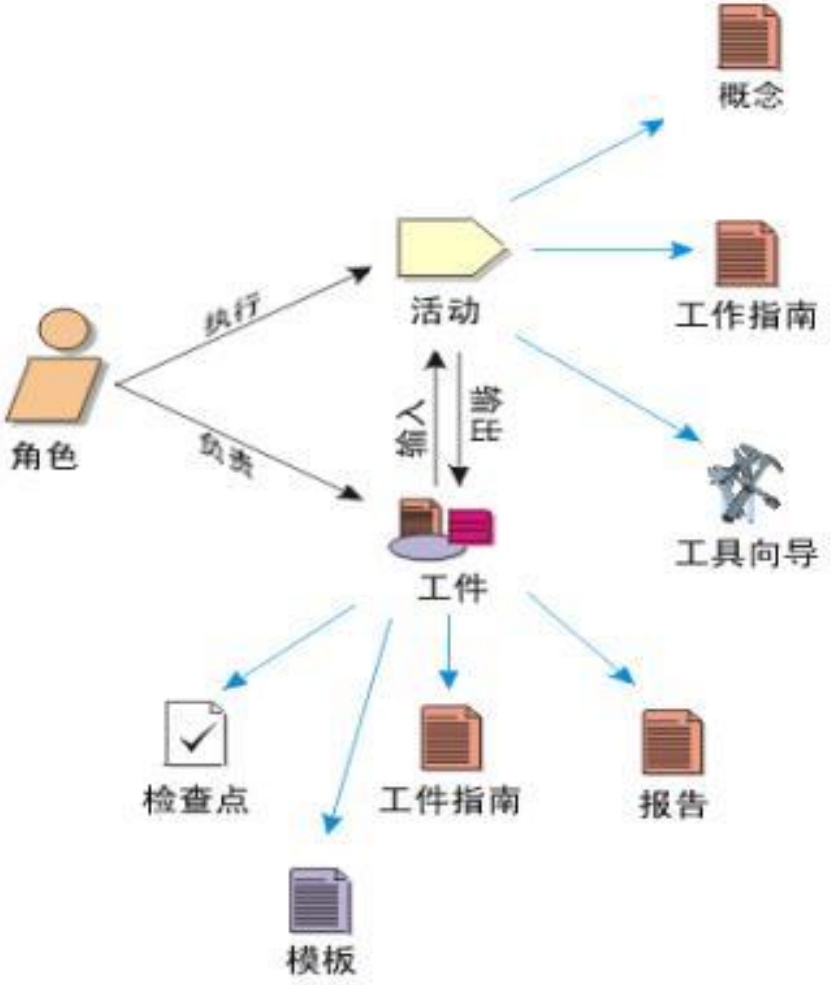


UML与RUP

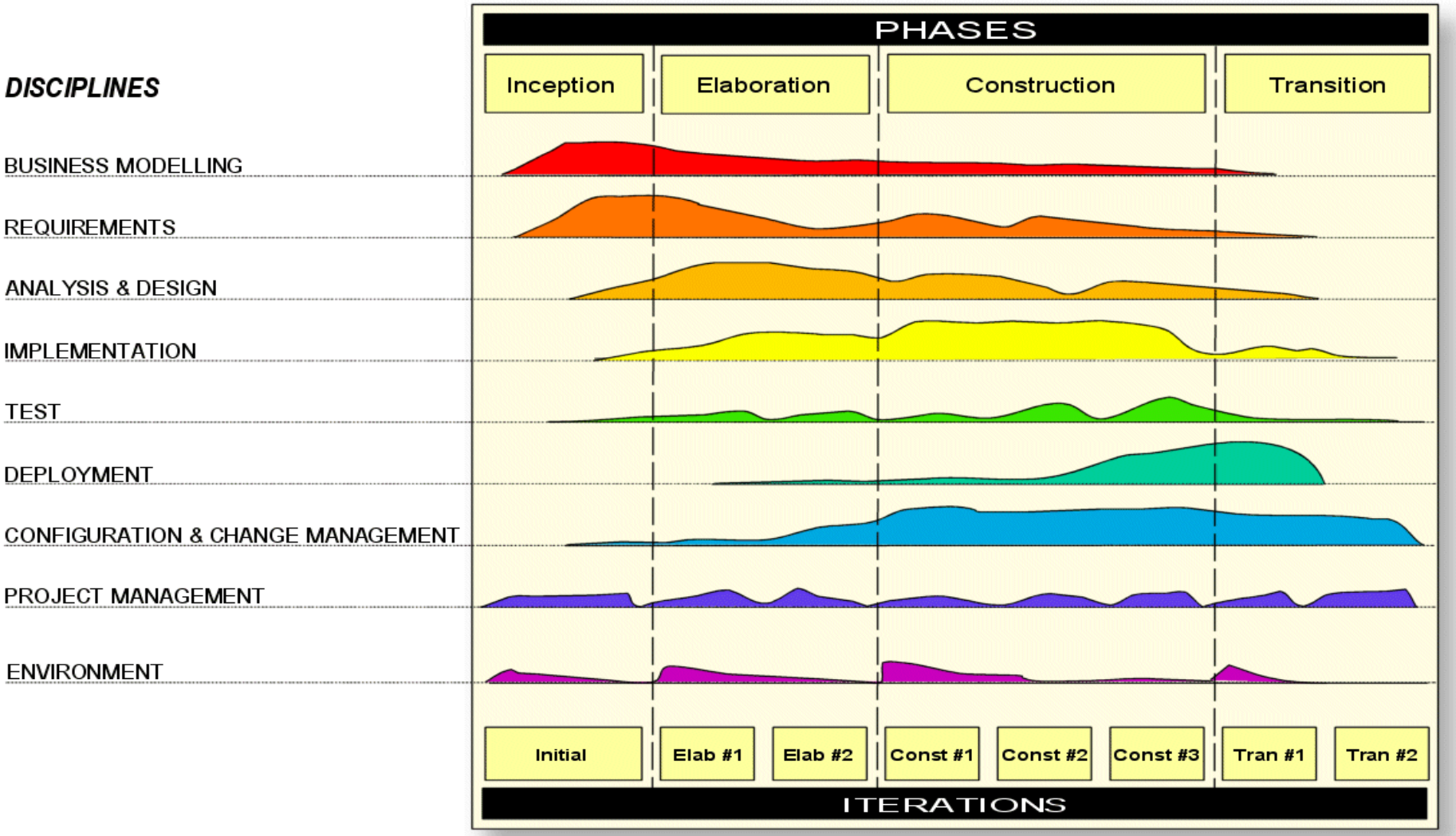
- RUP
  - 采用面向对象思想
  - 使用UML作为软件分析设计语言
  - 结合项目管理等软件工程的最佳实践

UML与RUP

- RUP构造块：
  - Role (who)：行为和职责
  - Activity (how)：要执行的任务
  - Artifact (what)：活动的结果
  - Workflow (when)：工作者执行活动的序列



UML与RUP


PHASES
DISCIPLINES
Inception
Elaboration
Construction
Transition
BUSINESS MODELLING
REQUIREMENTS
ANALYSIS & DESIGN
IMPLEMENTATION
TEST
DEPLOYMENT
CONFIGURATION & CHANGE MANAGEMENT
PROJECT MANAGEMENT
ENVIRONMENT
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #3
Tran #1
Tran #2
ITERATIONS

UML与RUP

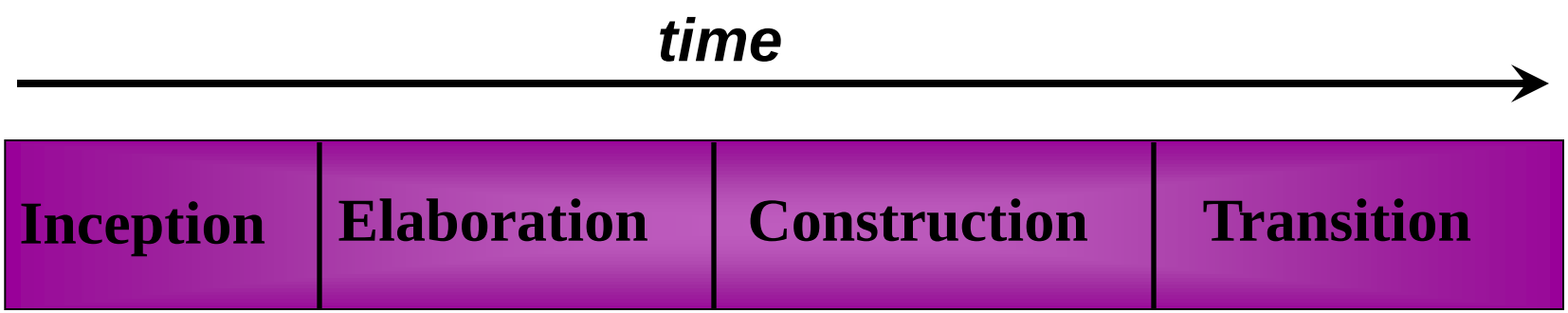
- RUP开发过程的二维结构
  - 纵轴：过程的静态结构
    - 工作流(Workflow)：对应于特定的迭代的连续活动
    - 工作者(Workers)：行为和责任
    - 活动(Activities)：工作者要执行的工作单元
    - 工件(Artifacts)：活动的结果

UML与RUP

- RUP开发过程的二维结构
  - 横轴：时间组织，体现过程的动态结构
    - 周期（Cycles）：每一个周期工作在产品新的一代上
    - 阶段（Phases）：先启、精化、构建、提交
    - 迭代（Iterations）：每个阶段进行若干次
    - 里程碑（Milestones）：迭代正式结束的时间点

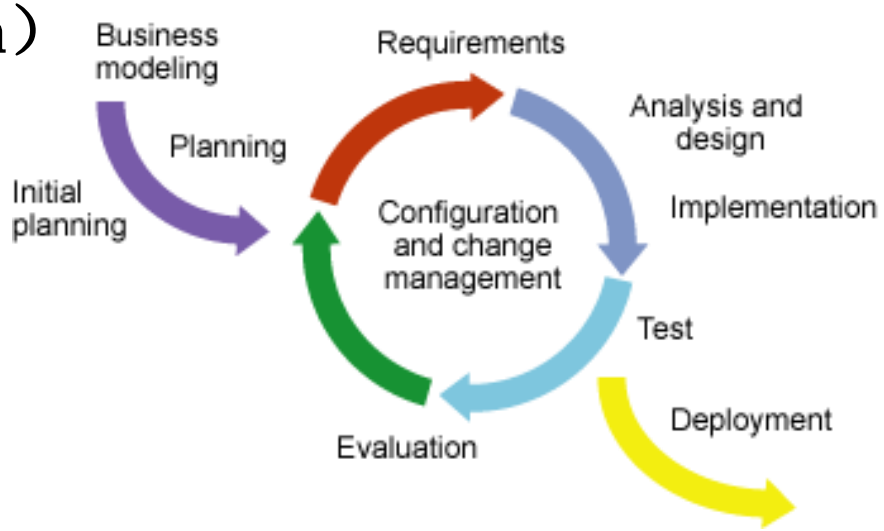
UML与RUP

- RUP中的软件生命周期（四个阶段）
  - 先启：定义项目的范围
  - 精化：计划项目，说明特性和构架基线
  - 构建：建立产品
  - 提交：提交产品到最终用户团体



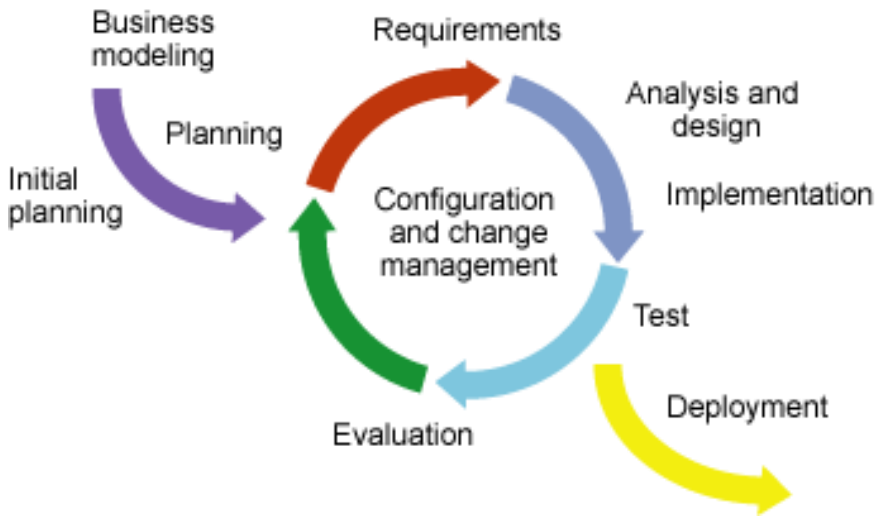
UML与RUP

- RUP的九个核心工作流
  - 核心过程工作流
    - 商业建模（Business Modeling）
    - 需求（Requirements）
    - 分析和设计（Analysis & Design）
    - 实现（Implementation）
    - 测试（Test）
    - 部署（Deployment）



UML与RUP

- RUP的九个核心工作流
  - 核心支持工作流
    - 配置和变更管理（Configuration & Change Management）
    - 项目管理（Project Management）
    - 环境（Environment）



小结

- 重点
  - 了解UML的历史、作用与目标
  - 掌握UML与面向对象、OOAD方法的关系
  - 了解UML开发工具Rational Rose

第二部分：如何运用UML建模

第四章 UML的概念模型

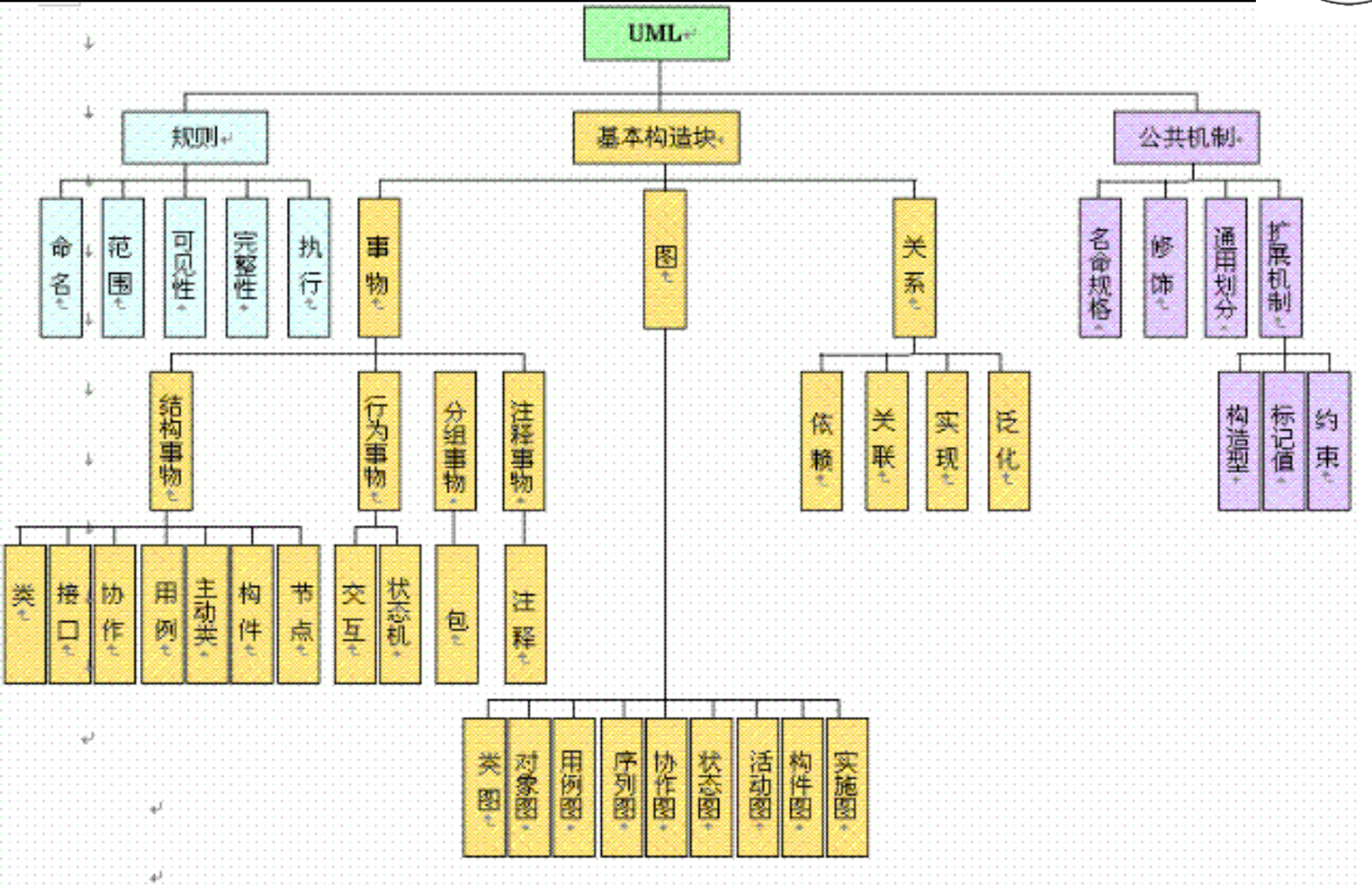
大纲

- 概述
- UML的构造块
- UML的规则
- UML的公共机制
- “Hello，World！”示例

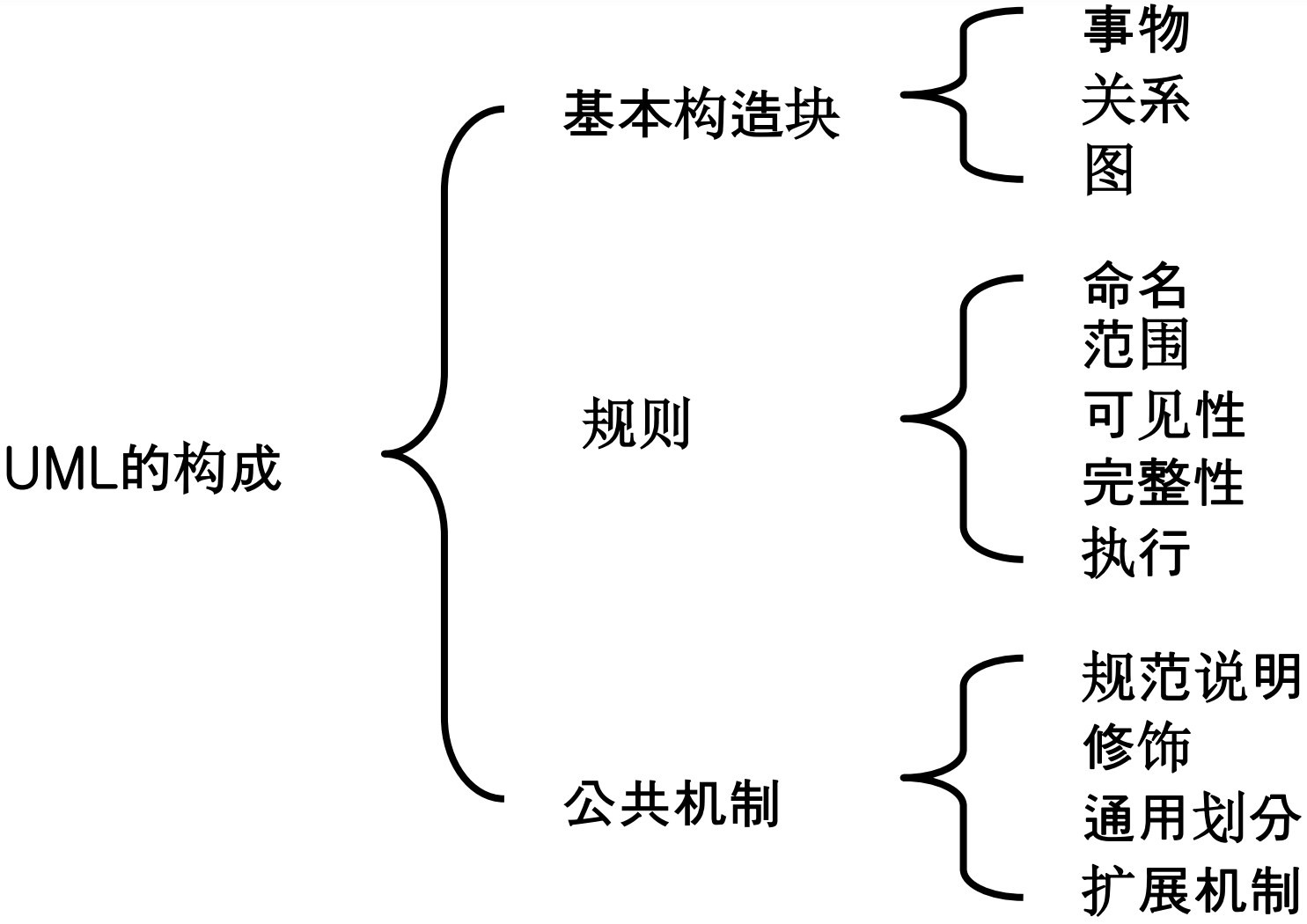
概述

- UML的概念模型——建模3要素
  - 基本构造块：词汇表
  - 规则：支配构造块如何放在一起
  - 公共机制：运用于整个UML

概述


UML
规则
基本构造块
公共机制
命名
范围
可见性
完整性
执行
事物
图
关系
名命规格
修饰
通用划分
扩展机制
结构事物
行为事物
分组事物
注释事物
依赖
关联
实现
泛化
构造型
标记值
约束
类
接口
协作
用例
主动类
构件
节点
交互
状态机
包
注释
类图
对象图
用例图
序列图
协作图
状态图
活动图
构件图
实施图

UML的构成



UML的构造块

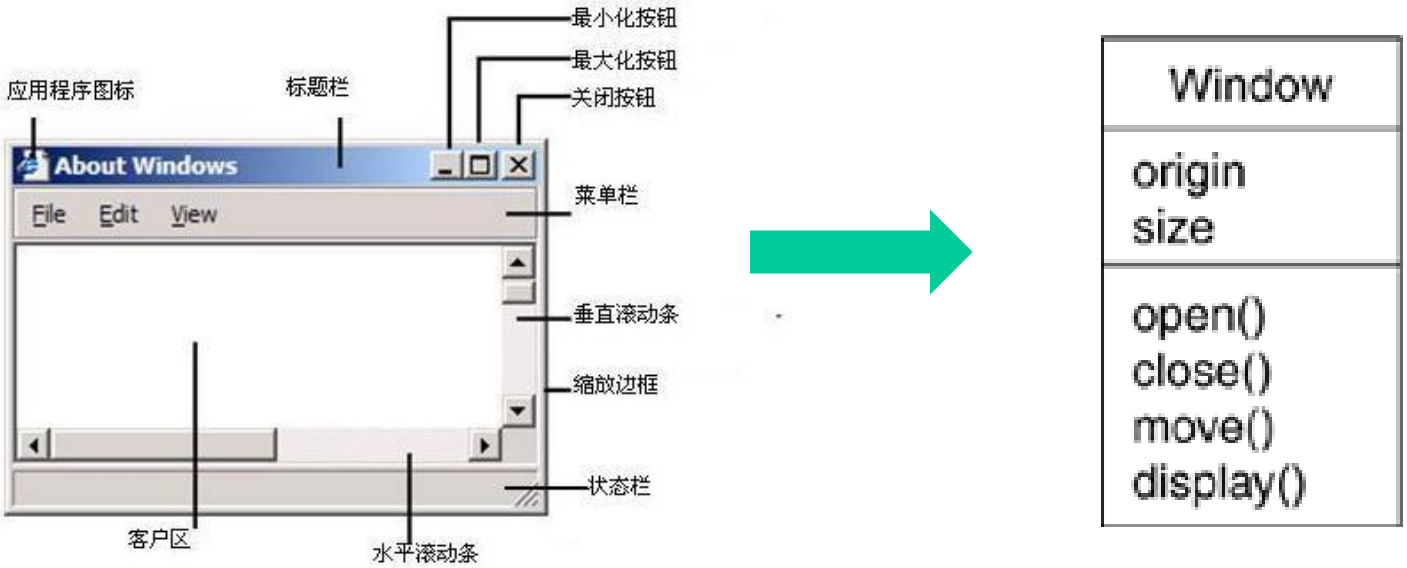
- 构造块 (Building blocks)
  - 事物 (Things)
    - 基本图示符号，表示一些面向对象的基本概念
    - 对模型中最具代表性成分的抽象
  - 关系 (Relationships)
    - 表示基本图示符号之间的关系
    - 建模元素之间的语义联系
  - 图 (Diagrams)
    - 特定的视角对系统所作的抽象描述
    - 聚集了相关的事物

UML的构造块

- UML中的事物
  - 结构事物 (Structural things)
    - 静态部分，描述概念或物理元素
  - 行为事物 (Behavioral things)
    - 动态部分，描述了跨越时间和空间的行为
  - 分组事物 (Grouping things)
    - 组织部分
  - 注释事物 (Annotational things)
    - 解释部分

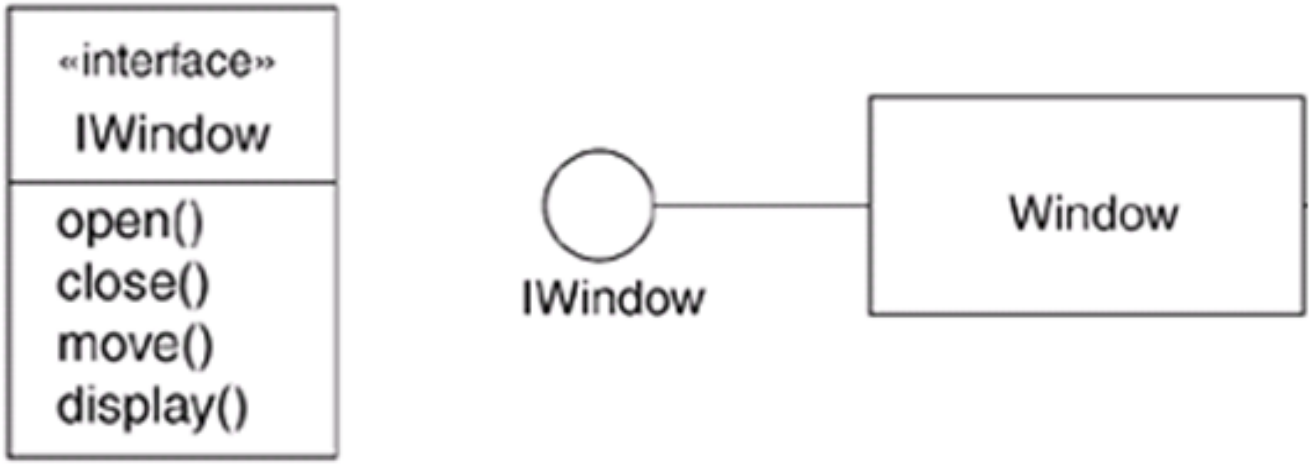
UML的构造块

- 结构事物：1）类（Class）
  - 对一组具有相同属性、方法、关系和语义的对象的描述
  - 实现了一个或多个接口



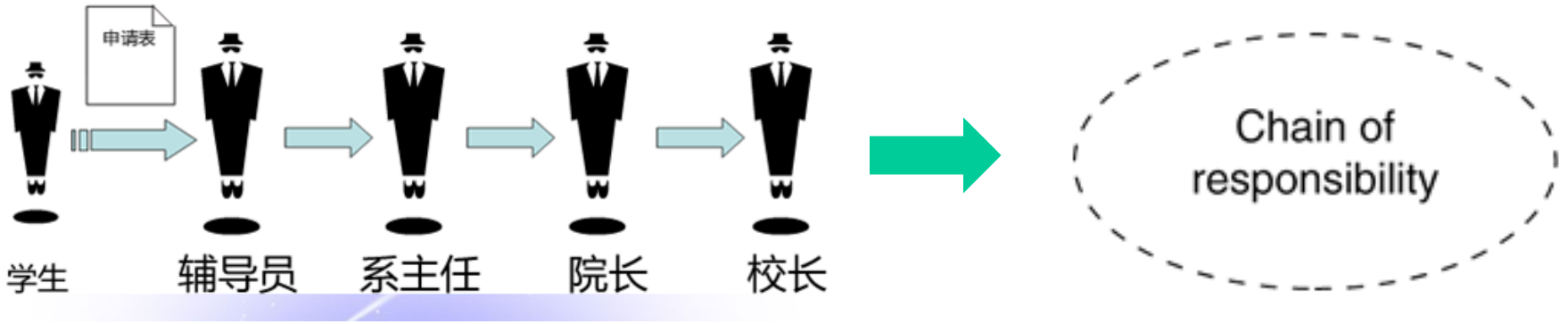
UML的构造块

- 结构事物：2）接口（Interface）
  - 描述了一个类或构件的一个服务的操作集
  - 定义了一组操作的规范，而非具体实现



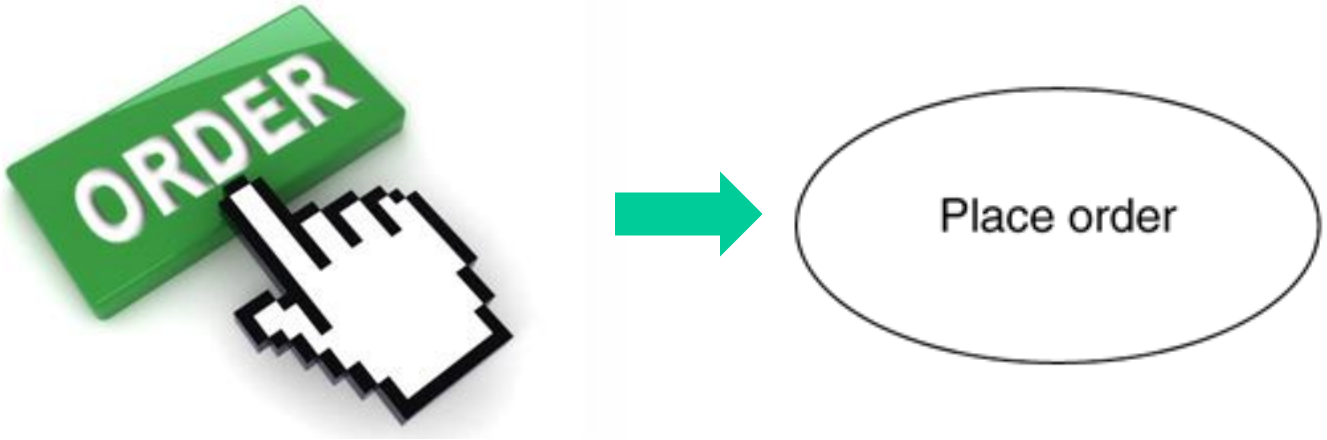
UML的构造块

- 结构事物：3）协作（Collaboration）
  - 定义了一个交互
  - 是一组类、接口和其它元素的群体
  - 协作行为大于所有元素各自行为的总和
  - 只能引用其它地方定义的结构事物



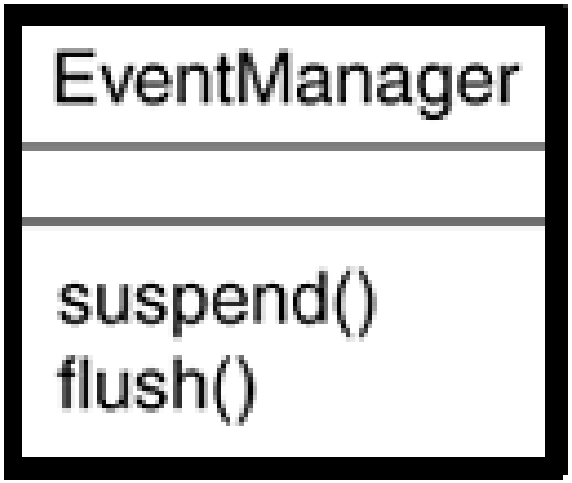
UML的构造块

- 结构事物：4）用例（Use case）
  - 对一组动作序列的描述，系统执行这些动作将产生一个对特定的参与者（Actor）有价值且可观察的结果
  - 对模型中的行为事物结构化，通过协作实现



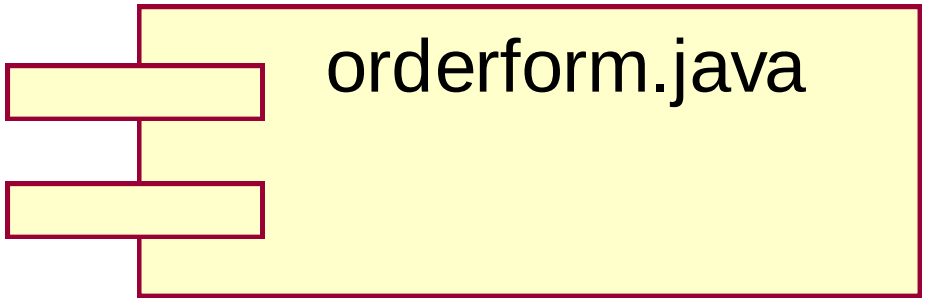
UML的构造块

- 结构事物：5）主动类（Active class）
  - 其对象至少拥有一个进程或线程
  - 能启动控制活动



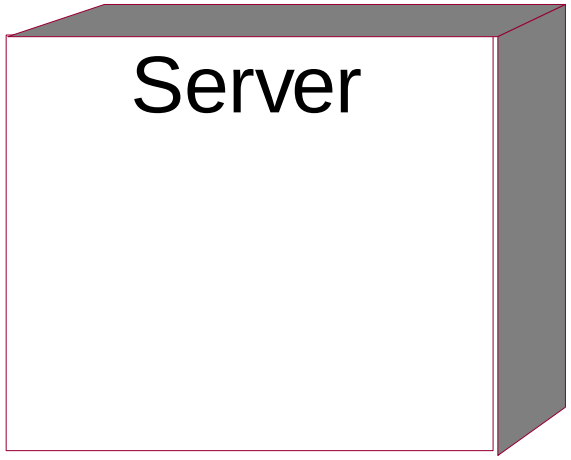
UML的构造块

- 结构事物：6）组件（Component）
  - 系统中物理的、可替代的部件
  - 一个描述了一些逻辑元素（如类、接口和协作）的物理包
  - 遵循且提供一组接口的实现



UML的构造块

- 结构事物：7）节点 (Node)
  - 在运行时存在的物理元素
  - 代表一种可计算的资源
  - 通常至少有一些记忆能力和处理能力



UML的构造块

- 结构事物
  - 类
  - 接口
  - 协作
  - 用例
  - 主动类
  - 组件
  - 节点

概念逻辑事物

物理事物

UML的构造块

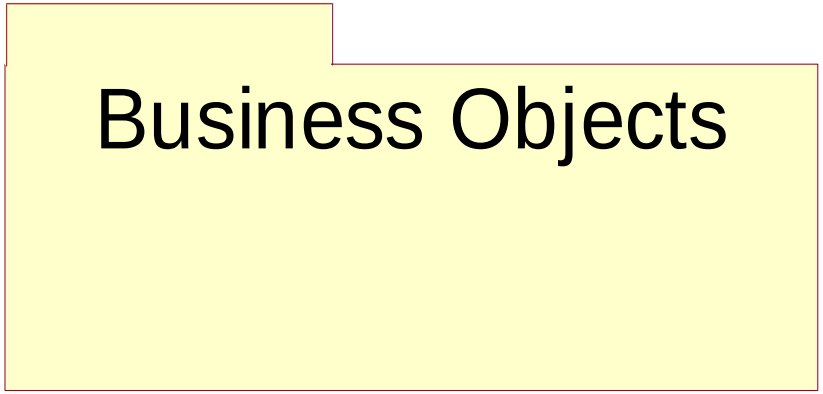
- 行为事物：1）交互（Interaction）
  - 由一组对象及之间交换的消息组成
  - 一个对象群体的行为或单个操作的行为可用一个交互来描述
  - 涉及一些其他元素，包括消息、动作序列（由一个消息所引起的行为）、链（对象间的连接）

UML的构造块

- 行为事物：2）状态机（Statement machine）
  - 描述了一个对象在其生命周期内响应事件所经历的状态序列
  - 可以描述单个类或一组类之间协作的行为
  - 涉及到的元素：状态转换、事件、活动

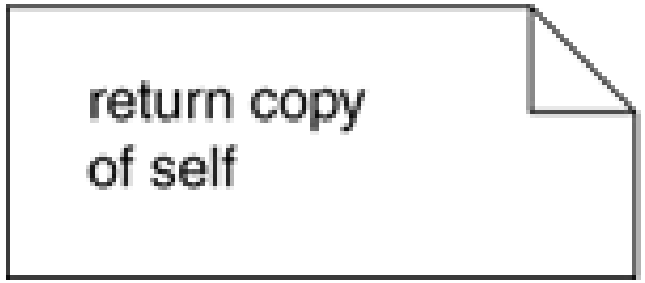
UML的构造块

- 分组事物：包（Package）
  - 把元素组织成组的机制
  - 结构、行为及其他分组事物都可放入包中
  - 一个包形成了一个命名空间



UML的构造块

- 注释事物：注解 (Note)
  - 描述、说明和标注模型的任何元素
  - 依附于一个或一组元素之上，对它进行约束或解释的简单符号

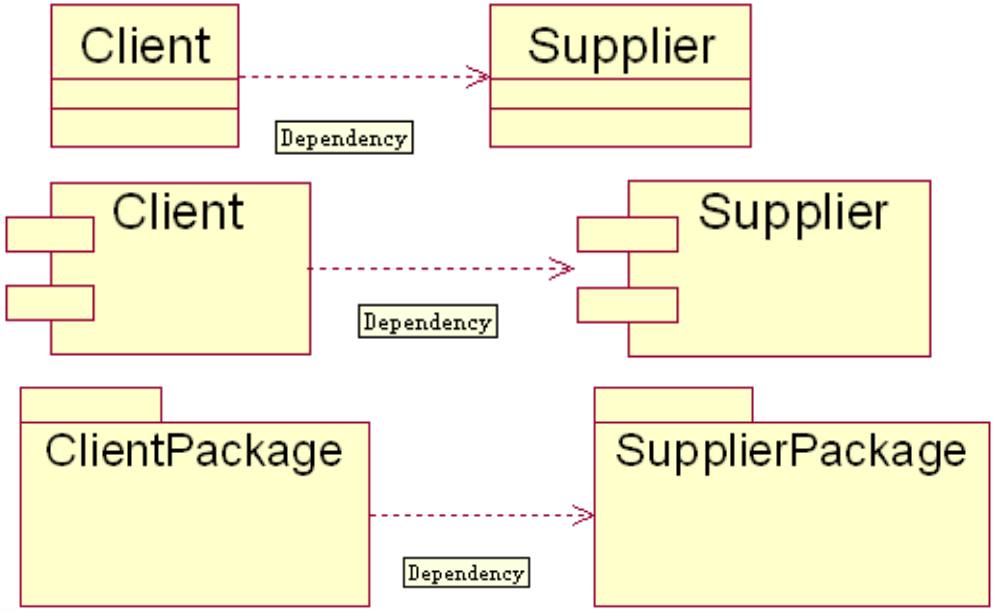


UML的构造块

- UML中的关系
  - 依赖 (Dependency)
  - 关联 (Association)
  - 泛化 (Generalization)
  - 实现 (Realization)

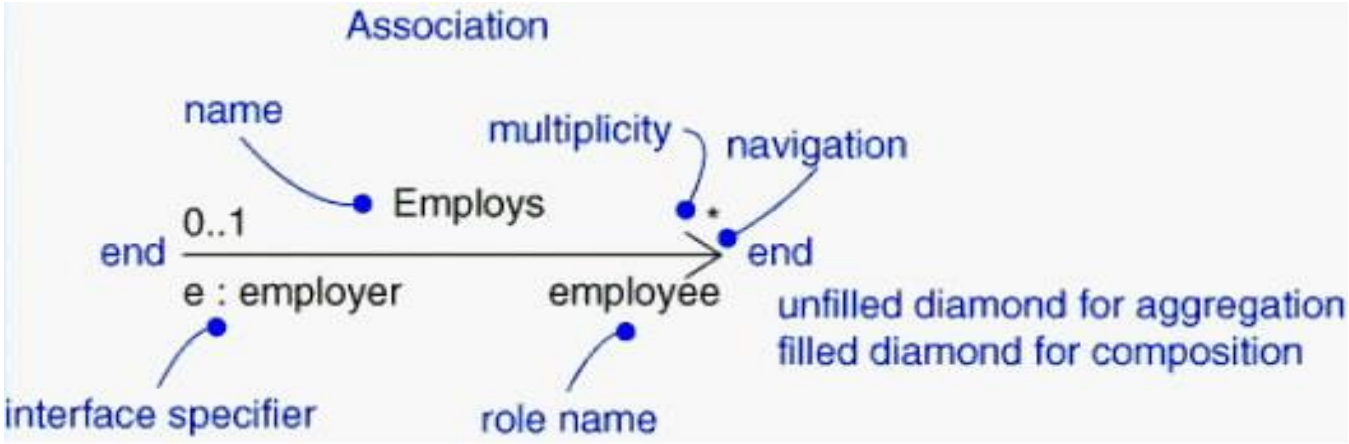
UML的构造块

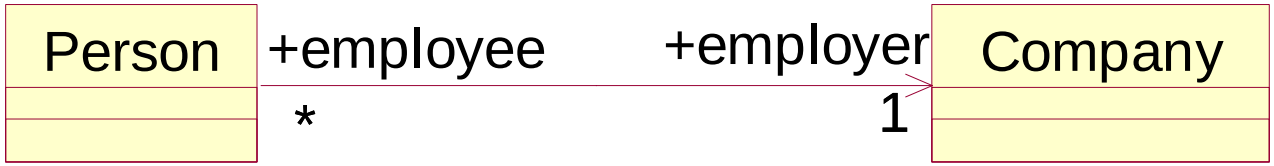
- 依赖 (Dependency)
  - 两个事物间的语义关系
  - 其中一个（独立事物）发生变化会影响另一个（依赖事物）的语义
  - 通常发生在
    - 对象/类和类之间
    - 包和包之间
    - 组件和组件之间



UML的构造块

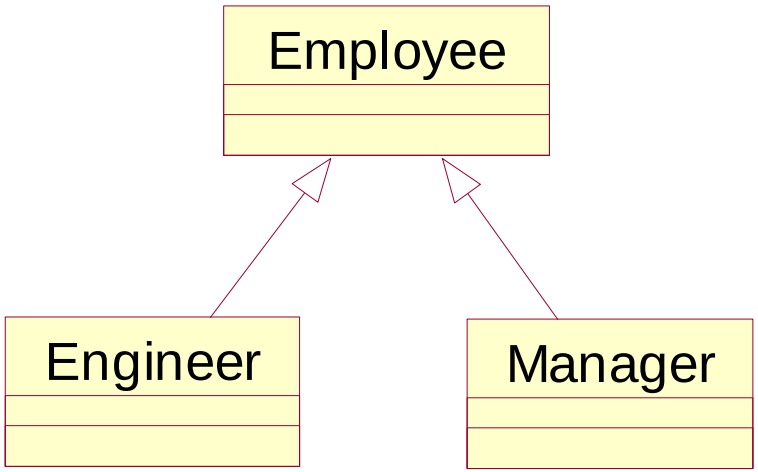
- 关联（Association）
  - 两个或多个类之间的结构性关系
  - 它描述了一组链，链是对象之间的连接





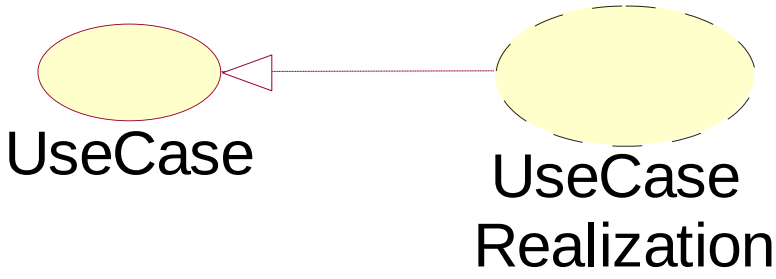
UML的构造块

- 泛化 (Generalization)
  - 一种特殊/一般关系
  - 特殊元素（子元素）的对象可替代一般元素（父元素）的对象
  - 子元素共享了父元素的结构和行为

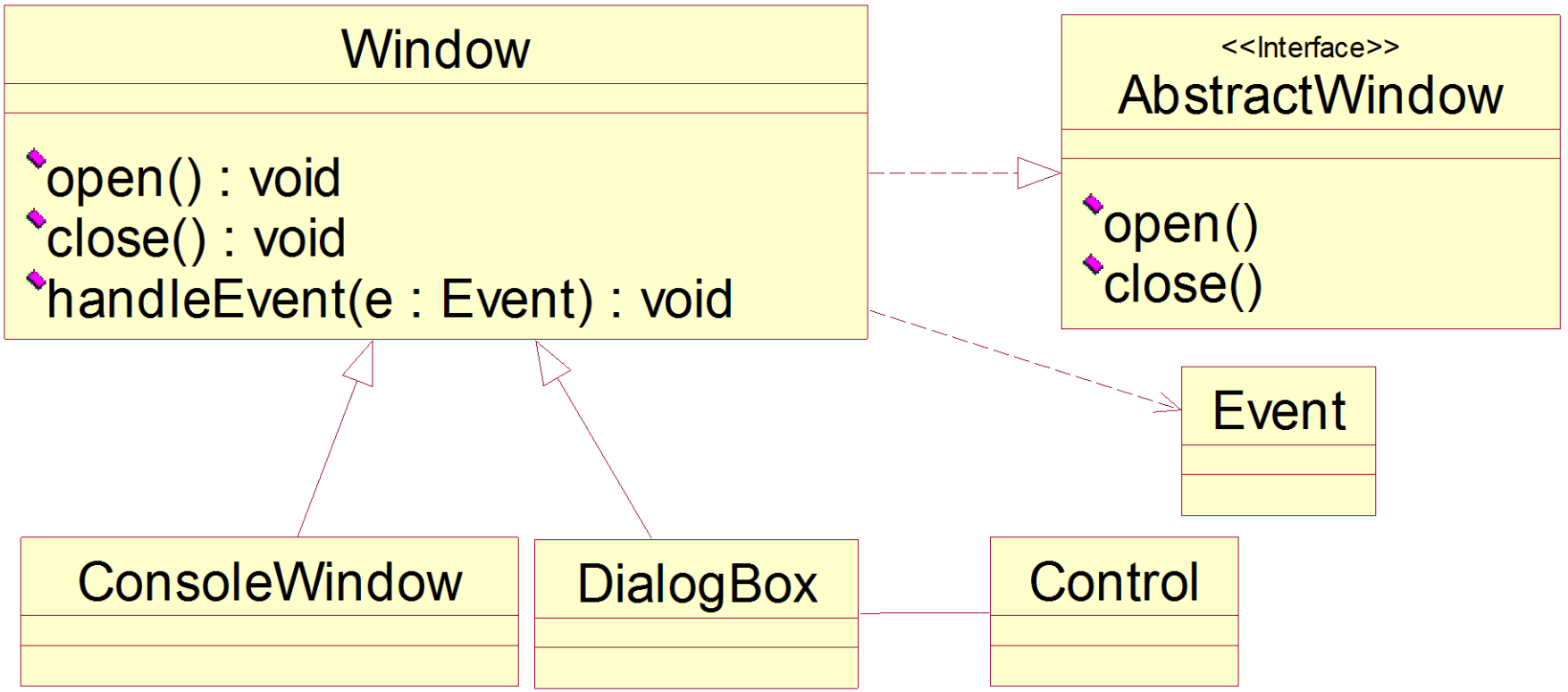


UML的构造块

- 实现 (Realization)
  - 类元之间的语义关系，一个类元描述了另一个类元保证实现的契约
  - 两种情况中存在
    - 在接口和实现它们的类或构件之间
    - 在用况和实现它们的协作之间

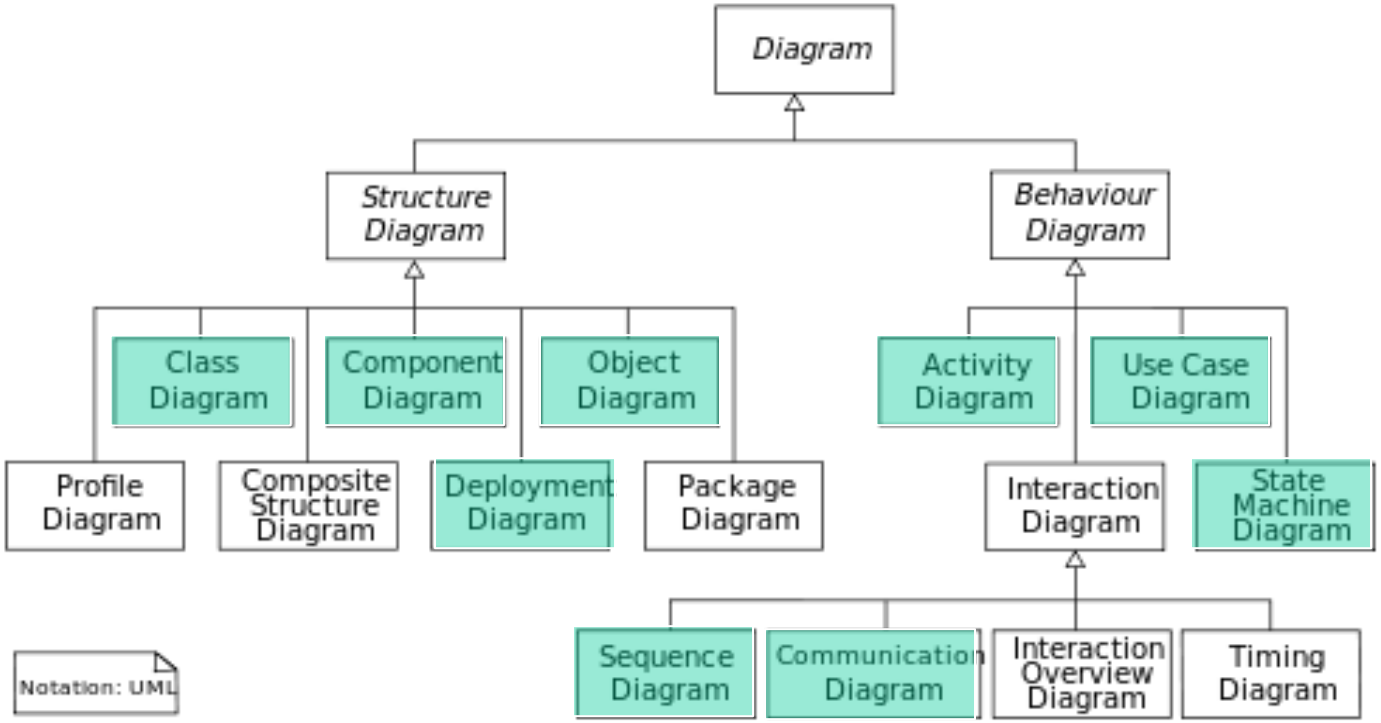


UML的构造块


Window
open() : void
close() : void
handleEvent(e : Event) : void
<<Interface>>
AbstractWindow
open()
close()
Event
ConsoleWindow
DialogBox
Control

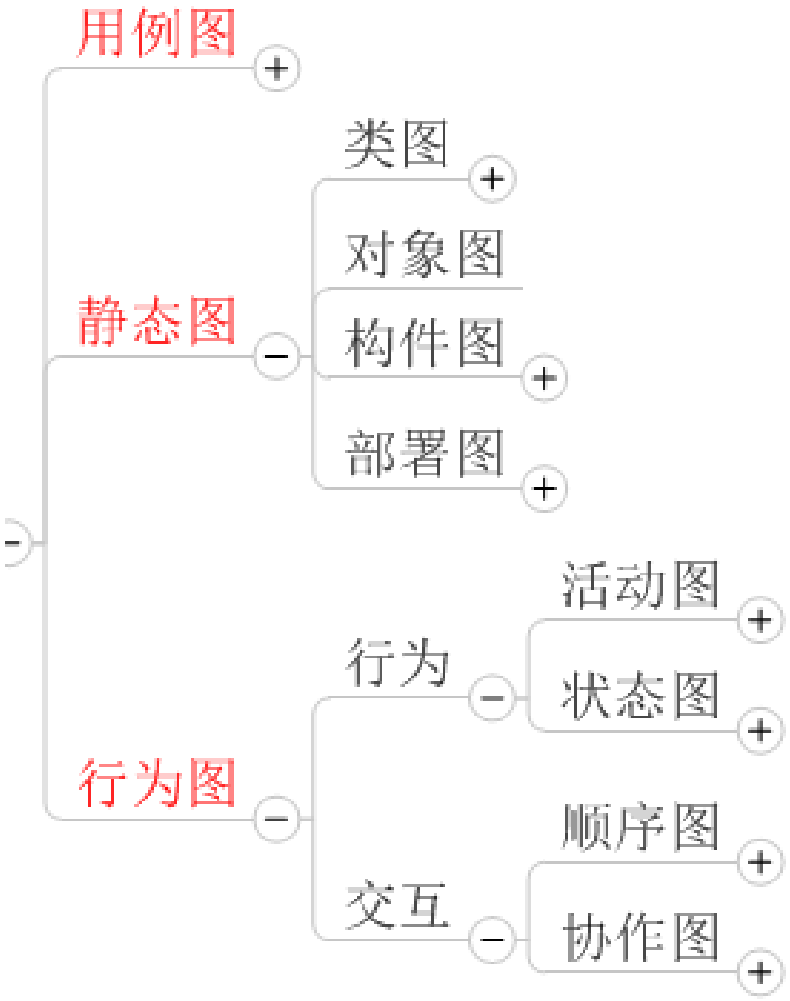
UML的构造块

- UML中的图
  - 静态结构：对系统相对稳定的骨架的表示
  - 动态行为：对系统变化部分的表示



UML的构造块

- UML中的图



UML的构造块

- 对需求建模
  - 用例图：从用户角度描述系统的行为
- 对结构建模
  - 类图：显示一组类、接口、协作及它们之间的关系
  - 对象图：显示一组对象及它们之间的关系
  - 构件图：显示一组构件之间的组织和依赖
  - 部署图：显示对运行时处理节点以及其中的构件的配置

逻辑角度

物理角度

UML的构造块

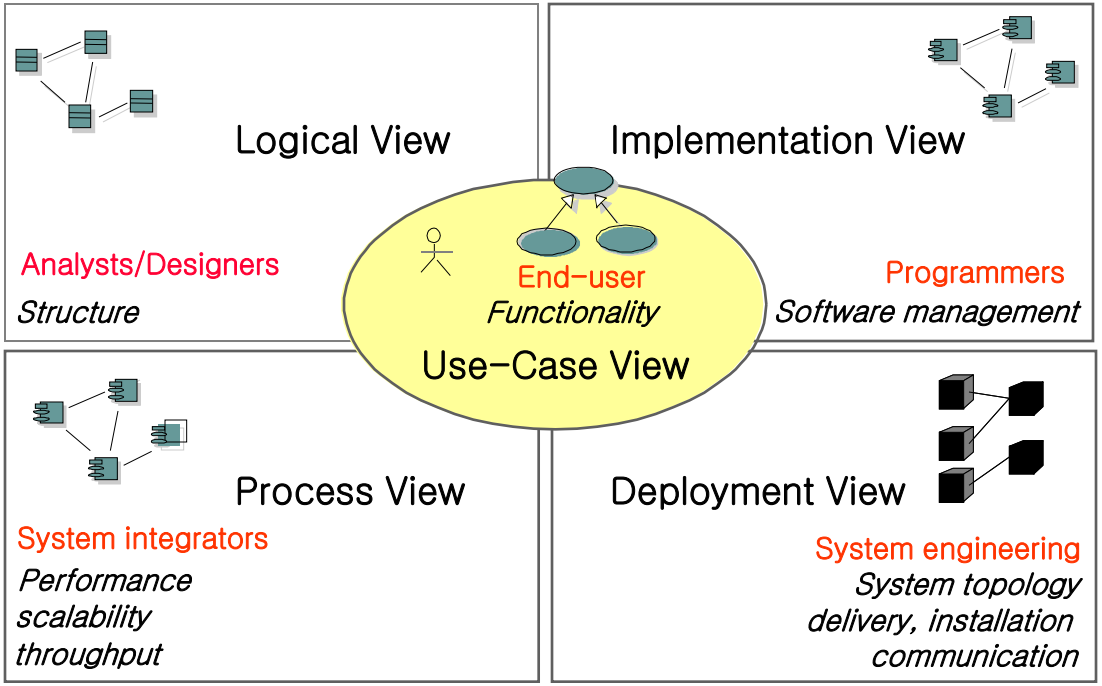
- 对行为建模
  - 顺序图：注重于消息的时间次序
  - 协作图：注重于收发消息的对象的结构组织
  - 状态图：注重于由事件驱动的对象状态变化序列
  - 活动图：注重于从活动到活动的控制流程

交互的投影

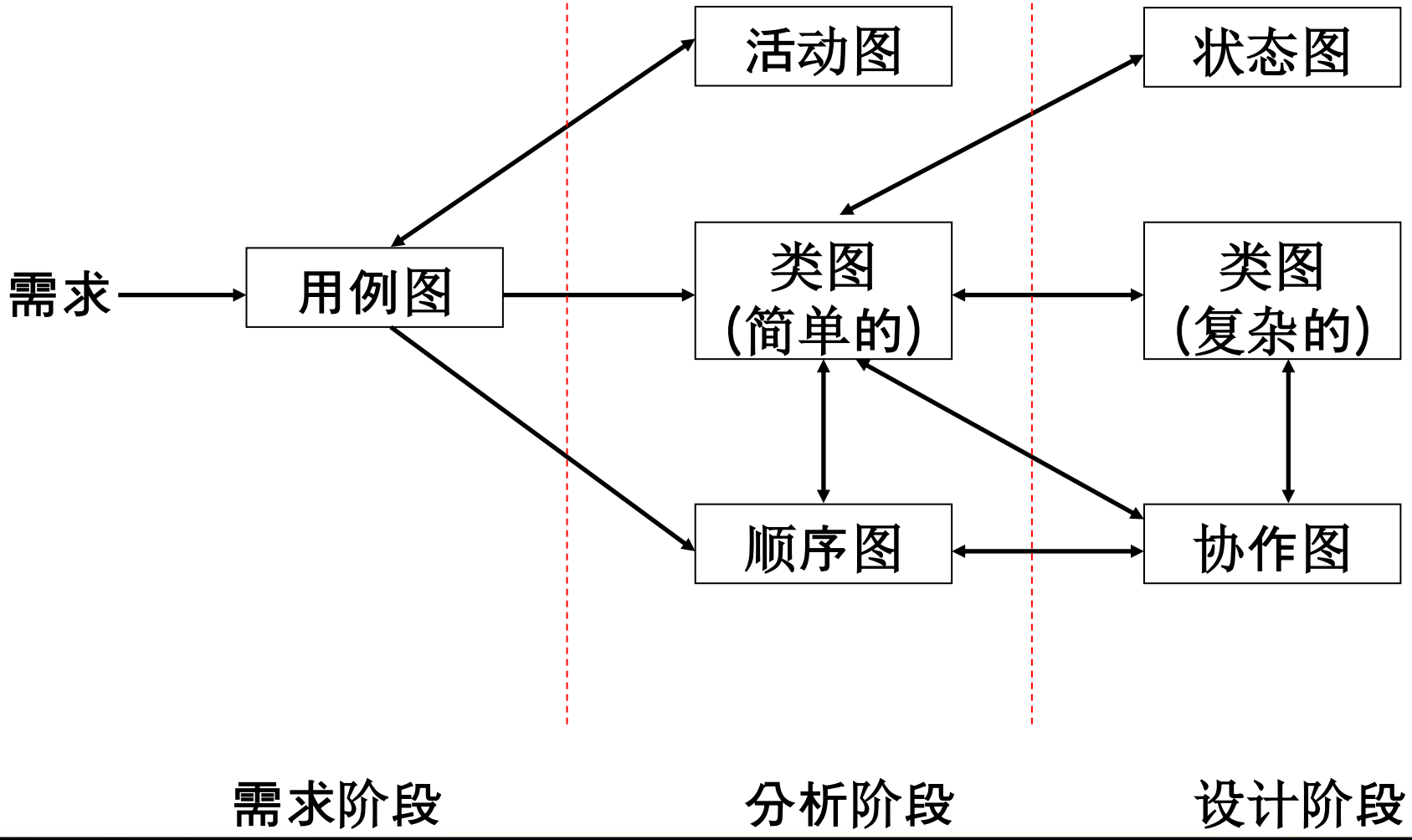
状态机的投影

UML的构造块

- UML的“4+1视图”
  - 视图：系统模型组织和结构在特定方面的投影
  - 用5个互连的视图描述软件系统的体系结构



UML的构造块



UML的构造块

- 用例视图 (Use case view)
  - 由专门描述可被最终用户、分析人员和测试人员看到的系统行为的用例组成
- 逻辑视图 (Logical view)
  - 展示对象和类如何组成系统、实现系统所需行为
- 进程视图 (Process view)
  - 建模系统中作为活动类的可执行线程和进程
- 实现视图 (Implementation view)
  - 建模组成系统的物理代码的文件和组件
- 部署视图 (Deployment view)
  - 描述对组成物理系统的部件的分布、交付和安装

UML的构造块

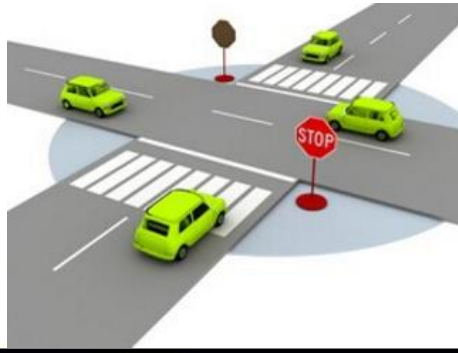
视图	包含的图
use case view	use case diagrams, interaction diagrams
logical view	class diagrams, object diagrams, state diagrams, interaction diagrams, activity diagrams
process view	class diagrams, object diagrams, component diagram
implementation view	component diagram
deployment view	deployment diagram

UML的规则

- 规则 (Rules)
  - 描述了一个结构良好的模型看起来应该像什么样子
  - 结构良好的模型
    - 在语义上前后一致
    - 与所有的相关模型协调一致

UML的规则

- 规则 (Rules)
  - UML的语义规则
    - 命名：为事物、关系和图起名
    - 范围：给一个名称以特定含义的语境
    - 可见性：怎样让其他人使用或看见名称
    - 完整性：事物如何正确、一致地相互联系
    - 执行：运行或模拟动态模型的含义是什么

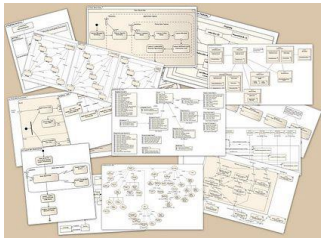


UML的公共机制

- 公共机制 (Common mechanisms)
  - 描述为达到建模目的四种策略
  - 目的：与具有公共特征的模式取得一致性
  - 四种机制
    - 规格说明 (Specifications)
    - 修饰 (Adornments)
    - 通用划分 (Common divisions)
    - 扩展机制 (Extensibility mechanisms)

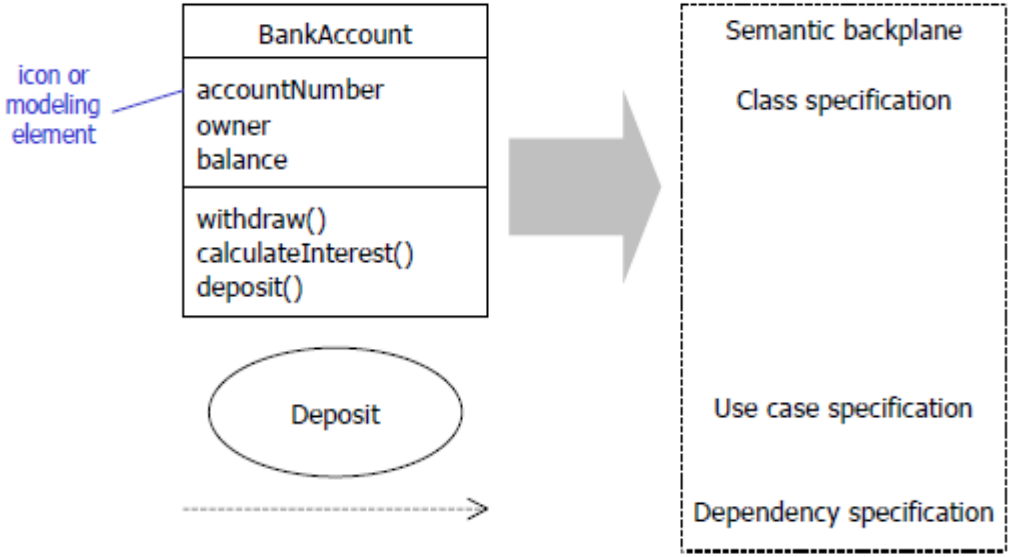
UML的公共机制

- 规格说明 (Specifications)
  - UML模型元素的两种维度
    - 图形维度
    - 文本维度



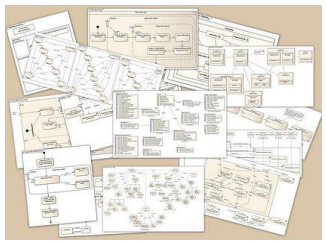
+

  - 规格说明：构造块的语法及语义的文本描述



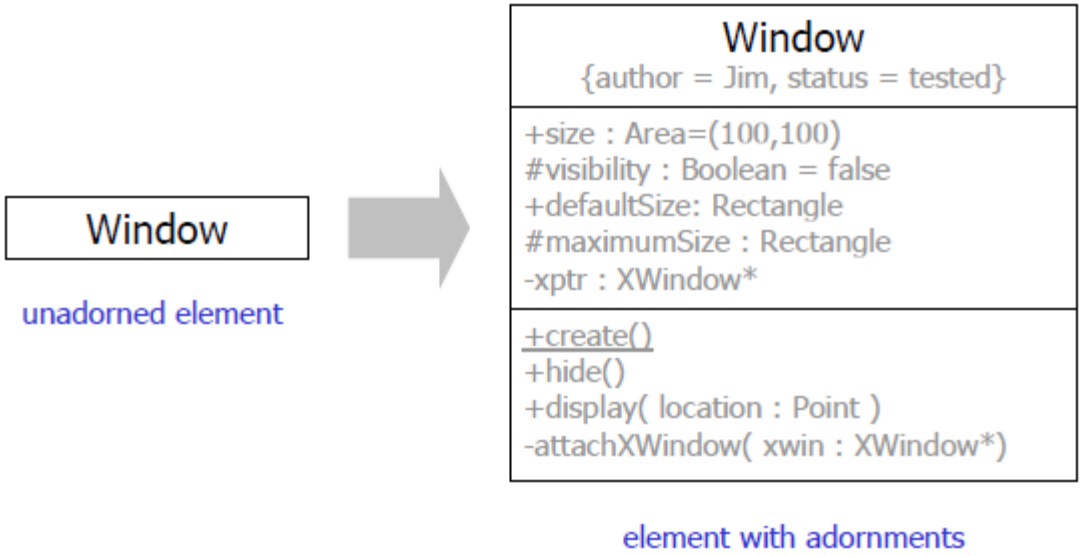
UML的公共机制

- 规格说明 (Specifications)
  - 语义底板 (Semantic backplane)
    - 承载模型，赋予其意义
    - 包含一个系统中相互联系并保持一致性的所有组成部分
    - 一个UML图是对底板的一个可视化投影
  - 模型真正的“肉”



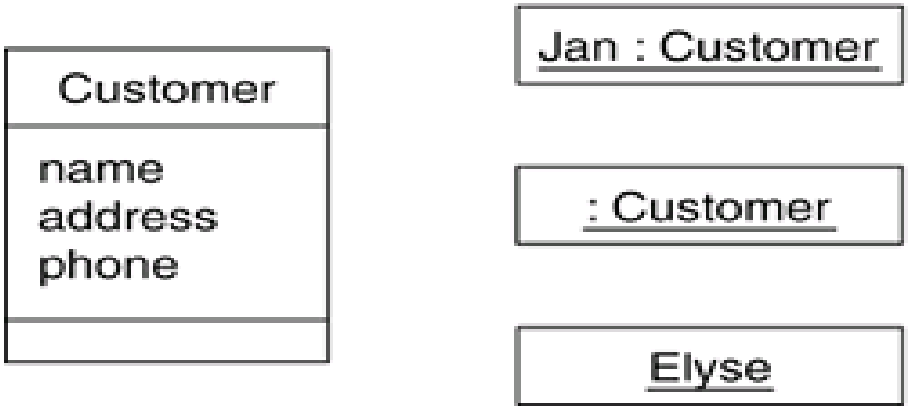

UML的公共机制

- 修饰 (Adornments)
  - 每个元素都有一个基本符号，可以把各种修饰细节加到这个符号上
  - 使得元素的规格说明可见
  - 可根据需要裁减可视化信息的数量



UML的公共机制

- 通用划分 (Common divisions)
  - 类元 (Classifier) /实例
    - 类元是一类事物的抽象概念；实例是具体的事物
    - 类/对象；用例/用例实例；组件/组件实例
    - 实例与类元图标相同，名称具有下划线



UML的公共机制

- 通用划分 (Common divisions)
  - 类元 (Classifier) /实例
    - UML提供了33个类元
    - 常用类元

Classifier	Semantics
Actor	A role played by an outside user of the system to whom the system delivers some value
Class	A description of a set of objects that share the same features
Component	A modular and replaceable part of a system that encapsulates its contents
Interface	A collection of operations that are used to specify a service offered by a class or component
Node	A physical, run-time element that represents a computational resource – an example might be a PC
Signal	An asynchronous message passed between objects
Use case	A description of a sequence of actions that a system performs to yield value to a user

UML的公共机制

- 通用划分 (Common divisions)
  - 接口和实现
    - 接口声明一个契约；实现表示对该契约的具体实施
    - 几乎每个构造块都有像接口/实现这样的二分法
    - 用况/实现它们的协作；操作/实现它们的方法

UML的公共机制

- 扩展机制 (Extensibility mechanisms)
  - UML不是闭合的语言
  - 根据要解决的问题塑造UML
  - 三种扩展机制
    - 构造型 (Stereotype)
    - 标记值 (Tagged value)
    - 约束 (Constraint)

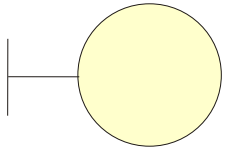
UML的公共机制

- 构造型 (Steroetype)
  - 扩展了UML的词汇
  - 基于已有元素引入新的建模元素
  - 新构造块从现有的派生，又需针对特定问题
  - UML中的预定义构造型
    - 业务用例/业务用例实现
    - 接口
    - 边界类、实体类、控制类
    - ……

UML的公共机制

定义边界类的构造型

<<boundary>>
LoginForm

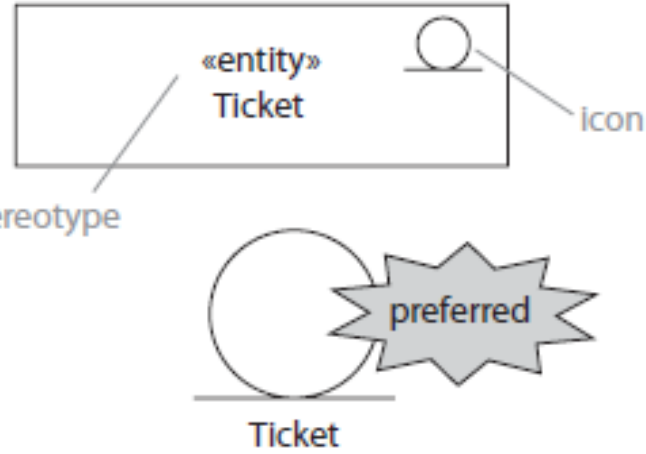


LoginForm

定义接口的构造型

«interface»
IWindow
open()
close()
move()
display()

IWindow

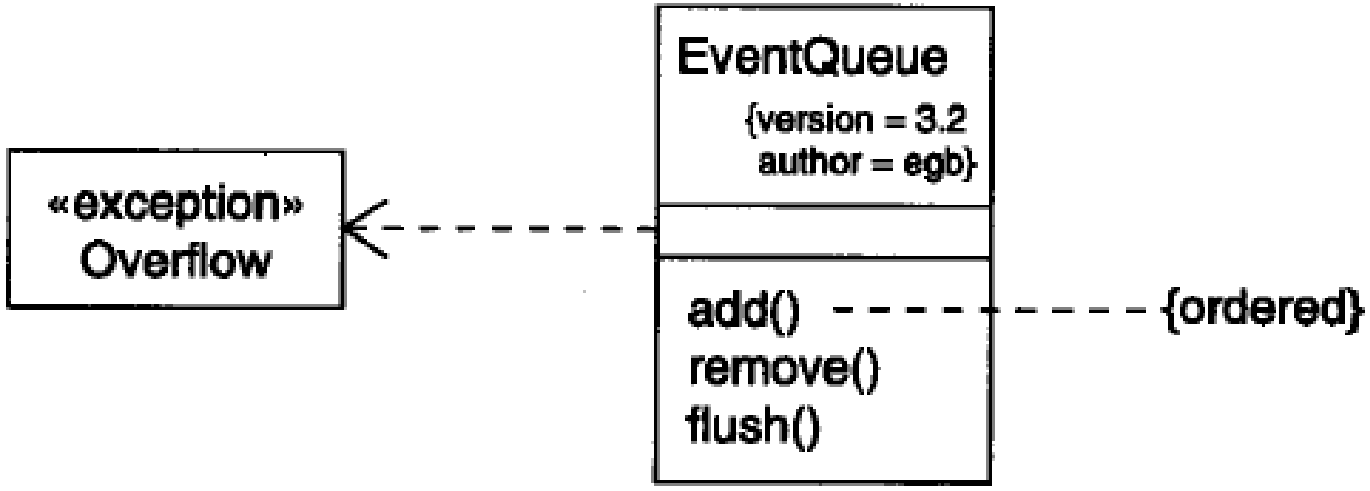


UML的公共机制

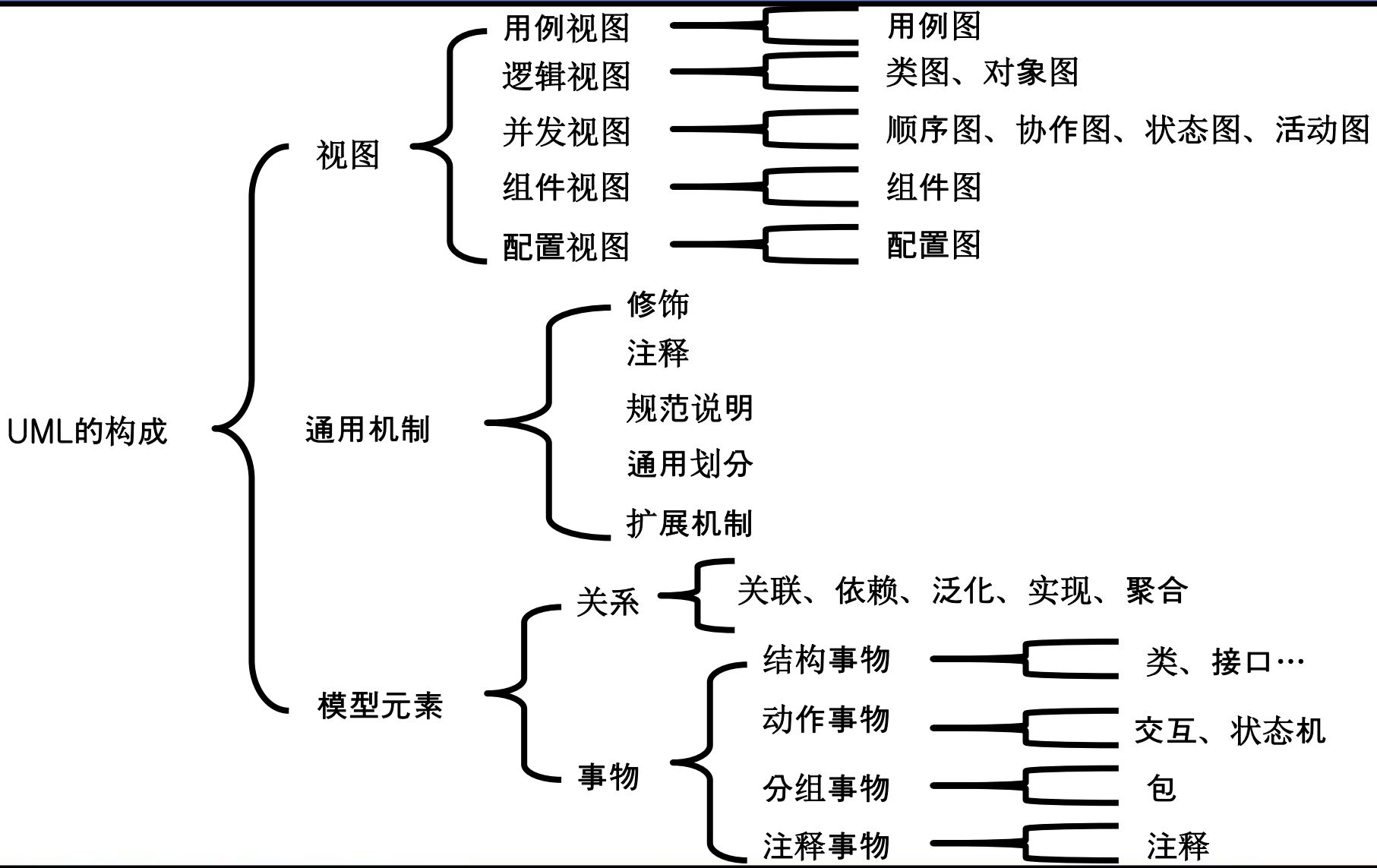
- 标记值 (Tagged value)
  - 为建模元素添加新的属性
  - UML元素属性的扩展
  - 一个标记值是一个“关键字-值”对
  - 应用
    - 作用于模型元素的附加信息
    - 表示由构造型定义的新建模元素的特性

UML的公共机制

- 约束 (Constraints)
  - 扩展UML构造块的语义
  - 在某些方面约束元素的某些特征
  - 对象约束语言OCL



UML的构成


UML的构成
视图
用例视图
用例图
逻辑视图
类图、对象图
并发视图
顺序图、协作图、状态图、活动图
组件视图
组件图
配置视图
配置图
通用机制
修饰
注释
规范说明
通用划分
扩展机制
模型元素
关系
关联、依赖、泛化、实现、聚合
事物
结构事物
类、接口…
动作事物
交互、状态机
分组事物
包
注释事物
注释

“Hello，World！”示例

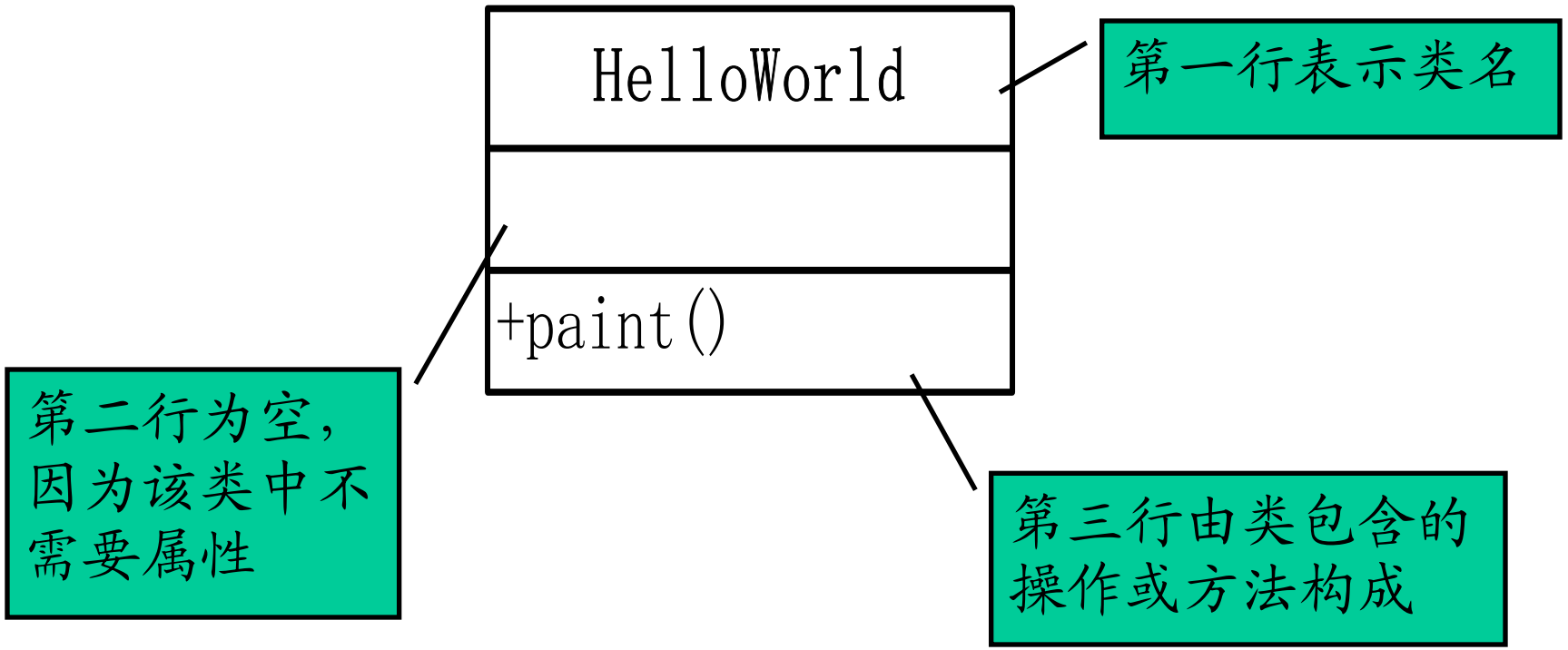
“Hello，World！”示例

显示“Hello world!”的简单 Java Applet程序

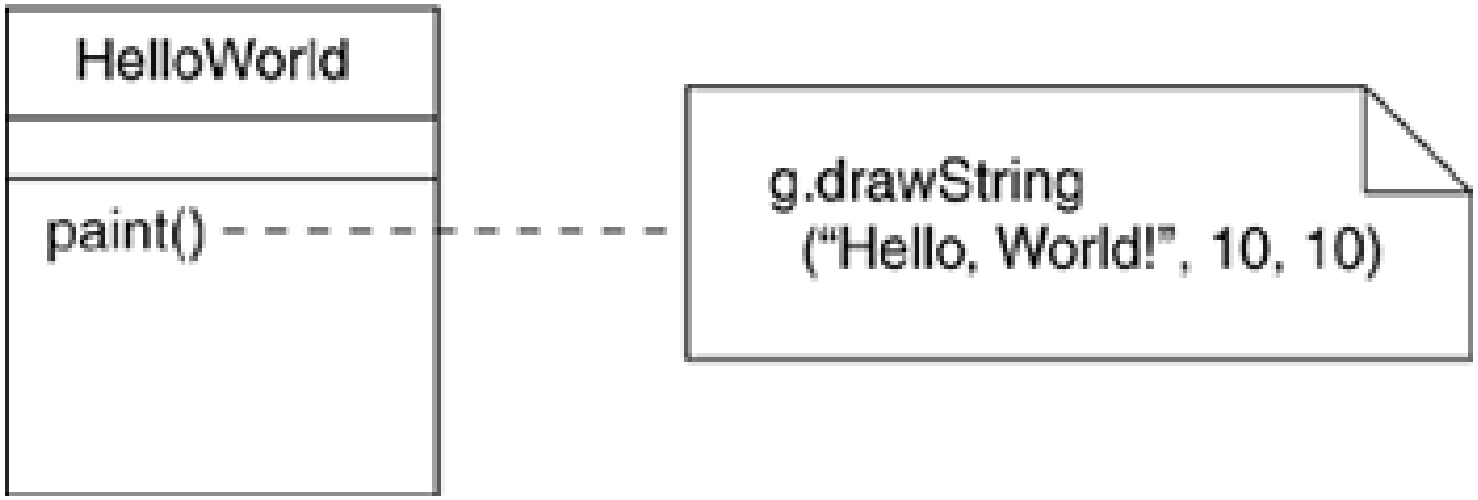
——先来看代码

```
import java.awt.Graphics;
public class HelloWord extends java.applet.Applet{
       public void paint(Graphics g){
              g.drawString(“Hello world!”,10,10);
       }
}
```

“Hello，World！”示例

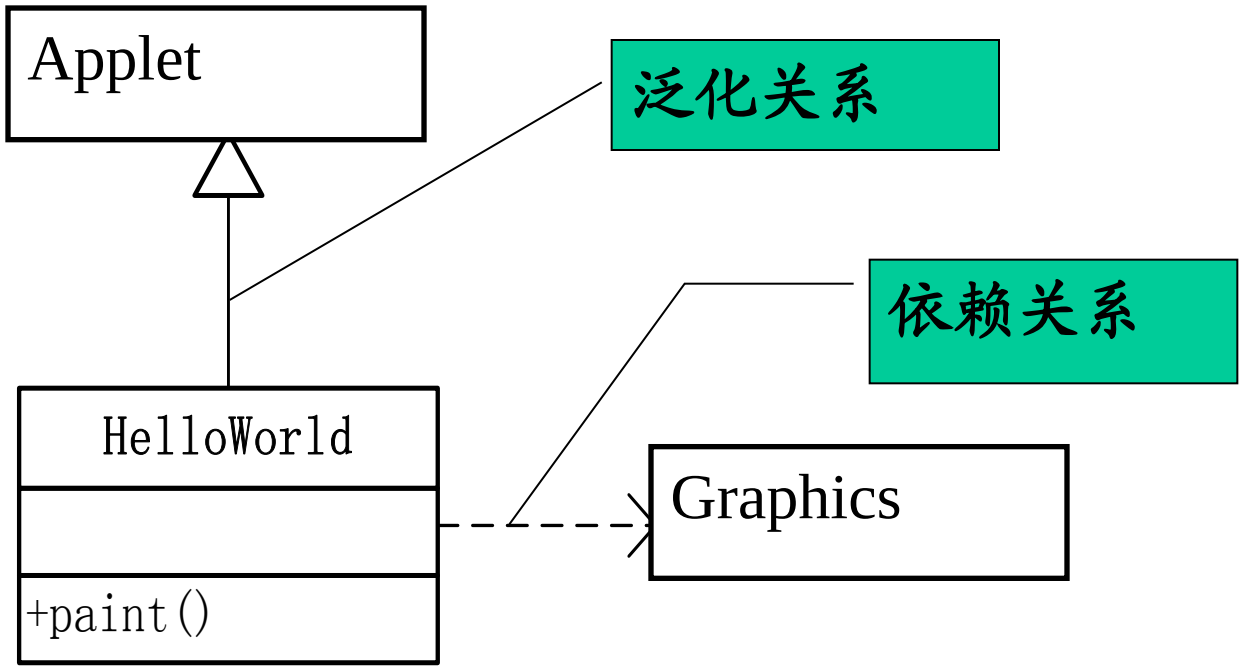


“Hello，World！”示例



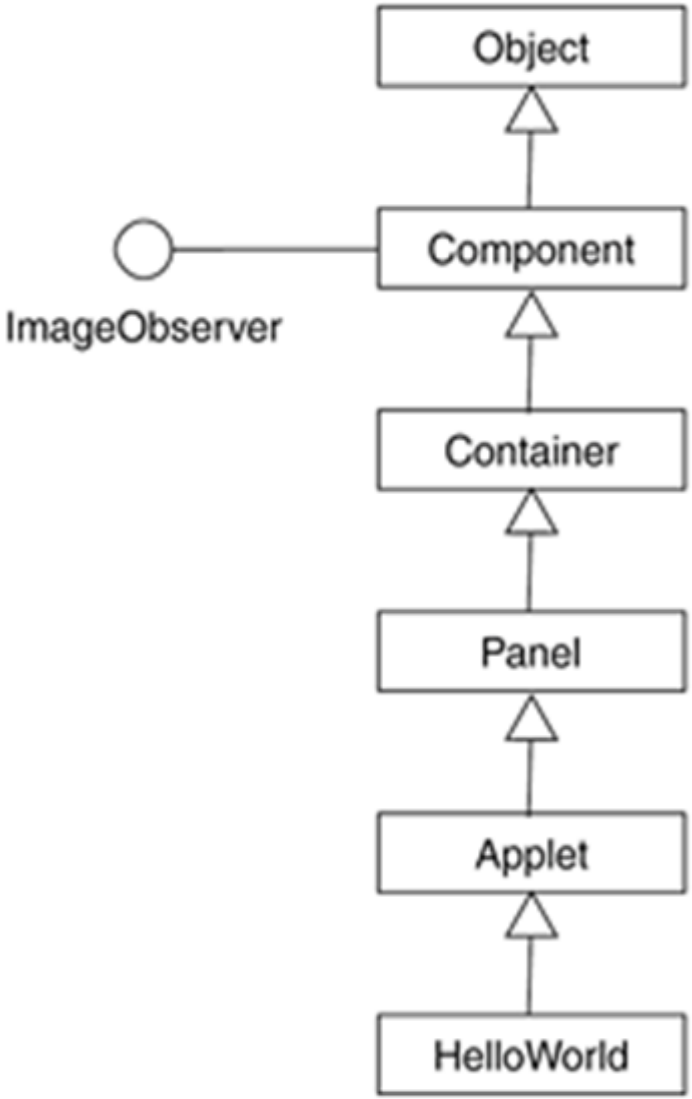
“Hello，World！”示例

类的关系



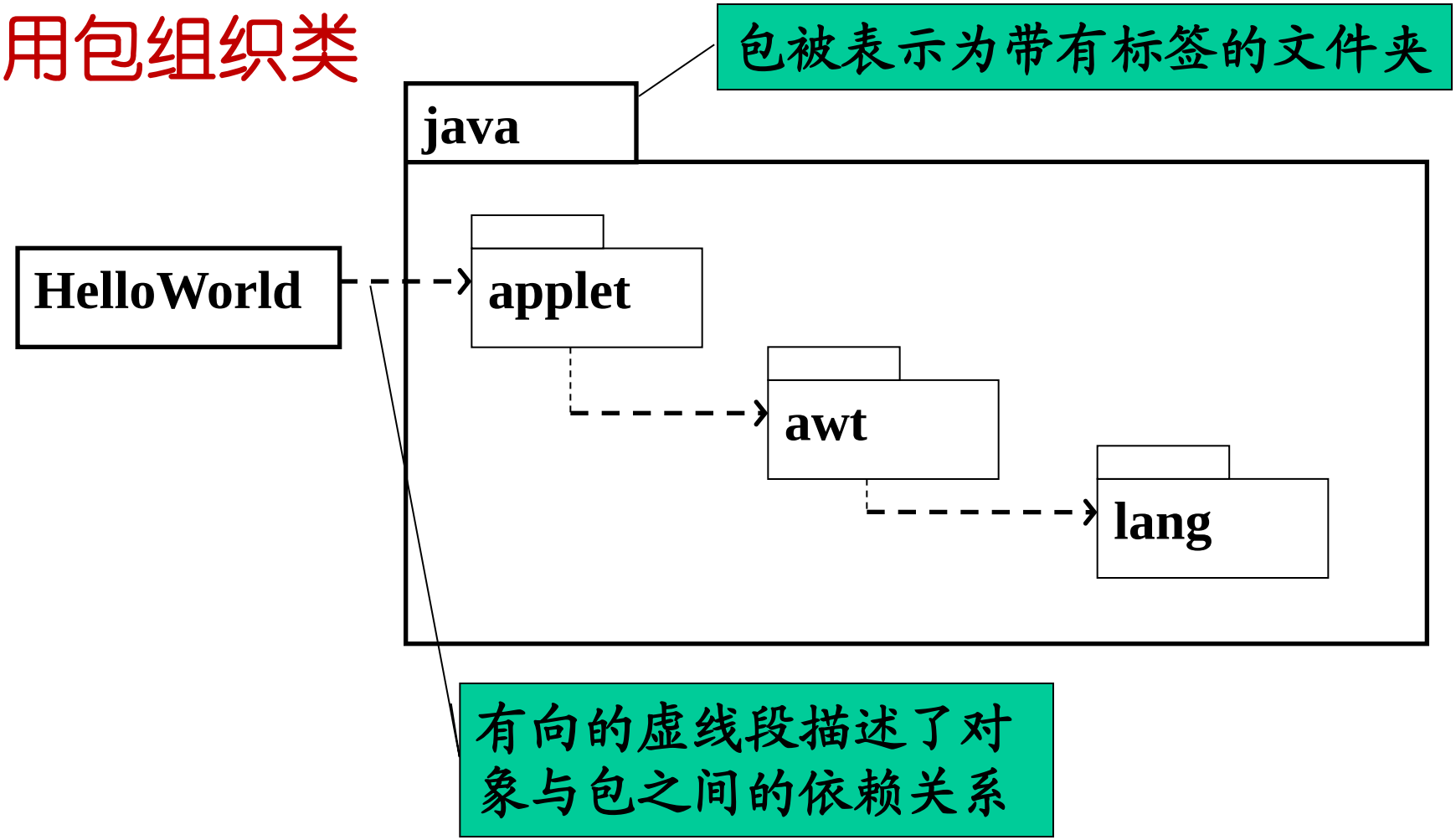
“Hello，World！”示例

继承层次



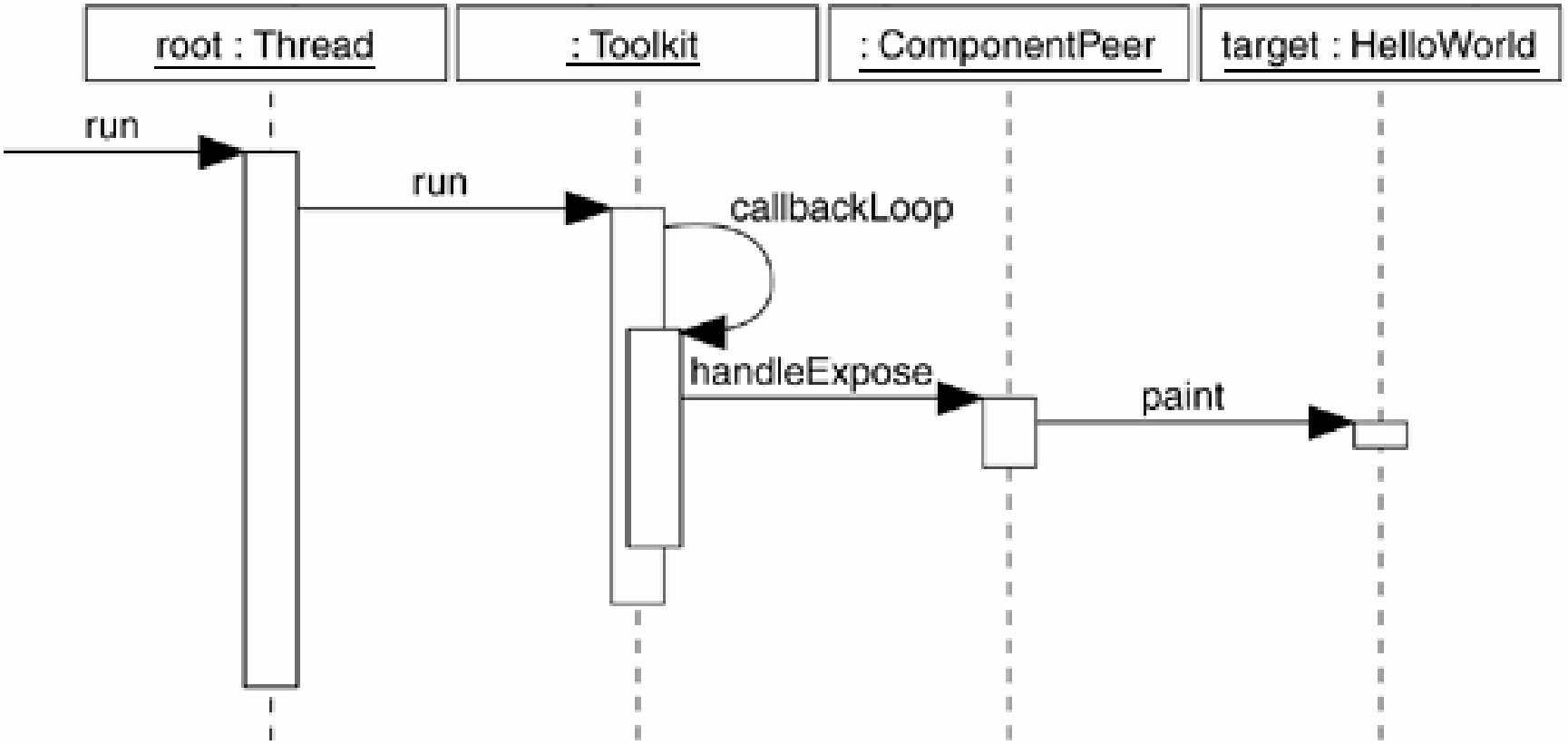
“Hello，World！”示例

用包组织类



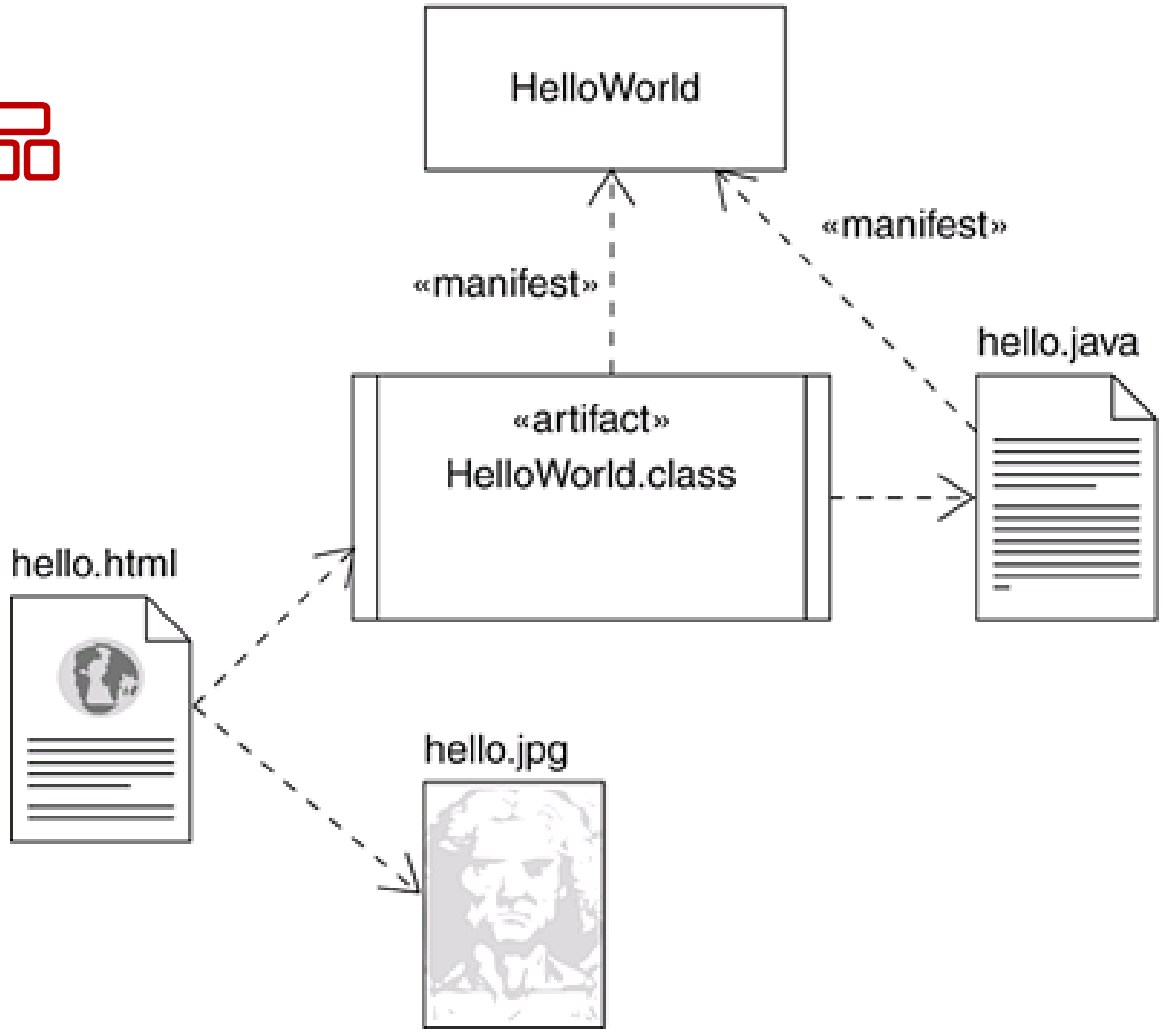
“Hello，World！”示例

交互行为



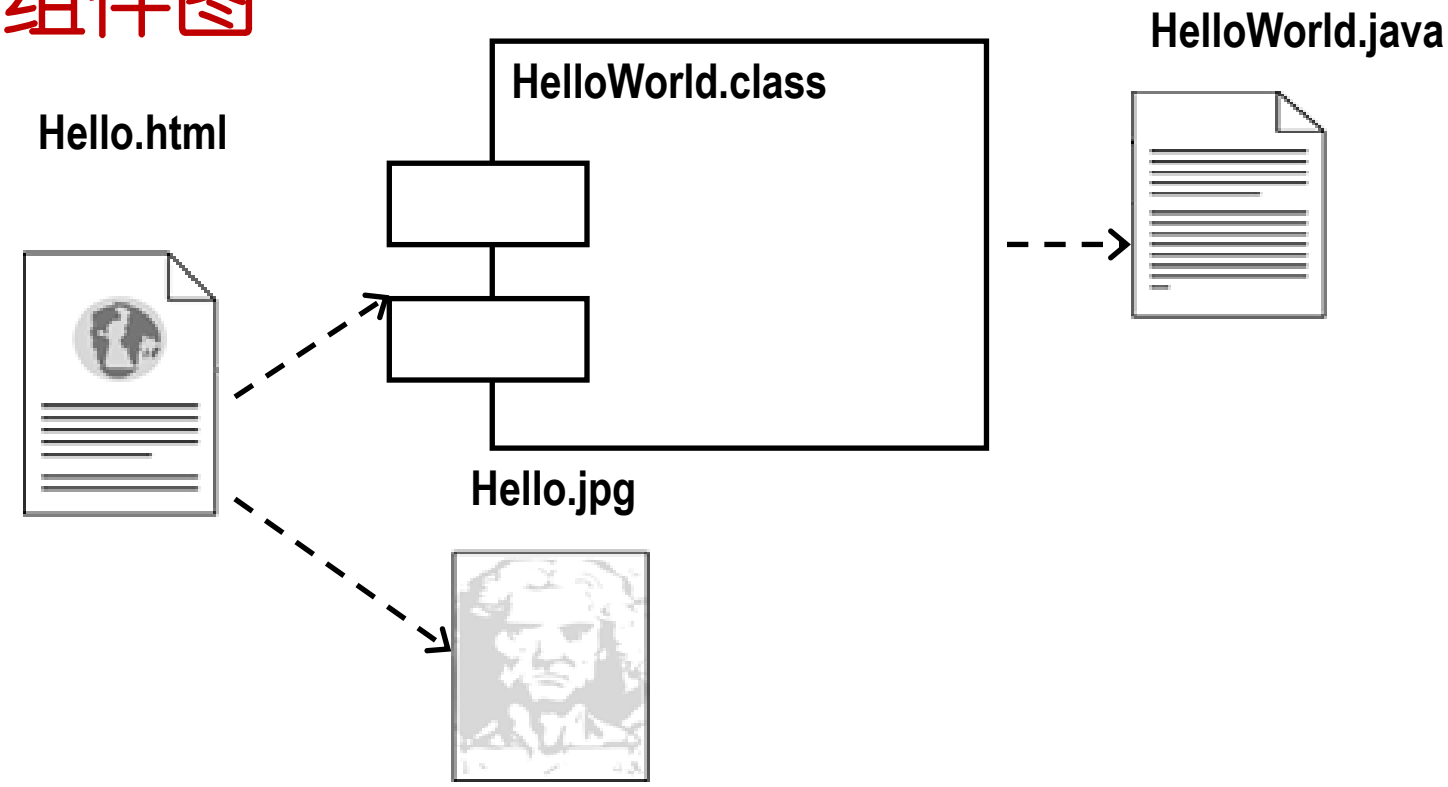
“Hello，World！”示例

制品



“Hello，World！”示例

组件图



小结

- 重点
  - 了解UML的构造块，规则和公共机制
  - 理解规格说明的重要性
  - 掌握UML的4+1视图
  - 掌握UML的构造型