面向对象和UML2

Chapter 3

内容安排

- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

内容安排

- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

素数问题

素数的定义: 除了1与本身之外，不能被其他正整数整除的数，叫作素数，也叫质数

按照习惯规定，1 不算素数，最小的素数是 2，其余的是 3、5、7、11、13、17、19……等等

由定义判断素数

对于数 n ,从i=2,3,4,5…到 n-1 判断 n 能否被 i 整除,如果全部不能整除,则 n 是素数,只要有一个能除尽,则 n 不是素数,为了压缩循环次数,可将判断范围从 2 ~ n-1 改为 2 ~ sqrt(n)

筛选法求素数序列

筛选法：生成 2< i <n 的素数序列，设n=50

筛掉2的倍数：2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 …

筛掉3的倍数：2 3 5 7 9 11 13 15 17 19 21 23 25 27 29 …

筛掉5的倍数：2 3 5 7 11 13 17 19 23 25 29 31 35 37 41 …

筛掉7的倍数：2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 49

留下素数序列：2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

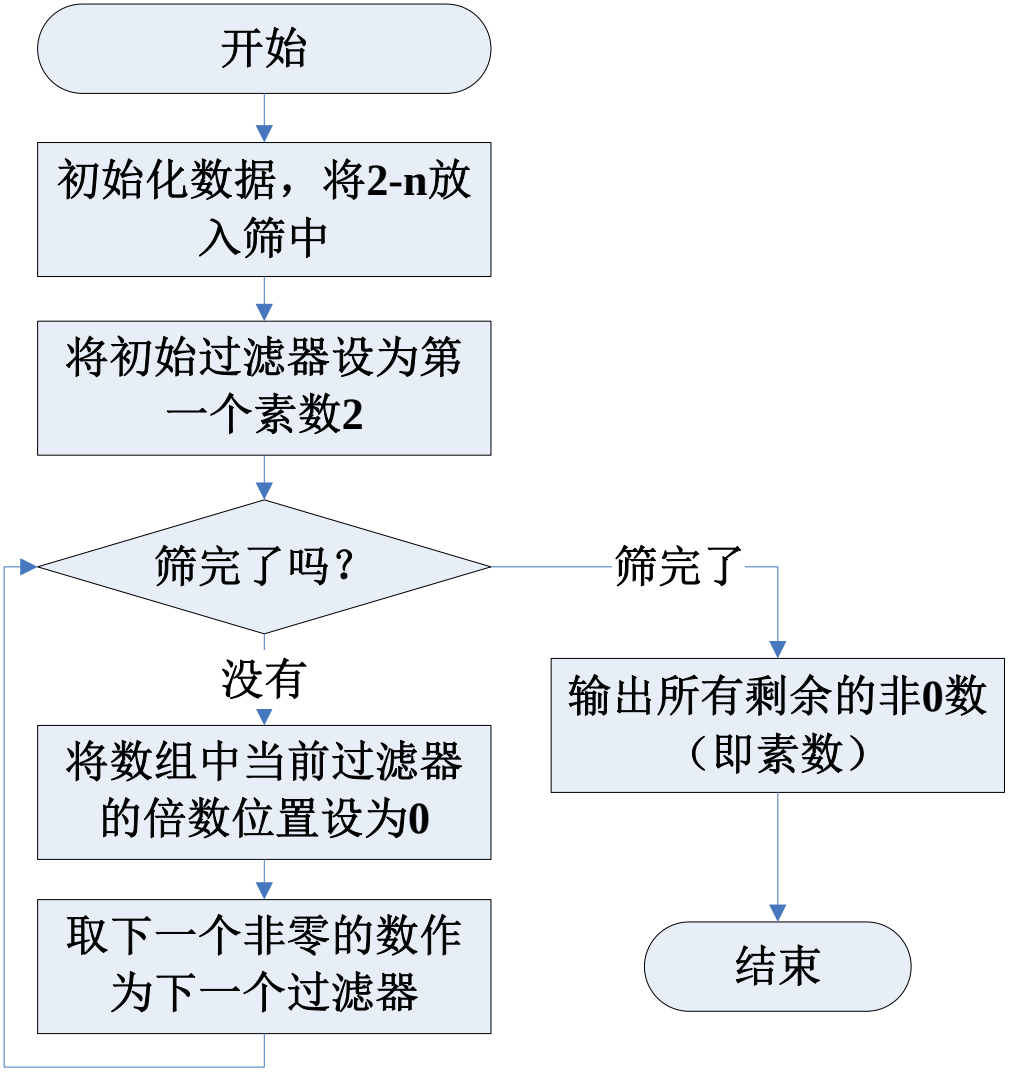
思考?

- 用传统结构化思维解决上述问题
- 用对象思维解决上述问题
- 将解决思路用合适的方式记录下来
- 思考：
  - 结构化思维与对象化思维有什么本质的不同？体现了怎样的思维差异？对象思想有何优势？
  - 如何表达设计思想：代码？图形？

结构化实现

```
//PrimerNumber.c
main(){
  int *sieve,n;
  int iCounter=2, iMax, i;
  printf("Please input max number:");
  scanf("%d", &n);
  sieve=malloc((n-1)*sizeof(int));
  for(i=0;i<n-1;i++) { sieve[i]=i+2; }
  iMax = sqrt(n);
  while (iCounter<=iMax) {
     for (i=2*iCounter-2; i<n-1; i+=iCounter)
        sieve[i] = 0;
     iCounter++;  }
  for(i=0; i<n-1; i++)
    if (sieve[i]!=0) printf("%d ",sieve[i]);
}
```

结构化设计

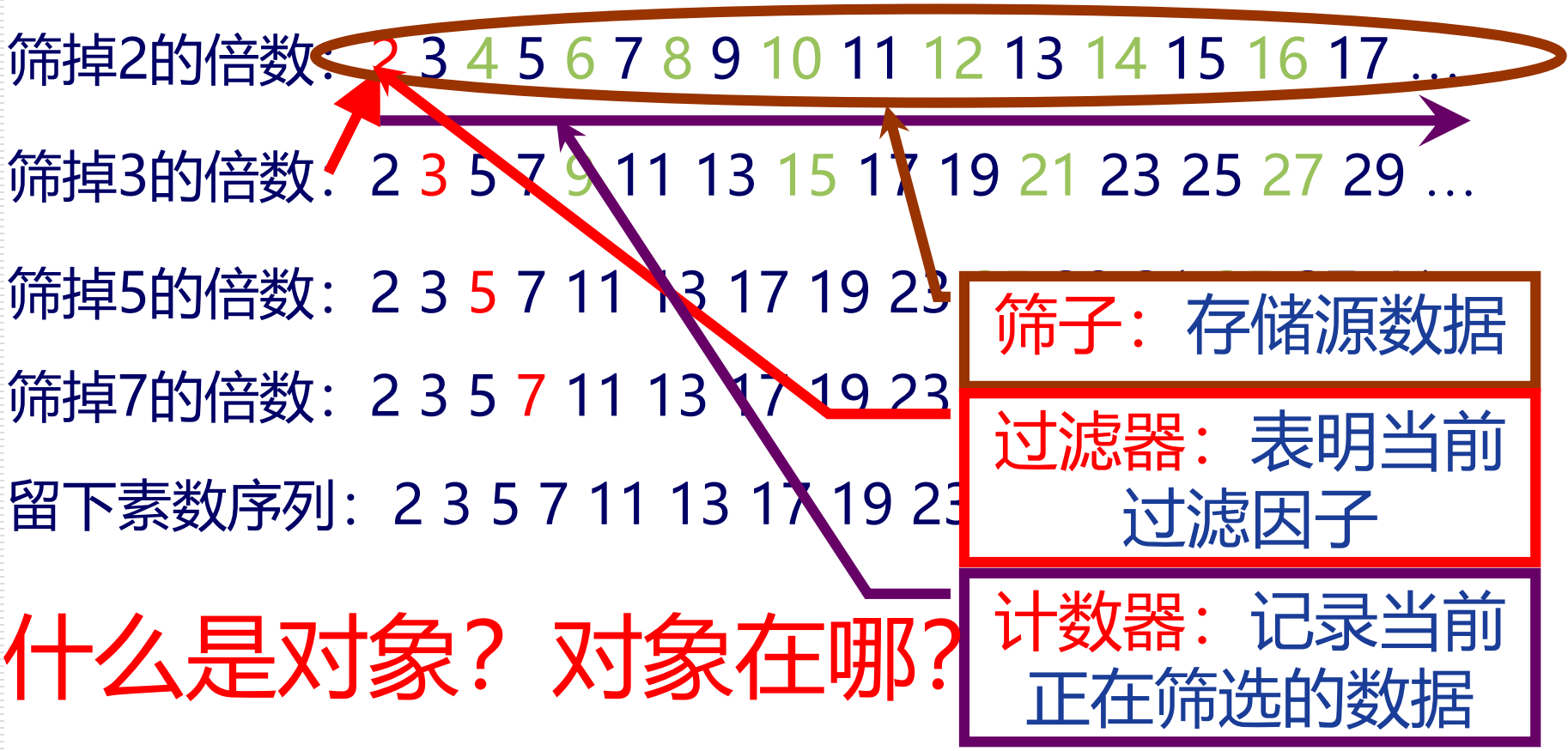


Java实现-是对象思维吗?

```
import java.lang.Math;
public class PrimerNumber{
    public static void main(String args[]) {
        int n=50;
        int sieve[]=new int[n-1];
        int iCounter=2, iMax, i;
        for(i=0;i<n-1;i++) {sieve[i]=i+2;}
        iMax=(int)Math.sqrt(n);
       while(iCounter<=iMax){
            for (i=2*iCounter-2; i<n-1; i+=iCounter)
                sieve[i]=0;
            iCounter++;
        }
        for(i=0; i<n-1; i++)
            if (sieve[i]!=0) System.out.println(sieve[i]);
    }}
```

用对象思维解决问题?

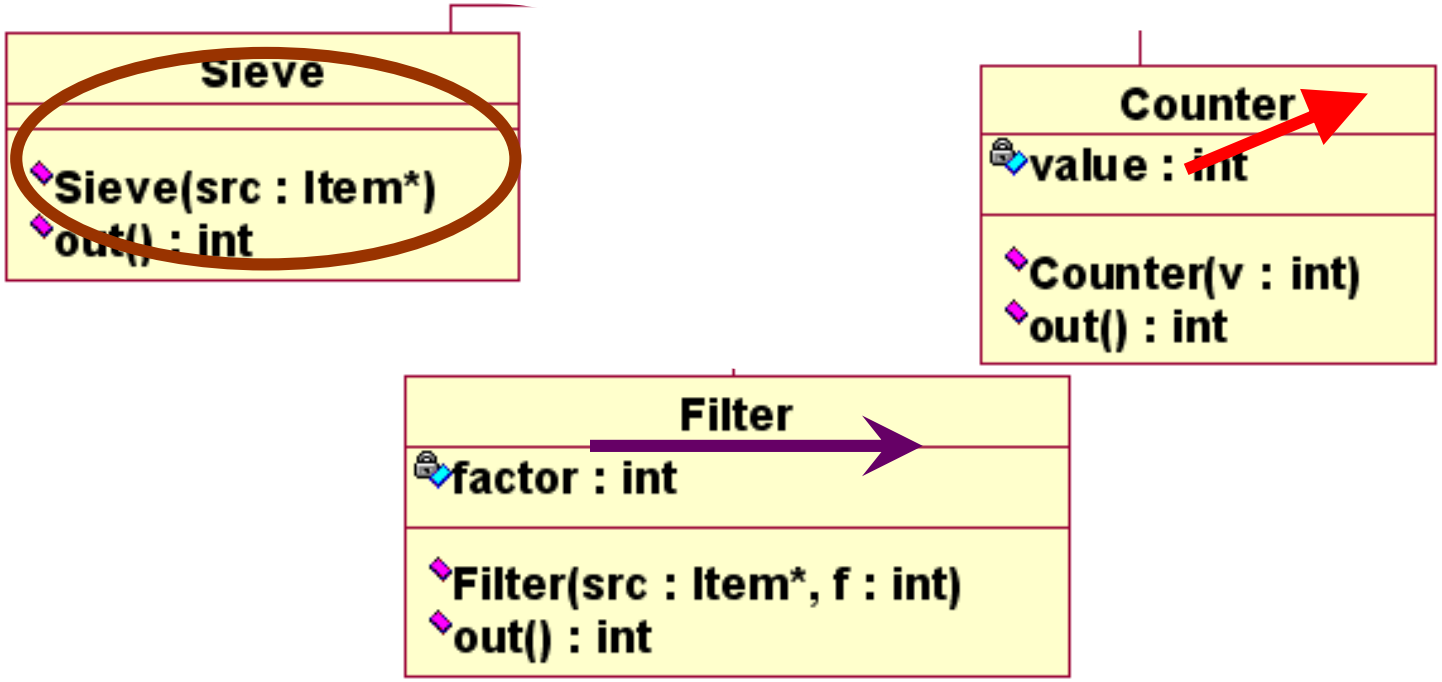
筛选法：生成 2< i <n 的整数序列，设n=50



什么是对象？对象在哪？

这才是对象思维!

象基类，为程序提供多态



验证设计方案

```
void main(){
        Counter c(2);
        Sieve s(&c);
        int next, n;
        cin>>n;
        while(1){
                next=s.out();
                if(next>n) break;
                cout<<next<<" ";
        }
        cout<<endl;
}
```

关键代码只有一行，
筛子自己知道如何找出素数

对象方法小结

- 通过UML类图（面向对象建模）可以更清楚表达设计思想，并为代码实现提供框架
- 针对数据的抽象：类
  - 类拥有自己的数据和行为，能够完成自身的工作职责
  - 过程是类的组成部分，为类提供行为
  - 通过类的对象之间的协作完成系统功能

面向对象技术的思考

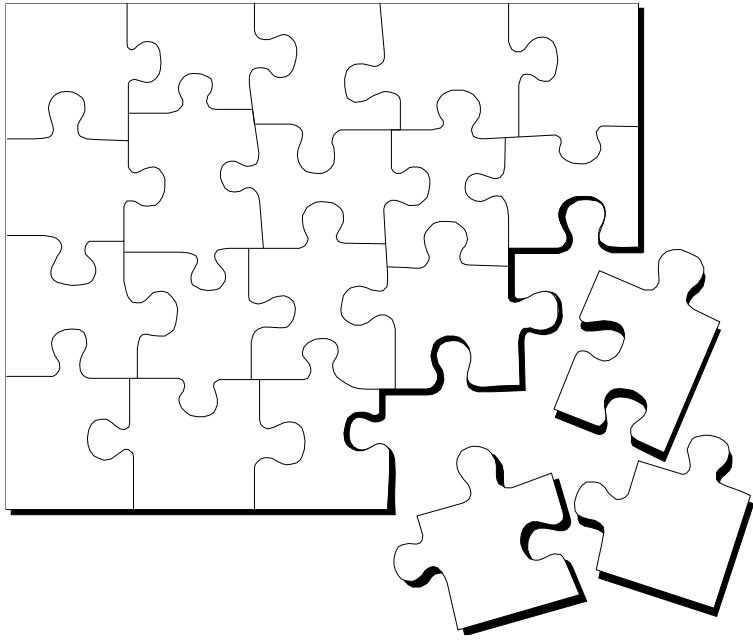
- 对象思维具有更大的灵活性，更好的模块化，可以进行更大规模的设计
- 面向对象设计和开发的难度更大，面临着对象识别、职责分配、多态抽象等一系列问题
  - 学习更多知识和技术，并掌握一系列面向对象的设计原则和模式
  - 图形化工具（UML）有助于表达和交流设计思想，并简化实现的过程

内容安排

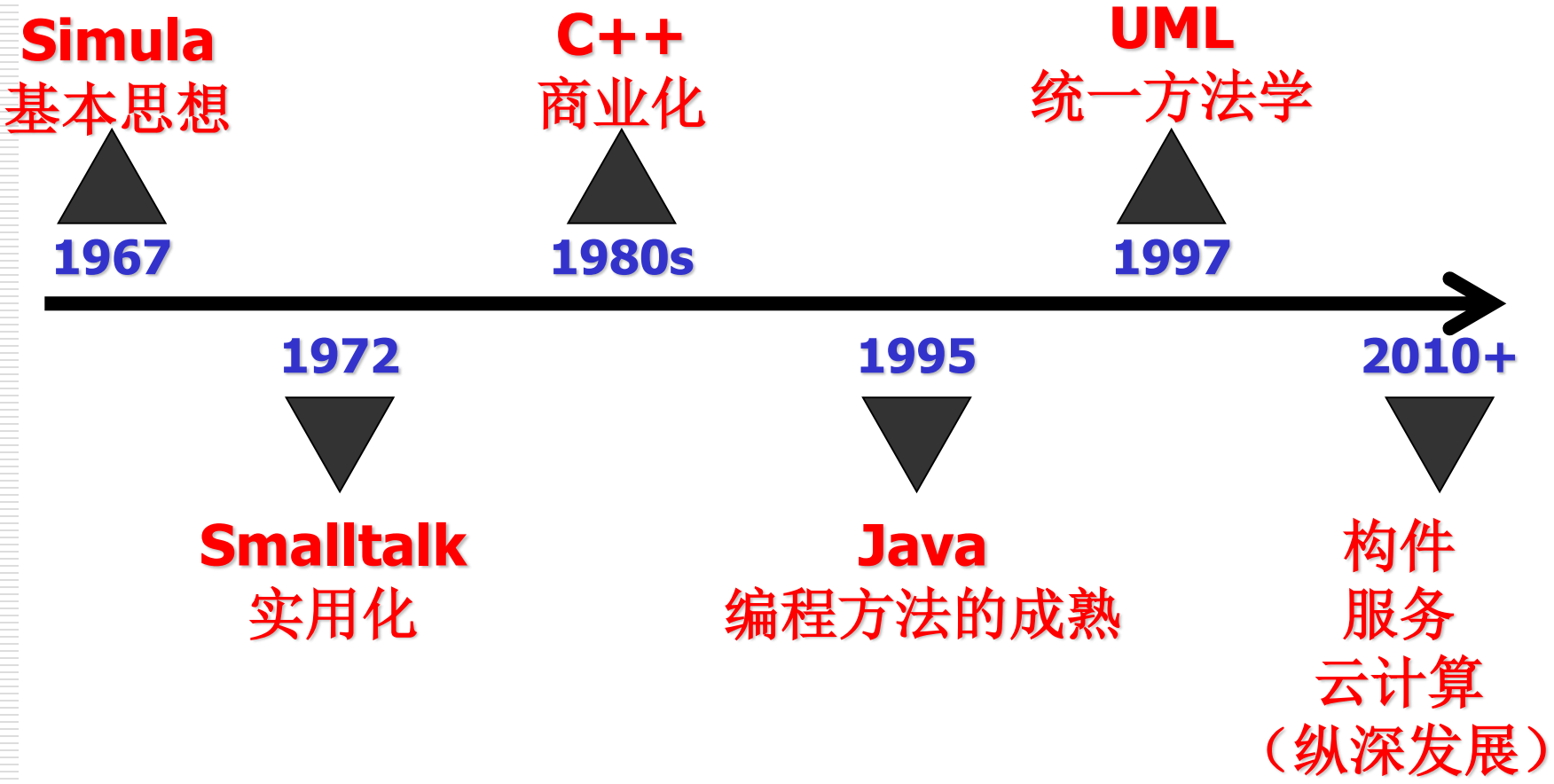
- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

对象技术？

- 面向对象技术是一系列指导软件构造的原则（如抽象、封装、多态等），并通过语言、数据库和其它工具来支持这些原则
  - 从本质上讲，对象技术对一系列相关原则的应用
  - 面向对象技术 = 类+对象+抽象+封装+分解+泛化+多态+分层+复用…



对象技术发展历史

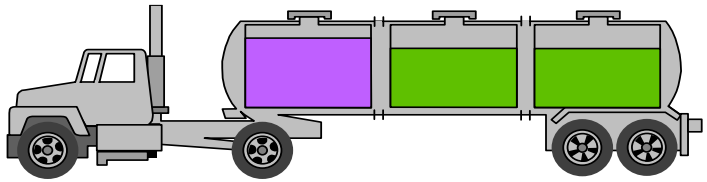

Simula
基本思想
1967
C++
商业化
1980s
UML
统一方法学
1997
1972
Smalltalk
实用化
1995
Java
编程方法的成熟
2010+
构件
服务
云计算
（纵深发展）

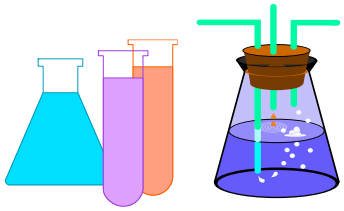
对象技术的优势

- 沟通
  - 顺应人类思维习惯，让软件开发人员在解空间中直接模拟问题空间中的对象及其行为
- 稳定
  - 较小的需求变化不会导致系统结构大的改变
  - 用较稳定的把不稳定的封装起来
- 复用
  - 提供了多种软件复用的手段：代码、设计、需求、领域…
- ……

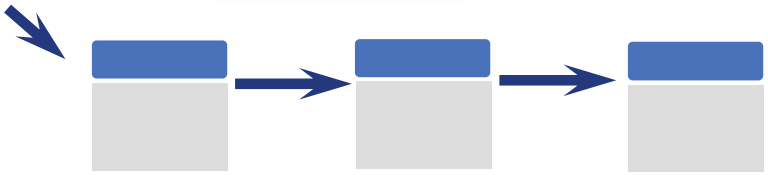
对象

- 对象（Object）是一个实体、一件事、一个名词，可以获得的某种东西，可以想象有自己标识的任何事物
  - 物理实体
  - 概念实体
  - 软件实体





化学过程



链表

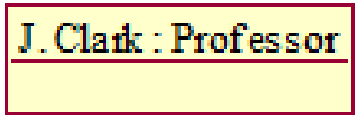
对象-正式定义

- 对象是一个实体，这个实体：
  - 具有明确定义的边界和标识
    - 边界意味着对象是一个封装体，通过封装来与其它对象分隔
    - 标识则表明每一个对象都是唯一的
  - 对象封装了状态和行为
    - 对象的状态通过对象的属性（attribute）和关系（relationship）来表达
    - 对象的行为通过对象的操作（operation）、方法（method）和状态机（state machine）来表达

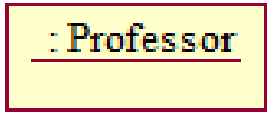
在UML中表示对象

- 在UML中，对象用矩形框来表示，对象的名字写在矩形框内部，并加上下划线来表示

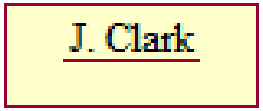
Professor J Clark



对象名+类名



只有类名
(匿名对象)



只有对象名

类

- 类就是一系列对象的抽象描述，这些对象共享相同的属性、操作、关系和语义
  - 一个具体的对象是该类的一个实例
- 类是一种抽象
  - 将相似的实体抽象成相同的概念
  - 抽象过程强调相关特征而忽略其它的特征

类的抽象，取决于需求

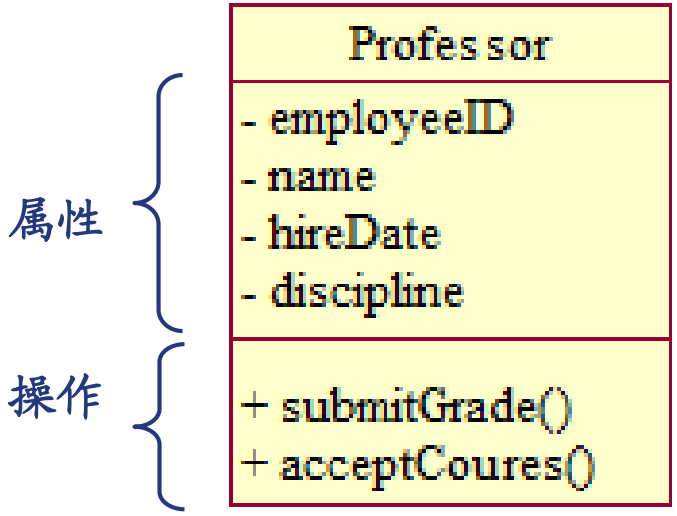
A. 人
B. 男人、女人
C. 老板、员工
D. 老师、学生

- 类的抽象过程与需求紧密相关
  - 强调相关的特征，而忽略不相关特征

与需求紧密相关

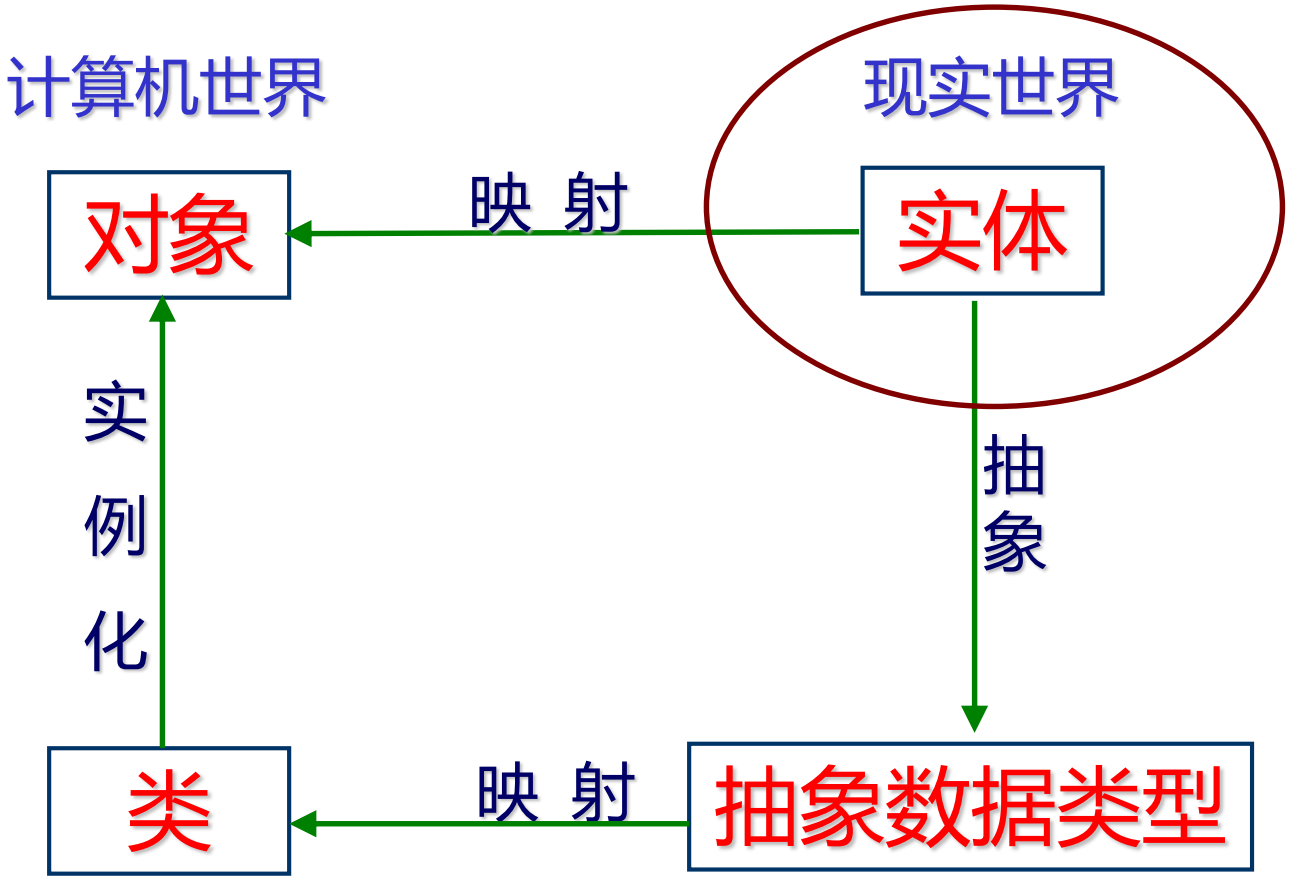
UML中的类

- 在UML中，采用矩形框表示类，可以将矩形框划分为三个区域，分别表示类名、属性和操作



Professor J Clark

类和对象



其他更多的对象原则

- 面向对象技术基本原则
  - 抽象（Abstraction）、封装（Encapsulation）、分解（Decomposition）
  - 泛化（Generalization）、多态（Polymorphism）
  - 分层（Hierarchy）、复用（Reuse）
  - ……

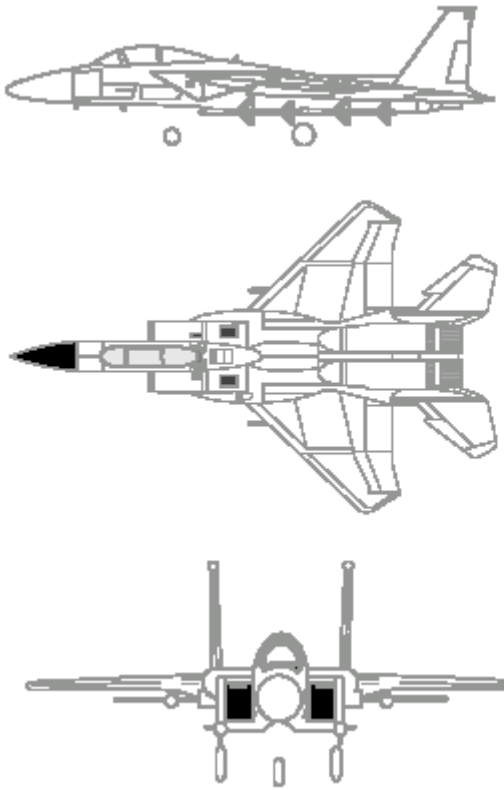
详细讲解请阅读教材，推荐观看课程视频（访问清华大学出版社官网：1.4-对象技术相关原则，扫二维码或访问以下链接）
https://appgijaeprl2324.h5.xiaoeknow.com/v1/course/video/v_5e52775bbf884_dj5wJI7U?type=2&pro_id=p_5e4a58fa34ba8_U2mShSMQ&from_multi_course=1

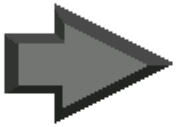
内容安排

- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

模型

- 模型是现实世界的简化

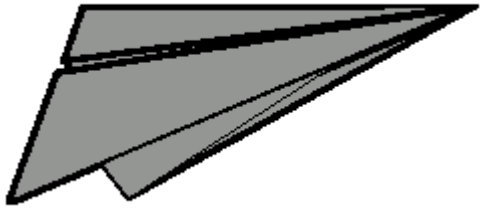




模型的重要性

不重要

非常重要



纸飞机

喷气式战斗机

建模的目的

- 建模的根本目的是为了更好地理解待开发的系统
  - 模型有助于按照所需的样式可视化(Visualize)目标系统
  - 模型能够描述(Specify)系统的结构和行为
  - 模型提供构造(Construct)系统的模板
  - 模型可以文档化(Document)设计决策

建模的基本原则

- 建模过程需要遵循的原则
  - 选择合适的模型：所创建的模型对解决方案的形成具有重要的影响
  - 模型具有不同的精确程度：面向不同的用户提供不同抽象层次的模型
  - 好的模型是与现实相联系的：简化不能掩盖掉任何重要的细节
  - 单一模型是不够的：需要从多个视角创建不同的模型

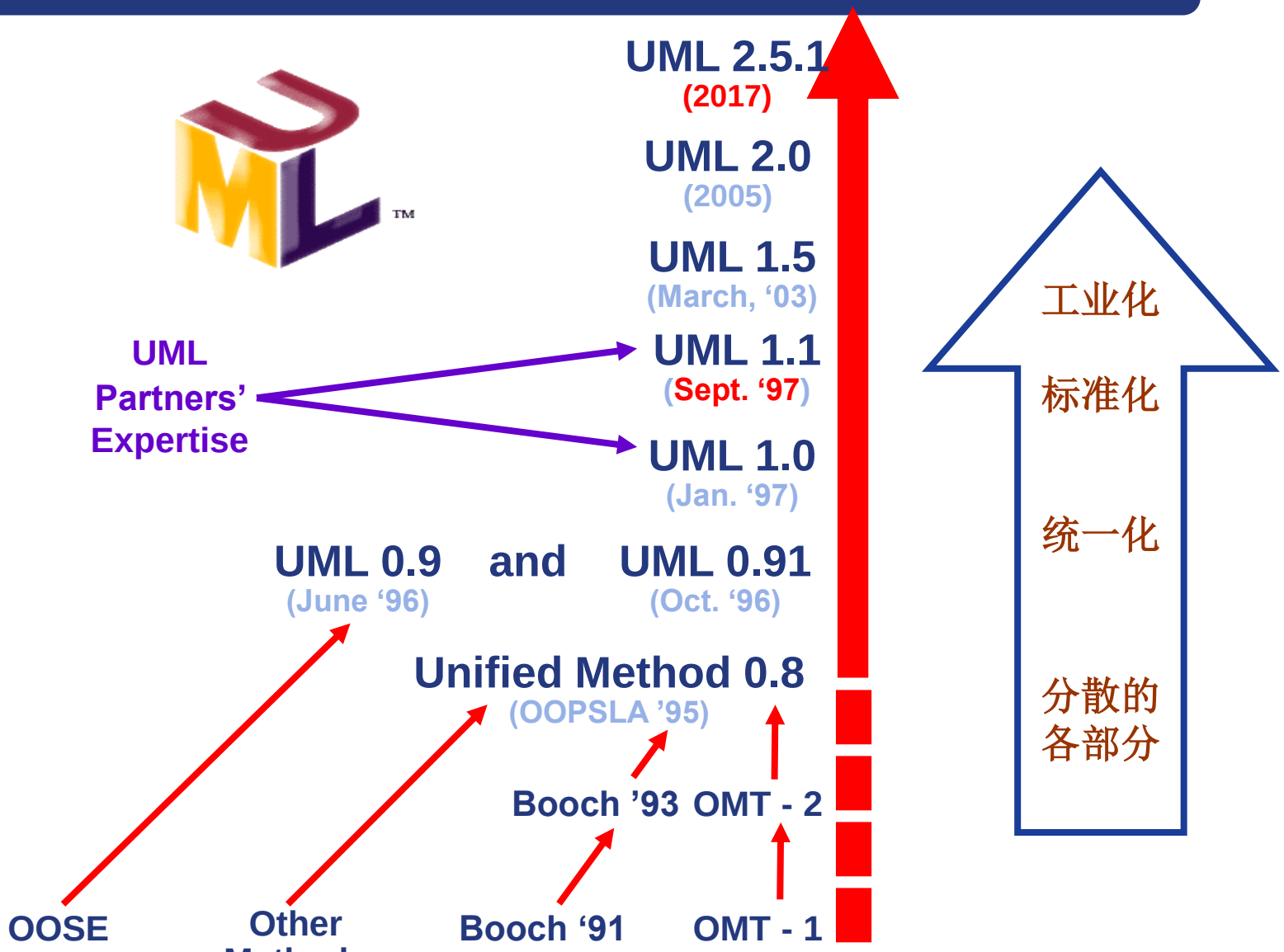
什么是统一建模语言?

- 统一建模语言 is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

the artifacts of a software-intensive system

Unified Modeling Language（统一建模语言）是对象管理组织（OMG）制定的一个通用的、可视化的建模语言标准，可以用来可视化（visualize）、描述（specify）、构造（construct）和文档化（document）软件密集型系统的各种工件（artifacts，又译制品）

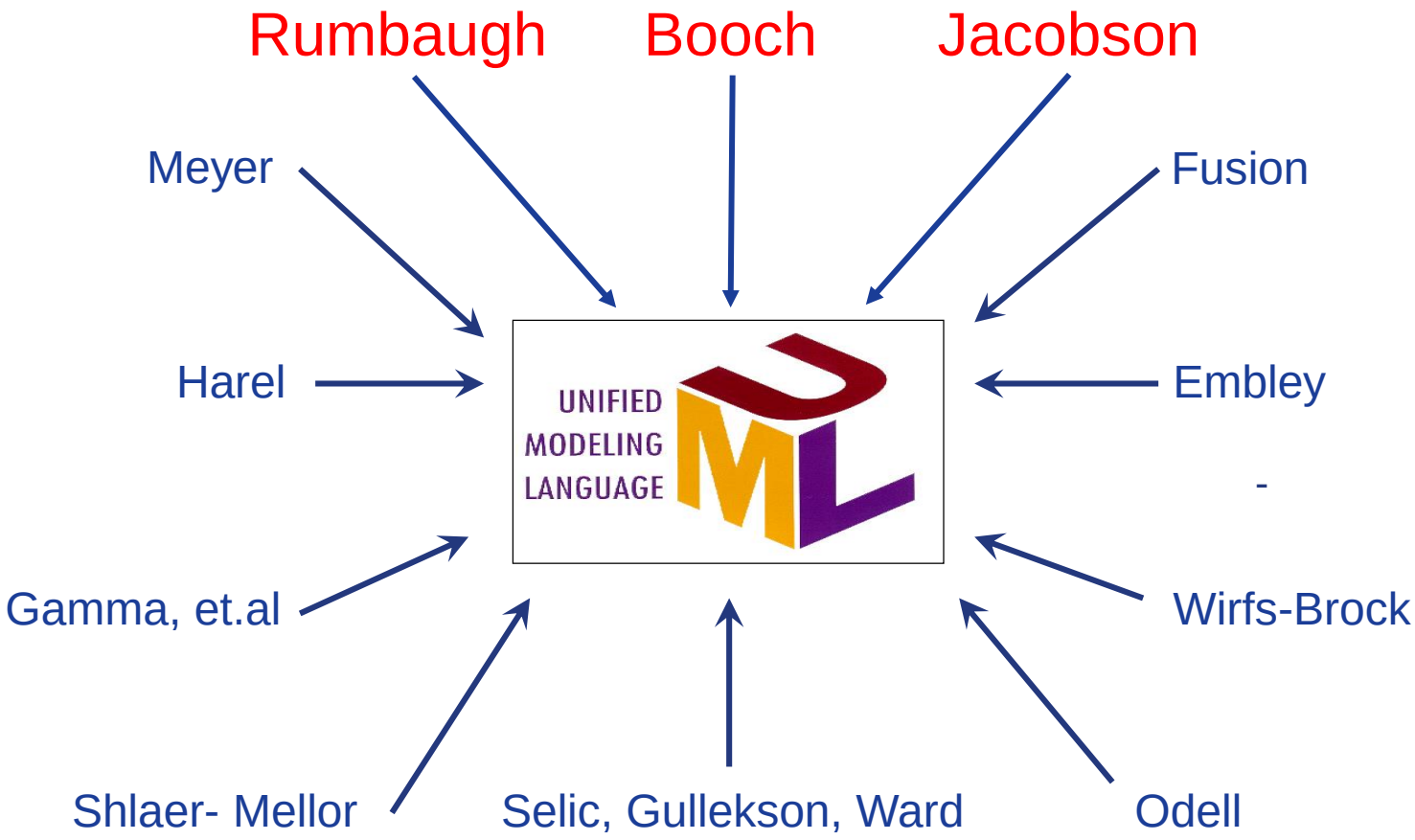
UML发展历史



UML现状

- UML 1.x
  - 目前应用较为普及，包括从1.1~1.5
  - 2003年3月发布的UML 1.5
- UML 2
  - 2005年7月正式发布UML 2.0，分为基础结构和上层结构
  - 2011年3月UML2.4，2011年8月UML2.4.1，2012年5月ISO/IEC 19505-1:2012， ISO/IEC 19505-2:2012
  - 2015年6月UML2.5
  - 2017年12月UML2.5.1

UML的统一


Rumbaugh
Booch
Jacobson
Meyer
Fusion
Harel
Embley
UNIFIED
MODELING
LANGUAGE
Gamma, et.al
Wirfs-Brock
Shlaer- Mellor
Selic, Gullekson, Ward
Odell

UML的统一（续）

- 统一了什么?
  - 开发生命周期
  - 应用领域
  - 实现语言和平台
  - 开发过程
  - 自身的内部概念

内容安排

- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

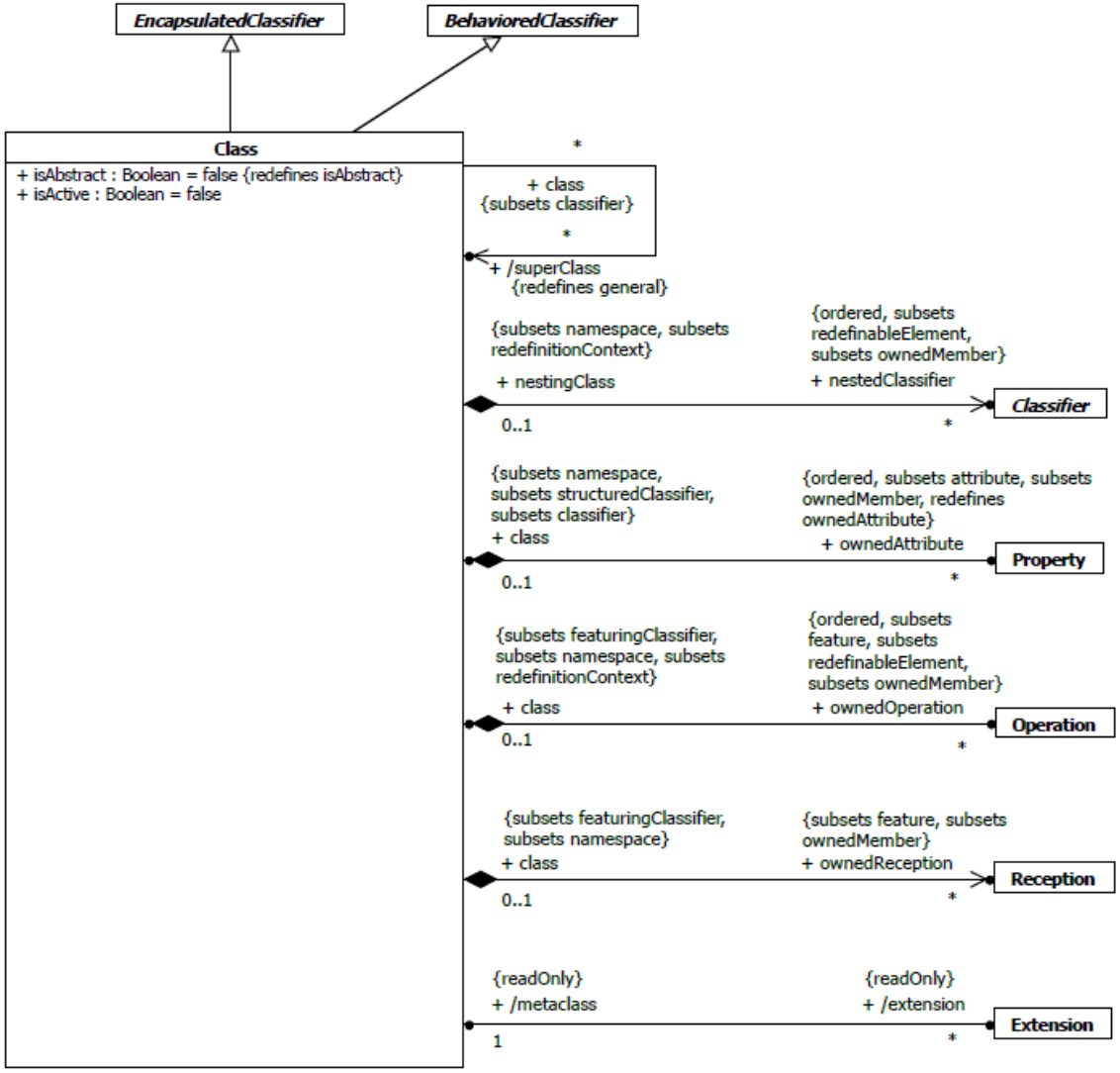
UML组织结构

- 从UML2开始，整个UML规范被划分成基础结构和上层结构两个相对独立的部分
  - 基础结构（Infrastructure）是UML的元模型，它定义了构造UML模型的各种基本元素
  - 上层结构（Superstructure）则定义了面向建模用户的各种UML模型的语法、语义和表示
- 从UML2.5开始，为了消除冗余并简化UML规范，基础结构部分不再作为UML规范的一部分，UML元类在UML规范相应的章节中被完整地定义

UML语法结构

- UML的抽象语法使用UML元模型来定义
  - 这个元模型本身也是用UML来定义（准确来说是一个受限的UML子集，这个子集符合OMG的MOF规范）
  - 在UML规范中，主要采用UML类图来描述各元素的抽象语法，采用约束机制和自然语言（文本）来描述模型语义

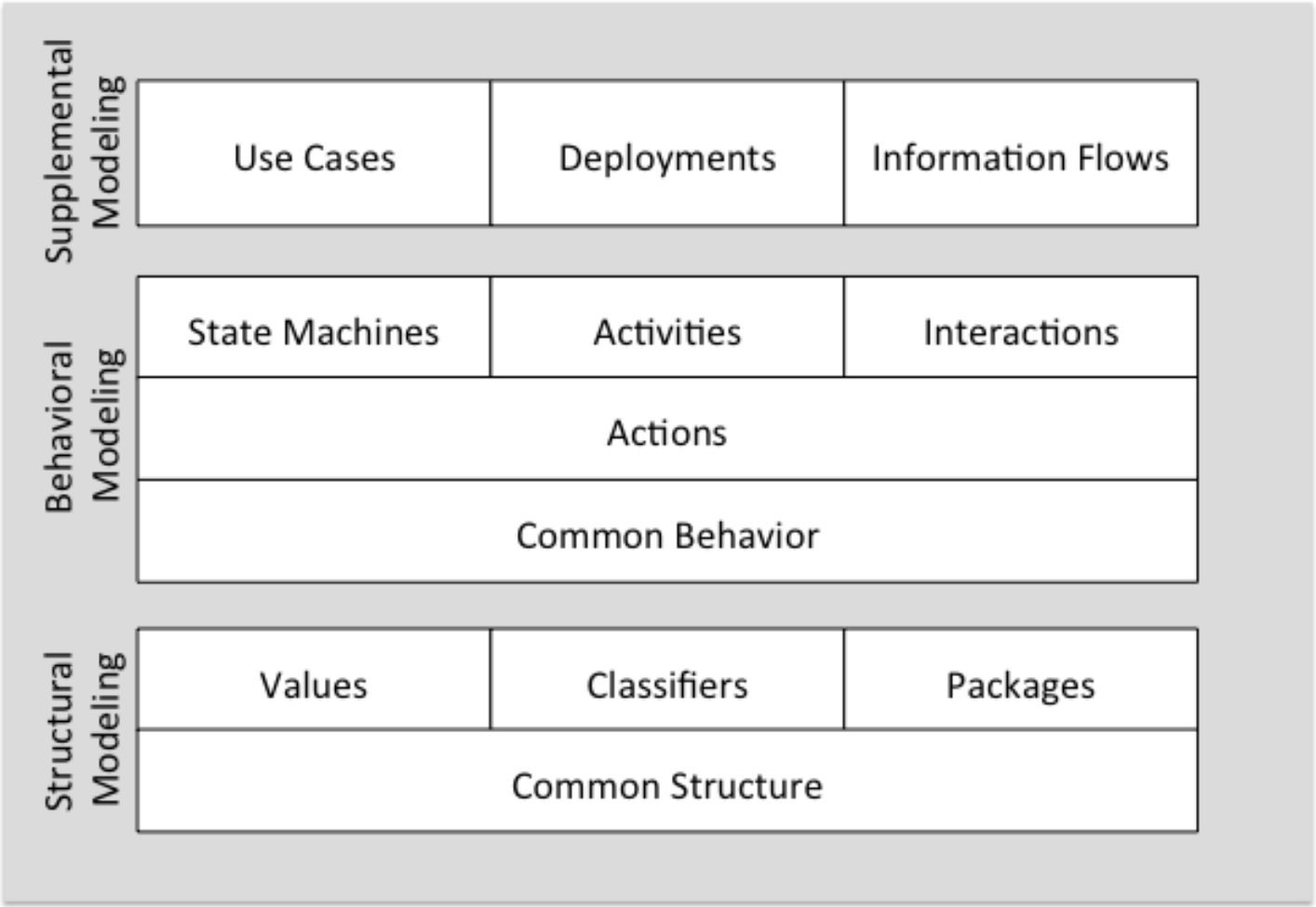
类的元模型（语法结构）



UML语义结构

- UML自身的语义与被建模系统的UML模型上所声明的标准含义有关，这有时被称为UML运行时语义
- UML模型划分为两类语义域。
  - 结构语义：定义了在建模域中关于个体的UML结构化模型元素的含义，也称为静态语义
  - 行为语义：定义了在建模域中关于个体如何随着时间变化而做出不同行为的UML行为模型元素，也称为动态语义。

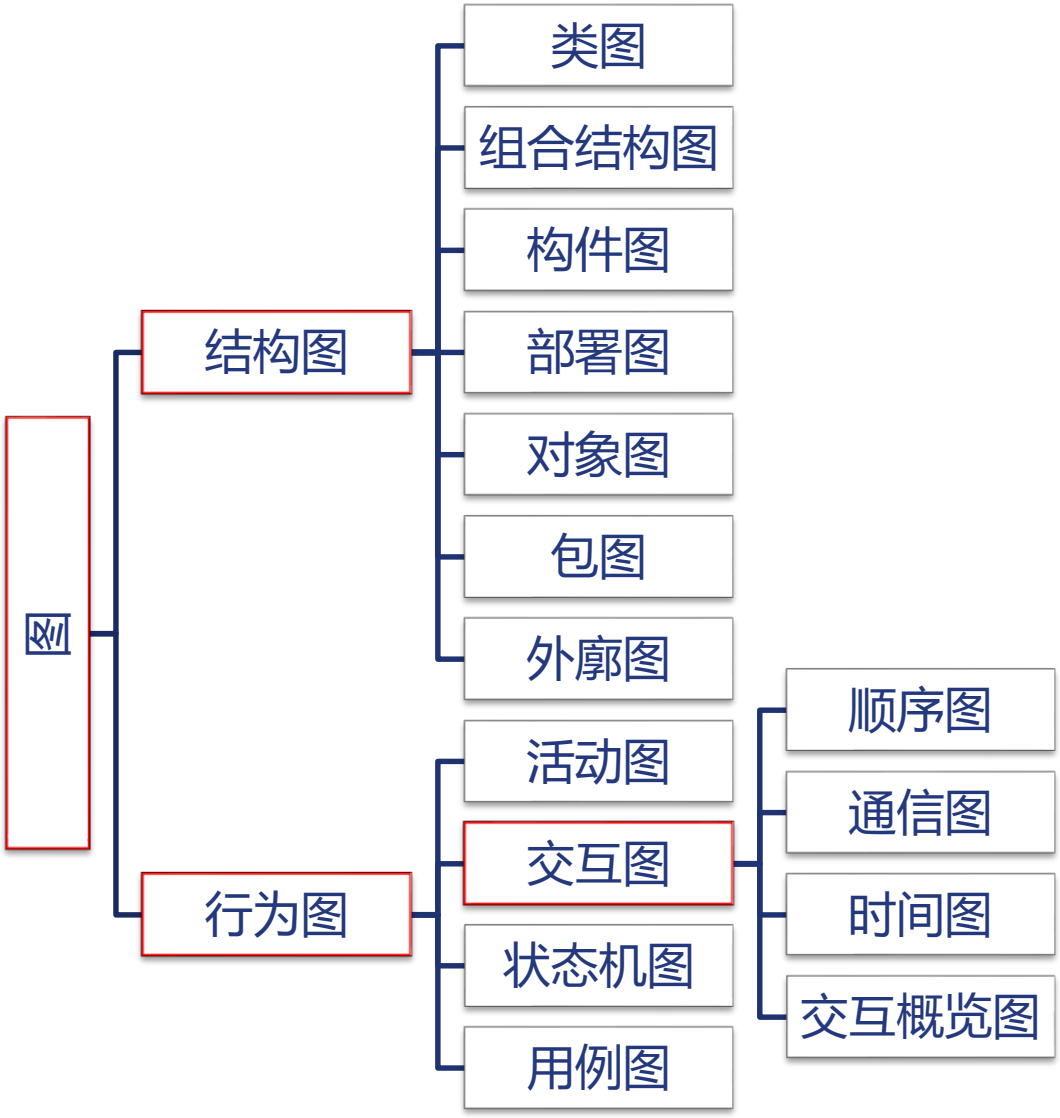
UML语义域



内容安排

- 从传统到对象
- 对象技术
- 可视化建模和UML
- UML组织结构
- UML建模实践

UML图



UML2 14种图-结构模型

- 类模型
  - 类图：类、接口以及它们之间的关系
  - 包图：包以及它们之间的依赖关系
  - 对象图：对象以及它们之间的链接关系
- 构件模型
  - 构件图：构件以及之间的依赖关系
- 组合结构模型
  - 组合结构图：系统某一部分(组合结构)的内部结构
- 部署模型
  - 部署图：构件在节点上的部署情况
- 外廓模型
  - 外廓图：外廓以及所包含构造型、标记值、约束和基类等

UML2 14种图-行为模型

- 活动模型
  - 活动图：动作及活动执行的控制流或数据流
- 交互模型
  - 顺序图：对象间交互序列的执行顺序
  - 通信图：对象间的协作以及交互序列
  - 时间图：对象交互中真实的时间信息
  - 交互纵览图：多个交互之间的执行顺序
- 状态机模型
  - 状态机图：对象所经历的状态迁移过程
- 用例模型
  - 用例图：以外部用户视角描述系统能力

UML建模工具

- IBM Rational Suite
  - Rational Rose 2003
    - 经典的UML建模工具，目前仍有广泛的应用
    - 不支持UML2.0
  - Rational Software Architecture
    - IBM兼并Rational之后，重新基于Eclipse平台构建的集成开发平台，提供从业务建模、需求分析、设计到系统实现的完整环境
  - IBM Rational Rhapsody
    - IBM兼并另一家UML建模工具后重新发布的产品
    - 主要用于嵌入式领域建模，涉及软硬件等各个层次的模型

UML Tools (cont.)

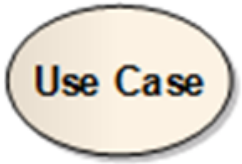
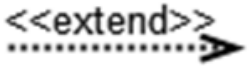
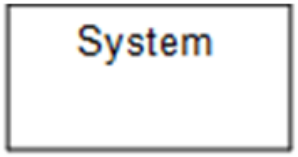
- Enterprise Architect
  - http://www.sparxsystems.com.au/
- PowerDesigner
  - http://www.sybase.com/products/powerdesigner/
- MagicDraw
  - http://www.magicdraw.com/
- StarUML
  - http://staruml.sourceforge.net
- UModel
  - http://www.altova.com/products/umodel/uml_tool.html
- Kant&Plato
  - 楚凡科技（中国） http://www.trufun.net/
- 更多： http://www.umlchina.com/Tools/Newindex1.htm

示例：图书馆管理系统

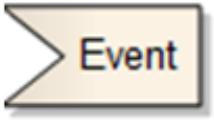
- 某图书馆管理系统
  - 是一个基于Web的计算机应用系统
  - 读者可以查询图书信息以及借阅信息
  - 读者可以通过系统预约所需的图书
  - 图书馆工作人员利用该系统完成读者的借书、还书业务
  - 图书馆工作人员可以对图书信息、读者信息等进行维护
  - 对于到期的图书，系统会自动向读者发送催还信息
  - 管理员会定期进行系统维护
  - ……

详细建模过程和工具使用，请阅读教材，并推荐观看课程视频（课程网站或访问清华大学出版社官网：2.5-使用EA进行UML建模实践，扫二维码或访问以下链接）
https://appgijaeprl2324.h5.xiaoeknow.com/v1/course/video/v_5e4a6dca2d209_bII1ogXs?type=2&pro_id=p_5e4a58fa34ba8_U2mShSMQ&from_multi_course=1

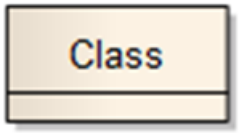
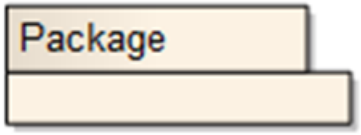
附录：用例图元语

结构元素	图形符号	关系元素	图形符号
用例 (Use Case)		关联 (Association)	
参与者 (Actor)		扩展 (Extend)	
系统边界 (System Boundary)		包含 (Include)	
		泛化 (Generalization)	
注释 (Note)		注释连接 (Note link)	

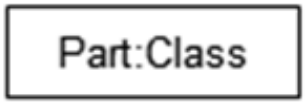
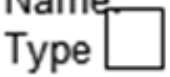
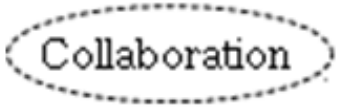
附录：活动图元语

元素	图形符号	元素	图形符号
活动/动作 (Activity/Action)		起点 (InitialNode)	
对象 (Object)		终点 (FinalNode)	
发送事件 (SendEvent)		流结束 (FlowFinal)	
接收事件 (AcceptEvent)		分叉/合并 (Fork/Join)	
分区 (Partition)		控制流 (ControlFlow)	
决策点 (Decision)		对象流 (ObjectFlow) (UML1.x 中为虚线)	

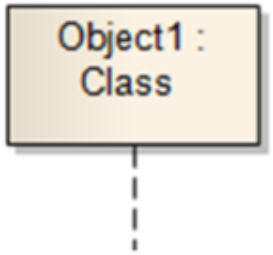
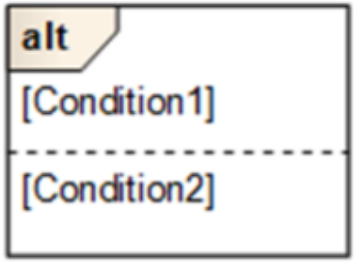
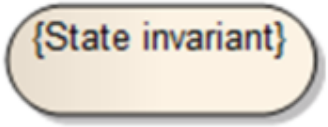
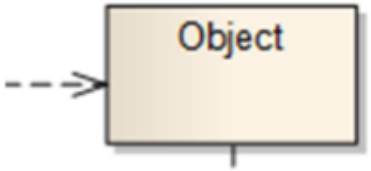
附录：类图、对象图、包图元语

结构元素	图形符号	关系元素	图形符号
类 (Class)		依赖 (Dependency)	
包 (Package)		关联 （Association）	
接口 (Interface)	 或 	聚合 (Aggregation)	
		组合 (Composition)	
对象 (Object)		泛化 (Generalization)	
		实现 （Realization）	

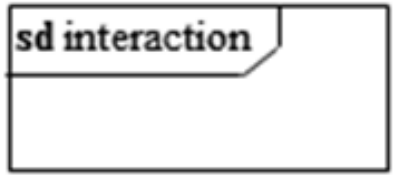
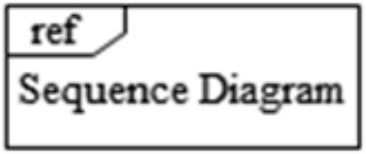
附录：组合结构图元语

结构元素	图形符号	关系元素	图形符号
部件 (Part)		连接 (Connector)	
端口 (Port)		角色绑定 (Role binding)	
协作 (Collaboration)			

附录：顺序图元语

元素	图形符号	元素	图形符号
对象/生命线 (Object/Lifeline)		同步消息 (Synchronous Message)	
交互片段 (Interaction Frame)		异步消息 (Asynchronous Message)	
执行发生 (Execution Occurrence)		返回消息 (Return Message)	
状态不变式 (State Invariant)		创建消息 (Create Message)	

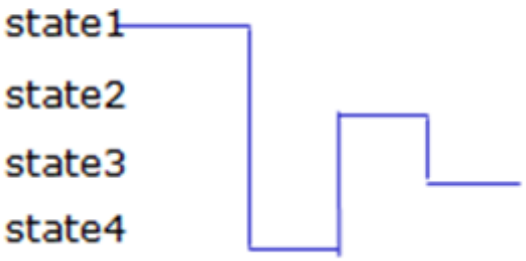
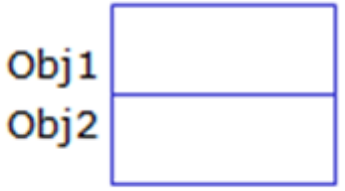
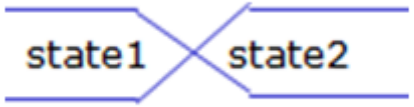
附录：交互纵览图元语

元素	图形符号	元素	图形符号
片段 (Frame)		起点 (InitialNode)	
交互引用 (Interaction Use)		终点 (FinalNode)	
决策点 (Decision)		控制流 (ControlFlow)	

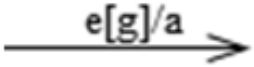
附录：通信图元语

元素	图形符号	元素	图形符号
对象/生命线 (Object/Lifeline)		同步消息 (Synchronous Message)	
链接 (Link)		异步消息 (Asynchronous Message)	
		返回消息 (Return Message)	

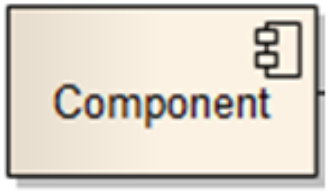
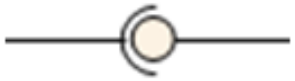
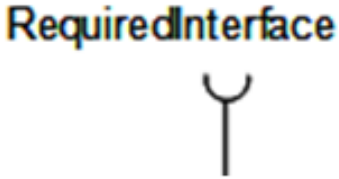
附录：时间图元语

元素	图形符号	元素	图形符号
状态/条件时间线 (State or condition timeline)		生命线 (Lifeline)	
取值生命线 (General value lifeline)		消息 (Message)	

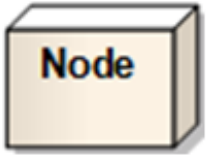
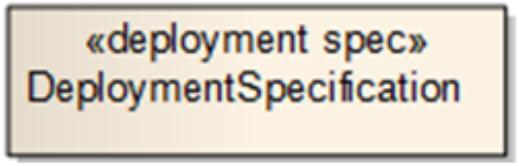
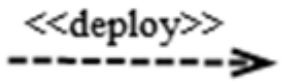
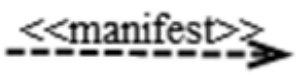
附录：状态机图元语

元素	图形符号	元素	图形符号
状态 (State)		初态 (Initial State)	
浅度历史 (Shallow History)		终态 (Final State)	
深度历史 (Deep History)		转移 (Transition)	
选择 (Choice)		分叉/合并 (Fork/Join)	

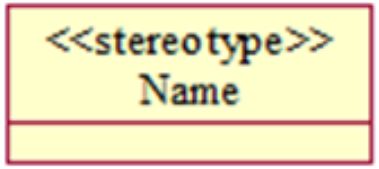
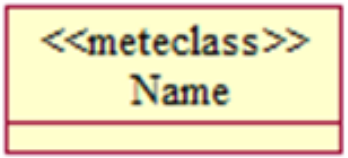
附录：构件图元语

结构元素	图形符号	关系元素	图形符号
构件 (Component)		装配连接 (Assembly connector)	
所供接口 (Shallow History)		委托连接 (Delegate connector)	
所需接口 (Required Interface)		依赖 (Dependency)	

附录：部署图元语

结构元素	图形符号	关系元素	图形符号
节点 (Node)		通信路径 (Communication Path)	
制品 (Artifact)		依赖 (Dependency)	
部署规范 （Deployment specification）		部署 (Deploy)	
		承载 (Manifestation)	

附录：外廓图元语

结构元素	图形符号	关系元素	图形符号
构造型 (Stereotype)		扩展 (Extension)	
元类 (Metaclass)		外廓应用 (ProfileApplication)	
外廓 (Profile)		引用 （Reference）