

面向对象软件开发概述



- 面向对象程序设计的基本概念
- 面向对象的软件开发过程
- 面向对象程序设计方法的优点



- 程序设计方法的发展
 - 程序
 - 程序设计语言
 - 程序设计方法
- 软件危机与结构化程序设计
- 面向对象程序设计



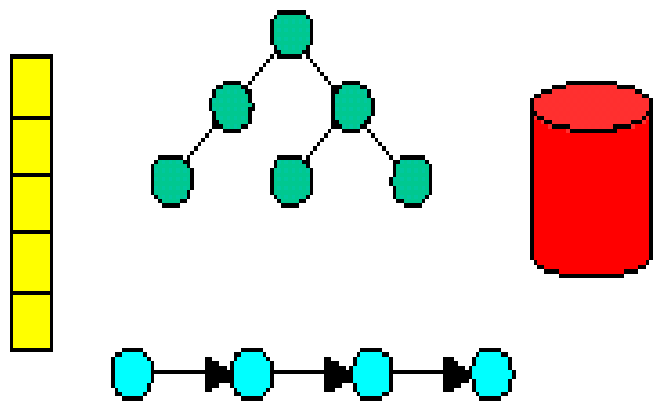
- 面向过程的程序设计: 自顶向下、功能分解、模块化

- 执行一系列算法来解决问题
- 在数据结构上执行特定的功能

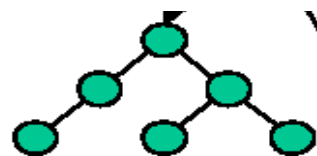
- 面向过程的程序设计的缺点

- 数据与过程分离
- 可重用性差
- 系统维护困难
- 难以开发大规模复杂软件系统





```
#include <iostream.h>
main() {
    int i;
    for (i=0; i<100; i++) {
        if (i%2=0)
            cout << "\n";
        cout << "Hello, world...";
    } // for
} // main()
```



find(item_t item, note_t n);

delete(index_t i, array_t a);

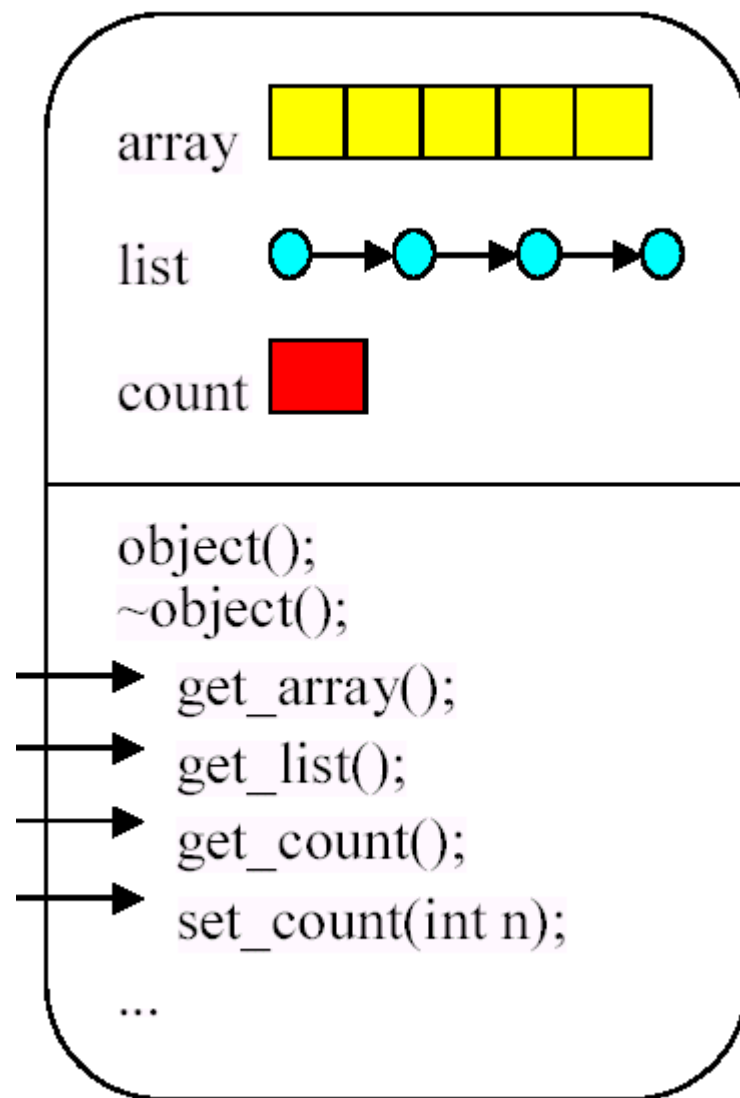
print(list_t list, size_t n);

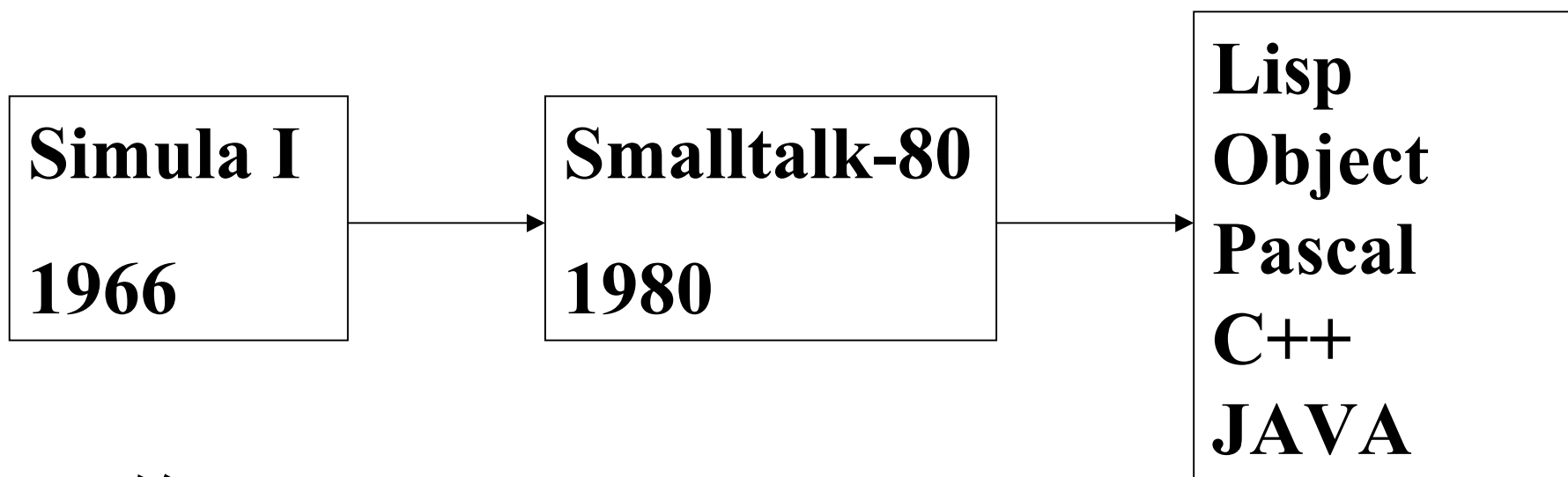
save(item_t item, db_t db);

main();

[返回](#)

- 数据与算法是*集成的*
- 对象 (实例) 是由对象名、属性 (数据) 和操作 (过程) 组成





开始 SIMULA67

- 具有面向对象特性的模拟语言

Smalltalk

- 第一个“纯”的面向对象语言
- 类和方法具有模块化概念

Java

- 完全 面向对象程序设计语言



- 一般意义: **everything**

对象是现实世界的实体或概念在计算机逻辑中的抽象表示

- 不同观点:

现实世界

问题世界

计算机系统

← 模拟

现实对象

→

问题对象

→

计算机对象

抽象

→

表示



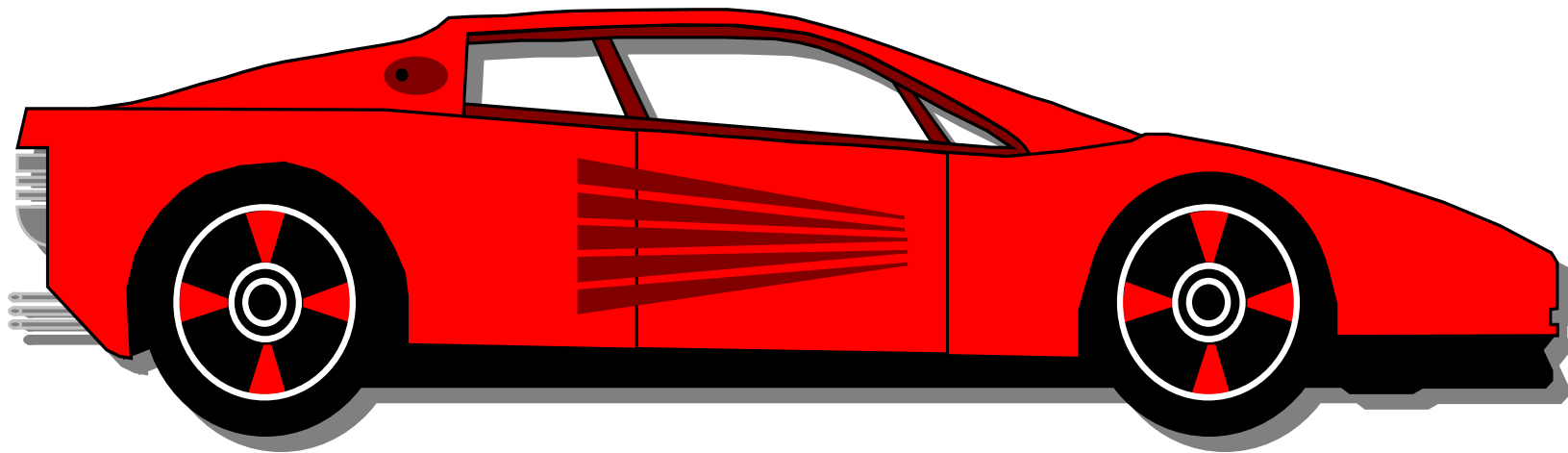
- 什么是对象?

- 软件对象是模仿现实世界的对象 – 具有属性和操作
- 每个对象必须有一个唯一的 **ID**
- 软件对象通过属性来表示其状态，用方法来实现其操作
 - 属性：对象的变量
 - 方法：操作，与其它语言中的函数类似，用于模拟对象的行为

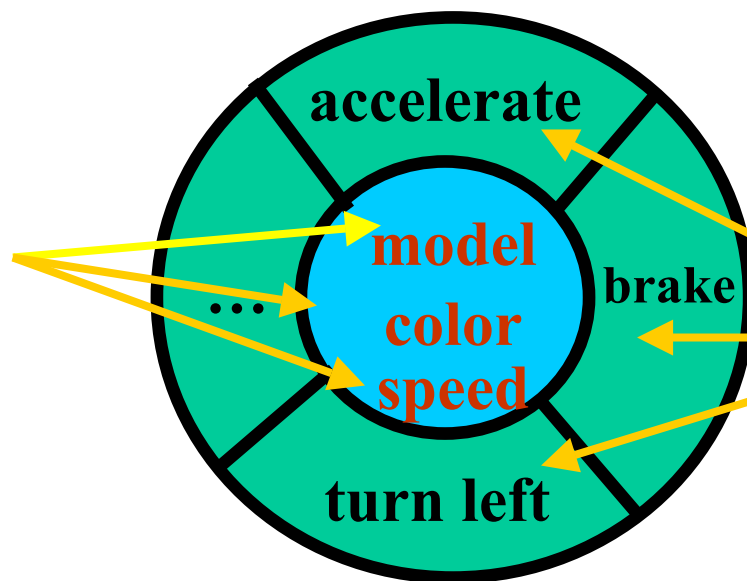
- 例如：

- **Car: model, color, year, turn left, accelerate, ...**
- **TV: brand, channels, set channel, display channel, ...**



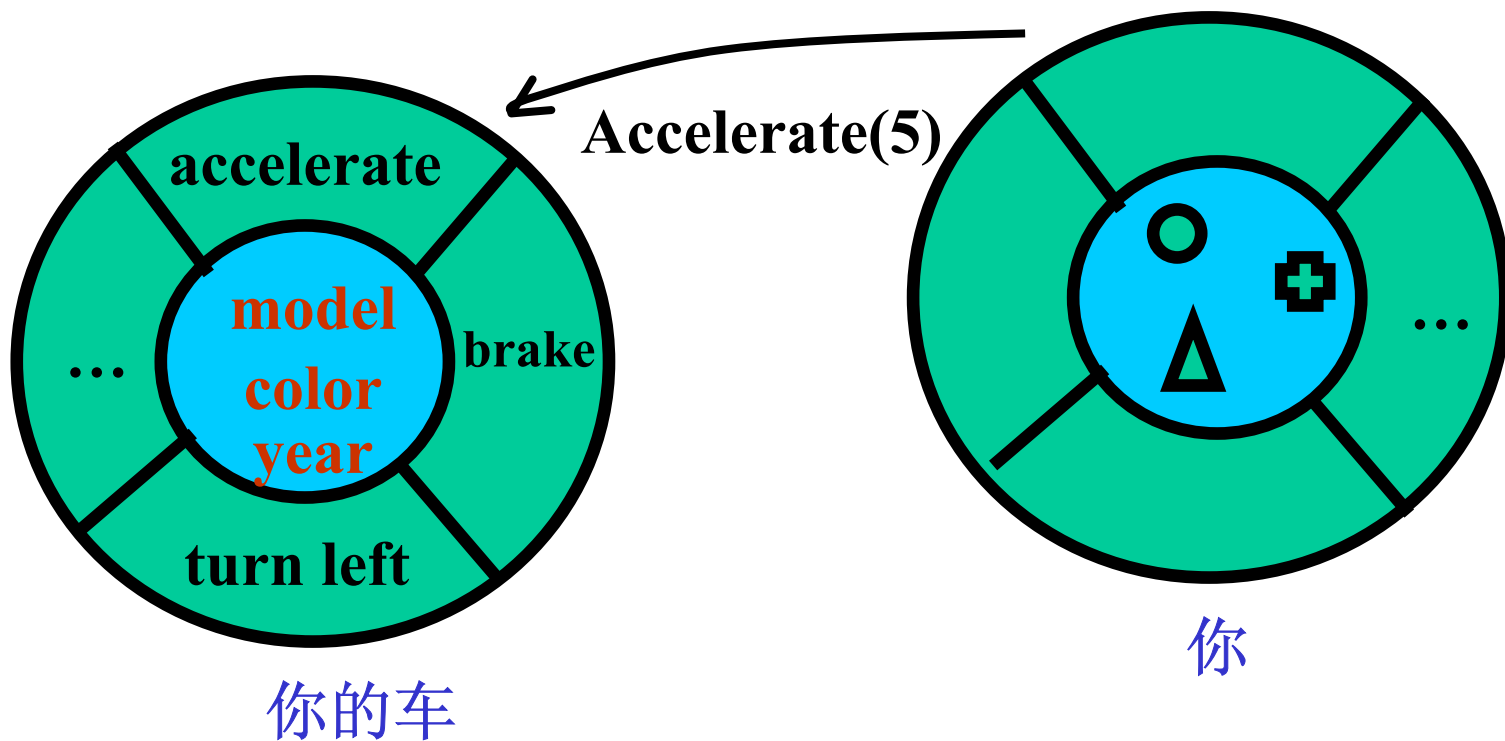


**attributes
/variables**



**behaviors
/methods**

- 对象通过彼此之间发送消息来相互作用和通讯
- 对象发送消息通过调用其它对象的公共接口（方法）



- 什么是类?
 - 类是同种对象的集合与抽象
 - 类是一种抽象的数据类型，它是所有具有一定共性的对象的抽象
 - 属于类的某一个对象称为是该类的一个实例
 - 类有助于软件的重用
 - 一个类可以用来产生多个对象

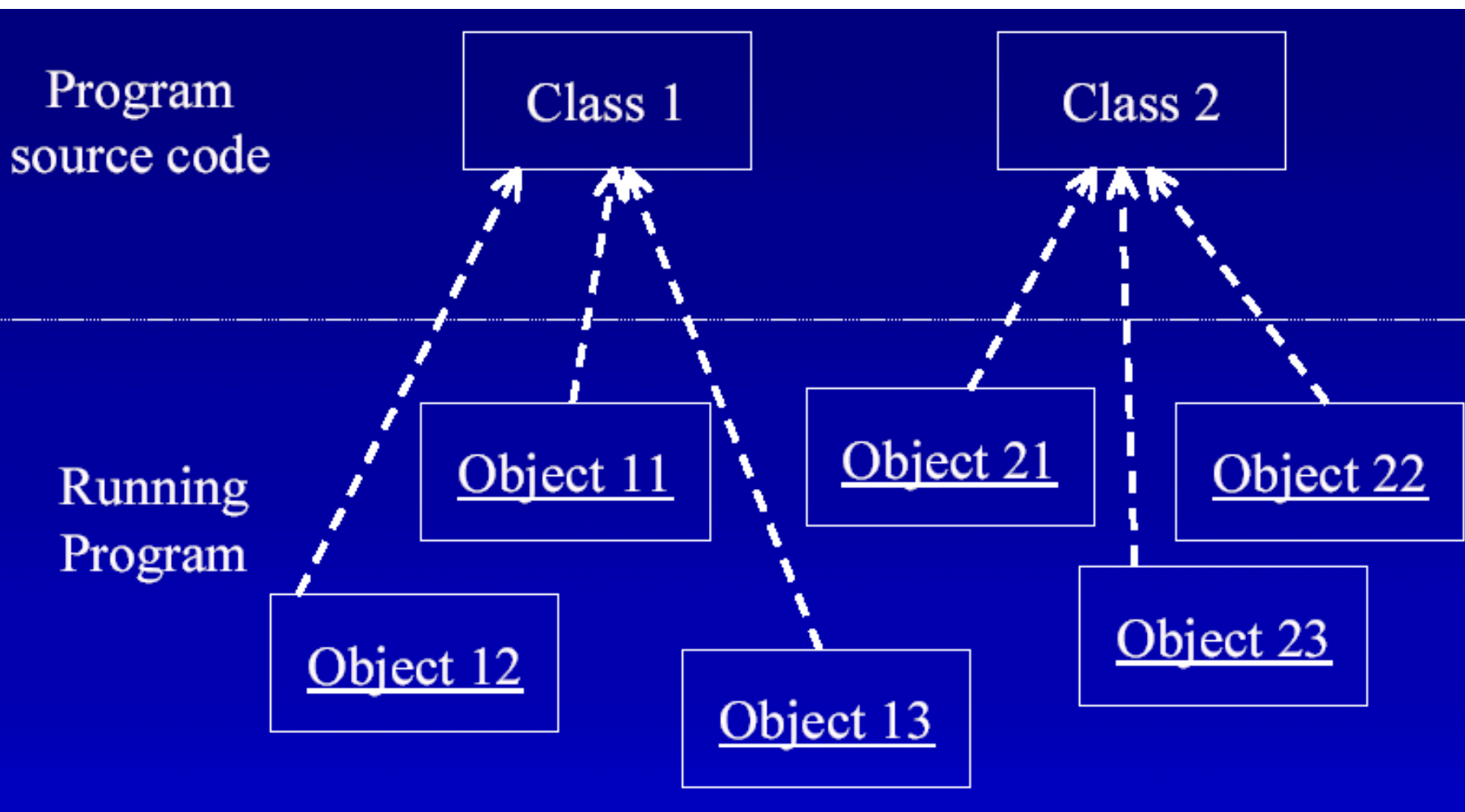


- **Class vs. Instance (object)**
- **Abstract**
- **Information Hiding**
- **Encapsulation**
- **Class vs. Type**
- **Kinds of Class Relationships**
- **Inheritance (a special association type)**
- **Static vs. Dynamic Binding**
- **Polymorphism**



- **A class represents a template for several objects and describes how these objects are structured internally. Objects of the same class have the same definition both for their operations and for their information structures**
- **An instance is an object created from a class. The class describes the (behavior and information) structure of the instance, while the current state of the instance is defined by the operations performed on the instance**





- 过程抽象：将整个系统的功能划分为若干部分，强调功能完成的过程和步骤
- 数据抽象：把系统中需要处理的数据和这些数据上的操作结合在一起，根据功能、性质、作用等因素抽象成不同的抽象数据类型



- 私有的实现细节被封装，只提供一个公共的访问接口

- 数据抽象

- 接口与实现相分离

- 模块化

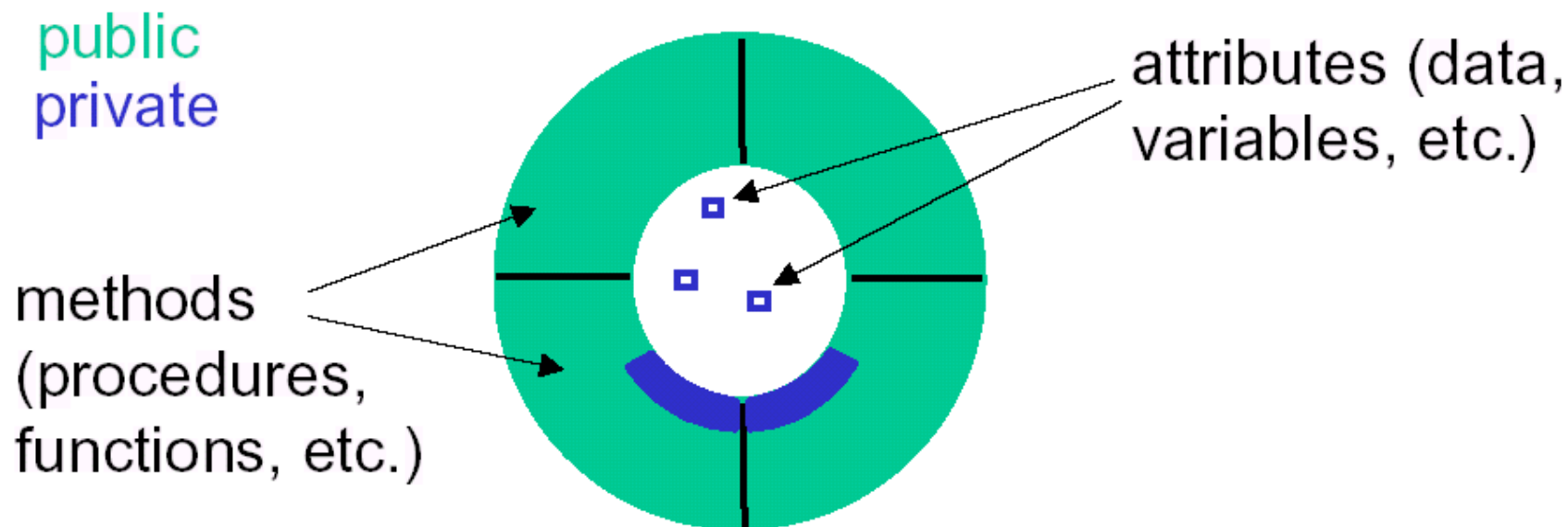
- 实现的变化不会影响到对该程序的调用，因为对外的接口没有受到影响

- 信息隐藏

- 程序可以调用该对象而不必关心该对象的具体实现



- 利用抽象数据类型将数据和基于数据的操作封装在一起
- 边界
- 接口



- 类与类型是不同的
 - 类是类型的一种特殊实现



- 三种关系

- 关联

- 聚合（包含） “*part of*”

- 花瓣是玫瑰花的一部分 *part-of*
 - 允许由已存在的类构造新的类

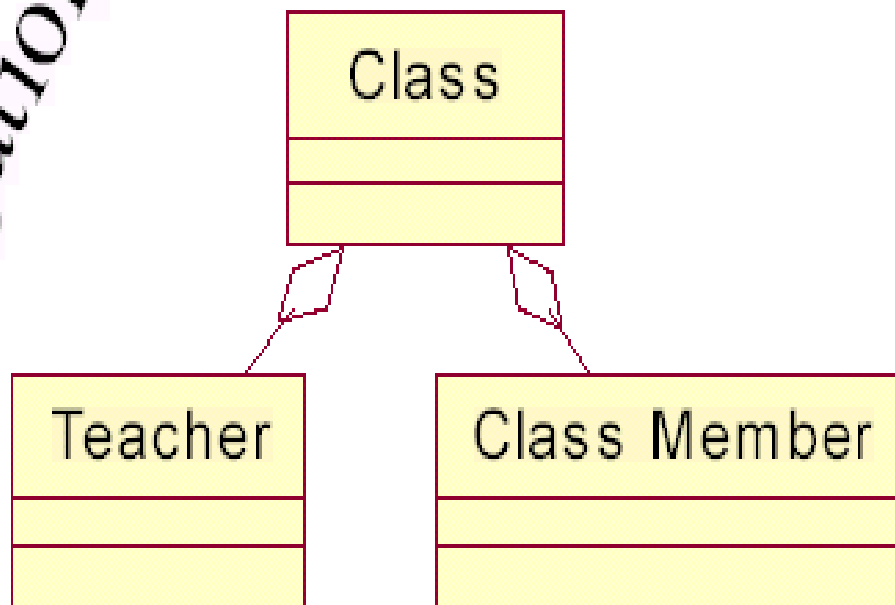
- 继承 “*kind of*”、 “*is a*”

- 玫瑰是花中的一种 *is a kind-of*
 - **Generalization** 提供创建子类的功能
 - 子类共享父类的结构



- “have”, “is related to”...

Aggregation

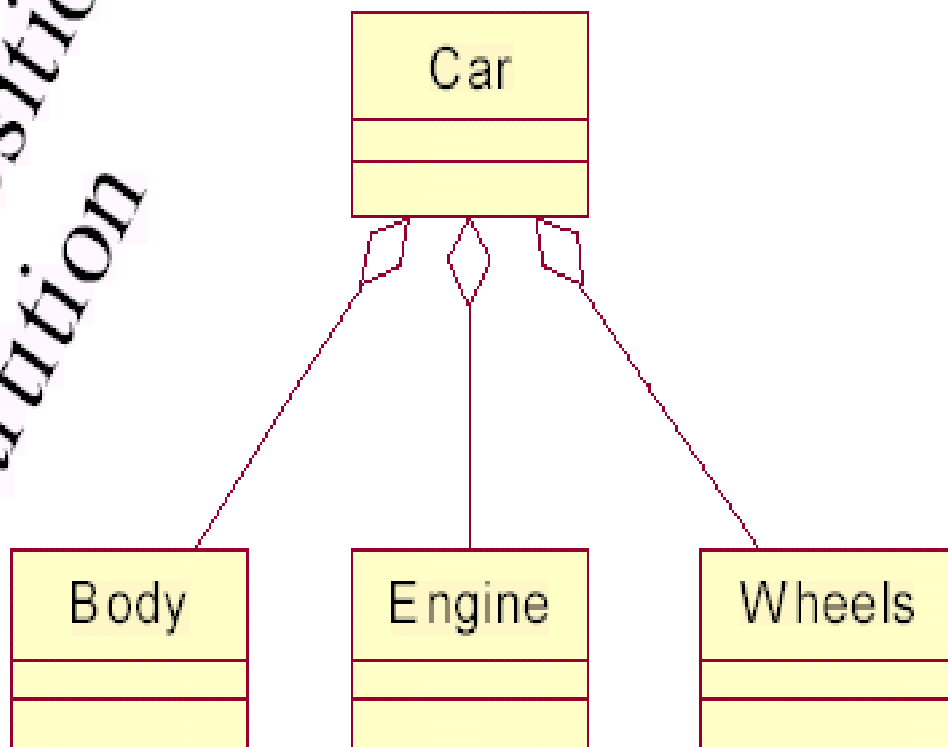


[返回](#)



- “is part of”, “is composed of”, ... (组成部分, 缺少则不能构成整体)

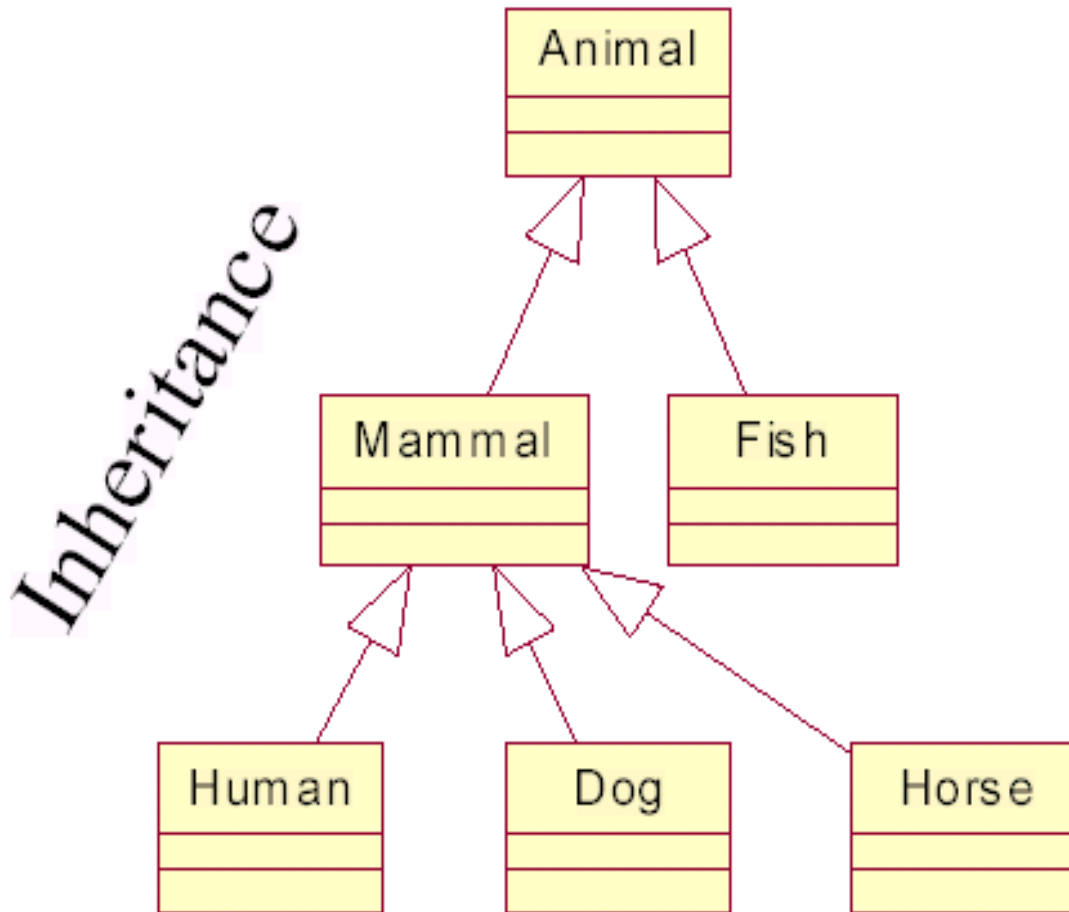
Composition
Partition



[返回](#)



- “is a” (IS-A), “a kind of” (AKO)



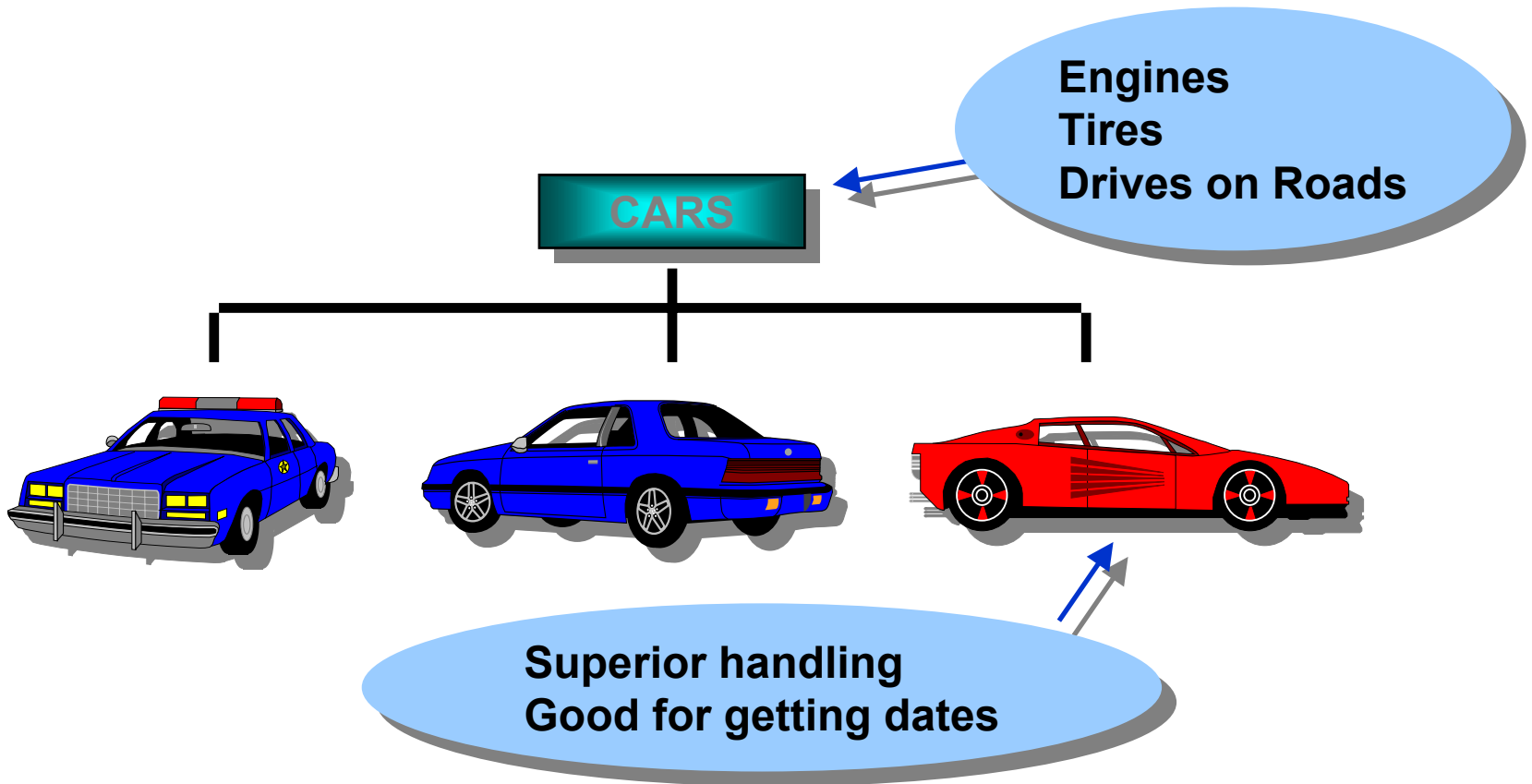
[返回](#)



- 在许多面向对象程序设计语言中用于描述 **generalization** 关系
- 继承表明一种关系，如果与一个（单继承）或多个（多继承）类共享结构或操作



- 对象被组织到一个类的层次结构中
- 通过特征与其他对象相关



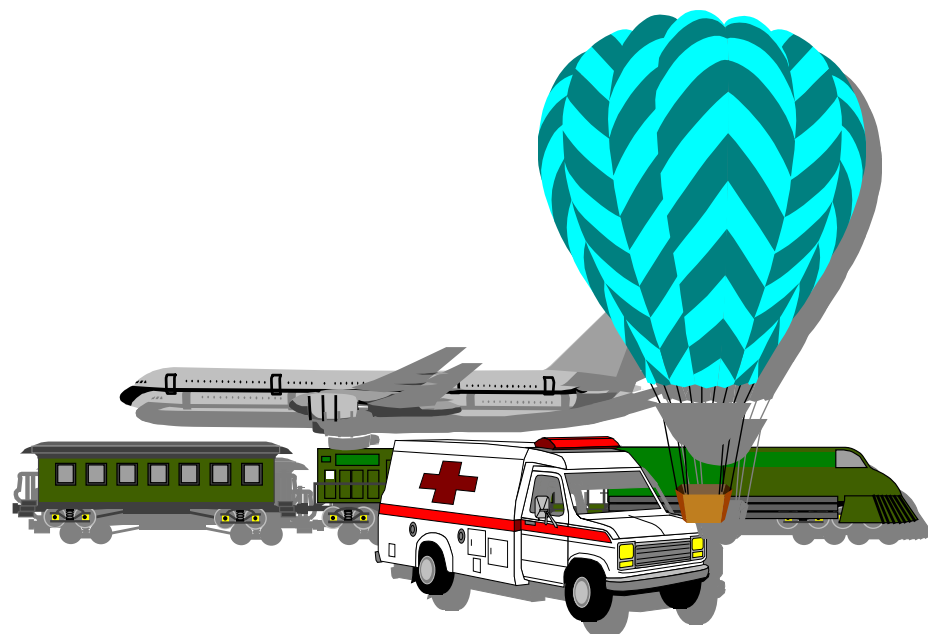
- **Generalize vs. Specialize**
- **后代 (child, sub-class) vs. 祖先 (parent, super-class)**
- **抽象类 vs. 具体类**
- **目的:**
 - 可重用
 - **Subtyping** (方法的兼容性, A'的后代可以代替A)
 - **Specialization** (overriding / redefinition, deletion, etc.)
 - **Conceptual**
- **多继承**



- 多态意味着“有多种形态”
- 多态是指程序中同名的不同方法共存的情况
- 多态有多种情况，可以通过子类对父类方法的覆盖实现多态，也可以利用重载在同一个类中定义多个同名的不同方法

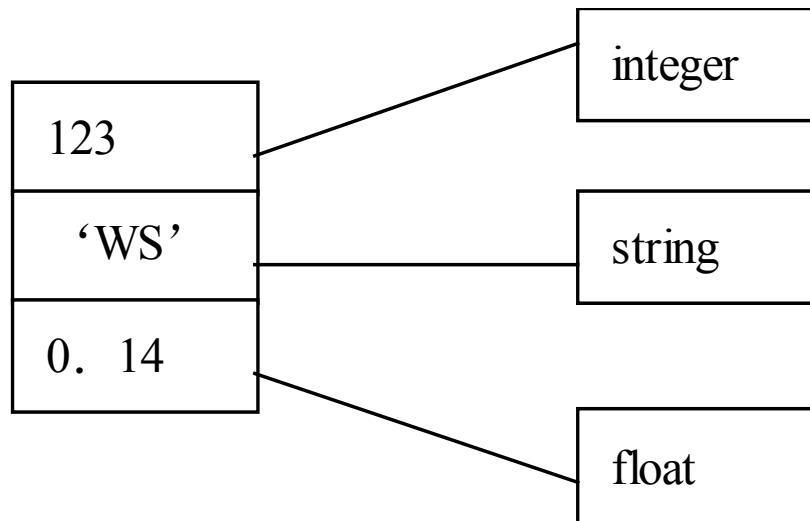
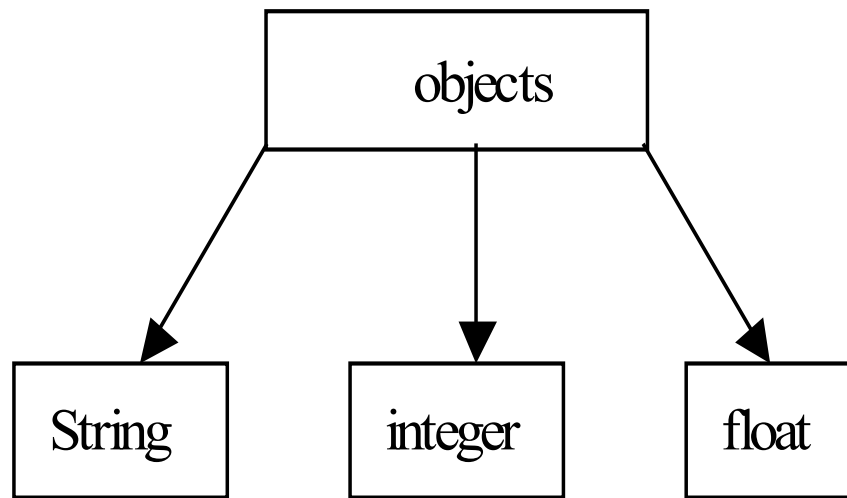


- 命令的执行将根据适当的环境采取相应的措施
- 启动命令
 - 汽车
 - 火车
 - 飞机
- 阅读命令
 - 录象带
 - 报纸上的文章
 - 程序源代码



- 命令的准确含义是与具体对象相关联
- 可以用一个简单的命令获得我们想要的多个不同对象的结果
- 重载**Overloading**/改写**Overriding**





- 面向对象程序设计是以数据为中心

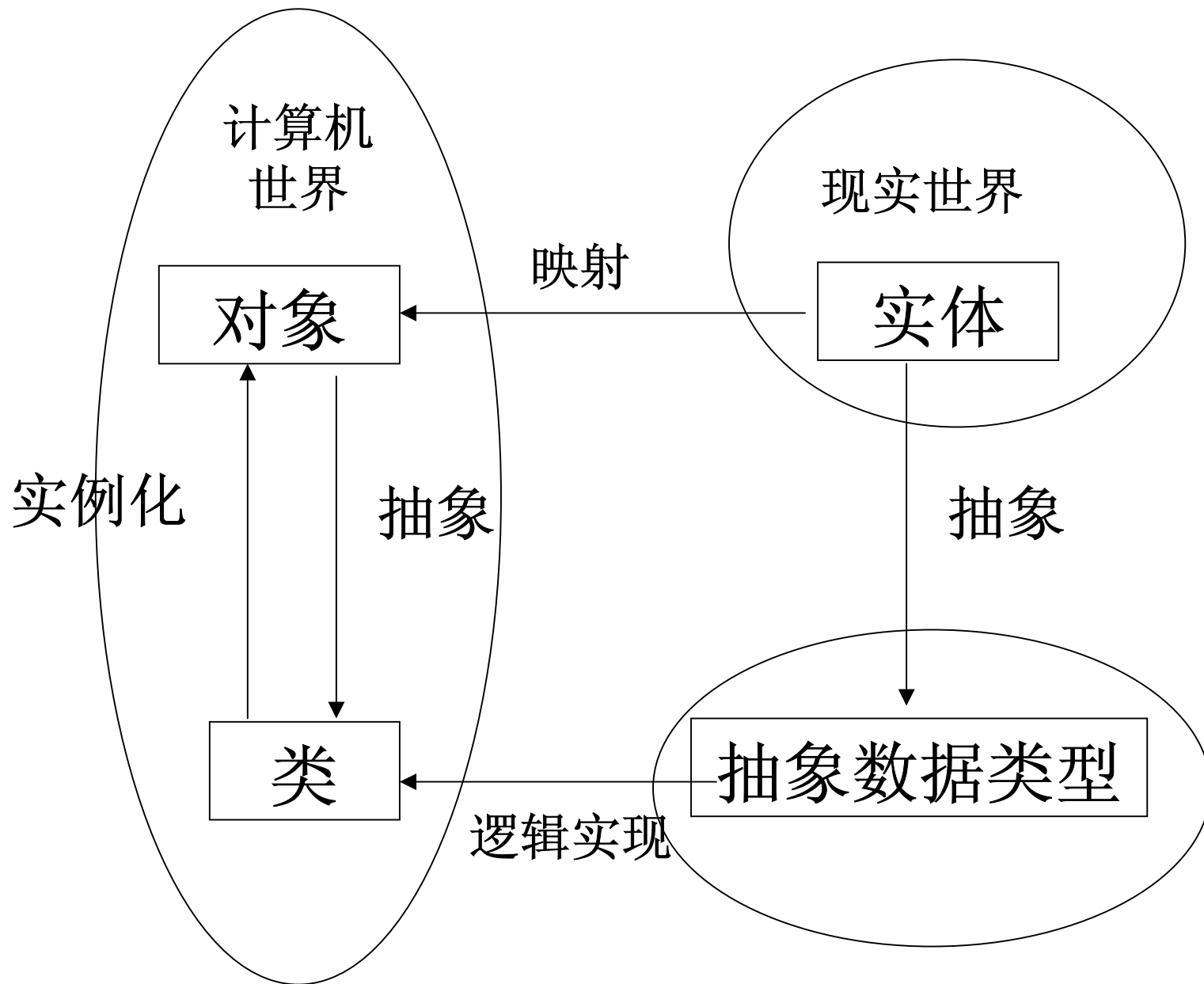
Programs = Objects / Classes / ADT + Inheritance

**Objects = Interfaces + Data (State) + Operations
(Behaviors) (*Objects as automata*)**

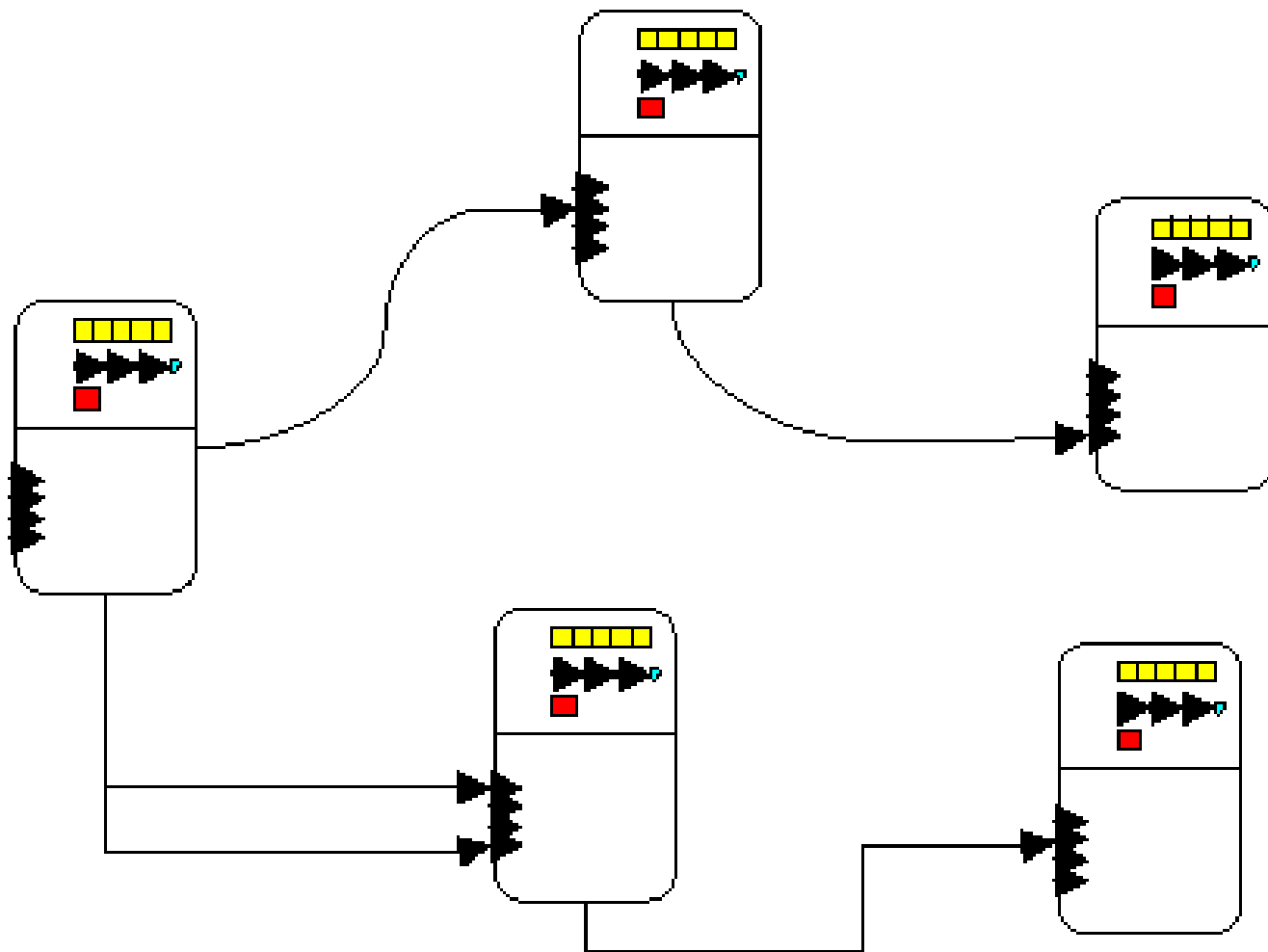
Messages

- 分类: **Class- / Prototype- based**
- 典型的语言: **C++, Java, Smalltalk, Eiffel...**



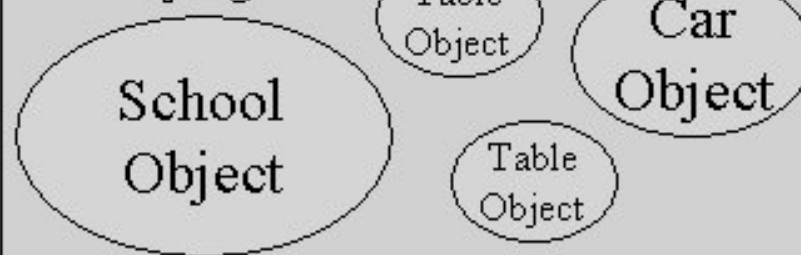


- 一个面向对象程序是一些对象的集合，对象之间通过定义的接口进行相互作用

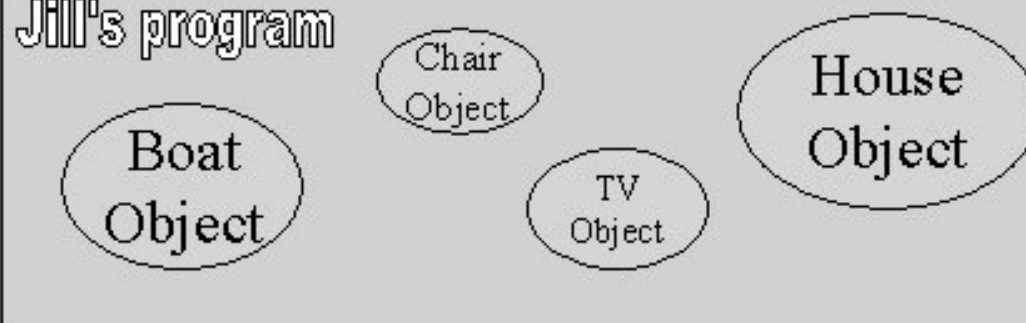


Computer memory

Jack's program



Jill's program



- 可重用性
- 可扩展性
- 可管理性

