

第1章 绪论

潘宇

E-mail: ypan@zju.edu.cn

面向对象程序设计(C++)



绪论

- 面向对象程序设计思想的由来
- 面向对象程序设计概述



1. 面向对象程序设计思想由来



Where are we?

- ☒ 程序设计方法的发展历程
- 评价软件质量的因素
- 过程式程序设计的局限性
- 从过程式转变到面向对象



1.1 程序设计方法的发展历程(1)

◆ 早期：面向机器的语言，用于科学计算。

例1: X86 ASM

```
00401010  push    ebp
00401011  mov     ebp,esp
00401013  sub     esp,40h
00401016  push    ebx
00401017  push    esi
00401018  push    edi
00401019  lea     edi,[ebp-40h]
0040101C  mov     ecx,10h
00401021  mov     eax,0CCCCCCCCh
00401026  rep stos dword ptr [edi]
```

00401028	push	offset string "hello world" (0042501c)
0040102D	mov	ecx,offset cout (00428bc0)
00401032	call	ostream::operator<< (0040b5a0)
00401037	xor	eax, eax
00401039	pop	edi
0040103A	pop	esi
0040103B	pop	ebx
0040103C	add	esp, 40h
0040103F	cmp	ebp, esp
00401041	call	__chkesp (00401060)
00401046	mov	esp, ebp
00401048	pop	ebp
00401049	ret	

缺点:

- 极难掌握，软件规模小
- 对非数字运算难以胜任



1.1 程序设计方法的发展历程(2)

- ◆ **60年代**：面向过程的“技巧式”程序设计
(如：早期的**fortran, Cobol, Algol 60**)

例2: algol 60

```
BEGIN
  FILE F (KIND=REMOTE);
  EBCDIC ARRAY E [0:11];
  REPLACE E BY "HELLO WORLD!";
  WHILE TRUE DO
    BEGIN
      WRITE (F, *, E);
    END;
  END.
```


例3: COBOL

```
IDENTIFICATION DIVISION.  
  PROGRAM-ID.  HELLO-WORLD.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.  
  DISPLAY "Hello, world!".  
STOP RUN.
```

缺点:

- 程序控制复杂,
- 过于依赖程序员的经验和技巧,
- 难读、难改、难移植



1.1 程序设计方法的发展历程(3)

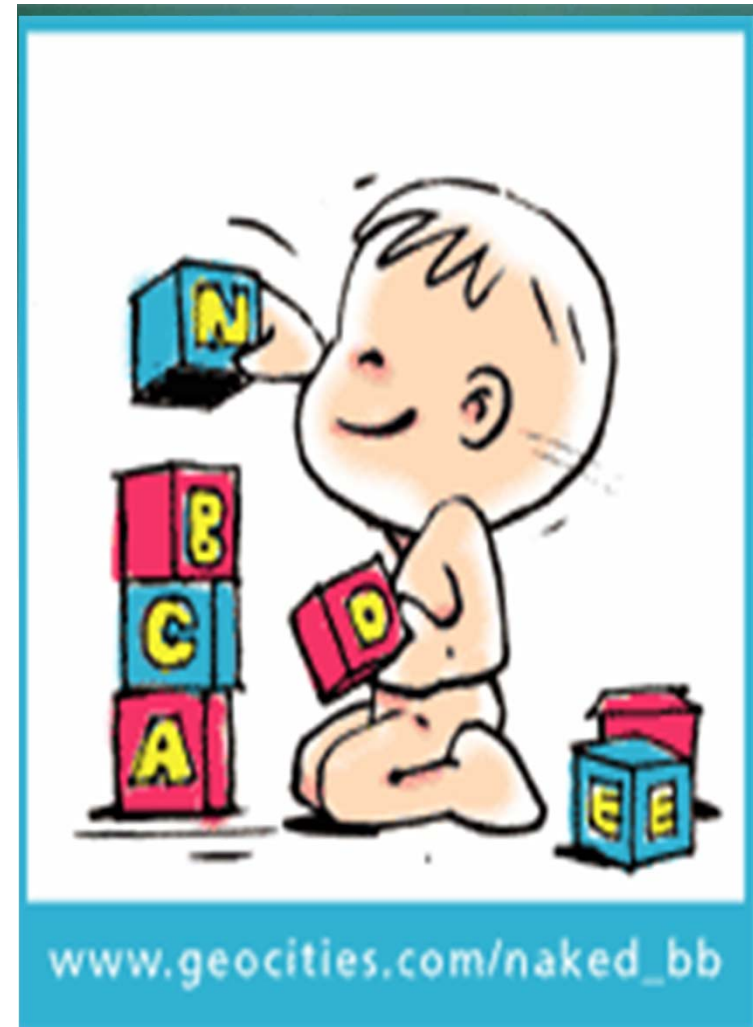
- ◆ **70年代早期：结构化的过程式程序设计**
(如： **C,Pascal**)

例4: C语言

```
#include <stdio.h>
int max(int,int);    // max模块
int main(void) {
    int i;
    i=max(10,20);
    printf("Hello, world!\n");
    return 0;
}
```

1.1 程序设计方法的发展历程(3续)

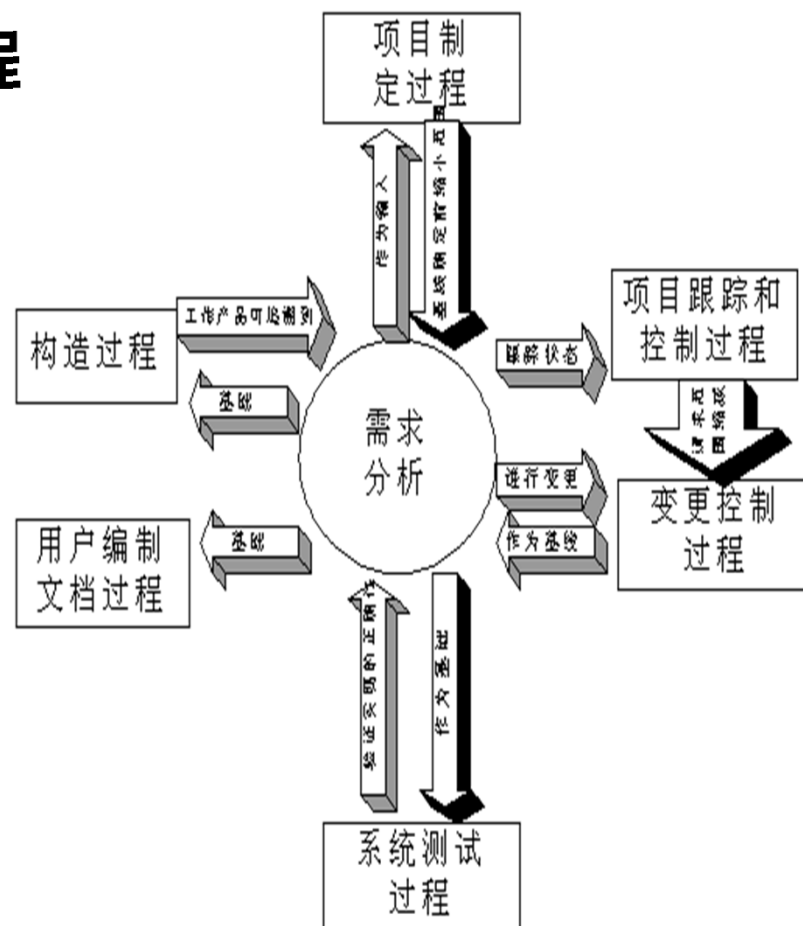
- 分解原则、模块独立原则、编码结构化原则
- 最大的软件：385万行
(美国导弹预警系统)
- 大型软件难以维护、难以修改和移植。



1.1 程序设计方法的发展历程(4)

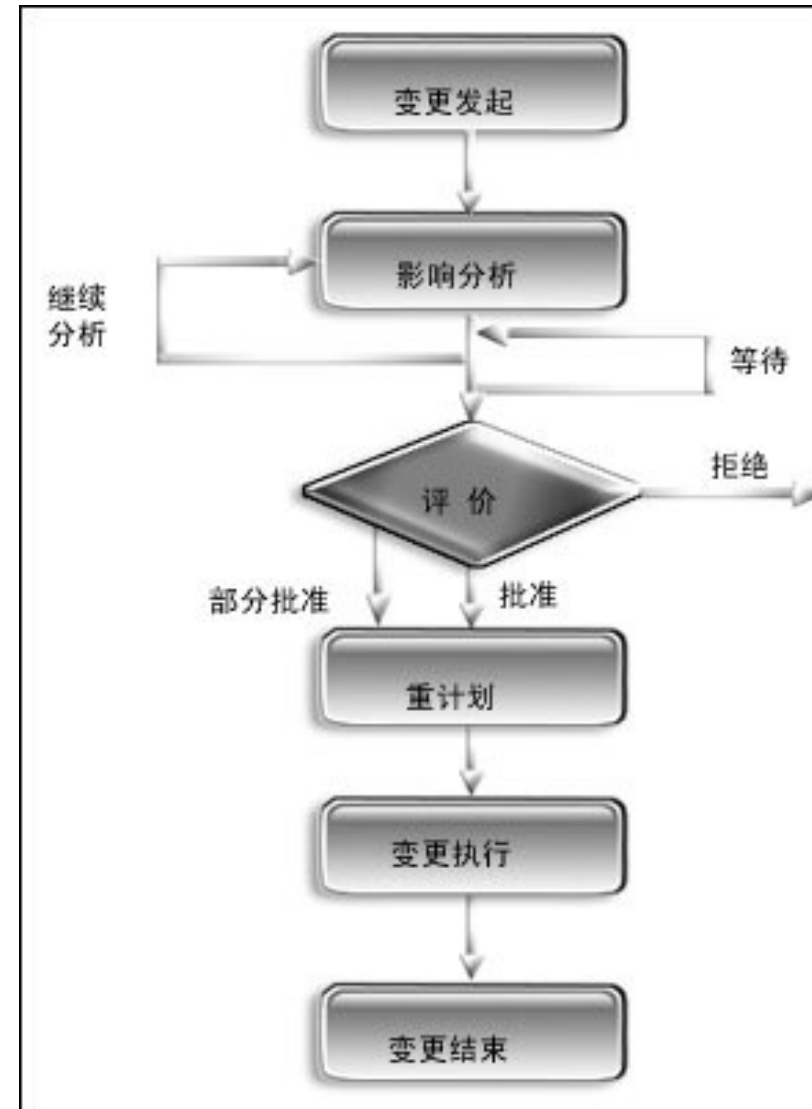
◆ 70年代中后期：软件工程

- 有成套的管理方法
- 强调模块性、抽象性、易维护、可修改、可移植
- 软件工程原则：将软件“做什么”和“怎么做”分离
- 最大的软件：4000万行（美国航天飞机监控系统）



**“需求分析”处于软件
工程的核心地位。一旦需
求发生变更，工作量是巨
大的！**

**因此,描述需求的方法
是关键!!**





现实情况怎么样?

一方面，各种程序设计方法学被提出来

👉 **大得可怕,却并不好用!**

✌️ **看上去好像我们要花绝大多数的时间写文档,
以至于没有时间编写程序.....**

另一方面，软件危机没有得到根本上的解决

👁️ **超过50%的大型软件项目以失败告终!**

Why ???



Where are we?

- 程序设计方法的发展历程
- ☒ 如何评价软件的质量?
- ☒ 过程式程序设计的局限性
- 从过程式转变到面向对象



1.2.1 如何评价软件的质量？

- 软件的内部质量

- 正确性

- 健壮性

- 可扩充性

- 可复用性

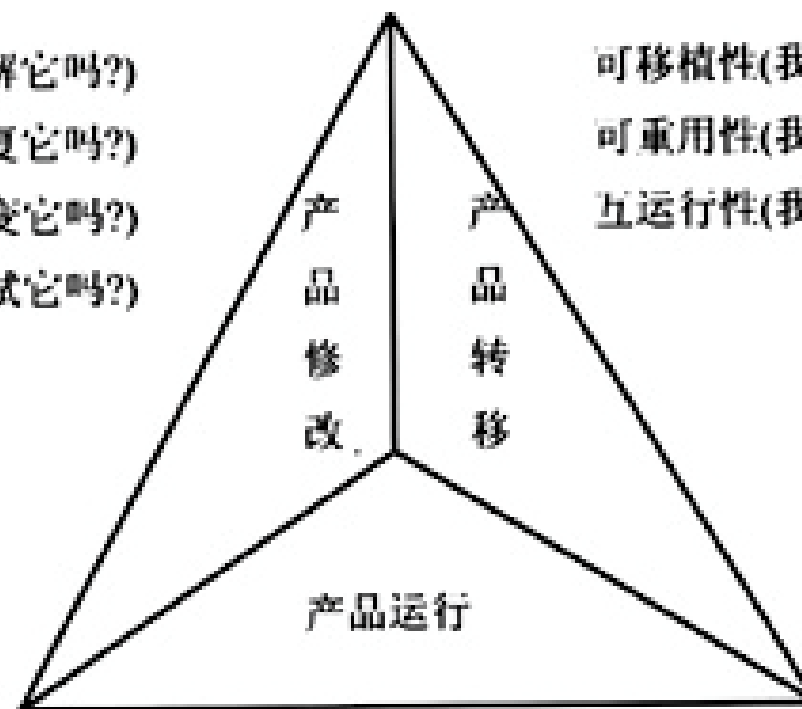
- 可读性

- 可维护性

- 其他（兼容性，效率，可移植性...）

☞ 事实上，这些方面正是结构化程序设计和软件工程追求的目标。

可理解性(我能理解它吗?)
可维修性(我能修复它吗?)
灵活性 (我能改变它吗?)
可测试性(我能测试它吗?)



可移植性(我能在另一台机器上使用它吗?)
可重用性(我能再用它的某些部分吗?)
互运行性(我能把它和另一个系统结合吗?)

正确性(它按我的需要工作吗?)
健壮性(对意外环境它能适当地响应吗?)
效率(完成预定功能时它需要的计算机资源多吗?)
完整性(它是安全的吗?)
可用性(我能使用它吗?)
风险性(能按预定计划完成它吗?)



1.2.2 过程式程序设计本身的局限性

过程式程序设计是一种以功能为中心、基于功能分解的程序设计方法。

一个过程式程序由一些子程序构成，每个子程序对应一个子功能，它实现了功能抽象。子程序描述了一系列的操作，它是操作的封装体。

过程式程序的执行过程体现为一系列的子程序调用。在过程式程序中，数据处于附属地位，它独立于子程序，在子程序调用时作为参数传给子程序使用。

下面的经典公式刻划了过程式程序设计的本质特征：

程序 = 算法 + 数据结构



例：自我介绍, whoami

定义一个**结构体**
PERSON,包含成员
name;生成一个**结构体**
变量,调用**函数**设置他
的姓名;调用**函数**
whoami打印他的姓名.

```
struct Person{  
    char name[20];  
    ...;  
};
```

```
void whoami(struct Person);  
void main(){  
    struct Person mike;  
    strcpy(mike.name, "mike");  
    whoami(mike);  
}  
  
void whoami(struct Person p)  
{  
    printf("%s", p.name);  
}
```



1.2.2 过程式程序设计本身的局限性(续)

程序员只能按过程式程序设计的逻辑结构(而不是按问题本身的逻辑结构)去描述问题;因此,问题的求解实际上是一种过程的抽象。

也就是说,程序员必须在实际问题模型(问题空间)和机器模型(解空间)之间进行转换。

然而,这种转换并不是线性的。因此,当实际问题发生改变时,程序中各种成分也随之改变。其中:

- 。 功能: 很容易变
- 。 过程执行顺序: 很容易变
- 。 接口: 极容易变
- 。 数据: 极容易变



Where are we?

- 程序设计方法的发展
- 评价软件质量的因素
- 结构化程序设计的局限性
- ☒ 从过程式转变到面向对象



1.3 从过程式转变到面向对象

- 结构体+过程（易变）→对象（稳定）
对象：把问题空间中的事物在解空间中的表示称为“对象”。
- 基于过程→基于对象
- 按计算机的结构建模→对问题本身建模

也就是说：

当我们读描述解决方案的代码时，也就是在读表达该问题的文字！

OOP允许程序员用问题本身的术语来描述问题，而不是用(要运行解决方案的)计算机的术语来描述问题。

每个对象看上去就象一台小计算机，它有状态，有可以执行的运算。



1.3.1 例：自我介绍, whoami

OOP(C++):

- 定义一个**类**PERSON, 包含**属性**name和**行为**whoami;生成一个名为mike的**对象**(mike);mike调用**自己**的行为whoami**说出**他的名字.

```
class PERSON{
public:
    char name[20];
    void
    whoami(){cout<<name;};
    ...; //
};

void main(){
    PERSON mike("mike");
    mike.whoami();
}
```




1.3.2 面向对象程序设计的编程思想

对象式程序设计是一种以对象为中心、基于数据抽象的程序设计方法。

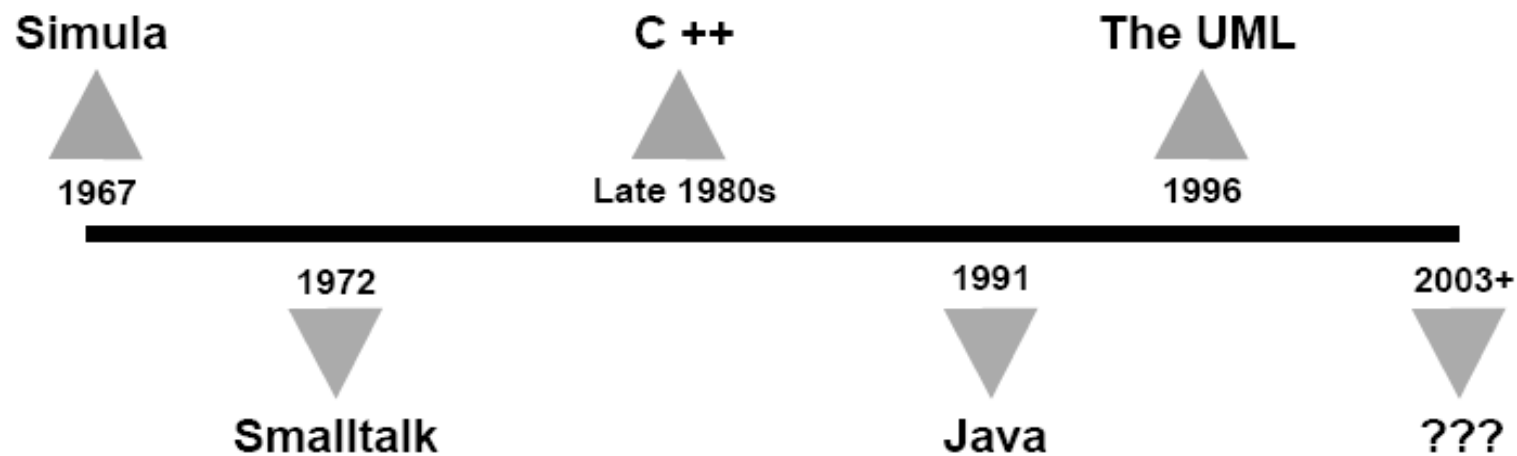
对象式程序设计通常称为面向对象程序设计。一个面向对象程序由一些对象构成，对象是由一些数据及可施于这些数据上的操作所构成的封装体。对象的特征由相应的类来描述，一个类可以从其它的类继承。

面向对象程序的执行过程体现为各个对象之间相互发送和处理消息。面向对象程序可简单地表示成下面的公式：

$$\text{程序} = \text{对象/类} + \text{对象/类} + \dots$$
$$\text{对象/类} = \text{数据} + \text{操作}$$



1.3.3 对象技术(OT)的年代表





1.4 对象技术(OT)的年代表(续)

Simula:

Smalltalk

C++: 在兼容原有C语言的基础上, 进一步加入支持面向对象技术的要素, 如数据抽象、继承、多态等。

Java:

UML:



2. 面向对象程序设计概述



Where are we?

- ☒ 什么是对象
- 什么是类
- **OO**的四个基本原理
- 多态性和泛化



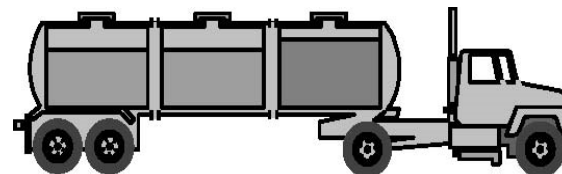
2.1 什么是对象？

- 对象的定义
- 对象的标识
- 对象的状态
- 对象的操作

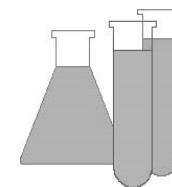
2.1.1 对象的非正式定义

非正式地，一个对象用来描述一个实体(entity)，这个实体可以使物理实体、概念上的实体或者是软件。

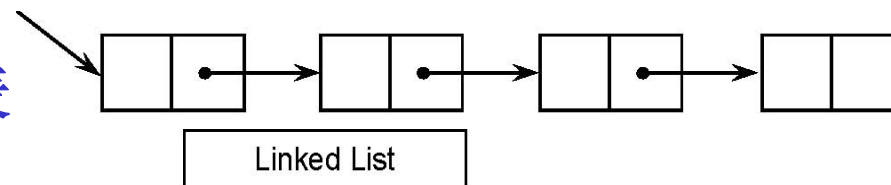
physical entity: 卡车



Conceptual entity : 化学反应过程



Software entity : 链表

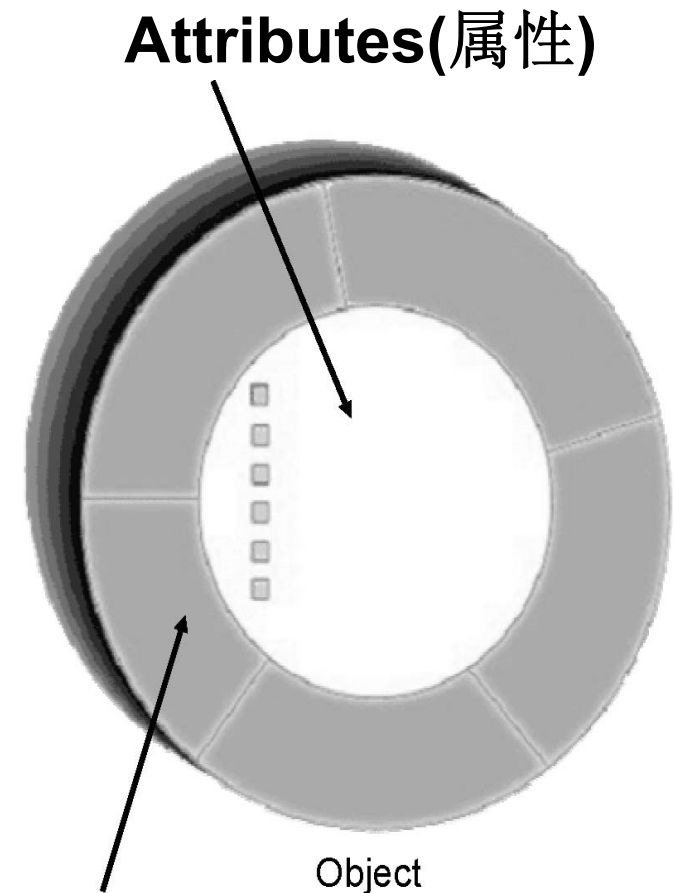


2.1.2 对象的一个更正式的定义

对象，是现实世界中某个实体在计算机逻辑中的映射和描述。

对象具有标识(**identity**)，在一个明确的边界(**boundary**)里封装了状态(**state**)和行为(**behavior**)。

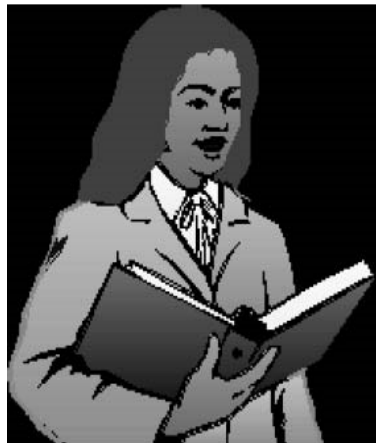
- State is represented by attributes and relationships.
- Behavior is represented by operations, methods, and state machines.



Operations(操作)

2.1.3 对象的标识(Identity)

每一个对象都有一个唯一的标识，即使它的状态跟其他对象相同。



**Professor “J Clark”
teaches Biology**

Identity: ProfJClark_f



**Professor “J Clark”
teaches Biology**

ProfJClark_m

2.1.4 对象的状态(state)

状态(state)描述的是对象在不同时刻所表现出来的一种状况(condition)或情形(situation): 例如, 满足某种条件, 在做某件事情或者等待某个事件的发生。

通常对象的状态随时间的变化而改变。



Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured (终身职位)
Discipline(学科): Finance(金融学)
Maximum Course Load(最大课程负担): 3 classes



Professor Clark

2.1.5 对象的行为(Behavior)

**对象的行为确定了一个对象如何行动(acts),
如何对事件进行响应(reacts)。**



Professor Clark's behavior

- **Submit Final Grades**
- **Accept Course Offering**
- **Take Sabbatical(休假年)**



Where are we?

- 什么是对象?
- ☒ 什么是类
- **OO**的四个基本原理
- 多态性和泛化



2.2 什么是类？

- **类的定义**
- **类的属性**
- **类的操作**
- **举例**
- **类和对象的关系**

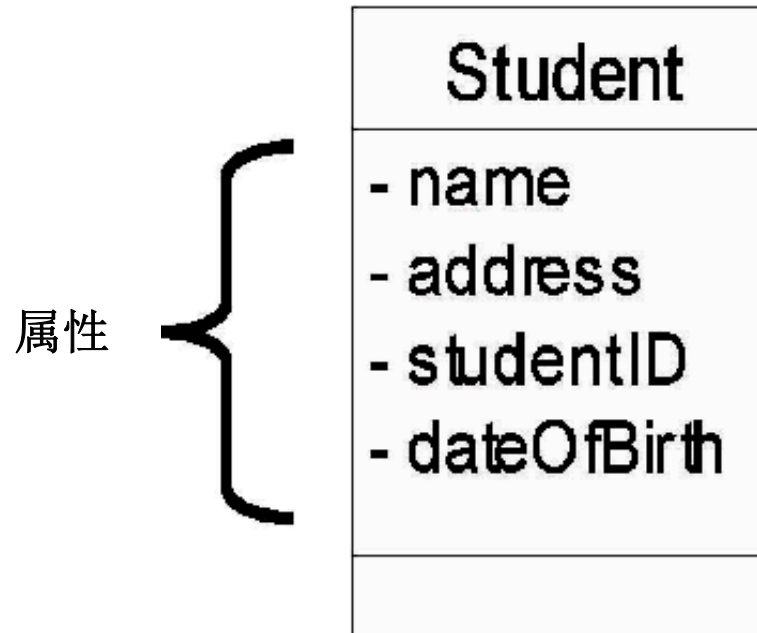


2.2 类(Class)

- A **class**(类) is a description of **a set of objects** that **share** the same *attributes*(属性), *operations*(操作), *relationships*(关系), and *semantics*(语义).
 - An object is an instance(实例) of a class.
- A class is an abstraction(抽象) in that it
 - Emphasizes relevant characteristics.
(强调相关的本质特征)
 - Suppresses other characteristics.
(舍去其他的无关特征)

2.2.1 类的属性(Attributes)

- An attribute is a **named property** of a class that describes **the range of values** that instances of the property may hold.
 - ✦ A class may have any number of attributes or no attributes at all.

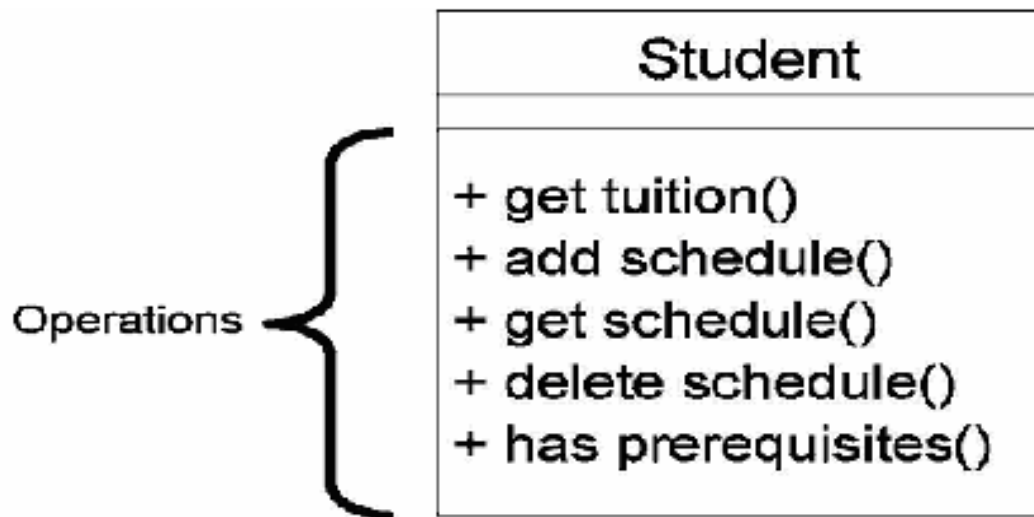


2.2.2 类的操作(Operation)

- **A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.**

(操作是可以被另一个对象请求执行而实现某种行为的服务。一个操作有一个签名，该签名可以对实际参数进行约束。)

- **A class may have any number of operations or none at all.**

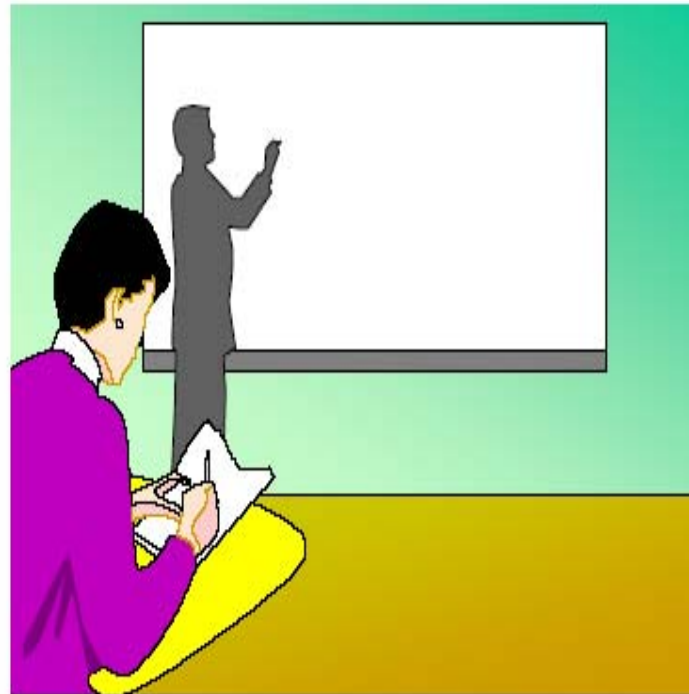


2.2.3 类course

Class Course

Properties

Name
Location
Days offered
Credit hours
Start time
End time

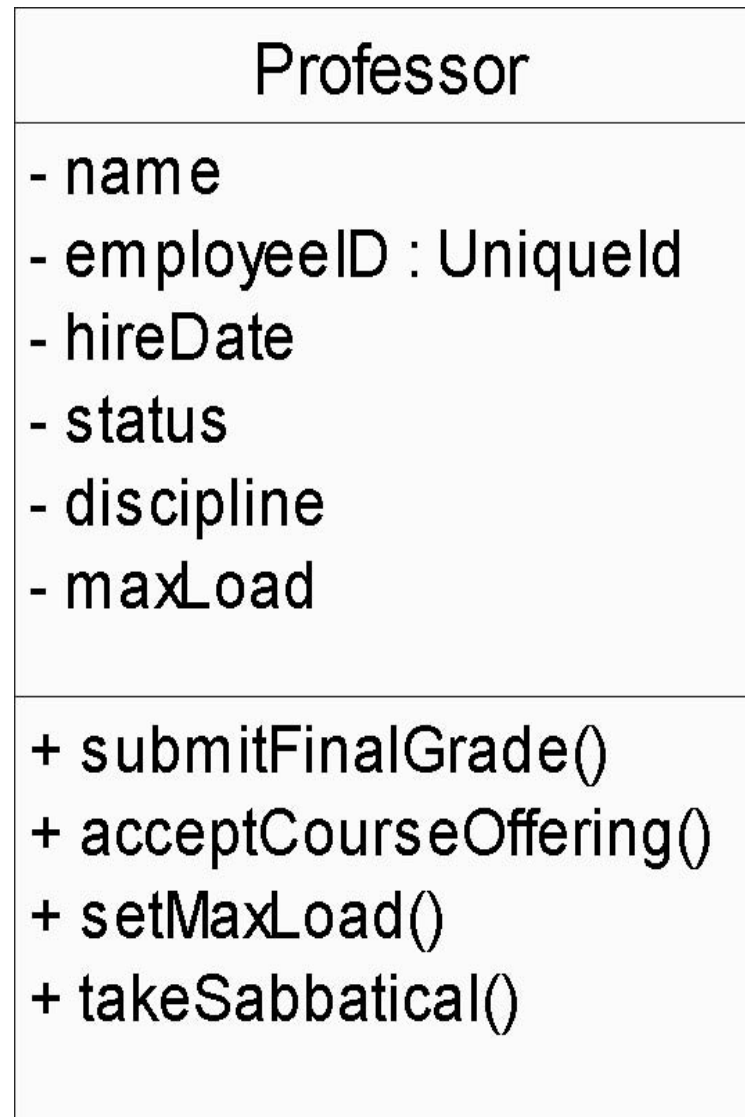


Behavior

Add a student
Delete a student
Get course roster (花名册)
Determine if it is full



2.2.4 类的UML表示

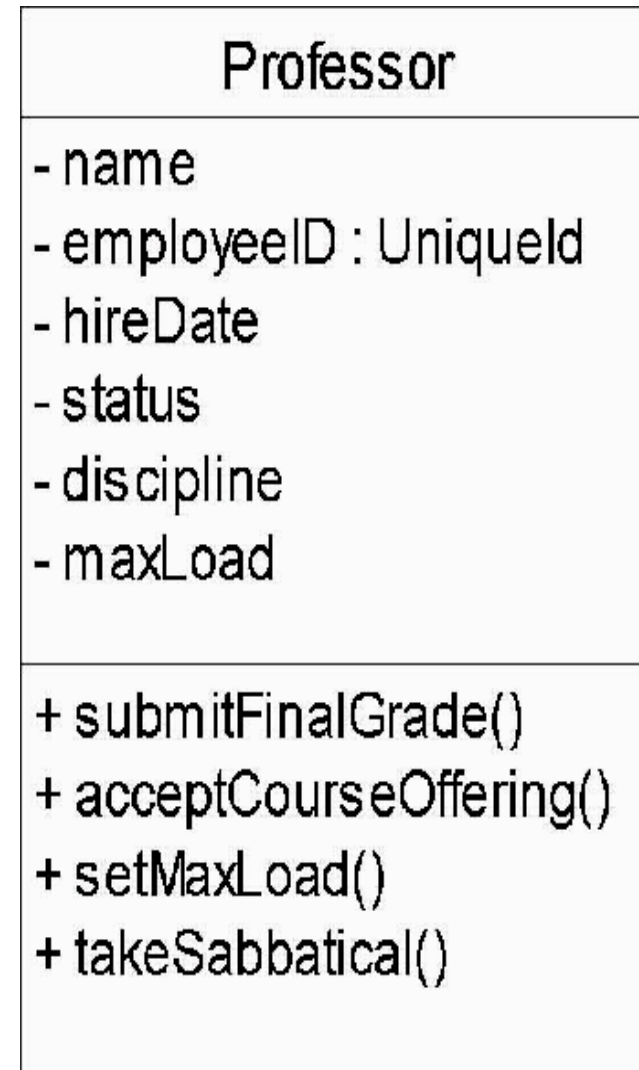




2.2.5 类的UML表示的组成部分

- **A class has three sections:**

- ➔ **The class name (类名)**
- ➔ **The structure (属性)**
- ➔ **The behavior (操作)**

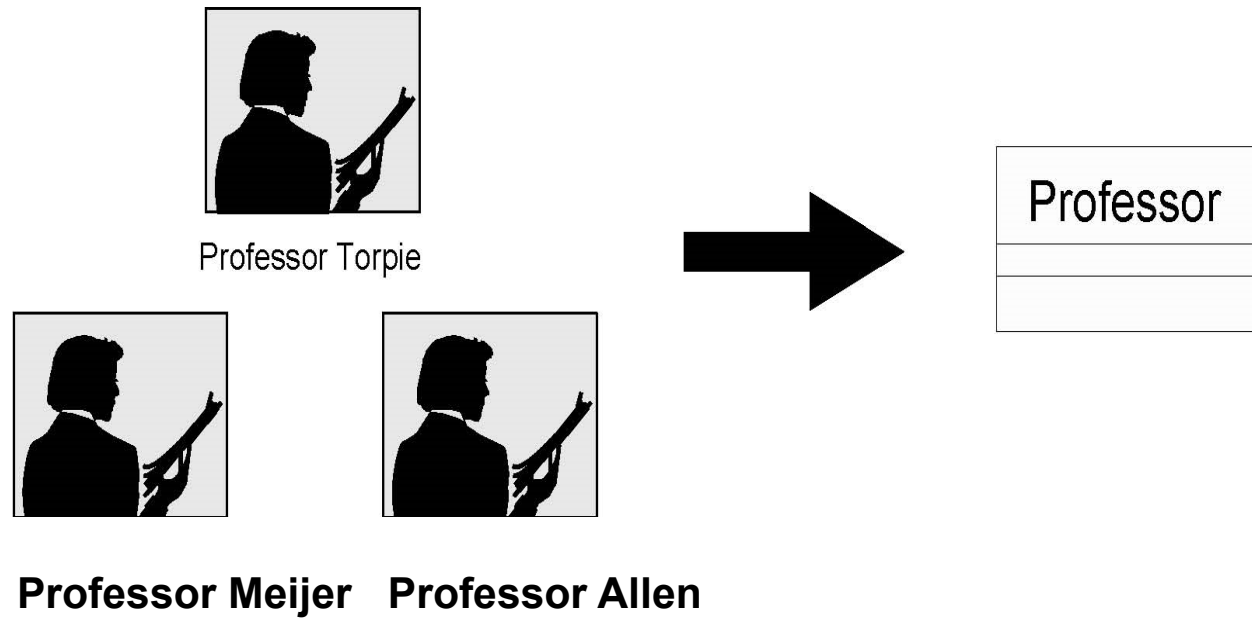




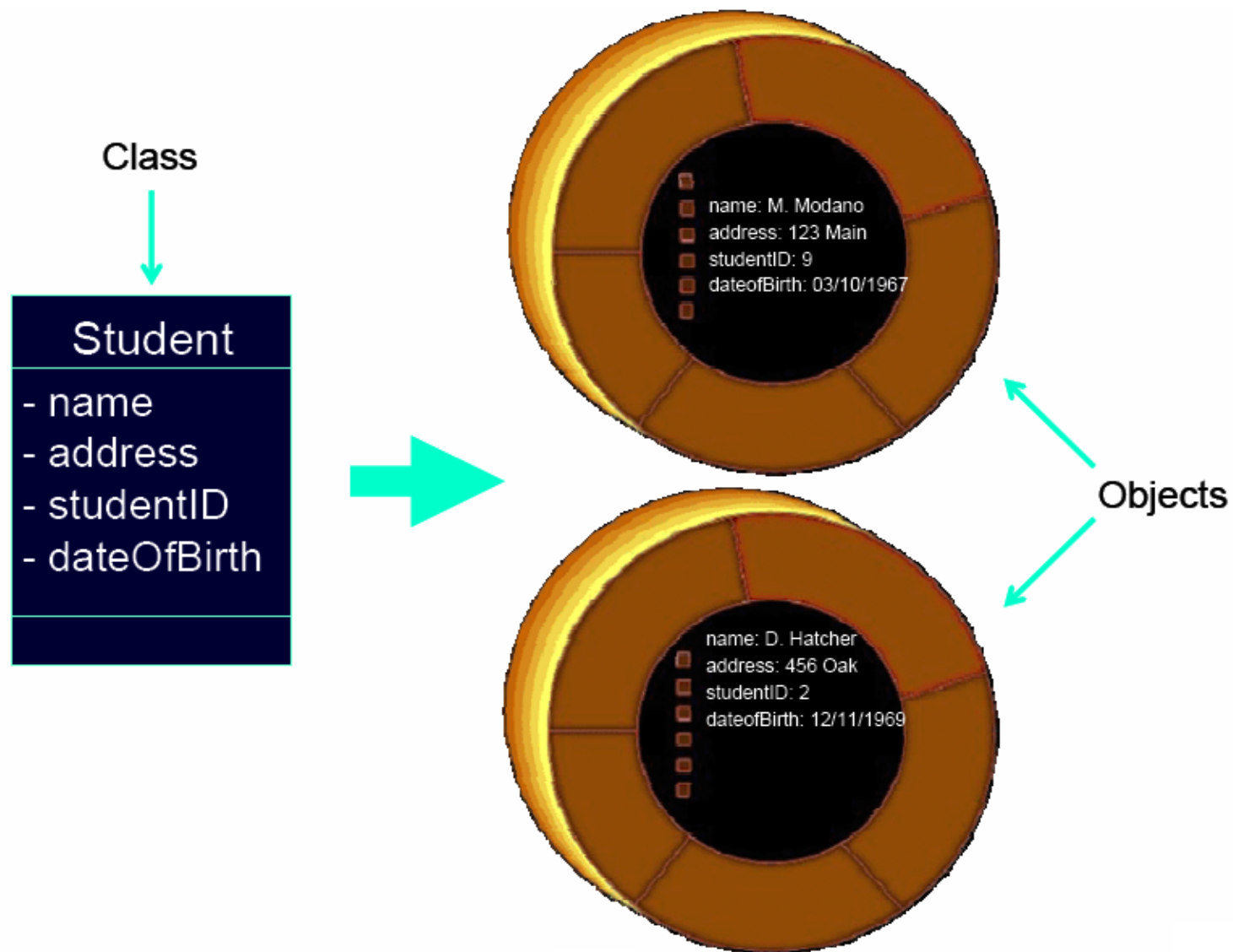
2.2.6 Objects 和Classes的关系

- **A class is an abstract definition of an object.**
(类是对象的抽象定义)
 - ⊕ **It defines the structure and behavior of each object in the class.**
(定义类中每一个对象的结构和行为)
 - ⊕ **It serves as a template for creating objects.**
(类是创建对象的模板)
- **Classes are not collections of objects.**

2.2.6 Objects 和Classes的关系(例)



2.2.6.1 类和对象中的属性



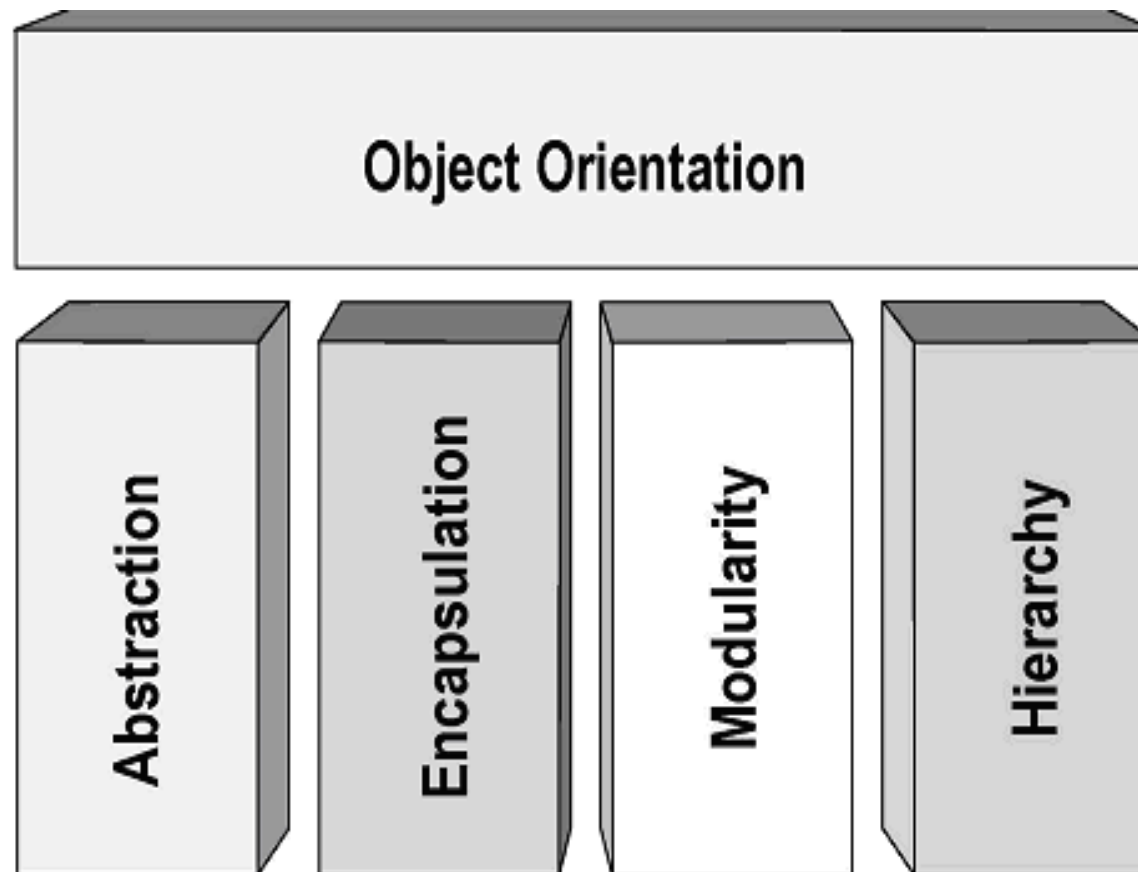


Where are we?

- 什么是对象？
- 什么是类？
- ☒ OO的四个基本原理
- 多态性和泛化



2.3 面向对象的四个基本原理





2.3.1 什么是抽象(**Abstraction**)



2.3.1 什么是抽象(Abstraction)

- The **essential characteristics** of an entity that distinguishes it from all other kinds of entities.
(区别于其他实体的**本质特征**)
- Defines a **boundary** relative to the perspective of the viewer.
(定义一个相对于观察者的角度的**边界**)
- Is not a concrete manifestation, denotes the ideal essence of something.
(不是一种有形的表现，表示的是事物的本质)

2.3.1.1 例：抽象

Example: Abstraction



Student



Professor



**Course Offering (9:00 a.m.,
Monday-Wednesday-Friday)**



Course (e.g. Algebra)



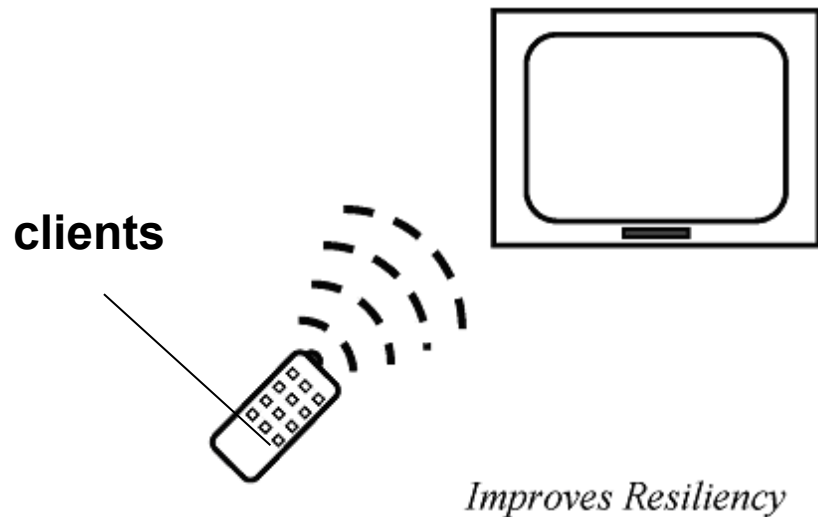
2.3.2 什么是封装(**Abstraction**)

2.3.2 什么是封装(Encapsulation)

- **Hides implementation from clients.**

(封装把具体实现隐藏起来，对客户使用者不可见。增加了实现的“弹性” --resiliency)

➤ **Clients depend on *interface*(接口).**





2.3.2.1 对象的接口(Interface)

对象的接口(Interface)规定我们能向特定的对象发出什么请求，而具体的实现(属性)则被隐藏起来了。

```
Class TV{  
public:  
    void turn_on(){ };  
    void turn_off(){ };  
    void volume_up(){ };  
    void volume_down(){ };  
    void change_channel(int n){ };  
private:  
    .....; // 不能使用  
};
```



2.3.2.2 封装的本质：隐藏实现

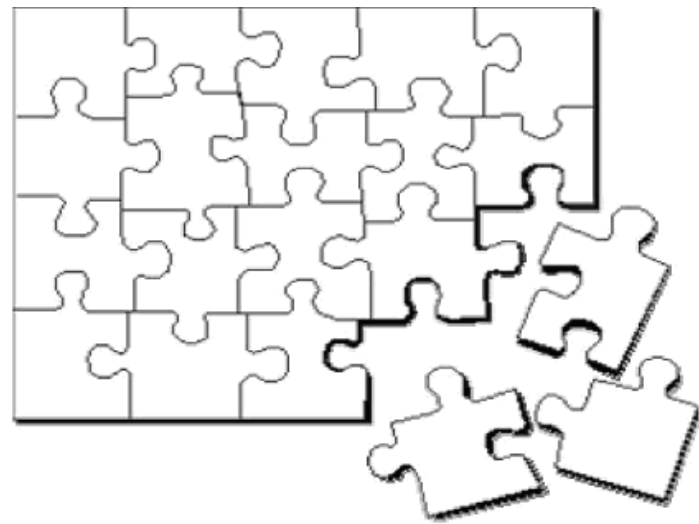
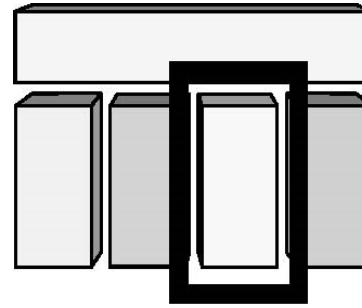
- **通过封装，使得程序员分为：**
 - **类创建者：专注于实现类**
 - **客户程序员：使用类的用户，专注于使用类**
- **优点：**
 - **避免客户程序员插手他们不应当接触的部分。**
(通过public, private,protected控制客户的访问)
 - **实现者可以随时修改被隐藏的内部工作方式，而不影响客户程序员(的代码)。**



2.3.3 什么是模块性(Modularity)

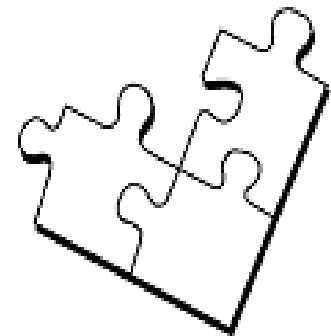
2.3.3 什么是模块性(Modularity)

- **Breaks up something complex into manageable pieces.**
(把一个复杂的事物分成易于处理的碎片)
- **Helps people understand complex systems.**
(帮助人们理解复杂的系统)



例：模块性—大问题分解成小问题

- ♦ For example, break complex systems into smaller modules.



**Course Registration
System**

(选课系统)



**Billing
System**

(缴费系统)



**Course
Catalog
System**

(课程编目系统)



**Student
Management
System**

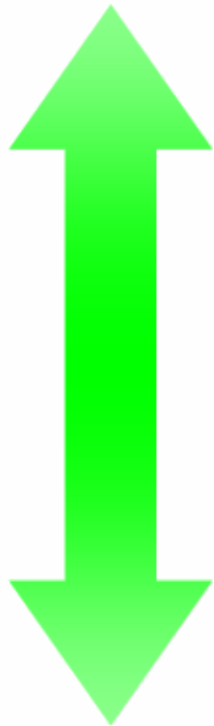
(学生管理系统)



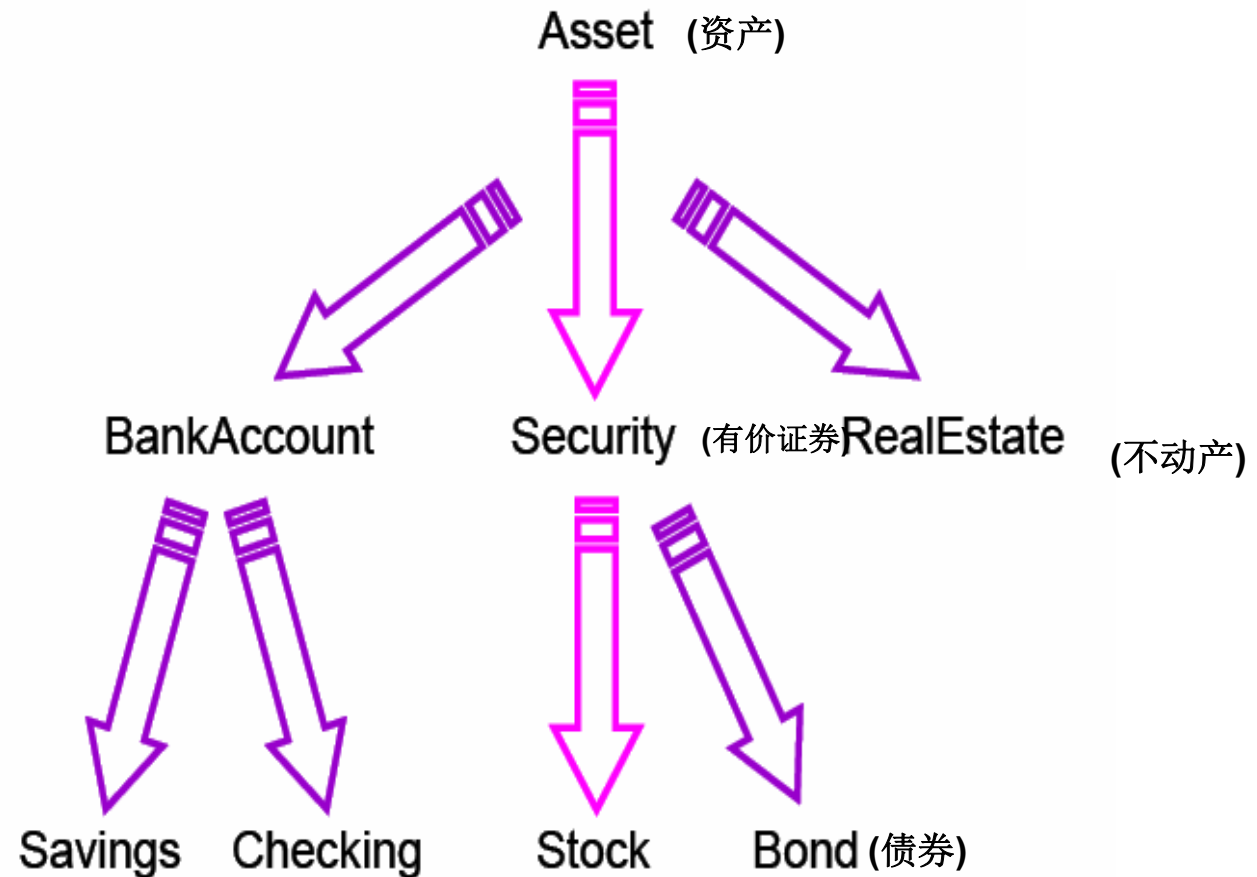
2.3.4 什么是层次(Hierarchy)

2.3.4 什么是层次(Hierarchy)

提高抽象程度



降低抽象程度



处在层次中同一层(level)的元素，也应该处在抽象的同一层。

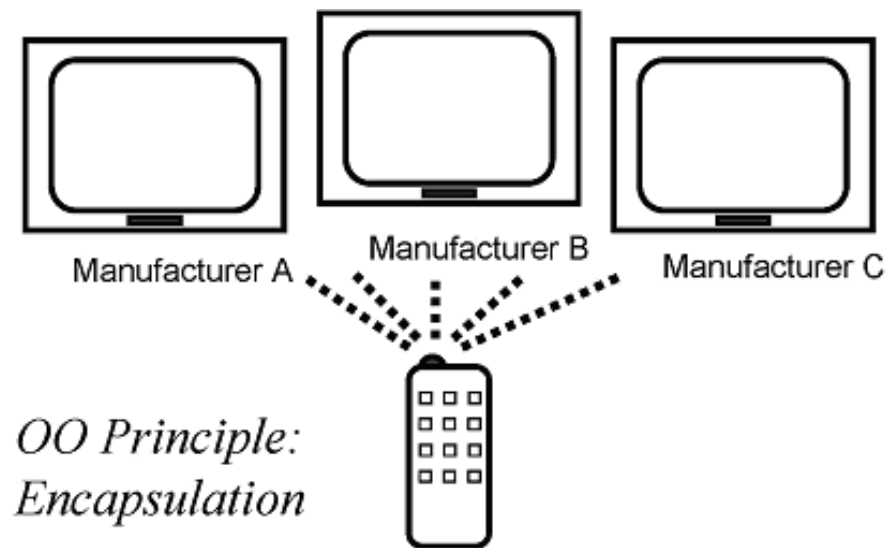


Where are we?

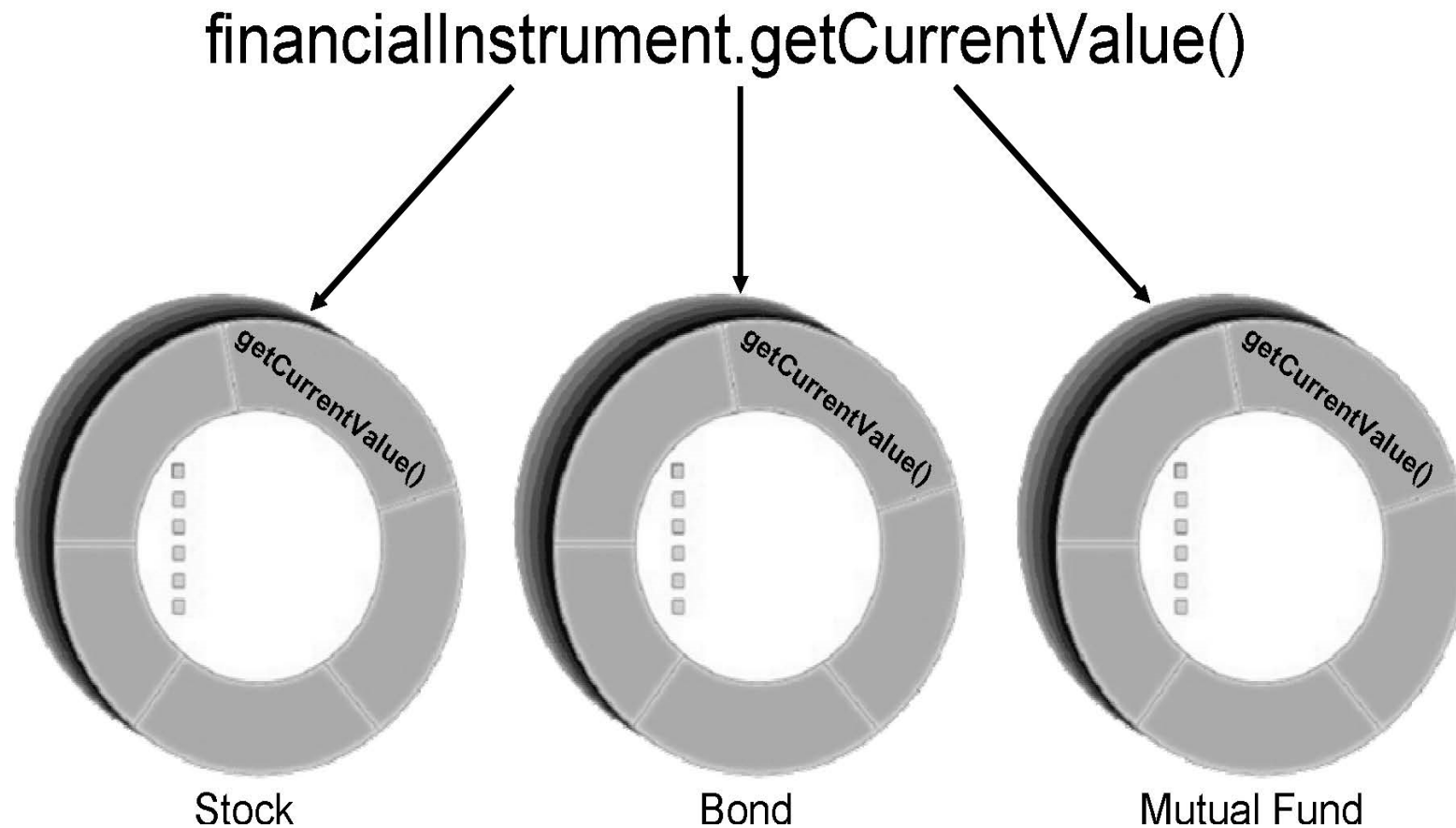
- 什么是对象？
- OO的四个基本原理
- 什么是类
- ☒ 多态性和泛化

2.4.1 多态性(Polymorphism)

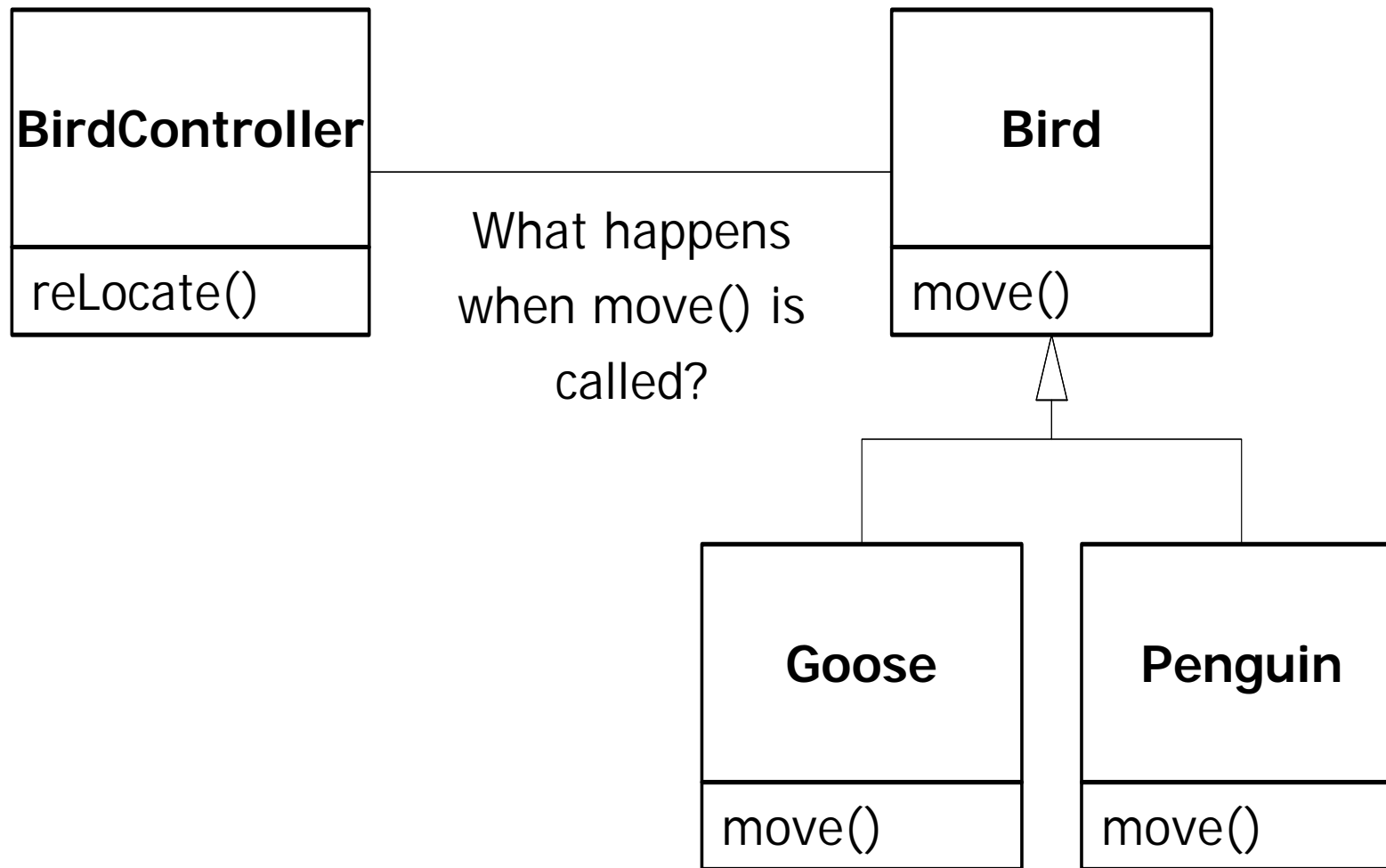
- **The ability to hide many different implementations behind a single interface.**
- 多态性是能够把多种不同的实现隐藏在一个单一的接口后面的能力。



2.4.1.1 多态性：例1

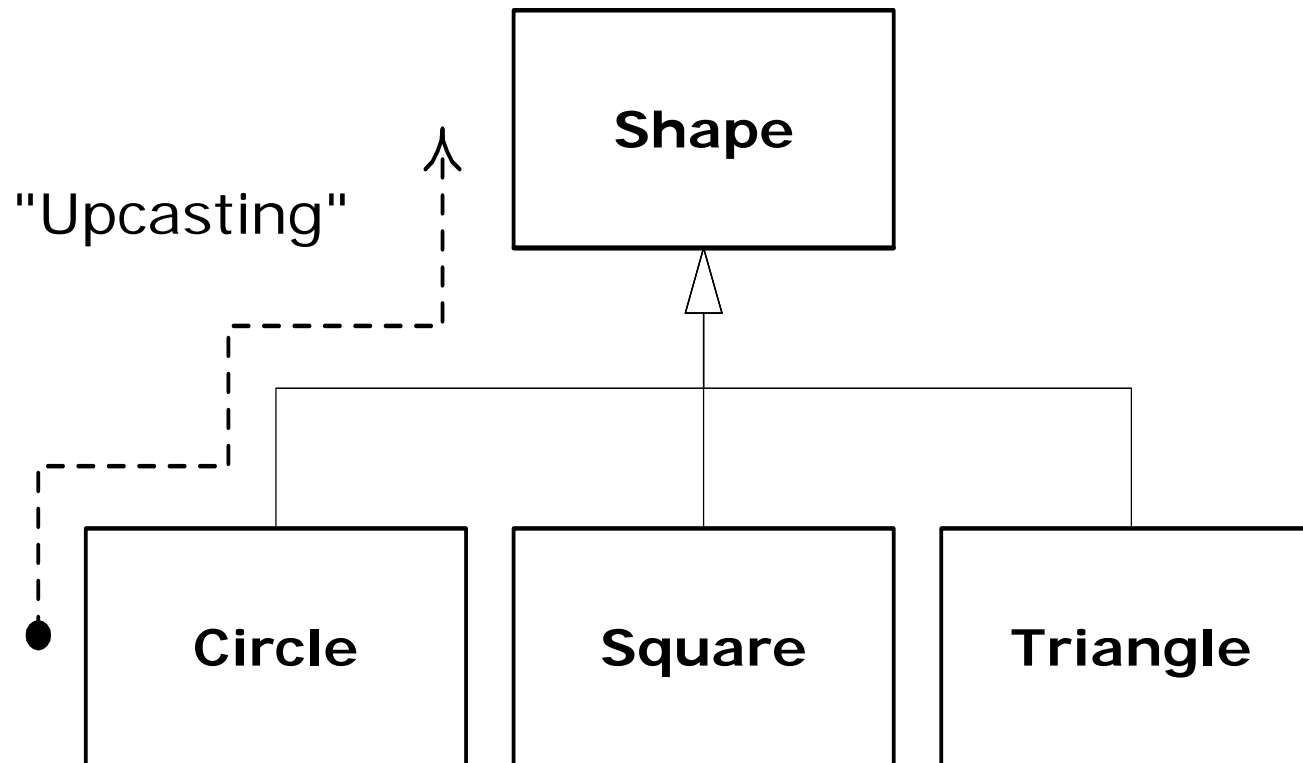


2.4.1.2 多态性：例2



2.4.1.3 多态性的实现机制

- ◆ 晚绑定 vs 早绑定
- ◆ Upcasting vs Downcasting





2.4.2 泛化(Generalization)

- **A relationship among classes where one class shares the structure and/or behavior of one or more classes.**

(泛化是类与类之间的这样一种关系：其中一个类共享一个或多个其它类的结构和行为)

- **Defines a hierarchy of abstractions in which a subclass inherits from one or more super classes.**

(定义了之类继承父类的抽象层次)

- **Single inheritance.**(单继承)

- **Multiple inheritance.** (多继承)

- **Is an “is a kind of” relationship.**

(是一种 “同一类” 的关系)

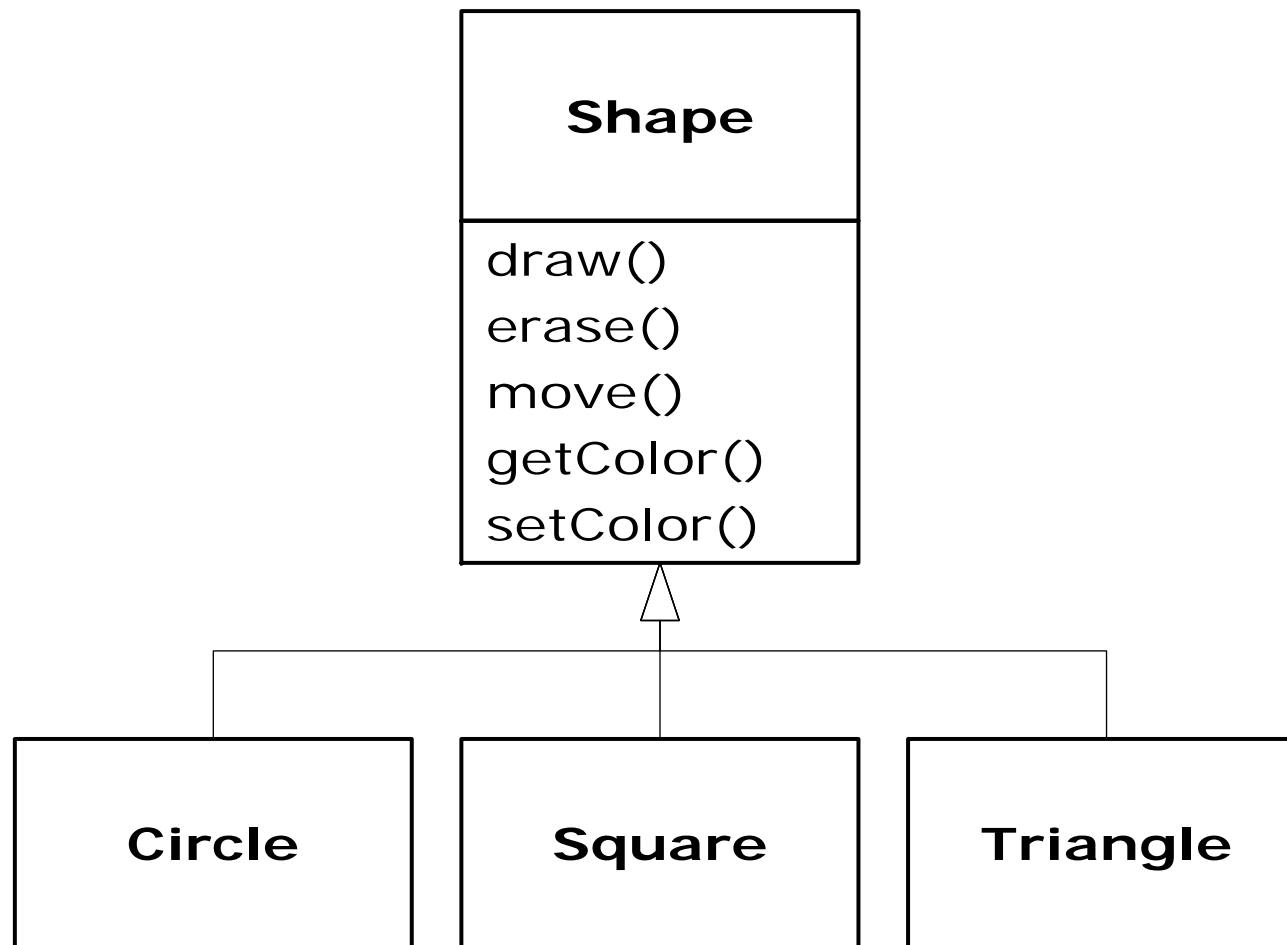


2.4.3 继承：重用接口

- 通过继承，可以创建一个与已存在的类（基类）具有类似功能的类（派生类）。
 - 派生类包含了基类的所有成员；
 - 派生类复制了基类的接口，因此派生类与基类是相同类型的。
 - 可以对派生类加以扩充：
 - 方法一：添加全新的函数 (is-like-a)**
 - 方法二：只改变基类中函数的行为，即“重载”函数。 (is-a)**

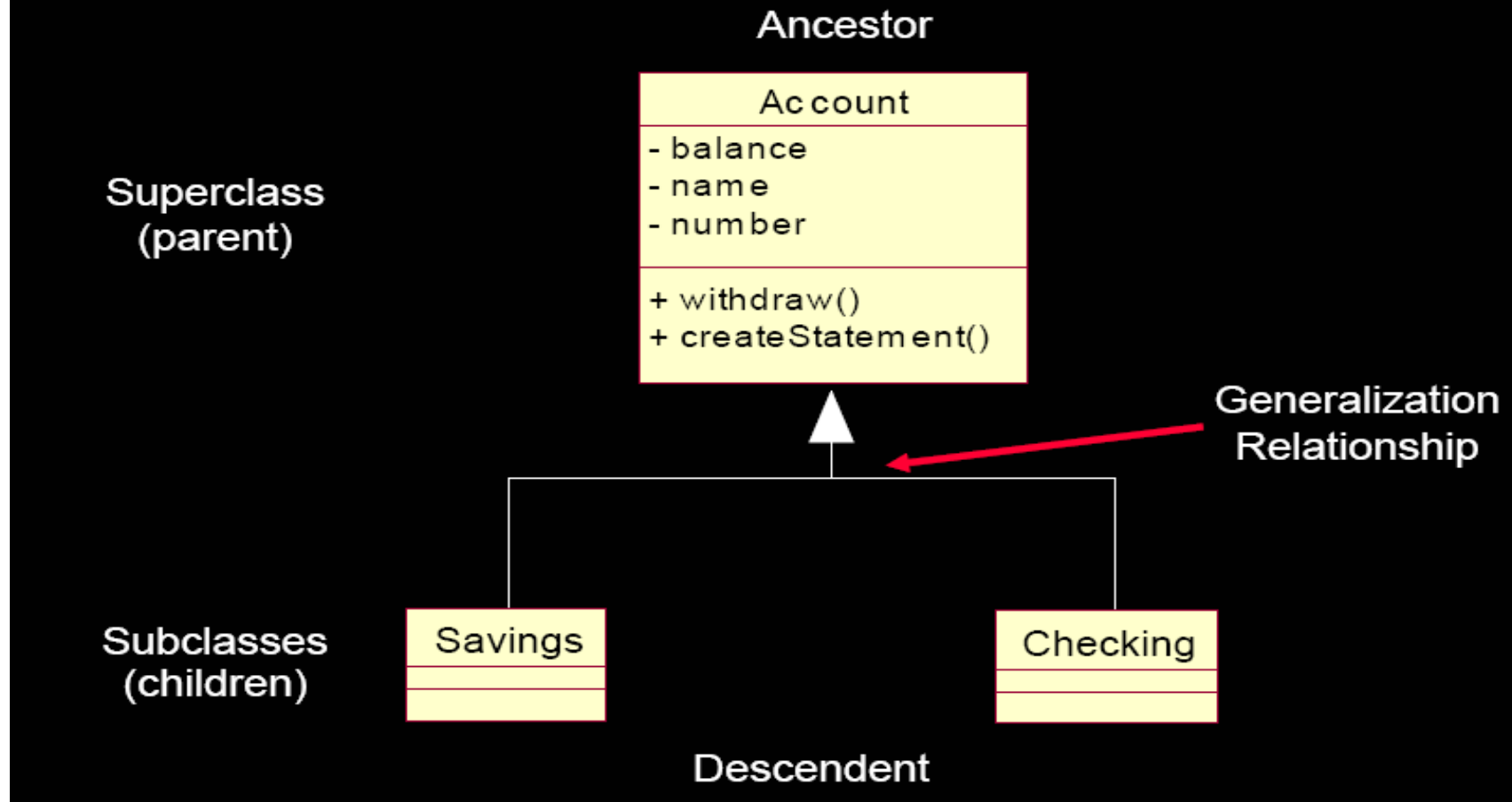
2.4.3.1 单继承：例1

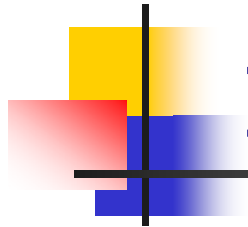
基类Shape含有大小、颜色位置等属性及图示的方法。



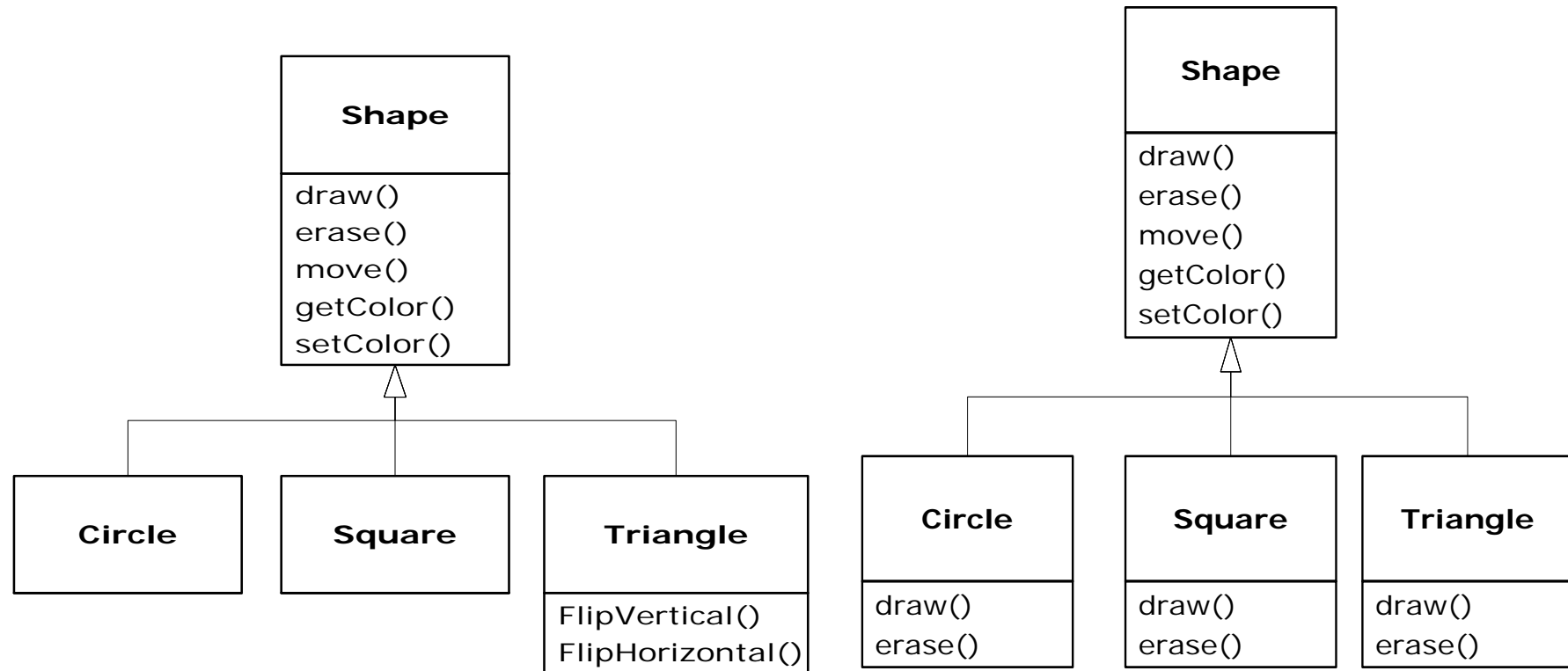
2.4.3.2 单继承：例2

◆ One class inherits from another.

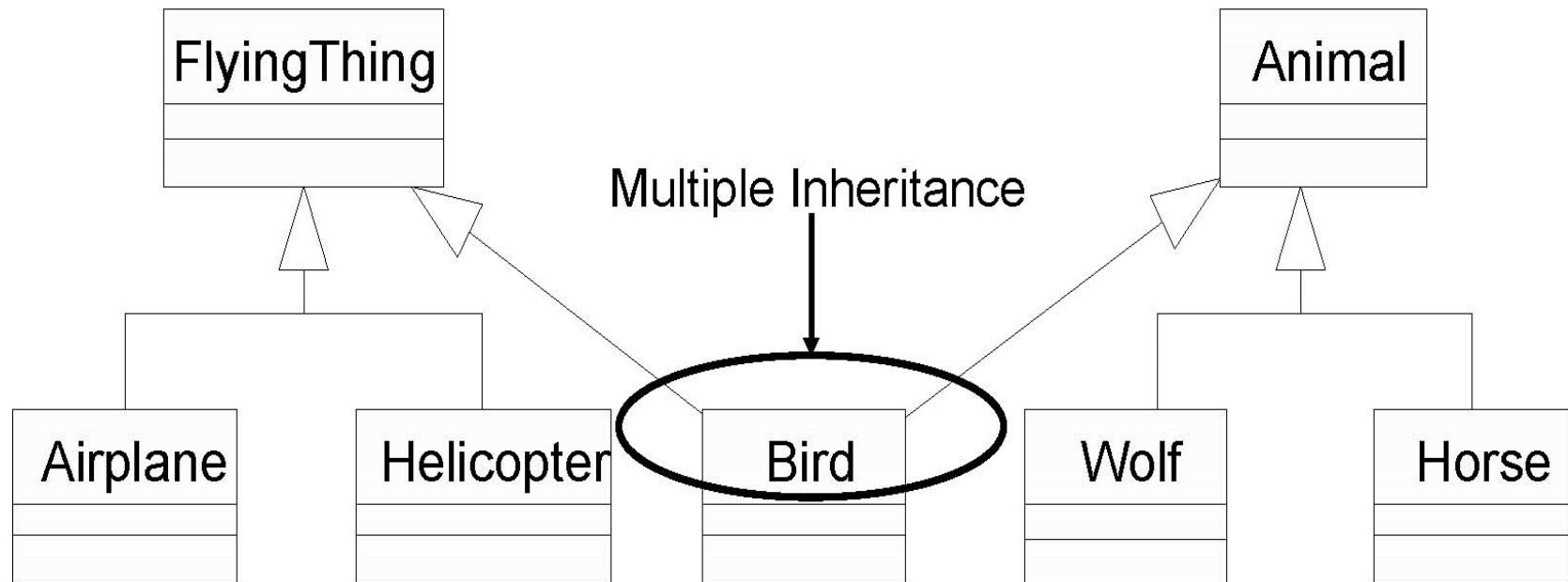




Is-like-a vs is-a-kind-of



2.4.3.3 多继承(Multiple Inheritance)



Use multiple inheritance only when needed and always with caution!



2.4.3.4 继承的本质

- A subclass inherits its parent's attributes, operations, and relationships.
- A subclass may:
 - Add additional attributes, operations, relationships.
 - Redefine inherited operations. (Use caution!)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

Inheritance leverages the similarities among classes.

继承提高了类与类之间的相似性。



3. 小结(Review)

- ② **What is an object?**
- ② **What are the four principles of object orientation? Describe each.**
- ② **What is a class? How are classes and objects related?**
- ② **What is an attribute? An operation?**
- ② **Define polymorphism. Provide an example of polymorphism.**
- ② **What is generalization?**