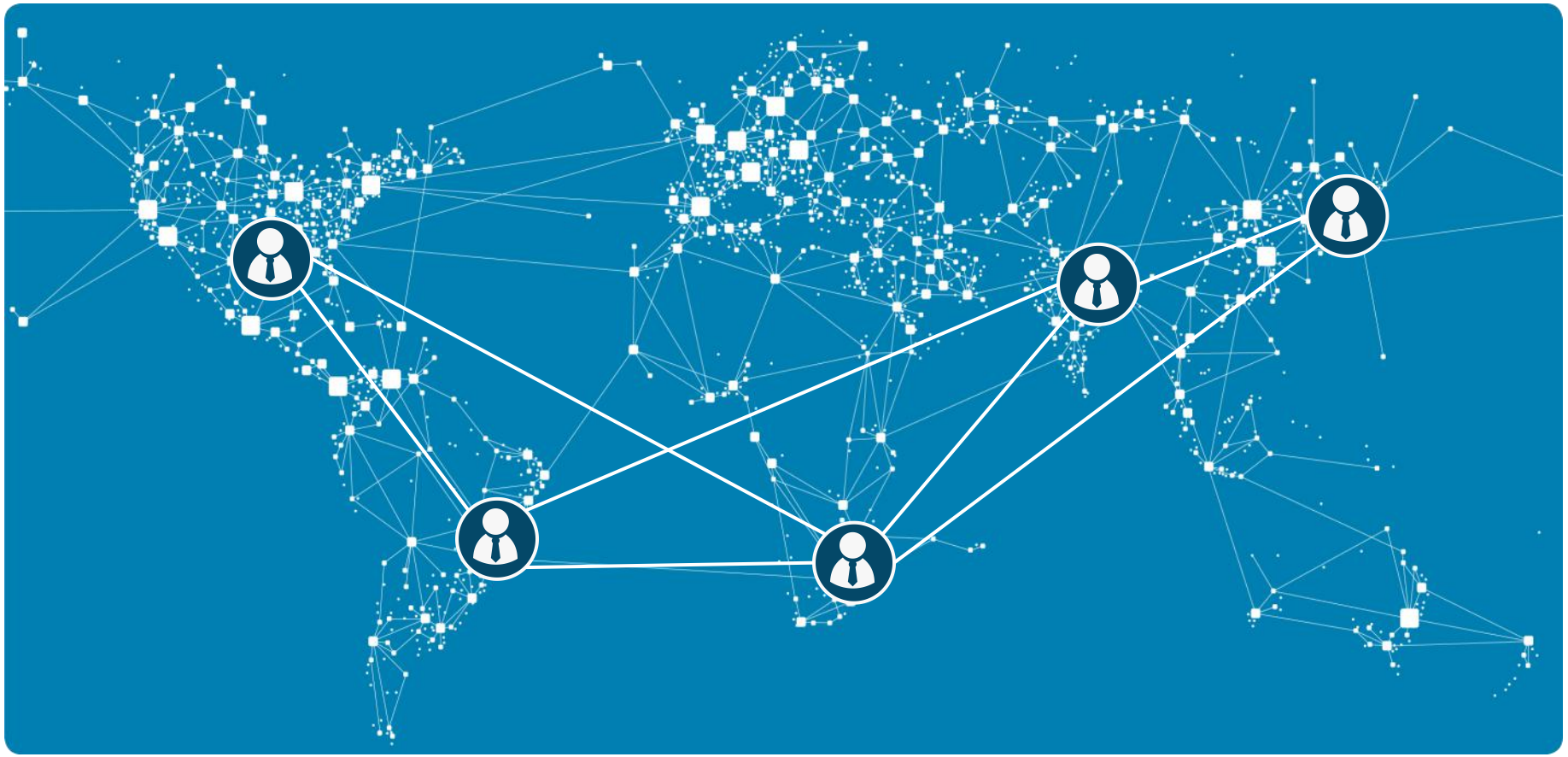


对象模型

对象模型

- 对象、属性与方法
- 面向对象方法的要素
- UML简介

面向对象

定义：面向对象方法就是为计算机软件的创建提出的一个模型化世界的抽象方法，它可以帮助我们更好地理解和探索世界。

软件工程专家Peter Codd和Edward Yourdon提出：

面向对象 = 对象 + 分类 +继承 +消息通信

面向对象方法的几个基本特征：

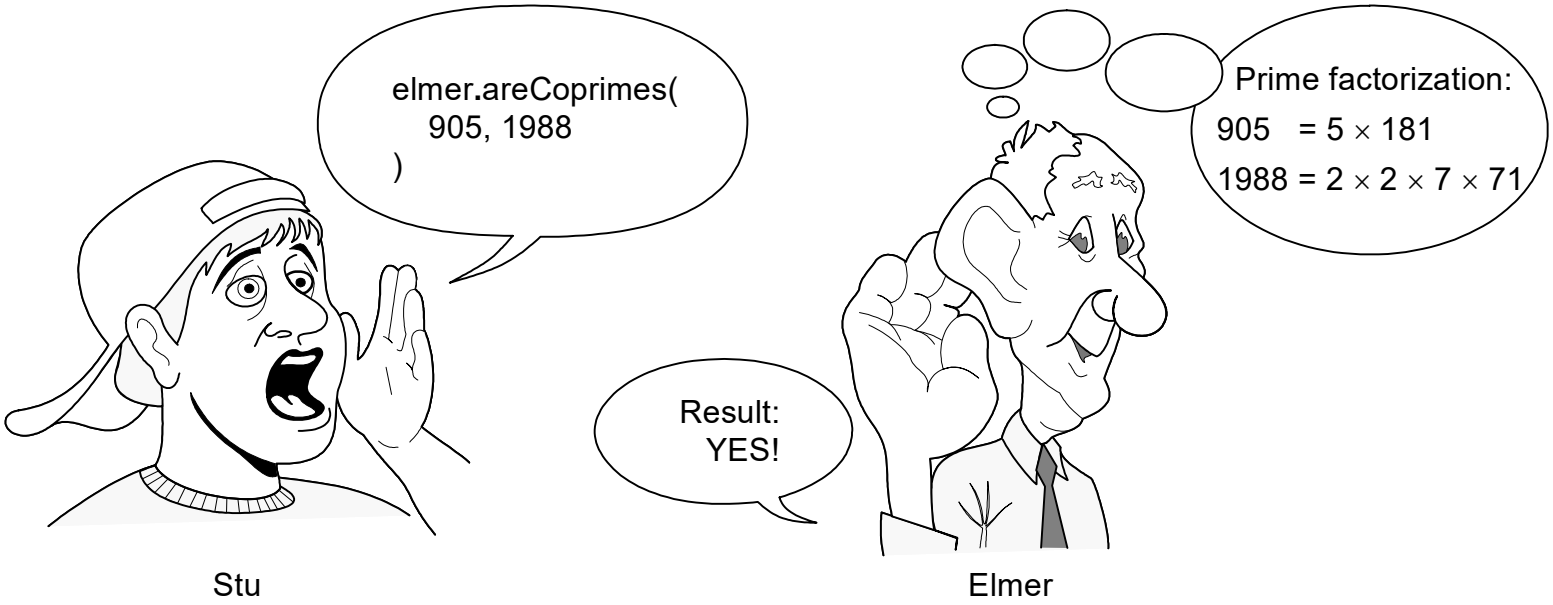
（1）客观世界是由对象组成的。

（2）具有相同的数据和操作的对象可以归并为一个类。

（3）类可以派生出子类，子类继承父类全部特征。

（4）对象之间通过消息传递互相联系。

对象、属性与方法



Prime factorization of 905:

5×181 (2 distinct factors)

Prime factorization of 1988:

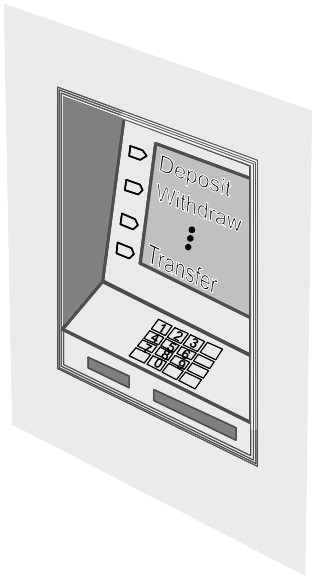
2×2×7×71 (4 factors, 3 distinct)

Two integers are said to be coprime or relatively prime if they have no common factor other than 1 or, equivalently, if their greatest common divisor is 1.

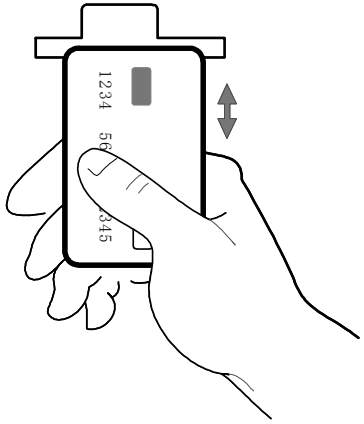
对象不会接受任意的消息传递

对象可接受的消息通常被定义为该对象的一组“方法”。
(或者说是，函数、操作、过程、子程序)

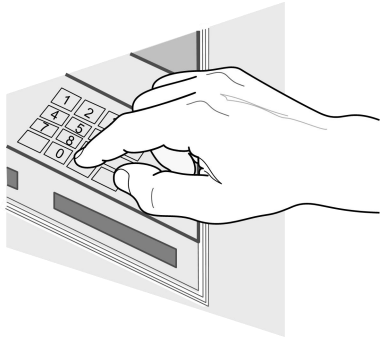
Object:
ATM machine



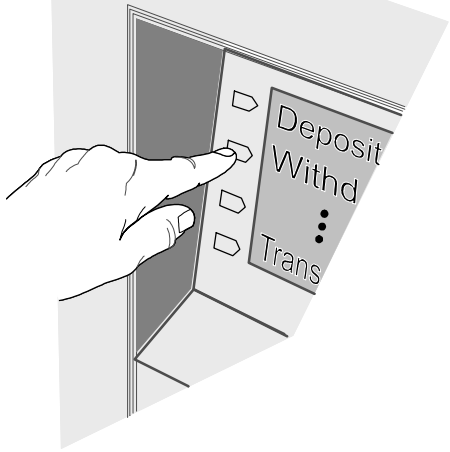
method-1:
Accept card



method-2:
Read code



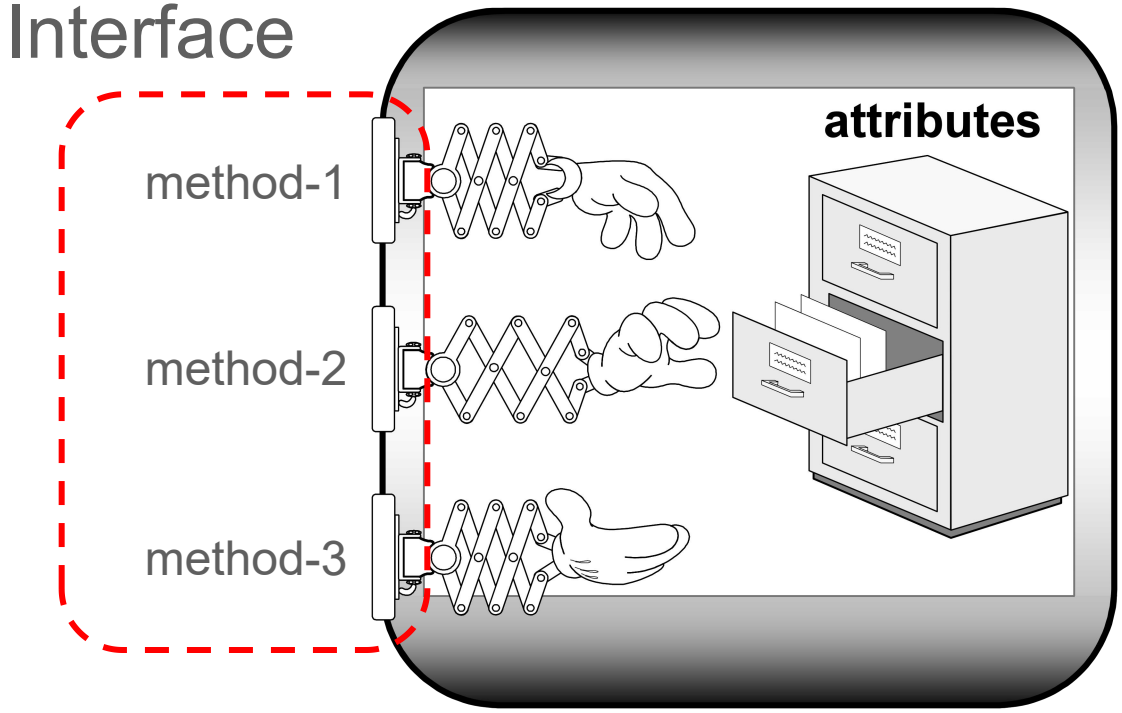
method-3:
Take selection



对象接口

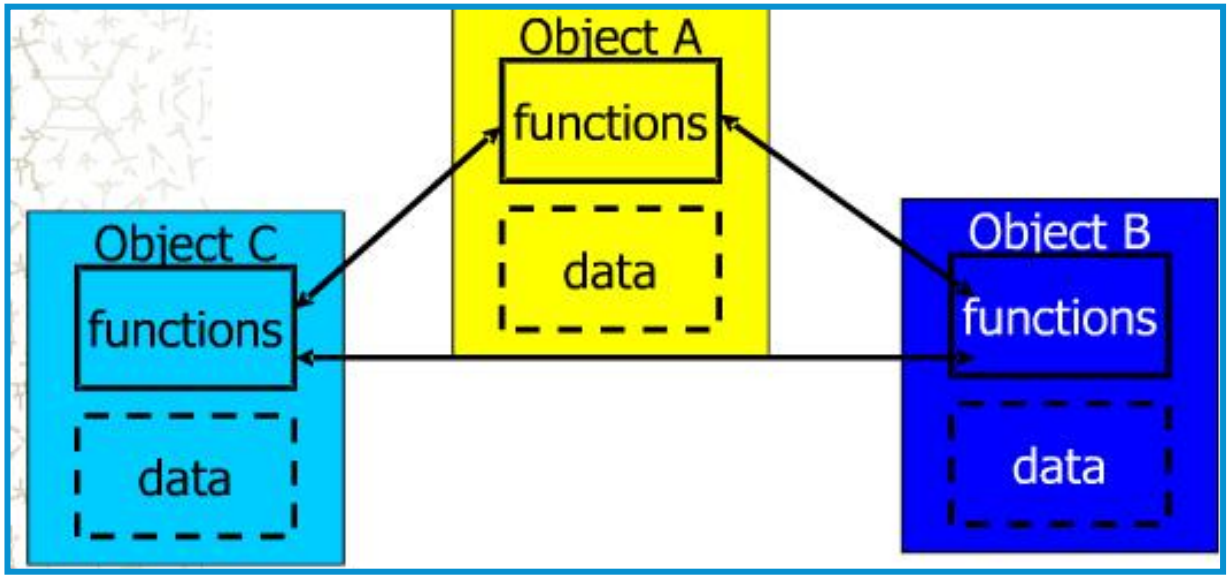
接口（Interface）：“方法声明(Method Signature)”的集合。

Method Signature: name, parameters, parameter types, return type



Object hides its state (attributes). The attributes are accessible only through the interface.

客户对象、服务对象以及消息传递

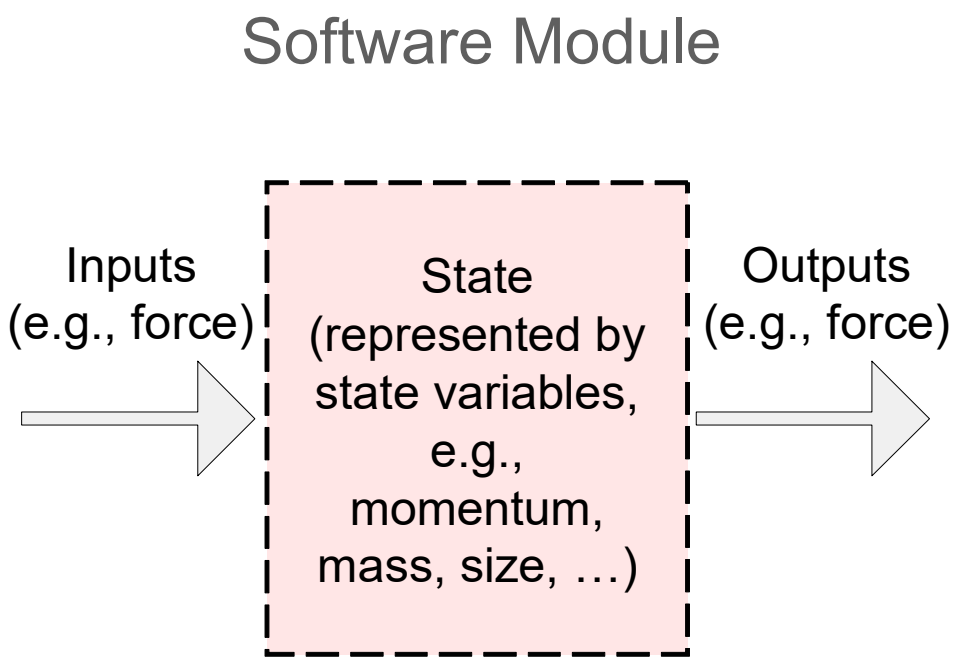


- 对象通过发送消息互相通信。
- **客户对象(Client Object)**: 发送消息并请求服务。
- **服务对象(Server Object)**: 提供服务并返回结果。

对象接口

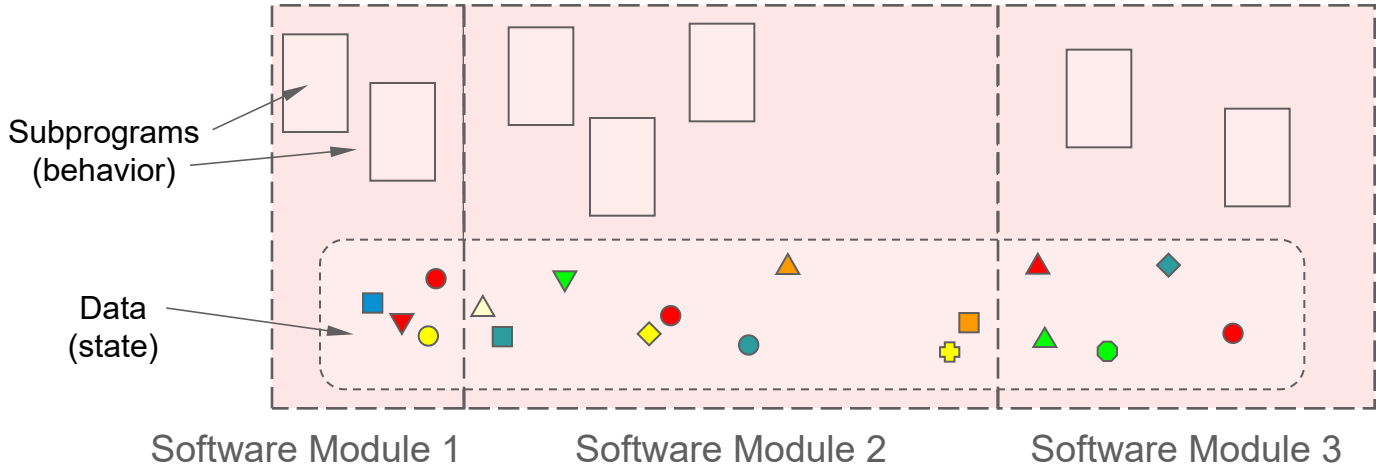
- 接口是一个软件对象所能提供的所有功能属性（**服务**）的集合。
- 接口中的方法定义了服务对象所能实现的各种“**服务**”。
- 这些方法（或**服务**）的产生以及命名，都必须遵循使用该服务的客户对象的需求。
  - **“On-demand” 设计**— 应该从客户对象的需求中“**提取**”服务对象类的接口以及各方法的具体实现，而不是向客户对象“**推送**”我们认为这个服务对象类应该提供的功能。

对象是软件模块(Software Module)



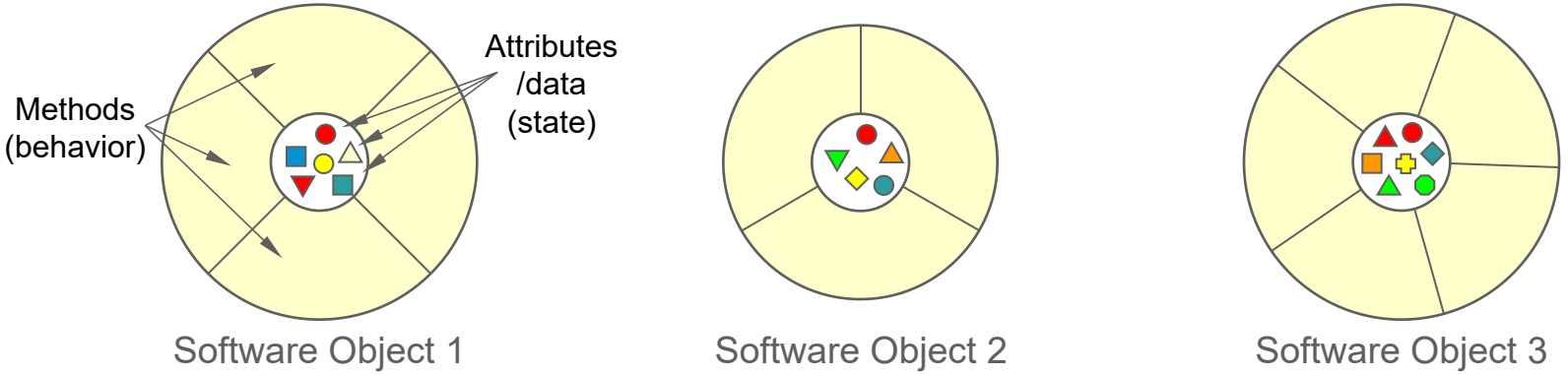
软件模块 & 软件对象

软件模块只是子程序块和数据的一种松散组合



数据的“交叉”访问常常导致错误的发生

对象封装数据



对象职责（1）

- 面向对象的关键特征就是职责的概念，即一个对象为其它对象所承担的职责。
  - 所有对象的职责描述了整个系统的设计。

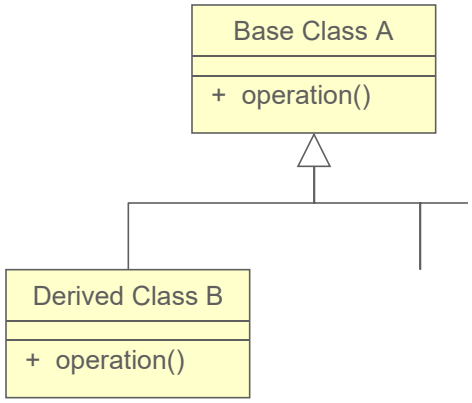
- 面向对象的其他特性，如封装、多态等，则是对象自身所具有的特性。

对象职责（2）

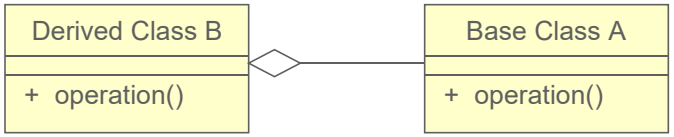
- 确定对象应该知道什么(状态)以及应该做什么(行为)的过程，称为职责分配。
- 对象的主要职责：
  - ✓ “知道”型职责（knowing）：数据的存储或对象属性
  - ✓ “做”型职责（doing）：用“方法”来实现具体算法
  - ✓ “交流”型职责（communicating）：通过发送消息与其他对象进行通信

对象关系(1)

- Composition: using instance variables that are references to other objects
- Inheritance: inheriting common properties through class extension



Inheritance



Composition

B acts as “front-end” for A and uses services of A
(i.e., B may implement the same interface as A)

对象关系(2)

这些关系又可以进一步细化为：

- *Is-a* 关系 : A class “inherits” from another class
- *Has-a*关系: A class “contains” another class
  - 组合
  - 聚合
- *Uses-a* 关系 : A class “uses” another class
- *Creates* 关系： A class “creates” another class

通过继承和组合实现重用和扩展

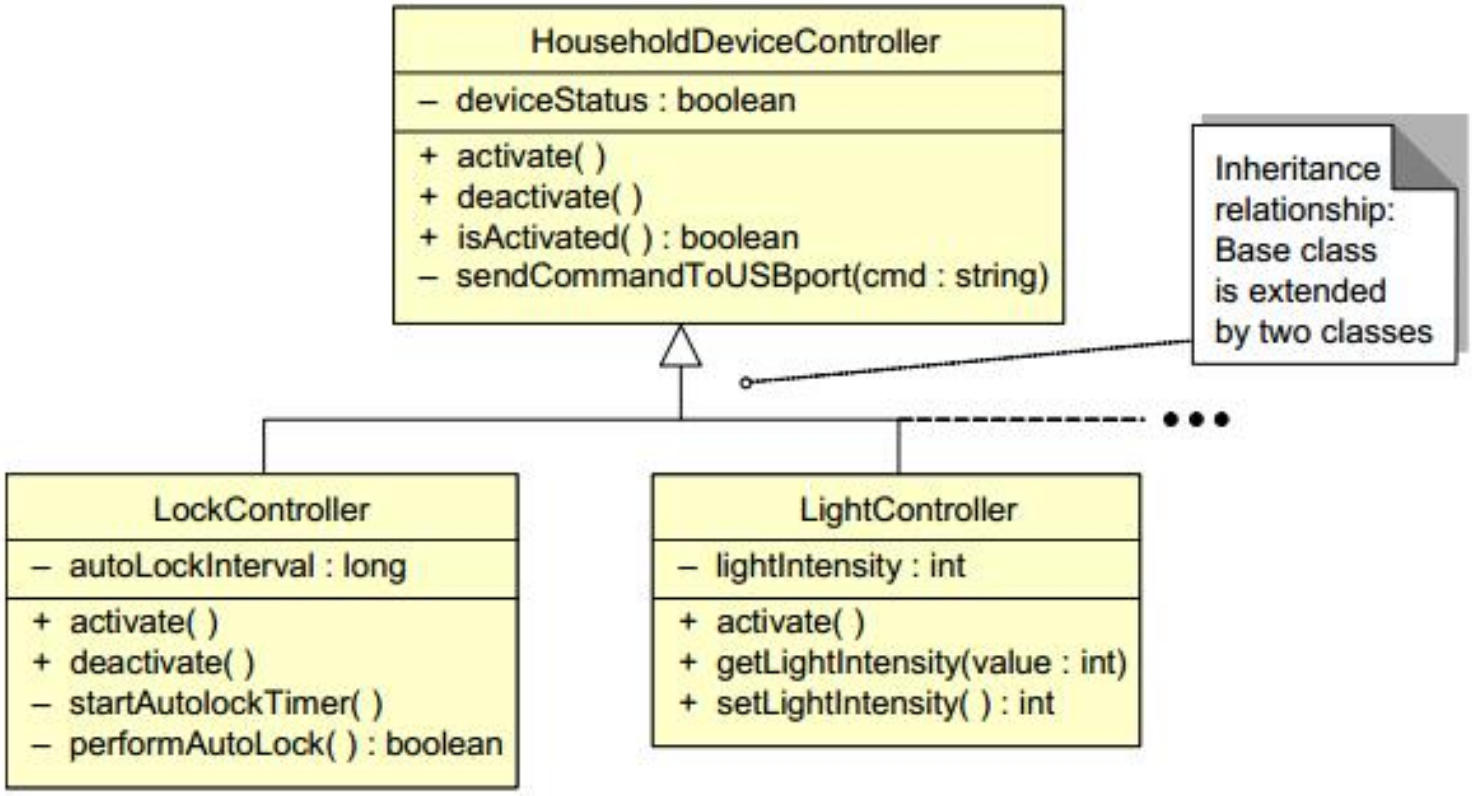
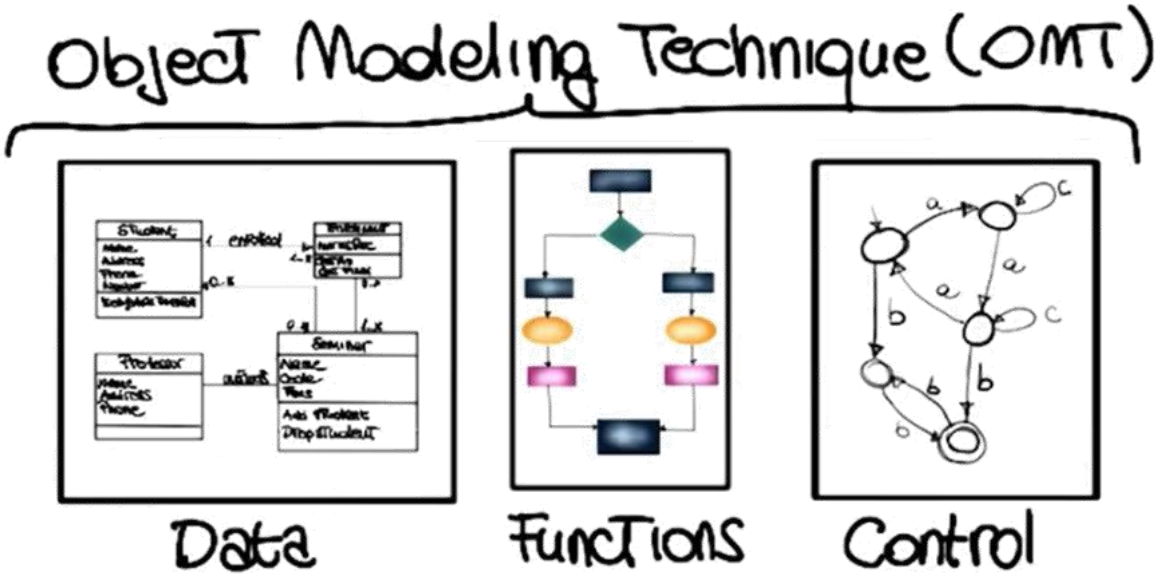


Figure 1-31: Example of object inheritance.

OOAD(OO Analysis and Design)

OOAD(OO Analysis and Design)

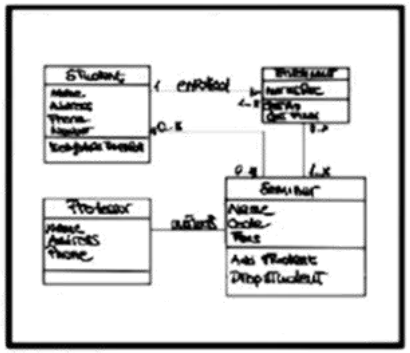
Rumbaugh



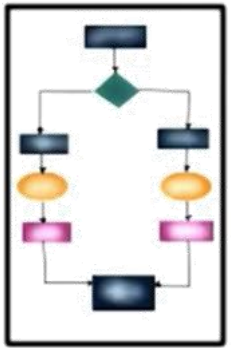
OOAD(OO Analysis and Design)

Rumbaugh

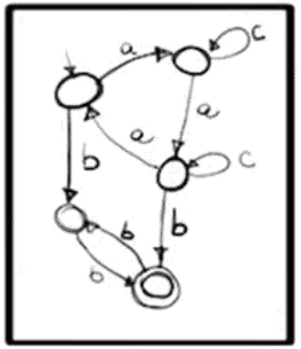
Object Modeling Technique (OMT)



Data



Functions

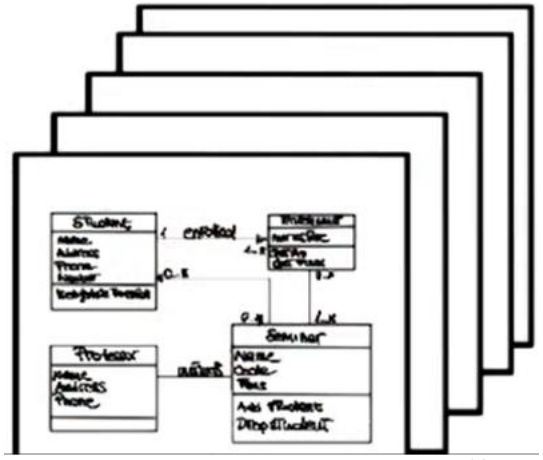


Control

Jacobson

Booch

Unified Modeling Language (UML)



What is UML?

- Pictures or views of an OO system
  - Programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- What is legal UML?
  - A descriptive language: rigid formal syntax (like programming)
  - A prescriptive language: shaped by usage and convention
  - It's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

UML: Unified Modeling Language

Union of Many Languages

- Use case diagrams
- Class diagrams
- Object diagrams
- Sequence diagrams
- Collaboration diagrams
- Statechart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams
- ….

A very big language!

Uses for UML

- As a sketch: to communicate aspects of system
  - Forward design: doing UML before coding
  - Backward design: doing UML after coding as documentation
  - Often done on whiteboard or paper
  - Used to get rough selective ideas
- As a blueprint: a complete design to be implemented
  - Sometimes done with CASE (Computer-Aided Software Engineering) tools
- As a programming language: with the right tools, code can be auto-generated and executed from UML
  - Only good if this is faster than coding in a "real" language