3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 多重继承:
  - 允许一个类有一个以上的直接父类。
  - 对应着客观世界中一种事物同时具有多种事物的特征、又有自己的特征这样的现实。

```
class EmpStudent : public Employee,
                   public Student {
/* ... */
};
```

3.2 程序设计语言中的OOP机制

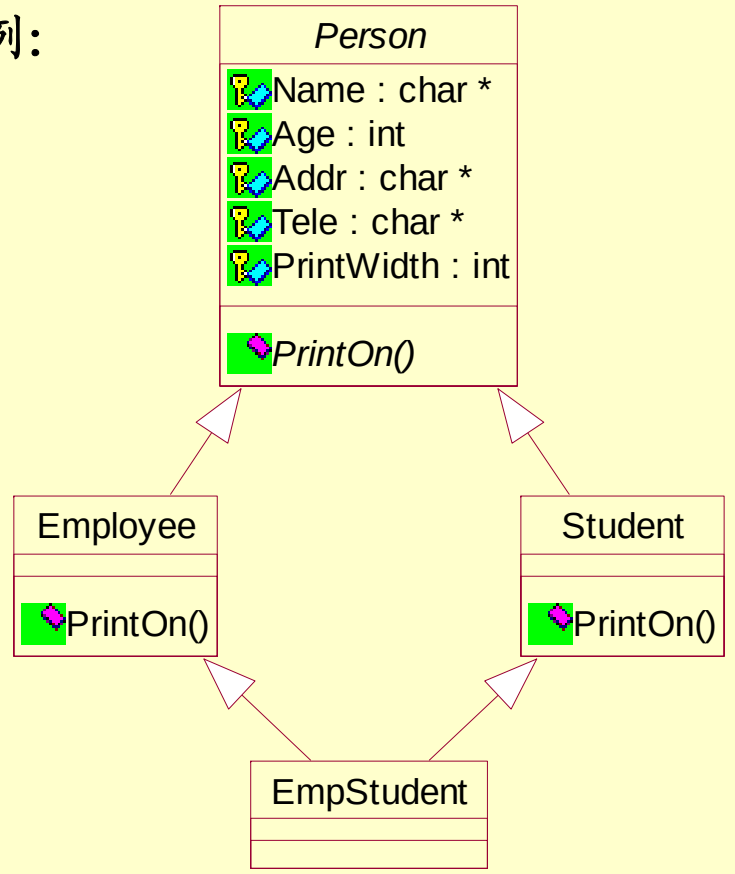
3.2.12 多重继承

- 类EmpStudent的实例化过程将两次间接地使用类Person定义的数据结构。这里存在着两种都可以成立的解释:

  （1）一个EmpStudent实例中应当含有两个Person实例的结构，因为地址、电话等与身份有关。

  （2）一个EmpStudent实例中应当只含有一个Person实例的结构，因为姓名、年龄等应唯一。

例:



3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 这种二义性的解决办法：由程序员自己决定采用何种解释，并且用相应的语法表示所采用的解释。

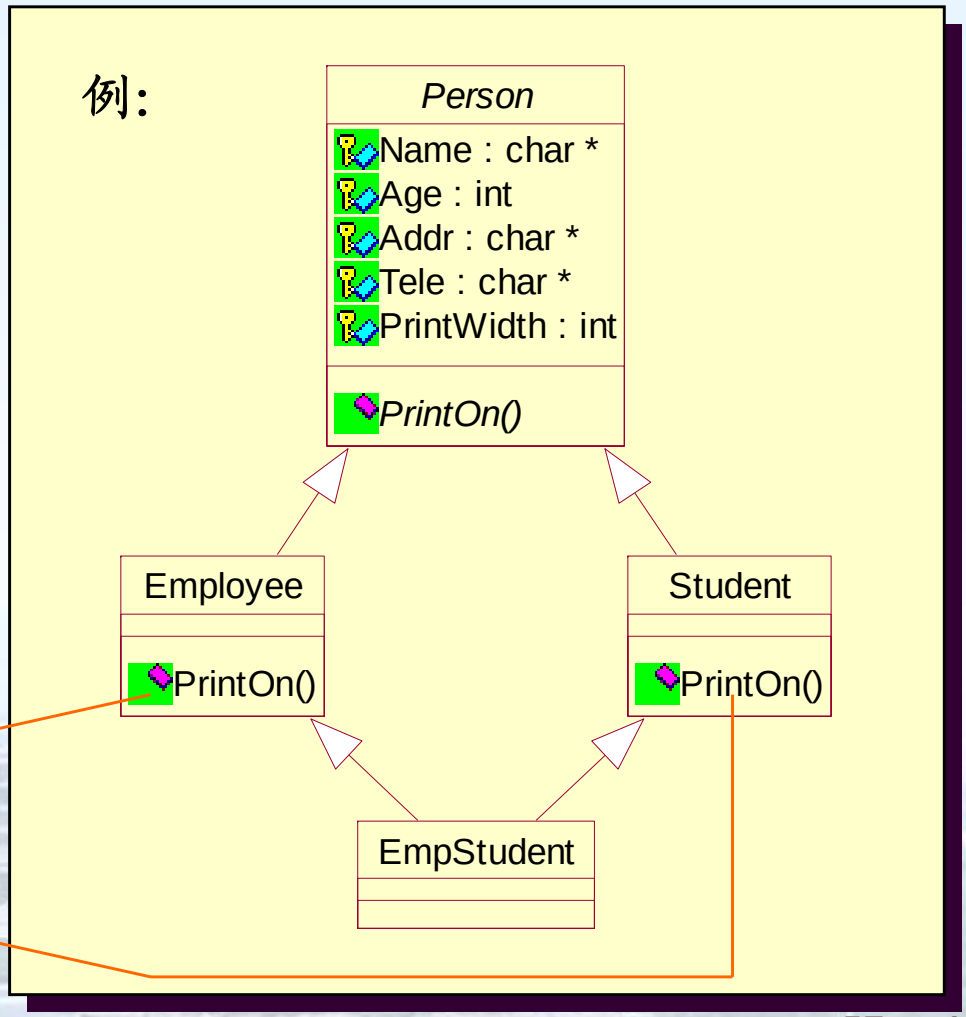
```
// EmpStudent实例复制Person定义的结构多次
class Employee : public Person {
 /* ... */
};
class Student : public Person {
 /* ... */
};
class EmpStudent : public Employee,
                   public Student {
 /* ... */
};
```

```
// EmpStudent实例复制Person定义的结构一次
class Employee : virtual public Person {
 /* ... */
};
class Student : virtual public Person {
 /* ... */
};
class EmpStudent : public Employee,
                   public Student {
 /* ... */
};
```

3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 如果类Employee和Student都对它们所继承的某个数据成员（如PrintWidth）进行了初始化，则当EmpStudent实例中只含有一个Person实例的结构时，它的实例化就与继承其直接父类的顺序有关。



3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 继承直接父类的顺序的规则：继承方式不同者，虚拟继承优先；继承方式相同者，按继承说明顺序。

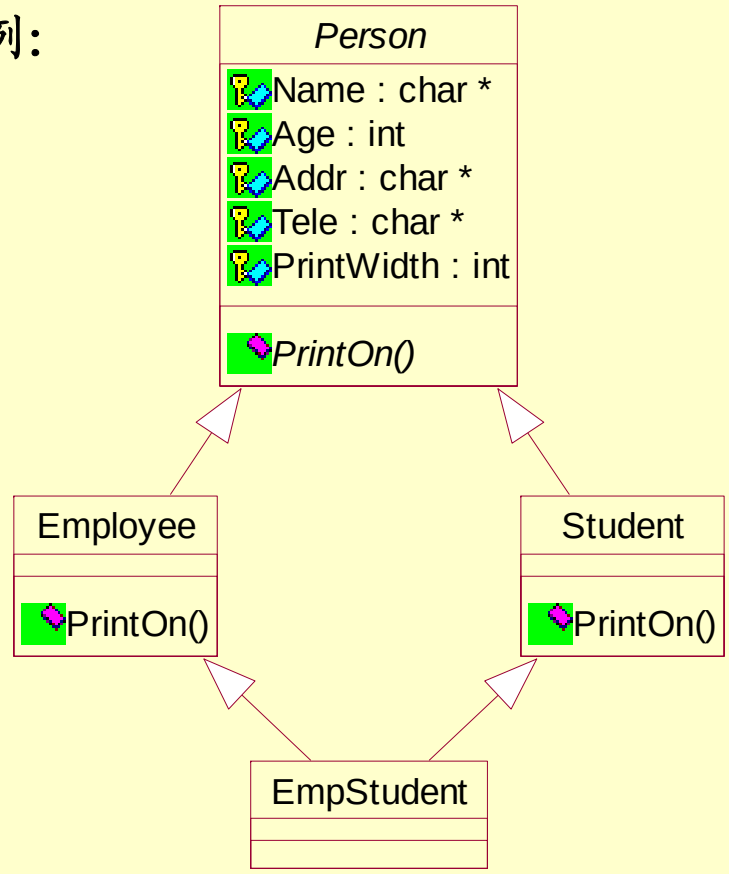
```
// A1继承直接父类的顺序：X2，X4，X1，X3
class A1 : public X1, virtual public X2,
           public X3, virtual public X4 {
 /* ... */
};
```

3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 多重继承使得类**EmpStudent**中有两个接口完全相同的方法：**PrintOn**，引起了成员的同名。

例：



3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- C++要求程序员对所继承的同名成员用其所在类的类名加以区分。

```
 EmpStudent es1, es2;
/* … */
es1.Employee::PrintOn();
/* … */
es2.Student::PrintOn();
```

3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 面向对象程序设计语言 Eiffel 提供了一种重命名机制。

设有3个类：

screen_object、text、tree。

它们定义的方法分别是：

screen_object：

setHeight、setWidth、displayObject、moveObject;

text：

displayText、addText、removeText、setHight、setWidth;

tree：

addChildNode、removeNode;

欲定义类window，继承以上3个类。发现继承的方法有重名，另外，类tree中的方法addChildNode和removeNode在类window中使用，其名称又过于一般化，不能准确表达操作的含义，但其方法体是合用的。

3.2 程序设计语言中的OOP机制

3.2.12 多重继承

操作接口定义

继承性定义

```
class window
export
   setHeight, setWidth, displayObject, moveObject,
   displayText, addText, removeText, setTextHeight, setTextWidth,
   addChildWindow, removeWindow
inherit
   screen_object;
   text rename
            setHeight as setTextHeight,
            setWidth as setTextWidth;
   tree rename
            addChildNode as addChildWindow,
            removeNode as removeWindow
end
```

3.2 程序设计语言中的OOP机制

3.2.12 多重继承

- 由于多重继承对应的类层次结构拓扑图是格结构，使得从一个类出发、沿着继承路径到达它的某个父类的路径不是唯一的，因而引发了多种的二义性问题。
- 多重继承的滥用（如：用多重继承代替组装关系），既扩大了这样的二义性问题出现的范围，又破坏了设计的自然性。
- 鉴于多重继承存在着一些问题，现在的趋势是在C++程序中尽量不用多重继承，而用单重继承和组装关系相结合来代替之，而Java则不支持多重继承。

3.3 面向对象的程序

3.3.1 经典的程序实例：单链表

- 传统的程序设计模式（采用Pascal）：

```
type nextnode = ↑node;
node = record
         number : integer;
         next : nextnode;
       end;
```

应用数据

链表支撑结构

```
Procedure f( var s_head : nextnode );
  var  p : nextnode;
Begin
  p:= s_head;          { 关于链表的操作 }
  While p<>null do     { 关于链表的操作 }
    Begin
      p↑.number := p↑.number + 1;
                       { 关于数据的操作 }
      p := p↑.next     { 关于链表的操作 }
    end
end;
```

将关于链表的操作与关于数据的操作混合一起，降低了程序的可重用性。

3.3 面向对象的程序

3.3.1 经典的程序实例：单链表

- 面向对象的程序设计模式（C++类的定义）：

```
typedef void* ent;
class slink {
 /* 根本没有public成员,用friend 规定了外
部访问范围 */
  friend class slist;
  friend class slist_iterator;
  slink* next;
  ent e;
  slink(ent a, slink* p)
    { e = a; next = p; }
};
```

```
class slist {
  friend class slist_iterator;
  slink* last;
public:
  int insert(ent a);
  int append(ent a);
  void clear();
  /* … … */
  slist() { last = 0; };
  slist(ent a)
    { last = new slink(a,0);
      last->next = last; }
  ~slist() { clear(); };
};
```

3.3 面向对象的程序

3.3.1 经典的程序实例：单链表

```
class slist_iterator {
  slink* ce;
  slist* cs;
public:
  slist_iterator(slist& s)
    { cs = &s; ce = 0; }
  ent opertor ++();       // 循环运算符
};
```

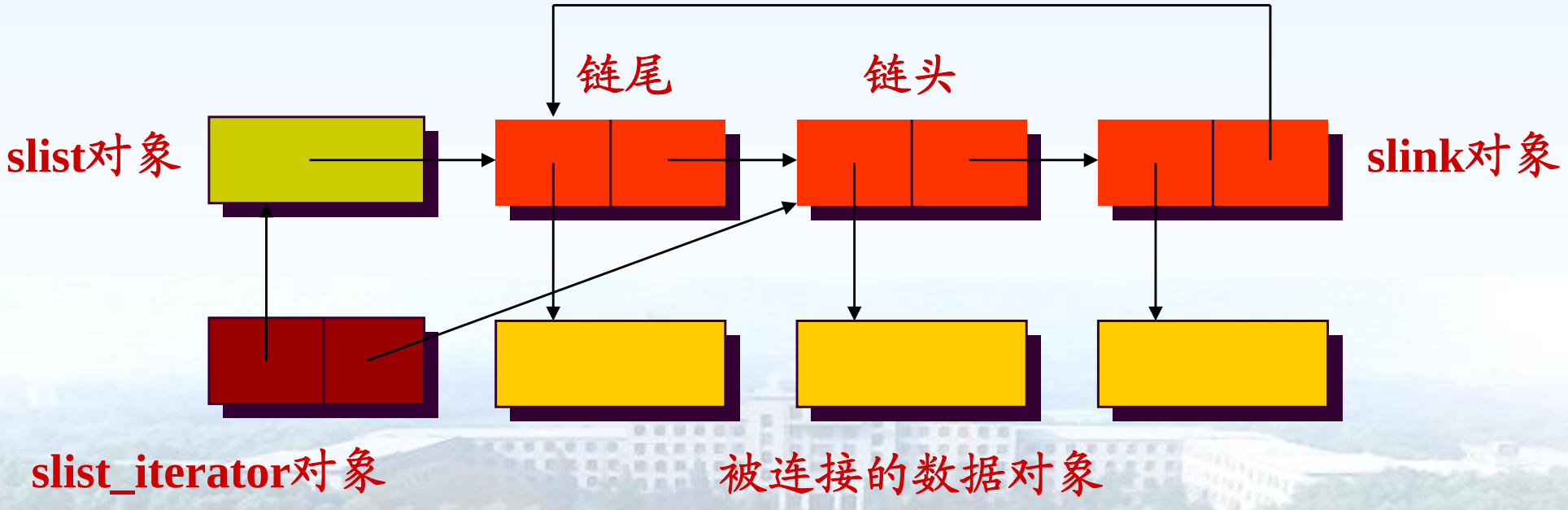
```
void f(slist & s)
{
  /* 生成一个与s关联的slist_iterator对
     象，作为循环控制机构的载体 */
  slist_iterator ff(s);
  ent p;      // 指向当前被链对象的指针

  while (p = ++ff)    // 在链表s中遍历
   {
    /* 利用强制多态对p指向的被链对象进行
       既定的操作，如
       ((Employee *)p)->changeAge(21);
    */
   }
}
```

3.3 面向对象的程序

3.3.1 经典的程序实例：单链表

- 面向对象的程序设计模式（对象组织结构）：



3.3 面向对象的程序

3.3.1 经典的程序实例：单链表

- 分析:
  - 把类的功用尽量设计成单纯的，将有利于重用。
  - 不同性质/不同层次的数据和操作，要分离到不同的类中，再设计相应的关联结构和协同操作。
  - 从对象的组织方式，可以看出程序设计的功底。
  - 语言提供的机制支持肯定是有应用背景的（例如本例中使用**friend**机制的效果），不能滥用。

3.3 面向对象的程序

3.3.2 定义有类型控制和操作投影的“外壳”

- 标识符名表类的定义:

```
/* 这里，用struct定义的是其成员的外部能
   见度全部为public的类 */
struct nlist : private slist {
  void insert(name* a)
    { slist::insert(a); }
  void append(name* a)
    { slist::append(a); }
  name* get()
    { return (name*)slist::get(); }
  nlist() {}
  nlist(name* a) : slist(a) {}
};
```

3.3 面向对象的程序

3.3.2 定义有类型控制和操作投影的“外壳”

- 分析:
  - nlist用类型name*（name是另一个表示符号名的类）以及相应的方法罩住了slist中的类型ent以及相应的方法，使得使用nlist时已经是专用链表的感觉，而实际上nlist并没有定义实质性的链表操作，也没有定义任何数据成员，它是slist的一个有确定的类型控制的外壳。
  - 这里的关键手段是private继承。
  - nlist还对slist的public方法进行了有选择的操作投影。
  - Java中的Interface结构与此例的作用类似。

3.3 面向对象的程序

3.3.3 利用统一接口的方法指针

- 定义了标准用户界面操作的抽象类及其方法指针类型:

```
class Std_interface {
public:
  virtual void start() = 0;
  virtual void suspend() = 0;
  virtual void resume() = 0;
  virtual void quit() = 0;
  virtual void full_size() = 0;
  virtual void small() = 0;

  virtual ~Std_interface() {}
};

typedef void (Std_interface::* Pstd_mem)();
```

3.3 面向对象的程序

3.3.3 利用统一接口的方法指针

- 构成用户界面操作命令的一个解释器：

```
// 界面类型（必须是Std_interface的子类）映射表
map<string, Std_interface *> variable;
// 界面操作（必须符合统一的接口规范）映射表
map<string, Pstd_mem> operation;

/* 将界面操作命令串中的类别和对应的对象指针、界面
   操作命令串中的操作和对应的方法指针分别填入上面
   两张表中。这一部分是会扩充的，但很单纯。
*/

// 解释执行特定界面类型的特定操作，这里是稳定的。
void call_member(string var, string oper)
{
  (variable[var]->*operation[oper])();
}
```

```
// 得到方法指针的方式
Pstd_mem s;
s = &Std_interface::suspend;
/* ... */
```

3.3.3 利用统一接口的方法指针

- 这个例子所体现的深层次的意义何在？
  - 将系统的扩充从以扩充操作的功能为主（因而会带来数据结构的相关制约、操作代码的大量修改等问题），变成了以扩充类型为主。
  - 前提：所扩充的类型与已有类型有（语法方面）比较密切的关联，如继承、纯虚拟函数等。
  - 这是值得提倡的利用继承性机制的做法。

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- An object is an abstraction of a set of real-world things such that
  - all the things in the set - the instances - have the same characteristics, and
  - all instances are subject to and conform to the same set of rules and policies.

— Shlaer, Mellor

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- An object is a thing real or abstract about which we store data and those methods which manipulate the data.

— Martin

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- An object is characterized by a number of operations and a state which remembers the effect of these operations.

— Jacobson

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- An object is a thing.
- It is created as the instance of an object type.
- Each object has a unique identity.
- Each objects offers one or more operations.

— OMG

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- An object has
  - state,
  - behavior, and
  - identity.
- The structure and behavior of similar objects are defined in their common class.
- The terms instance and object are interchangeable.

— Booch

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- OBJECT: An abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it (attributes, states) and interact with it (services).
- CLASS: A collection of one or more objects with a uniform set of attributes and services, including a description of how to create new objects in the class.

— Coad, Yourdon

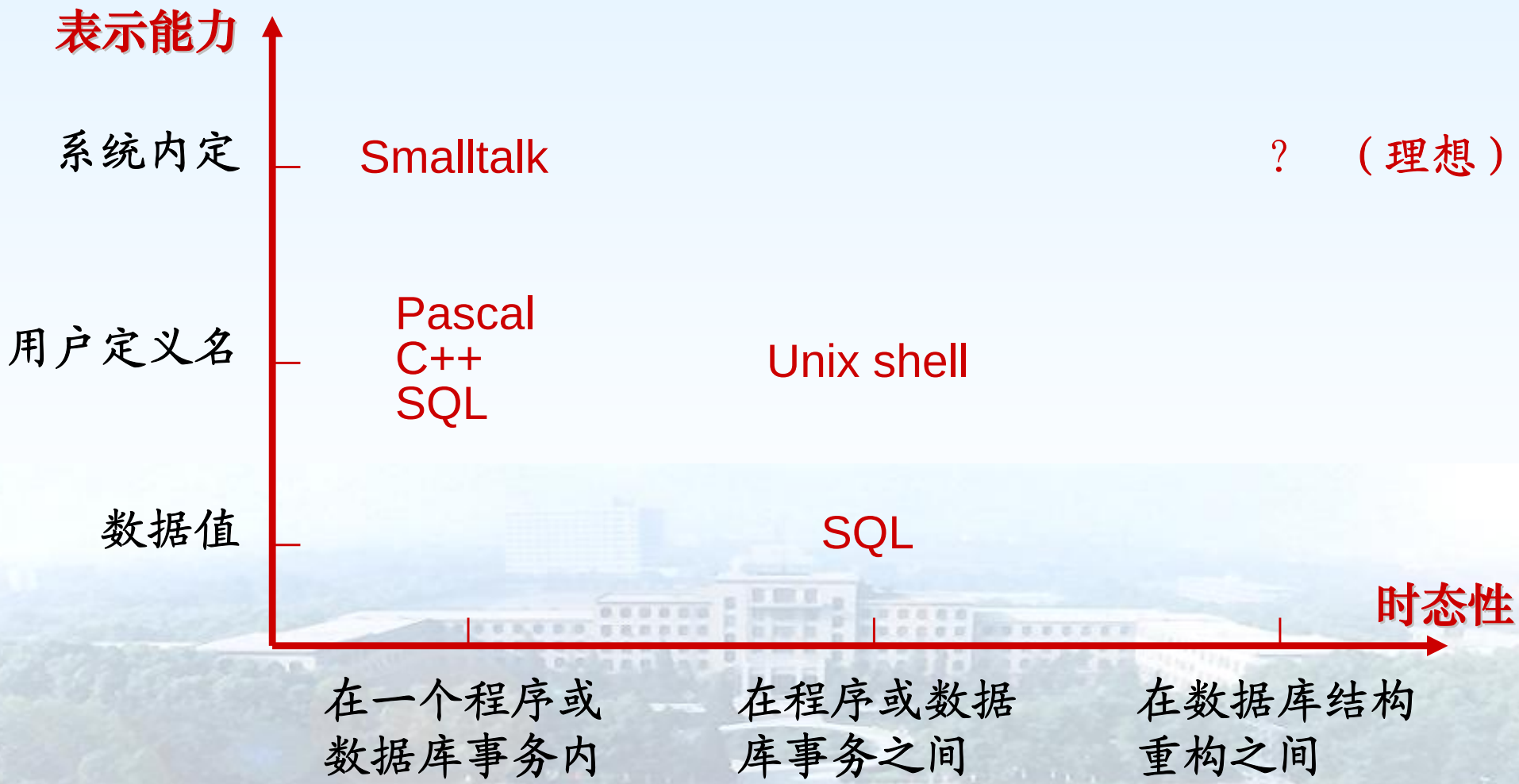
3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?

- 对象标识（Object Identity）：在一个完全的面向对象系统中，一个对象的标识和这个对象永久地结合在一起，不管这个对象的状态和结构经历了怎样的改变。
  - 对象标识的一致性：没有两个不同的对象有相同的标识；每一个标识对应于一个对象。
  - 对象标识的归一性：任何计算都可以归一地施加于持久特性不同的对象。
- 对象标识有两个正交（相互独立）的属性：表示能力和时态性。

3.4 关于面向对象程序设计的讨论

3.4.1 什么是对象?



3.4 关于面向对象程序设计的讨论

3.4.2 什么是类?

- 类既是一个对象集合中对象特征的抽象（对象工厂、对象的模板），又是这个集合的整体表示（对象仓库）。
- 类属于运行时的概念，不是用于检查程序的正确性，而是用于产生或操纵对象，还可以在运行时当作特殊的对象而被操纵，包括动态地扩充和修改。
- 语言中的所有实体（包括类本身）都是某个类的实例，因此语言中的所有实体都是对象。
- 典型代表：**Smalltalk-80**。

— 基于类概念的系统

3.4 关于面向对象程序设计的讨论

3.4.2 什么是类?

- 类仅仅是对象特征的抽象，不是对象集合。
- 类属于编译时的概念，用于检查程序的正确性、实现继承性和保证对象的ADT特性，其实例化功能被翻译成可执行的程序段。在运行时类已经不再是独立的实体，不允许也不可能被修改。
- 存在一个由编译程序决定的、不允许动态修改的运行形态，类不能被看成是对象。
- 典型代表：C++。

— 基于类型概念的系统

3.4 关于面向对象程序设计的讨论

3.4.3 什么是继承性?

- 替代继承：父类对象可以用子类对象替代。
- 包含继承：关于父类对象的操作包含了关于子类对象的操作，子类对象的结构可有扩展;
- 限制继承：子类对象在父类对象的属性上加以某些约束（如值域的约束）后构成;
- 特化继承：子类对象在父类对象的属性集合上增加某些属性（即结构化约束）后构成。

3.4 关于面向对象程序设计的讨论

3.4.3 什么是继承性?

- 代理（Delegation）机制的基本思想:
  - 没有类的概念，对象既可以有状态的变化，又可以作为原型来产生新的对象－共享（而不是分类）是占主导地位的概念。
  - 产生新的对象时，原型的当前值也赋予新的对象。
  - 对象可以互相共享，而不是彼此独立－一个对象可以由其他相关对象来代理执行一些自己不具备的操作。
  - 实例生成所涉及的只是当前还不能与其他实例所共享的那些属性和操作，以及用来描述这种共享关系的消息传递。

要点与引伸

- 多重继承中出现的问题，多与继承路径的拓扑方式相关。
- 对象的组织，通常需要同时利用多种机制的支持。
- 一个对象的生命期要由其他对象来控制，避免用一般的过程来控制。
- 如果一个方法的接口比较复杂，而且无法再简化了，说明这个方法的功能已经太复杂了，那就坚决地分解它！
- 在同一个类层次结构分枝中对方法接口进行统一，既有利于进行基于解释的动态调用，还有利于今后基于类型的扩充。

第二次作业（7 天后提交，10天后截止）

1. A system is required for managing data processing at a University. Three different classes in the system are **Student**, **LectureTheatre** and **Module**. For each class, a method is required which will produce a textual summary of an instance. The summary of a Student instance contains the name of the student and his/her registration number. The summary of a LectureTheatre instance contains its room number (a floating point number; for example, 10.102 indicates room 102 in building 10) and the number of seats it contains. The summary for a Module instance is the title of the module, its code (e.g., 01001) and the number of students currently attending it.

Implement the Student, LectureTheatre and Module classes in C++ or Java. For each class, you only need to show the instance variables, a constructor, and the necessary method(s). Your implementation should pass a testing program. For example, if you implement these classes in C++, the testing program will be:

```
int main ()
{
   UniversityObject   *a1 = (UniversityObject *) new Student ("张三", "0320121113"),
                      *a2 = (UniversityObject *) new LectureTheatre (3.205, 186),
                      *a3 = (UniversityObject *) new Module ("面向对象技术", "0321011", 185);
   printf ("汇总结果:\n    学生: %s\n    教室: %s\n    课程: %s\n",  a1->getSummary(), a2->getSummary(), a3->getSummary());
   return 0;
}
```

And the testing program must have the following output:

```
汇总结果:
      学生: 学号 0320121113, 张三
      教室: 3号楼 205, 共 186 个座位
      课程: 编号0321011, 《面向对象技术》, 185人选修
```

2. 设计和实现一个C++类模板，来提供一种采用数组来存储的、元素为任意类型的环形队。要求提供的操作：加入元素；提取元素；返回环形队允许存储的元素个数最大值；返回当前的有效元素个数。

选做：当加入元素时，若环形队的存储空间已满，能够自动扩张一定大小的存储空间，并保证环形队的行为特征不变。

§4. 面向对象系统分析与设计

引言

What is Systems Analysis and Design (SAD)?

- Systems analysis is the study of a business（商务/业务） problem domain to recommend improvements and specify the business requirements for the solution.
- Systems design is the specification or construction of a technical, computer-based solution for the business requirements identified in a systems analysis.

引言

Why is SAD important?

- Success of information systems depends on good SAD.
- Widely used in industry - proven techniques.
- Part of career growth in IT - lots of interesting and well-paying jobs!
- Increasing demand for systems analysis skills.

引言

Skills Required by Systems Analysts

- Working knowledge of information technology
- Computer programming experience and expertise
- General business knowledge
- Problem-solving skills
- Interpersonal communication skills
- Interpersonal relations skills
- Flexibility and adaptability
- Character and ethics
- Systems analysis and design skills

引言

Skills Required by Systems Analysts

- Working knowledge of
- Computer programmin
- General business kno
- Problem-solving skills
- Interpersonal commun
- Interpersonal relations
- Flexibility and adaptab
- Character and ethics
- Systems analysis and

Current Information Technologies

- Automatic data capture
- Client/server architecture
- Component programming languages
- Electronic commerce
- ERP
- GUI
- Internet, Intranet, and extranet
- Object programming languages
- Rapid application development
- Relational DBMS
- Services
- Telecommunications and networking

引言

Skills Required by Systems Analysts

- Working knowledge of information technology
- Computer programming experience and expertise
- General business knowledge
- Problem-solving skills
- Interpersonal commun
- Interpersonal relations
- Flexibility and adaptab
- Character and ethics
- Systems analysis and

Business Literacy Subjects

- Accounting（会计）
- Business Law and ethics
- Economics
- Manufacturing
- Marketing
- Operations management
- Organizational behavior

引言

Skills Required by Systems Analysts

- Working knowledge of information technology
- Computer programming experience and expertise
- General business knowledge
- Problem-solving skills
- Interpersonal commun
- Interpersonal relations
- Flexibility and adaptab
- Character and ethics
- Systems analysis and

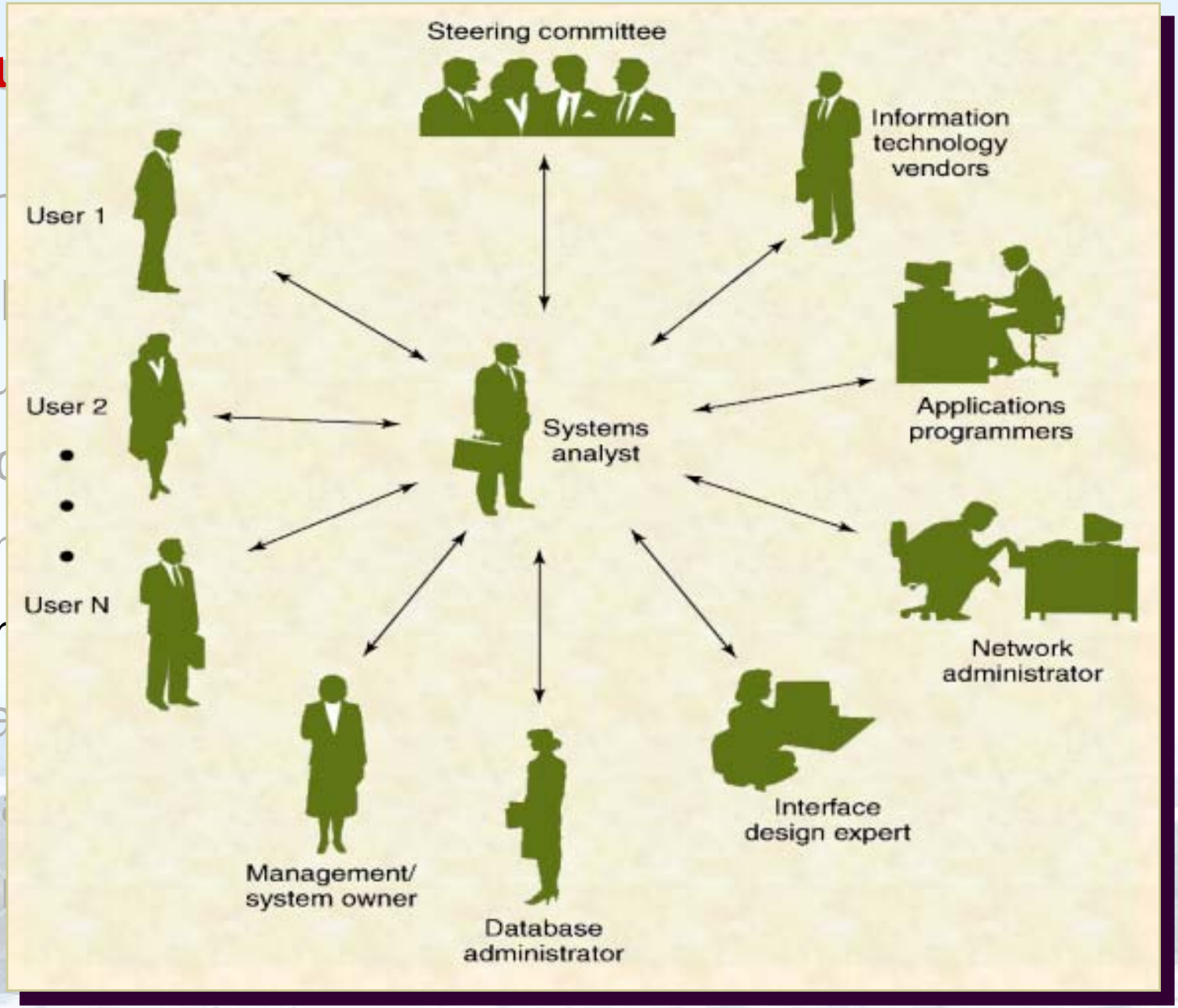
Interpersonal Comm. Subjects

- Business speaking
- Business writing
- Interviewing（面谈）
- Listening（倾听，应大智若愚）
- Persuasion（说服力）
- Technical discussion
- Technical writing

引言

Skills Requ

- Working kn
- Computer
- General bu
- Problem-so
- Interperson
- Interperson
- Flexibility a
- Character
- Systems a



引言

Skills Required by Systems Analysts

- Working knowledge of information technology
- Computer programming experience and expertise
- General business knowledge
- Problem-solving skills
- Interpersonal communication skills
- Interpersonal relations skills
- Flexibility and adaptab
- Character and ethics
- Systems analysis and

Ethics（伦理）

- Ethics is a personal character trait in which an individual understands the difference between “right” and “wrong” and acts accordingly.

引言

Skills Required by Systems Analysts

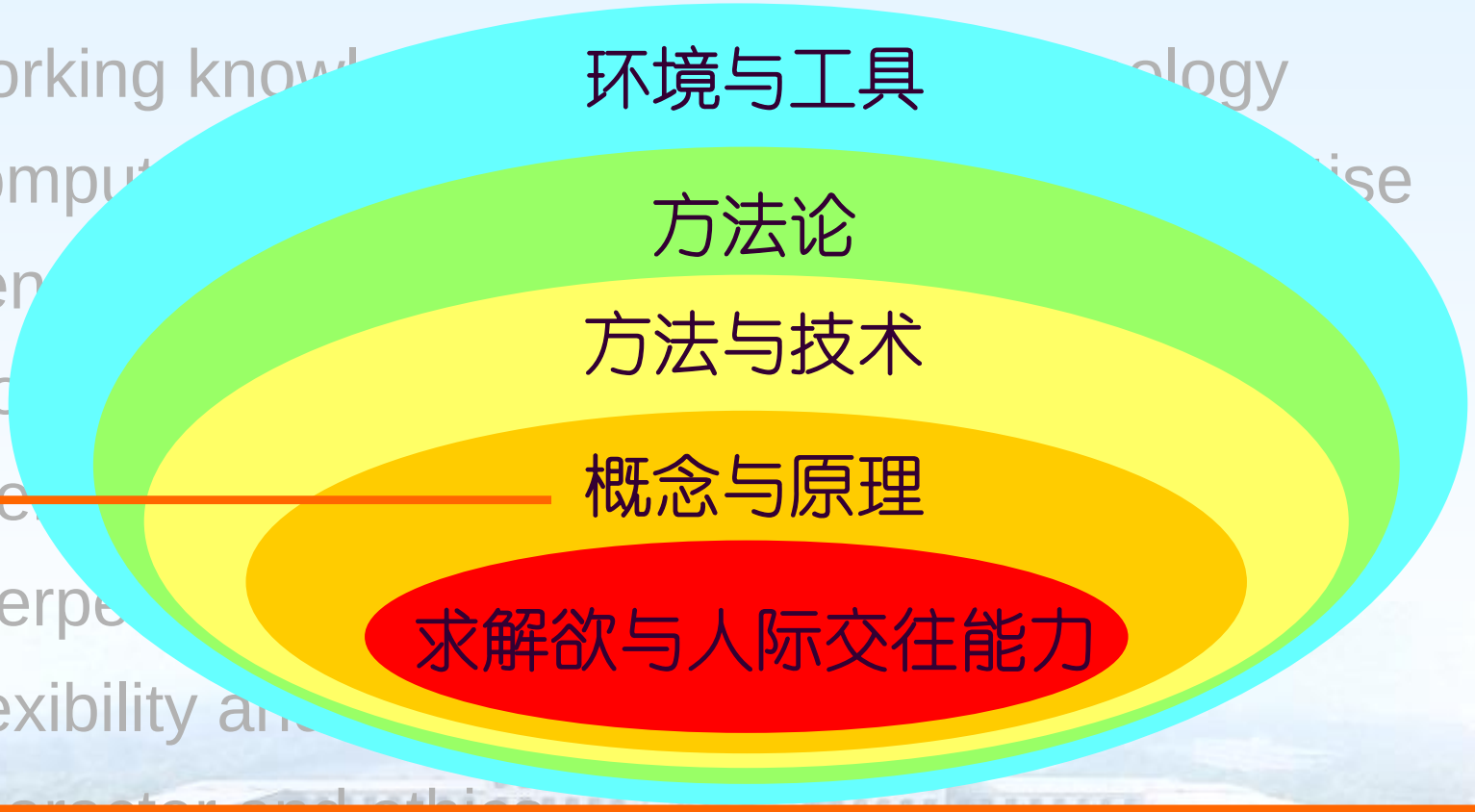
- Working knowl… …ology
- Comput… …ise
- Gen…
- Pro…
- Inte…
- Interpe…
- Flexibility an…
- Character and ethics

要有解决问题的欲望，乐于、善于与不同的人交往。

引言

Skills Required by Systems Analysts

- Working knowl… …ology
- Comput… …ise
- Gen…
- Pro…
- Inte…
- Interpe…
- Flexibility an…
- Character ethics…

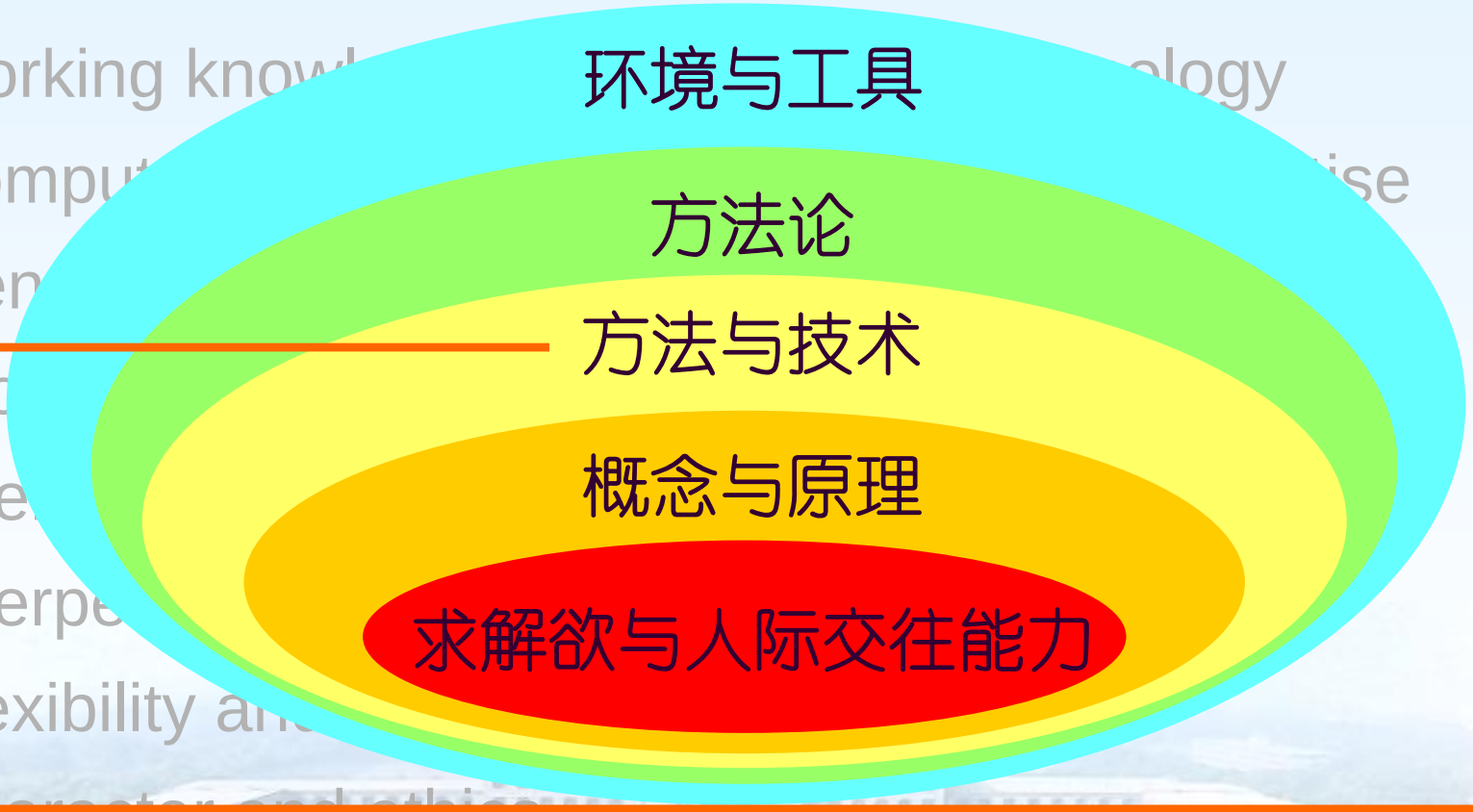


掌握系统分析与设计的基本概念和原理，许多原理几十年来并没有本质的变化。

引言

Skills Required by Systems Analysts

- Working knowl… …ology
- Comput… …ise
- Gen…
- Pro…
- Inte…
- Interpe…
- Flexibility an…

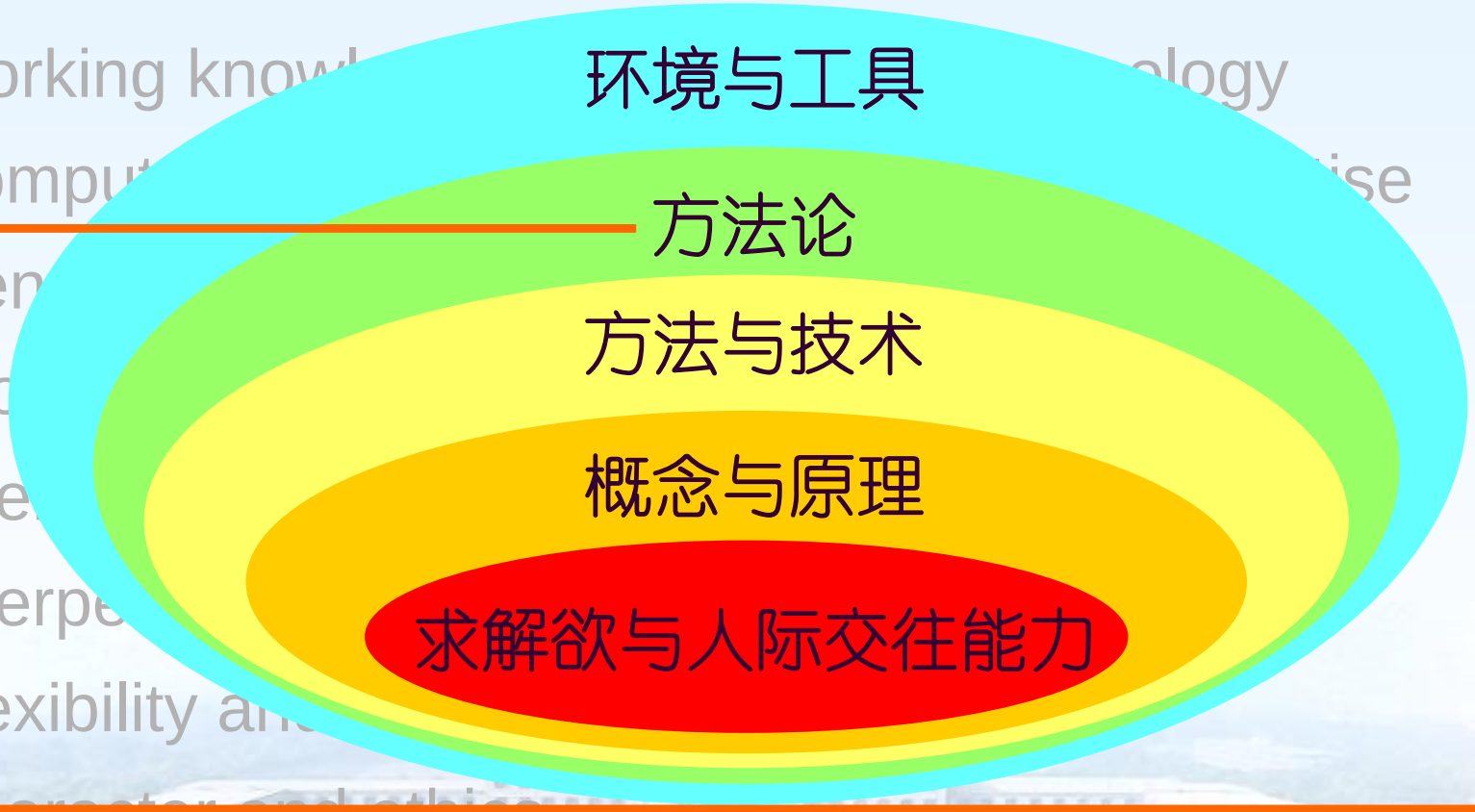


概念与原理要靠方法与技术来操作，而只有在掌握方法与技术时，才能对概念与原理有根本的认识。

引言

Skills Required by Systems Analysts

- Working knowl… …ology
- Comput… …ise
- Gen…
- Pro…
- Inte…
- Interpe…
- Flexibility an…
- Character ethics…

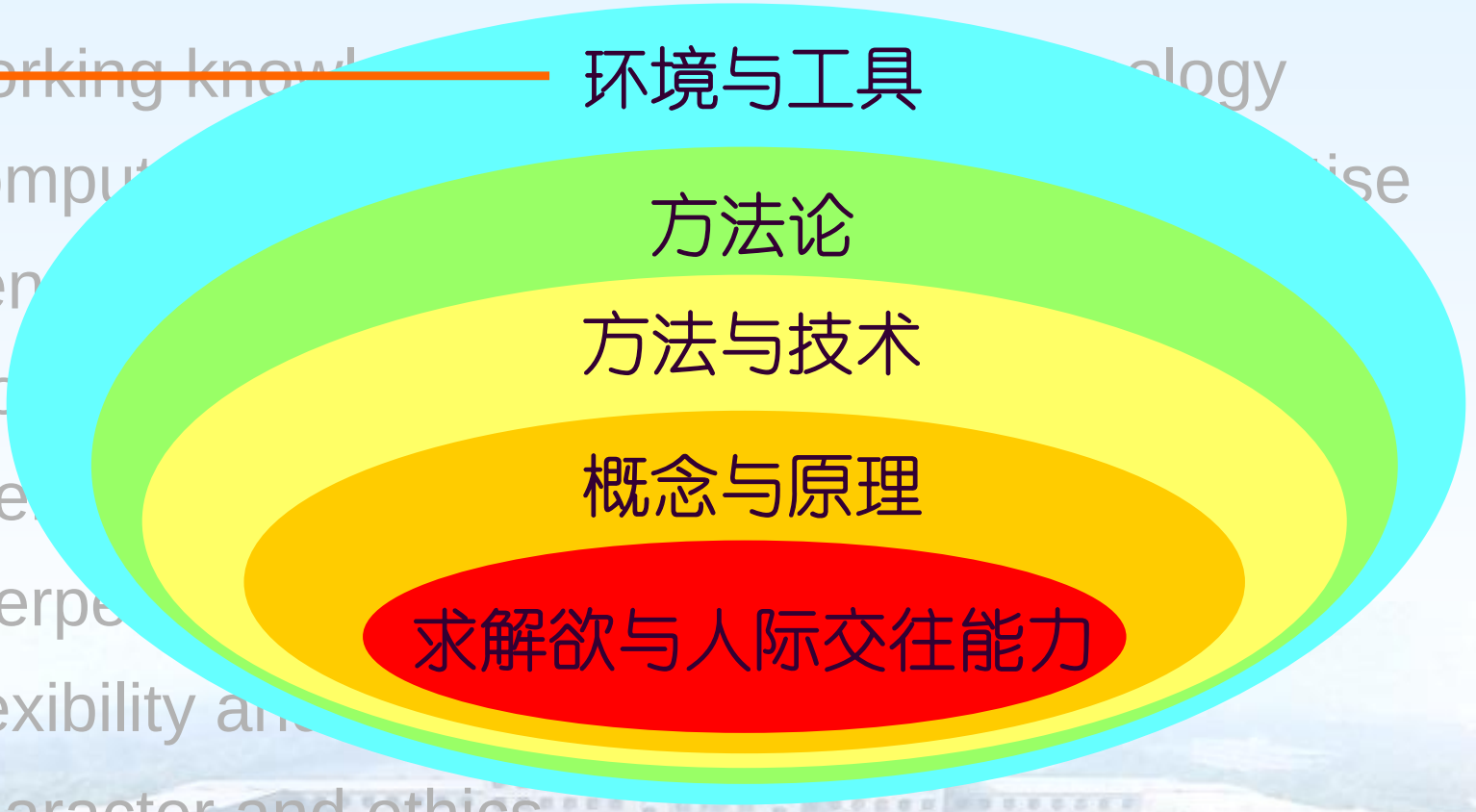


把方法与技术包装在一起就形成了方法论，其目标是通过预先选择的方法与技术，提炼出特定的问题求解策略。

引言

Skills Required by Systems Analysts

- Working knowl… …ology
- Comput… …ise
- Gen…
- Pro…
- Inte…
- Interpe…
- Flexibility an…
- Character and ethics



环境与工具用来具体地支持方法、技术和方法论。

4.1 信息系统及其基本特征

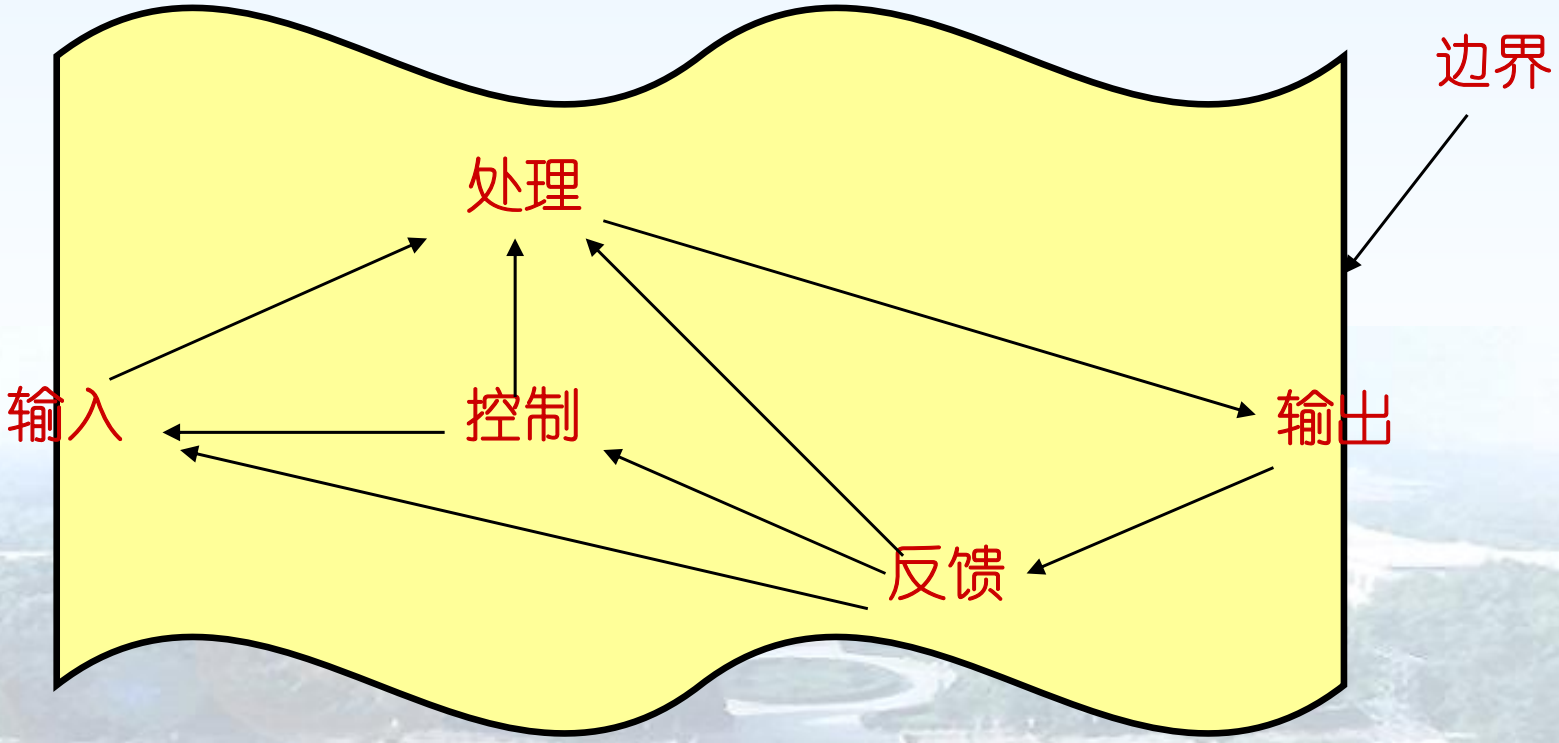
4.1.1 什么是系统?

- 一个系统是一组相互关联的部件，它们协同工作以实现一个共同的目的。
- 系统分为：自然的系统和人造的系统。
  - 自然的系统：太阳系，生态系统，……
  - 人造的系统：政府，学校，计算机，……
- 系统是递归的：
  - 一个系统可以分解成若干个子系统，也可以是另一个系统的一个子系统。

4.1 信息系统及其基本特征

4.1.1 什么是系统?

- 系统的一般模型:
  - 6个成分: 输入、处理、输出、控制、反馈、边界。



4.1 信息系统及其基本特征

4.1.2 信息系统

- 信息系统是由人来使用的、人造的系统，以帮助人来完成某项使命。
- 以下事物都是信息系统:
  - 名片夹、通信录
  - 足球彩票销售点
  - 日程安排表
  - 财务帐本
  - ……

4.1 信息系统及其基本特征

4.1.2 信息系统

- 信息系统的形状和尺寸是任意的。
- 信息系统除了系统的6个成分外，还有3个额外的成分：人、过程、数据。
  - 人以某种方式与系统中的成分（如输入、处理、输出、控制、反馈）进行交互。
  - 与信息系统的界面，通常以动作过程的形式来阐述。
  - 与信息系统的交流，通常与数据相关。

4.1 信息系统及其基本特征

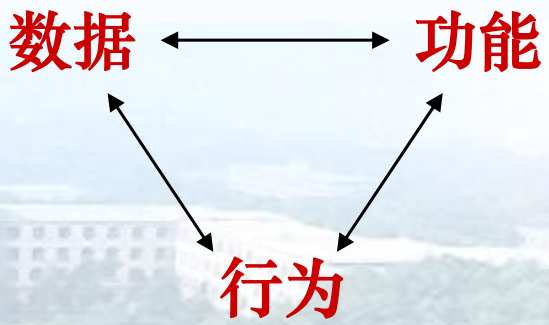
4.1.2 信息系统

- 用计算机软硬件来完成信息系统中可以自动进行或需要重复进行的处理，就是自动信息系统。
  - 我们的论题只涉及自动信息系统，因此以下简称为信息系统。

4.1 信息系统及其基本特征

4.1.3 信息系统的基本特征

- 系统输入、存储、输出的是数据;
- 系统所履行的业务活动是功能;
- 系统关于一个请求的、能够观察到的效果是行为;
- 因此，信息系统的分析与设计应准确、有效地表现这三个特征。



4.1 信息系统及其基本特征

4.1.4 信息系统的分析与设计是十分棘手的工作

- 问题的边界和结构一开始就不是定义明确的，因此用户说不清楚真正的要求和满足要求的标准;
- 问题没有唯一的正确解;
- 在信息系统开发过程中，问题本身还在动态演变;
- 技术的发展要求系统分析人员不断地学习;
- 胜任系统分析工作需要多种领域的知识和技能;
- 系统分析过程本质上是一种认知活动。

下一次课的内容

- 面向对象系统分析与设计－传统软件开发过程的特点及存在的问题
  - 为什么首先、单独讨论软件开发过程？
  - 传统的软件开发过程模型
  - 传统的软件开发过程存在的问题
- 面向对象系统分析与设计－传统软件开发方法的特点及存在的问题
  - 方法论的概念
  - 功能分解方法
  - 结构化方法
  - 信息建模方法
  - 传统开发方法的思维特征
  - 传统开发方法存在的问题

下一次课的内容（续）

- 面向对象系统分析与设计 – Rational统一开发过程（RUP）
  - 什么是RUP
  - RUP的思路：Implementing Best Practices
  - RUP的基本特征
  - RUP中受控的迭代式增量开发的二维模型
  - RUP产品
  - RUP带来的观念变化
  - 我们使用RUP的体会