Chapter 2　Software

Unit 2.5 Object-oriented Programming

备课时间：2019-10-17

词汇与词组

1. Object-oriented programming (OOP) is a programming **paradigm** that uses “**objects**” and their **interactions** to design applications and computer programs.
   - **OOP** 面向对象编程
   - **Paradigm** /ˈpærədaɪm/ 范式

     编程范式（Programming Paradigm）是某种编程语言典型的编程风格或者说是编程方式【Robert W. Floyd 1978 ACM Turing Award Lecture】
   - **Object** 对象
   - **Interaction** 交互，协作

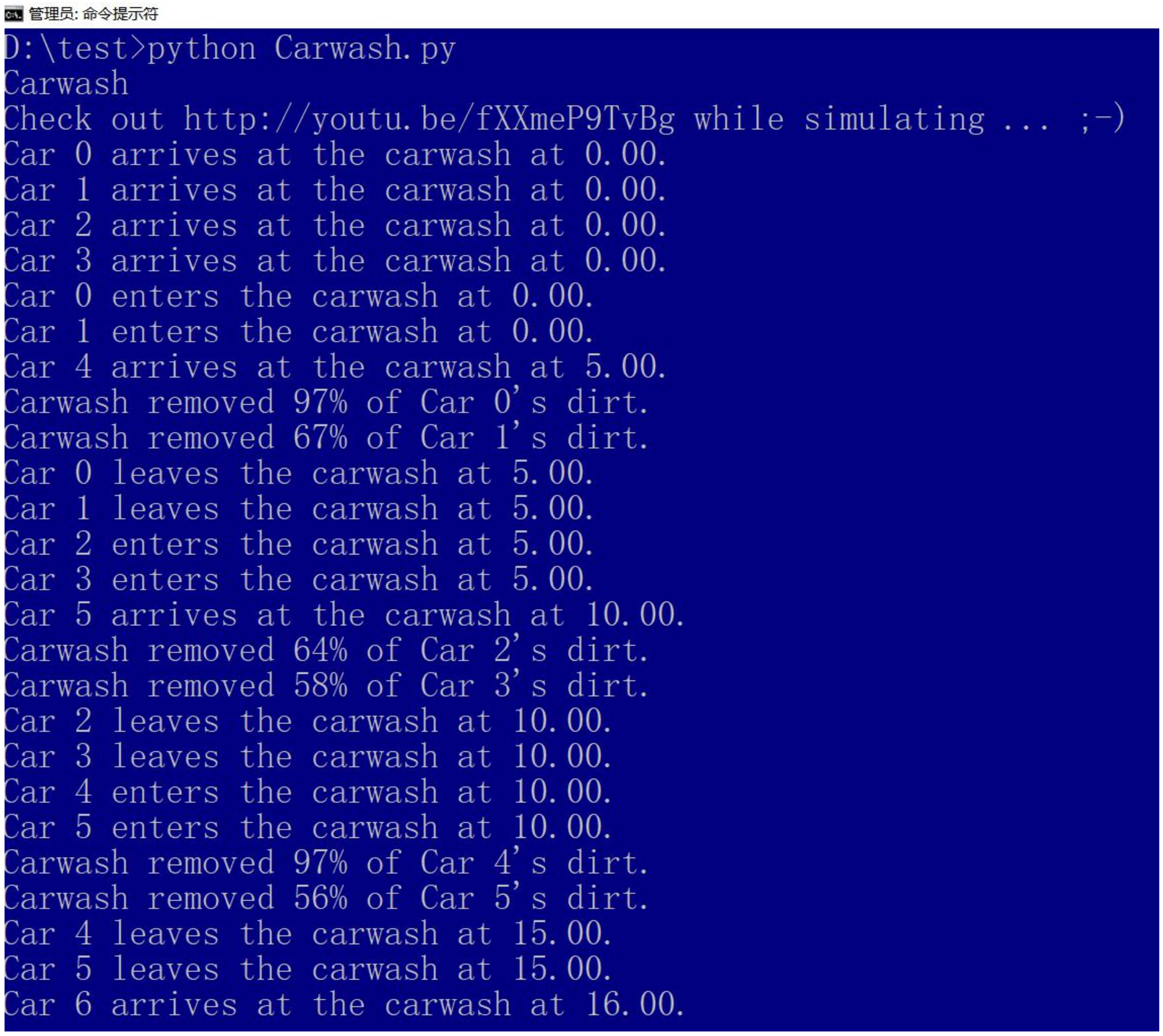
2. It is based on several techniques, including **inheritance**, **modularity**, **polymorphism**, and **encapsulation**.
   - **Inheritance** /ɪnˈherɪtəns/ 继承
   - **Modularity** /,mɒdjʊ'lærɪtɪ/ 模块化
   - **Polymorphism** /,pɒlɪ'mɔ：fɪz(ə)m/ 多态
   - **Encapsulation** /ɪnˌkæpsjuˈleɪʃn/ 封装

3. The Simula programming language was the first to introduce the concepts underlying object-oriented programming (objects, **classes**, **subclasses**, **virtual methods**, **coroutines**, **garbage collection**, and **discrete event simulation**) as a superset of Algol.
   - **Class** 类
   - **Subclass** 子类
   - **Virtual method** 虚方法
     1) In OOP, a virtual function or virtual method is an **inheritable** and **overridable** function or method for which **dynamic dispatch** is facilitated.
     2) This concept is an important part of the (runtime) **polymorphism** portion of OOP.
     3) A virtual function defines a **target function** to be executed, but the target might **not be known at compile time**.
   - **Coroutine /ˌkəuru:'ti:n/** 协程

     Coroutine is similar to subroutine/threads. The difference is once a caller invoked a subroutine/threads, it will never return back to the caller function.

     But a coroutine can return back to the caller after executing a few piece of code allowing the caller to execute some of its own code and get back to the coroutine point where it stopped execution and continue from there.

- **Garbage collection** 垃圾回收，当分配好的内存空间不再使用的时候，系统回收这部分内存空间
- **discrete event simulation** 离散事件模拟



```
D:\test>python Carwash.py
Carwash
Check out http://youtu.be/fXXmeP9TvBg while simulating ... ;-)
Car 0 arrives at the carwash at 0.00.
Car 1 arrives at the carwash at 0.00.
Car 2 arrives at the carwash at 0.00.
Car 3 arrives at the carwash at 0.00.
Car 0 enters the carwash at 0.00.
Car 1 enters the carwash at 0.00.
Car 4 arrives at the carwash at 5.00.
Carwash removed 97% of Car 0's dirt.
Carwash removed 67% of Car 1's dirt.
Car 0 leaves the carwash at 5.00.
Car 1 leaves the carwash at 5.00.
Car 2 enters the carwash at 5.00.
Car 3 enters the carwash at 5.00.
Car 5 arrives at the carwash at 10.00.
Carwash removed 64% of Car 2's dirt.
Carwash removed 58% of Car 3's dirt.
Car 2 leaves the carwash at 10.00.
Car 3 leaves the carwash at 10.00.
Car 4 enters the carwash at 10.00.
Car 5 enters the carwash at 10.00.
Carwash removed 97% of Car 4's dirt.
Carwash removed 56% of Car 5's dirt.
Car 4 leaves the carwash at 15.00.
Car 5 leaves the carwash at 15.00.
Car 6 arrives at the carwash at 16.00.
```

4. Object-oriented programming may be seen as a collection of **cooperating** objects, as opposed to a traditional view in which a program may be seen as a list of instructions to the computer.

- **Cooperate /kəʊ'ɒpəreɪt/** 合作，配合

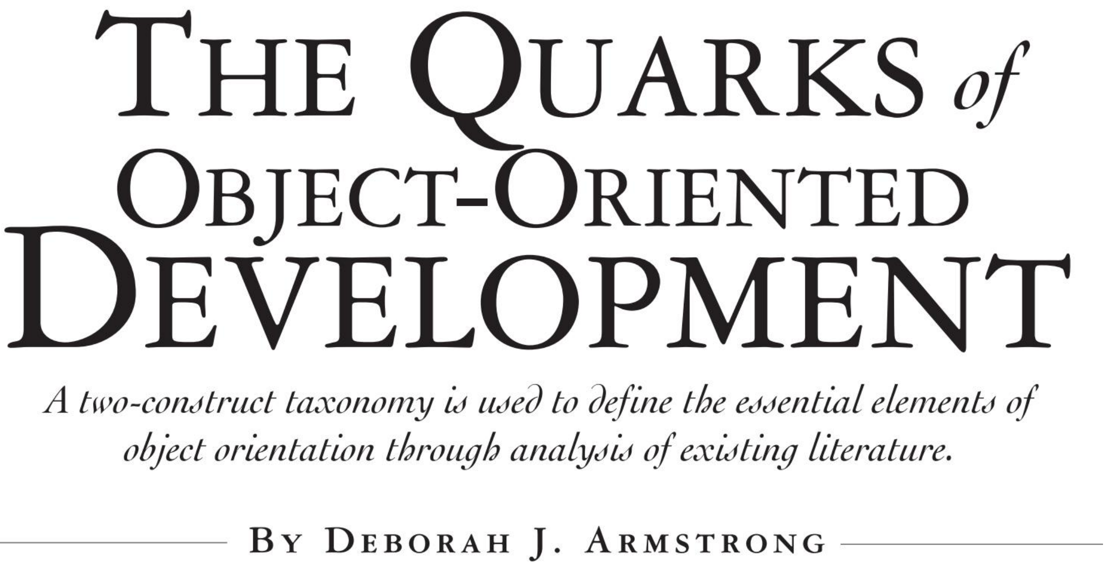
5. By way of “**objectifying**” software modules, object-oriented programming is intended to **promote** greater **flexibility** and **maintainability** in programming, and is widely popular in **large-scale** software engineering.
   - **Objectifying** 对象化
   - **Promote** 提升
   - **Flexibility** 灵活性
   - **Maintainability** 可维护性
   - **large-scale** 大规模的

6. **By virtue of** its strong **emphasis** on modularity, object oriented code is intended to be simpler to develop and easier to understand later on, **lending itself to** more direct analysis, coding, and understanding of complex situations and procedures than less modular programming methods.
   - **By virtue of** 由于
   - **Emphasis** 强调
   - **Lend itself to** 适合...,有助于...

7. fundamental concept 基本概念

8. A **survey** by Deborah J. Armstrong of nearly 40 years of computing **literature** identified a number of “quarks,”...

- ➢ **Survey** 综述，指就某**一时间内**，作者针对某**一专题**，对**大量**原始研究论文中的数据、资料和主要观点进行**归纳整理**、**分析提炼**而写成的论文
- ➢ **Literature** /ˈlɪtrətʃə(r)/ 文献 【不可数】


THE QUARKS *of* OBJECT-ORIENTED DEVELOPMENT

A two-construct taxonomy is used to define the essential elements of object orientation through analysis of existing literature.

BY DEBORAH J. ARMSTRONG

COMMUNICATIONS OF THE ACM

- ➢ ACM：美国计算机学会（Association of Computing Machinery），它是世界上第一个、也是最有影响的计算机组织，计算机领域最高奖——图灵奖是由该组织设立和颁发的。

找论文的方法：

- ➢ http://xueshu.baidu.com/
- ➢ https://dblp.uni-trier.de/db/
- ➢ https://lovescihub.wordpress.com/

9. A **class** defines the abstract characteristics of a thing (object), including the thing’s characteristics (its **attributes**, **fields** or **properties**) and the thing’s behaviors (the things it can do or methods or features).

- **class** 类
- **Attribute** is a quality or object that we attribute(**把…归于**) to someone or something. For example, the scepter is an attribute of power and statehood. 王权是权力和国家地位的象征
- **Property** is a quality that exists without any attribution. 与生俱来的
- For example, clay（黏土） has adhesive qualities; or, one of the properties of metals is electrical conductivity. 泥巴有粘性，金属有导电性
- Properties demonstrate themselves though physical phenomena **without the need attribute** them to someone or something.
- By the same token, saying that someone has masculine attributes is self-evident. 某人有肌肉，是不证自明的
- In effect, you could say that a property is owned by someone or something. 属性是属于某人或某物的

10. For example, the class of Dog would consist of **traits** shared by all dogs, such as **breed** and **fur** color (characteristics), and the ability to **bark** (behavior).

- ➢ **Trait** 特性，特点
- ➢ **Breed** /briːd/ 繁殖
- ➢ **Fur** 毛皮
- ➢ **Bark** /bɑːk/ 狗叫；尖叫

11. Also, the code for a class should be relatively **self-contained.**

- ➢ **Self-contained** 独立

A paragraph should be a **self-contained** unit, which means that you should finish off your idea or issue in one block of text before moving on to another.
一个段落应该是一个独立的单元，意思是，在继续下一个想法或者议题（争论点）之前，你应当在同一文字单元完成它们。

12. **Collectively**, the properties and methods defined by a class are called **members**.

- ➢ **Collectively** 总体地;总之
- ➢ **Member** 成员；会员

  1) **Member** variable. 成员变量

  2) A **member** of an organization such as a club or a political

party is a person who has officially joined the organization. 会员; 成员

3) Membership 资格；会员身份

13. **Object** is a particular **instance** of a class.

- ➢ **Object 对象**
- ➢ **Instance 实例**

鲁迅：《呐喊 · 风波》

这村庄的习惯有点特别，女人生下孩子，多喜欢用秤称了轻重，便用斤数当作小名。九斤老太自从庆祝了五十大寿以后，便渐渐的变了不平家，常说伊年青的时候，天气没有现在这般热，豆子也没有现在这般硬；总之现在的时世是不对了。何况六斤比伊的曾祖，少了三斤，比伊父亲七斤，又少了一斤，这真是一条颠扑不破的实例。

For instance 例如

Instantiation 实例化:指对象在类声明基础上被创建（构造）的过程

14. The class of Dog defines **all possible dogs** by listing the characteristics and behaviors they can have; the object **Lassie** is one particular dog, with particular versions of the characteristics.

- ➢ **Lassie /'læsɪ/ 灵犬莱西;**

《灵犬莱西》是英国作家艾瑞克·莫布里·奈特的作品，又称《莱西回家》。1938 年发表后成为全世界家喻户晓的儿童读本。电影上映时间：1943 导演：弗雷德·威尔科克斯
主演：伊丽莎白·泰勒/爱尔莎·兰切斯特/尼格尔·布鲁斯

　　莱西是小主人乔的爱犬，但因为乔的爸爸山姆失业了，无奈只好把狗卖给了公爵。由于想念小主人乔，莱西数次出逃，都被山姆送了回去。后来公爵带着莱西迁到了苏格兰北部的庄园，莱西又因为思念小主人而逃了出来，这一次，它跋山涉水，走了足足四百英里，历尽千山万苦才回到小主人身边。

莱西在狗舍里，由于对家的思念，它选择了出逃。它奋力地撕扯着铁丝网，纵身跃起，从铁丝网上跳了出去；由于对家的思念，它选择了回归。在遥远的北方，它从公爵的苏格兰大庄园里逃了出来，它选择了通往南方的归路没有片刻的忧郁，在回家的途中，它感到了巨大的满足，因为只有这样，它体内不安的骚动才能得以平息。

对于整个人类来说，莱西不仅是条忠诚的狗，而且它代表了一种金钱也无法夺走的尊严。

在归途中，它脚上的肉垫伤痕累累，刺扎进了肉垫间娇嫩的薄膜，化脓了。在翻滚的河流中，它在岩石上撞断了一根肋骨。肌肉和右脚关节严重挫伤。它不能赶路了，躲在荆豆下的隐蔽所里。动物是高贵的，它们不同于人。人在生病时，总要张扬，以此换取别人的同情。动物则不然，它不寻求同情，在动物眼里，任何形式的软弱都是可耻的，它只会钻进某个隐蔽所，在那里独自等待结果——康复或者死亡。

在旅途的最后，它瘫软在了地上，白茫茫的雪温柔地盖在了它的身上，它疲惫地躺在温暖的白毯下。它来到了小主人的学校门口——每天下午四点钟等待男孩的地方。疲惫已经让它连头也无力抬起来，它残破的尾巴上挂满了荆棘和苍耳。

它到了，它遵守了自小的约定……

15. In programmer **jargon**, the object Lassie is an instance of the Dog class.
  - **Jargon** 行话；术语

16. The set of values of the attributes of a particular object is called its state.

17. Within the program, using a **method** should only affect one particular object; all Dogs can bark, but you need **one particular dog** to do the barking.
  - **Method 方法（用于定义行为的）**
  - **Affect 影响**

18. “The process by which an object **sends data** to another object or asks the other object to **invoke** a method.” Also known to some programming languages as interfacing.
  - Send 发送
  - Invoke 调用
  - Interface 接口

19. “Subclasses” are more **specialized** versions of a class, which **inherit** attributes and behaviors from their parent classes, and can introduce their own.
  - **specialized** 特别的

➢ **inherit** /ɪnˈherɪt/ 继承【动词】

➢ **Introduce** 引入

20. For example, the class Dog might have sub-classes called Collie, Chihuahua, and Golden Retriever.

➢ **Collie** **/ˈkɑːli/** 柯利牧羊犬

In this case, Lassie would be an instance of the Collie subclass.

➢ **Chihuahua** **/tʃɪˈwɑːwə/** 吉娃娃

➢ **Golden Retriever** 金毛猎犬

21. Each subclass can alter its inherited traits.

➢ **Alter** **/'ɔltər/** 修改，变更

22. In fact, inheritance is an **"is-a" relationship**: Lassie is a Collie.

23. **Multiple inheritance** is inheritance from more than one ancestor class, neither of these ancestors being an ancestor of the other.

➢ Multiple inheritance 多继承

➢ Ancestor class 祖先类

Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.

```
#include<iostream>
using namespace std;

class A {
public:
  A() {
      cout << "A's constructor called" << endl;
  }
};

class B {
public:
  B() {
      cout << "B's constructor called" << endl;
  }
};

class C: public B, public A
{
public:
  C() {
      cout << "C's constructor called" << endl;
  }
};

int main() {
  C c;
  return 0;
}
```

24. Encapsulation conceals the functional details of a class from objects that send messages to it.
   - **Conceal /kənˈsiːl/** 隐藏

25. The result is that each object **exposes** to any class a certain interface——those members **accessible** to that class.
   - **Expose /ɪkˈspəʊz/** 暴露
   - **Exhibit /ɪɡˈzɪbɪt/** 展览；显示；提出（证据等）

➢ 暴露狂

①	Exhibitionism	/ˌeksɪˈbɪʃənɪzəm/
②	Flasher	/ˈflæʃə(r)/
③	Indecent Exposure	/ɪnˈdiːsnt/ /ɪkˈspəʊʒə(r)/
④	Rabies Exposure	/ˈreɪbiːz/

➢ **Accessible /əkˈsesəbl/ 可访问的**

26. The reason for encapsulation is to **prevent** clients of an interface **from** depending on those parts of the implementation that **are likely to** change **in future**, **thereby** allowing those changes to be made more easily, that is, without changes to clients.

➢ **Prevent A from B** 阻止 A 做 B

➢ **be likely to** 倾向于，很可能

➢ **In future** 从今以后，往后= **from now on**

Please be punctual in future. 今后请准时。

✧**in the future 以后，将来**

1) Who knows what will happen in the future?

2) The boy wants to become a philosopher in the future.

➢ **thereby 从而，因此**

27. Members are often specified as **public**, **protected** or **private**, **determining** whether they are **available** to all classes, sub-classes or only the defining class.

- ➢ public 公共的（所有类均可访问）
- ➢ protected 受保护的（仅同包其他类、自己以及子类可访问；希望被子类重写）
- ➢ private 私有的（仅自己可访问）
- ➢ default 默认的，同包其他类和自己可访问

  Some language go further: Java uses the **default access modifier** to restrict access also to classes in the same package.
- ➢ **determine** **/dɪˈtɜːmɪn/** 决定，确定【动词】
- ➢ **Available** **/əˈveɪləbl/** 可用的

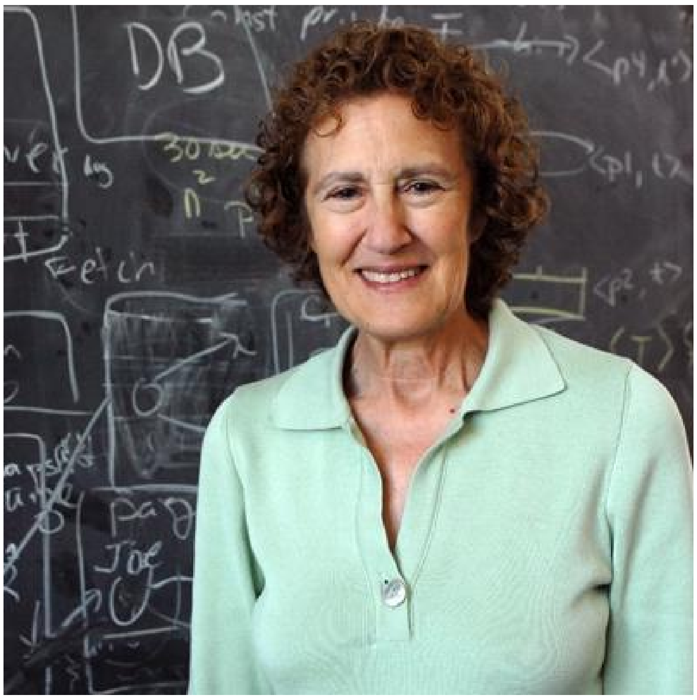
访问控制级别表

	private	default	protected	public
同一个类中	√	√	√	√
同一个包中		√	√	√
子类中			√	√
全局范围内				√

28. Abstraction is simplifying complex reality by modeling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem.

- **Abstraction** 抽象性
- **Barbara Liskov**，2008 年图灵奖得主,2004 年约翰·冯诺依曼奖得主。现任麻省理工学院电子电气与计算机科学系教授。

Lecture Video：https://amturing.acm.org/vp/liskov_1108679.cfm



面向对象型编程语言的一大革命性发展标志正是上世纪七十年代中期 CLU 语言的诞生。CLU 的设计与开发由麻省理工学院的 Barbara Liskov 负责领导，而她也是美国历史上第一位获得计算机科学博士学位的女性。通过 CLU 语言，Liskov 提出了诸多概念（或者说将其引入流行），其中包括抽象数据类型、迭代器以及并行作业等等。CLU 本身并不属于面向对象语言，因为它某些关键性的面向对象特性，例如继承。CLU 虽然从来没能得到广泛应用,但它却给其后的众多著名语言带来了巨大的影响,例如 Java、Python 以及 C++,它们或多或少地采纳了来自 CLU 的先驱性概念。

SOLID principles of object-oriented programming

1) **Single responsibility，单一职责原则（SRP）**表明一个类有且只有一个职责。一个类就像容器一样，它能添加任意数量的属性、方法等。然而，如果你试图让一个类实现太多，很快这个类就会变得笨重。

2) **Open-closed，开放封闭原则（OCP）**指出，一个类应该对扩展开放，对修改关闭。这意味一旦你创建了一个类并且应用程序的其他部分开始使用它，你不应该修改它。

3) **Liskov Substitution，里氏替换原则（LSP）**，派生的子类应该是可替换基类的，也就是说任何基类可以出现的地方，子类一定可以出现。值得注意的是，当你通过继承实现多态行为时，如果派生类没有遵守 LSP，可能会让系统引发异常。所以请谨慎使用继承，只有确定是“is-a”的关系时才使用继承。

4) **Interface Segregation 接口隔离原则（ISP）**表明类不应该被迫依赖他们不使用的方法，也就是说一个接口应该拥有尽可能少的行为，它是精简的，也是单一的。

5) **Dependency Inversion. 依赖倒置原则（DIP）**表明高层模块不应该依赖低层模块，相反，他们应该依赖抽象类或者接口。这意味着你不应该在高层模块中使用具体的低层模块。因为这样的话，高层模块变得紧耦合低层模块。

29. Abstract is also achieved through **Composition**.

- ➢ **Composition** 组合

To build the Car class, one **does not need to know the different components work internally**, but only how to interface with them, i.e., send messages to them, receive message from them, and perhaps make the different objects composing the class interact with each other.
不需知道组件内部如何工作。只需知道组件之间如何交互与配合。

30. **Polymorphism** allows you to treat **derived class** members just like their parent class’s members.

- ➢ **Polymorphism** 多态性
- ➢ **Derived class** 派生类，子类

31. They both inherit speak() from Animal, but their derived class methods **override** the methods of the parent class; this is **Overriding Polymorphism**.

- ➢ **Override** 重写，覆盖
- ➢ **Overriding Polymorphism** 重写多态性

32. The “+” operator, for example may be used to perform **integer** addition, **float** addition, list **concatenation**, or string concatenation.

- ➢ **Integer** /ˈɪntɪdʒə(r)/ 整数
- ➢ **Float** /fləʊt/ 浮点数
- ➢ **Concatenation** /kənˌkætəˈneɪʃn/ 连接，拼接

33. Any two subclasses of Number, such as Integer and Double, are expected to add together properly in an OOP language. The language must therefore **overload** the concatenation operator, “+”, to work this way.

- ➢ **Overload** 重载

34. Parametric polymorphism 参数化多态

采用参数化模板，通过给出不同的类型参数，使得一个结构有多种类型。

1) 声明与定义函数、复合类型、变量时不指定其具体的类型
2) 而把这部分类型作为参数使用，使得该定义对各种具体类型都适用
3) 参数化多态使得语言更具表达力，同时保持了完全的静态类型安全。这被称为泛型函数、泛型数据类型、泛型变量，形成了泛型编程的基础

Fundamental Concepts in Programming Languages

CHRISTOPHER STRACHEY
Reader in Computation at Oxford University, Programming Research Group, 45 Banbury Road, Oxford, UK

Abstract. This paper forms the substance of a course of lectures given at the International Summer School in Computer Programming at Copenhagen in August, 1967. The lectures were originally given from notes and the paper was written after the course was finished. In spite of this, and only partly because of the shortage of time, the paper still retains many of the shortcomings of a lecture course. The chief of these are an uncertainty of aim—it is never quite clear what sort of audience there will be for such lectures—and an associated switching from formal to informal modes of presentation which may well be less acceptable in print than it is natural in the lecture room. For these (and other) faults, I apologise to the reader.

There are numerous references throughout the course to CPL [1–3]. This is a programming language which has been under development since 1962 at Cambridge and London and Oxford. It has served as a vehicle for research into both programming languages and the design of compilers. Partial implementations exist at Cambridge and London. The language is still evolving so that there is no definitive manual available yet. We hope to reach another resting point in its evolution quite soon and to produce a compiler and reference manuals for this version. The compiler will probably be written in such a way that it is relatively easy to transfer it to another machine, and in the first instance we hope to establish it on three or four machines more or less at the same time.

The lack of a precise formulation for CPL should not cause much difficulty in this course, as we are primarily concerned with the ideas and concepts involved rather than with their precise representation in a programming language.

Keywords: programming languages, semantics, foundations of computing, CPL, L-values, R-values, parameter passing, variable binding, functions as data, parametric polymorphism, ad hoc polymorphism, binding mechanisms, type completeness

下载地址：http://www.cs.cmu.edu/~crary/819-f09/Strachey67.pdf

下载上述论文，并将论文（总共 39 页）翻译成中文，每个学生翻译一部分，由学习委员整合成完整的（中文）论文（WORD 格式）

作为期末考查

最后一次课交