



Chap8 Object-oriented

第8章 面向对象程序设计

Department of Computer Science and Technology
Department of University Basic Computer Teaching
Nanjing University

```
','.join(['Amy', 'John', 'Tom'])
```

```
fp.read()
```

8.1

面向对象程序设计基本概念

程序设计范式 (paradim)

4



面向
过程
程序
设计



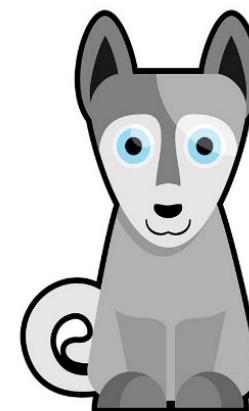
面向
对象
程序
设计



函数
式设
计

8.1.1 面向对象程序设计

- 对象（实例）
 - 由数据及能对其实施的操作所构成的封装体
- 类
 - 类描述了对对象的特征（数据和操作）



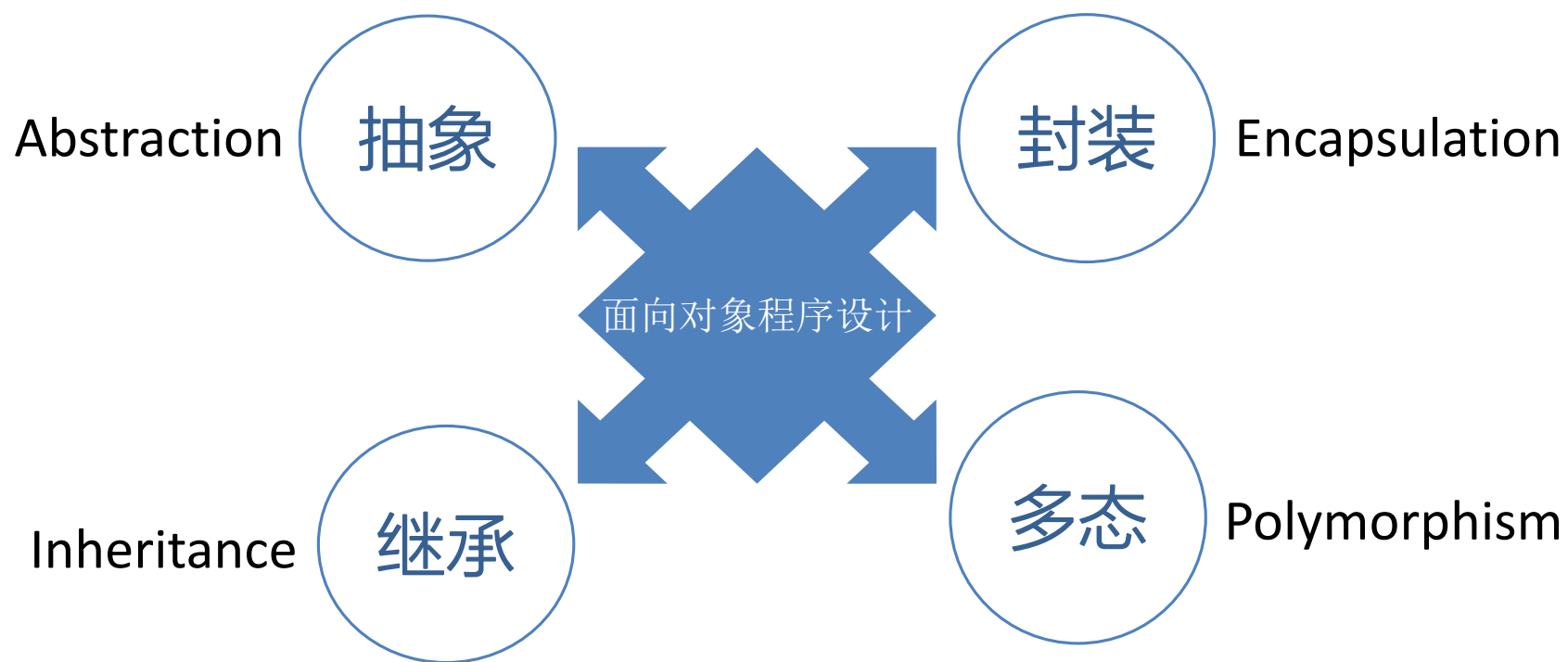
- **面向对象程序设计**：以数据为中心的程序设计方式
 - 隐藏数据细节
 - 更接近自然思考方式
- **面向过程程序设计**：以功能为中心的程序设计方式
 - 需要数据细节

```
>>> x = [4, 2, 7, -1, 3]
>>> x.sort()
>>> x
[-1, 2, 3, 4, 7]
```

[4, 2, 7, -1, 3]构建了一个列表对象，x是对其的引用，由于列表对象允许使用sort()方法进行排序，因此通过x.sort()的调用将x引用的对象进行排序

8.1.2 面向对象程序设计的基本特征

面向对象程序设计 (OOP)



面向对象程序设计的基本特征

11

- **抽象** (Abstraction) 与**封装** (Encapsulation)
 - **抽象**是指对现实世界问题和实体的本质表现；
问题分解成数据和数据上的操作
 - **封装**是将程序具体的实现细节进行隐藏的一种
机制
 - **抽象**是抽取现实问题和实体的本质，**封装**是将
这些本质包装起来进行内部实现

Abstraction



Encapsulation

- **继承 (Inheritance)**
 - 新创建的类的一些特征（包括属性和方法）
可以从其他已有的类获得
 - 子类继承父类的所有属性和方法，允许修改
或添加其他的特征，父类保持不变
 - 提高程序设计的代码复用性

Inheritance

- **多态** (Polymorphism) 与**绑定** (Binding)

- **多态**指一个事物有多种不同的解释，根据传递参数的不同执行不同的函数或操作不同的代码
- **绑定**是指在具体某次使用多态元素时确定使用的是哪一种形式

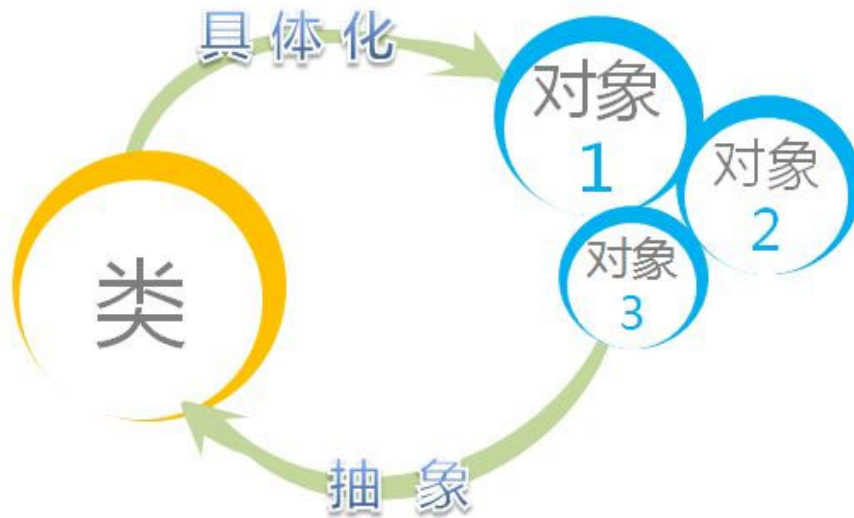
Polymorphism



Binding

8.2

类与对象



- 类是对象的特征抽象
- 对象是类的具体化

8.2.1 类

类的定义和方法1



`class` `ClassName`:

② `"类文档字符串"`

③ 类体

- ① **类名**, 类的名称
- ② **类文档字符串**, 提供查询时的帮助信息
- ③ **类体**, 定义一些类的属性和方法

类的定义和方法1

18

File

```
class Dog:
    '''define Dog class'''
    def greet(self):
        print('Hi')
```

File

```
class MyDate:
    '''
    this is a very simple example class
    '''
    pass
```

命名空间

类的定义和方法2

19



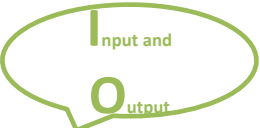
```
class ClassName(1object2):  
    """类文档字符串"""  
    类体
```

- ① **父类**, 可选, 指定从某个已定义的类继承
- ② **object**, 万类之源

类的定义与方法2

20

- 定义类的属性
- 定义类的函数（方法）



```
>>> Dog.counter  
0
```



```
class Dog(object):  
    '''define Dog class'''  
    counter = 0  
    def greet(self):  
        print('Hi')
```

例8-1 学院人员信息类

- 人员的信息包括姓名、年龄
- 可以通过方法增删改和输出



Filename: Prog8-1.py

class PersonInfo:

'''define PersonInfo Class'''

def person(self, dep):


self.dep = dep

self.num = 0

self.plist = []

例8-1 学院人员信息处理


- 添加方法

 *# Filename: Prog8-1.py*

```
class PersonInfo:
    '''define PersonInfo Class'''
    def person(self, dep):
        ...
    def insertp(self, name, age):
        for x in self.plist:
            if name in x:
                print("{0} already in list".format(name))
                return False
        self.num += 1
        self.plist.append([name,age])
        return True
```

例8-1 学院人员信息处理


- 删除方法

 *# Filename: Prog8-1.py*

```
class PersonInfo:
    '''define PersonInfo Class'''
    def person(self, dep):
        ...
    def delp(self, name):
        for x in self.plist:
            if name in x[:]:
                print("Delete {0}".format(name))
                self.plist.remove(x)
                self.num -= 1
                return True
        print("{0} not in list".format(name))
        return False
```

例8-1 学院人员信息处理

- 查询方法

 *# Filename: Prog8-1.py*

```
class PersonInfo:
    '''define PersonInfo Class'''
    def person(self, dep):
        ...
    def searchp(self, name):
        for x in self.plist:
            if name in x:
                print("{0} in list".format(name))
                print(x)
                return True
        print("{0} not in list".format(name))
        return False
```


例8-1 学院人员信息处理

File

Filename: Prog8-1.py

class PersonInfo:

'''define PersonInfo Class'''

def person(self, dep):

...

def insertp(self, name, age):

...

def delp(self, name):

...

def searchp(self, name):

...

def printplist(self):

for x in self.plist:

print(x)

- 输出方法

8.2.2 实例

- 类实例化的形式如下

```
变量 = 类名(<参数>)
```

- 创建实例后，可以使用实例调用方法
- 类方法的第一个参数总是self，指向实例本身

```
>>> x = Dog()
```

```
>>> x.greet()
```

```
Hi
```



Filename: Prog8-2.py

```
class Dog(object):  
    "define Dog class"  
    def setName(self, name):  
        self.name = name  
    def greet(self):  
        print('Hi')  
if __name__ == '__main__':  
    x = Dog()  
    x.setName("Paul")  
    print(x.name)
```

Python自动将对
象x作为第一个
参数传入方法中

例8-3 人员信息类的运用

File

```
# Filename: Prog8-3.py
..... # 类定义见例8.1
if __name__ == '__main__':
    cs = PersonInfo()
    cs.person('CS')
    cs.insertp('WangTian', 18)
    cs.insertp('ZhangWei', 20)
    cs.insertp('LiJianGuo', 40)
    print("There are {0} people in
          Dep.{1}".format(cs.num,cs.dep))
    cs.printplist()
    cs.searchp('ZhangWei')
    cs.delp('LiJianGuo')
    cs.printplist()
```

Input and Output

```
There are 3 people in Dep.CS
['WangTian', 18]
['ZhangWei', 20]
['LiJianGuo', 40]
ZhangWei in list
['ZhangWei', 20]
Delete LiJianGuo
['WangTian', 18]
['ZhangWei', 20]
```

8.2.3 `__init__()`与`__del__()`方法

__init__() 方法

- __init__()方法永远会在对象创建完成后被Python自动调用
- 和其他方法一样，实例对象本身会作为self参数传递
- 是在对象创建后被Python自动调用的第一个方法



Filename: Prog8-4.py

```
class Dog(object):  
    '''define Dog Class'''  
    def __init__(self, name):  
        self.name = name  
    def greet(self):  
        print('Hi')  
if __name__ == '__main__':  
    dog = Dog("Paul")  
    print(dog.name)
```

__init__()方法调用的时机

Step 1

- 系统创建对象

Step 2

- 检查是否实现__init__()

Step 3

- 调用__init__(), 并且把实例自身作为self参数传入

Step 4

- 对象创建完成

__init__() 举例

File

Filename: Prog8-5.py

class PersonInfo:

'''define PersonInfo Class'''

def __init__(self, dep): *#将person()方法改写成__init__()方法*

self.dep = dep

self.num = 0

self.plist = []

..... *# 其他代码与例8.1一样*

if __name__ == '__main__':


cs = PersonInfo('CS')

..... *#其他代码及运行结果与例8.2一样*

例8.3 使用__init__()方法改写例8.1中定义的人员信息类。

__del__()方法

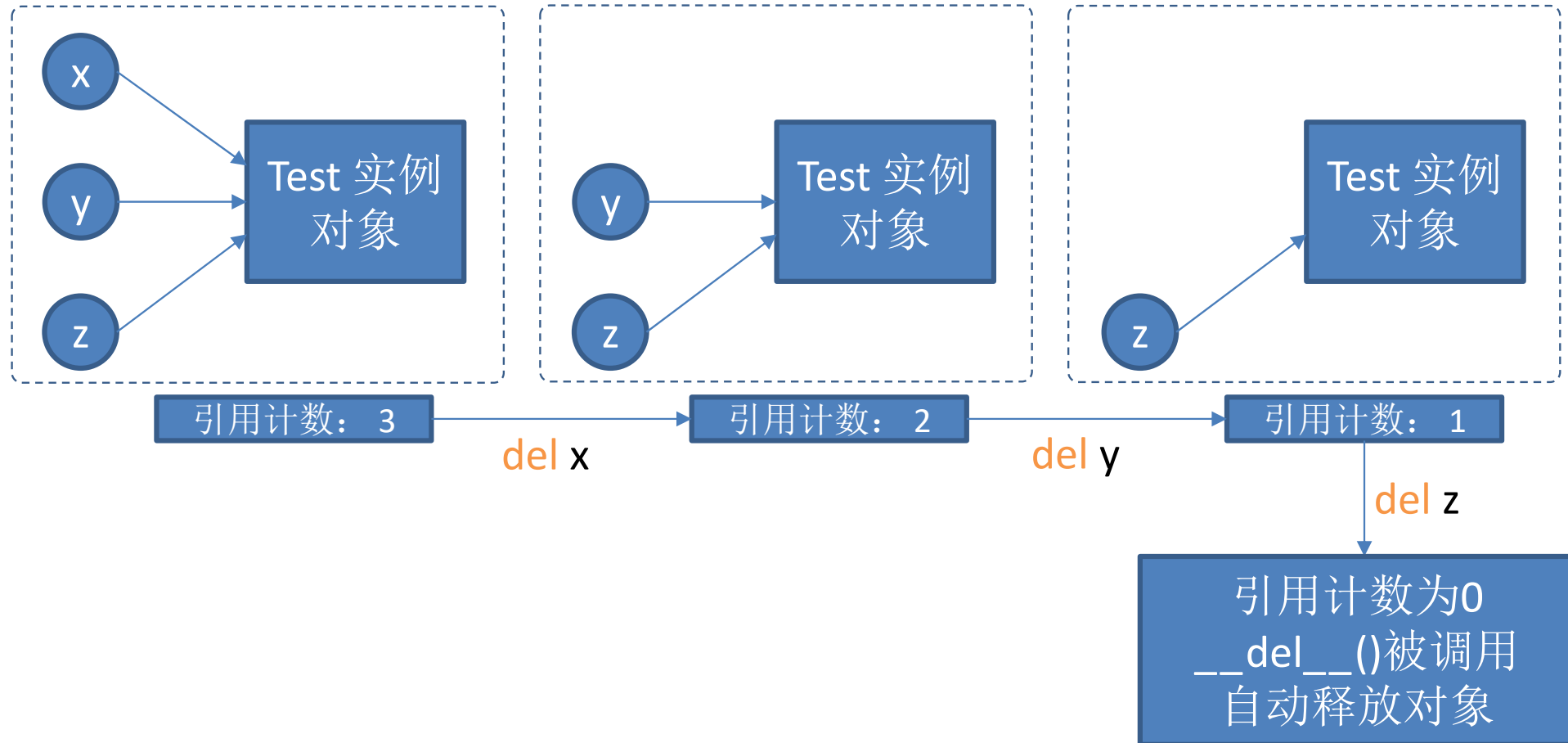
- 当引用计数减少到0的时候, Python会自动释放对象
- 程序执行退出或者显式调用 del 都会减少引用计数
- 在Python自动释放对象之前 最后一个调用的方法

 Source

```
>>> class Test:
    def __init__(self):
        print("initialized")
    def __del__(self):
        print("deleted")

>>> x = Test()
initialized
>>> y = x
>>> z = y
>>> del x
>>> del y
>>> del z
Deleted
```

引用计数



8.2.4 实例属性与类属性

实例属性 (Instance Attributes)

- 实例属性创建时间: 实例创建时或者实例创建之后
- 所有实例属性保存在名为 `__dict__` 的内嵌属性里



```
>>> class Date:  
    pass
```

```
>>> curDate = Date()  
>>> curDate.month = 6  
>>> curDate.day = 1  
>>> curDate.__dict__  
{ 'day': 1, 'month': 6 }
```



```
class Dog(object):  
    "define Dog class"  
    def greet(self):  
        print("Hi")  
  
if __name__ == '__main__':  
    x = Dog()  
    x.name = "Paul"  
    print(x.__dict__)
```

Output:
{'name': 'Paul'}

类属性 (Class Attributes)

- 类的数据属性（静态成员）仅仅是所定义的类的变量
- 在类创建后被使用
- 可以由类中的方法来更新，也可以在主程序中更新
- 类属性和实例无关，修改类属性需要使用类名
- 通常用来保存和类相关的值，例如对象计数器



```
>>> class TestII:
        data = 10

>>> TestII.data
10
>>> TestII.data += 20
>>> TestII.data
30
```

类属性应用举例

F_{ile}

Filename: Prog8-7.py

```
class Dog(object):  
    '''define Dog class'''  
    counter = 0  
    def __init__(self, name):  
        self.name = name  
        Dog.counter += 1  
    def greet(self):  
        print("Hi, I am {s}, my number is  
              {d}".format(self.name, Dog.counter))  
if __name__ == '__main__':  
    x1 = Dog("Zara")  
    x1.greet()  
    x2 = Dog("Paul")  
    x2.greet()
```

I_{nput and} O_{utput}

Hi, I am Zara, my number is 1
Hi, I am Paul, my number is 2

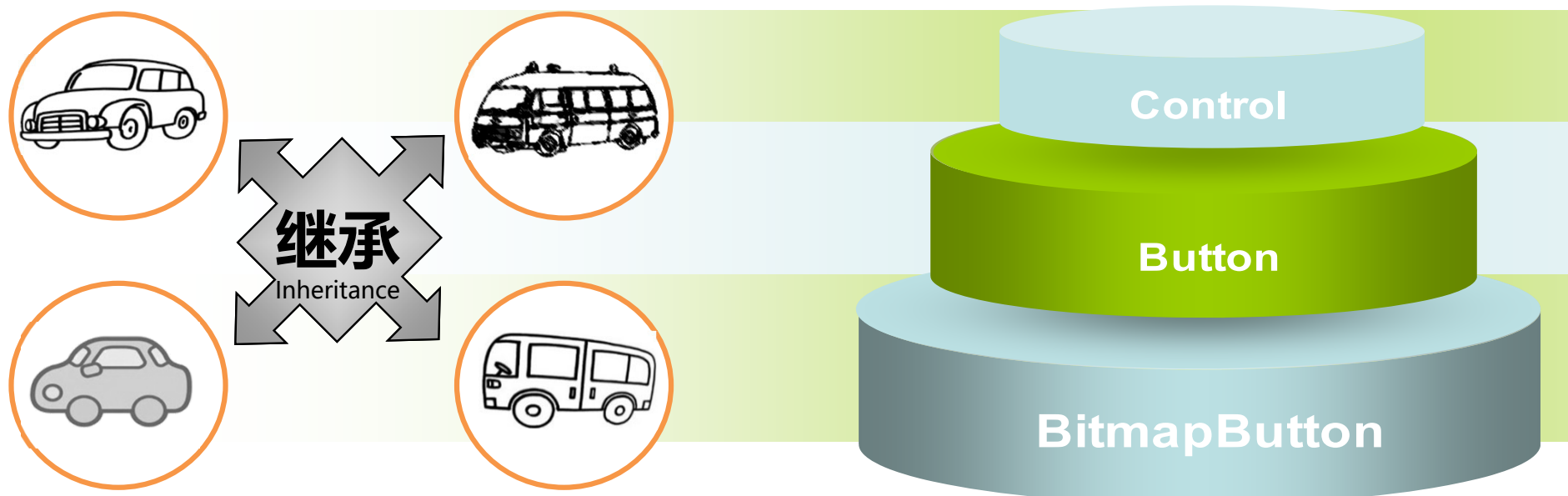
类属性 *counter* 被用作追踪已创建实例的计数器

8.3

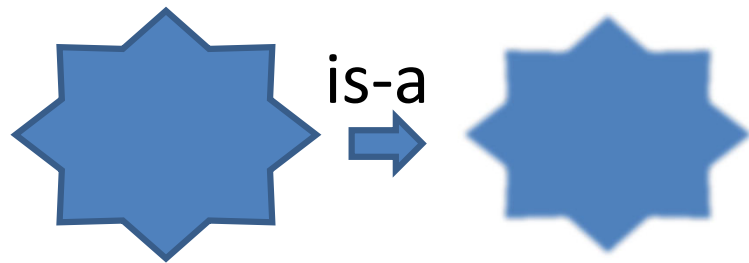
继承、重写与其他

父类 (基类) 子类 (派生类)

42



更为具体和细化的特例



包含已有的类的数据、方法



8.3.1 子类的创建与继承

子类的定义

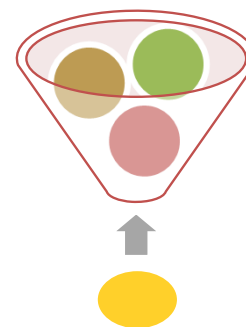
45

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    '''类文档字符串'''  
    类体
```

单
继
承



多
继
承



子类的定义

F_{ile}

Filename: Prog8-9.py

```
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am {:s}, my number is
              {:d}.".format(self.name, Dog.counter))
```

```
class BarkingDog(Dog):
    "define subclass BarkingDog"
    def bark(self):
        print("barking")

if __name__ == "__main__":
    x = BarkingDog("Zoe")
    x.greet()
    x.bark()
```

8.3.2 重写

重写(overriding)

48

- 重写：子类改写父类的方法，从而部分地改变父类的行为
- 包括运算符、构造器在内的方法都可以被重写
- 重写父类方法的时候，父类方法中定义的操作不会被自动调用

子类定义举例和重写

49



Filename: Prog8-10.py

```
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am {:s}, my number is
              {:d}.".format(self.name, Dog.counter))
```

```
class BarkingDog (Dog):
    "define subclass BarkingDog"
    def greet(self):
        "initial subclass"
        print("Woof! I am {:s}, my number is
              {:d}.".format(self.name, Dog.counter))
    if __name__ == '__main__':
        x = BarkingDog("Zoe")
        x.greet()
        Dog.greet(x)
```

子类定义举例和重写

50

File

Filename: override2.py

```
class Dog(object):  
    "define Dog class"  
    def __init__(self, name):  
        self.name = name  
    ...
```

```
class BarkingDog (Dog):  
    "define subclass BarkingDog"  
    def __init__(self, name):  
        super().__init__(name)  
        print("My name is", self.name)
```

重定义 (redefine)

51

- 子类重写父类方法时，要求参数保持一致，若要修改参数并改变功能，则称为重定义

子类__init__()方法重定义

52



```
# Filename: Prog8-11.py
class Dog(object):
    ... # Prog8-10.py
class BarkingDog (Dog):
    "define subclass BarkingDog"
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color
```

```
def greet(self):
    "initial subclass"
    print(Woof! I am {:s} {:s}, my number
is {:d}.".format(self.color, self.name,
Dog.counter))
if __name__ == '__main__':
    x = BarkingDog("Zoe", "black")
    x.greet()
```

*重载 (overloading)

53

- 重载指的是有相同的方法名但参数列表不同，根据参数类型和个数选择方法，一般在同一个类中实现，Python由于是动态语言且有可变长参数等特殊类型的参数，因此不显式支持普通的函数重载，但可以进行运算符重载。

*运算符重载

54

运算符	关联方法
+	C.__*add__(self, obj)
-	C.__*sub__(self, obj)
*	C.__*mul__(self, obj)
/	C.__*div__(self, obj)
%	C.__*mod__(self, obj)
//	C.__*floordiv__(self, obj)
**	C.__*pow__(self, obj[,mod])

8.3.3 访问控制

- 默认情况下，Python 类的成员属性与方法都是公共的
- 提供“访问控制符”来限定成员函数的访问

__ # (双下划线)

限定属性和方法在类内部可见，防止父类与子类中的同名冲突



```
>>> class P:
    def __init__(self, name):
        self.__name = name
```

```
>>> x = P('John')
```

```
>>> x.__name
```

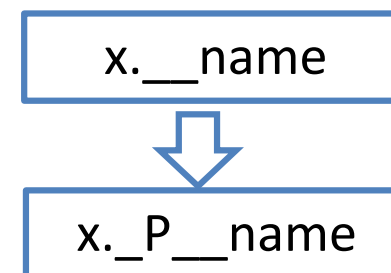
Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

x.__name

AttributeError: 'P' object has no attribute '__name'

为什么出错?



8.4

*常用类和实例相关内建函数

常用类和实例相关内建函数

59

<code>issubclass(classa, classb)</code>
<code>isinstance(obj, classn)</code>
<code>hasattr(obj, attr)</code>
<code>setattr(obj, attr, val)</code>
<code>getattr(obj,attr[,default])</code>
<code>delattr(obj, attr)</code>
<code>vars(obj = None)</code>
<code>dir(obj = None)</code>

- 面向对象程序设计基本概念
- 面向对象程序设计的特征
- 类与对象
- 继承、重写与其他
- 常用类和实例相关内建函数

