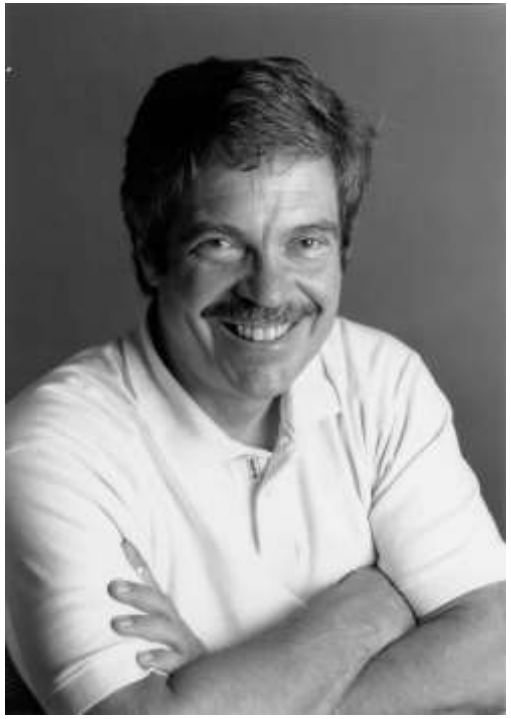
面向对象编程（1）

什么是面向对象编程

- 对象：数据和操作的封装
  - 内部状态
    - 实例变量（instance variables）、字段（fields）、成员变量（member variables）、属性（attributes）…
    - 往往外部不可见，但有些语言中可以是public的
  - 外部可见操作
    - 方法（methods）、成员函数（member functions）

- 面向对象编程
  - 每个对象都是一个独立个体，对象间通过发送消息交互
    - 对象收到消息后，可以完成相应任务，修改自己的内部状态，并把结果返回给发送方，完成任务过程中也可能发送消息给其他对象
  - “server-oriented”或者 “message-oriented”

OOP to me means only **messaging**, local retention and protection and **hiding of state-process**, and **extreme late-binding** of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I'm not aware of them.



Alan Kay

Smalltalk主要发明者
2003年图灵奖得主

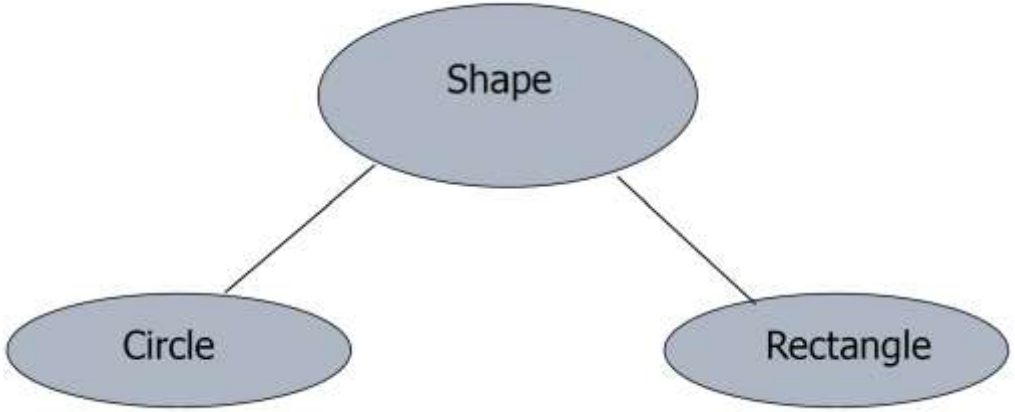
面向对象的四个核心概念

- 动态派遣（dynamic dispatch）
- 封装
- 多态（子类型）
- 继承（有较大争议）

OO编程的程序结构

- 类：对象的模板/类型/接口
  - 实例变量/方法，类（static）变量/方法
  - 实例方法调用：o.m(e1, ..., en)
    - 向对象o发送消息m(e1, ..., en)
    - 实际等同于过程式语言中的函数调用 m(o, e1, ..., en)
    - `x.add(y)` vs. `add(x, y)`
- 继承
  - 子类复用父类的方法，避免代码重复
  - 也可以重新实现父类的方法（override）
- 子类型
  - 子类型的对象天然也是父类型的对象
  - 提供多态能力

实例：几何图形库



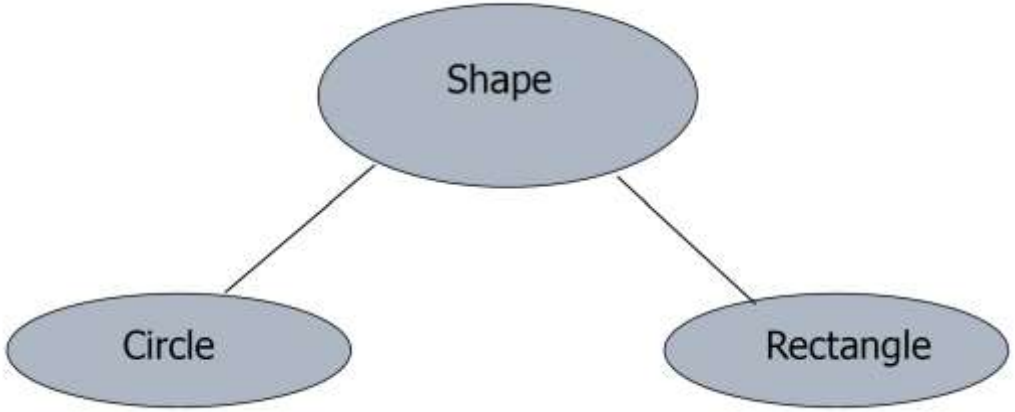
- 通用概念：形状（Shape）
- 具体形状：矩形（Rectangle）和圆（Circle）
- 共性概念
  - center、move、rotate、print
- 允许扩展
  - 具体形状（矩形和圆）中**添加新的概念**（成员变量或方法），或者重新定义上述共性概念
  - **添加新的具体形状**
- 易扩展性是面向对象编程的一个重要理念

实例：几何图形库

```
class Shape {
    func move(x: Int, y: Int): Unit {
        print("Shape moved to ($x, $y)")
    }
    // Other common methods for shapes
}

class Circle <: Shape {
    func move(x: Int, y: Int): Unit {
        print("Circle moved to ($x, $y)")
    }
    // Other common methods for circles
}

class Rectangle <: Shape {
    func move(x: Int, y: Int): Unit {
        print("Rectangle moved to ($x, $y)")
    }
    // Other common methods for rectangles
}

let s1: Shape = Circle()
let s2: Shape = Rectangle()
s1.move(10, 20) // Should print "Circle moved to (10, 20)"
s2.move(30, 40) // Should print "Rectangle moved to (30, 40)"
```

继承和override

动态派遣



循环处理Shape队列中的每一个元素：

从队列中取出形状s
执行操作s.move(10, 20)

队列中实际存放的形状可以是Circle或Rectangle

根据实际形状，自动调用相应的move方法

面向对象语言发展历史 —— Simula

- Simula 67: 第一个OO语言
- 最初用于仿真
  - 后扩展为通用编程语言
- 基于Algol 60扩展
- 1977年后标准化为Simula（不再带67）
- 影响了后续多种语言的设计
  - Smalltalk
  - C++
  - Java
  - C#
  - …

面向对象语言发展历史 —— Simula

- 前身：Simula-1
  - 1962年发明与挪威计算中心（Norwegian Computing Center）
- 发明人
  - Ole-Johan Dahl, Bjørn Myhrhaug, and Kristen Nygaard
- 最初设计
  - 受到Tony Hoare关于data type的想法的影响
  - 在Algol 60基础上引入类和前缀（prefix，实现子类型）
- Nygaard
  - 运筹学专家、政治活动家
  - 希望语言能够描述社会和工业系统，希望普通人也能理解政治系统的变化
- Myhrhaug 和 Nygaard
  - 计算机专家，关心通用编程

Simula中的面向对象设计

- 类：返回其活动记录指针的**函数**（过程）
- 对象
  - 每次调用类之后产生的活动记录
- 对象访问
  - 用“点运算符”（dot notation）访问对象的成员变量/函数
  - o.var
- 内存管理
  - 采用垃圾收集来自动管理

示例

func

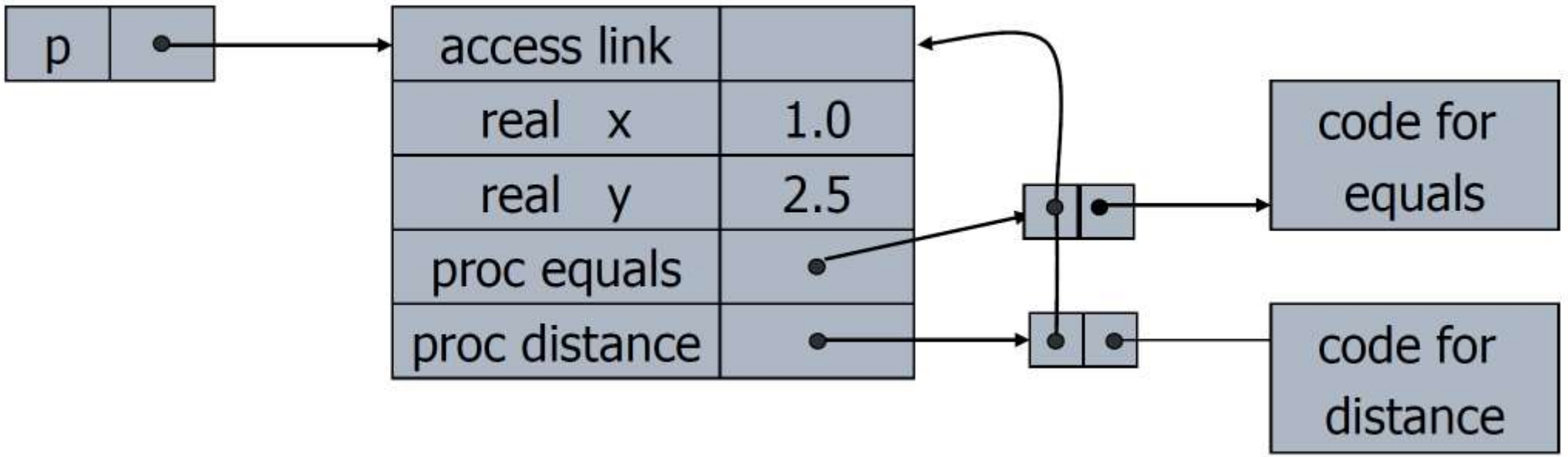
```
class Point(x,y); real x,y;
    begin
      boolean procedure equals(p); ref(Point) p;
        if p =/= none then
            equals := abs(x - p.x) + abs(y - p.y) < 0.00001
      real procedure distance(p); ref(Point) p;
        if p == none then error else
            distance := sqrt(( x - p.x )**2 + (y - p.y) ** 2);
end ***Point***

p :- new Point(1.0, 2.5);
q :- new Point(2.0,3.5);
if p.distance(q) > 2 then ...
```

formal p is pointer to Point

uninitialized ptr has value none

pointer assignment

对象格式



对象 和 **闭包** 的关系

对象为活动记录

Access link用于访问全局变量

实例2

```
class Line(a,b,c); real a,b,c;
    begin
      boolean procedure parallelto(l); ref(Line) l;
          if l =/= none then  parallelto := ...
      ref(Point) procedure meets(l); ref(Line) l;
          begin real t;
              if l =/= none and ~parallelto(l) then ...
          end;
      real d;   d := sqrt(a**2 + b**2);
      if d = 0.0 then error else
          begin
              d := 1/d;
              a := a*d;  b := b*d;  c := c*d;
          end;
end *** Line***
```

Local variables

line determined by ax+by+c=0

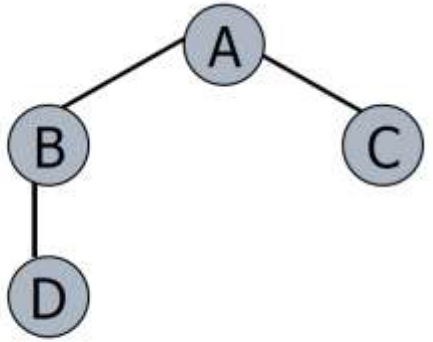
Procedures

Initialization: "normalize" a,b,c

Simula中的继承

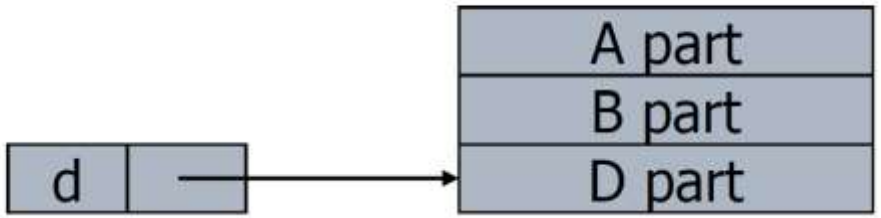
- 类的声明前面可以加上父类的名字作为前缀

```
class A
A class B
A class C
B class D
```



- 子类的对象是父类对象和子类对象自身的拼接

```
d :- new D(...)
```



Simula主要特性总结

- 类、对象、继承（prefixing）、子类型、虚方法（virtual methods）…
- 协程（Coroutine）
- 垃圾收集（GC）

- 缺失特性
  - 封装、self/super、类成员变量（静态变量）、异常

面向对象语言发展历史 —— Smalltalk

- 首次提出面向对象概念
- 在Xerox PARC 开发
- 纯面向对象语言，一切皆对象
  - 类（class）也是对象
  - 所有操作变现为“向对象发送消息”
  - 非常灵活和强大
    - 类似于Lisp的“一切皆list”的思想，但更加强大
    - 例如：对象能够发现它收到一条自己不能理解的消息（不支持的消息/方法），并现场决定如何回应

主要开发动机和核心应用——Dynabook

- 概念主要由Alan Kay提出
  - Smalltalk项目的愿景领袖和命名者
- 小型便携计算机
  - 在1970年代，是一种革命性概念
    - 当时，所谓的mini-computer往往放在专门的房间里，一台机器由十余人共享
  - 一种可以在飞机上飞行旅途中使用的计算机
- 对Smalltalk语言的影响
  - 语言同时作为编程语言和操作系统界面
  - 为“非程序员”设计
  - 语法为语言相关的编辑器设计

程序示例：Point

程序写成表格形式：

class name	Point
super class	Object
class var	pi
instance var	x y
class messages and methods	
⟨...names and code for methods...⟩	
instance messages and methods	
⟨...names and code for methods...⟩	

程序示例：类消息和方法

Three class methods

```
newX:xvalue Y:yvalue  | |
^ self new x: xvalue
        y: yvalue


newOrigin | |
^ self new x: 0
        y: 0


initialize | |
pi <- 3.14159
```

- Explanation
  - selector is mix-fix newX:Y:
    e.g, Point newX:3 Y:2
  - symbol ^ marks return value
  - new is method in all classes,
    inherited from Object
  - | | marks scope for local decl
  - initialize method sets pi, called automatically
  - <- is syntax for assignment

程序示例：实例消息和方法

Five instance methods

```
x: xcoord y: ycoord | |
    x <- xcoord
    y <- ycoord
moveDx: dx Dy: dy | |
    x <- dx + x
    y <- dy + y
x | | ^x
y | | ^y
draw | |
    ⟨...code to draw point...⟩
```

- Explanation

set x,y coordinates,

e.g, pt x:5 y:3

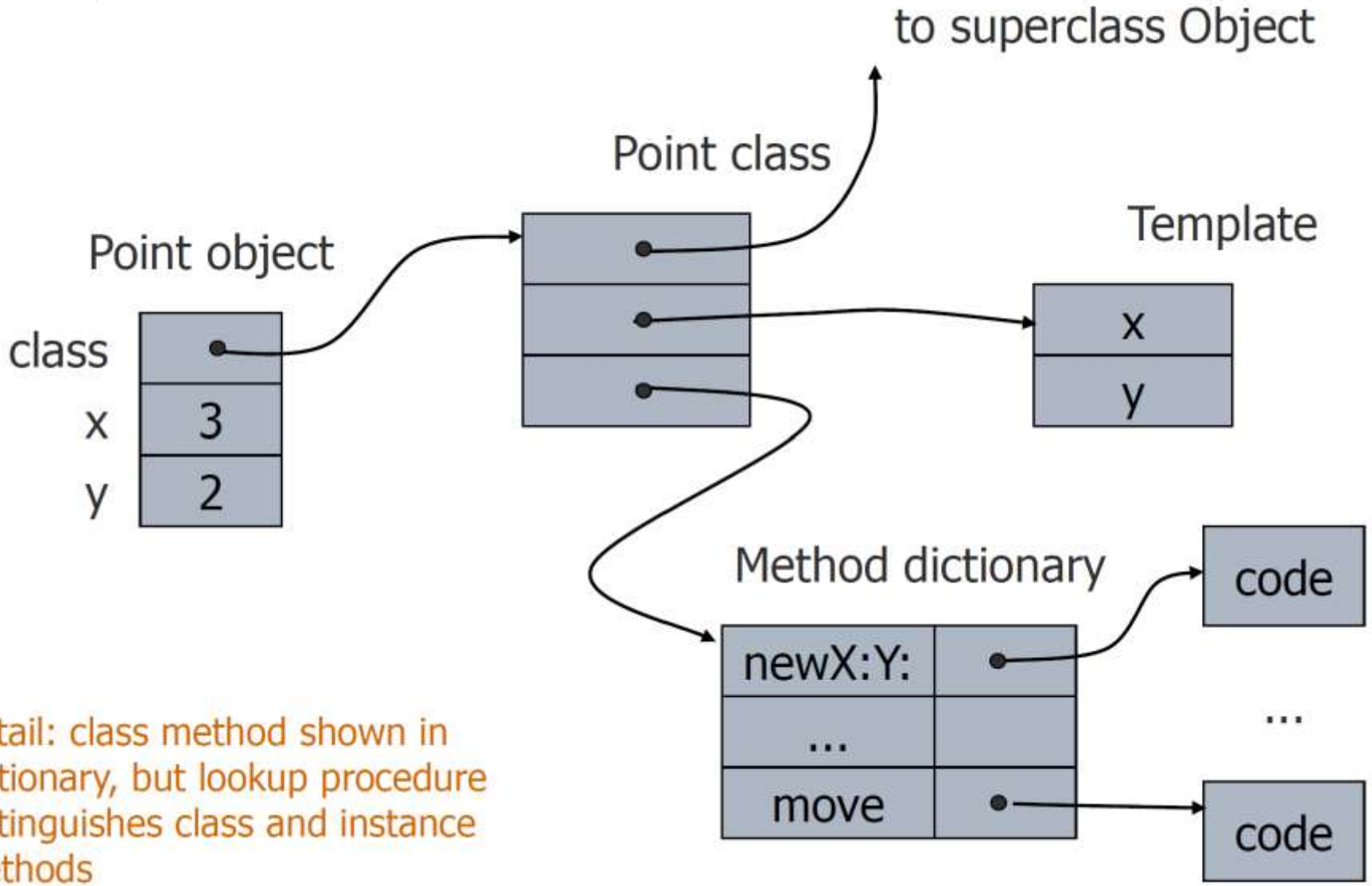
move point by given amount

return hidden inst var x

return hidden inst var y

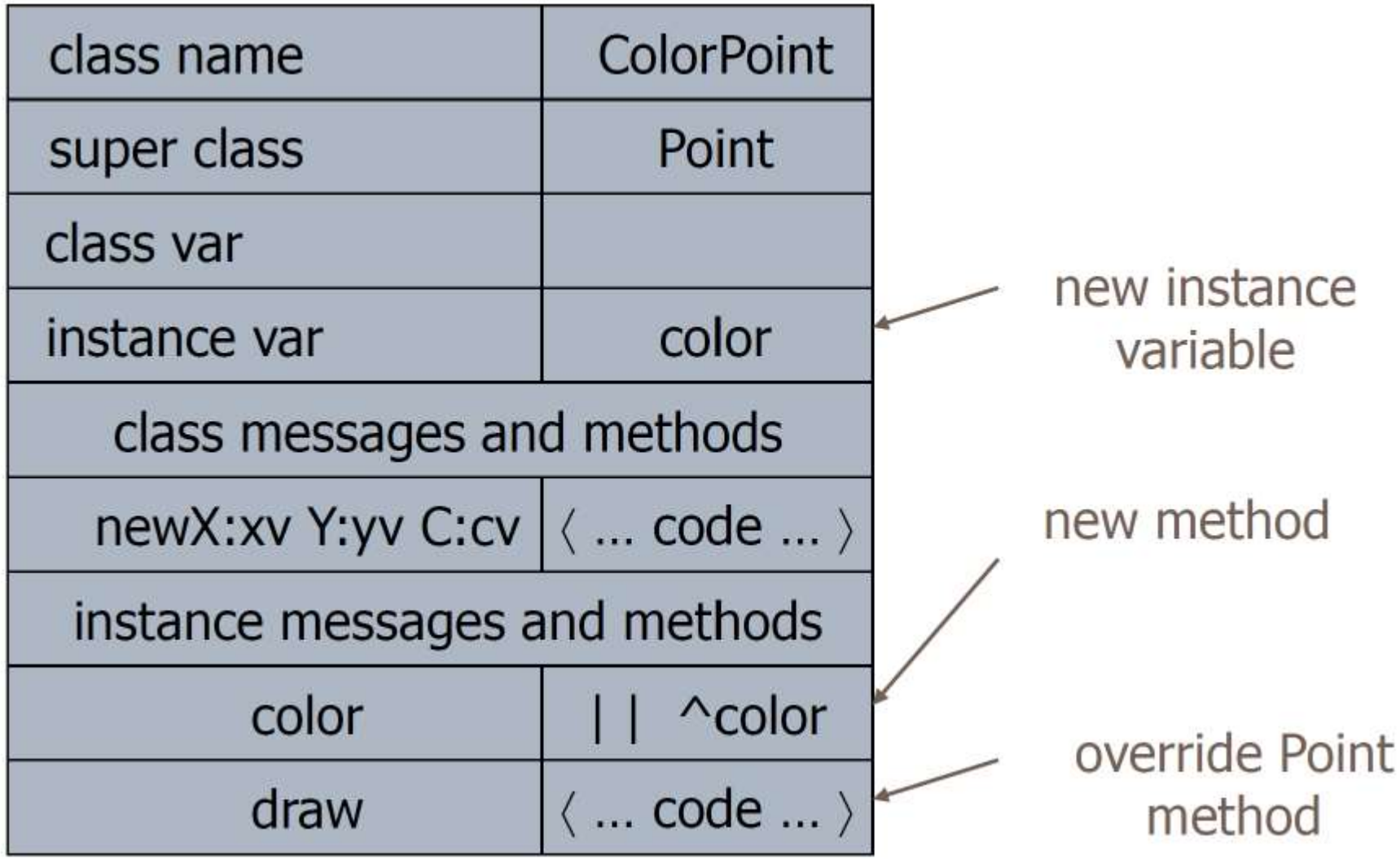
draw point on screen

对象的表示



Detail: class method shown in dictionary, but lookup procedure distinguishes class and instance methods

子类：ColorPoint



子类：ColorPoint