Konzepte von Programmiersprachen

Chapter 9: Objects and Classes

Peter Thiemann

Universität Freiburg

9. Juli 2009

Objects

- state encapsulation (instance vars, fields)
- package behavior with state (methods)

OO View of the World

- everything is an object (Smalltalk)
- computation proceeds by message passing:
- one object send method name and arguments to receiver object
- synchronous mp: procedure call
- asynchronous mp: blends with concurrent programming

Special Case: Procedures

- Procedure (closure) $\approx$ object with one kind of behavior
- Free variables $\approx$ fields
- (possible implementation of objects)
- in general: different behaviors wanted

Class-based OO Languages

- Class specifies fields and methods
- inheritance allows sharing specification and code between similar classes
- derived class
  - may add fields and methods
  - may reuse and/or override methods of the superclass
- Examples: Smalltalk, Eiffel, Java, C++, ...

Prototype-based OO Languages

- Objects are constructed as needed
- Some objects serve as templates / prototypes
- Either cloned or otherwise reused
- Examples: Self, JavaScript

Objects vs. Modules

- Commonality: mechanism for defining opaque types
- Differences
  - Object— data structure with behavior
  - Module— just bindings, but with sophisticated control

CLASSES

The Language CLASSES

```
class c1 extends object
  field i
  field j
  method initialize(x)
    begin set i = x; set j = -(0,x) end
  method countup(d)
    begin set i = -(i,-(0,d)); set j = -(j,d) end
  method getstate()list(i,j)

let o1 = new c1(3)   t1 = 0   t2 = 0
in begin
    set t1 = send o1 getstate();
    send o1 countup(2);
    set t2 = send o1 getstate();
    list(t1,t2)
   end
```

Dynamic Dispatch

```
class interior_node extends object
  field left
  field right
  method initialize(l,r)
   begin set left = l; set right = r end
  method sum()
   -(send left sum(), -(0, send right sum()))
class leaf_node extends object
  field value
  method initialize(v)set value = v
  method sum()value
let o1 = new interior_node (
           new interior_node (
             new leaf_node(3), new leaf_node(4)),
           new leaf_node(5))
in send o1 sum()
```

Mutually Recursive Methods

```
class oddeven extends object
  method initialize()1
  method even(n)
    if zero?(n) then 1 else send self odd (-(n,1))
  method odd(n)
    if zero?(n) then 0 else send self even (-(n,1))

let o1 = new oddeven()
in send o1 odd(13)
```

Inheritance

Terminology

- Inheritance: define new class C by modification of existing ones $D_1, \ldots, D_n$
- C **extends** $D_1 \ldots D_n$
  - D_i is parent or superclass of C
  - C is child, subclass, or derived class of D_i
- Single Inheritance: $n \leq 1$, tree-shaped class hierarchy
- Multiple Inheritance: $n > 0$, class relations form a DAG
- Substitution principle / subclass polymorphism:
  an instance of a child class can be used wherever an instance of its parent class is expected

Example of Inheritance

```
class point extends object
  field x
  field y
  method initialize (initx, inity)
    begin
      set x = initx;
      set y = inity
    end
  method move (dx, dy)
    begin
      set x = -(x,-(0,dx));
      set y = -(y,-(0,dy))
    end
  method get_location () list(x,y)
```

Example of Inheritance (contd)

```
class colorpoint extends point
  field color
  method set_color (c) set color = c
  method get_color () color
let p = new point(3, 4)
    cp = new colorpoint(10, 20)
in begin
     send p move(3, 4);
     send cp set_color(87);
     send cp move(10, 20);
     list(send p get_location(),     % returns (6 8)
          send cp get_location(),    % returns (20 4
          send cp get_color())       % returns 87
   end
```

Example of Field Shadowing

```
class c1 extends object
  field x
  field y
  method initialize() 1
  method setx1 (v) set x = v
  method sety1 (v) set y = v
  method getx1 () x
  method gety1 () y

class c2 extends c1
  field y
  method sety2 (v) set y = v
  method getx2 () x
  method gety2 () y
```

Example of Field Shadowing (contd)

```
let o = new c2()
in begin
     send o2 setx1 (101);
     send o2 sety1 (102);
     send o2 sety2 (999);
     list(send o2 getx1(),
          send o2 gety1(),
          send o2 getx2(),
          send o2 gety2())
   end
```

Method Overriding

```
class c1 extends object
  method initialize()1
  method m1()1
  method m2()100
  method m3()send self m2()
class c2 extends c1
  method m2()2
let o1 = new c1()
    o2 = new c2()
in list(send o1 m1(),     % returns 1
        send o1 m2(),     % returns 100
        send o1 m3(),     % returns 100
        send o2 m1(),     % returns 1 (from c1)
        send o2 m2(),     % returns 2 (from c2)
        send o2 m3()      % returns 2 (c1's m3 calls
       )
```

Super Calls — The Need

```
class point extends object
  field x
  field y
  method initialize (initx, inity)
    begin
      set x = initx;
      set y = inity
    end
  method move (dx, dy)
    begin
      set x = -(x,-(0,dx));
      set y = -(y,-(0,dy))
    end
  method get_location () list(x,y)
```

Super Calls — The Need (contd)

```
class colorpoint extends point
  field color
  method set_color (c) set color = c
  method get_color () color
let p = new point(3, 4)
    cp = new colorpoint(10, 20)
in begin
     send p move(3, 4);
     send cp set_color(87);
     send cp move(10, 20);
     list(send p get_location(),     % returns (6 8)
          send cp get_location(),    % returns (20 4
          send cp get_color())       % returns 87
   end
```

Super Calls — The Solution

```
class colorpoint extends point
  field color
  method set_color (c) set color = c
  method get_color () color
  method initialize (x,y,c)
   begin super initialize (x,y); set color = c end
let p = new point(3, 4)
    cp = new colorpoint(10, 20, 30)
in begin
     send p move(3, 4);
     send cp set_color(87);
     send cp move(10, 20);
     list(send p get_location(),     % returns (6 8)
          send cp get_location(),    % returns (20 4
          send cp get_color())       % returns 87
   end
```

Interaction of Super Calls with Self

```
class c1 extends object
 method initialize () 1
 method m1 () send self m2 ()
 method m2 () 13
class c2 extends c1
 method m1 () 22
 method m2 () 23
 method m3 () super m1 ()
class c3 extends c2
 method m1 () 32
 method m2 () 33
let o3 = new c3()
in send o3 m3()
```