1

Object-Oriented Programming Languages: A Comparative Survey

• Simula -- the first object-oriented programming language (1966) defined as an extension of ALGOL 60, intended for simulation applications

• Smalltalk -- developed as an independent language where everything is an object and all computation defined in terms of message passing

• Object Pascal -- an object-oriented extension of Pascal

• Objective C, C++ -- object-oriented extensions of C

2

Object-Oriented Programming Languages: Simula-67

• The first object-oriented programming language (Ole-Johan Dahl '67), defined as an extension of ALGOL60 for simulation applications

Class -- a combination of a record type and a process

Class instances are partly record structures and partly process instances; when the process instance completes its execution, a class instance remains as a "passive" record structure.

Consider frequency histograms which keep track of the frequency of a random variable between n+1 intervals $(-\infty, X_1)$, (X_1, X_2), ...(X_n, ∞). Histograms will be represented by instances of the class `histo`

3

Simula-67 Example

`n` specifies the number of interval points, while `X` gives their values. The number of observations for each interval is stored in array `T` (which is private to `histo` instances)

```
class histo (X, n);array X;integer n;
begin integer N; integer array T[0;n];
   procedure tabulate (Y); real Y;
      begin integer i; i := 0; ... end;
   procedure frequency (i); integer i;
      frequency := T[i]/N;
   integer i;
   for i := 0 step 1 until n do
      T[i] := 0; N := 0
   end
end.
```


4

Simula: Creating Class Instances

We can now use this class definition to create a number of histograms:

```
begin
 ref(histo) heights, weights, lengths;
 real array A[1:7], B[1:12];
   ...initialize A, B,...
  heights :- new histo(A, 7);
  weights :- new histo(B, 12);
  lengths :- new histo(A, 7);
```

...rest of program...

To update one of the instances we use their associated operations:

```
...
weights.tabulate(w);
heights.tabulate(h);
...
```

5

Comparison of Four Languages: Object Pascal, Smalltalk, Objective C, C++

Let's consider a playing card program which represents and displays playing cards

`Card`
- `suit` -- return card suit
- `rank` -- return card rank
- `colour` -- return card colour

`CardView`
- `theCard` -- (type `Card`) the card value
- `up` -- (Boolean) face-up or down status
- `locx`, `locy` -- location on playing surface

`methods`
- `erase`, `draw` -- erase or draw card image
- `faceUp`, `flip` -- test or flip card
- `includes(int, int)` -- test whether point is within card boundary
- `moveTo(int, int)` -- move card to new location

6

Object Pascal

- Developed by Apple, based on module facility of Apple Pascal; modules called units

- Units are divided into an interface component and an implementation component (but are kept together, unlike other languages)

- The declaration of `CardView` includes a var of type `Card`; when a `CardView` object is created, space is set aside for a pointer to a `Card`

7

Classes and Methods: Object Pascal

Classes have an interface and an implementation part

Here is the interface part for the two classes

```
unit card;
   interface
    type
      suits = {Heart,Club,Diamond, Spade};

      colors = {Red, Black};

      Card = object
           suitValue:   suits;
           rankValue:   integer;

       procedure init(c:integer,s:suits);
        function  color:  colors;
        function  rank:integer;
        function  suit:suits;
       end;
```


8

...and the Interface of `CardView`

```
CardView = object
 theCard:    Card;
 up:    boolean;
 xcoord:   integer;
 ycoord:   integer;

 function  card: Card;
 function  faceUp: boolean;
 function  x: integer;
 function  y: integer;

 procedure flip;
 procedure moveTo(x, y: integer);
 procedure draw;
 function includes(x, y: integer)
     : boolean;
end;
```

9

The Implementation Portion of Class Definitions

```
implementation
   const
      CardWidth = 65;
      CardHeight = 75;

   function Card.color: colors;
   begin
      case suit of
         Diamond:   color := Red;
         Heart:     color := Red;
         Club:      color := Black;
         Spade:     color := Black;
      end;
   end;
...
   function CardView.includes(
              x, y: integer): boolean
   begin
      includes := (x ≥ xcoord)
            and (x ≤ xcoord + CardWidth)
            and (y ≥ ycoord)
            and (y ≤ ycoord + CardHeight);
   end;
...
```


@1993 John Mylopoulos

10

Smalltalk

• After Simula, the most influential object-oriented language; developed by Xerox

• Smalltalk is an interactive language, much like Lisp; users communicate with the Smalltalk system through the browser; unlike Lisp, the browser is window+mouse-based

Class Cats	Classes	Method Cats	Methods
Graphics	Array	Insertion	add:
Collections	Bag	Removal	addAll
Numerics	Set	Testing	
System	Dictionary	Printing	

```
message selector and argument names
    "comment stating purpose of message"
    | temp variable names |
    statements
```


@1993 John Mylopoulos

11

Classes and Methods in Smalltalk

Definition of the class `Card`

```
Object subclass:   #Card
   instanceVariableNames:
           'suit rank up x y'
   classVariableNames:      ''
   category: 'Graphics
```

`Card` is made a subclass of `Object`; definition of the method `setSuit:Rank`

```
setSuit:   s  rank:   c
   suit ← s.
   rank ← c.
   up ← false.
   x ← y ← 0.
```

...and the method `suit`:

```
suit ↑suit
```

Class and instance variables are untyped; moreover, instance variables are not accessible outside a class

Class variables are associated with the class itself, rather than its instances (e.g., `author`)

Classes are objects too -- and are therefore instances of metaclasses, which are objects too...

12

Classes and Methods in Objective C

Definition, again, partitioned into an interface and an implementation portion

```
@interface Card: Object
{ int   suit; int  rank;}
- (void)   suit:   (int) s    /* init */
           rank:   (int) c;
-  (int)   color;
-  (int)   rank;
-  (int)   suit;
@end

@interface CardView: Object
{ id theCard; int up;
              int x; int y; id link; }
-          card;
- (int)    faceUp;
- (void)   flip;
- (int)    includesX: (int) testx
                    y:  (int) texty;
- (void)   moveTo: (int) newx
              and:  (int) newy;
- (int)    x;
- (int)    y;
@end
```

13

...and the Implementation Portion in Objective C

```
# import "card.h"
#  define cardWidth 68
# define cardHeight 75

@implementation Card
- (int) color {return suit;}
...
- (void) flip {up = ! up;}
...
- (void) moveTo:
        (int) newx and: (int) newy;
        {x = newx; y = newy;}
...
```

• The implementation file begins by importing the interface file

• Objective C code can be freely intermixed with C code

14

C++

Probably the inter-galactic standard for object-oriented programming

• This is an extension of C; as in C, it is useful to distinguish between interface files (`__.h`) and implementation files

• Since there is no importation facility in C, a conditional inclusion facility (`ifndef`) can be used instead

• Like OOT, C++ allows you to define pointers to objects (see `theCard` field of `CardView` definition)

15

Classes and Methods in C++

```
// Interface description for class Card
#  ifndef cardh
# define cardh  // include this file once
class Card
{ public
      int   setSuitAndCount(int,int);
      int   suit();
      int   color();
      int   rank();
   private
      int   s;
      int   r;    }
class CardView
{  public
      const int   CardWidth = 68;
      const int   CardHeight = 75;
      Card *  card();
      void    draw();
      ...
   private
      Card *  theCard;
      int     up;
      ...              }
```


@1993 John Mylopoulos

16

Messages, Instances and Initialization

- How are new instances of a class created and manipulated?
- How does the programmer describe the process of message passing?
- How does the programmer initialize new objects?

@1993 John Mylopoulos

17

Message Passing

• Object Pascal -- a message is a request to an object to invoke one of its methods

```
x.moveTo(25, 35)
x.flip
```

• C++ -- methods in C++ usually called "member functions" and message passing usually referred to as member function invocation

```
if x.includes(25, 35) ... else...
```

• Smalltalk -- message passing indicated in terms of a <receiver> <message> pair

```
x flip
x moveTo: 25 and: 35
```

• Objective C -- uses syntax similar to Smalltalk's

```
[ x flip ];
[ x moveTo: 25 and: 35 ];
```


18

Creation and Initialization: Storage Management

• Automatic storage allocation -- storage is allocated for a variable at the time the procedure or block within which the variable is declared is entered

• Dynamic storage allocation -- now storage is allocated when the programmer says so in terms of a statement such as `new(x)`

• For dynamic storage allocation, we need to worry about storage recovery, for storage that is no longer in use (why not for automatic?)

Two alternatives: recover storage through explicit statements,

e.g., `dispose (x)`	Object Pascal, C++
,or, use garbage collection	CLOS, Smalltalk

19

Creation and Initialization in C++

• Supports both automatic and dynamic variables

• Objects can be initialized implicitly through constructor functions; a constructor function is a method having the same name as its class and is invoked every time the class is instantiated

```
class Complex
{  public
      float:  real;
      float:  imag;
      Complex();
      Complex(float rp);
      Complex(float rp, float ip);
      ...                }
   Complex::Complex();
   { real = imag = 0.0;}
   Complex::Complex(float rp);
   { real = rp; imag = 0.0; }
   Complex::Complex(float rp, float ip);
   { real = rp; imag = ip;}
...
c = new Complex(3.14159, 2.4);
```

`Complex` is an overloaded function

20

Creation and Initialization in Object Pascal

• All objects are dynamic, have to be constructed, destroyed explicitly through `new`, `dispose`

• There is no support for implicit initialization (Object Pascal is unique among object-oriented languages in that respect)

```
type Complex: object
      rp:  real; im:  real;
      procedure initial(r,i: real);
var   p: Complex;
...
procedure Complex.initial(r,i: real);
begin rp := r; ip := i; end;

begin
new(p);
p.initial(3.14159, 2.4);
...
dispose(p);
```

21

Creation and Initialization in Smalltalk

All variables are dynamic and dynamically typed; new instances of a class are created by passing to the class object the message `new`

Class methods used to help with initialization. Here is a class method for class `Complex`

```
r: rp i: ip   | n |
   n ← Complex new.  "Create new number"
   n real: rp imag: ip. "assign values"
   ↑ n
```

this can be used to create and initialize new classes

```
c ← Complex r: 3.14159 i: 2.4
```


22

Creation and Initialization in Objective C

- Combines C declarations with Smalltalk syntax

- Objects are usually declared to be instances of `id` and space is allocated dynamically, as in Smalltalk; however, storage is not automatically reclaimed

- Factory methods are similar to class methods, can be used for initialization; here is an example of a factory method:

```
@implementation Complex: Object
{ int realpart; int imagpart;}

+ r: (int) rp i: (int) ip
{ self = [ Complex new ];
realpart = rp;
imagpart = ip;
return self; }
...
```

23

Control Structures in Object-Oriented Programming Languages

• For Object Pascal, Objective C and C++ the control structures provided are mostly those of the underlying (imperative) programming language

• Smalltalk relies exclusively on message passing; this means that control structures such as conditional, iteration etc. are all defined in terms of message passing

Conditional -- it is a special object which takes messages of the form

<test-expression>
`ifTrue: [` <expression-sequence> `]`
`ifFalse: [` <expression-sequence> `]`

and returns the value of the first or second block, depending on the value of the test expression

Iteration -- is also a special object which takes messages of the form

<test-expr> `whileTrue: [` <expr-sequence `]`

or

<test-expr> `whileFalse: [`<expr-sequence> `]`

and keeps evaluating the block every time it receives the message `value`, until <test-expr> becomes false (true)

@1993 John Mylopoulos

24

```
class stack
   instance variables
      elements, top, size
   class methods
      new: aSize
         ↑((self new) initialize: aSize)
   instance methods
      isEmpty
         ↑top = 0
      isFull
         ↑top ≥ size
      push: anElement
         self isFull
          ifTrue: [self error:'full']
          ifFalse:
            [top ← top + 1.
             elements at: top put:
                              anElement]
      pop |temp|
         self isEmpty
            ifTrue: [self error:'empty']
            ifFalse:
             [temp ← elements at: top.
              top ← top - 1. ↑temp]
      initialize: aSize
       top ← 0. size ← aSize.
       elements ← Array new: (aSize + 1).
```


@1993 John Mylopoulos