CS 242

Simula and Smalltalk

John Mitchell

Simula 67

- First object-oriented language
- Designed for simulation
  - Later recognized as general-purpose prog language
- Extension of Algol 60
- Standardized as Simula (no "67") in 1977
- Inspiration to many later designers
  - Smalltalk
  - C++
  - …

Brief history

- Norwegian Computing Center
  - Designers: Dahl, Myhrhaug, Nygaard
  - Simula-1 in 1966 (strictly a simulation language)
  - General language ideas
    - Influenced by Hoare's ideas on data types
    - Added classes and prefixing (subtyping) to Algol 60
  - Nygaard
    - Operations Research specialist and political activist
    - Wanted language to describe social and industrial systems
    - Allow "ordinary people" to understand political (?) changes
  - Dahl and Myhrhaug
    - Maintained concern for general programming

Comparison to Algol 60

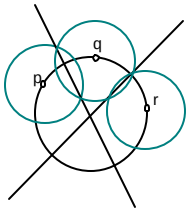
- Added features
  - class concept
  - reference variables (pointers to objects)
  - pass-by-reference
  - char, text, I/O
  - coroutines
- Removed
  - Changed default par passing from pass-by-name
  - some var initialization requirements
  - own (=C static) variables
  - string type (in favor of text type)

Objects in Simula

- Class
  - A procedure that returns a pointer to its activation record
- Object
  - Activation record produced by call to a class
- Object access
  - Access any local variable or procedures using dot notation: object.
- Memory management
  - Objects are garbage collected
    - user destructors considered undesirable

Example: Circles and lines

- Problem
  - Find the center and radius of the circle passing through three distinct points, p, q, and r
- Solution
  - Draw intersecting circles Cp, Cq around p,q and circles Cq', Cr around q, r (Picture assumes Cq = Cq')
  - Draw lines through circle intersections
  - The intersection of the lines is the center of the desired circle.
  - Error if the points are colinear.



Approach in Simula

- Methodology
  - Represent points, lines, and circles as objects.
  - Equip objects with necessary operations.
- Operations
  - Point
    - equality(anotherPoint) : boolean
    - distance(anotherPoint) : real (needed to construct circles)
  - Line
    - parallelto(anotherLine) : boolean (to see if lines intersect)
    - meets(anotherLine) : REF(Point)
  - Circle
    - intersects(anotherCircle) : REF(Line)

Simula Point Class

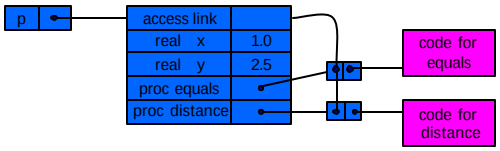
```
class Point(x,y); real x,y;
      begin
        boolean procedure equals(p); ref(Point) p;
          if p =/= none then
              equals := abs(x - p.x) + abs(y - p.y) < 0.00001
        real procedure distance(p); ref(Point) p;
          if p == none then error else
              distance :=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

Representation of objects



Object is represented by activation record with access link to find global variables according to static scoping

Simula line class

```
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

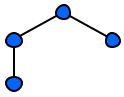
Procedures

Initialization: "normalize" a,b,c

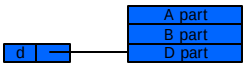
Derived classes in Simula

- A class decl may be prefixed by a class name

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



- An object of a "prefixed class" is the concatenation of objects of each class in prefix
  - d :- new D(...)



Subtyping

- The type of an object is its class
- The type associated with a subclass is treated as a subtype of the type assoc with superclass
- Example:

```
class A(...);  ...
A class B(...); ...
ref (A) a :- new A(...)
ref (B) b :- new B(...)
a := b          /* legal since B is subclass of A */
...
b := a          /* also legal, but run-time test  */
```

Main object-oriented features

- Classes
- Objects
- Inheritance ("class prefixing")
- Subtyping
- Virtual methods
  - A function can be redefined in subclass
- Inner
  - Combines code of superclass with code of subclass
- Inspect/Qua
  - run-time class/type tests

Features absent from Simula 67

- Encapsulation
  - All data and functions accessible; no private, protected
- Self/Super mechanism of Smalltalk
  - But has an expression this⟨class⟩ to refer to object itself, regarded as object of type ⟨class⟩. Not clear how powerful this is...
- Class variables
  - But can have global variables
- Exceptions
  - Not an OO feature anyway ...

Simula Summary

- Class
  - "procedure" that returns ptr to activation record
  - initialization code always run as procedure body
- Objects: closure created by a class
- Encapsulation
  - protected and private not recognized in 1967
  - added later and used as basis for C++
- Subtyping: determined by class hierarchy
- Inheritance: provided by class prefixing

Smalltalk

- Major language that popularized objects
- Developed at Xerox PARC
  - Smalltalk-76, Smalltalk-80 were important versions
- Object metaphor extended and refined
  - Used some ideas from Simula, but very different lang
  - Everything is an object, even a class
  - All operations are "messages to objects"
  - Very flexible and powerful language
    - Similar to "everything is a list" in Lisp, but more so
    - Example: object can detect that it has received a message it does not understand, can try to figure out how to respond.

Motivating application: Dynabook

- Concept developed by Alan Kay (now Disney?)
- Small portable computer
  - Revolutionary idea in early 1970's
    - At the time, a *minicomputer* was shared by 10 people, stored in a machine room.
  - What would you compute on an airplane?
- Influence on Smalltalk
  - Language intended to be programming language and operating system interface
  - Intended for "non-programmer"
  - Syntax presented by language-specific editor

Smalltalk language terminology

- Object — Instance of some class
- Class — Defines behavior of its objects
- Selector — Name of a message
- Message — Selector together with parameter values
- Method — Code used by a class to respond to message
- Instance variable — Data stored in object
- Subclass — Class defined by giving incremental modifications to some superclass

Example: Point class

- Class definition written in tabular form

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...⟩	

Class messages and methods

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

Instance messages and methods

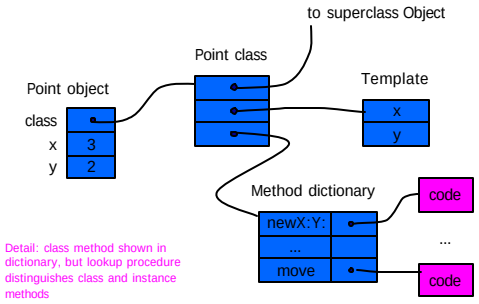
Five instance methods

```
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

Run-time representation of point



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

Inheritance

- Define colored points from points

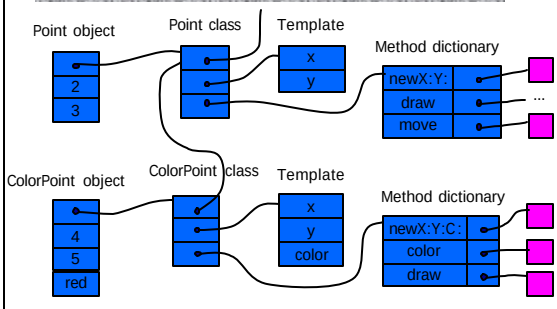
class name	ColorPoint
super class	Point
class var	
instance var	color
class messages and methods	
newX:xv Y:yv C:cv	⟨...code...⟩
instance messages and methods	
color	| | ^color
draw	⟨...code...⟩

new instance variable

new method

override Point method

Run-time representation



This is a schematic diagram meant to illustrate the main idea. Actual implementations may differ.

Encapsulation in Smalltalk

- Methods are public
- Instance variables are hidden
  - Not visible to other objects
    - pt x is not allowed unless x is a method
  - But may be manipulated by subclass methods
    - This limits ability to establish invariants
    - Example:
      - Superclass maintains sorted list of messages with some selector, say insert
      - Subclass may access this list directly, rearrange order

Object type

- Each object has interface
  - Set of instance methods declared in class
  - Example:
    Point { x:y:, moveDx:Dy:, x, y, draw}
    ColorPoint { x:y:, moveDx:Dy:, x, y, color, draw}
  - This is a form of type
    Names of methods, does not include type/protocol of arguments
- Object expression and type
  - Send message to object
    p draw p x:3 y:4
    q color q moveDx: 5 Dy: 2
  - Expression OK if message is in interface

Subtyping

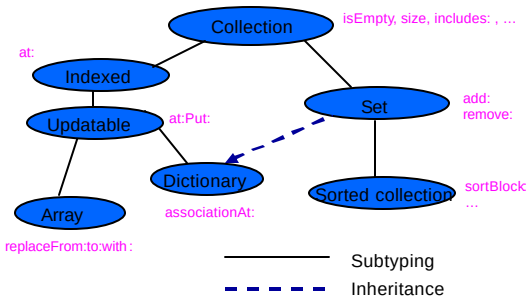
- Relation between interfaces
  - Suppose expression makes sense
    p msg:pars -- OK if msg is in interface of p
  - Replace p by q if interface of q contains interface of p
- Subtyping
  - If interface is superset, then a subtype
  - Example: ColorPoint subtype of Point
  - Sometimes called “conformance”

Can extend to more detailed interfaces that include types of parameters

Subtyping and Inheritance

- Subtyping is implicit
  - Not a part of the programming language
  - Important aspect of how systems are built
- Inheritance is explicit
  - Used to implement systems
  - No forced relationship to subtyping

Collection Hierarchy



Smalltalk Flexibility

- Measure of PL expressiveness:
  - Can constructs of the language be defined in the language itself?
  - Examples:
    - Lisp cond: Lisp allows user-defined special forms
    - ML datatype: sufficient to define polymorphic lists, equivalent to built-in list type
    - ML overloading: limitation, since not available to programmer
    - C/C++: ???
- Smalltalk is expressive in this sense
  - Many constructs that would be “primitives” other are definable in Smalltalk
  - Example: Booleans and Blocks

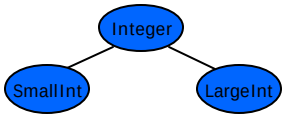
Smalltalk booleans and blocks

- Boolean value is object with ifTrue:ifFalse:
  - Class boolean with subclasses True and False
  - True ifTrue:B1 ifFalse:B2 executes B1
  - False ifTrue:B1 ifFalse:B2 executes B2
- Example expression

```
i < j  ifTrue: [i add 1] ifFalse: [j subtract 1]
```

  - i < j is boolean expression, produces boolean object
  - arg's are *blocks*, objects with execute methods
- Since booleans and blocks are very common
  - Optimization of boolean
  - Special syntax for blocks

Self and Super




```
Factorial |   |
    self <= 1
        ifTrue: [^1]
        ifFalse: [^(self-1) factorial * self ]
```

This method can be implemented in Integer, and works even if SmallInt and LargeInt are represented differently. C++ and Java type systems can't really cope with this.

Ingalls' test

- Dan Ingalls: principal designer Smalltalk system
  - Grace Murray Hopper award for Smalltalk and bitmap graphics work at Xerox PARC
  - 1987 ACM Software Systems Award with Kay, Goldberg
- Proposed test for "object oriented"
  - Can you define a new kind of integer, put your new integers into rectangles (which are already part of the window system), ask the system to blacken a rectangle, and have everything work?
  - Smalltalk passes, C++ fails this test

Smalltalk integer operations

- Integer expression
  - x plus: 1 times: 3 plus: (y plus: 1) print
- Properties
  - All operations are executed by sending messages
  - If x is from some "new" kind of integer, expression makes sense as long as x has plus, times, print methods.

Actually, compiler does some optimization. But will revert to this if x is not built-in integer.

Costs and benefits of "true OO"

- Why is property of Ingalls test useful?
  - Everything is an object
  - All objects are accessed only through interface
  - Makes programs extensible
- What is implementation cost?
  - Every integer operation involves method call
    - Unless optimizing compiler can recognize many cases
  - Is this worth it?
    - One application where it seems useful ?
    - One application where it seems too costly?
    - Are there other issues? Security? (wait for Java final classes…)

Smalltalk Summary

- Class
  - creates objects that share methods
  - pointers to template, dictionary, parent class
- Objects: created by a class, contains instance variables
- Encapsulation
  - methods public, instance variables hidden
- Subtyping: implicit, no static type system
- Inheritance: subclasses, self, super

  Single inheritance in Smalltalk-76, Smalltalk-80