CS 242

Concepts in Object-Oriented Programming Languages

John Mitchell

Outline of lecture

- Object-oriented design
- Primary object-oriented language concepts
  - dynamic lookup
  - encapsulation
  - inheritance
  - subtyping
- Program organization
  - Work queue, geometry program, design patterns
- Comparison
  - Objects as closures?

Objects

- An object consists of
  - hidden data
    - instance variables, also called member data
    - hidden functions also possible
  - public operations
    - methods or member functions
    - can also have public variables in some languages

hidden data	
msg_1	$method_1$
. . .	. . .
msg_n	$method_n$

- Object-oriented program:
  - Send messages to objects

What's interesting about this?

- Universal encapsulation construct
  - Data structure
  - File system
  - Database
  - Window
  - Integer
- Metaphor usefully ambiguous
  - sequential or concurrent computation
  - distributed, sync. or async. communication

Object-oriented programming

- Programming methodology
  - organize concepts into objects and classes
  - build extensible systems
- Language concepts
  - encapsulate data and functions into objects
  - subtyping allows extensions of data types
  - inheritance allows reuse of implementation

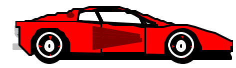
Object-oriented Method [Booch]

- Four steps
  - Identify the objects at a given level of abstraction
  - Identify the semantics (intended behavior) of objects
  - Identify the relationships among the objects
  - Implement these objects
- Iterative process
  - Implement objects by repeating these steps
- Not necessarily top-down
  - "Level of abstraction" could start anywhere

This Method

- Based on associating objects with components or concepts in a system
- Why iterative?
  - An object is typically implemented using a number of constituent objects
  - Apply same methodology to subsystems, underlying concepts

Example: Compute Weight of Car



- Car object:
  - Contains list of main parts (each an object)
    - chassis, body, engine, drive train, wheel assemblies
  - Method to compute weight
    - sum the weights to compute total
- Part objects:
  - Each may have list of main sub-parts
  - Each must have method to compute weight

Comparison to top-down design

- Similarity:
  - A task is typically accomplished by completing a number of finer-grained sub-tasks
- Differences:
  - Focus of top-down design is on program structure
  - OO methods are based on modeling ideas
  - Combining functions and data into objects makes data refinement more natural (I think)

Object-Orientation

- Programming methodology
  - organize concepts into objects and classes
  - build extensible systems
- Language concepts
  - dynamic lookup
  - encapsulation
  - subtyping allows extensions of concepts
  - inheritance allows reuse of implementation

Dynamic Lookup

- In object-oriented programming,

  object → message (arguments)

  code depends on object and message

- In conventional programming,

  operation (operands)

  meaning of operation is always the same

Fundamental difference between abstract data types and objects

Example

- Add two numbers x → add (y)

  different add if x is integer, complex

- Conventional programming add (x, y)

  function add has fixed meaning

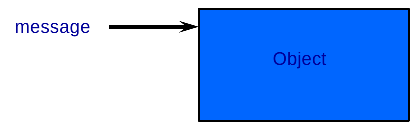
Very important distinction:
Overloading is resolved at compile time,
Dynamic lookup at run time

Language concepts

- "dynamic lookup"
  - different code for different object
  - integer "+" different from real "+"
- encapsulation
- subtyping
- inheritance

Encapsulation

- Builder of a concept has detailed view
- User of a concept has "abstract" view
- Encapsulation is the mechanism for separating these two views



Comparison

- Traditional approach to encapsulation is through abstract data types
- Advantage
  - Separate interface from implementation
- Disadvantage
  - Not extensible in the way that OOP is

We will look at ADT's example to see what problem is

Abstract data types

```
abstype q
   with
     mk_Queue : unit -> q
     is_empty : q -> bool
     insert   : q * elem -> q
     remove   : q -> elem
   is  ...
   in
          program
   end
```

Priority Q, similar to Queue

```
abstype pq
  with mk_Queue : unit -> pq
       is_empty : pq -> bool
       insert   : pq * elem -> pq
       remove   : pq -> elem
  is ...
   in
       program
  end
```

But cannot intermix pq's and q's

Abstract Data Types

- Guarantee invariants of data structure
  - only functions of the data type have access to the internal representation of data
- Limited "reuse"
  - Cannot apply queue code to pqueue, except by explicit parameterization, even though signatures identical
  - Cannot form list of points, colored points
- Data abstraction is important part of OOP, innovation is that it occurs in an *extensible* form

Language concepts

- "dynamic lookup"
  - different code for different object
  - integer "+" different from real "+"
- encapsulation
- subtyping
- inheritance

Subtyping and Inheritance

- Interface
  - The external view of an object
- Subtyping
  - Relation between interfaces
- Implementation
  - The internal representation of an object
- Inheritance
  - Relation between implementations

Object Interfaces

- Interface
  - The messages understood by an object
- Example: point
  - x-coord : returns x-coordinate of a point
  - y-coord : returns y-coordinate of a point
  - move : method for changing location
- The interface of an object is its *type*.

Subtyping

- If interface A contains all of interface B, then A objects can also be used B objects.

Point	Colored_point
x-coord y-coord move	x-coord y-coord color move change_color

- Colored_point interface contains Point
  - Colored_point is a *subtype* of Point

Inheritance

- Implementation mechanism
- New objects may be defined by reusing implementations of other objects

Example

```
class Point
    private
        float x, y
    public
        point move (float dx, float dy);
class Colored_point
    private
        float x, y; color c
    public
        point move(float dx, float dy);
        point change_color(color newc);
```

- Subtyping
  - Colored points can be used in place of points
  - Property used by client program
- Inheritance
  - Colored points can be implemented by resuing point implementation
  - Property used by implementor of classes

OO Program Structure

- Group data and functions
- Class
  - Defines behavior of all objects that are instances of the class
- Subtyping
  - Place similar data in related classes
- Inheritance
  - Avoid reimplementing functions that are already defined

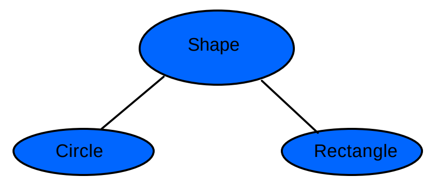
Example: Geometry Library

- Define general concept shape
- Implement two shapes: circle, rectangle
- Functions on implemented shapes
  center, move, rotate, print
- Anticipate additions to library

Shapes

- Interface of every shape must include
  center, move, rotate, print
- Different kinds of shapes are implemented differently
  - Square: four points, representing corners
  - Circle: center point and radius

Subtype hierarchy



- General interface defined in the shape class
- Implementations defined in circle, rectangle
- Extend hierarchy with additional shapes

Code placed in classes

	center	move	rotate	print
Circle	c_center	c_move	c_rotate	c_print
Rectangle	r_center	r_move	r_rotate	r_print

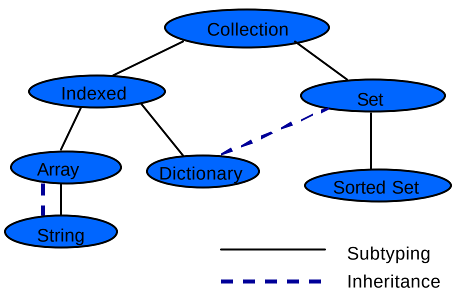
- Dynamic lookup
  - circle → move(x,y) calls function c_move
- Conventional organization
  - Place c_move, r_move in move function

Example use: Processing Loop

Remove shape from work queue
Perform action

Control loop does not know the type of each shape

Subtyping differs from inheritance



Design Patterns

- Classes and objects are useful organizing concepts
- Culture of *design patterns* has developed around object-oriented programming
  - Shows value of OOP for program organization and problem solving

What is a design pattern?

- General solution that has developed from repeatedly addressing similar problems.
- Example: singleton
  - Restrict programs so that only one instance of a class can be created
  - Singleton design pattern provides standard solution
- Not a class template
  - Using most patterns will require some thought
  - Pattern is meant to capture experience in useful form

Standard reference: Gamma, Helm, Johnson, Vlissides

OOP in Conventional Language

- Records provide “dynamic lookup”
- Scoping provides another form of encapsulation

Try object-oriented programming in ML.

Will it work? Let’s see what’s fundamental to OOP

Dynamic Lookup (again)

receiver → operation (arguments)

code depends on receiver and operation

This is may be achieved in conventional languages using record with function components

Stacks as closures

```
fun create_stack(x) =
  let val store = ref [x] in
    {push = fn (y) =>
         store := y::(!store),
     pop  = fn () =>
         case !store of
            nil => raise Empty |
            y::m => (store := m; y)
    } end;

val stk = create_stack(1);
    stk = {pop=fn,push=fn} : {pop:unit -> int, push:int -> unit}
```

Does this work ???

- Depends on what you mean by "work"
- Provides
  - encapsulation of private data
  - dynamic lookup
- But
  - cannot substitute extended stacks for stacks
  - only weak form of inheritance
    - can add new operations to stack
    - not mutually recursive with old operations

Varieties of OO languages

- class-based languages
  - behavior of object determined by its class
- object-based
  - objects defined directly
- multi-methods
  - operation depends on all operands

This course: class-based languages

History

- Simula 1960's
  - Object concept used in simulation
- Smalltalk 1970's
  - Object-oriented design, systems
- C++ 1980's
  - Adapted Simula ideas to C
- Java 1990's
  - Distributed programming, internet

Next lectures

- Simula and Smalltalk
- C++
- Java

Summary

- Object-oriented design
- Primary object-oriented language concepts
  - dynamic lookup
  - encapsulation
  - inheritance
  - subtyping
- Program organization
  - Work queue, geometry program, design patterns
- Comparison
  - Objects as closures?

Example: Container Classes

- Different ways of organizing objects
  - Set: unordered collection of objects
  - Array: sequence, indexed by integers
  - Dictionary: set of pairs, (word, definition)
  - String: sequence of letters
- Developed as part of Smalltalk system