CLOS

CSC 358/458
6.3.2003
Outline

- Homework #8
- CLOS
  - Basic features
  - OO programming
- Examples
Homework #8

- Monkey
- Student
CLOS

- Adds object-oriented programming to Lisp
- OOP is a choice
  - like C++
CLOS features

- Built-in classes integrated with type system
- User-defined classes also part of type system
- Multiple inheritance
- Class variables (static) and instance variables
- Programmer-selected method combination
- Multiple dispatch
defclass

- creates a new class
- supply
  - name
  - superclasses
  - slot specifiers
  - class options
Example

(defclass submarine (boat)
  (speed depth))
make-instance

- Creates an instance of a class
- Example
  - (make-instance 'submarine)
slot-value

- Allows access to defined slot
  - instance variables

- Example
  - `(setf s1 (make-instance 'submarine))`
  - `(setf (slot-value s1 'speed) 100)`
defmethod

- Methods are defined separately from classes
- Methods are specialized by the class they operate on
Example

(defmethod dive ((sub submarine) new-depth)
  (setf (slot-value sub 'depth) new-depth))
Messaging Model of OOP

Most OO languages assume a message passing model
- The method signature = message
- Object with the method = recipient of message

But what about double-dispatch
- transfer (account1, account2)
- or even
- equals (obj1, obj2)

We are forced to pick one object
- implement method there
Dispatch Model
Dispatch Model of OOP

- Methods independent of objects
- Methods specialized for objects
  - but can be specialized for multiple objects
Example

- In polynomials domain
- add can be multiply defined
  - add (term poly)
  - add (term term)
  - add (poly term)
  - add (poly poly)
Slot specification

- slot-value is awkward
  - can't be compiled
- defclass has slot specifications
  - :accessor
  - :reader
  - :writer
  - :initform
  - :initarg
  - :allocation
  - :type
initform

- Default initial value
- Can be overridden by arguments to make-instance
- Evaluated each time make-instance is called
initarg

- A keyword that can be passed to make-instance
- Gives slot its initial value
- Overrides initform
Example

(defclass Submarine ()
  ((Depth :initarg :depth :initform 100)
   (Speed :reader Speed :initform 5)))

(setq Sub-1 (make-instance 'Submarine :depth 200))
Accessors

- reader
  - Name of a function that will read the slot

- writer
  - Name of a function that will set the slot's value

- accessor
  - Name of a function to read and (with setf) write the slot's value
Example 1

(defclass Submarine ()
    ((Depth :reader Depth :initform 100)
     (Speed :reader Speed :initform 5)))

(setq Sub-1 (make-instance 'Submarine))
(Depth Sub-1) ==> 100
(Speed Sub-1) ==> 5
Example 2

(defclass Submarine ()
   ((Depth :reader Depth :writer Set-Depth :initform 100)
    (Speed :reader Speed :writer Set-Speed :initform 5)))

(setq Sub-1 (make-instance 'Submarine))
(Depth Sub-1) ==> 100
(Speed Sub-1) ==> 5
(Set-Depth 200 Sub-1)
(Set-Speed 4 Sub-1)
(Depth Sub-1) ==> 200
(Speed Sub-1) ==> 4
Example 3

(defclass Submarine ()
   ((Depth :accessor Depth :initform 100)
    (Speed :accessor Speed :initform 5)))

(setq Sub-1 (make-instance 'Submarine))
(Depth Sub-1) ==> 100
(Speed Sub-1) ==> 5
(setf (Depth Sub-1) 200)
(setf (Speed Sub-1) 4)
(Depth Sub-1) ==> 200
(Speed Sub-1) ==> 4
type

- Specifies the type of the argument
- Can be used for optimization
allocation

- Specifies instance or class allocation
- Instance allocation means new copy of slot for each instance
  - instance variable in Java
- Class allocation means one copy of instance for all instances
  - class variable in Java
    - declared with "static" (yuk!)
Other Class Options

- documentation
  - doc string associated with class
- default-initargs
  - useful for overriding initforms from superclass
defgeneric

- Sort of like interface definition
- Specify method name and argument list
- Also
  - method-combination
  - argument-precedence-order
Method combination

- How to inherit methods?
  - normally evaluate most-specific method
  - polymorphism!

- But there are other possibilities
  - Discounts
    - Supplier discount = 10%
    - Preferred supplier discount = an additional 2%
  - We might want to specify that discounts are combined rather than overridden
    - in Java, must call super.method directly
(defgeneric get-discount (firm) 
   (:method-combination +))

(defclass supplier ()
   ((discount :initform 10)))

(defmethod get-discount + ((firm supplier))
   (slot-value firm 'discount))

(defclass preferred-supplier (supplier) ()

(defmethod get-discount + ((firm preferred-supplier))
   2)
Method Combination

- Specify type of method combination in generic function
- Each method is marked with its combination method
- Methods are called from least to most-specific
- Combination methods
  - progn, +, list, and, or, max, min, append and nconc
Example

- Polynomials
  - both terms and polynomials can be evaluated
- We can change our code to create
  - term objects
  - polynomial objects
Method combination

- What about canonicalization?
  - we'd like to have new instances canonicalized when they are created
- Can do this with an "after" method
  - write a method that is applied after initialize-instance does its thing
Example
Another example

- iterators
  - infinite iterators
  - integer iterators
  - list iterators
  - accumulation iterators
  - mapping iterators
Review

- What have we covered in this class?