1

Object-Oriented Programming Style

The general programming style encouraged by object-orientation has a number of features:

• Programs can be defined incrementally, through the addition of classes; this can be done through a refine-ment-by-specialization strategy

• Programs can be reused, through the reuse of classes, found in class libraries

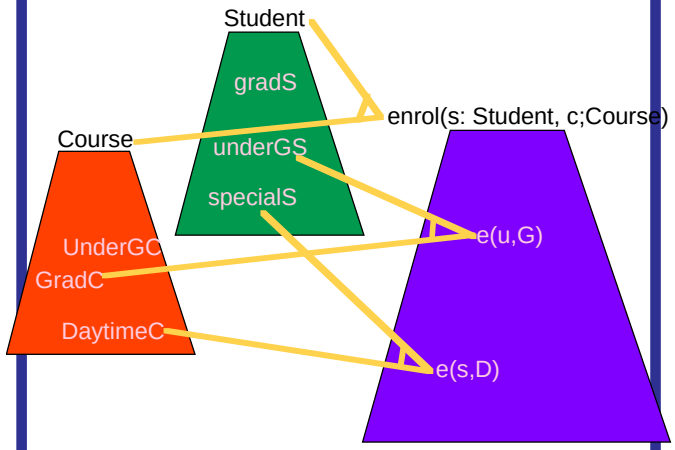
• Programs are easier to understand because of their taxonomic structure

©1993 John Mylopoulos

2

Refinement Through Specialization

Start by defining the most general classes relevant to your application, along with their associated methods, then refine these until you have build into your program sufficient detail



Courses X Students ≈ enrol(C,S)

©1993 John Mylopoulos

3

Designing Protocols

A protocol is a definition of the behaviour of a set of objects. Protocols are usually defined in terms of a set of generic functions which present an external view to the users of the set of objects. This view is intended to be independent of the implementation chosen.

For example, we may define a protocol for triangles with a set of functions:

```
(defun make-triangle (side-a side-b side-c)
    (   ...                                    ))
(defun change-side (tri which-side new-length)
    (   ...                                    ))
(defun fetch-side (tri which-side)
    (   ...                                    ))
(defun fetch-angle (tri which-angle)
    (   ...                                    ))
(defgeneric area (f figure)
    (   ...                                    ))
```

Note that this protocol is independent of the particular implementation (e.g., are all three sides stored as slots or computed)

©1993 John Mylopoulos

4

Designing Protocols

Useful guidelines on the design of protocols:

• Restrict the user's access to the internal data structures. The user should not even have to know what information is stored explicitly (as values of slots) and what is computed in terms of methods.

• Provide constructor functions for creating the internal data structures (like make-triangle)

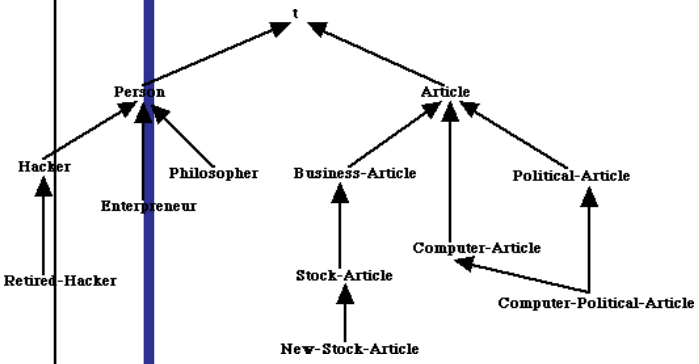
• Design the protocol in anticipation of specific user needs. For triangles, depending on the application, the sides may never change. If that is the case, do not provide functions such as change-side.

• Make allowances for program evolution. This may involve changing the implementation to make it more efficient (while the protocol remains unchanged) or changing the protocol to meet new user needs

©1993 John Mylopoulos

5

An Example: Persons and Articles



Interests

Person	Interest
hacker	computer articles
retired-hacker	political-articles
enterpreneur	business-articles software articles
philosopher	any articles

©1993 John Mylopoulos

6

Persons and Articles

```
                    ;; person class, three direct subclasses,
                        one sub-subclass
(defclass person () ((name :initarg :name)) )
(defclass hacker (person) ())
(defclass enterpreneur (person) ())
(defclass philosopher (person) ())
(defclass retired-hacker (hacker) ())
                    ;; class article, three direct subclasses,
                        five sub-subclasses

(defclass article () ((title :initarg :title)))
(defclass computer-article (article) () )
(defclass business-article (article) ())
(defclass political-article (article) ())
(defclass stocks-article (business-article) ())
(defclass new-stocks-article (stocks-article) ())
(defclass software-article (computer-article) ())
(defclass computer-political-article
                    (computer-article political-article) ())
```

©1993 John Mylopoulos

7

Interests Methods

```
(defmethod interested ((p hacker) (art computer-article)) 'yes)
(defmethod interested
                ((p retired-hacker) (art political-article)) 'yes)
(defmethod interested
                ((p enterpreneur) (art business-article)) 'yes)
(defmethod interested
                ((p enterpreneur) (art software-article)) 'yes)
(defmethod interested ((p philosopher) (art article)) 'yes)
(defmethod interested ((p art)) 'no)
```


8

Instances of Persons and Articles

```
(setf dan (make-instance 'hacker :name 'dan)
      gerry (make-instance 'hacker :name 'gerry)
      andre (make-instance 'retired-hacker :name 'andre)
      philip (make-instance 'enterpreneur :name 'philip)
      david (make-instance 'philosopher :name 'david)

      a-p-d (make-instance 'business-article
                  :title "memory prices down")
      current-stocks (make-instance 'new-stocks-article
                  :title "current stocks")
      m-s-u (make-instance 'computer-article
                  :title "memory speeds up")
      security (make-instance 'computer-political-article
                  :title "computer security")
      lispcraft (make-instance 'software-article
                  :title "lisp craft")
      m-i (make-instance 'political-article
                  :title "memory impunged")
```

Who is interested in what?

→ (interested dan m-p-d)
no

→ (interested dan m-s-u)
yes

→ (interested dan security)
yes

→ (interested dan lispcraft)
yes

→ (interested dan m-i)
no

→ (interested david current-stocks)
yes

©1993 John Mylopoulos