Functional Programming – Laboratory 12
CLOS (Common Lisp Object System)

Isabela Drămnesc

May 16, 2012

1 Concepts

- Classes, Instances
- Slot Properties
- Methods, Generic Methods
- Superclasses, Precedence
- Inheritance

2 Classes and instances:

```
> (use-package :clos)

; defining a class people with the components: lives,
; works, married

> (defclass people ()
    (lives works married))

; in order to create an instance of the class people

> (setq isabela (make-instance 'people))

> (describe isabela)

#<people #x19F33DED> is an instance of the CLOS class #1=#<STANDARD

-CLASS people>.
Slots:
  lives    unbound
  works    unbound
  married  unbound
```

3 Slot Properties

Modify and access the component values:

```
; we use :initform in order to set an implicit value to
; one component

> (defclass people ()
      ((lives :initform 'Timisoara)
       (works :initform 'UVT)
       (married :initform '??)))

> (setq isabela (make-instance 'people))

> (describe isabela)

; if we want to set an explicit particular value
; we use the keyword :initarg

> (defclass people ()
     ((lives :initform 'Timisoara)
      (works :initform 'UVT :initarg :works_for_now)
      (married :initform 'no :initarg :married_initial)))

> (setq ionela (make-instance 'people :works_for_now
'Alcatel))

> (describe ionela)

> (setq ionut (make-instance 'people :works_for_now 'Conti
:married_initial 'yes))

> (describe ionut)

> (setq irina (make-instance 'people :married_initial 'yes))

> (describe irina)

;; we access a component value by using slot-value

> (setq florian (make-instance 'people))

> (slot-value florian 'lives)

> (setf (slot-value florian 'lives) 'Cluj)

> (describe florian)

;;; reading, writing or accessing a certain component
; can be made easily by defining: reader, writer, accessor

> (defclass rectangle ()
      ((length :reader length? :writer length! :accessor
leng)
```

```
        width))

> (setq figure1 (make-instance 'rectangle))

> (describe figure1)

> (setf (leng figure1) 10)

> (length? figure1)

> (leng figure1)

> (describe figure1)

;;; another option for modifying the component values of an
; object by allocation for a class and not for an instance

> (defclass student_at_info_UVT ()
      ((address_UVT :initform 'V_Parvan :allocation :class)
       (has_classes :initform 'informatics)))

> (setq student1 (make-instance 'student_at_info_UVT))

> (describe student1)

> (setq student2 (make-instance 'student_at_info_UVT))

> (describe student2)

> (setf (slot-value student1 'address_UVT) 'Bogdanesti)

> (describe student2)

> (describe student1)
```

4 Superclasses, Precedence, Inheritance

```
;;;; properties inheritance in CLOS

> (defclass engineer (people) ())

> (defclass soft_developer (engineer) ())

> (defclass doctor (people) ())

> (defclass doctor_dev_soft (doctor soft_developer) ())

> (subtypep 'doctor_dev_soft 'engineer)
```

```
> (setq dan (make-instance 'doctor))

> (subtypep (type-of dan) 'people)

> (subtypep (type-of dan) 'engineer)

;; in order to establish the order  order of the inherited
; component values, we can construct for each class the
; class precedence list

> (clos::class-precedence-list (find-class 'doctor_dev_soft))
(#<STANDARD-CLASS DOCTOR_DEV_SOFT> #<STANDARD-CLASS DOCTOR>
 #<STANDARD-CLASS soft_developer> #<STANDARD-CLASS engineer>
 #<STANDARD-CLASS people :VERSION 2> #<STANDARD-CLASS STANDARD-
OBJECT>
 #<BUILT-IN-CLASS T>)

> (clos::class-direct-superclasses (find-class 'doctor_dev_soft))
```

4.1 Hierarchy of classes:

```
;;;; example:

> (defclass c1 ()
     ((s1 :initform 1 :initarg :1s1 :accessor a1s1)))

> (defclass c2 (c1)
    ((s1 :initform 2 :initarg :2s1 :accessor a2s1)))

> (defclass c3 (c1)
     ((s1 :initform 3 :initarg :3s1 :accessor a3s1)))

> (defclass c4 (c1)
     ((s1 :initform 4 :initarg :4s1 :accessor a4s1)))

> (defclass c5 (c1)
       ((s1 :initform 5 :initarg :5s1 :accessor :a5s1)))

> (defclass c6 (c2 c3)
       ((s1 :accessor a6s1)))

> (defclass c7 (c4 c5)
        ((s1 :initform 7 :accessor a7s1)))

> (defclass c8 (c6 c7)
         ((s1 :accessor a8s1)))

;;; the class precedence list for the class c8 is:
```

```
> (clos::class-precedence-list (find-class 'c8))
(#<STANDARD-CLASS C8> #<STANDARD-CLASS C6> #<STANDARD-CLASS C2>
 #<STANDARD-CLASS C3> #<STANDARD-CLASS C7> #<STANDARD-CLASS C4>
 #<STANDARD-CLASS C5> #<STANDARD-CLASS C1> #<STANDARD-CLASS
STANDARD-OBJECT>
 #<BUILT-IN-CLASS T>)

> (setq instance-of-c8 (make-instance 'c8))

> (slot-value instance-of-c8 's1)
2    ;the inherited value from c2 (the first defined value in the
     ; class precedence list)

;; all the specifications :accessor and :initarg are inherited and
; can be used in the class c8

> (a1s1 instance-of-c8)

> (a8s1 instance-of-c8)

> (setq instance-of-c8-2 (make-instance 'c8 :4s1 29))

> (a1s1 instance-of-c8-2)
```

5 Methods, Generic Methods

Define methods:

```
> (defmethod speaks ((him people) something)
      (format t "~%_I_live_in_~A_I_work_at_~a_and_I_speak_~A"
       (slot-value him 'lives)
       (slot-value him 'works)
        something)
     'end)

> (speaks ionut 'about_LISP)

> (speaks ionela 'something_new?)

> (defclass dog ()
    (color name))

> (setq doggie (make-instance 'dog))

> (defmethod speaks ((him dog) something) (print 'hamham))

> (speaks doggie 'lala)

;;; the methods can be inherited

> (setq danut (make-instance 'doctor_dev_soft
```

```
:works_for_now 'La-PC))

> (describe danut)

> (speaks danut 'what_I_want)

;;;; the method speaks is not defined in the class
; doctor_dev_soft, but is inherited from the class people

> (speaks ionela 'a_lot)

;; the set of methods with the same name form a generic function.
; Each list of applicable methods is a subset of the methods of
; a certain generic function.

; the customization of a method for a certain instance:

> (defmethod speaks ((him (eql ionut)) something)
          (declare (ignore something))
        (call-next-method)
       (print 'I_m_getting_married)
      'out)

[87]> (speaks ionut 'something)

; call-next-method calls the next applicable method
; this is a way of combining methods.
; In order to see this list please study the following
; example:

> (defmethod mmm ((him people)) (print 'people))

> (defmethod mmm ((him engineer))
                 (print 'engineer)
                 (call-next-method))

> (defmethod mmm ((him doctor))
       (print 'doctor)
       (call-next-method))

> (defmethod mmm ((him doctor_dev_soft))
              (print 'doctor)
            (call-next-method))

> (mmm ionut)

> (mmm dan)

> (mmm danut)
```

```
> (defmethod is_getting_married ((him people) (her people))
        (setf (slot-value her 'lives) (slot-value him
'lives)
              (slot-value him 'married) 'yes
              (slot-value her 'married) 'yes))

> (describe ionela)

> (describe danut)

> (is_getting_married danut ionela)

> (describe ionela)

> (describe danut)
```

6 Exercise from [St.Trausan-Matu]

```
> (use-package :clos)

> (defclass physical_object ()
     ((material :initarg :material)
      (color :initarg :color)))

> (defclass spherical_object (physical_object)
     ((radius :initarg :radius)))

> (defmethod volume ((x spherical_object))
     (* 4 pi (expt (slot-value x 'radius) 3)))

> (defclass cubic_object (physical_object)
      ((side :initarg :side)))

> (defmethod volume ((x cubic_object))
      (expt (slot-value x 'side) 3))

> (defclass object_from_plastic (physical_object)
      ((material :initform 'plastic)))

> (defclass object_from_iron (physical_object)
      ((material :initform 'iron)))

> (defclass ball (spherical_object object_from_plastic) ())

> (setf ball1 (make-instance 'ball :color 'red :radius 2)
            ball2 (make-instance 'ball :color 'white :radius 3))

> (slot-value ball2 'material)
```

```
> (volume ball1)

> (volume ball2)

> (describe ball1)

> (defclass cube (cubic_object object_from_iron) ())

> (setf cube1 (make-instance 'cube :side 10 :color
'purple))

> (describe cube1)

> (volume cube1)
```

7 Homework (deadline: next lab)

1. Follow the exercises above and create a program with the following requirements:

- a class Shape with the components: name, a generic method to calculate the area and a method for printing the shape;
- a class Circle which inherits the name and the method area from the class Shape and in plus has a component radiuscircle and a method for printing;
- a class Triangle Circle which inherits the name and the method area from the class Shape and in plus has a method for printing;
- a class Rectangle: length, width, a method to calculate the area;
- o class Square: the side of a square is set to be the length of a rectangle, a method to calculate the area and a method for printing.
- create at least two instances for each class;
- send messages between the objects of the classes;