Clos-TD1
Introduction au langage

Mireille Blay et Stéphane Ducasse
ESSI3-DESS-DEA

June 11, 1996

L'objectif de ce TD est de vous familiariser avec le langage Clos. Aussi allons-nous au travers d'exemples simples tenter d'acquérir les concepts fondamentaux du langage: création de classes, héritages, appel des fonctions génériques.

Environnement de travail

Le programme *clos* se trouve dans `~ arditi/clisp`.

Quelques fonctions Common-Lisp indispensables

Nous vous avons fourni un memento des fonctions de Common Lisp en voici quelques unes:

quit : pour sortir de Common-Lisp.

load : chargement d'un fichier.

format : pour afficher du texte.

```
(format t " ceci est ~D ~S de test avec ~D sauts de ~S ~% ~% "
1 'exemple 2 'lignes )
```

1 Définition de classes

1. Définir la classe **laboratoire** ayant les slots :
   - **nom** : avec
     - pour arguments d'initialisation **:nom-labo**
     - l'accesseur `nom`
   - **adresse** : avec
     - pour arguments d'initialisation **:adresse**
     - l'accesseur `adresse`
   - **chercheurs-associés** : avec
     - l'accesseur `chercheurs`
     - la valeur par défaut ()
     - pour arguments d'initialisation **:chercheurs**
   - **nbre-chercheurs** : slot partagé par toutes les instances avec
     - pour valeur par défaut 0

    - l'accesseur en lecture `nbre-chercheurs`
  - **nbre-labos** : slot partagé par toutes les instances avec
    - pour valeur par défaut 0
    - l'accesseur en lecture `nbre-labos`

2. Définir la classe **personne** ayant les slots :

   - **nom** : avec
     - pour arguments d'initialisation **:nom** et **:name**
     - l'accesseur `nom`
   - **fils** : avec
     - l'accesseur `fils`
     - la valeur par défaut ()
   - **pere** : avec
     - l'accesseur `pere`

   Par défaut, l'argument d'initialisation **nom** a pour valeur le symbole `dupont`.

Notes : Vous pourrez préciser le type des slots et prévoir une documentation.

Tests :

```
(setq adam (make-instance 'personne :nom 'adam))
(setq jean (make-instance 'personne))
(describe adam)
(slot-value jean 'nom)
```

2 Initialisation d'instances et accès aux slots

1. La création d'un nouveau laboratoire implique de mettre à jour le slot partagé **nbre-labos**. (Indice: pensez à la fonction `initialize-instance`).

2. Compléter l'accesseur `chercheurs` de la classe laboratoire de telle sorte qu'une affectation d'une **liste** de chercheurs à un laboratoire modifie le slot partagé **nbre-chercheurs**. Attention, il faut décrémenter (`decf`) avant affectation puis incrémenter (`incf`) aprés affectation ce slot en fonction de l'ancienne liste et de la nouvelle liste de chercheurs. (Indice: pensez aux :before et :after).

   Tests :

   ```
   (setq i3s (make-instance 'laboratoire :nom-labo "i3S" :chercheurs '(jean)))
   (nbre-labos i3s)  1
   (nbre-chercheurs i3s)  0
   (setf (chercheurs i3s) (list 'iaa 'ibb 'iccc))
   (nbre-chercheurs i3s)  3
   (setf (chercheurs i3s) (list 'iaa 'ibb 'iccc 'jean))
   (nbre-chercheurs i3s)  4
   ```

3. Il s'agit d'implanter la relation père-fils au travers des slots *pere* et *fils*.
   ∀ `X, (slot-value X 'pere) = P si et seulement si (member X (slot-value P 'fils))`.

   Nous sommes donc amenés à redéfinir :

- la fonction d'initialisation d'instance
- l'affectation de fils ou d'un père à une personne.

4. Ecrire une fonction générale, qui, lors d'un accès en lecture à un slot n'ayant pas de valeur, renvoie la chaîne "sans valeur" et, dans le cas d'un accès au slot `pere` alors que celui-ci n'a pas de valeur, renvoie la chaîne "pere inconnu".

3 Définition de sous-classes

1. Définir une sous-classe *chercheur* de *personne*, ayant les champs :

   - **nbr-membres** : dont la valeur par défaut correspond au nombre de chercheurs affiliés à l'I3S.
   - **laboratoire** : avec
     - pour arguments d'initialisation :labo
     - l'accesseur `labo`
     - la valeur par défaut "i3s"

   et tel que le nom par défaut d'un chercheur soit "Einstein" (utilisation de *:default-initargs*).

   De plus l'affectation d'un laboratoire à un chercheur implique de l'ajouter à la liste des chercheurs associés aux laboratoire.

2. Définir les sous-classes de *laboratoire* *labo-etat* et *labo-privé* laquelle redéfinit les slots *nbre-labos* et *nbre-chercheurs* comme des slots d'instances.

   Tests :
   `(setq IMRA (make-instance 'labo-prive))`
   `(setq BULL (make-instance 'labo-prive))`
   `(nbre-labos i3s)` *1*
   `(nbre-labos IMRA)` *1*
   `(setq INRIA (make-instance 'labo-etat))`
   `(nbre-labos i3s)` *2*
   `(nbre-labos INRIA)` *2*
   `(nbre-labos IMSA)` *2*
   `(setq Paul (make-instance 'chercheur :nom 'Paul))`
   `(describe Paul)` *numero et labo*
   `(chercheurs i3s)`
   `(nbre-chercheurs INRIA)`
   `(nbre-chercheurs IMRA)` *0*
   `(setq sugy (make-instance 'chercheur :nom 'sugy :labo IMRA))`
   `(nbre-chercheurs INRIA)` *non modifié*
   `(nbre-chercheurs IMRA)`*1*
   `(nbre-chercheurs Bull)`*0*

3. Pour l'exercice suivant, nous utiliserons la hièrarchie suivante :

   - une sous-classe *etudiant* de *personne*, ayant le slot *université.*
   - une sous-classe *thesard* de *etudiant* et *chercheur*, telle que un argument d'initialisation du slot *laboratoire* soit celui de l'université.
   - une sous-classe *linguiste* de *chercheur*, qui définit le slot *spécialité.*

- une sous-classe *informaticien* de *chercheur*, qui définit le slot *langage-de-programmation*.
- une sous-classe *informaticien-linguiste* de *informaticien* et *linguiste*.
- une sous-classe *thesard-info-ling*de *informaticien-linguiste* et *thesard*.

4 Héritage et combinaison de méthodes

1. Définir la classe *revue* et la sous-classe *revue-info* puis définir les méthodes *transmettre* qui permettent la transmission d'une revue scientifique à un chercheur, déclare non recommandée la transmission d'une revue quelconque à un informaticien et déclare non recommandée la transmission d'une revue d'informatique à un chercheur quelconque. Tester la transmission d'une revue d'informatique à un informaticien.

2. Que se passerait-t-il si nous pouvions redéfinir l'ordre de précédence des arguments avec `:argument-precedence-order`.

3. L'objectif de cet exercice est de tester les combinaisons standards de méthodes (*before, after, around*).

   Définir une fonction *Display* qui affiche le statut d'une personne en fonction de sa classe, puis marque le passage à une description plus complète, et enfin présente la personne (nom, laboratoire, ...). Utiliser les combinaisons de méthodes sur toute la hierarchie.

5 Contrôle sur les accès aux slots

1. Redéfinir *no-applicable-method* de telle sorte que, lorsque l'on demande l'adresse d'un chercheur, on est l'adresse de son laboratoire.

2. Redéfinir la fonction générique
   slot-missing *class object slot-name operation &optional new-value* → *{result}**
   pour qu'un accès en lecture au slot *bilan* d'un laboratoire retourne une valeur calculée (durant ce TP, la valeur retournée sera, par exemple, une constante).

6 Changement de classe

1. Définir la classe *review* ayant le slot *contents*, avec une valeur par défaut et l'accesseur *contents*.

2. Ecrire la méthode qui permet lors du changement de classe d'une instance de la classe *revue* en une instance de la classe *review* d'initialiser le slot *contents* de cette dernière avec la valeur de l'ancien slot *contenu* en faisant éventuellement appel à une pseudo fonction de traduction.

   tests :
   ```
   (setq r (make-instance 'revue))
   (change-class r 'review)
   (describe r)
   "version anglaise de :  informations scientifiques"
   ```