ANNÉE UNIVERSITAIRE 2007/2008
2IÈME SESSION DE PRINTEMPS

Parcours : CSB6
Code UE : INF356
Épreuve : Projet de Programmation 3
Date : juin 2008
Heure : h
Durée : 1h30
Documents : autorisés
Épreuve de Mme Irène Durand

Le barème est donné à titre **indicatif**. Le sujet comporte 4 pages dont une annexe.

Exercice 1 *(3pts)*

On dispose de `SBCL` et de `SLIME` sous `Emacs` et on est connecté à Internet. On souhaite installer la bibliothèque `mcclim` disponible à partir des sites standards.

1. Donner la suite d'expressions qui permet d'installer cette bibliothèque.
2. Citer les solutions possibles pour que cette bibliothèque soit chargée automatiquement à chaque fois qu'on lance `SBCL`.

Exercice 2 *(4pts)*

Soit le fichier texte `data.txt` dont le contenu est le suivant :

```
ligne un
123 (+ 1 2 3)
"une chaine" 10.2
```

1. Donner une expression permettant d'ouvrir un flot en lecture sur ce fichier et de le mémoriser dans la variable *flot*.
2. Quel serait le résultat (effet, valeur de retour) de chacun des appels suivants effectués séquentiellement.

```
(read-line *flot*)
(read *flot*)
(read *flot*)
(read *flot*)
(read *flot* nil 'fin)
(read *flot* nil 'fin)
(read *flot* nil 'fin)
```

Exercice 3 *(6pts)*

Soit le bout de programme words.lisp donné en annexe qui permet de manipuler des *mots* constitués de *lettres*. À partir des fonctions `make-letter` et `make-word`, on peut créer des lettres et des mots.

```
CL-USER> (defparameter *l-a* (make-letter 'a))
*L-A*
CL-USER> (defparameter *m-abb*
           (make-word
             (list (make-letter 'a) (make-letter 'b) (make-letter 'b))))
*M-ABB*
CL-USER> *l-a*
{A}
CL-USER> *m-abb*
[{A}{B}{B}]
```

Cependant, dans certains cas, on préfèrerait pouvoir taper nos lettres et mots directement avec la syntaxe suivante : les lettres sont entourées de `{}` et les mots de `[]`.

```
CL-USER> {a}
{A}
CL-USER> [{a}{b}{b}]
[{A}{B}{B}]
```

Proposer une solution à l'aide de read-macros.

Exercice 4 *(7pts)*
On se place dans le cadre d'une application gérant des comptes bancaires.

```
(defclass compte () ...)
(defgeneric transfert (montant compte1 compte2)
  :documentation "tranfere MONTANT du COMPTE1 vers COMPTE2")
```

1. Définir la classe `compte` avec un créneau `solde`, un accesseur `solde`, un mot-clé d'initialisation `:solde` et une valeur initiale de 0.00.

   ```
   CL-USER> (defparameter *compte1* (make-instance 'compte :solde 10000))
   *COMPTE*
   ```

2. Modifier l'implémentation de l'opération `print-object` de sorte que le solde soit affiché comme indiqué dans l'exemple qui suit.

   ```
   CL-USER> *compte1*
   #<COMPTE {AFC7761}> solde: 10000.00
   ```

3. Implémenter l'opération `transfert` sans vérification sur le solde des comptes.

   ```
   CL-USER> (defparameter *compte2* (make-instance 'compte :solde 500))
   *COMPTE2*
   CL-USER> *compte2*
   #<COMPTE {B01AAD9}> solde: 500.00
   CL-USER> (transfert 300 *compte1* *compte2*)
   800
   CL-USER> *compte1*
   #<COMPTE {AF7BEA1}> solde: 9700.00
   CL-USER> *compte2*
   #<COMPTE {B01AAD9}> solde: 800.00
   ```

Certains comptes sont *plafonnés*. Le fait d'être plafonné est capturé par une classe *mixin* `plafonne-mixin` avec un créneau `plafond` destiné à mémoriser la valeur du plafond.

```
(defclass plafonne-mixin ()
  ((plafond :initarg :plafond :reader plafond :initform nil)))
```

4. En utilisant la classe mixin `plafonne-mixin`, définir une classe concrète `compte-plafonne`.
5. Modifier l'implémentation de `print-object` de manière à obtenir l'affichage suivant

```
CL-USER> (defparameter *cp*
           (make-instance 'compte-plafonne :solde 1000 :plafond 5000))
*CP*
CL-USER> *cp*
#<COMPTE-PLAFONNE {B07EAD9}> plafond: 5000.00 solde: 1000.00
```

6. En utilisant une méthode `:before` sur l'opération `transfert` ainsi que la macro `assert`, faire en sorte qu'une erreur soit signalée lorsque le plafond est dépassé.

```
CL-USER> (transfert 4500 *compte1* *cp*)

The assertion (< (+ (SOLDE COMPTE2) MONTANT) (PLAFOND COMPTE2)) failed.
   [Condition of type SIMPLE-ERROR]
...
CL-USER> (transfert 500 *compte1* *cp*)
1500
```

Annexe

words.lisp

```
(defun display-list (l stream &key (sep " "))
  (when l
    (mapc (lambda (e)
            (format stream "~A~A" e sep))
          (butlast l))
    (format stream "~A" (car (last l)))))

(defclass letter ()
  ((name :initarg :name :reader name))
  (:documentation "class of letters"))

(defmethod print-object ((letter letter) stream)
  (format stream "{~A}" (name letter)))

(defmethod make-letter (name)
  (make-instance 'letter :name name))

(defclass word ()
  ((letters :initform '() :initarg :letters :reader letters))
  (:documentation "class of words"))

(defmethod make-word ((letters list))
  (make-instance 'word :letters letters))

(defmethod print-object ((word word) stream)
  (format stream "[")
  (display-list (letters word) stream :sep "")
  (format stream "]"))
```