Université Bordeaux 1 Licence S&TS Mention Informatique S6 2009/2010

UE INF356 **Projet de Programmation 3**

Devoir surveillé

Tous documents autorisés. Mardi 16 Mars 2010
Durée : 1h20.
Le barème est donné à titre **indicatif**.

Exercice 1 *(2pts)*

Soit un système Lisp nommé `my-system`.

1. En supposant `my-system` installé de manière standard sur le compte de l'utilisateur, quelle expression permet de charger ce système dans `SBCL`.
2. Supposons maintenant que `my-system` ne soit pas installé de manière standard mais dans le répertoire `~/lisp/` avec les sources (contenant un fichier `my-system.asd`) dans le répertoire dans `~/lisp/sources/` et un lien vers le fichier `my-system.asd` dans le répertoire `~/lisp/systems/`. Proposer une solution pour charger ce système.

Exercice 2 *(12pts)*

On souhaite gérer un stock d'objets ayant un poids.
À sa création, un objet se voit attribuer un numéro unique donné par le compteur `*numero-objet*`.
Certains objets appelés `conteneurs` peuvent contenir d'autres objets. Le poids total d'un conteneur est son poids à vide plus le poids des objets qu'il contient.
Supposons exécuté l'extrait de code donné en Annexe Page 3.

1. Donner les valeurs de retour des appels suivants :

```
CL-USER> (numero *o10*)
                                        ;; réponse 1
CL-USER> (poids *o10*)
                                        ;; réponse 2
CL-USER> (length (objets *conteneur*))
                                        ;; réponse 3
CL-USER> (numero *conteneur*)
                                        ;; réponse 4
CL-USER> (poids *conteneur*)
                                        ;; réponse 5
CL-USER> (poids-total *conteneur*)
                                        ;; réponse 6
```

2. Compléter le code de manière à avoir le comportement suivant :

```
CL-USER> *o1*
#<OBJET {1002861401}> Numero: 1, Poids: 1
```

3. Définir une opération `ajout-objet (objet conteneur)` qui rajoute un `objet` dans un `conteneur` s'il ne s'y trouve pas déjà.
4. Définir une nouvelle classe pour des conteneurs ayant un poids total limité.

5. Proposer un mécanisme qui empêche l'ajout d'un objet dans un conteneur à poids total limité si le poids total final dépasse la limite.
6. Redéfinir l'opération `poids-total` en utilisant une méthode non-standard de manière à éviter l'appel à `call-next-method` dans la méthode `poids-total` définie pour `conteneur`. Indication : utiliser une combinaison de méthodes non standard.

Exercice 3 *(6pts)*

1. Soit la séquence suivante :

```
(set-macro-character
 #\&
 (lambda (stream char)
   (declare (ignore char))
   (list 'make-array (read stream t nil t) :initial-element 1)))
```

   Donner les valeurs de retour des appels suivants :

```
CL-USER> (read)
&4
                        ;; réponse 1
CL-USER> &4
                        ;; réponse 2
```

2. Donner une suite d'expressions qui après évaluation permet d'obtenir le comportement suivant :

```
CL-USER> (read)
#{1 2 3}
(APPLY #'+ '(1 2 3))
CL-USER> #{1 2 3}
6
```

FIN

Annexe

```
(defvar *numero-objet*)

(defgeneric poids (o)
  (:documentation "Poids d'un objet à vide"))

(defgeneric poids-total (o)
  (:documentation "Poids d'un objet rempli"))

(defclass objet ()
  ((poids :reader poids :initform 0 :initarg :poids)
   (numero :initform (incf *numero-objet*) :reader numero)))

(defmethod poids-total ((o objet))
  (poids o))

(defun make-objet (poids)
  (make-instance 'objet :poids poids))

(defclass conteneur (objet)
  ((objets :accessor objets :initarg :objets)))

(defmethod poids-total ((conteneur conteneur))
  (+ (call-next-method)
     (loop
       for objet in (objets conteneur)
       sum (poids-total objet))))

(setf *numero-objet* 0)

(defparameter *o1* (make-objet 1))
(defparameter *o10* (make-objet 10))
(defparameter *o100* (make-objet 100))

(defparameter *conteneur*
  (make-instance
   'conteneur
   :objets (list *o1* *o10* *o100*)
   :poids 50))
```