LOO

Langages orientés objets

Licence d'informatique,
Université de Bretagne Occidentale

31 mai 2001

Question 1

- Dans le fragment de programme suivant, faire un décompte *minimum* des objets créés et détruits :

```
 | v  p f |
v := 1 / 3.
p := v @ 2.
p x: ( p y / p x).
p := v - 1.
```

- Expliquer le principe de fonctionnement des ramasse-miettes incrémentaux. Donner un exemple de programme Smalltalk-80 montrant sur quelques instructions une situation où ce ramasse-miette sera inopérant. (environ 10 à 15 lignes rédigées)

Question 2

1. On se donne un tableau A formé de nombres arbitraires appartenant à différentes classes.
   (a) former un tableau contenant les carrés de ces nombres, triés par ordre croissant, et en éliminant les doublons.
   (b) calculer le nombre de classes ayant des instances dans A.
   (c) former un dictionnaire dont les clés sont les noms de classes présentes dans A, chaque valeur associée à une clé C étant un tableau regroupant les instances de cette classe C présentes dans A.
2. On se donne deux nombres entiers x et y, calculer tous les tableaux triés distincts contenant au plus 5 éléments égaux soient à x, soient à y.

Question 3

Modélisation

On souhaite construire et représenter graphiquement des arbres généalogiques. Ces arbres sont formés à partir de couples, associant une femme et un homme, et à partir de personnes qui sont soit des enfants d'un couple, soit des racines de l'arbre.

1. On demande de construire 2 classes Smalltalk-80, dont on donnera les variables d'instances, les méthodes de classes et les méthodes d'instances les plus significatives, pour représenter cette situation.
2. Programmer un exemple simple faisant appel à ces classes dans le but de construire un arbre généalogique correspondant à la situation suivante :
   - Rémi et Janine se marient.
   - Ils ont deux enfants Erwan et Aziz.
   - Aziz se marie avec Rebecca.
   - Ils ont une fille Mathilde.

Présentation graphique

On souhaite présenter les arbres généalogiques dans des interfaces très simples, où on fera apparaitre les couples et les personnes.

Donner les éléments de programme permettant de réaliser cette opération à partir des classes décrites ci-dessus et du widget `objgraph`. Commenter précisément la manière d'obtenir des formes textuelles pour les objets présentés.

Expliquer comment, en le configurant et en programmant vos classes, vous permettez au widget de traverser votre graphe, pourtant formé de nœuds et de feuilles appartenant à des classes différentes.

LOO,
LICENCE D'INFORMATIQUE – U.B.O.

C. Dezan et B.Pottier

Mai 2002

1 Modélisation

On désire modéliser la gestion d'un parking payant. Ce parking comporte plusieurs niveaux qui peuvent être fermés si 95% des places d'un niveau sont occupées (situation niveau complet). Chaque automobiliste entrant dans le parking retire un ticket précisant l'heure d'arrivée du véhicule. Au moment de sortir du parking, le client doit s'acquitter des frais de stationnement qui sont évalués par tranche d'une heure (1,50 *euro* par heure) si la durée de stationnement est inférieure à 6 heures, sinon à la journée (10 *euros* par jour). Pour connaitre le taux de remplissage du parking, une camera scrute régulièrement chacun des étages pour connaître le nombre de places disponibles. Toutes les heures, ces données sont enregistrées est permettent de faire des estimations sur le taux de remplissage des différents étages. Ces estimations peuvent conduire notamment à fermer un étage si le taux de remplissage estimé est inférieur à 10% du nombre de places disponibles de l'étage (situation niveau sous-utilisé).

Questions

1 - Proposer un jeu de classes permettant de modéliser l'ensemble de la problématique du parking exposé ci-dessus. Précisez la structure de données que vous utilisez pour stocker les données enregistrées par la caméra.
2. Définir la(les) méthode(s) nécessaire(s) pour la création d'un parking où le nombre d'étages et le nombre de places total disponibles sont passées en paramètre (on suppose que les différents étages possède un nombre égal de places).
3. Définir une méthode permettant d'enregistrer un client nouveau arrivant dans le parking.
4. Définir une méthode permettant de calculer pour un client voulant recupérer sa voiture dans le parking, le prix de son stationnement.
5. Définir une méthode permettant de calculer le taux de remplissage du parking. Cette méthode renvoie pour chaque étage son taux de remplissage par heure et par jour .
6. Définir une méthode qui permet de calculer le taux de remplissage estimé pour chaque étage par jour et par heure. L'estimation est faite en faisant la moyenne des informations enregistrées sur un mois pour un jour et une heure considérés.
7. Définir une nouvelle méthode qui à partir du taux de remplissage estimé de chaque étage, pour un jour et une heure donnés conclut à la fermeture d'un étage car il est considéré soit comme complet, soit comme sous-utilisé. Un étage fermé est un étage sur lequel une nouvelle voiture n'a plus le droit de stationner mais les anciennes voitures de cet etage ne partent que lorsque leurs propriétaires viennent les récupérer.

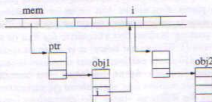
2 Ramasses-miettes

La classe Memory représente une mémoire d'objets. Les instances de Memory contiennent un tableau *theObjects* de taille n via lequel on entend gérer des objets. On s'intéresse à un objet de classe Memory, que l'on désigne par *mem*, et qui va nous servir à modéliser une mémoire objet pour expliciter quelques questions de cet examen.

Chaque élement du tableau de *mem* est une instance de la classe MemObjectPtr. Les objets de cette classe ont un champs *object* permettant de référencer un objet de classe MemObject, et des champs utilitaires destinés à la gestion de la mémoire que vous pourrez librement définir. Les instances de MemObject ont une variable d'instance *fields*, de classe Array, permettant de conserver des index sur d'autres *MemObjectPtr* enregistrés dans *mem*. Un index est un entier précisant quel objet de *mem* on veut accéder.

La figure 2 montre l'organistion des objets :

- *mem* appartient à la classe Memory.
- *ptr* appartient à la classe MemObjectPtr.
- *obj1*, *obj2* appartiennent à la classe MemObject.



2.1 Allocation et libération

La classe Memory possède une variable *freePtrs* qui permet de retrouver une collection des cases de *mem* non occupées par des objets.

Elle possède également une variable *freeObjects* qui permet de retrouver la tete de liste des objets déja alloués, mais qui ne sont plus utilisés.

Chaque case du tableau de *mem* non-utilisée peut etre marquée libre en y plaçant la valeur *nil*.

1. On demande de décrire les opérations d'allocation d'un objet de classe MemObject de taille *size* dans une méthode *basicNewObject : size* de la classe Memory.
2. On demande de décrire les opérations de libération (*basicFree : index*) d'un couple pointeur-objet présent dans *mem* à l'index *index*. On ne doit perdre ni le pointeur (MemObjectPtr), ni l'objet (MemObject).

Ces deux questions doivent bien entendu etre traitées de manière cohérente et en s'appuyant sur des fonctions d'accès aux deux listes de pointeurs et d'objets libres dont on décrira la structure et les opérations (insertion et suppression).

2.2 Ramasse-miettes

1. **incrémental** : en vous donnant le support nécessaire dans la classe MemObjectPtr, décrire précisément l'implantation des opérations de lecture et d'écriture dans un objet : `basicObject : index at : fieldNumber` et `basicObject : index at : fieldNumber put : anotherIndex`.
   La seconde opération doit provoquer des modifications dans les pointeurs des objets désignés par des indexs, et une libération éventuelle.
2. **mark-and-collect** : on se donne une collection d'objet *roots* qui constituent les racines des arborescence utiles.
   Décrire un algo de ramasse-miettes qui marque les objets utiles, puis libère ceux qui n'ont pas étés visités.