SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PCOMP(LU3IN032)
Année 2022/2023
Séance 3 de TD/TME

Objectifs

1. programmation objet
   - classes, liaison tardive et héritage
   - Design Patterns : composite et visiteur
   - comparaison avec la programmation fonctionnelle
   - introspection

Travaux Dirigés

Exercice 1 : Objets géométriques & graphiques (classe, héritage, liaison tardive)

Q.1.1 Définir la classe de base des formes géométriques comportant

- des opérations `area` pour calculer la surface et `perimeter` pour calculer son périmètre.
- le nom de la forme stocké dans une variable d'instance qui sera définie dans les sous-classes,
- une méthode `get_name` générique (non virtuelle) donnant le nom de la forme.

Q.1.2 Écrire la fonction qui prend une liste de telles formes géométriques et affiche leurs aires et surfaces.

Q.1.3 En dériver deux classes pour les disques et les rectangles, puis dériver une classe pour les carrés héritant de celle des rectangles.

Q.1.4 Appeler la fonction d'impression sur une liste constituée d'instances des classes que vous venez de définir.

Q.1.5 Ajouter à la classe des formes une méthode `print` qui affiche son nom, sa surface et son périmètre. Décriver le fonctionnement de cette fonction (en particulier quelles méthodes sont appelées) sur une instance de carré.

Q.1.6 Que vaut `(new square 2. : rectangle)#get_name` ? L'expression `(new circle 3. : rectangle)#get_name` est-elle bien typée ? Expliquez.

Q.1.7 Un décorateur est un patron de conception (*design pattern*) permettant de changer le comportement d'un objet sans changer son type (il peut ainsi être utilisé partout où l'objet qu'il décore peut l'être).

Écrivez un décorateur pour ajouter un peu de couleurs aux formes (un `"rectangle"` pourra devenir un `"red rectangle"` par exemple.).

On veut aussi pouvoir obtenir et modifier la couleur d'un objet via les méthodes `color` et `set_color`.

Q.1.8 Écrivez les fonctions `red` et `blue` coloriant des objets et donnez un exemple d'utilisation en imprimant une liste d'objets colorés.

Exercice 2 : Arbres homogènes (Composite)

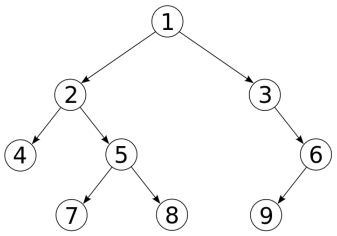
On cherche à définir une petit bibliothèque sur les arbres d'étiquettes homogènes. Le langage d'implantation est libre (Java, OCaml, Python). Le type de l'étiquette des arbres peut être indiqué comme paramètre de type.

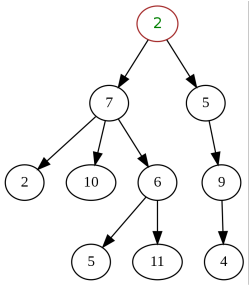
Q.2.1 Définir une interface ou une classe abstraite `tree` contenant les déclarations des quatre méthodes suivantes : `size` qui compte le nombre de nœuds de l'arbre, `height` sa hauteur, `mem` qui prend un élément et indique s'il appartient à l'arbre et `toList` qui construit la liste des étiquettes de l'arbre.

Q.2.2 Définir selon le " design pattern " Composite deux sous-classes concrètes de `tree`, la première `emptyTree` pour construire des arbres vides, la deuxième `binaryNode` pour construire des arbres binaires.

Q.2.3 Définir une sous-classe `naryNode` pour construire des arbres n-aires, des nœuds possédant entre 0 et n fils.

Q.2.4 Construire les arbres suivants et indiquer les résultats des appels de `.size()`, `.height()`, `.mem(3)` et `.toList()`.





Q.2.5 On cherche à définir des arbres binaires de recherche en réutilisant les fonctionnalités déjà écrites pour les arbres binaires et pour cela on ajoutera une méthode `insert` qui prend un élément du type des étiquettes d'un arbre et construit un nouvel arbre de deux manières différentes :

- par modification physique en ajoutant physiquement l'élément
- par copie en retournant un nouvel arbre contenant ce nouvel élément

Proposer une architecture correcte pour cette extension et implanter les deux méthodes d'insertion.

Exercice 3 : Evaluation de formules logiques (Visiteur)

On cherche à évaluer des formules logiques contenant des constantes, des variables, la négation, la conjonction et la disjonction. Les variables sont à évaluer dans un environnement fourni. Pour cela on utilisera un visiteur pour explorer les formules. On donne le canevas en Java.

```
interface IVisiteur<T> {
  T visite(Constante c);
  T visite(Not e);
  T visite(And e);
  T visite(Or e);
  T visite(Var v);
}
abstract class Formule {
  public abstract <T> T accepte(IVisiteur<T> v);
}
```

Q.3.1 Ecrire les classes concrètes `Cte` et `Var`, pour les constantes et les variables booléennes, sous-classes de `Formule`

Q.3.2 Même questions pour les classes `Not`, `And` et `Or`

Q.3.3 Construire les deux formules suivantes :

- (a OU b) ET ((NON a) OU (NON b))
- (a ET (NON b)) OU ((NON a) ET b)

Q.3.4 Ecrire un visiteur, appelé `VisiteurEval`, qui évalue en un booléen une formule sans variable. Une exception est déclenchée si une variable apparait dans la formule.

Q.3.5 Comment feriez un visiteur qui évalue une formule en tenant compte d'un environnement de calcul où toutes les variables de la formule sont valuées ? Implanter votre proposition.

Travaux sur Machines Encadrés

Exercice 4 : Introspection

On va chercher à voir et à manipuler les états internes des objets.

Q.4.1 En reprenant la classe `Lecture` suivante :

```
import java.lang.reflect.*;

public class Lecture {
  public static void main(String args[]) {
    Class c = null;
    Field[] champs = null;
    Method[] methodes = null;

    try {
      c = Class.forName(args[0]);
      champs = c.getDeclaredFields();
      methodes = c.getMethods();
    }
    catch (ClassNotFoundException e) { // ...;
       System.exit(0);
    }
    catch (SecurityException e) { // PB d'autorisation
       System.exit(0);
    }

    for (int i=0; i< champs.length;i++) {
      Field uc = champs[i];
      System.out.println("champs "+i+" : "+uc);
    }

    for (int i = 0; i < methodes.length; i++) {
      Method um = methodes[i];
      System.out.println("methodes "+i+ " : " + um);
    }
  }

}
```

afficher et commenter les informations pour les classes `Integer`, `ArrayList` et `Hashtable`.

Q.4.2 En Python tester sur une liste la fonction dir qui retourne l'ensemble des attributs et utiliser l'appel à la méthode __doc__ qui retourne la documentation. Faites de même sur la classe `point` du cours 3.

Q.4.3 En Java, construire une classe avec des méthodes possédant les différents qualifieurs (public, protected, **private**) puis en reprenant la fonction apply_f du cours 3 (planche 55), qui permet d'invoquer directement la méthode d'un objet, appeler ces différentes méthodes. Que pouvez-vous en dire ?

Exercice 5 : Composite et Visiteur pour Mini-interprète

En s'inspirant de l'exercice 3 on cherche à implanter l'exercice 6 du TME2 sur l'interprète d'un mini-langage impératif en utilisant des visiteurs : un pour les expressions et un pour les instructions. Il faudra adapter les classes `Expr` (respectivement `Stmt`) et ses classes descendantes pour qu'elles acceptent un visiteur pour les expression (respectivement pour les instructions). Vous pouvez répondre dans le langage objet de votre préférence en tenant compte des consignes de l'énoncé.

Q.5.1 Ecrire l'interface pour les visiteurs des expressions. Même question pour les visiteurs des instructions.

Q.5.2 Ecrire les classes pour les expressions et pour les instructions.

Q.5.3 Ecrire un visiteur pour l'évaluation des expressions. Pour les variables contenues dans une expression il sera nécessaire d'accéder à un environnement des déclarations.

Q.5.4 Ecrire un visiteur pour l'interprétation des `Stmt`.

Q.5.5 Ecrire une fonction principale qui construit un programme, affiche l'environnement de départ, évalue le programme et affiche l'environnement à la fin du programme si celui-ci termine.

Q.5.6 Tester l'ensemble avec un programme, dans le mini-langage impératif, qui calcule la somme des n premiers nombres entiers :

```
% calcul de la somme des n premiers nombres entiers
n = 5 ;
y = 1 ;
z = 1 ;
while ( (y == n) == false) { y = y + 1; z = z + y; }
```

Exercice 6 : Héritage multiple

On cherche à comprendre comment se passe le masquage de méthodes en héritage multiple.

Q.6.1 Dans un langage objet acceptant l'héritage multiple on définit quatre classes `A`, `B`, `C` et `D`, telles que `B` et `C` héritent de `A` et `D` hérite de `B` et `C`. Chaque classe contient une méthode qui retourne la chaîne de caractères du nom de la classe de construction de l'objet concaténée à l'appel des méthodes `m` des ancêtres directs. Ecrire ces 4 classes (Python ou OCaml).

Q.6.2 que donne les appels suivants :

```
a = A()
b = B()
c = C()
d = D()
a.m()
b.m()
c.m()
d.m()
```

Quelques Liens

- en Python : module inspect pour l'introspection
  - `https://docs.python.org/3.10/library/inspect.html`
- en OCaml :
  - chapitre 15 de DAOC sur les objets en OCaml : `https://www-apr.lip6.fr/~chaillou/Public/DA-OCAML/book-ora139.html`
- en Java :
  - Design Patterns comportementaux et structuraux, cours 6 et 8 de LU3IN002 : `http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2020/ue/LU3IN002-2020oct/`