Université Paris 7 — Java
Licence Informatique — Année 2005-2006

TD n°5 - Correction

Classification et Héritage

Exercice 1 On se donne la liste des classes suivantes : `Etudiant`, `Personne`, `EtudiantTravailleur`, `Enseignant`, `EtudiantSportif` et `Travailleur`.

1. dessinez une arborescence cohérente pour ces classes en la justifiant,
2. où se situeront les champs suivants : `salaire`, `emploiDuTemps`, `anneDEtude`, `nom` et `sportPratique`.

Exercice 2 Les exemples suivants sont ils corrects ? Justifiez.

1. ```
   class Ex1a {
       private int a;
       public Ex1a() { System.out.println(a); }
       public Ex1a(int a) { this.a = a; this(); }
   }
   ```
   Correction : Non : dans le constructeur à un paramètre l'appel au constructeur sans paramètre doit être effectué en premier.

2. ```
   class Ex1b {
       int a;
   }
   class Ex1b2 extends Ex1b {
       int b;
       Ex1b2(int a, int b) { this.a = a; this.b = b; }
   }
   ```
   Correction : Oui.

3. ```
   class Ex1c {
       int a;
       Ex1c(int a) { this.a = a; }
   }
   class Ex1c2 extends Ex1c {
       int b;
       Ex1c2(int a,int b) { this.a = a; this.b = b; }
   }
   ```
   Correction : Non : En l'absence d'appel explicite à un constructeur de la super classe dans le constructeur de la classe `Ex1c2`, un appel implicite au constructeur par défaut de la super classe est tenté ; or la définition d'un constructeur quelconque annihile l'existence du constructeur par défaut. La correction est donc d'appeler explicitement dans la sous-classe le constructeur de la super classe : `this(a);`.

4. ```
   class Ex1d {
       public void f() { System.out.println("Bonjour."); }
   }
   ```

```
class Ex1d2 extends Ex1d {
    private void f() { System.out.println("Bonjour les amis."); }
}
```

Correction : Non : on ne peut redéfinir une méthode en restreignant sa visibilité.

Exercice 3 Qu'affiche le programme suivant :

```
class A {
    int i;
    int f() { return i; }
    static char g() { return 'A'; }
    char h() { return g(); }
}
class B extends A {
    int i=2;
    int f() { return -i; }
    static char g() { return 'B'; }
    char h() { return g(); }
}
class C {
    public static void main(String [] argv) {
      B b = new B();
      System.out.println(b.i);
      System.out.println(b.f());
      System.out.println(b.g());
      System.out.println(b.h());
      A a = b;
      System.out.println(a.i);
      System.out.println(a.f());
      System.out.println(a.g());
      System.out.println(a.h());
    }
}
```

Correction :

```
2
-2
B
B
0
-2
A
B
```

La « surprise » est en ligne 5, où l'on s'attend à lire `2` (comportement des méthodes). En réalité il n'y a pas de dynamisme pour l'accès aux champs, le type de l'objet est déterminé à la compilation (ici `A`), donc c'est bien le champ de la « structure » `A` auquel on accède, alors même que l'objet est dynamiquement un `B`.

Exercice 4 Dans cet exercice, on veut écrire une classe `EnsembleDEntiers` fournissant des méthodes permettant d'ajouter ou enlever un entier de l'ensemble et d'afficher cet ensemble :

1. donnez la liste des prototypes des constructeurs et méthodes de la classe `EnsembleDEntiers`,
2. implémentez cette classe,
3. on veut maintenant étendre cette classe en `EnsembleOrdonneDEntiers` dans laquelle l'ensemble apparaîtra comme toujours ordonné (ses éléments seront visiblement ordonnés).

Correction :

```
import java.util.Random;

class EnsembleDEntiers {
    private int [] elements;
    private int nElements;
    private void retaille() {
if (nElements<elements.length) return;
int [] tmp = new int[elements.length+10];
for (int i=0; i<nElements; i++)
    tmp[i] = elements[i];
elements = tmp;
    }
    public EnsembleDEntiers() {
elements = new int[10];
nElements = 0;
    }
    public EnsembleDEntiers(EnsembleDEntiers e) {
this();
for (int i=0; i<e.nombreDElements(); i++) ajoute(e.element(i));
    }
    public void ajoute(int element) {
retaille();
elements[nElements++] = element;
    }
    private int cherche(int element) {
for (int i=0; i<nombreDElements(); i++)
    if (element(i)==element) return i;
return -1;
    }
    public int element(int indice) {
return elements[indice];
    }
    public void enleve(int element) {
int indice = cherche(element);
if (indice==-1) return;
for (int i=indice+1; i<nombreDElements(); i++)
    elements[i-1] = elements[i];
nElements--;
    }
    public int nombreDElements() {
return nElements;
    }
    public String toString() {
StringBuffer tmp = new StringBuffer("{");
if (nombreDElements()>0) {
    for (int i=0; i<nombreDElements()-1; i++)
tmp.append(element(i) + ",");
    tmp.append(element(nombreDElements()-1));
```

```
}
tmp.append("}");
return tmp.toString();
    }
    public static void main(String [] argv) {
EnsembleDEntiers e = new EnsembleOrdonneDEntiers();
Random r = new Random();
for (int i=0; i<23; i++) {
    e.ajoute(r.nextInt(100));
}
System.out.println(e);
    }
    protected void modifie(int indice,int valeur) {
elements[indice] = valeur;
    }
}

class EnsembleOrdonneDEntiers extends EnsembleDEntiers {
    private int cherchePremierPlusGrandOuEgal(int element) {
for (int i=0; i<nombreDElements()-1; i++)
    if (element(i)>=element) return i;
return nombreDElements()-1;
    }
    public void ajoute(int element) {
super.ajoute(element);
int ppg = cherchePremierPlusGrandOuEgal(element);
int valeur = element(nombreDElements()-1);
for (int i=nombreDElements()-1; i>ppg; i--)
    modifie(i,element(i-1));
modifie(ppg,valeur);
    }
}

class EnsembleFaineantOrdonneDEntiers extends EnsembleDEntiers {
    private boolean ordonne;
    public void ajoute(int element) {
ordonne = false;
super.ajoute(element);
    }
    public String toString() {
if (!ordonne) trie();
return super.toString();
    }
    public int element(int indice) {
if (!ordonne) trie();
return super.element(indice);
    }
    private void trie() {
if (ordonne) return;
// tri quelconque du tableau...
ordonne = true;
    }
}
```