Contrôle POO en Java 1A informatique
Juin 2016
Najib Tounsi

Durée : 45 mn. Barème : 1/3 1/3 1/3

Question 1 : Le fichier source Java qui suit, déclare deux classes, une classe `Mere` qui porte une méthode `g()` qui imprime un message, et une sous classe `Fille` qui porte la même méthode. Une classe Bidon ensuite qui contient qui contient deux méthodes `f(Mere)` et `f(Fille)`.

Réponse:

```
class Mere {
    void g() {
        System.out.println ("g() de Mere");
    }
}

class Fille extends Mere {
    void g() {
        System.out.println ("g() de Fille");
    }
}

class Bidon {
    public static void f(Mere m) {
        System.out.println (" f(Mere m) appelée");
        p.g();
    }
    public static void f(Fille p) {
        System.out.println (" f(Fille f) appelée");
        p.g();
    }
}

class Hello  {
    static public  void main(String args[]) {
        Bidon.f(new Mere());          // (a)
        Bidon.f(new Fille());         // (b)
        Mere m = new Fille();
        Bidon.f(m);    }              // (c)
}
```

1) Donner le résultat de trois appels `Bidon.f(...)` numérotés (a), (b) et (c)

(a)

```
 f(Mere m) appelée
g() de Mere
```

(b)

```
 f(Fille f) appelée
g() de Fille
```

(c)

```
 f(Mere m) appelée
g() de Fille
```

2) Y a-t-il une différence entre les appels à f () dans le cas (b) et (c) ? Laquelle ? Justifier.

> « Oui », dans un cas (b) c'est `void f(Fille p)` qui est appelé, dans l'autre c'est `void f(Mere m)`. Deux fonctions différentes, mais de même nom. Surcharge.
> « Non », dans les deux cas, f est appelée avec une instance de Fille.

3) Y a-t-il une différence entre les appels internes à g() dans le cas (a) et (c) ? Laquelle ? Justifier.

> Même remarque.
> « Non », Dans les deux cas cas (a) et (c), c'est la même fonction *g*() de `void f(Mere m)` qui est appelée. Une fois sur une instance de Mere, et une fois sur une instance de Fille.
> « Oui », La fonction *g*() étént polymorphe, en (a) elle s'exécute sur une instance Mere et une fois sur une instance Fille.

Question 2 :
Donner un exemple d'héritage multiple entre quelques classes de votre choix (*), et montrer comment réaliser cela en Java.
(*) Indication : donner uniquement les noms des classes, et dire à quels objets elles correspondent, par exemple classe *`Auto`* et *`Vehicule`* correspond à des engins en circulations etc. Donner aussi quelques caractéristiques qui justifient l'héritage, par exemple, une auto est un véhicule)

> Indication de réponse :
>
> 1) Il faut au moins trois unités :
> Une interface I, une classe C et une sous classe B de C
> Donc :
> class B *extends* C *implements* I {...}
>
> 2) donner aussi au moins une opération de A, dont on hérite, et une opération de I qu'on implémente dans B.
>
> Si l'exemple est réel, c'est encore mieux. Comme dans le cours :
>
> `class Vetement extends Article imlements Transport.`
>
> `Vetement` hérite des caractéristiques de `Article` et de la méthode `prixTransport` de `Transport.`

Question 3 : Soit le TAD Vecteur de Réels (de dimension M au maximum)

```
TAD Vecteur
début
   init :                        → Vecteur

   put  : ℝ × ℕ × Vecteur        → Vecteur
        // mettre un réel à un rang donné dans le vecteur

   get  : ℕ × Vecteur            → ℝ
        // retourne le réel qui se trouve au rang donné
fin
```

a) Réaliser ce TAD en Java.

```
class Vecteur{
     int M = 2 ;
     double t[] ;

    public void init (){
         // on peut l'appeler Vecteur pour le constructeur
     t = new double[M] ;
    }

    public put(int i, double x){
     t[i] = x ;            // on peut vérifier les bornes...
    }

    public get (int i){
     return t[i] ;         // on peut vérifier les bornes...
    }
}
```

NB : une réalisation par une Interface est **juste aussi**, à condition de définir une classe qui l'implémente et qui définit les méthodes comme ici. C'est ce que j'ai fait en classe pour les TAD.

b) On voudrait définir l'opération produit scalaire de deux vecteurs. Donner une méthode pour cela.

```
public double produitScalaire (Vecteur v) {
     // calcul avec this.t[i] et v.t[i] ...
}

ou bien

static double produitScalaire (Vecteur v1, Vecteur v2) {
     // calcul avec v1.t[i] et v2.t[i] ...
}
```

c) Donner le code Java pour

1. créer deux vecteurs r et s à deux composants,
2. les initialiser avec resp. (1,2) et (3,4),
3. et faire leur produit scalaire

```
Vecteur r = new Vecteur() ;
Vecteur s = new Vecteur() ;
r.put (0,1) ;
r.put (1,2) ;
s.put (0,3) ;
s.put (1,4) ;

double x = r.produitScalaire(s) ;
ou bien
double x = produitScalaire(r, s) ;
```

Commentaires :
1) On peut aussi déclarer une classe avec la méthode `main` etc pour inclure ce code.
2) Juger surtout l'usage de put pour initialiser `r` et `s`, et l'application exact du profile de `produitScalaire` dans l'appel.