Nom prénom

Examen Programmation Par les Objets en Java
Juin 2108, 1A Informatique.

D. Ghanami et *N. Tounsi*

Durée : 1h 15.

Question 1)
Que signifie un clonage d'objets et à quoi cela sert-il ?

Le clonage et une façon de dupliquer ou créer une copie exacte des objets en Java. Etc…
La méthode clone () de Object est utilisée pour cela.

Cf. `http://www.emi.ma/ntounsi/COURS/Java/PooJavaPart-3.html#[54]`

Combien de type de clonage peut-on considérer. Pourquoi en faut-il autant ?

- **Il y le « clonage » (ou plutôt la simple affectation) qui est la copie des références. Deux références qui désignent le même objet.**
- **Mais il y a le clonage superficiel, copie bit à bit des champs de l'objet.**
- **Clonage en profondeur, duplication total de toute la structure de l'objet.**

Question 2)
Soit la classe `Article` vue en cours/TP. On voudrait en dériver un certain nombre de sous-classes dont une classe TV qui redéfinit la *prixTTC*() d'un article avec une TVA de 21% au lieu de 12% normalement.

a) Donner la définition d'une telle classe TV.

```
class TV extends Article{

    public double prixTTC() { return prixHT * 1.21;}

    public TV(){ super() ;}  // facultatif
}
```

Par ailleurs, on constate pour plusieurs sous classes de la classe `Article`, le besoin de méthodes pour calculer (1) le prix de transport (50Dh en plus du prix), et (2) pour certaines classes d'articles fragiles, un prix de transport particulier (100Dh en plus).

b) Comment pourrait-on décrire en une seule fois le profile de deux méthodes, et partager cette description par toutes les sous classes qui ont besoin de définir de telles méthodes.

On peut définir

- une interface *Transport* avec une méthode *prixTranport*() — +50Dh— est une interface *Fragile* qui en hérite et redéfinit la méthode,

```
interface Transport {
    public double prixTransport();
    //...
}

interface Fragile extends Transport {
    public double prixTransport();
    //...
}
```

- deux interface *Transport* et *TransportFragile*, avec chacune sa méthode *prixTransport*().

```
interface Transport {
    public double prixTransport();
    //...
}
interface TransportFragile {
    public double prixTransport();
    //...
}
```

c) Comment modifier la classe TV précédente, sachant qu'un téléviseur est un article à transporter par le marchand et en plus il est fragile.

Dans un cas :

```
class TV extends Article implements Fragile{
     // comme avant
    public double prixTransport() {
          return (prixHT * 1.21) + 100;
     }
}
```

Dans l'autre

```
class TV extends Article implements TransportFragile{
     // comme avant
    public double prixTransport() {
          return (prixHT * 1.21) + 100;
     }
}
```

Exercices comptant pour le TP

Exercice I)

1) Définir une interface `Loisir` qui contient une constante chaîne `s` de valeur "`Sortir au cinéma ou au restaurant`" et une méthode `sortir` sans paramètres.

```
interface Loisir {
    public String s = "Sortir au cinéma ou restaurant";
    public void sortir();
}
```

2) Définir une classe `Cinema` qui implémente cette interface et dont la méthode affiche simplement le message " `Je vais au cinéma` " précédé de la chaîne `s`.

```
class Cinema implements Loisir {
  //Implémentation de la méthode courirOuMarcher
  public void sortir(){
     System.out.println("J'aime " + s);
    }
};
```

3) Ecrire un programme *main* qui crée une instance de cette classe et en appelle la méthode.

```
class Test{
    static public void main(String args[]){
        Cinema c = new Cinema() ;
        c.sortir();
    }
}
```

4) Peut-on déclarer :

a) `Loisir l = new Cinema() ;`
b) `Cinema c = new Cinema() ;`
c) `Loisir ll = new Loisir() ;`

Justifier.

a) Oui, Cinema est une instance de Loisir
b) Oui, évidemment
c) NON, on n'instancie pas une interface

5) Rajouter à la classe `Cinema` une méthode `prefere` () qui affiche simplement le mot " `Western` ".

On peut ne pas reprendre toute la classe Cinema

```
class Cinema implements Loisir {
  //Implémentation de la méthode courirOuMarcher
  public void sortir(){
     System.out.println("J'aime " + s);
    }

  public void prefere(){
     System.out.println("Western");
     }

};
```

6) Peut-on écrire (d'après les déclarations ci-dessus) :

d) `l.prefere() ;`

ou

e) `c.prefere();`

Pourquoi ?

d) Non, l est de type `Loisir` n'a pas la méthode `prefere`.
e) Oui. Bien sûr.

7) Définir une classe `Restaurant` qui implémente aussi cette interface et dont la méthode affiche cette fois-ci le message " `Je vais au restaurant`".

… même chose que la classe Cinema de la question 2, avec le message qui change

8) Soient trois variables `c`, `r`, `l` déclarée de type `Cinema`, `Restaurant et Loisir` respectivement. Y-a-t-il un problème à faire :

f) `l = c ;`
g) `c = l ;`
h) `r = c ;`
i) `l = r ;`

Pourquoi ?

f) non, il n'y a pas de problème, `c` est une instance de `l`
g) OUI il y a problème, types incompatibles !
h) idem
i) non, comme f ci-dessus

Exercice II)

Soit notre classe Pile :

```
public class Pile {
    char t[];
    int top;
    public Pile(int n) {
        t = new char[n]; top = -1;
    }

    public void empiler(char c) {
            t[++top] = c;
    }

    public char sommet() {
            return t[top];
    }

    public void depiler() {
             top--;
    }

    public boolean estVide() {
        return (top < 0);
    }

    public boolean estPleine() {
        return (top >= (t.length - 1));
    }
};
```

1) Ecrire la séquence d'instructions pour déclarer un objet `Pile`, l'initialiser et empiler les caractères d'une chaîne initialisée « alibaba ».

Devrait être

```
String s = "alibaba"
Pile p = new Pile(s.length()) ; // ou Pile(7)
for (int i = 0 ; i<s.length() ; i++)
     p.empiler (s.charAt(i)) ;
```

ou bien avec

```
for (char c : s){ p.empiler (c) ;}
```

2) Quelles sont les cas d'erreurs possibles que cette programmation de la classe `Pile` peut engendrer ?

Les cas d'erreurs sont évidemment
- empiler sur une pile pleine
- chercher le sommet de la pile ou dépiler sur une pile vide

3) Quelles sont les méthodes concernées dans ce cas ?

- **Empiler (sur pile pleine) , dépiler et sommet (sur pile vide)**

4) On a les déclarations suivantes :

```
class enHaut extends Throwable {};
class enBas extends Throwable {};
```

A quoi peut servir `extends Throwable` .

C'est pour déclarer des classes d'exception débordement par le bas ou le haut, à utiliser avec throw etc. etc.

Comment les utiliser ces deux classes pour gérer les cas particulier d'erreurs de la question II-2) et II-1) précédentes ?

Il faut refaire les méthodes concernées, par exemple :

```
public char sommet() throws enBas {
        if (!estVide())
            return t[top];
        else
            throw new enBas();
    }
```

et ensuite

```
    try {p.empiler (c) } catch (enBas e) {…}
```

etc…

NB. Pour tout cet exercice

Cf. TP2 et `http://www.emi.ma/ntounsi/COURS/Java/TD/PileException.java`