2019

Corrigé de l'examen de la matière P.O.O- 2ème année Licence - socle commun

Réponses aux questions (02 points)

1) Quelle différence existe t-il entre les interfaces List et Set ?

Contrairement à set, List permet les doublons.

2) Citer les méthodes de l'interface Iterator.

next(), hasNext(), remove()

3) Pourquoi ne peut-on pas invoquer directement la méthode iterator() (elle n'existe pas) sur un objet de type Map < clé , valeur >.

Parce que Map n'implémente pas l'interface Iterable.

4) Pour parcourir un objet de la classe hashmap, on peut utiliser des méthodes qui nous renvoie des objets qui implémentent l'interface Iterable, qu'on pourra itérer via un objet de type Iterator, citer deux méthodes en indiquant ce qu'elles renvoient ainsi que leur contenus.

1- la méthode keySet(). Elle renvoie un objet de type Set, c'est un ensemble de clés.
2-La méthode entryset(). Elle renvoie un objet de type Set, c'est un ensemble d'éléments de type Map.Entry<K,V> -un ensemble de couples (clé, valeur)-.

Exercice1 (04 points)

Pour les lignes 1;2;3 (mentionnées en commentaires dans le programme principal) vous dites si c'est juste en précisant ce qui sera affiché, en cas d'erreurs vous donnez la cause.

```
class mere {
mere (int i) { this(i,i); System.out.println(i); }
mere (String ch) { System.out.println(ch); }
mere (int i, int j) { this("i+j"); System.out.println(i+j); }
}
```

```
class fille extends mere{
fille () { super(22); System.out.println(("vide"); }
fille (int i) { this(); System.out.println(i); }
fille (String ch) { System.out.println(ch); }
fille (String ch, int i) { System.out.println(ch); super(i);}    }
```

```
public class  MAIN {
        public static void main(String [] args){
/*1*/    fille f = new fille(22);
/*2*/    fille ff = new fille("22");
/*3*/    fille fff = new fille ("22",22); }      }
```

Solution

/*1*/ i+j
44
22
vide
22

/*2*/ erreur, en appelant le constructeur de la classe fille qui a pour paramètre une chaine de caractères, java va insérer un super() comme une première instruction, or, dans la classe mère il n'existe pas un constructeur sans paramètres ➔ erreur lors de la compilation.

/*3*/ Il existe deux erreurs :
-la première est similaire à celle de /*2*/

- la deuxième est : le super(i), qui apparait comme une deuxième instruction, or, on sait bien qu'un appel à un constructeur de la classe mère- super(..)- ou à un constructeur de la même class -this(...)- doivent toujours figurer comme une première instruction.

Exercice2 (04 points)

```
public class Article {
        int code;
        String nom;
        double prixHT;
       Article(String nom, double prixHT) {      // a compléter          }
       double prixTransport() {  return prixHT * 0.05;  // 5% du Prix Hors Taxes de l'article } }
```

1. Ecrire le constructeur de la classe article. Le code doit être unique pour chaque article(vous pouvez ajouter une variable-attribut-).

On ajoute une classe des articles Fragiles : on va l'appeler Fragile :

2. Donner deux constructeurs de la classe Fragile à partir (ils font appel) du constructeur de sa classe de base, l'un va préciser l'emballage (de type String) l'autre non.
3. Le prix de transport d'un article fragile est deux fois le prix de transport normal d'un Article. Redéfinir la méthode prixTransport() de la classe Fragile.
4. Peut-on écrire : Article tv = new Fragile ("LG", 7000.00);
   Si oui, expliquer pourquoi et quel est le prix de transport de l'article tv ?

Maintenant on ajoute la classe Magasin (un ensemble d'articles) :

Equiper cette classe avec une structure de données du framework Collection qui va contenir l'ensemble des articles.

5. Donner la définition de la classe en JAVA. (la structure de données qui va contenir les articles et le constructeur de cette classe).
6. La méthode add() permet d'ajouter un article et la méthode contains() vérifie si un article est disponible au magasin. Donner le plus simplement possible le code de ces deux méthodes (pour la méthode contains(), vous supposez qu'il existe une méthode getcode() dans la classe article qui renvoie le code d'un article).

Interface Exporter :

Supposons que **Exporter** est une interface qui nous permet de rendre nos articles exportables à l'étranger et qui contient deux méthodes pour :

- spécifier le pays de destination.
- régler les droits de douane.

7. Expliquer en quelques mots comment rendre nos articles exportables à l'étranger (sans code)?

Solution

1) Le constructeur de la classe article

```
public class Article {
 int code;
 String nom;
 double prixHT;

 static int c;

 Article(String nom, double prixHT) {
        this.code = c++;
        this.nom = nom;
        this.prixHT = prixHT;
    } ...}
```

2) **Les deux constructeurs de la classe Fragile sont :**

```
Fragile(String nom, double prixHT) {
        super(nom, prixHT);
    }

Fragile(String emballage, String nom, double prixHT) {
        super(nom, prixHT);
        this.emballage = emballage;
    }
```

3) **Redéfinition de la méthode prixTransport() de la classe Fragile**

```
public double prixTransport() {
        return 2 * super.prixTransport();
}
```

4) **Oui on peut écrire** Article tv = new Fragile ("LG", 7000.00); **C'est un upcasting !**
le prix de transport de l'article tv est : **2* 0.05* 7000.00 = 0.1*7000.00=700.00**

5) **la structure de données qui va contenir les articles et le constructeur de cette classe Magasin**

On va utiliser une structure de donnée a partir du framework Collection, par exemple une liste dont les éléments sont des objets de la classe Article.
On va doter la classe magasin d'un constructeur qui crée la liste –pour pouvoir la remplir ultérieurement- en utilisant une liste de type ArrayList ou une LinkedList.

```
public class Magasin {

    private List<Article> articles;

    public Magasin() {
        articles = new ArrayList<>();
        /*ou bien
        articles = new ArrayList<>();*/
    }
```

6) **La methode add() qui permet d'ajouter un article**

```
public void Add(Article a) {
        articles.add(a);
    }
```

la méthode contains() vérifie si un article est disponible au magasin

```
public boolean contains(Article a) {
        for (Article art : articles) {
            if (art.getCode() == a.getCode()) {
                return true;
            }
        }
        return false;
    }
```

7) **Pour rendre nos articles exportables vers l'étranger, la classe article doit implémenter l'interface exporter et définir toutes ses méthodes.**

Exercice3 (05 points)

Indiquez pour chaque ligne s'il y a des fautes. En cas d'erreur vous indiquez, s'elle est signalée pendant la compilation ou bien l'exécution (indiquez par exemple si c'est un downcasting implicite ou explicite ; upcasting).

```
interface Int {...}
class pere implements I {...}
class fils extends pere{...}
public class poly {
        public static void main(String [] args){
            /*1*/  fils f1= new fils(); pere p1=f1; fils f2=p1;
            /*2*/  Int p2= new fils() ; pere p3=p2;
            /*3*/  Int g = new pere(); pere w=(pere)g;
            /*4*/  pere h = new fils(); fils j = new fils(); j=(fils)h;
            /*5*/  Int m=new fils(); Object o = m; pere v= (fils)o; pere z= (pere)o;        }}
```

Solution

/*1*/ fils f1= new fils(); **une simple affectation**
pere p1=f1 ; **upcasting**
fils f2=p1; **downcasting implicite → erreur lors de la compilation.**

/*2*/ Int p2= new fils() ; **upcasting** /* on peut utiliser une interface comme un type d'objet dont la classe implémente cette inteface*/
pere p3=p2; **downcasting implicte → erreur lors de la compilation.**

/*3*/ Int g = new pere(); **upcasting**
pere w=(pere)g; **downcasting explicite correct, l'objet g , a déjà subi un upcasting et il pointe réellement sur un objet de la classe pere → un cast (downcasting explicite) correct.**

/*4*/ pere h = new fils(); **upcasting**
fils j = new fils(); **une simple affectation**
j=(fils)h; **downcasting explicite correct, l'objet h. a déjà subi un upcasting et il pointe réellement sur un objet de la classe fils → un cast (downcasting explicite) correct.**

/*5*/ Int m=new fils(); **upcasting**
Object o = m; **upcasting**
pere v= (fils)o; **upcasting + downcasting explicte de l'objet o, et qui est correct, parce que, l'objet o pointe sur un objet de la classe fils.**
pere z= (pere)o; **downcasting explicite correct. L'objet o pointe sur un objet de la classe fils, et tout objet de cette classe est aussi un objet de la classe pere, donc on peut le caster vers un objet de la classe pere.**

Exercice4 (05 points)

Pour chacune des lignes : 1;2;3;4;5;6;7;8 (mentionnées en commentaires dans le programme principal), vous dites ce qui va être affiché avec de brèves explications.

```
class peree {
  void A() {  System.out.println("A de mere");  B(this);  }
  void B(peree p) {  System.out.println("B de mere"); }
  void C() { System.out.println(" C de mere"); this.A(); }
  static void stat() { System.out.println("static  de mere"); }
 }
```

```
class filss extends peree {
  void A() {  System.out.println("A de fille"); super.stat();  }
  void B(peree p) { System.out.println("B de fille"); p.stat(); ((peree) p).A() ;
}
  void C() { System.out.println("C de fille");   super.C();  }
  static void stat() { System.out.println("static  de fille");}    }
```

```
public class Main {
 public static void main(String [] args) {
          filss fil = new filss();
          peree  per = new peree() ;
/*1*/     per.B(per);
/*2*/     per.stat();
          per=fil;
/*3*/     per.B(per);
/*4*/     per.stat();
/*5*/     ((filss) per).stat();
/*6*/     fil.C() ;
/*7*/     ((filss) per).A() ;
/*8*/     (new filss()).B(fil);}}
```

Solution

----/*1*/-----
B de mere
----/*2*/-----
static de mere
----/*3*/-----
B de fille
static de mere
A de fille
static de mere
----/*4*/-----
static de mere
----/*5*/-----
static de fille
----/*6*/-----
C de fille
 C de mere
A de fille
static de mere
----/*7*/-----
A de fille
static de mere
----/*8*/-----
B de fille
static de mere
A de fille
static de mere