Examen POO Java Juin 2015 (Corrigé)

1ère Année Informatique

NB. Sont donnés ici les éléments de réponses aux questions (ce qui est demandé) et des explications supplémentaires utiles.

On se propose de faire un programme pour gérer une collection d'entiers.

Question 1. On voudrait définir les fonctions de manipulation suivantes:

init :		→	*Collection*	/* initie une collection à vide */
ajouter :	*Collection* X *Entier*	→	*Collection*	/* ajoute un entier à la collection*/
supprimer :	*Collection* X *Entier*	→	*Collection*	/* supprime un entier de la collection*/
appartient :	*Collection* X *Entier*	→	*Booléen*	/* teste si un entier est dans la collection*/
card :	*Collection*	→	*Entier*	/* rend le nombre d'éléments de la collection*/
vide :	*Collection*	→	*Booléen*	/* teste si la collection est vide*/
pleine :	*Collection*	→	*Booléen*	/* teste si la collection est pleine*/

1. Définir une classe Java pour cette structure de donnée, sans programmer les méthodes pour l'instant.

```
class Collection {
    final int MAX = 20;
    int []  t;
    int card;

    public init(){}
    public void ajouter (int x) {}
    public void supprimer (int x) {}
    public boolean appartient (int x) {}
    public int card () {}
    public boolean vide () {}
    public boolean pleine () {}
}
```

   La classe ne compile pas encore. Il manque au moins les instructions `return` dans les méthodes à résultat.

2. Comment faire de la méthode *init* un constructeur?

   remplacer `public void init() { card=0;...}`
   par `public Collection() { card=0;...}` Sans `void`.

3.
4. Ecrire un programme de test qui ajoute dans une collection de 5 nombres entiers donnés (lus par une fonction `lire()` à valeur entière existant dans une classe `maClasse` dans le package `lang`).

   Un exemple de programme test

```
class Test {
    public static void main (String arg[]){
        Collection c = new Collection();
        int x;
        for (int i = 0; i<5; i++){
            x = maClasse.lire();
            c.ajouter(x);
        }
    }
}
```

qui utilise `maClasse.lire();` supposée existante.

---

5. Rajouter les instructions pour tester pour un entier à ajouter n'est pas déjà dans la collection.

Remplacer la ligne `c.ajouter(x);`
par

```
if (!c.appartient(x)) c.ajouter(x);
```

---

6. Programmer les méthodes de cette classe collection.

```
    public Collection() {
        card=0;
        t = new int [20];
    }

    public void ajouter (int x) {
            // La collection peut contenir plusieurs fois le même élément
            // Pour l'instant on ne vérifie pas le débordement.
        t[card++]=x;
    }

    public void supprimer (int x) {
        for (int i =0; i<card; i++){
            if (t[i] == x) {
                t[i] = t[--card];
                break;
            }
        }
    }

    public boolean appartient (int x) {
        for (int i =0; i<card; i++){
            if (t[i] == x) return true;
        }
        return false;
    }

    public int card () {return card;}
    public boolean vide () {return (card==0);}
    public boolean pleine () {return (card== MAX);}
```

---

7. Quels sont les cas d'erreur à prévoir dans les méthodes de classe *Collection*? Les programmer.

Le seul cas d'erreur à prévoir c'est l'ajout dans une collection pleine. On peut le programmer ainsi

```
if(!pleine()){
     t[card++]=x;
}
else System.out.println("Collection pleine.");
```

Ou, mieux, avec les exceptions .

```
public void ajouter (int x) throws CollectionPleine {
       if(!pleine()){
           t[card++]=x;
           System.out.println ("+   "+x);
       }
       else throw new CollectionPleine();
   }
```

Avec: `class CollectionPleine extends Throwable {} ;`

---

8. Donner un exemple d'utilisation dans votre programme de la question 3.

```
try {
      c.ajouter(x);
} catch (CollectionPleine e) {
      System.out.println("Collection Pleine");
}
```

Question 2: Maintenant, soit la classe:

```
class Element {
      int x;
      public  Element(int i) { x = i;}
      public void set (int i) { x = i;}
      public int get () { return x;}
      public boolean equals (Element e) { return e.x == this.x; }
}
```

1. Comment peut-on changer la classe Collection pour avoir une collection d'objets `Element`. (Décrire brièvement ce changement sans le programmer).

On changera `int` *en* `Element` *là où il faut. A la déclaration du tableau t*

```
Element []  t;
```

et au constructeur

```
t = new Element [MAX];
```

et dans les paramètres des méthodes concernées

`ajouter (Element x)`, `supprimer (Element x)`, `appartient (Element x)` .

et on ***veillera*** *à ce que dans cette dernière méthode,* ***on remplace***

```
if (t[i] == x)
```

par

```
if (t[i].equals (x))
```

car on compare des objets quelconques, donc il faut utiliser leur méthode equals().

---

2. Que faut-il changer dans le programme 3. précédent?

Remplacer `int x` *par* `Element x = new Element(maClasse.lire());`
Ou bien définir une méthode `lireElement()` *dans la classe* `Element` *et l'utiliser ici.*

---

3. Décrire brièvement comment peut-on généraliser pour avoir des collections de réels, de caractères, de booléen etc.

Tant qu'il s'agit de types primitifs, dans la classe `Element`, *il suffit de changer* `int` *en* `float` *par exemple, pour avoir une collection de* `float`.

---

4. Peut-on généraliser pour avoir des collections d'autres objets non primitifs?

En principe oui. Comme précédemment. Mais il faut faire attention au sens de `{ x = i;}` *dans la méthode set() et au sens de* `{ e.x == this.x }` *dans equals(). Utiliser éventuellement les opérations "d'affectation" et de "test d'égalité" de l'objet concerné .*
