2I002 - Introduction à la programmation Objet
Groupe 2 – Test n°2 – Correction

Thomas Robert Remi Cadene

4 mai 2018 - Durée : 40 minutes

Partie 1 – Egalité, copie, static

1.1 Questions de cours

1. **Quelle est la signature standard de la méthode d'égalité pour une classe nommée `A` ?**

```
public boolean equals(Object o)
```

2. **Quelles sont les deux techniques pour réaliser la copie d'un objet de classe `A`, et les signatures associées ?**

Méthode de copie `public A clone()` ou constructeur de copie `public A(A a)`.

3. **Décrivez en quelques lignes le comportement des variables et méthodes déclarées `static` par rapport à leur équivalent sans ce modificateur.**

Les attributs statiques sont liés à la classe et non aux instances, on les appelle d'ailleurs variables et méthode de classe. Ainsi, ils ne dépendent pas d'une instance, ils existent avant la création d'une instance, n'ont pas de notion de `this`, ils sont communs à toutes les instances de la classe, la modif d'une variable statique par une instance affecte la prochaine lecture de cette variable par n'importe quelle instance. Les méthodes statiques ne peuvent avoir accès qu'aux attributs et méthode statiques et aux instances qui leurs sont passées en paramètre.

4. **On veut déclarer dans une classe quelconque une constante décrivant une taille maximale (de tableau par exemple) à laquelle on affectera la valeur `20`, quelle est la ligne de code correspondante ? Décrivez le rôle de chaque mot clé utilisé et sa pertinence dans la déclaration d'une constante.**

`public static final int TAILLE_MAX = 20` `public` car une constante est généralement lisible par les autres car étant statique cela ne présente pas de risque de l'exposer sans encapsulation, `static` car une constante n'a pas de raison d'être liée à une instance et a toutes les raisons d'exister et être accessible via la classe plutot que via les instances, `final` pour que sa valeur ne soit pas modifiable, `TAILLE_MAX` est le nom en majuscules pour respecter les conventions de nommage.

1.2 Code

5. Ecrire une classe `Copie` qui possède :

— **un identifiant** `id` sous forme d'un nombre entier. Cet identifiant lui est assigné **automatiquement** lors de l'instanciation (la première copie a le numéro 1, puis 2, 3, etc.).
Il permet d'identifier une copie de manière unique, mais plusieurs instances peuvent avoir le même `id`, si on a fait une copie d'une instance.
Note : Ce n'est pas grave si on "saute" certains numéros.
— **une note** `note` sous forme de nombre réel (assignée par la suite)
— un **constructeur sans paramètre**
— une méthode `noter(n)` qui permet de donner une note à la copie. Cette méthode ne met à jour la note de la copie que si elle n'a pas encore de note

— une méthode `estNotee()` qui indique si la copie a été notée ou pas
— un *getter* pour obtenir la note de la copie
— une façon d'**obtenir un "double" de la copie** (donc une copie de la copie...) *(cf Q 2.)*
— une méthode pour **tester l'égalité** entre deux instances de `Copie` (elles sont égales si elles ont le même `id`) *(cf Q 1.)*

Note : vous pouvez "compresser" votre code pour les méthodes courtes en ne mettant pas de retour à la ligne, ex `getX() { return x; }`

Note 2 : le code de cette partie est réutilisé dans la partie suivante. Si vous ne savez pas comment faire quelque chose, écrivez au moins la signature sans le contenu.

```
public class Copie {
    private static int cpt = 1;
    private int id;
    private float note;
    private boolean noteSet;

    public Copie () {
        id = cpt++;
        noteSet = false;
    }

    public Copie(Copie c) {
        id = c.id;
        note = c.note;
        noteSet = c.noteSet;
    }

    public void noter (float n) {
        if (noteSet) return;
        note = n;
        noteSet = true;
    }

    public boolean estNotee () { return noteSet; }
    public float getNote () { return note; }

    public boolean equals (Object o) {
        if (o == null || getClass() != o.getClass())
            return false;
        return ((Copie) o).id == this.id;
    }
}
```

Partie 2 – Tableaux

2.1 Questions de cours

On dispose de la classe `A` ci-dessous, qui ne fait que stocker la valeur passée au constructeur :

```
public class A {
    private int x;
    public A (int x) { this.x = x; }
    public getX() { return x; }
}

[...]

// dans un main
/* type a compléter */ tab1 = /* création du tableau à compléter */;
/* type a compléter */ tab2 = /* création du tableau à compléter */;
```

6. **Donner la version complétée du code ci-dessus pour que `tab1` soit un tableau de 3 `int`, et que `tab2` soit un tableau de 3 `A`.**

```
int[] tab1 = new int[3];
A[]   tab2 = new A[3];
```

7. **Faire une représentation de l'état de la mémoire après l'exécution de ce code.**

```
tab1 -> [0, 0, 0]
tab2 -> [-> null, -> null, -> null]
```

8. **Quelles lignes de code faut-il rajouter ou modifier pour que les tableaux contiennent les valeurs 1, 2 et 3 ?**

```
int[] tab1 = new int[]{1,2,3};
A[]   tab2 = new A[]{new A(1), new A(2), new A(3)};
// ou
for (int i = 0; i < 3; i++) { tab1[i] = i; tab2[i] = new A(i); }
// ou
tab1[0] = 1; tab1[1] = 2; tab1[2] = 3;
tab2[0] = new A(1); tab2[1] = new A(2); tab2[2] = new A(3);
```

2.2 Code

9. Ecrire une classe `Examen` qui consiste en un ensemble de copies. Cette classe possède :

— une **constante** indiquant le nombre de copies maximum que l'on peut stocker, fixée à `30`
— un ensemble de **copies stockées dans une liste**
— un **constructeur sans paramètre**
— une méthode `ramasser(copie)` pour ajouter une copie à l'examen
— une méthode `moyenne()` qui renvoie la moyenne des copies de l'examen (les copies non notées ne sont pas comptées dans la moyenne)
— une façon de **copier un examen**

— une méthode pour **tester l'égalité** de deux examens : deux examens sont égaux s'ils contiennent des copies égales aux mêmes positions dans le tableau de copies

```
public class Examen {
    private Copie[] copies;
    private int nbCopies;
    public static final int NB_COPIES_MAX = 20;
    public Examen() {
        copies = new Copie[NB_COPIES_MAX];
        nbCopies = 0;
    }

    public Examen(Examen e) {
        this();
        for (int i = 0; i < e.copies.length; i++)
            if (e.copies[i] != null)
                copies[i] = new Copie(e.copies[i]);
        nbCopies = e.nbCopies;
    }

    public void ramasser (Copie c) {
        copies[nbCopies++] = c;
    }

    public float moyenne () {
        float total = 0;
        int nbCopiesNotees = 0;
        for (Copie c: copies) {
            if (c != null && c.estNotee()) {
                total += c.getNote();
                nbCopiesNotees++;
            }
        }
        return total / nbCopiesNotees;
    }

    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass())
            return false;
        Examen e = (Examen) o;

        for (int i = 0; i < copies.length; i++) {
            if (copies[i] == null) {
                if (e.copies[i] != null)
                    return false;
                else continue;
            }

            if (!copies[i].equals(e.copies[i]))
                return false;
        }

        return true;
```

```
    }
}
```