

Titre :
(LgP6Linf.EPS)
Auteur :
Adobe Illustrator(TM) 3.2
Aperçu :
Cette image EPS n'a pas été enregistrée
avec un aperçu intégré.
Commentaires :

Contrôle continu écrit de novembre texte et corrigé

Durée : **1h45** - Tous les documents *personnels*, manuscrits ou imprimés, sont autorisés.
Tous les appareils électroniques sont interdits et doivent être rangés.

NE RIEN ÉCRIRE AU CRAYON NI À L'ENCRE ROUGE S.V.P.

Ce sujet comporte 4 pages.

A étude d'un programme

On considère les deux classes suivantes :

```
1 class eleve {
2     private String nom;
3     private int niveau;
4     public int nbEleve;
5
6     // constructeurs
7     eleve(String n, int niv) {
8         super();
9         nom= n; niveau= niv;
10        nbEleve++;
11    }
12    eleve() { this("???", 0); }
13    String eleve(int niv) {
14        niveau= niv;
15        return nom;
16    }
17
18    // encapsulation
19    public String getNom() { return nom; }
20    public void SetNiveau(int niv) { niveau= niv; }
21
22    // redefinition de toString
23    String toString() {
24        return "Eleve "+nbEleve+" "+nom+" niveau "+niveau;
25    }
26 }

1 class Test {
2     public static void main(String a) {
3         Eleve e1= new Eleve("Paul",5);
4         System.out.println(e1.toString());
5         Eleve e2,e3=e1;
6         e1= new Eleve();
7         e2= new Eleve("Marie",2);
8         System.out.println(e1.toString()+"\n"+e2.toString()+"\n"+e3);
9     }
10 }
```

La classe `eleve` compile sans aucun message d'erreur. Elle présente pourtant plusieurs problèmes que nous résoudrons progressivement en cheminant parmi les questions qui suivent.

a1 ligne 1

Une convention d'écriture de Java n'a pas été respectée. Donner la nouvelle ligne. En conséquence, quelles autres lignes faudra-t-il modifier dans la suite du programme ?

Par convention, le nom d'une classe débute par une majuscule, on doit donc avoir class Eleve. Il faut modifier eleve dans les lignes 7 et 12 qui sont effectivement des constructeurs et éventuellement dans la ligne 13 si on considère que c'est un nouveau constructeur (comme le suggère le commentaire au dessus).

a2 lignes 2 à 4

On a affaire à des définitions de variables de classe ou d'objets (attributs de classe ou d'instance). Pour chacune d'elles indiquez de quelle catégorie il s'agit ? Justifiez votre réponse.

nom, niveau et nbEleve ont été syntaxiquement déclarées comme variables d'instance, c'est-à-dire que chaque nouvelle instance de la classe créée aura un attribut nom, un attribut niveau et un attribut nbEleve, mais on peut d'ores et déjà remarquer que nbEleve serait plus logiquement variable de classe

a3 ligne 8

Quel est le principe fondamental qui permet d'écrire cette instruction (totalement valide).

Toute classe dérive par défaut de la classe Object, super() invoque donc le constructeur sans paramètre d'Object

a4 ligne 10

Quel est l'effet de cette instruction ?

Ajouter 1 à la variable nbEleve, qu'elle soit variable de classe ou variable d'instance. Manifestement, on se sert de cette variable pour compter le nombre d'élèves créés, et il faudrait donc que ce soit une variable de classe (ce qui se note en rajoutant static dans sa déclaration).

a5 ligne 12

Que fait ce constructeur ?

C'est le constructeur par défaut, il appelle le constructeur Eleve à deux paramètres avec les valeurs ??? pour nom et 0 pour niveau.

a6 lignes 13 à 16

Quel est l'objectif de ces quatre lignes ? Rectifiez ce qui ne convient pas du tout : donnez la nouvelle écriture de ces lignes.

Ce peut être un constructeur à un paramètre, dans ce cas pas de type de retour, ni de return

```
Eleve(int niv) {  
    this ("???",niv);  
}
```

ou bien, par exemple, une fonction qui permette de récupérer la variable nom et de modifier le niveau (moitié accesseur, moitié modifieur) getNom_setNiv() ou une des deux fonctions

```
String getNomSetNiv(int niv) {  
    niveau=niv;  
    return nom;  
}
```

a7 ligne 22

Que signifie ce commentaire ?

La fonction toString est définie dans la classe Object, et définie de nouveau (redéfinie) dans Eleve

a8 ligne 23

Dans quelle mesure cette ligne ne correspond-elle pas au commentaire ? Rectifiez.

Il manque la majuscule au s dans toString au lieu de toString et la fonction n'est pas déclarée public, on définit donc ainsi une nouvelle fonction qui ne se substituera pas à toString et lorsque l'on essaiera d'afficher un Eleve on utilisera le toString de Object.

Si vous avez correctement rectifié la classe `eleve`, la classe `test` compile normalement. Elle présente pourtant un problème.

a9 ligne 2

Lorsqu'on lance l'exécution du programme en indiquant la classe `Test`, l'exécution ne démarre pas, et il est indiqué qu'on ne trouve pas la méthode `main`. Comment faut-il rectifier la ligne 2 pour pouvoir exécuter le programme ?

```
public static void main(String[] args) {
```

a10 ligne 5

Que veut dire `Eleve e3=e1` ; ? Attention : réfléchissez bien avant de répondre, car la réponse n'est pas forcément si simple qu'il y paraît.

On crée une nouvelle référence à un élève nommée `e3` qui pointe sur l'instance d'Eleve elle-même référencée par `e1`

a11 exécution de la ligne 8

Lorsqu'on lance l'exécution du programme on obtient l'affichage à l'écran suivant :

```
Eleve 1   Paul niveau 5
Eleve 1   ??? niveau 0
Eleve 1   Marie niveau 2
Eleve 1   Paul niveau 5
```

C'est la ligne 12 qui provoque l'affichage des trois dernières lignes. Expliquez l'affichage obtenu, et plus particulièrement la dernière ligne.

En ligne 8 `e3` pointe sur l'objet créé ligne 3, `e1` pointe en ligne 4 sur l'objet créé ligne 3 puis en ligne 8 sur un nouvel objet créé ligne 6

a12 résolution du problème à l'exécution : rectification de la déclaration de la classe `eleve`.

Le numéro de l'élève est toujours 1. Pourtant dans le constructeur de `eleve` on prend bien la précaution d'augmenter de 1 la variable `nbEleve`. Rectifier la déclaration de la variable `nbEleve` pour que l'on obtienne bien l'affichage à l'écran suivant :

```
Eleve 1   Paul niveau 5
Eleve 2   ??? niveau 0
Eleve 3   Marie niveau 2
Eleve 1   Paul niveau 5
```

Expliquez pourquoi on obtenait toujours `Eleve 1` ?

On obtenait toujours 1 car l'attribut d'instance `nbEleve` créé à chaque nouvelle création d'instance était initialisé par défaut à 0 et dans le constructeur on ajoutait 1 (cf question a4)

Si l'on déclare l'attribut `nbEleve` comme attribut de classe en l'initialisant à 0, cet attribut aura bien les valeurs 1, 2, 3 à la création des instances Paul, ??? et Marie, mais au niveau de l'affichage on obtiendra `nbEleve` à 3 pour les trois dernières lignes. Pour obtenir un tel affichage, il faudrait définir de plus une variable d'instance `num` qui prendrait la valeur de `nbEleve` à la création de l'instance et dans le `toString` afficher `num` et non pas `nbEleve`.

B Héritage et écriture de classes

On considère un établissement de restauration qui détermine des menus.

Un menu est composé de trois plats. Le plat dénommé principal, est à choisir parmi la liste suivante : bœuf, mouton, œuf, blé complet. Les deux autres plats, dénommés entrée ou dessert, sont à choisir parmi la liste suivante : salade, gâteau, friand, glace.

Un plat principal est donc un plat que l'on choisit dans la liste adéquate.

Un plat entrée-dessert est donc un plat que l'on choisit dans la liste adéquate.

Ces deux propositions montrent qu'il peut y avoir de l'héritage à partir d'une seule idée plat, et qu'un menu est constitué de plats.

Exemple de menu : `friend boeuf gateau`

Exemple d'un ensemble de menus :

```
glace oeuf glace
glace oeuf salade
friand boeuf gateau
salade blé complet salade
gateau mouton glace
```

Il faut définir les classes suivantes : une classe mère `UnPlat` et ses deux classes filles `Principal` et `EntreeDessert`, une classe `UnMenu` qui comporte les trois champs correspondant aux trois plats et la classe `Menus` pour l'ensemble des menus.

Très important Toute indentation mal faite ou incorrecte sera notée négativement.

b1 classe `UnPlat`

Elle définit un champ héritable, `plat`, qui est un nom. Elle définit également une méthode `autreChoix`, sans paramètre et qui fournit une chaîne en résultat, et dont les instructions ne peuvent être connues à ce niveau de la hiérarchie.

```
abstract public class UnPlat {
    protected String plat;
    abstract String autreChoix();
}
```

b2 classe `Principal`

Elle hérite de `UnPlat`. Elle comporte la liste des plats principaux, sous forme d'un tableau de chaînes; ce tableau est initialisé au sein de la définition de la classe. Elle comporte un champ `choisi` dont la valeur correspond à l'indice du plat choisi dans le tableau des plats principaux.

Le constructeur par défaut détermine le plat, en choisissant aléatoirement une des quatre possibilités. La méthode `toString` retourne en résultat le nom du plat choisi.

Les instructions de la méthode `autreChoix` seront définies à ce niveau de la hiérarchie. Son principe est simple : elle renvoie la liste des plats principaux, sauf celui qui a été choisi.

Donnez la définition de la classe `Principal`.

```
public class Principal extends UnPlat{
    private static String[] listePlat={"saumon", "canard", "boeuf", "truite"};
    private int choisi;

    public Principal() {
        super();
        choisi=(int)(Math.random()*4);
        plat=listePlat[choisi];
    }
    public String toString(){
        return plat;
    }
    String autreChoix(){
        String s="";
        for (int i=0; i<listePlat.length;i++){
            if (i!= choisi) s=s+" "+listePlat[i];
        }
        return s;
    }
}
```

b3 classe EntreeDessert

Elle a grosso modo la même définition que la classe Principal, sauf la spécificité des plats.

Que faut-il modifier à la classe Principal pour obtenir une bonne définition de la classe EntreeDessert ?

Il faut modifier l'initialisation de la liste des plats, et bien sûr les identificateurs des constructeurs.

b4 classe UnMenu

Pour décrire un menu, on a besoin de trois plats : plat1, plat2 et plat3. Les deux extrêmes sont des entrées ou des desserts, et celui du milieu un plat principal.

C'est le constructeur par défaut qui crée les trois objets correspondant aux trois plats. La méthode toString renvoie en résultat, sur une même ligne, le nom des trois plats, puis passe à la ligne.

Définir la classe UnMenu.

```
public class UnMenu {
    private UnPlat plat1, plat2, plat3;

    /** Creer trois instances de UnPlat */
    public UnMenu() {
        plat1 = new EntreeDessert();
        plat2 = new Principal();
        plat3 = new EntreeDessert();
    }
    public String toString(){
        return (plat1 + " " + plat2 + " " + plat3 + "\n");
    }
}
```

b5 classe Menus

Le nombre de menus à définir est fixé par une « constante » placée dans la définition de la classe Menus. L'ensemble des menus sera stocké dans un tableau de UnMenu.

Le constructeur par défaut crée les menus, et la méthode toString liste les menus, un par ligne.

Définir la classe Menus.

```
class Menus
    public class Menus {
        private final int nbMenu =4;
        private UnMenu[] ensMenu;

        /** Creates a new instance of Menus */
        public Menus() {
            ensMenu = new UnMenu[nbMenu];
            for (int i=0; i< nbMenu; i++){
                ensMenu[i] = new UnMenu();
            }
        }
        public String toString(){
            String s="";
            for (int i=0; i<nbMenu; i++){
                s=s+ensMenu[i].toString();
            }
            return s;
        }
    }
```

```
}
```

b6 classe de test

Donner la définition d'une classe permettant de tester le fonctionnement de ce programme.

```
public class TestMenus {  
  
    public static void main (String[] a){  
        Principal test=new Principal();  
        System.out.println("Le plat principal choisi est " +test+ "\n" +  
            "Les autres choix sont" +test.autreChoix() );  
        Menus premier=new Menus();  
        System.out.println(premier);  
    }  
}
```

b7 Autre héritage

Les classes Principal et EntreeDessert ont beaucoup de choses en commun : elles ne diffèrent en fait que par la liste des plats.

Que proposez-vous pour éviter d'avoir à dupliquer toutes ces instructions ? Pensez à une forme particulière d'héritage et faite attention à la restriction liée à extends.

On peut par exemple créer une classe abstraite qui contiendrait les différentes méthodes de Principal et EntreeDessert et faire dériver de cette classe Principal et EntreeDessert qui n'auraient plus qu'à initialiser les différents plats.

Fichier : ChoixPlat.java

```
abstract public class ChoixPlat extends UnPlat{  
    protected String[] listePlat;  
    protected int choisi;  
  
    public ChoixPlat() {  
        listePlat=new String[4];  
        choisi=(int)(Math.random()*4);  
    }  
    public String toString(){  
        return plat;  
    }  
    String autreChoix(){  
        String s="";  
        for (int i=0; i<listePlat.length;i++){  
            if (i!= choisi) s=s+listePlat[i];  
        }  
        return s;  
    }  
}
```

Fichier : Principal.java

```
public class Principal extends ChoixPlat{  
    /** Creates a new instance of Principal */  
    public Principal() {  
        super();  
    }  
}
```

```

    listePlat[0]="saumon";
    listePlat[1]= "canard";
    listePlat[2]="boeuf";
    listePlat[3]="truite";
    plat=listePlat[choisi];
}
}
Fichier : Principal.java
public class EntreeDessert extends ChoixPlat{
    /** Creates a new instance of EntreeDessert */
    public EntreeDessert() {
        listePlat[0]="Tomates";
        listePlat[1]= "Glaces";
        listePlat[2]="Tartes";
        listePlat[3]="Saucisson";
//    choisi=(int)(Math.random()*4);
        plat=listePlat[choisi];
    }
}
}

```