POO
Java
2020

PROGRAMMATION ORIENTÉE OBJET (POO)

Application au langage Java

Travaux pratiques

POO

Série N° 1

Initialisation

Java

Youssouf EL ALLIOUI

y.elallioui@usms.ma

Exercice 1. Afficher les arguments de la méthode *main* ( )

Dans l'en-tête d'une méthode $main$ () apparait ce qui s'appelle un paramètre ou un argument qui est ci-dessous nommé arg :

$$\textbf{\textit{public static void}}\ main(String[\,]\ arg)$$

On peut nommer cet argument selon son propre choix. L'en-tête pourrait aussi être :

$$\textbf{\textit{public static void}}\ main(String[\,]\ listeArguments)$$

Ou tout autre identificateur à la place de $listeArguments$, ou de arg plus haut. Le reste de l'en-tête de la méthode $main()$ est obligatoire. Cet argument est de type tableau de String ou encore tableau de chaînes de caractères, mais pour l'instant il suffit de savoir que si son nom est arg, le premier d'entre eux est accessible dans le programme par $arg[0]$, le deuxième par $arg[1]$ et ainsi de suite. Grâce à ce paramètre, on peut "passer des arguments à la méthode $main$", ce qu'on fait quelquefois, cela dépend du choix du programmeur.

On souhaite dans cet exercice passer un argument, de type tableau des String, à la méthode $main$.

1) Ecrivez le programme qui permet d'afficher tous les éléments du tableau passé en paramètre à la méthode $main()$.

Exercice 2. Modéliser un étudiant

1) Créez un nouveau projet sous Eclipse que vous nommez par exemple $Exercice02$.

2) Créez un nouveau paquetage, $etudiant$, dans le projet $Exercice02$.

3) Créez une nouvelle classe, $Etudiant$, dans le paquetage $etudiant$.

4) Complétez le code de cette classe en lui ajoutant :

a) Un attribut privé de type $String$ nommé nom ;

b) Un constructeur $public$ qui a un paramètre de type $String$ servant à initiliser le nom de l'étudiant ;

c) Une méthode publique sans paramètre et qui ne renvoie rien, nommée $travailler$ (), qui écrit à l'écran, si le nom de l'étudiant est toto, "$toto\ se\ met\ au\ travail$!".

d) Une méthode publique sans paramètre et qui ne renvoie rien, nommée $seReposer$ (), qui écrit à l'écran, si le nom de l'étudiant est toto, "$toto\ se\ repose$".

5) Vous allez ensuite créer parallèlement à la classe $Etudiant$ une classe $TestEtudiant$ (aussi dans le paquetage $etudiant$) contenant une méthode $main$ qui :

a) Crée un étudiant e, instance de la classe $Etudiant$, en lui donnant un nom écrit directement dans le fichier source ;

b) Invoque la méthode $travailler$ de l'étudiant e ;

c) Invoque la méthode $seReposer$ de l'étudiant e.

6) Exécutez votre programme.

7) Si tout fonctionne bien, améliorez un peu votre programme en sorte que le nom de l'étudiant soit indiqué comme argument de la méthode $main$ () de la classe $TestEtudiant$.

Exercice 3. Modéliser un segment de droite

Il s'agit de la modélisation d'un segment de droite dont les valeurs des abscisses des deux extrémités sont réelles. Les opérations que l'on souhaite faire sur ce segment sont :

- Calculer sa longueur ;
- Savoir si un point d'abscisse donné se trouve sur le segment.

1) Écrire le code d'une classe publique $Segment$ se trouvant dans un paquetage de nom $segment$ comportant :

a) Deux attributs privés de type $\boldsymbol{double}$, $extr1$ et $extr2$, représentant les abscisses des extrémités d'un segment sur un axe ;

b) Un constructeur de ce segment recevant en arguments les deux valeurs entières des abscisses des extrémités du segment

c) Une méthode publique retournant la longueur du segment ;

d) Une méthode dont le prototype est : public $\boldsymbol{boolean}\ appartient(\boldsymbol{double}\ x)$ indiquant si le point de coordonnée x appartient ou non au segment ;

e) Le getter $public\ double\ getExtr1()$;

f) Le setter $public\ void\ setExtr1(double\ a)$;

g) Le getter $public\ double\ getExtr2()$;

h) Le setter $public\ void\ setExtr2(double\ a)$;

i) Une méthode d'en-tête $public\ String\ toString$ () qui décrira une instance de la classe $Segment$ sous la forme d'une chaîne de caractères : par exemple, pour le segment d'extrémités -35 et 44, cette méthode retourne la chaîne "$segment\ [-35, 44]$".

2) Définissez aussi dans le paquetage $segment$ une classe $TestSegment$ pour tester la classe $Segment$.

Exemples d'exécution souhaitée :

Pour la méthode $main$ de la classe $TestSegment$, si les arguments $extr1$, $extr1$ et x ont respectivement les valeurs -35, 44 et 8, la sortie du test sera :

```
Longueur du segment [-35, 44] : 79
```

```
8 appartient au segment [-35, 44]
```

Si les arguments $extr1$, $extr1$ et x ont respectivement les valeurs -35, 44 et 100, la sortie du test sera :

```
Longueur du segment [-35, 44] : 79
100 n'appartient pas au segment [-35, 44]
```

Exercice 4. Le jeu Caillou-Ciseaux-Papier

Le jeu de Caillou-Ciseaux-Papier s'appelle aussi Chifoumi. Il se joue entre deux joueurs, en général avec les mains. Simultanément, les deux joueurs font un signe avec les mains qui représente soit un $caillou$, soit des $ciseaux$, soit un $papier$.

Nommons A et B les deux joueurs. Si les deux joueurs ont fait le même signe, on considère que c'est un cas d'égalité, aucun des deux joueurs ne marque un point.

- Si le joueur A a joué $Caillou$ et le joueur B $Ciseaux$, A marque un point car "le caillou émousse les ciseaux", et réciproquement.
- Si le joueur A a joué $Papier$ et le joueur B $Caillou$, A marque un point car "le papier enveloppe le caillou", et réciproquement.
- Si le joueur A a joué $Ciseaux$ et le joueur B $Papier$, A marque un point car "les ciseaux coupent le papier", et réciproquement.

On donne un nombre de points à atteindre (3 par exemple) et le premier joueur qui a atteint ce nombre de points a gagné.

1) Modélisez et programmez ce jeu.

Exercice 5. Modélisation d'un élève

Un élève sera ici modélisé par la classe $Eleve$ d'un paquetage nommé $gestionEleves$, de la façon suivante.

La classe $Eleve$ posséde trois attributs privés :

- Son nom, nommé nom, de type $String$,
- Un ensemble de notes, nommé $listeNotes$, qui sont des entiers rangés dans un $ArrayList$
- Une moyenne de type $\boldsymbol{double}$, nommée $moyenne$, qui doit toujours être égale à la moyenne des notes contenues dans l'attribut $listeNotes$. Un élève sans aucune note sera considéré comme ayant une moyenne nulle.

La classe $Eleve$ possède un constructeur permettant uniquement d'inialiser le nom de l'élève.

La classe $Eleve$ possède aussi cinq méthodes publiques :

- Une méthode $getMoyenne()$ pour récupérer la moyenne de l'élève à chaque moment.
- Une méthode $getNom()$ pour récupérer le nom de l'élève.
- Une méthode $getListeNotes()$ qui renvoie la liste des notes de l'élève.
- Une méthode $ajouterNote(\boldsymbol{double}\ note)$ qui ajoute la note reçue en paramètre à l'attribut $listeNotes$; si la note reçue en paramètre est négative, la note introduite est 0 (si la note reçue en paramètre est supérieure à 20, la note introduite est 20) ;
- La méthode $toString()$ qui retourne une description de l'élève considéré. Par exemple :

```
Youssouf (15.0) - 12.50, 15.50, 17.0
```

1) Après avoir terminé la classe $Eleve$, écrivez un programme qui teste cette classe.

Exercice 6. Modéliser une liste des élèves

Cet exercice fait suite à **`Exercice 5. Modélisation d'un élève`**, qu'il faut donc faire avant celui-ci.

Un groupe d'$Eleve$(s) (instances de la classe $gestionEleves.Eleve$ précédemment définie) sera ici modélisé par la classe $GroupeEleves$ du paquetage $gestionEleves$ de la façon suivante :

- La classe $GroupeEleves$ posséde un attribut privé : une collection d'$Eleve$(s) nommée $listeEleves$, de type $ArrayList$.
- La classe $GroupeEleves$ ne possède pas de constructeur explicite.
- La classe $GroupeEleves$ possède aussi cinq méthodes publiques :
  - Une méthode $nombre()$ qui renvoie le nombre d'$Eleve$(s) contenus dans $listeEleves$
  - Une méthode $getListe()$ qui renvoie $listeEleves$.
  - Une méthode $ajouterEleve(Eleve\ eleve)$ permet d'ajouter un $Eleve$ passé en paramètre à $listeEleves$.
  - Une méthode $chercher(String\ nom)$ qui renvoie l'$Eleve$ dont le nom est indiqué par le paramètre ; si plusieurs $Eleve$(s) ont même nom, la méthode renvoie le premier $Eleve$ ayant ce nom contenu dans $listeEleves$; si aucun $Eleve$ n'a le nom indiqué, la méthode retourne $\boldsymbol{null}$. On pourra utiliser la méthode $equals$ de la classe $String$ pour comparer une chaîne de caractères à une autre.
  - Une méthode $lister()$ qui affiche la liste des $Eleve$(s). Elle utilise une ligne par $Eleve$; elle utilise la méthode $toString$ de la classe $Eleve$.

1) Après avoir terminé la classe $GroupeEleves$, écrivez un programme qui teste cette classe.

Exercice 7. Des objets qui communiquent

Il s'agit de faire en sorte que deux classes partagent un même objet ; ce genre de situation se produit tout le temps lorsqu'on travaille avec un langage objet.

On définit ici quatre classes très simples :

- La classe $Gadget$ contient :
  - Un attribut privé nommé $valeur$ de type int,
  - Une méthode $toString()$ qui retourne la chaîne de caractères "$valeur\ du\ gadget$: " concaténée avec la chaîne de caractères donnant $valeur$.
  - Une méthode $incrementer()$ qui admet un paramètre de type int et incrémente l'attribut nommé $valeur$ de la quantité indiquée par le paramètre.
- La classe A est :

```
class A {
    private Gadget gadget = new Gadget();
    private int entier;

    Gadget getGadget () {
        return gadget;
    }

    public String toString () {
        return "Instance de A, entier = " + entier + ", "+ gadget.toString();
    }
}
```

- La classe B contient :
  - Un attribut privé de type $Gadget$, non initialisé au moment de sa définition (contrairement à celui de la classe A), nommé $gadget$;

- Un attribut privé de type int nommé $entier$;
- Une méthode $toString$ () identique à celle de la classe A, au nom de la classe près ;
- Un constructeur ayant un paramètre de type $Gadget$ servant à initialiser son attribut $gadget$;
- Une méthode $incrementer$ () ayant deux paramètres de type int ; cette méthode incrémente l'attribut $entier$ de la valeur de son premier paramètre et incrémente l'attribut $valeur$ de $gadget$ de la valeur de son second paramètre.

- La classe $EssaiAB$ contient uniquement une méthode $main$ () ; cette méthode :
  - Définit et initialise une variable locale a de type A ;
  - Définit et initialise une variable locale b de type B en faisant en sorte que l'attribut $gadget$ de b soit égal à l'attribut $gadget$ de a ;
  - puis son code est :

```
System.out.println(a);
System.out.println(b);
b.incrementer(5, 10);
System.out.println("Après incrémentation : ");
System.out.println(a);
System.out.println(b);
```

Un exemple d'exécution
Pour la commande java $EssaiAB$, on obtient :

```
Instance de A,  entier = 0, valeur du gadget = 0
Instance de B,  entier = 0, valeur du gadget = 0
Après incrémentation :
Instance de A,  entier = 0, valeur du gadget = 10
Instance de B,  entier = 5, valeur du gadget = 10
```

Discussion

Les classes A et B partagent un même objet de type $Gadget$; deux objets peuvent de même partager un même tableau : il est possible de partager un $objet$ ou un $tableau$ car ils sont manipulés par $\boldsymbol{référence}$, mais il n'est pas possible de partager un attribut de type $\boldsymbol{primitif}$.

Exercice 8. Méthodes de classe

Il s'agit d'écrire une classe nommée $\boldsymbol{OpTabInt}$ rasssemblant un ensemble de fonctions statiques pour gérer des tableaux d'int. Une seconde classe nommée $\boldsymbol{EssaiOp}$ devra uniquement contenir une fonction main destinée à tester les méthodes de la classe $OpTabInt$. Les tableaux considérés ne seront jamais "surdimensionnés", on en considérera toujours toutes les cases.

La classe OpTabInt :

Cette classe ne contient aucun attribut et ne définit pas de constructeur. Elle contient les méthodes suivantes :

- Une méthode statique $\boldsymbol{somme}$ () prend en paramètre un tableau d'int et retourne la somme des int contenus dans le tableau.
- Une méthode statique $\boldsymbol{getIndiceMax}$ () prend en paramètre un tableau d'int et détermine l'indice du plus grand int du tableau ; si le tableau est de longueur nulle, elle retourne -1.
- Une méthode statique $\boldsymbol{ajouter}$ () prend en paramètres un tableau d'int et un entier et retourne un nouveau tableau qui est obtenu en concaténant le tableau et l'entier.
- Une méthode statique $\boldsymbol{afficher}$ () prend en paramètre un tableau d'int et affiche son contenu ; les données doivent figurer sur une même ligne et on passe à la ligne après la dernière donnée.

La classe EssaiOp

Cette classe ne contient qu'une méthode $main$ (). Elle a pour objectif de tester les différentes méthodes de la classe $OpTabInt$.

La méthode $main$ possède un tableau d'int, nommé $tableau$, en variable locale. Elle utilise les chaînes de caractères passées en arguments par la ligne de commande pour remplir $tableau$. Le tableau $tableau$ doit avoir une longueur égale au nombre de ces arguments.

Après avoir rempli $tableau$, la méthode $main$ () utilise successivement les méthodes $afficher$ (), $somme$ () et $getIndiceMax$ () de la classe $OpTabInt$ en écrivant à l'écran les résultats de ces méthodes appliquées à $tableau$.

Après cela, la méthode $main$ () utilise la méthode $ajouter$ () de la classe $OpTabInt$ pour que $tableau$ devienne le tableau obtenu en ajoutant un entier (par exemple 20) au bout de $tableau$. Les contenus du tableau initial (qui doit être inchangé) et ce du tableau résultat sont affichés à l'écran.

Exemple d'exécution

Si la commande est $java\ EssaiOp\ 3\ -8\ 14\ 4$, le résultat peut être :

```
Le tableau : 3 -8 14 4
somme du tableau = 13
indice du max du tableau = 2
```

Après ajout :

```
Le tableau initial est : 3 -8 14 4
Et le tableau augmenté est : 3 -8 14 4 20
```