DE LA PAR DOUATI OMAR ET MOKRANI AMZIGH ET HADDAD KOUCIALA

C4 : Exercices corrigés de P.O.O
<-- ***C3 : La nature de l'***Objet C5 : Héritage et Polymorphisme -->

Exercice 1 : Savoir créer une classe simple puis l' utiliser dans un programme

Ecrire 3 classes Points1, Points2 et Points3, permettant de créer des objets points dans les espaces dimension 1, dimension 2 et dimension 3. Chacune de ces classes doit être dotée d' au moins un constructeur. Ecrivez toutes les méthodes que vous croyez pouvoir être appliquées à de tels objets.
Ensuite, créer un petit programme qui utilise ces points : affiche le nom de ces points, leur position, déplacement de ces points, distance entre deux points, vérifier si les points sont alignés.

Correction Points1

```
1.    class Points1
2.    {
3.        double x; char nom;
4.
5.        public Points1(char nom, double x)
6.        {
7.            this.x = x;
8.            this.nom = nom;
8.        }
9.
10.       public void afficherCoord()
11.       {
12.           System.out.println("Le point " + this.nom + " a pour abscisse " + x);
13.       }
14.
15.       public double distance(Points1 M)
16.       {
17.            double d = this.x - M.x;
18.            return Math.abs(d);
19.       }
20.
21.       public void deplacer(double dx)
22.       {
23.           this.x += dx;
24.       }
25.   }
```

```
1.    public class TesterPoints1
2.    {
3.        public static void main(String [] args)
4.        {
5.            Points1 A = new Points1('A', 2.6);
6.            A.afficherCoord();
7.            A.deplacer(1.1);
8.            A.afficherCoord();
9.            Points1 B = new Points1('B', 1.54);
10.           B.afficherCoord();
11.           System.out.println("Distance entre A et B = " + A.distance(B) );
12.       }
13.  }
```

Quelque chose de nouveau.
Dans le même fichier, on a 2 classes. C' est possible en java. A condition qu' une seule classe soit déclarée avec le mot clé public. Et c' est la classe déclarée avec le mot public qui porte le nom du fichier. Donc vous enregistrerez ce fichier sous le nom TesterPoints1.java

Compilez puis exécutez ce programme. Il sera affiché :

```
Le point A a pour abscisse 2.6
Le point A a pour abscisse 3.7
Le point B a pour abscisse 1.54
Distance entre A et B = 2.16
```

Correction Points2

```
1.    class Points2
2.    {
3.        double x; double y; char nom;
4.
5.        public Points2(char nom, double x, double y)
6.        {
7.            this.x = x; this.y = y;
8.            this.nom = nom;
8.        }
9.
10.       public void afficherCoord()
11.       {
12.           System.out.println("Le point " + this.nom + " a pour coordonnées " + x + " et " + y);
13.       }
14.
15.       public double distance(Points2 M)
16.       {
17.            double d = Math.sqrt( (this.x - M.x) * (this.x - M.x) + (this.y - M.y) * (this.y - M.y));
18.            return d;
19.       }
20.
21.       public void deplacer(double dx, double dy)
22.       {
23.           this.x += dx; this.y += dy;
24.       }
25.
26.       public boolean alignement(Points2 M, Points2 N)
27.       {
28.           boolean verite = false;
29.           if( (M.x - this.x)/(N.x - this.x) == (M.y - this.y)/(N.y - this.y) )
30.           {
31.               verite = true;
32.           }
33.           return verite;
34.       }
35.   }
```

```
1.    public class TesterPoints2
2.    {
3.        public static void main(String [] args)
4.        {
5.            Points2 A = new Points2('A', 0, 2);
6.            A.afficherCoord();
7.            A.deplacer(1, 1);
8.            A.afficherCoord();
9.            Points2 B = new Points2('B', 2, 5);
10.           B.afficherCoord();
```

```
123  11.           Points2 C = new Points2('C', 3, 7);
124  12.           C.afficherCoord();
125  13.           System.out.println("Distance entre A et B = " + A.distance(B) );
126  14.           System.out.print("A, B et C sont-ils alignés ? : ");
127  15.           if(A.alignement(B, C)
128  16.           {
129  17.                System.out.println("VRAI");
130  18.           }
131  19.           else
132  20.           {
133  21.               System.out.println("FAUX");
134  22.           }
135  23.       }
136  13.  }
137
138  compilez puis exécutez. Il sera affiché :
139
140  Le point A a pour coordonnées 0.0 et 2.0
141  Le point A a pour coordonnées 1.0 et 3.0
142  Le point B a pour coordonnées 2.0 et 5.0
143  Le point C a pour coordonnées 3.0 et 7.0
144  Distance entre A et B = 2.23606797749979
145  A, B et C sont-ils alignés ? : VRAI
146
147  Correction Points3
148
149  1.    class Points3
150  2.    {
151  3.        double x; double y; double z; char nom;
152  4.
153  5.        public Points3(char nom, double x, double y, double z)
154  6.        {
155  7.            this.x = x; this.y = y; this.z = z;
156  8.            this.nom = nom;
157  8.        }
158  9.
159  10.       public void afficherCoord()
160  11.       {
161  12.           System.out.println("Le point " + this.nom + " a pour coordonnées " + x
+ " et " + y + " et " + z);
162  13.       }
163  14.
164  15.       public double distance(Points3 M)
165  16.       {
166  17.            double d = Math.sqrt( (this.x - M.x) * (this.x - M.x) + (this.y - M.y
) * (this.y - M.y) + (this.z - M.z) * (this.z - M.z));
167  18.            return d;
168  19.       }
169  20.
170  21.       public void deplacer(double dx, double dy, double dz)
171  22.       {
172  23.           this.x += dx; this.y += dy; this.z += dz;
173  24.       }
174  25.
175  26.       public boolean alignement(Points3 M, Points3 N)
176  27.       {
177  28.           boolean verite = false;
178  29.           double a = (M.x - this.x)/(N.x - this.x);
179  30.           double b = (M.y - this.y)/(N.y - this.y); double c = (M.z - this.z)/(N
.z - this.z);
180  29.           if( a == b && a == c )
181  30.           {
182  31.               verite = true;
183  32.           }
184  33.           return verite;
185  34.       }
```

```
    }

    public class TesterPoints3
    {
        public static void main(String [] args)
        {
            Points3 A = new Points3('A', 0, 2, -2);
            A.afficherCoord();
            A.deplacer(1, 1, 1);
            A.afficherCoord();
            Points3 B = new Points3('B', 2, 5, 1);
            B.afficherCoord();
            Points3 C = new Points3('C', 3, 7, -2.2);
            C.afficherCoord();
            System.out.println("Distance entre A et B = " + A.distance(B) );
            System.out.print("A, B et C sont-ils alignés ? : ");
            if( A.alignement(B, C) )
            {
                 System.out.println("VRAI");
            }
            else
            {
                System.out.println("FAUX");
            }
        }
    }
```

compilez puis exécutez. Il sera affiché :

```
Le point A a pour coordonnées 0.0 et 2.0 et -2.0
Le point A a pour coordonnées 1.0 et 3.0 et -1.0
Le point B a pour coordonnées 2.0 et 5.0 et 1.0
Le point C a pour coordonnées 3.0 et 7.0 et -2.2
Distance entre A et B = 3.0
A, B et C sont-ils alignés ? : FAUX
```

Exercice 2 : Comprendre les constructeurs. La classe Livre

Voici une classe dans laquelle il n' y a que des champs et des constructeurs.

```
    public class Livre
    {
        public static int nombreLivres;
        private String auteur, titre, dateParution; private int nombrePages;
        public Livre(String auteur, String titre)
        {
            this.auteur = auteur; this.titre = titre;
        }

        public Livre(String auteur, String titre, String dateParution)
        {
            this.auteur = auteur; this.titre = titre; this.dateParution =
dateParution;
        }

        public Livre(String auteur, String titre, String dateParution, int
nombrePages)
        {
            this.auteur = auteur; this.titre = titre;
            this.dateParution = dateParution; this.nombrePages = nombrePages;
        }
    }
```

On peut créer un objet Livre en ayant besoin seulement de son titre et du nom de l' auteur. D' où le premier constructeur. On peut créer un objet Livre en ayant besoin en plus de sa date de parution. D' où le deuxième constructeur. On peut créer un objet Livre

en ayant besoin de en plus du nombre de pages. Question 1 : comment peut-on améliorer l'écriture du 2ème et du 3ème constructeur ? Cette amélioration aura pour but de diminuer le nombre d' instructions. Question 2 : La variable nombreLivres est déclarée avec le mot static. Que signifie ce mot (en terme d' espace mémoire)? Question 3 : Dans quel constructeur doit-on placer l' instruction servant à compter le nombre de livres dans un programme ? Quelle est cette instruction ? Question 4 : Ecrivez un petit programme qui permet de créer des livres tout en affichant le nombre de livres créés.

Correction

Réponse 1 :

```
9.        public Livre(String auteur, String titre, String dateParution)
10.       {
11.           this(String auteur, String titre); this.dateParution = dateParution;
12.       }

14.       public Livre(String auteur, String titre, String dateParution, int
nombrePages)
15.       {
16.           this(String auteur, String titre, String dateParution);
17.           this.nombrePages = nombrePages;
18.       }
```

Réponse 2 :

Toute variable déclarée avec le mot static se trouve en un seul exemplaire dans la mémoire.

Réponse 3 :

On doit placer cette instruction dans tous les contructeurs. Cette instruction est :

```
nombreLivres++;
```

Réponse 4 :

```
1.    class Livre
2.    {
3.        static int nombreLivres;
4.        String auteur; String titre; String dateParution; int nombrePages;
4.        public Livre(String auteur, String titre)
5.        {
6.            this.auteur = auteur; this.titre = titre; nombreLivres++;
7.        }
8.
9.        public Livre(String auteur, String titre, String dateParution)
10.       {
11.           this(auteur, titre); this.dateParution = dateParution;
12.       }
13.
14.       public Livre(String auteur, String titre, String dateParution, int
nombrePages)
15.       {
16.           this(auteur, titre, dateParution);
17.           this.nombrePages = nombrePages;
18.       }
19.   }
20.
21.    public class CompterLivres
22.    {
23.        public static void main (String [] args)
24.        {
25.            Livre A = new Livre("Beaudelaire", "Les fleurs du mal");
26.            Livre B = new Livre("Guy Des CARS", "La maudite", "1972");
```

```
27.            Livre C = new Livre("Anne TASSO", "initiation à JSP", "2002", 321);
28.            System.out.println("nombre de livres créés = " + Livre.nombreLivres);
29.        }
30.    }
```

Réponse 1 :

Lorsque dans une classe, un constructeur 2 utilise les mêmes instructions qu' un autre 1. On peut remplacer ces intructions grâce au mot clé this. Dans le constructeur 2, on écrit this, suivi des parenthèses à l' **intérieur desquelles on met les noms des paramètres du constructeur 1. Ces noms sont mis dans le même ordre que dans le constructeur 1. Et on ne mentionne pas les types de ces paramètres. D'** où this(auteur, titre); dans le constructeur 2. Attention, ne surtout pas mettre this(String auteur, String titre);

Pour le constructeur 3, on utilise les instructions du constructeur 1, plus une instruction du constructeur 2. Mais puisque l' autre instruction du constructeur 2 remplace les instructions du constructeur 1, alors le constructeur 3 fera appel directement au constructeur 2. D' **où : this(auteur, titre, dateParution); qui fait appel aux instructions du constructeur 1 et à l'** autre instruction de constructeur 2.

Réponse 2 :

En programmation orientée Objet, Chaque champ de la classe est recopié à chque création d' un objet. Il existe le même champs associé à chaque objet. Sauf évidemnt si le champ est déclaré avec le mot clé static. C' **est en général le cas lorsqu'** on veut compter le nombre d' **objets créés, ou le nombre de fois où on a utilisé une méthode, ou tout autre décompte. Si on enlève le mot static, sa valeur sera toujours égale à 1. Chaque objet est créé une seule fois.**

Réponse 3 :

On dois placer l' instruction permettant de créer des objets dans tous les constructeurs. Parce que si tel n' **est pas le cas, les seuls objets pris en compte seront les objets créés avec le constructeur dans lequel on a placé l'** intruction permettant de compter.

Cette instruction est naturellement nombreLivres++;Parce que en créant un premier objet, ce chapms est égal à zéro. nombreLivres++; permet d' incrémenter de 1 cette variable.

Réponse 3 : Compilez puis exécutez le programme, il sera affiché :

***nombre de livres créés* = 3**

Les classes Livre et CompterLivres dans le même fichier. Je vous l' ai déjà dit : c' **est possible. Une seule classe doit être créée avec le mot public. J'** ai améliorer la classe Livre en améliorant les constructeurs comme décrit en réponse 1. Puis le programme crée des livres. Et à chaque fois, le nombre de livres est incrémenté de 1. D' **où à l'** affichage : nombre de livres = 3. 3 correpond au nombre de livres créés.

Raappel : Dans la dernière instruction du programme, on voit Livre.nombreLivres. On demande à afficher le nombre de livres en faisant appel à la variable nombreLivre. Cet appel se fait en écrivant nom de la classe, suivi d' un point, suivi du nom de la variable. C' **est toujours comme ça qu'** on utilise une variable static dans une classe qui n' **est pas celle dans laquelle ellle a été déclarée. Si le mot static n'** existait pas dans sa déclaration, on aurait fait appel à ce type de variable en invoquant le nom de l' **objet, suivi d'** un point, suivi du nom de l' **objet.**

Exercice 3 : Comparer les objets partie 1 : opérateur ==

Prévoyez l' affichage (la sortie) de ce programme sans le compiler, ni l' **exécuter.**
Puis comparer ce que vous avez prévu avec la réalité après compilation, puis exécution.

```
1.    public class TestStrings
2.    {
```

```
3.        public static void main(String arg [] )
4.        {
5.            String nom1 = new String("pagall");
6.            System.out.println("nom1 = " + nom1);
7.            String nom2 = new String("pagall");
8.            System.out.println("nom2 = " + nom2);
9.            if(nom1 == nom2)
10.           {
11.               System.out.println("nom1 égal à nom2");
12.           }
13.           else
14.           {
15.               System.out.println("nom1 différent de nom2");
16.           }
17.
18.           String nom3 = "aristote";
19.           System.out.println("nom3 = " + nom3);
20.           nom3 = nom1;
21.           System.out.println("nom3 = nom1");
22.           if(nom3 == nom1)
23.           {
24.               System.out.println("nom1 égal à nom3");
25.           }
26.           else
27.           {
28.               System.out.println("nom1 différent de nom3");
29.           }
30.
31.       }
32.   }
```

La ligne 6 permet d' afficher : nom1 = pagall
La ligne 7 permet d' afficher : nom2 = pagall
La structure conditionnelle allant des lignes 9 à 16 permet d' afficher : nom1 different de nom2
Parce que effectivement, nom1 et nom2 font référence à 2 objets différents.
La ligne 19 permet d' afficher : nom3 = aristote
La ligne 21 permet d' afficher : nom3 = nom1
La structure conditionnelle allant de 22 à 29 permet d' afficher : nom1 égal à nom3
Parce qu' ici, l' instruction nom3 = nom1; permet de mettre la référence nom1 dans nom3.

L' opérateur == permet de comparer 2 objets. Il permet surtout de comparer les références. L' égalité des références entraine l' égalité des objets.

Exercice 4 : Comparer les objets 2 : méthode equals() de la classe Object

Prévoyez l' affichage (la sortie) de ce programme sans le compiler, ni l' exécuter.
Puis comparer ce que vous avez prévu avec la réalité après compilation, puis exécution.

```
1.    public class TestStrings
2.    {
3.        public static void main(String arg [] )
4.        {
5.            String nom1 = new String("pagall");
6.            System.out.println("nom1 = " + nom1);
7.            String nom2 = new String("pagall");
8.            System.out.println("nom2 = " + nom2);
9.            if(nom1.equals(nom2) )
10.           {
11.               System.out.println("nom1 égal à nom2");
12.           }
13.           else
14.           {
```

```
15.                 System.out.println("nom1 différent de nom2");
16.             }
17.
18.         }
19.     }
```

La ligne 6 permet d' afficher : nom1 = pagall
La ligne 7 permet d' afficher : nom2 = pagall
La structure conditionnelle allant des lignes 9 à 16 permet d' afficher : nom1 égal à nom2
Normal, la méthode equals compare les valeurs des Objets.
La méthode equals() se trouve dans la classe prédéfinie Object. Comme je vous l' ai déjà dit, tous les objets en java appartiennent à la classe Object. Notion d' héritage et de polymorphisme que nous verront au prochain chapitre. Toutes les classes (prédéfinies ou pas) héritent de toutes les méthodes (de type public) de la classe Object.

Exercice 5 : Champs et méthodes d' une classe partie 1
Parmi les champs et les méthodes de cette classe, dîtes quels sont les éléments qui caractérisent le P.O.O ?

```
1.      public class Quelconque
2.      {
3.          public static double a;
4.          private static int compteur;
5.          private int c; private String chaine;
6.
7.          public Quelconque()
8.          {
9.              ...
10.         }
11.
12.         public static int methode1()
13.         {
14.             ...
15.         }
16.
17.         public double methode2(...)
18.         {
19.             ...
20.         }
21.
22.         private String methode3(...)
23.         {
24.             ...
25.         }
26.     }
```

Dans cette classe, les champs c et chaine caractérisent la Programmation Orientée Objet. Ce sont des champs déclarés sans le mot clé static. Donc ils se trouvent en plusieurs exemplaires dans la mémoire. De même, les méthodes 2 et 3 caractérisent la P.O.O.
Mais le champ compteur, bien que déclaré avec le mot static caractérise aussi la P.O.O parce que le champs est privé. C' est la notion d' encapsulation qui est mise en oeuvre ici.

Exercice 6 : Champs et méthodes d' une classe partie 2
Quels sont les erreurs comises dans la définition de la classe Entiers, mais aussi dans son utilisation dans le programme TesterEntiers ?

```
1.      class Entiers
2.      {
3.          private int a;
5.          private static final int b = 20;
```

```
6.
7.          static int methode1(int n)
8.          {
9.              a = n;
10.         }
11.
12.         void methode2(int n)
13.         {
14.             a = n;
15.             b = n;
16.         }
17.
18.     }
19.
20.
21.      public class TesterEntiers
22.      {
23.          public static void main(String [] args)
24.          {
25.              Entiers e = new Entiers();
26.              int n = 5;
26.              e.methode2(n);
27.              e.methode1(n);
28.              methode1(n);
29.          }
30.      }
```

***Remarquez** : le mot **public** n'* existe pas dans les méthodes methode1 et methode2. Ce n' ***est pas grave**. **Le fichier contient déjà une classe déclarée avec le mot public**.* L' autre classe ne doit pas contenir le mot public. Dans ce cas, les méthodes publicde cette classes peuvent elles aussi se passer du mot public. Pas d' ***erreur à la compilation**. **Les champs publics peuvent aussi se passer du mot public dans ce cas là**.*

***Maintenant**, **les erreurs comises**...*

Exercice** 7 : **Champs et méthodes d' une classe partie 3

*L' encapsulation permet à une classe et à elle seule de manipuler ses champs privés. Mais si l' **on a quand même** **besoin d'*** afficher la valeur de ce champ privé dans un programme. Que faire ?

***Réponse** : **on crée une méthode public dans cette classe et on y introuduit une seule instruction dans sa définition**. **return champPrive**;*

Exemple** : **dans la classe Points2**. **J' ajouterais par exemple la méthode getAbscisse().

```
public double getAbsicce()
{
    return this.x;
}
```

x étant bien entendu le champ privé représentant l' abscisse du point sur lequel on ***applique la méthode**.*

Dans le jargon des programmeurs java**, **ce genre de méthode s' appelle getteur.

Exercice** 8 : **Champs et méthodes d' une classe partie 4 : méthodes surdéfinies

***Dans le programme ci-dessous**, **on a surdéfini certaines méthodes**. **Détecter les erreurs liées à la surdéfinition**.*

```
1.      public class Surdefinition
2.      {
3.          public void methode1(int a)
```

```
4.          {
5.              ...
6.          }
7.
8.          public int methode1(int b)
9.          {
10.             ...
11.         }
12.
13.         public void methode2(float f)
14.         {
15.             ...
16.         }
17.
18.         public void methode2(final double y)
19.         {
20.             ...
21.         }
22.
24.         public void methode3(long l)
25.         {
26.             ...
27.         }
28.
29.         public void methode3(final long p)
30.         {
31.             ...
32.         }
33.     }
```

Les points de suspension remplacent les instructions (la définition) des méthodes. On n' en a pas besoin pour détecter les erreurs dans cette classe.

La méthode methode1 est surdéfinie. Mais il y a une erreur. Deux méthodes peuvent avoir le même nom. Mais à condition que leurs signatures respectives soient différentes. La signature de la première methode1 est int a : 1 argument de type int. C' est la même chose pour la deuxième methode1. Si on appelle (utilise) alors la méthode1, le choix est impossible. Il y a ambiguité. Même si les types de retour sont différents, le choix reste impossible.

Pas d' erreur dans les méthodes methode2. Les signatures sont différentes. Une petite remarque tout de même. Dans la définition de la deuxième méthode méthode2, il ne saurait être question de changer la valeur de y. Puisque le préfixe final indique que la variable est constante.

La méthode methode3 est surdéfinie. Mais il y a une erreur. Deux méthodes peuvent avoir le même nom. Mais à condition que leurs signatures respectives soient différentes. La signature de la première methode3 est long l : 1 argument de type long. C' est la même chose pour la deuxième methode3. Si on appelle la méthode3, le choix est impossible. Il y a ambiguité. Même si les types de retour sont différents, le choix reste impossible. Et ce, même si l' ***argument de la deuxième methode3 a un préfixe supplémentaire final. Ce terme ne compte pas lorsqu'*** il s' ***agit de comparer les signatures des méthodes.***

Exercice 9 : Champs et méthodes d' une classe partie 5 : méthodes surdéfinies

Voici la classe Surdefinitions2 avec sa méthode methode1 surdéfinie. Dans le programme TesterSurdefinitions ci-dessous, on a appelé (utilisé) cette méthode 6 fois. Dîtes dans quels cas l' appel est correct. Et dans quel cas il ne l' ***est pas. Dans le cas ou l'*** appel est correct, quelle méthode methode1 a-t-on utilisée ?

```
1.      class Surdefinitions2
2.      {
3.          public void methode1(int a)
4.          {
5.              ...
```

```
6.          }
7.
8.          public void methode1(int a, int b)
9.          {
10.             ...
11.         }
12.
13.         public void methode1(int a, double d)
14.         {
15.             ...
16.         }
17.     }
18.
19.     public class TesterSurdefinitions
20.     {
21.         public static void main(String [] args)
22.         {
23.             Surdefinitions2 sf = new Surdefinitions2();
24.             byte b; short s; int i; long l; float f; double d;
25.
26.             sf.methode1(i);
27.             sf.methode1(i, l);
28.             sf.methode1(l);
29.             sf.methode1(s, i);
30.             sf.methode1(b, f);
31.             sf.methode1(l, f);
32.         }
33.     }
```

N' essayez pas de compiler. Encore moins d' **exécuter**. **Le programme n'** est pas correct. L' **exercice est purement théorique**. **Les ... ne sont pas acceptés dans un programme java**.

Ligne 26 : **on appelle la méthode avec un argument de type int**. L' appel est correct. On a appelé la première methode1() dont l' **argument est aussi de type int**.

Ligne 27 : **On appelle la méthode avec** 2 **arguments dont le premier est de type int et le deuxième de type long**. L' appel est correct. On a appelé la troisième methode1() dont les 2 arguments sont tels que le premier est de type int et le deuxième de type double. L' **argument de type long utilisé est implicitement converti en double**. L' appel de la ligne 27 ne peut pas être celui de la première methode1(). Le nombre d' **arguments ne correspond**. **Cet appel ne peut pas nom plus être celui de la deuxième methode1**(). **Le deuxième argument de type long ne peut pas être converti en int**. **Perte de précision**

Ligne 28 : **On appelle la méthode avec un seul argument**. **Ce qui pourrait correspondre à la première methode1**(). **C'** est la seule déclarée avec un seul argument. Mais c' **est impossible**. **Puisqu'** on a utilisé la méthode avec un argument de type long. Le type long ne peut pas être converti implicitement en type int.

Ligne 29 : **On appelle la méthode avec** 2 **arguments dont le premier est de type short et le deuxième de type int**. **Logiquement**, **les deux méthodes methode1**() **déclarées avec deux arguments peuvent correspondre**.
La deuxième methode1() **peut correspondre parce que le premier argument de type short** (**s**) **utilisé peut être implicitement converti en int et le deuxième argument de type int** (**i**) **utilisé correspond au type du deuxième argument de cette deuxième methode1**().
La troisième methode1() **peut aussi correspondre parce que le premier argument de type short** (**s**) **utilisé peut être converti en type int et le deuxième argument de type int** (**i**) **utilisé peut être converti en type double**.
Quelle méthode sera effectivement choisi dans l' appel ?...
C' est la deuxième methode1() qui sera choisie. Parce que dans ce cas, on a besoin seulement d' **une conversion**. **Alors que la troisième methode1**() **entraine forcément deux conversions des arguments**.

Ligne 30 : **On appelle la méthode avec** 2 **arguments dont le premier est de type byte et le deuxième de type float**. **Ce qui correspond à la troisième methode1**(). **Le byte sera converti en int**. **Et le float en double**.

Ligne 31 : On appelle la méthode avec 2 arguments dont le premier est de type long et le deuxième de type float.
Erreur : aucune méthode ne correspond.

Exercice 10 : Champs et méthodes d' une classe partie 5 : méthodes surdéfinies

Voici la classe Surdefinitions3 avec sa méthode methode1 surdéfinie.
Dans le programme TesterSurdefinitions2 ci-dessous, on a appelé (utilisé) cette méthode 9 fois.
Dîtes dans chaque cas laquelle des version est appelée.
Quelle conversion a été mise en jeu ?

```
1.      class Surdefinitions3
2.      {
3.          public void methode1(byte b)
4.          {
5.              ...
6.          }
7.
8.          public void methode1(int i)
9.          {
10.             ...
11.         }
12.
13.         public void methode1(float f)
14.         {
15.             ...
16.         }
17.         public void methode1(double d)
18.         {
19.             ...
20.         }
21.     }
22.
23.     public class TesterSurdefinitions3
24.     {
25.         public static void main(String [] args)
26.         {
27.             Surdefinitions3 sf = new Surdefinitions3();
28.             byte b; short s; int i; long l; float f; double d;
29.
30.             sf.methode1(b);
31.             sf.methode1(s);
32.             sf.methode1(i);
33.             sf.methode1(l);
34.             sf.methode1(f);
35.             sf.methode1(d);
36.             sf.methode1(2. * f);
37.             sf.methode1(b + 1);
38.             sf.methode1(b++);
39.         }
40.     }
```

N' essayez pas de compiler. Encore moins d' exécuter. Le programme n' est pas correct. L' exercice est purement théorique. Les ... ne sont pas acceptés dans un programme java.

Ligne 30 : on appelle la methode1() avec un argument de type byte. C' est donc la première version de la méthode methode1() qui est utilisée. Aucune conversion d' argument n' est nécessaire.

Ligne 31 : on appelle la methode1() avec un argument de type short. C' est la deuxième version de la méthode methode1() qui est utilisée. L' argument de type short utilisée est convertie en int.

Ligne 32 : on appelle la methode1() avec un argument de type int. C' est la deuxième version de la méthode methode1() qui est utilisée. Aucune conversion n' est nécessaire.

667
668 ***Ligne** 33 : **on appelle la methode1() avec un argument de type long. C'** est la troisième version de la méthode methode1() qui est utilisée. L' **argument de type long utilisée est convertie en float.***
669
670 ***Ligne** 34 : **on appelle la methode1() avec un argument de type float. C'** est la troisième version de la méthode methode1() qui est utilisée. Aucune conversion n' **est nécessaire.***
671
672 ***Ligne** 35 : **on appelle la methode1() avec un argument de type double. C'** est la quatrième version de la méthode methode1() qui est utilisée. Aucune conversion n' **est nécessaire.***
673
674 ***Ligne** 36 : **on appelle la methode1() avec un argument de type double. En effet, la multiplication d'** un float (f) par un double (2.) donne un double. C' **est la quatrième version de la méthode methode1() qui est utilisée. Aucune conversion n'** est nécessaire.
675 ***Rappel** : **un nombre entier qui n'** est pas dans une variable est considéré par défaut comme un int. Et un nombre réel qui n' **est pas dans une variable est considéré par défaut comme un double. 2. est donc un double par défaut.***
676
677 ***Ligne** 37 : **on appelle la methode1() avec un argument de type int. C'** est la deuxième version de la méthode methode1() qui est utilisée. Aucune conversion n' **est nécessaire.***
678 ***Rappel** : **addition d'** un byte et d' **un int est un int. 1 est un int par défaut.***
679
680 ***Ligne** 38 : **on appelle la methode1() avec un argument de type byte. C'** est donc la première version de la méthode methode1() qui est utilisée. Aucune conversion d' **argument n'** est nécessaire.
681 ***un type incrémenté est un type byte. b = byte. DOnc b++ est de type byte.***
682
683 ***Exercice** 11 : **Champs et méthodes d'** une classe partie 5 : méthodes surdéfinies (fin)*
684
685 ***Avant de compiler puis exécuter ce programme, prévoyez la méthode utilisée***
686 ***dans les** 3 **appels en expliquant avec les conversions éventuelles.***
687

```
688  1.      class Surdefinitions4
689  2.      {
690  3.          public void methode1(int a, float b)
691  4.          {
692  5.              System.out.println("methode de signature (int, float)" );
693  6.          }
694  7.
695  8.          public void methode1(float a, float b)
696  9.          {
697  10.             System.out.println("methode de signature (float, float)" );
698  11.         }
699  12.
700  13.         public void methode1(float a, int b)
701  14.         {
702  15.             System.out.println("methode de signature (float, int)" );
703  16.         }
704  17.     }
705  18.
706  19.     public class TesterSurdefinitions4
707  20.     {
708  21.         public static void main (String [] args)
709  22.         {
710  23.             short s =2; int i1 = 21, i2 = 22; float f = 23.09;
711  24.             Surdefinitions4 sf = new Surdefinitions4();
712  25.             sf.methode1(i1, f);
713  26.             sf.methode1(f, i1);
714  27.             sf.methode1(s, f);
715  28.
716  29.         }
717  30.     }
```

718 ***Appel de la ligne** 25. **On utilise la méthode methode1() avec** 2 **arguments. Le premier de type int (i1) et le deuxième, de type float(f).***

719 *Les deux premières méthodes methode1() conviennent.*
720
721 *Dans la première, le premier argument utilisé correspond au premier argument de la méthode :int i1 et int n. Le deuxième argument utilisé correpond aussi au deuxième argument de la méthode : float f et flat b.*
722 *Dans la deuxième, le premier argument utilisé (i1) peut être converti en float comme le premier argument de la méthode (float a). Le deuxième argument utilisé (f) correspond au deuxième argument de la méthode (float b).*
723 *Laquelle sera choisie par le compilateur ? ... Naturellement celle qui ne demande aucune conversion. Donc la première*
724
725 *Appel de ligne 26. On utilise la méthode methode1() avec 2 arguments. Le premier de type float (f1) et le deuxième, de type float(float f2).*
726 *Les deuxième et troisième méthodes methode1() conviennent.*
727
728 *Dans la deuxième, le premier argument utilisé correspond à celui de la méthode : float f et float a. Le deuxième argument de type int (i1) peut être converti en float (float b)*
729 *Dans la troisième, le premier argument utilisé correspond au premier argument de la méthode :i float f et float a. Le deuxième argument utilisé correpond aussi au deuxième argument de la méthode : int i1 et int b.*
730 *Laquelle sera choisie par le compilateur ? ... Naturellement celle qui ne demande aucune conversion. Donc la troisième*
731
732 *Appel de la ligne 27. On utilise la méthode methode1() avec 2 arguments. Le premier de type short et le deuxième de type float.*
733 *Les deux premières conviennent.*
734
735 *Dans la première méthode, le premier argument utilisé (short) sera converti en int. Le deuxième argument utilisé (float) n' a pas besoin d' être converti.*
736 *Dans la deuxième méthode, le premier argument (short) sera converti en float. Le deuxième argument utilisé (float) n' a pas besoin d' être converti.*
737 *Laquelle convient ? ... La première. Parce que convertir short en int consomme moins de mémoire que convertir short en float.*
738
739 *Ces exercices sur la surdéfinition n' ont qu' un seul but : vous montrer qu' il ne suffit pas de surdéfinir une méthode. Encore faut-il savoir laquelle sera appelée lors de l' exécution du programme dans lequel elles seront utilisées.*
740
741 *Exercice 12 :*
742 *Dans le programme ci-dessus, effacez les 3 instructions : lignes 25, 26 et 27. Puis y mettre l' instruction suivante :*
743 *sf.methode1(i1, i2);*
744 *Compilez et une erreur s' affichera. Expliquez et commentez.*
745 *Ref : c' est une erreur déjà survenue dans l' un des exercices ci-dessus.*
746
747 *Ces exercices sur la surdéfinition n' ont qu' un seul but : vous montrer qu' il ne suffit pas de surdéfinir une méthode. Encore faut-il savoir laquelle sera appelée lors de l' exécution du programme dans lequel elles seront utilisées.*
748
749 *Exercice 13 : Retour sur les constructeurs*
750 *Quelle erreur a été comise dans la conception de cette classe ?*
751

```
752  1.    public class CompteBancaire
753  2.    {
754  3.        private String nom;
755  4.        private String prenom;
756  5.        private String adresse;
757  6.        private int montantInitial;
758  7.        private String extraitKBis;
759  8.        private String tel;
760  9.
761  10.       public CompteBancaire(String nom, String prenom, String adresse, int
montantInitial)
762  11.       {
763  12.           this.nom = nom;
```

```
13.            this.prenom = prenom;
14.            this.adresse = adresse;
15.            this.montantInitial = montantInitial;
16.        }
17.
18.        public CompteBancaire(String nom, String prenom, String adresse, int
montantInitial, String extraitKBis, String tel)
19.        {
20.            this.extraitKBis = extraitKBis;
21.            this(nom, prenom, adresse, montantInitial);
22.            this.tel = tel;
23.        }
24.
25.    }
```

Lorsqu' un constructeur fait appel à un autre constructeur, Cette intruction est forcément la première. Donc, la ligne 21 devient la ligne 20 et vice versa.

Exercice 14 : ***emploi du mot clé this***

Créez une classe Points2 qui permet de manipuler des points dans le plan (espace dimension 2). ***N'*** y mettez pas de constructeur. Y mettre une méthode qui retourne la distance entre 2 points. Cette méthode sera en double exemplaire : version static et version non static. Puis y mettre une autre méthode qui, étant donné 2 points, retourne le point le plus éloigne de l' ***origine***. L' origine ayant pour coordonnées 0, 0. Cette méthode sera déclinée elle aussi en 2 exemplaire. Version static et version non static.
Au boulot.

Correction

Exercice 15 : ***initialisation d'*** un objet au clavier Créez une classe CompteBancaire avec un seul constructeur. Mais l' ***initialisation des champs se fera par saisie des valeurs des champs au clavier***. ***Puis un exemple de programme qui appelle ce constructeur***.

```
1.    class CompteBancaire
2.    {
3.        private String nom;
4.        private String prenom;
5.        private String adresse;
6.        private int montantInitial;
7.        private String extraitKBis;
8.        private String tel;
9.
10.       public CompteBancaire()
11.       {
12.           System.out.println("Vous allez créer un nouveau compte bancaire");
13.           System.out.print("Entrez le nom du titulaire : ");
14.           this.nom = Lire.chaine();
15.           System.out.print("\nEntrez le prénom du titulaire : ");
16.           this.prenom = Lire.chaine();
17.           System.out.print("\nEntrez l' adresse du titulaire : ");
18.           this.adresse = Lire.chaine();
19.           System.out.print("\nEntrez le montant initial du titulaire : ");
20.           this.montantInitial = Lire.entierInt();
21.           System.out.print("\nLa création du compte a été un succès ");
22.       }
23.
24.   }
25.
26.    public class TesterConstructeurClavier
27.    {
28.        public static void main (String [] args)
29.        {
30.            CompteBancaire c = new CompteBancaire();
31.        }
```

```
32.     }
COmpilez puis exécutez ce programme, il sera affiché :

Vous allez créer un nouveau compte bancaire
Entrez le nom du titulaire : _

Puis le curseur se met à clignoter, attendant que vous entriez le nom.

Entrez le prénom du titulaire : _

Puis le curseur se met à clignoter, attendant que vous entriez le prénom.

Entrez l' adresse du titulaire :_

Puis le curseur se met à clignoter, attendant que vous entriez l' adresse.

Entrez le montant initial du titulaire : _

Puis le curseur se met à clignoter, attendant que vous entriez le montant initial.

La création du compte a été un succès

Comme vous le savez déjà, l' appel d' un constructeur (ici, ligne 30) entraine l'
exécution des instructions de ce constructeur. C' est ce qui se passe ici.

```