

La programmation orientée objet en Java

Partie 1 Les bases

CONTENU

Structure de base	2
Les constructeurs	3
Accesseurs et Mutateurs	8
Les variables de classes	14
Le principe d'encapsulation	16

STRUCTURE DE BASE

Les exemples de codes de ce cours utiliseront le diagramme de classe suivant :



Une classe peut être comparée à un moule qui, lorsque nous le remplissons, nous donne un objet ayant la forme du moule ainsi que toutes ses caractéristiques.

Pour créer une classe allez dans **File > New > Class** ou utilisez le raccourci dans la barre d'outils, comme sur la figure suivante :



Nommez votre classe « Ville » (avec un « V » majuscule, convention de nommage oblige).

Vous devriez avoir le rendu de la figure suivante :

```

Ville.java
package cours_ville;
public class Ville {
}
  
```

La classe **Ville** est précédée du mot clé **public**, qui correspond à la portée de la classe. En programmation, la portée détermine qui peut faire appel à une classe, une méthode ou une variable. Vous avez déjà rencontré la portée **public** : cela signifie que tout le monde peut faire appel à l'élément. Ici dans le cas qui nous intéresse il s'agit d'une méthode. Une méthode marquée comme **public** peut donc être appelée depuis n'importe quel endroit du programme.

Nous allons ici utiliser une autre portée : `private`. Elle signifie que notre méthode ne pourra être appelée que depuis l'intérieur de la classe dans laquelle elle se trouve ! Les méthodes déclarées `private` correspondent souvent à des mécanismes internes à une classe que les développeurs souhaitent « cacher » ou simplement ne pas rendre accessibles de l'extérieur de la classe...

Il en va de même pour les variables. Nous allons voir que nous pouvons protéger des variables grâce au mot clé `private`. Le principe sera le même que pour les méthodes. Ces variables ne seront alors accessibles que dans la classe où elles seront nées...

LES CONSTRUCTEURS

Vu que notre objectif dans ce cours est de construire un objet `Ville`, il va falloir définir les données qu'on va lui attribuer. Nous dirons qu'un objet `Ville` possède :

- Un nom, sous la forme d'une chaîne de caractères ;
- Un nombre d'habitants, sous la forme d'un entier ;
- Un pays apparenté, sous la forme d'une chaîne de caractères.

Nous allons faire ceci en mettant des variables d'instance (de simples variables identiques à celles que vous manipulez habituellement) dans notre classe. Celle-ci va contenir une variable dont le rôle sera de stocker le nom, une autre stockera le nombre d'habitants et la dernière se chargera du pays ! Voici à quoi ressemble notre classe `Ville` à présent :

```

1 package cours_ville;
2
3 public class Ville {
4
5     String nomVille;
6     String nomPays;
7     int nbHabitants;
8
9 }
10

```

Contrairement aux classes, les variables d'instance présentes dans une classe sont `public` si vous ne leur spécifiez pas de portée. Alors, on parle de variable d'instance, parce que dans nos futures classes Java qui définiront des objets, il y aura plusieurs types de variables (nous approfondirons ceci dans ce cours). Pour le moment, sachez qu'il y a trois grands types de variables dans une classe objet :

- Les variables d'instance : ce sont elles qui définiront les caractéristiques de notre objet.
- Les variables de classe : celles-ci sont communes à toutes les instances de votre classe.
- Les variables locales : ce sont des variables que nous utiliserons pour travailler dans notre objet.

Dans l'immédiat, nous allons travailler avec des variables d'instance afin de créer des objets différents. Il ne nous reste plus qu'à créer notre premier objet, pour ce faire, nous allons devoir utiliser ce qu'on appelle des constructeurs.

Un constructeur est une méthode d'instance qui va se charger de créer un objet et, le cas échéant, d'initialiser ses variables de classe ! Cette méthode a pour rôle de signaler à la JVM (Java Virtual Machine) qu'il faut réserver de la mémoire pour notre futur objet et donc, par extension, d'en réserver pour toutes ses variables.

Notre premier constructeur sera ce qu'on appelle communément un constructeur par défaut, c'est-à-dire qu'il ne prendra aucun paramètre, mais permettra tout de même d'instancier un objet, et vu que nous sommes perfectionnistes, nous allons y initialiser nos variables d'instance. Voici votre premier constructeur :

```

3  public class Ville {
4
5      // stocke le nom de la ville
6      String nomVille;
7      // stocke le nom du pays de la ville
8      String nomPays;
9      // stocke le nombre d'habitants de la ville
10     int nbHabitants;
11
12     // constructeur par défaut
13     public Ville()
14     {
15         System.out.println("Création d'une ville par défaut");
16         nomVille = "inconnu";
17         nomPays = "inconnu";
18         nbHabitants = 0;
19     }
20 }
21

```

Vous avez remarqué que le constructeur est en fait une méthode qui n'a aucun type de retour (void, double...) et qui porte le même nom que notre classe ! Ceci est une règle immuable : le (les) constructeur(s) d'une classe doit (doivent) porter le même nom que la classe !

Son corollaire est qu'un objet peut avoir plusieurs constructeurs. Il s'agit de la même méthode, mais surchargée ! Dans notre premier constructeur, nous n'avons passé aucun paramètre, mais nous allons bientôt en mettre.

Vous pouvez d'ores et déjà créer une instance de Ville. Cependant, commencez par vous rappeler qu'une instance d'objet se fait grâce au mot clé new, comme lorsque vous créez une variable de type String.

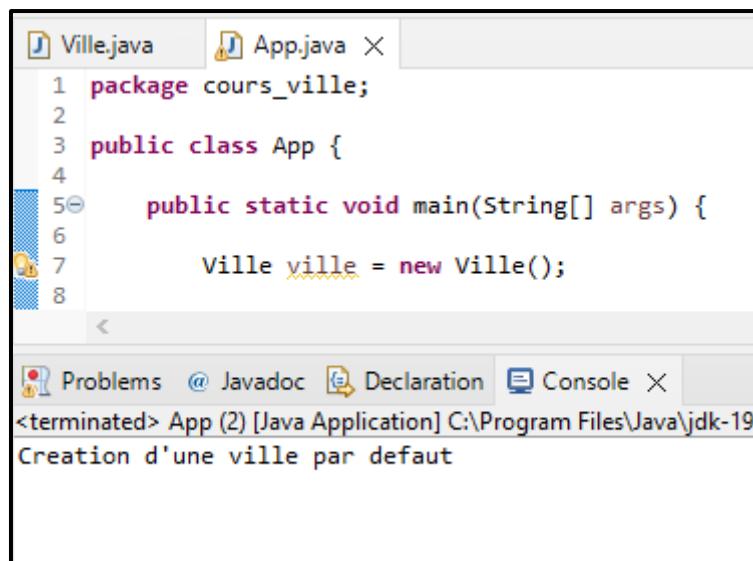
Maintenant, vu que nous allons créer des objets Ville, nous allons procéder comme avec les String. Vérifions que l'instanciation s'effectue comme il faut. Allons dans notre classe contenant la méthode main et instancions un objet Ville. Je suppose que vous avez deviné que le type de notre objet sera Ville !

```

3  public class App {
4
5      public static void main(String[] args) {
6
7          Ville ville = new Ville();
8
9      }
10 }
11

```

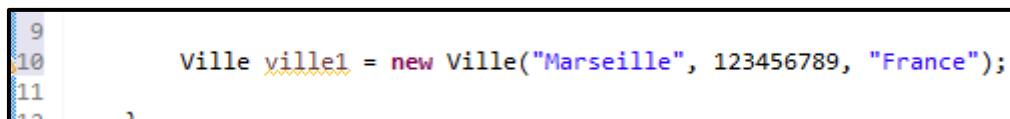
Exécutez ce code, vous devriez avoir l'équivalent de la figure suivante sous les yeux :



```
Ville.java  App.java X
1 package cours_ville;
2
3 public class App {
4
5     public static void main(String[] args) {
6
7         Ville ville = new Ville();
8     }
9
10
11
12 }
```

Problems @ Javadoc Declaration Console X
<terminated> App (2) [Java Application] C:\Program Files\Java\jdk-19
Creation d'une ville par defaut

Maintenant, nous devons mettre des données dans notre objet, ceci afin de pouvoir commencer à travailler... Le but sera de parvenir à une déclaration d'objet se faisant comme ceci :



```
9
10     Ville ville1 = new Ville("Marseille", 123456789, "France");
11
12 }
```

Vous avez remarqué qu'ici, les paramètres sont renseignés : eh bien il suffit de créer une méthode qui récupère ces paramètres et initialise les variables de notre objet, ce qui achèvera notre constructeur d'initialisation.

Voici le constructeur de notre objet `Ville`, celui qui permet d'avoir des objets avec des paramètres différents :

```

3  public class Ville {
4
5      // stocke le nom de la ville
6      String nomVille;
7      // stocke le nom du pays de la ville
8      String nomPays;
9      // stock le nombre d'habitants de la ville
10     int nbHabitants;
11
12     // constructeur par défaut
13     public Ville()
14     {
15         System.out.println("Creation d'une ville par defaut");
16         nomVille = "inconnu";
17         nomPays = "inconnu";
18         nbHabitants = 0;
19     }
20
21     // constructeur avec paramètres
22     // J'ai ajouté un << p >> en 1ère lettre des paramètres
23     // ce n'est pas une convention, mais ça peut être un bon moyen de les repérer.
24     public Ville(String pNom, int nbre, String pNomPays)
25     {
26         System.out.println("Creation d'une ville avec des parametres");
27         nomVille = pNom;
28         nomPays = pNomPays;
29         nbHabitants = nbre;
30     }
31 }
32
33

```

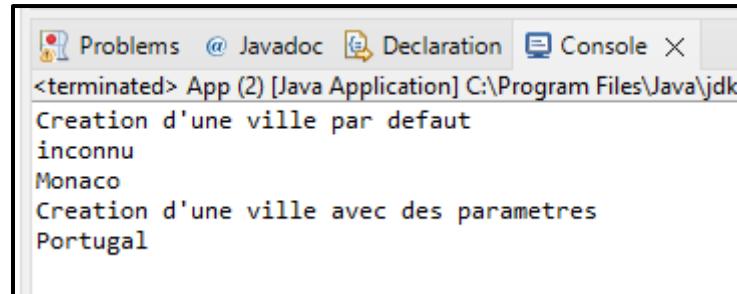
Dans ce cas, l'exemple de déclaration et d'initialisation d'un objet `Ville` que je vous ai montré un peu plus haut fonctionne sans aucun souci ! Mais il vous faudra respecter scrupuleusement l'ordre des paramètres passés lors de l'initialisation de votre objet : sinon, c'est l'erreur de compilation à coup sûr !

Cependant, notre objet présente un gros défaut : les variables d'instance qui le caractérisent sont accessibles dans votre classe contenant votre `main` ! Ceci implique que vous pouvez directement modifier les attributs de la classe. Testez ce code et vous verrez que le résultat est identique à la figure suivante :

```

public static void main(String[] args) {
    Ville ville = new Ville();
    System.out.println(ville.nomVille);
    ville.nomVille = "Monaco";
    System.out.println(ville.nomVille);
    Ville ville1 = new Ville("Marseille", 123456789, "France");
    ville1.nomPays = "Portugal";
    System.out.println(ville1.nomPays);
}

```



```

Problems @ Javadoc Declaration Console
<terminated> App (2) [Java Application] C:\Program Files\Java\jdk
Creation d'une ville par defaut
inconnu
Monaco
Creation d'une ville avec des parametres
Portugal

```

Vous constatez que nous pouvons accéder aux variables d'instance en utilisant le « . », comme lorsque vous appelez la méthode `subString()` de l'objet `String`. C'est très risqué, et la plupart des programmeurs Java vous le diront. Dans la majorité des cas, nous allons contrôler les modifications des variables de classe, de manière qu'un code extérieur ne fasse pas n'importe quoi avec nos objets ! En plus de ça, imaginez que vous souhaitiez faire quelque chose à chaque fois qu'une valeur change ; si vous ne protégez pas vos données, ce sera impossible à réaliser... C'est pour cela que nous protégeons nos variables d'instance en les déclarant `private`, comme ceci :

```
public class Ville {  
  
    private String nomVille;  
    private String nomPays;  
    private int nbHabitants;
```

Désormais, ces attributs ne sont plus accessibles en dehors de la classe où ils sont déclarés ! Nous allons maintenant voir comment accéder tout de même à nos données.

ACCESSEURS ET MUTATEURS

Un accesseur est une méthode qui va nous permettre d'accéder aux variables de nos objets en lecture, et un mutateur nous permettra d'en faire de même en écriture ! Grâce aux accesseurs, vous pourrez afficher les variables de vos objets, et grâce aux mutateurs, vous pourrez les modifier :

```
***** ACCESSEURS *****

// retourne le nom de la ville
public String getNom()
{
    return nomVille;
}

// Retourne le nom du pays
public String getNomPays()
{
    return nomPays;
}

// Retourne le nombre d'habitants
public int getNbHabitants()
{
    return nbHabitants;
}

***** MUTATEURS *****

// Définit le nom de la ville
public void setNom(String pNom)
{
    nomVille = pNom;
}

// Définit le nom du pays
public void setNomPays(String pNomPays)
{
    nomPays = pNomPays;
}

// Définit le nombre d'habitants
public void setNbHabitants(int nbre)
{
    nbHabitants = nbre;
}
```

Nos accesseurs sont bien des méthodes, et elles sont `public` pour que vous puissiez y accéder depuis une autre classe que celle-ci : depuis le `main`, par exemple. Les accesseurs sont du même type que la variable qu'ils doivent retourner. Les mutateurs sont, cependant, de type `void`. Ce mot clé signifie « rien » ; en effet, ces méthodes ne retournent aucune valeur, elles se contentent de les mettre à jour.

Je vous ai fait faire la différence entre accesseurs et mutateurs, mais généralement, lorsqu'on parle d'accesseurs, ce terme inclut également les mutateurs. Autre chose : il s'agit ici d'une question de convention de nommage. Les accesseurs commencent par `get` et les mutateurs par `set`, comme vous pouvez le voir ici. On parle d'ailleurs parfois de Getters et de Setters.

À présent, essayez ce code dans votre méthode `main` :

```
public static void main(String[] args) {
    Ville ville = new Ville();
    Ville ville1 = new Ville("Marseille", 870321, "France");
    Ville ville2 = new Ville("Mulhouse", 108038, "France");

    System.out.println("\n ville = "+ville.getNom()+" ville de "+ville.getNbHabitants()
                      +" habitant(s) se situant en "+ville.getNomPays());
    System.out.println(" ville1 = "+ville1.getNom()+" ville de "+ville1.getNbHabitants()
                      +" habitant(s) se situant en "+ville1.getNomPays());
    System.out.println(" ville2 = "+ville2.getNom()+" ville de "+ville2.getNbHabitants()
                      +" habitant(s) se situant en "+ville2.getNomPays()+"\n\n");
    /*
    Nous allons interchanger les villes 1 et 2
    tout ça par l'intermédiaire d'une autre objet ville
    */
    Ville temp = new Ville();
    temp = ville1;
    ville1 = ville2;
    ville2 = temp;
    System.out.println(" ville1 = "+ville1.getNom()+" ville de "+ville1.getNbHabitants()
                      +" habitant(s) se situant en "+ville1.getNomPays());
    System.out.println(" ville2 = "+ville2.getNom()+" ville de "+ville2.getNbHabitants()
                      +" habitant(s) se situant en "+ville2.getNomPays()+"\n\n");
    /*
    Nous allons maintenant interchanger leur noms
    par le biais de leurs mutateurs
    */
    ville1.setNom("Honk Kong");
    ville2.setNom("New York");
    System.out.println(" ville1 = "+ville1.getNom()+" ville de "+ville1.getNbHabitants()
                      +" habitant(s) se situant en "+ville1.getNomPays());
    System.out.println(" ville2 = "+ville2.getNom()+" ville de "+ville2.getNbHabitants()
                      +" habitant(s) se situant en "+ville2.getNomPays()+"\n\n");
}
```

À la compilation, vous devriez obtenir la figure suivante :

```
Creation d'une ville par defaut
Creation d'une ville avec des parametres
Creation d'une ville avec des parametres

ville = inconnu ville de 0 habitant(s) se situant en inconnu
ville1 = Marseille ville de 870321 habitant(s) se situant en France
ville2 = Mulhouse ville de 108038 habitant(s) se situant en France

Creation d'une ville par defaut
ville1 = Mulhouse ville de 108038 habitant(s) se situant en France
ville2 = Marseille ville de 870321 habitant(s) se situant en France

ville1 = Honk Kong ville de 108038 habitant(s) se situant en France
ville2 = New York ville de 870321 habitant(s) se situant en France
```

Vous voyez bien que les constructeurs ont fonctionné, que les accesseurs tournent à merveille et que vous pouvez commencer à travailler avec vos objets Ville. Cependant, pour afficher le contenu, on pourrait faire plus simple, comme par exemple créer une méthode qui se chargerait de faire tout ceci... Je sais ce que vous vous dites : « Mais les accesseurs, ce ne sont pas des méthodes ? ». Bien sûr que si, mais il vaut mieux bien distinguer les différents types de méthodes dans un objet :

- Les constructeurs -> méthodes servant à créer des objets ;
- Les accesseurs -> méthodes servant à accéder aux données des objets ;
- Les méthodes d'instance → méthodes servant à la gestion des objets.

Avec nos objets Ville, notre choix est un peu limité par le nombre de méthodes possibles, mais nous pouvons tout de même en faire une ou deux pour l'exemple :

- Faire un système de catégories de villes par rapport à leur nombre d'habitants (<1000 -> A, <10 000 -> B...). Ceci est déterminé à la construction ou à la redéfinition du nombre d'habitants : ajoutons donc une variable d'instance de type char à notre classe et appelons-la categorie. Pensez à ajouter le traitement aux bons endroits ;
- Faire une méthode de description de notre objet Ville ;
- Une méthode pour comparer deux objets par rapport à leur nombre d'habitants.

Nous voulons que la classe Ville gère la façon de déterminer la catégorie elle-même, et non que cette action puisse être opérée de l'extérieur. La méthode qui fera ceci sera donc déclarée private.

Cependant, un problème va se poser ! Vous savez déjà qu'en Java, on appelle les méthodes d'un objet comme ceci : monString.substring(0,4);. Cependant, vu qu'il va falloir qu'on travaille depuis l'intérieur de notre objet, vous allez encore avoir un mot clé à retenir... Cette fois, il s'agit du mot clé this. Voici tout d'abord le code de notre classe Ville en entier, c'est-à-dire comportant les méthodes dont on vient de parler :

```

1 package cours_ville;
2
3 public class Ville {
4
5
6     private String nomVille;
7     private String nomPays;
8     private int nbHabitants;
9     private char categorie;
10
11
12     // constructeur par défaut
13     public Ville()
14     {
15         System.out.println("Creation d'une ville par defaut");
16         nomVille = "inconnu";
17         nomPays = "inconnu";
18         nbHabitants = 0;
19         this.setcategorie();
20     }
21
22
23     // constructeur avec paramètres
24     public Ville(String pNom, int nbre, String pNomPays)
25     {
26         System.out.println("Creation d'une ville avec des paramètres");
27         nomVille = pNom;
28         nomPays = pNomPays;
29         nbHabitants = nbre;
30         this.setcategorie();
31     }

```

```
32
33     //***** ACCESSEURS *****
34
35     // retourne le nom de la ville
36     public String getNom()
37     {
38         return nomVille;
39     }
40
41     // Retourne le nom du pays
42     public String getNomPays()
43     {
44         return nomPays;
45     }
46
47     // Retourne le nombre d'habitant
48     public int getNbHabitants()
49     {
50         return nbHabitants;
51     }
52
53     // Retourne la catégorie
54     public char getCategories()
55     {
56         return categories;
57     }
58
59     //***** MUTATEURS *****
60
61     // Définit le nom de la ville
62     public void setNom(String pNom)
63     {
64         nomVille = pNom;
65     }
66
67     // Définit le nom du pays
68     public void setNomPays(String pNomPays)
69     {
70         nomPays = pNomPays;
71     }
72
73     // Définit le nombre d'habitant
74     public void setNbHabitants(int nbre)
75     {
76         nbHabitants = nbre;
77         this.setcategorie();
78     }
79
```

```

80 // Définit la catégorie de la ville
81 private void setcategorie()
82 {
83     int bornesSuperieurs [] = {0, 1000, 10000, 100000, 500000, 1000000, 5000000, 10000000};
84     char categories [] = {'?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};
85
86     int i = 0;
87
88     while(i < bornesSuperieurs.length && this.nbHabitants > bornesSuperieurs[i])
89     {
90         i++;
91     }
92     this.categories = categories[i];
93 }
94
95 // Retourne la description de la ville
96 public String decrisToi()
97 {
98     return "\t"+this.nomVille+" est une ville de "
99             +this.nomPays
100            +" elle comporte : "
101            +this.nbHabitants
102            +" habitant(s) => elle est donc de catégorie : "
103            +this.categories;
104 }
105
106 // Retourne une chaîne de caractères selon le résultat de la comparaison
107 public String comparer(Ville v1)
108 {
109     String str = new String();
110
111     if(v1.getNbHabitants() > this.nbHabitants)
112     {
113         str = v1.getNom()+" est plus peuplée que "+this.nomVille;
114     }
115     else
116     {
117         str = this.nomVille+" est plus peuplée que "+v1.getNom();
118     }
119
120     return str;
121 }
122

```

Pour simplifier, `this` fait référence à l'objet courant ! Bien que la traduction anglaise exacte soit « *ceci* », il faut comprendre « *moi* ». À l'intérieur d'un objet, ce mot clé permet de désigner une de ses variables ou une de ses méthodes.

Pour expliciter le fonctionnement du mot clé `this`, prenons l'exemple de la méthode `comparer(Ville V1)`. La méthode va s'utiliser comme suit :

```

Ville ville1 = new Ville("Marseille", 870321, "France");
Ville ville2 = new Ville("Mulhouse", 108038, "France");

ville1.comparer(ville2);

```

Dans cette méthode, nous voulons comparer le nombre d'habitants de chacun des deux objets `Ville`. Pour accéder à la variable `nbHabitants` de l'objet `ville2`, il suffit d'utiliser la syntaxe `ville2.getNbHabitants()` ; nous ferons donc référence à la propriété `nbHabitants` de l'objet `ville2`. Mais l'objet `ville1`, lui, est l'objet appelant de cette méthode. Pour se servir de ses propres variables, on utilise alors `this.nbHabitants`, ce qui a pour effet de faire appel à la variable `nbHabitants` de l'objet exécutant la méthode `comparer(Ville v1)`. Explicitons un peu les trois méthodes qui ont été décrites précédemment.

La méthode `setCategorie()`

Elle ne prend aucun paramètre, et ne renvoie rien : elle se contente de mettre la variable de classe `categories` à jour. Elle détermine dans quelle tranche se trouve la ville grâce au nombre d'habitants de l'objet appelant, obtenu au moyen du mot clé `this`. Selon le nombre d'habitants, le caractère renvoyé changera. Nous l'appelons lorsque nous construisons un objet `Ville` (que ce soit avec ou sans paramètre), mais aussi lorsque nous redéfinissons le nombre d'habitants : de cette manière, la catégorie est automatiquement mise à jour, sans qu'on ait besoin de faire appel à la méthode.

La méthode `decrisToi()`

Celle-ci nous renvoie un objet de type `String`. Elle fait référence aux variables qui composent l'objet appelant la méthode, toujours grâce à `this`, et nous renvoie donc une chaîne de caractères qui nous décrit l'objet en énumérant ses composants.

La méthode `comparer(Ville V1)`

Elle prend une ville en paramètre, pour pouvoir comparer les variables `nbHabitants` de l'objet appelant la méthode et de celui passé en paramètre pour nous dire quelle ville est la plus peuplée ! Et si nous faisions un petit test ?

```
Ville ville = new Ville();
Ville ville1 = new Ville("Marseille", 870321, "France");
Ville ville2 = new Ville("Mulhouse", 108038, "France");

System.out.println("\n\n"+ville1.decrisToi());
System.out.println(ville.decrisToi());
System.out.println(ville2.decrisToi()+"\n\n");
System.out.println(ville1.comparer(ville2));
```

Ce qui devrait donner le résultat de la figure suivante :

```
Marseille est une ville de France elle comporte : 870321 habitant(s) => elle est donc de categorie : E
inconnu est une ville de inconnu elle comporte : 0 habitant(s) => elle est donc de categorie : ?
Mulhouse est une ville de France elle comporte : 108038 habitant(s) => elle est donc de categorie : D
```

```
Marseille est plus peuplee que Mulhouse
```

LES VARIABLES DE CLASSES

Il y a plusieurs types de variables dans une classe. Nous avons vu les variables d'instance qui forment la carte d'identité d'un objet ; maintenant, voici les variables de classe.

Celles-ci peuvent s'avérer très utiles. Dans notre exemple, nous allons compter le nombre d'instances de notre classe `Ville`, mais nous pourrions les utiliser pour bien d'autres choses (un taux de TVA dans une classe qui calcule le prix TTC, par exemple).

La particularité de ce type de variable, c'est qu'elles seront communes à toutes les instances de la classe ! Créons sans plus attendre notre compteur d'instances. Il s'agira d'une variable de type `int` que nous appellerons `nbInstance`, et qui sera `public` ; nous mettrons aussi son homologue en `private` en place et l'appellerons `nbInstanceBis` (il sera nécessaire de mettre un accesseur en place pour cette variable). Afin qu'une variable soit une variable de classe, elle doit être précédée du mot clé `static`. Cela donnerait dans notre classe `Ville` :

```
// les attributs
private String nomVille;
private String nomPays;
private int nbHabitants;
private char categories;
//variable publique qui compte les instances
public static int nbInstances = 0;
//variable privée qui compte les instances
private static int nbInstancesBis = 0;
```

```
// constructeur par défaut
public Ville()
{
    System.out.println("Creation d'une ville par defaut");
    nomVille = "inconnu";
    nomPays = "inconnu";
    nbHabitants = 0;
    this.setcategorie();
    // on incremente les variables à chaque appel du constructeur
    nbInstances++;
    nbInstancesBis++;
}

// constructeur avec parametres
public Ville(String pNom, int nbre, String pNomPays)
{
    System.out.println("Creation d'une ville avec des parametres");
    nomVille = pNom;
    nomPays = pNomPays;
    nbHabitants = nbre;
    this.setcategorie();
    // on incremente les variables à chaque appel du constructeur
    nbInstances++;
    nbInstancesBis++;
}
```

```
***** ACCESSEURS *****

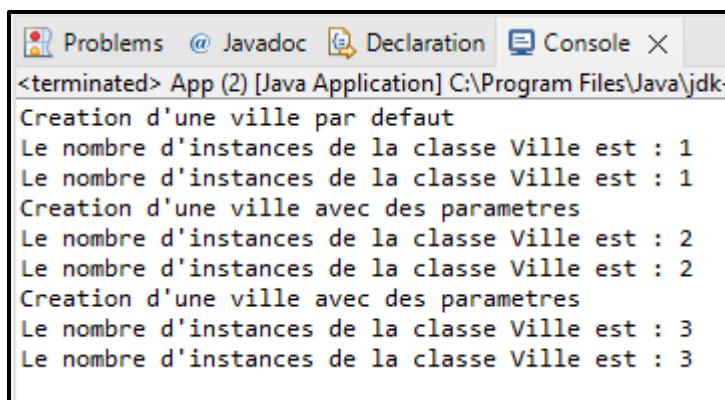
// retourne le nombre d'instance
public static int getNbInstances()
{
    return nbInstancesBis;
}
```

Vous avez dû remarquer que l'accesseur de notre variable de classe déclarée privée est aussi déclaré **static** : ceci est une règle ! Toutes les méthodes de classe n'utilisant que des variables de classe doivent être déclarées **static**. On les appelle des méthodes de classe, car il n'y en a qu'une pour toutes vos instances. Cependant ce n'est plus une méthode de classe si celle-ci utilise des variables d'instance en plus de variables de classe...

À présent, si vous testez le code suivant, vous allez constater l'utilité des variables de classe :

```
Ville ville = new Ville();
System.out.println("Le nombre d'instances de la classe Ville est : "+ville.nbInstances);
System.out.println("Le nombre d'instances de la classe Ville est : "+ville.getNbInstances());
Ville ville1 = new Ville("Marseille", 870321, "France");
System.out.println("Le nombre d'instances de la classe Ville est : "+ville1.nbInstances);
System.out.println("Le nombre d'instances de la classe Ville est : "+ville1.getNbInstances());
Ville ville2 = new Ville("Mulhouse", 108038, "France");
System.out.println("Le nombre d'instances de la classe Ville est : "+ville2.nbInstances);
System.out.println("Le nombre d'instances de la classe Ville est : "+ville2.getNbInstances());
```

Le résultat, visible à la figure suivante, montre que le nombre augmente à chaque instanciation.



```
Problems @ Javadoc Declaration Console X
<terminated> App (2) [Java Application] C:\Program Files\Java\jdk-
Creation d'une ville par defaut
Le nombre d'instances de la classe Ville est : 1
Le nombre d'instances de la classe Ville est : 1
Creation d'une ville avec des parametres
Le nombre d'instances de la classe Ville est : 2
Le nombre d'instances de la classe Ville est : 2
Creation d'une ville avec des parametres
Le nombre d'instances de la classe Ville est : 3
Le nombre d'instances de la classe Ville est : 3
```

Lorsque vous avez vu les méthodes, vous les avez déclarées **public**. Vous auriez également pu les déclarer **private**, mais attention, dans les deux cas, il faut aussi qu'elles soient **static**, car elles sont exécutées dans un contexte **static** : la méthode **main**.

LE PRINCIPE D'ENCAPSULATION

Voilà, vous venez de construire votre premier objet « maison ». Cependant, sans le savoir, vous avez fait plus que ça : vous avez créé un objet dont les variables sont protégées de l'extérieur. En effet, depuis l'extérieur de la classe, elles ne sont accessibles que via les accesseurs et mutateurs que nous avons définis. C'est le principe d'encapsulation !

En fait, lorsqu'on procède de la sorte, on s'assure que le fonctionnement interne à l'objet est intégré, car toute modification d'une donnée de l'objet est maîtrisée. Nous avons développé des méthodes qui s'assurent qu'on ne modifie pas n'importe comment les variables.

Prenons l'exemple de la variable `nbHabitants`. L'encapsuler nous permet, lors de son affectation, de déduire automatiquement la catégorie de l'objet `Ville`, chose qui n'est pas facilement faisable sans encapsulation. Par extension, si vous avez besoin d'effectuer des opérations déterminées lors de l'affectation du nom d'une ville par exemple, vous n'aurez pas à passer en revue tous les codes source utilisant l'objet `Ville` : vous n'aurez qu'à modifier l'objet (ou la méthode) en question, et le tour sera joué. Si vous demandez l'utilité de tout cela, dites-vous que vous ne serez peut-être pas seuls à développer vos logiciels, et que les personnes utilisant vos classes n'ont pas à savoir ce qu'il s'y passe : seules les fonctionnalités qui leurs sont offertes comptent. Java est souple parce qu'il offre beaucoup de fonctionnalités pouvant être retravaillées selon les besoins, mais gardez à l'esprit que certaines choses vous seront volontairement inaccessibles, pour éviter que vous ne « cassiez » quelque chose.

- Une classe permet de définir des objets. Ceux-ci ont des attributs (variables d'instance) et des méthodes (méthodes d'instance + accesseurs).
- Les objets permettent d'encapsuler du code et des données.
- Le ou les constructeurs d'une classe doivent porter le même nom que la classe et n'ont pas de type de retour.
- L'ordre des paramètres passés dans le constructeur doit être respecté.
- Il est recommandé de déclarer ses variables d'instance private, pour les protéger d'une mauvaise utilisation par le programmeur.
- On crée des accesseurs et mutateurs (méthodes getters et setters) pour permettre une modification sûre des variables d'instance.
- Dans une classe, on accède aux variables de celle-ci grâce au mot clé `this`.
- Une variable de classe est une variable devant être déclarée `static`.
- Les méthodes n'utilisant que des variables de classe doivent elles aussi être déclarées `static`.
- On instancie un nouvel objet grâce au mot clé `new`.

FIN DU DOCUMENT ---