Catégories de variables

Nos variables jusqu'à maintenant :

1. Variables d'instance (= attributs)
   - Décrivent les attributs d'un objet
2. Variables locales
   - Déclarées à l'intérieur d'une méthode
3. Paramètres
   - Pour envoyer des valeurs à une méthode
   - S'utilisent comme des variables locales

Nouveauté :

4. Variables statiques = variables de classe
   - Indiquées par le modificateur `static`
   - Ressemblent aux variables d'instance
   - Déclarées en dehors des méthodes
   - Visibles partout dans la classe
   - Héritées par les sous-classes

Le modificateur `static`

- S'utilise pour les variables et les méthodes
- Si on ajoute `static` à la déclaration d'une variable :
  - La valeur de la variable est partagée entre toutes les instances de la classe
  - Pas possible pour les variables locales
- Si on ajoute `static` à une méthode:
  - On peut appeler la méthode sans construire d'objet
  - Diverses restrictions sur le contenu de la méthode statique

Variables d'instance et de classe

- Variable d'instance:
  - Réservation d'une zone pour chaque objet construit avec `new`
  - Résultat : chaque objet a sa propre zone/valeur pour la variable d'instance
- Variable de classe (statique) :
  - Déclaration précédée par `static`
  - Réservation d'une zone lors du chargement de la classe
  - Aucune zone réservée quand un objet est construit avec `new`
  - Résultat : tous les objets se réfèrent à la même zone/valeur pour la variable de classe

Variable statique : exemple

```
class Abc {
   public static void main(String[] args) {
      A.c++;
      A v1;
      v1 = new A();
      v1.modifier();
   }
}
class A {
   int b = 1;          // variable d'instance
   static int c = 10; // variable de classe
   void modifier() {
      b++;
      c++;
   }
}
```

Pourquoi utiliser `static`?

1. Modification d'une variable d'instance :
   - La valeur change *seulement* pour l'objet actuel
2. Modification d'une variable de classe :
   - La valeur change pour *tous* les objets de la classe

A quoi sert une variable statique ?

1. Bonne raison d'utiliser une variable statique :
   - Représentation d'une valeur qui est commune à tous les objets de la classe
2. Mauvaise raison d'utiliser une variable statique :
   - Programmer de manière *non* orientée objet en Java

Valeur commune

Exercice : intégrons à une classe `Employe` le fait que 65 ans est l'âge officiel de départ à la retraite

Considérons les deux versions suivantes :

- avec une variable d'instance `ageRetraite`
- avec une variable statique `ageRetraite`

Classe `Employe1`

- Version avec une variable d'instance pour l'âge de la retraite :

```
class Employe1 {
  private String nom;
  private int ageRetraite;
  // ...
  public Employe1(String unNom,  int unAgeRetraite) {
    nom = unNom;
    ageRetraite = unAgeRetraite;
    // ... reste des initialisations
  }
  // ...
}
```

Utilisation de `Employe1`

- Version avec une variable d'instance pour l'âge officel de la retraite :

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

    Employe1[] employes = new Employe1[350];
    employes[0] = new Employe1("Albus", 65);
    employes[1] = new Employe1("Oz", 65);
    // ...

    // La modification de l'âge de la retraite
    // nécessite un parcours du tableau car chaque
    // employé a sa propre version de la variable :
    for (int i = 0; i < employes.length; ++i) {
        employes1[i].ageRetraite = 67;
    }
  }
}
```

Classe `Employe2`

- Version avec une variable *statique* pour l'âge officiel de départ à la retraite

```
class Employe2 {
  // ...
  // Seule modification,  ageRetraite devient static:
  static int ageRetraite;
  // ...
}
class Entreprise {
  public static void main (String[] args) {
  //   ...
  // Remplissage du tableau comme avant.
  // Modification de l'âge de la retraite :
  // aucun parcours du tableau nécessaire
  Employe2.ageRetraite = 67;
  employes[0].ageRetraite = 67;   // alternative
  employes[250].ageRetraite = 67; // alternative
  }
}
```

Constantes : `final` et `static`

Pour les constantes communes à toutes les instances d'une classe :

- inutile de stocker une valeur pour chaque objet de la classe
- les déclarer en `final static`

```
class Planete {
   // G = constante gravitationnelle
   // Une variable G pour chaque planète :
   // Possible
   private final double G = 6.674E-8;

   // Une variable G pour toutes les planètes :
   // BEAUCOUP MIEUX !
   private final static double G = 6.674E-8;

   // ...
}
```

Levons le voile...

Nous sommes maintenant capables de comprendre le format bizarre de certaines instructions :

- `System.out.println()` par exemple!

`System.out.println()`

Analysons `System.out.println()` :

- `System` : classe prédéfinie de Java
- `out` :
  - Variable statique de la classe `System`
  - Il doit s'agir d'un objet car suivi d'un point
- `println` : méthode de l'objet `out`

```
class System {
   //...
   static PrintStream out = new PrintStream(...);
   //...
}
class PrintStream {
   void println (...)
      {...}
    //..
}
```