

M2103 – Bases de la Programmation Orientée Objets



Java – Cours 5

Tableaux

Motivation – Exemple du calcul du niveau de chute de pluie

Ecrire l'algorithme en pseudo-code qui lit le niveau de chute de pluie pour 6 mois consécutifs, imprime à l'écran le niveau moyen semestriel ainsi que les niveaux mensuels lus chacun sur une ligne individuelle.

Calcul du niveau de chute de pluie – Solution 1

Lire niveauPluie1

Lire niveauPluie2

...

Lire niveauPluie6

Moyenne \leftarrow (niveauPluie1 + niveauPluie2 + niveauPluie3 + niveauPluie4
+ niveauPluie5 + niveauPluie6) / 6

Imprime Moyenne

Imprime niveauPluie1

Imprime niveauPluie2

...

Imprime niveauPluie6

Discussion

- Pour calculer la moyenne, nous devons lire toutes les données.
- Pour imprimer la moyenne avant d'imprimer les niveaux mensuels, nous devons sauvegarder les données
 - Six variables individuelles de même type pour les six mois
 - On ne peut pas utiliser de boucle pour les lire
- On veut :
 - Un moyen efficace de sauvegarder les données
 - Une approche *générique* pour résoudre le problème
 - si l'on considère par exemple les niveaux de pluie sur une période de 18 mois (au lieu de 6), on veut faire le minimum de modifications possibles

Motivation

- Programmes doivent souvent traiter des ensembles de données
 - Un enfant a un ensemble de jouets
 - On peut fournir un ensemble de paramètres à un programme depuis la ligne de commande
 - Une banque gère un ensemble de comptes bancaires
 - ...
- Ces ensembles peuvent être larges et croître avec le temps
- Classes 'conteneurs'
 - Contiennent des objets de n'importe quelle classe
 - Peuvent gérer des ensembles de données

Plan du Cours

- Tableaux
- Classe `ArrayList`
- Associations '1 à Plusieurs'
- Agrégation & Composition

Introduction du tableau (1)

- Un tableau maintient une collection de valeurs de même type
 - *int, double, Chien...*
- Structure de données de base
 - Permettant de sauvegarder des données similaires
 - Utilisant un seul nom (ou identifiant) pour se référer à toutes ces données
 - Utilisant des indices pour se référer à chaque élément sauvegardé individuellement

Introduction du tableau (2)

Tableau de Strings – Jours de la semaine:

nomJour :

Dim	Lu	Mar	Mer	Jeu	Ven	Sam
0	1	2	3	4	5	6

nomJour(2)

- Nom du tableau : *nomJour*
- Chaque élément a un *indice* (un entier)
- Même type d'éléments (String)

Introduction du tableau (3)

- Nombre d'éléments est appelé la taille (size) ou longueur (length)
 - Il y a un nombre fini d'éléments
- Entier entre parenthèses après le nom du tableau spécifie un élément particulier
e.g. `nomJour(5)` est le 6ème élément du tableau
- Une modification de la valeur d'un élément n'affecte pas les autres éléments

Niveau de chute de pluie – Solution Tableau

Somme = 0

Pour indice = 0 à 5

 Lire niveauPluie(indice)

 Somme = Somme + niveauPluie(indice)

Fin Pour

Moyenne = Somme / 6

Imprimer Moyenne

Pour indice = 0 à 5

 Imprimer niveauPluie(indice)

Fin Pour

Discussion

- Utilisation d'une boucle
 - Facile de modifier le nombre de mois
- Solution plus concise et efficace
 - Surtout si l'on considère un 'grand' nombre de mois

Tableaux comme collections de données (1)

- On peut déclarer chacune des variables suivantes individuellement

```
private int note1 = 12;  
private int note2 = 14;  
private int note3 = 18;
```
- Mais il est plus concis de les déclarer dans une structure unique

```
private int [ ] notes = {12, 14, 18};
```

Tableaux comme collections de données (2)

- On peut traiter chacune des variables suivantes individuellement

total = total + note1;

total = total + note2;

total = total + note3;

- Solution plus concise et élégante

for (i = 0; i < 3; i++)

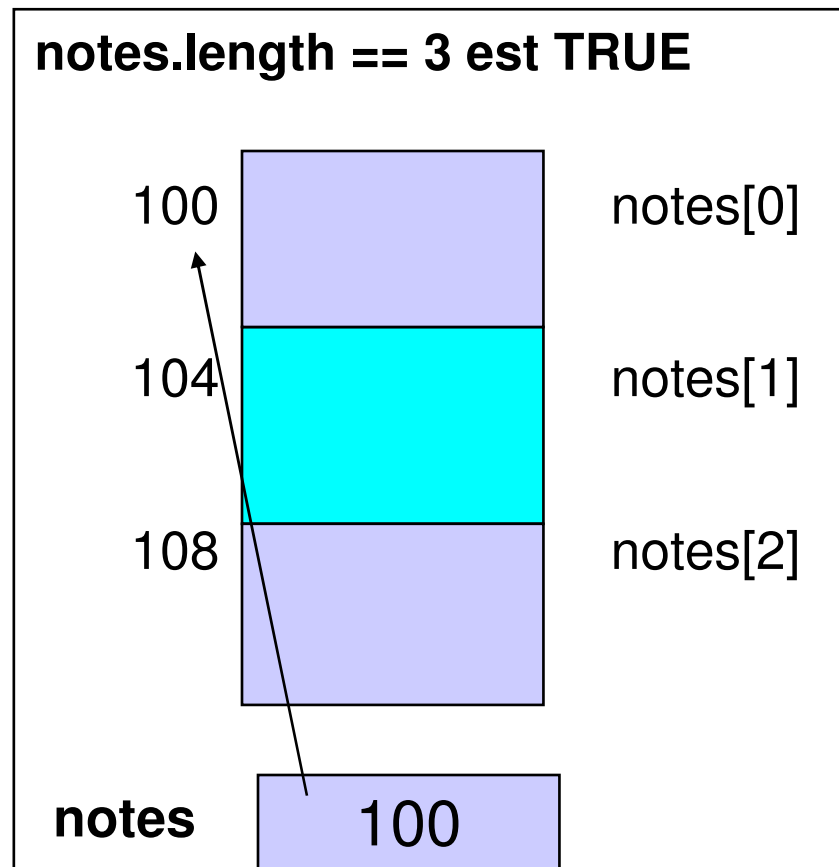
total = total + **notes[i];**

Tableaux – Définition

- Séquence finie contigüe de variables de même type de données.
- En Java, les tableaux sont encapsulés dans des objets
- Les objets Tableau ont un membre

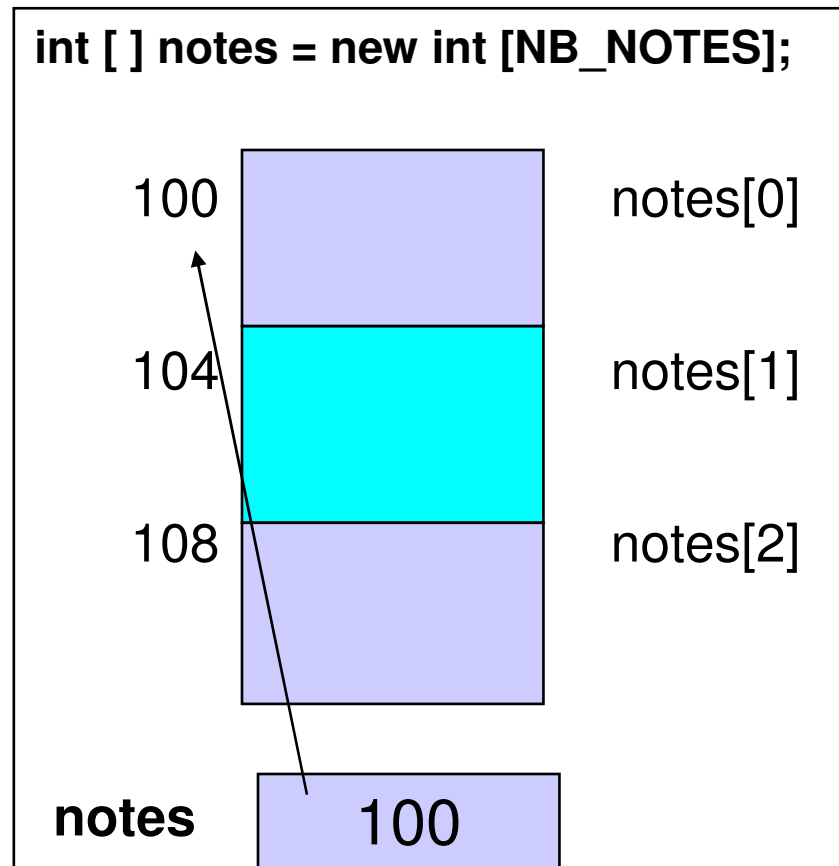
`public int length`

décrivant le nombre
d'éléments du tableau



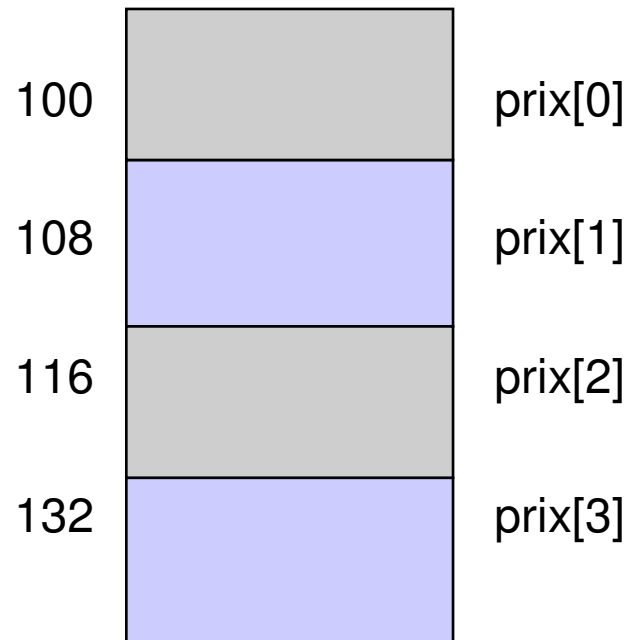
Tableaux - Déclaration

- Déclaration par le type suivi de []
`int []`
- La référence au premier élément a un nom/identifiant unique qui caractérise les autres éléments
`notes`
- Le nombre d'éléments est spécifié avec une instruction 'new'
`new int [NB_NOTES];`

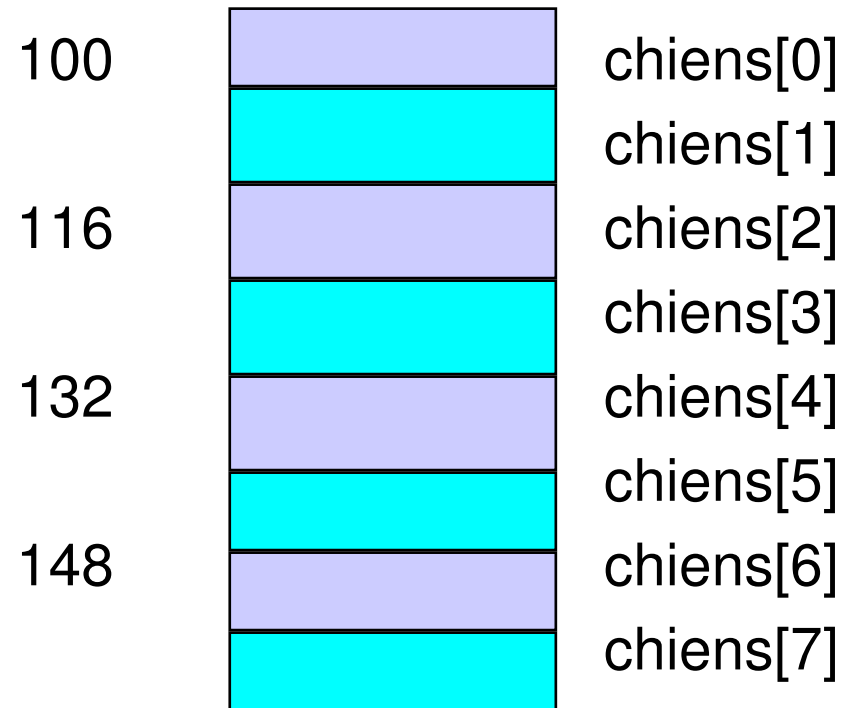


Tableaux – Exemples de déclaration

double [] prix = new double [4];

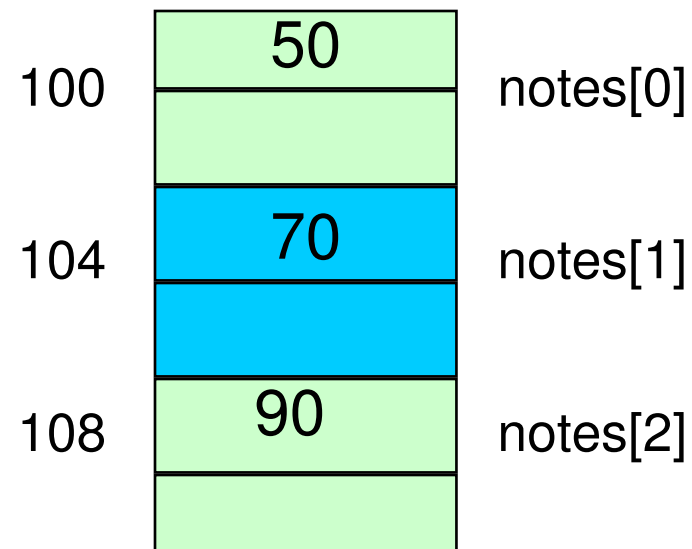


Chien [] chiens = new Chien [8];



Tableaux – Accès aux éléments

- En Java, le premier élément se trouve à l'indice 0
- Les éléments du tableau sont caractérisés par :
 - Le nom du tableau
`notes`
 - Leur indice, relativement à la position du premier élément, entre crochets
`[0]`, `[1]`, `[2]`



```
notes[0] = 50;  
notes[1] = 70;  
notes[2] = 90;
```

Utilisation des tableaux

- Utiles dans des cas où l'on désire sauvegarder des données de même type
 - Utilisation de boucles pour traiter les données
- Attention à l'initialisation des tableaux
 - A partir d'un fichier, entrées clavier, valeurs par défaut etc.
 - Après l'initialisation, les traitements peuvent alors se faire
- Possibilité de sauvegarder les éléments du tableau dans un fichier pour lecture/traitement ultérieurs (on traitera les fichiers plus tard)

Problèmes potentiels avec les tableaux

Indice hors limites

- Java génère une 'exception' `ArrayIndexOutOfBoundsException` dans le cas suivant :

```
int notes[4]; // indice maximum possible est 3
```

```
notes[4] = 5; // sauvegarde 5 à une position en  
              // dehors des limites du tableau
```

- Dans les boucles de traitement, on préférera :

```
for (i= 0; i < notes.length; i++)
```

```
...
```

Problèmes potentiels avec les tableaux

Attention aux opérations usuelles!

- En particulier, l'affectation :
 - Ne permet pas la copie des éléments dans un nouvel emplacement mémoire
 - Crée une nouvelle référence au premier élément dans le tableau de départ
- Opérations invalides
Supposons : `int a [20]; int b[20];`
 - `a = b;` // *ne copie pas les éléments de b dans a*
 - `if (a == b)` // *ne permet pas de vérifier si a et b ont les mêmes éléments*
 - `System.out.println (a);` // *n'affiche pas le contenu de a*
 - `a = a * b;` // *ne multiplie pas les éléments de a et de b*



Passage en paramètre

- Un tableau peut toutefois être passé en paramètre par l'intermédiaire de sa référence (i.e. l'emplacement du premier élément du tableau dans la mémoire)

```
int a[20]; double b[20];  
traiteTableaux(a,b);
```

Tableaux d'Objets

- Un tableau peut comprendre des éléments de tout type, y compris des éléments de type Classe
 - Le type Classe 'Etudiant'
 - Etudiant[] : un tableau d'objets Etudiant

`Etudiant[] etudIUT = new Etudiant[50];`

- Accès aux membres (attributs, méthodes) des éléments du tableau en utilisant l'opérateur pointé après le ']' :

`etudIUT[5].toString()` → affiche les valeurs des attributs de l'étudiant à l'indice 5

Plan du Cours

- Tableaux
- **Classe** `ArrayList`
- Associations '1 à Plusieurs'
- Agrégation & Composition

La classe `ArrayList`

La classe `ArrayList` permet d'instancier des objets pour manipuler une collection d'objets de même type, qui sont donc *comme* des tableaux mais :

- Possibilité d'ajouter dynamiquement plus d'objets dans un objet de type `ArrayList`
- Possibilité de supprimer des objets sans se 'tracasser'

Utilisation de la classe `ArrayList`

- Déclaration d'un objet de type `ArrayList` requiert de spécifier le type des objets manipulés:

```
ArrayList<Chien> mesChiens;
```

- Instanciation d'un objet requiert la syntaxe suivante :

```
mesChiens = new ArrayList<Chien>();
```

Ne pas oublier les parenthèses

- Initialement, un objet `ArrayList` est vide.
- Ajout d'éléments – passage de l'objet ajouté en paramètre de la méthode `add`

```
mesChiens.add( fido );
```

- Suppression – utilisation de l'indice

```
mesChiens.remove( 0 );
```

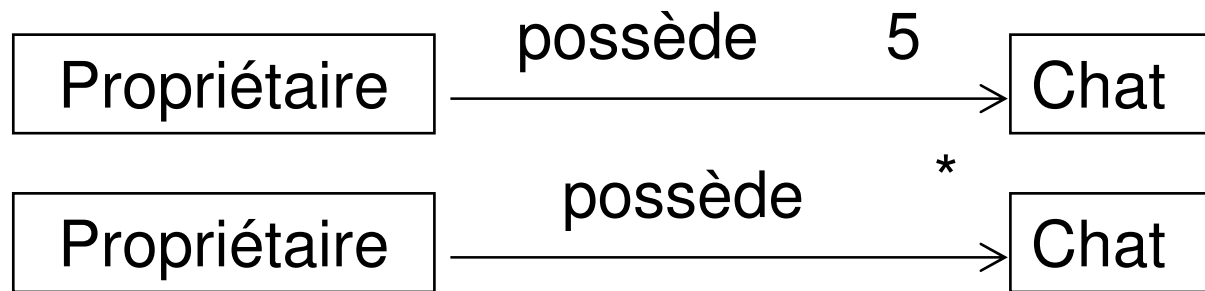
- Nb. éléments de l'`ArrayList` : **mesChiens.size();**

Plan du Cours

- Tableaux
- Classe `ArrayList`
- Associations '1 à Plusieurs'
- Agrégation & Composition

Multiplicité : 1 à Plusieurs

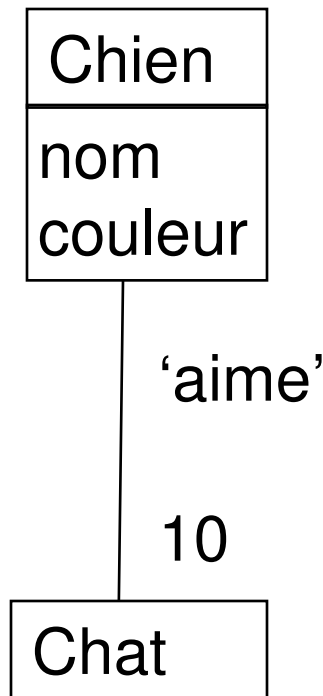
- Instance d'une classe liée à un nombre spécifié (ou non) d'instances de l'autre classe



- Implémentation
 - 1 à un nombre spécifique (petit)
 - Utilisation d'un tableau pour enregistrer les références
 - Compteur dans la classe expéditrice du nombre d'instances de la classe cible qui sont liées à l'instance de la classe expéditrice
 - 1 à plusieurs (grand nombre)
 - Enregistrement des instances de la classe cible dans un objet conteneur (e.g. `ArrayList`) déclaré dans la classe expéditrice

Multiplicité

Collections pour Association 1 à Plusieurs



```
class Chien {
    private String nom;
    private String couleur;
    private Chat[ ] portee;
    private int totalChatsAimes;
    private static final MAX=10;

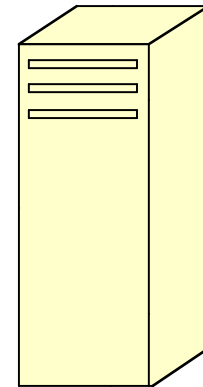
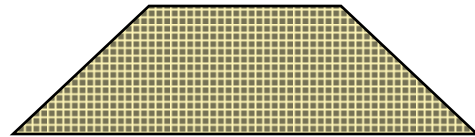
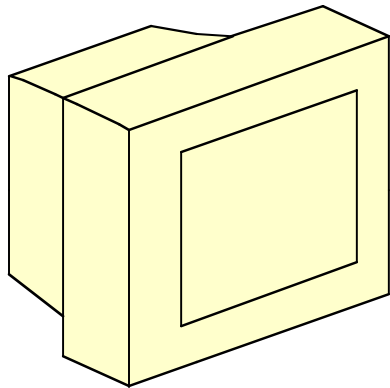
    Chien (Chat [ ] portee) {
        nom = "Bitsa";
        couleur = "marron";
        this.portee = new Chat[10];
        for (int j=0; j<MAX && j<portee.length; j++)
            this.portee[ j ] = portee[ j ];
        totalChatsAimes = j;
    }
}
```

Plan du Cours

- Tableaux
- Classe `ArrayList`
- Associations '1 à Plusieurs'
- Agrégation & Composition

Agrégation

- Une association peut quelquefois être caractérisée par une relation *partie-de*



- Un écran est une *partie d'*un PC
 - Un PC est une *collection* de composants et *a* ou *contient* un écran
- Une relation partie-tout est appelée *agrégation*
Un ordinateur est l'agrégation d'un écran, unité centrale, clavier, souris...

Composition

- Forme d'agrégation pour laquelle l'existence de l'objet composé dépend de l'existence de ses composants
 - Toutefois, les composants peuvent exister individuellement
 - Les composants ne peuvent appartenir à plus d'une composition à un temps t
 - Durée de vie des composants est la même que la durée de vie de l'objet composé
- Exemple : Une chaise est un objet composé d'un dossier, d'un siège et de quatre pieds.
- Les parties peuvent être créées avant l'objet composé, mais une fois qu'elles sont liées à l'objet, elles vivent et meurent avec.
 - Peuvent quelquefois être dégagées de l'objet composé lorsque ce dernier est détruit
 - L'objet composé gère souvent la création et la désintégration de ses parties

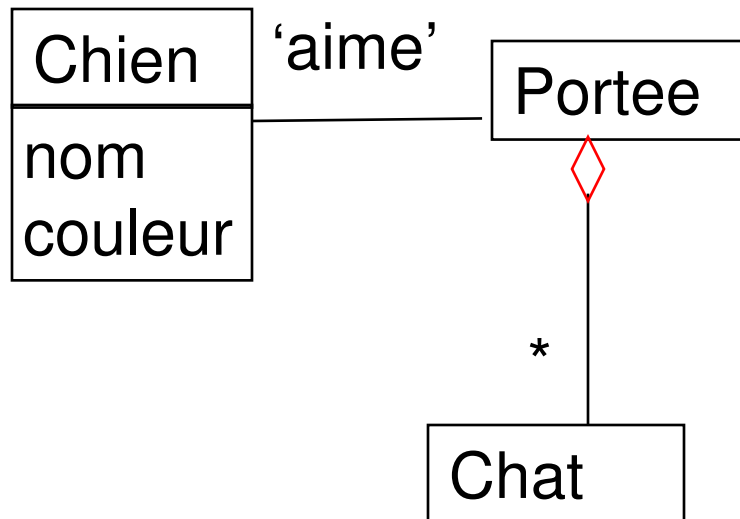
Distinction entre Agrégation & Composition

- Propriétés des relations :
 - *Transitivité*
Si A est une partie-de B et B est une partie-de C alors A est une partie-de C
Exemple d'une relation non-transitive : “est la mère de”
 - *Antisymétrie*
Si A est une partie-de B et B est une partie-de A alors $A = B$
- Si le tout contrôle la partie pendant sa durée de vie, il s'agit fort probablement d'une relation de composition

Agrégation & Composition en Java

- Implémentées de la même manière que les associations
- Technique
 - Les parties sont des variables de référence à l'intérieur du tout
 - Quelquefois gérées par des objets conteneur (comme par exemple `ArrayList`)

Objets Conteneurs comme Agrégats



- En déplaçant le code pour gérer un tableau de Chats en dehors de la classe Chien et à l'intérieur d'une classe Portee, il devient réutilisable pour d'autres classes
- Nous séparons également la problématique liée à la gestion des données (SDD) des applications qui les utilisent (algorithmes)

Objets Conteneurs comme Agrégats

```
class Chien {  
    private String nom;  
    private String couleur;  
    private Portee portee;  
  
    Chien (Portee portee) {  
        name = "Bitsa";  
        colour = "marron";  
        this.portee = portee; }  
}
```

Code à comparer
avec celui de la slide 18

```
class Portee {  
    private Chat[ ] chats;  
    private int totalChats;  
    private static final MAX=10;  
  
    Portee () {  
        chats = new Chat[MAX];  
        totalChats = 0;  
    }  
    public void ajout(Chat chat) {  
        chats[totalChats] = chat;  
        totalChats++;  
    }  
}
```