Les classes et les objets

[Tasso, chapitre 7]

(acétates basés sur matériel conçu par Sébastien Roy et François Duranleau)

Types simples, type complexes

Types simples («primitifs») : byte, short, int, long, float, double, char, boolean

Dans la vie réelle :

Types complexes : colis postaux, cercles, comptes de banque, dossiers étudiants, ...

Comment représenter ces *objets* du monde réel ?
→ par des *objets* du monde logiciel...

Philosophie de programmation OO

Système logiciel = Collection d'objets qui coopèrent pour résoudre un problème.

Les **objets** sont les abstractions fondamentales d'un système logiciel.

Ils correspondent souvent à des entités du monde réel.

Un système logiciel doit accomplir un certain nombre de **tâches**.

Ces tâches constituent la *fonctionnalité* du système.

Les objets sont les éléments responsables d'accomplir chacune des tâches.

Objet = groupement «naturel» de tâches et d'information

Propriétés d'un objet

Un objet *contient* de pièces d'information associés.
→ propriétés de l'objet.

Objet	Propriété
`Compte de banque`	`Solde en $` (entier)
`Cercle`	`Rayon` (réel)
`Dé`	`Nombre de faces` (réel)

Un objet associe une valeur à chaque propriété.
Un objet doit pouvoir travailler avec ces valeurs sur demande.

Où est la différence entre types simples et des objets ?

Différence 1 : plusieurs propriétés en même temps.

Propriétés d'un objet 2

Un objet peut avoir plusieurs propriétés.

Objet	Propriété
`Colis postal`	`Largeur` (réel)
	`Hauteur` (réel)
	`Profondeur` (réel)
	`Poids` (réel)
	`Distance` (entier)
`Étudiant`	`Nombre de crédits à faire` (entier)
	`Nombre de crédits réussis` (entier)
	`Pointure des souliers` (entier)
	`Taille en cm` (entier)

Il faut se limiter aux propriétés pertinentes au problème !

Commandes et état

Les valeurs des propriétés peuvent changer.

À un instant précis, la liste des propriétés et leur valeurs associées constituent l'**état** d'un objet.

L'objet doit répondre à un certain nombre de **commandes** pour modifier les valeurs de ses propriétés.

Ex : Étudiant vient de réussir un cours de 3 crédits, etc...
Ex : Ajouter $100 au compte de banque
Ex : Calculer les intérêts sur le solde du compte

Bref,
Un objet éxécute des requêtes et des commandes (*méthodes* en Java)

Utilisation des objets (Exemples)

Création d'un système logiciel :

- Quels sont les objets ?
- Quels sont les requêtes et commandes requises ?

$\Rightarrow$ identification de tâches et de l'information pour définir les objets + leurs interactions

C'est là que réside l'art de programmer....

Pour l'instant : On fait l'hypothèse que nous connaissons les spécifications exactes des objets.

L'objet composite

Un objet composite sert uniquement à rassembler des valeurs (ni requêtes ni commandes).

But : Simplifier la manipulation

Objet	Propriété
`Colis postal`	`Largeur` (réel)
	`Hauteur` (réel)
	`Profondeur` (réel)
	`Poids` (réel)
	`Distance` (entier)
`Date`	`Jour` (entier)
	`Mois` (entier)
	`Année` (entier)

→ utile comme argument à une fonction ou comme valeur retournée

Objets et classe

Une **classe** représente un ensemble d'objets qui partagent les même propriétés, répondent aux même requêtes et commandes.

Chaque objet est une **instance** d'une classe.

Classe → On **sait d'avance** ce qui est un colis postal

- propriétés (Largeur, Hauteur, ...)
- requêtes (Calcule le volume, ...)
- commandes (Lit un colis au clavier)

Objets → Chaque colis envoyé est une instance et n'est **pas connu d'avance**.

- valeurs spécifiques des propriétés (Largeur = 10.5, ...)

Exemple : Colis postal

Propriétés :

- dimensions (3 variables)
- poids (1 variable)
- distance d'envoi (1 variable)

```
public class ColisPostal
{
   public double Largeur, Profondeur, Hauteur;
   public double Poids;
   public int Distance;
}
```

C'est joli mais . . .

- Qu'est-il arrivé au `static` (variables de classe) ?
- Comment *créer* un objet à partir de cette classe ?
- Comment *manipuler* les objets ainsi crées ?

Variable de classe, variable d'objet

Variables de classe : Une seule instance globale existe pour tout le programme.

```
public class ColisPostal
{
   public static double Largeur;
   ...
}
```

Variables d'objet : Une nouvelle instance est créée pour chaque nouvel objet issu de la classe.

```
public class ColisPostal
{
   public double Largeur;
   ..
}
```

Remarque : on utilise rarement les variables de classe (exemple typique : valeurs constantes comme π dans la classe `java.lang.Math`)

Variable de classe

Si on veut compter le nombre total et le poids total des colis, on aura :

```
public class ColisPostal
{
   public static int NbColisEnvoye;
   public static double PoidsTotal;

   public double Largeur, Profondeur, Hauteur;
   ...
}
```

- Une seule instance de `NbColisEnvoye` et `PoidsTotal` pour le programme (variables de classe).
- Une instance de `Largeur, Profondeur`,. . . pour chaque objet (ici, un colis) (variables d'objet).

Comment déclarer et créer une instance d'un objet ?

Déclaration : `Nom-de-la-classe Nom-de-la-variable ;`

Exemple : `ColisPostal A;`

Instanciation : `variable = new Nom-de-la-classe ();`

Exemple : `A = new ColisPostal();`

Attention : Une variable de type simple (int, double, ...) est toujours instanciée lors de sa déclaration. **Pas les objets !** Mais on peut combiner déclaration et instanciation.

Exemple : `ColisPostal A = new ColisPostal();`

Comment manipuler une instance d'un objet ?

Comment référer à une variable d'objet ?

```
Nom-de-la-variable-objet . Nom-de-la-propriété
```

Exemple :

```
public static void main(String args[])
{
    ColisPostal A = new ColisPostal();

    // Modifier l'etat de l'objet A
    A.Largeur = 40.8;
    A.Hauteur = 103.5;

    // Lire l'etat de l'objet A
    Volume = A.Largeur * A.Hauteur * A.Profondeur;
}
```

Important : Ici, `ColisPostal` est un objet composite.

Théorie de langages de programmation

Déf. Un type est un ensemble (possiblement infini) de valeurs et d'opérations sur celles-ci.

Le langage de programmation prédéfinit des types et offre une manière de définir des autres dans votre programme

Déf. Une variable est l'abstraction d'un emplacement en mémoire. *(von Neumann)*

Plus précisément : variable =

nom	
+**adresse**	*«l-value»*
+**valeur**	*«r-value»*
+**type**	
+**portée**	*(visibilité du nom)*

Variables

Assez simple :

```
int x = 5;
int y = x;
x = 6;
```

. . . aucune surprise

Variables pour objets (y inclus les tableaux) en Java : r-value (valeur droite) est une référence

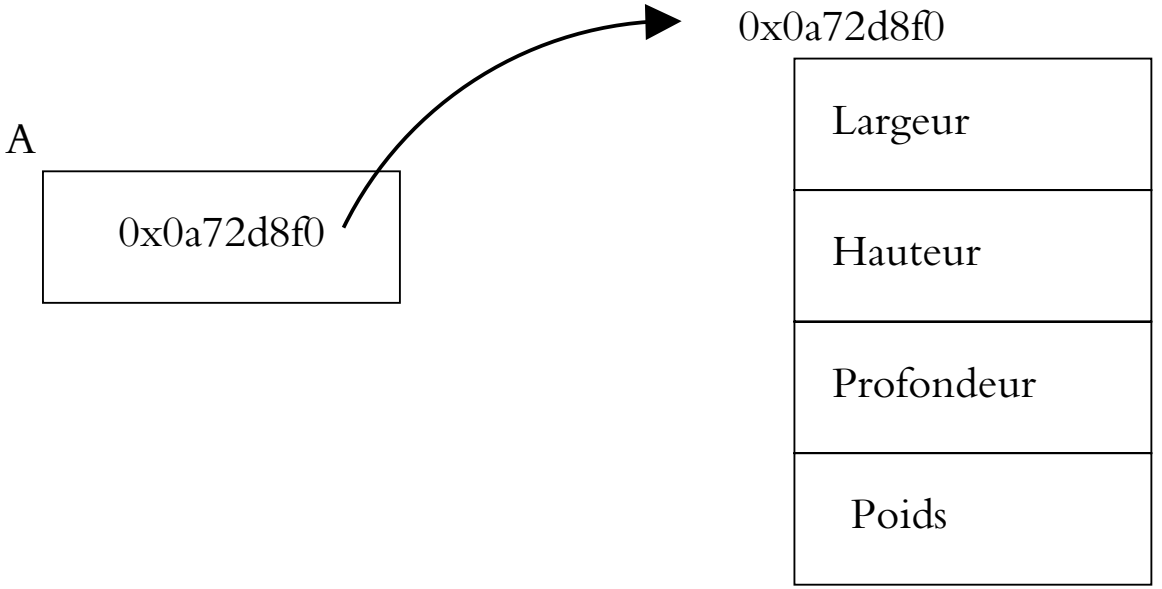
```
int[] x = new int[1];
x[0]=0;
int[] y = x;
y[0] = 1; // et x[0]==1 aussi!
```

Référence

```
ColisPostal A = new ColisPostal();
```

- Allocation de la mémoire pour un nouvel objet
- Stockage de l'adresse mémoire dans A

  → `A` contient une *référence* à l'objet, pas l'objet lui-même.

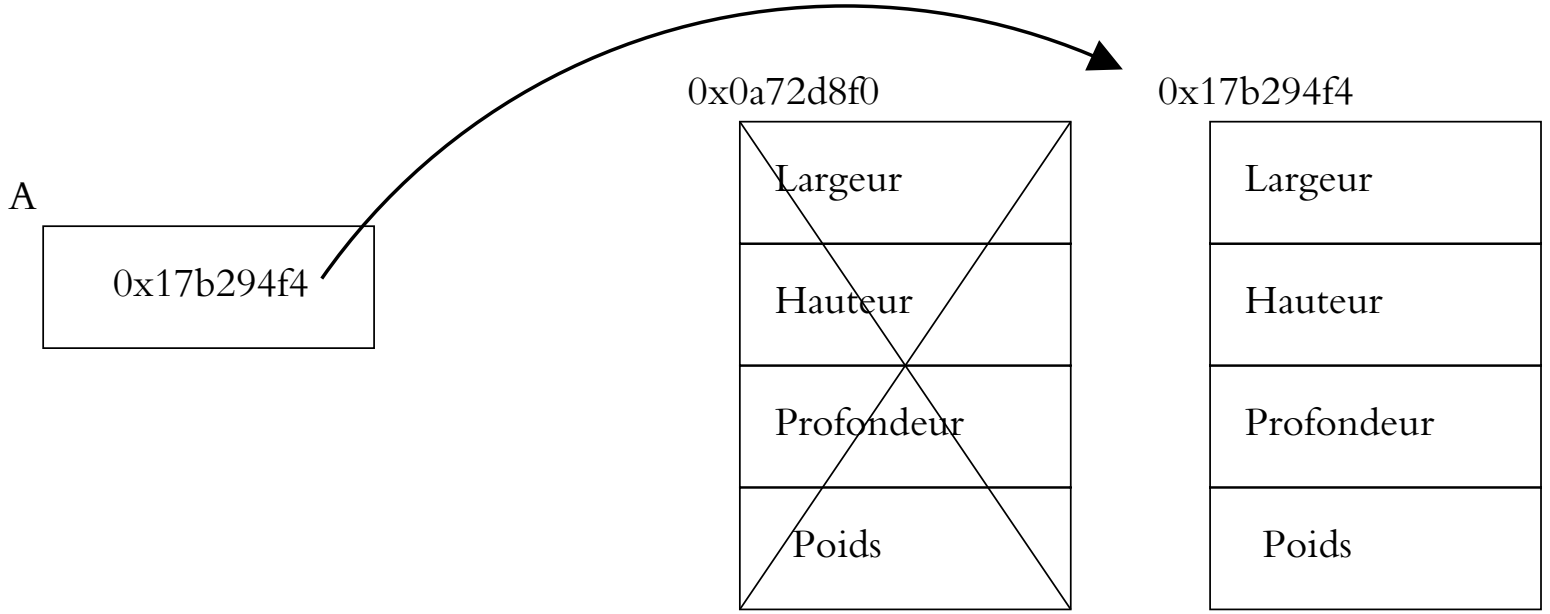


Références

Si on éxécute ensuite

```
A = new ColisPostal();
```

on remplace l'ancienne référence (vers `0x0a72d8f0`) par référence au nouvel objet (ici à l'adresse `0x17b294f4`).
L'objet original est perdu (et sa mémoire peut retourner au système).

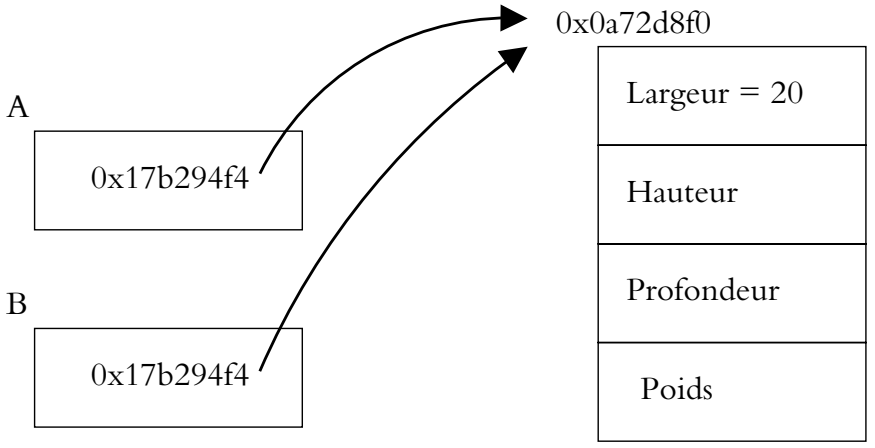


Référence nulle : `A = null;`

Références

Deux variables peuvent référer au même objet...

```
ColisPostal A,B;
A = new ColisPostal();
A.Largeur=10;
B = A;
B.Largeur=20;
```



Objets composites

Objet composite

=

On peut lire et modifier directement les propriétés

=

Propriétés déclarées `public` dans la classe

```
public class ColisPostal
{
   public double Largeur, Profondeur, Hauteur;
   public double Poids;
   public int Distance;
}
```

Si on remplace `public` devant les propriétés par `private` ?

`private` restreint l'accès (permis seulement dans la classe contenant la déclaration)

→ On doit utiliser des requêtes et commandes.

Objet avec propriétés publiques

Déclaration :

```
public class ColisPostal
{
   public double Largeur, Profondeur, Hauteur,Poids;
}
```

Utilisation :

```
ColisPostal A = new ColisPostal();

// Modifier l'etat de l'objet A
A.Largeur = 40.8;
A.Hauteur = 103.5;

// Lire l'etat de l'objet A
Volume = A.Largeur * A.Hauteur * A.Profondeur;
```

Objet avec propriétés privées

```
public class ColisPostal
{
   private double Largeur, Profondeur, Hauteur,Poids;

   public double LitLargeur() { return Largeur; }
   public void ChangeLargeur(double L) { Largeur=L; }
}
```

Utilisation :

```
ColisPostal A = new ColisPostal();

// Modifier l'état de l'objet A
A.ChangeLargeur(40.8);
A.ChangeHauteur(103.5);

// Lire l'etat de l'objet A
Volume = A.LitLargeur() * A.LitHauteur() * A.LitProfondeur();
```

Objet Non Composite

Pourquoi ne pas ajouter le volume dans l'objet ?

```
public class ColisPostal
{
   private double Largeur, Profondeur, Hauteur,Poids;

   public double LitLargeur() { return Largeur; }
   public void ChangeLargeur(double L) { Largeur=L; }
   public double Volume() { return Largeur*Profondeur*Hauteur; }
}
```

Utilisation :

```
...
Volume = A.Volume();
...
```

Composite ou Non composite ?

En pratique, on utilise rarement des propriétés publiques

Pourquoi ?

⋆ Un objet composite peut voir ses valeurs modifiées n'importe comment n'importe où dans le programme.

⟶ erreur plus fréquentes et plus difficiles à corriger

Par exemple, pour garantir que `Largeur` est positif

```
public void ChangeLargeur(double L)
{
   if( L>=0.0 ) Largeur=L; else Largeur=0.0;
}
```

⋆ On peut changer l'implantation d'une classe en préservant les même requêtes sans devoir changer le code ailleurs

⟶ principe d'opacité

Objets comme paramètre de fonction

Un objet peut être passé en paramètre à une fonction.

```
Etudiant A = new Etudiant();
Classe B = new Classe();

...
A.inscrire(B);
```

Important : On passe une référence, pas l'objet lui-même. La fonction peut donc modifier l'objet à sa guise.

Objet retourné par une fonction

Un objet peut être retourné par une fonction.

```
ColisPostal A;

A = LireNouveauxColis();
```

→ plus de problème pour retourner des valeurs multiples !

Objets comme propriétés d'un objet

Une propriété d'un objet peut être une classe plutôt qu'un type simple.

Objet	Propriété
`Étudiant`	`CreditsReussis` (entier)
	`Nom` (String)
	`Prenom` (String)
	`PartenaireDeLabo` (Étudiant)

Ici, la variable d'objet `PartenaireDeLabo` contient une référence vers un objet de la classe étudiant.

Exemple complet : Point

Nous désirons manipuler des points 2D : un point est défini par les deux coordonnées, x et y.

Version *objet composite* :

```
public class Point
{
    public double x,y;  // les coordonnees
}
```

Pour utiliser :

```
Point A = new Point();

A.x = 10.0;
A.y = 20.0;
```

Point II

Version *objet non composite* :

```
public class Point
{
    private double x,y;  // les coordonnees

    public void initialise(double nX, double nY)
    {
        x=nX;
        y=nY;
    }
}
```

Pour utiliser :

```
    Point A = new Point();

    A.initialise(10.0,20.0);
```

Point III

Et si on veut, on pourrait même changer l'implantation

```
public class Point
{
    private double r,phi;

    public void initialise(double nX, double nY)
    {
        r = Math.sqrt(nX*nX+nY*nY);
        phi = Math.atan2(nY, nX);
    }
    public double lireX()
    {
        return r*Math.cos(phi);
    }
    ...
}
```

rien à changer dans d'autres classes . . .