IFT 1015 - Objets 1

Professeur:

Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier

Département d'informatique et de recherche opérationnelle

Université de Montréal

hiver 2006

Au programme

[Tasso:7] et [Niño: 2.3-6]

- Type simple, type complexe
- Propriétés et Requêtes
- Commandes et état
- Utilisation des objets (Exemples)
- Classe, objets, instances (2.4)
- Objets comme propriétés d'un objet (2.5)

Types simples, type complexes

Types simples : 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?

$\Rightarrow$ par des *objets* du monde logiciel...

Objets

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 *fonctionalité* du système.

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

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

Objets : exemple

Système d'inscription des étudiants aux cours.

- *Tâche à accomplir*: Inscription d'un étudiant à un cours
- *Objets pertinents*: `étudiant` et `cours`

...

Propriétés et Requêtes

Un objet *contient* de l'information $\Rightarrow$ 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 nous retourner ces valeurs sur demande.

Hummm.... Ça ressemble aux type simples...

Combien de propriétés en même temps?

Propriétés et Requêtes

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 pouvent changer.

Les propriétés et leur valeurs associées constituent l'**état** d'un objet.

L'objet répond à certaine **commandes** pour modifier son état

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

Utilisation des objets (Exemples)

Création d'un système logiciel:

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

Utilisation des objets (Exemples)

Création d'un système logiciel:

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

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

- Pour l'instant, supposons que nous connaissons les spécifications exactes des objets.

L'objet composite

Un *objet composite* sert uniquement à rassembler des valeurs ensemble ($\longrightarrow$ pas de requête ni de commande).

But: Simplifier la manipulation

Objet	Propriété
`Colis postal`	`largeur` (réel) `hauteur` (réel) `profondeur` (réel) `poids` (réel)
`Date`	`jour` (entier) `mois` (entier) `année` (entier)

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 $\longrightarrow$ On **sait d'avance** ce qu'est un colis postal

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

Objets $\longrightarrow$ Chaque colis est une instance et n'est **pas connu d'avance**.

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

Exemple: Colis postal

```
public class ColisPostal {
    public double largeur, profondeur, hauteur;
    public double poids;
    public int distance;
}
```

Important:

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

Variable de classe, variable d'objet

Variables de classes: Une seule instance pour tout le programme.

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

Variables d'objet: Une instance pour chaque objet de la classe

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

$\Rightarrow$ on utilise rarement les variables de classe.

Variable de classe, variable d'objet

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

```
public class ColisPostal {
   public static int nbColisEnvoyes;
   public static double poidsTotal;

   public double largeur, profondeur, hauteur;
   public double poids;
   public int distance;
}
```

- Une seule instance de `nbColisEnvoye` et `poidsTotal`.
- Une instance de `largeur, profondeur, ...` pour chaque colis.

Déclarer/créer une instance

Déclaration: `NomDeLaClasse nomDeLaVariable;`

Ex: `ColisPostal a;`

Instanciation: `variable = new NomDeLaClasse();`

```
a = new ColisPostal();
```

Souvent on combine les deux:

Ex: `ColisPostal a = new ColisPostal();`

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

Manipuler une instance

Comment référer à une variable d'objet?

`nomDeLObjet . nomDeLaPropriété`

Exemple:

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

    // Modifier l'état de l'objet a.
    a.largeur = 40.8;
    a.hauteur = 103.5;

    // Lire l'état de l'objet a.
    double volume = a.largeur * a.hauteur * a.profondeur;
}
```

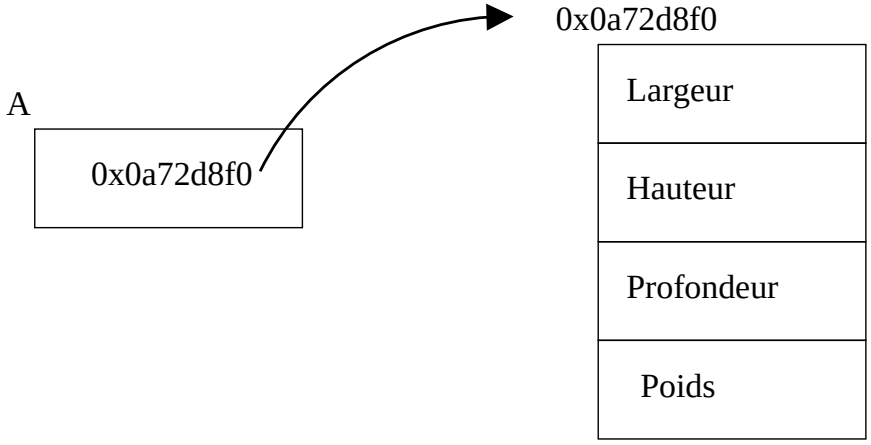
Ici, `ColisPostal` est un objet composite.

Valeur & référence

```
ColisPostal a = new ColisPostal();
```

- Allocation de la mémoire pour un nouvel objet
- Storage de l'adresse mémoire dans a

  $\longrightarrow$ `a` contient une *référence* à l'objet, pas l'objet lui-même.



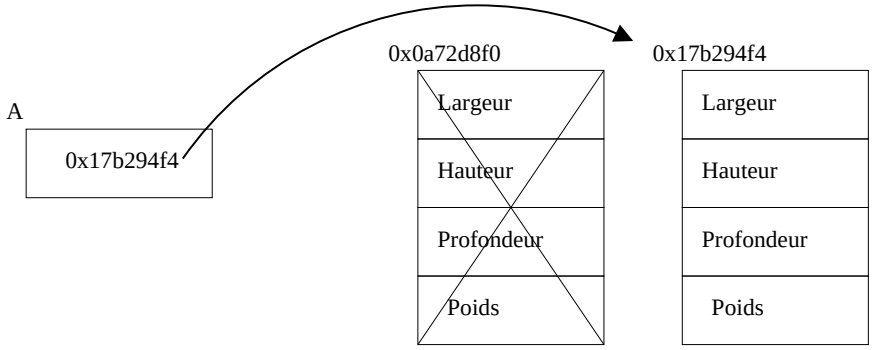
Une référence est une adresse mémoire (ici `0x0a72d8f0`).

Valeur & référence (suite)

Si on éxécute ensuite

```
a = new ColisPostal();
```

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

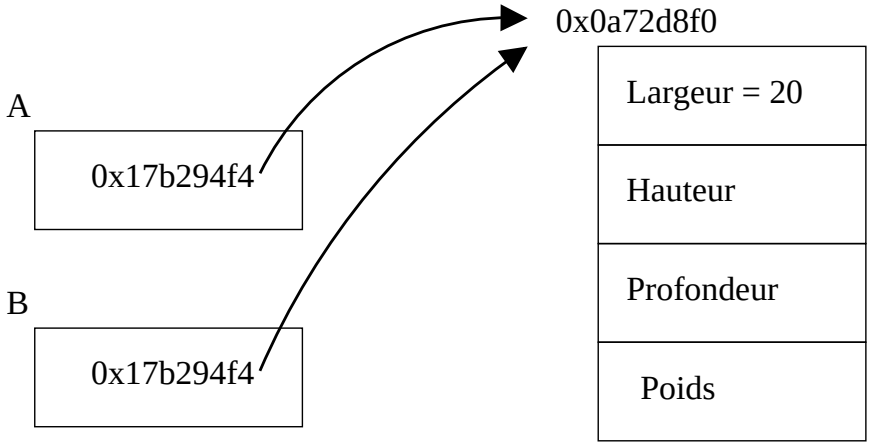


On peut aussi libérer l'objet en éxécutant `a = null;`

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 `publique` 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`?

$\longrightarrow$ On doit utiliser des requêtes et commandes.

Objet Composite (exemple)

Déclaration:

```
public class ColisPostal {
   public double largeur, profondeur, hauteur, poids;
}
```

Utilisation:

```
ColisPostal a = new ColisPostal();

// Modifier l'état de l'objet a
a.largeur = 40.8;
a.hauteur = 103.5;

// Lire l'état de l'objet a
volume = a.largeur * a.hauteur * a.profondeur;
```

Objet non composite

```
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'état 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 *
}
```

Utilisation:

```
...
volume = a.volume();
...
```

Composite ou non composite?

En pratique, on utilise rarement les objets composites.

Pourquoi?

Composite ou non composite?

En pratique, on utilise rarement les objets composites.

Pourquoi?

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

$\longrightarrow$ erreur plus fréquentes et plus difficiles à corriger

E.g. garantir que `largeur` est positif:

```
public changeLargeur (double l) {
  largeur = (l >= 0.0 ? l : 0.0);
}
```

Objets comme paramètre de fonction

Un objet peut être passé en paramètre à une fonction.

```
Etudiant a = new Etudiant();
Etudiant b = new Etudiant();


...
t = sontIlsEnsemble(a, b);
```

Important: On passe une référence à l'objet, 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();
```

$\longrightarrow$ 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.

Classes & Fichiers

Jusqu'ici, on utilise la règle:

Une seule classe définie par fichier

La vraie règle est:

1. Une et une seule classe `public` par fichier
2. Autant de classes non `public` qu'on veut dans un fichier
   (ces classes ne seront visibles que par la classes publique de la règle 1)

Exemple complet: Points

Nous désirons manipuler des points 2d, qui possèdent deux coordonnées entières, x et y.

Version *objet composite*:

```
public class Point {
   public int x, y;  // les coordonnées.
}
```

Pour utiliser:

```
Point a = new Point();

a.x = 10;
a.y = 20;
```

Exemple complet: Points

Version *objet non composite*:

```
public class Point {
    private int x, y;  // les coordonnées.
    public void initialise(int nX, int nY) {
        x = nX;
        y = nY;
    }
}
```

Pour utiliser:

```
Point a = new Point();
a.initialise(10, 20);
```

Constructeurs

Pour simplifier l'initialisation, on peut utiliser un *constructeur*.

Avant:

```
public class Point {
   private int x, y;  // les coordonnées.
   public void initialise (int nX, int nY) { x = nX; y = n
}
```

Après:

```
public class Point {
   private int x, y;  // les coordonnées.
   public Point (int nX, int nY) { x = nX; y = nY; }
}
```

Qu'elle est la différence?

Constructeurs

Utilisation sans constructeur:

```
Point a = new Point();
a.initialise(10, 20);
```

Utilisation avec constructeur:

```
Point a = new Point(10, 20);
```

On sauve une ligne! Incroyable! :-)

En pratique: Les constructeurs sont très utiles.

Exemple complet: Points

Pour déplacer et afficher un point:

```
public class Point {
    ...
    public void deplace (int dx, int dy)
    { x += dx; y += dY; }

    public void affiche() {
        System.out.println("(" + x + ", " + y + ")");
    }
    ...
```

Pour utiliser:

```
Point a = new Point (10, 20);
a.deplace(5, -7); a.affiche();
```

Références : copie d'objets

```
Point a = new Point(3, 5);
Point b;
```

Après `a = b;` les deux variables réfèreraient au même objet.

Références : copie d'objets

```
Point a = new Point(3, 5);
Point b;
```

Après `a = b;` les deux variables réfèreraient au même objet.

Si on veut deux objets, alors il faut copier explicitement...

```
public Point copie ()
{ return new Point(x, y); }
```

Utilisation:

```
Point a = new Point(3, 5);
Point b = a.copie();
```

Références : comparaison d'objets

```
Point a, b;
```

`a == b` compare les adresses, pas les objets eux-mêmes!

Références : comparaison d'objets

```
Point a, b;
```

`a == b` compare les adresses, pas les objets eux-mêmes!

```
public boolean estEgal (Point b)
{ return (x == b.x) && (y == b.y); }
```

Utilisation:

```
if (a.estEgal(b)) { ... }
```

Références : comparaison d'objets

```
Point a, b;
```

On peut aussi utiliser une méthode de classe.... le bon vieux `static`, utilisé depuis le début du cours...

```
public static boolean compare (Point a, Poin
{ return (a.x == b.x) && (a.y == b.y); }
```

Utilisation:

```
if (Point.compare(a, b)) { ... }
```