

IFT 1010 - Programmation 1

Objets 1

Sébastien Roy & François Duranleau



Département d'informatique et de recherche opérationnelle
Université de Montréal
automne 2004

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)

1

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 ?

→ 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 *fonctionnalité* du système.

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

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

2

3

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.

→ 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.

Humm... Ça ressemble aux type simples...

Combien de propriétés en même temps ?

4

5

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 doivent pouvoir 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 exécute des requêtes et des commandes

6

7

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

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

8

L'objet composite

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

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)

9

Objets et Classe

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

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

Classe → On **sait d'avance** ce qu'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, ...)

10

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 ?

11

Variable de classe, variable d'objet

Variables de classes : 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;
    ..
}
```

→ on utilise rarement les variables de classe.

12

Comment déclarer et créer une instance d'un objet ?

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

Ex : `ColisPostal A ;`

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

`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.

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

14

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 NbColisEnvoye;
    public static double PoidsTotal;

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

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

13

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'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;
}
```

Important

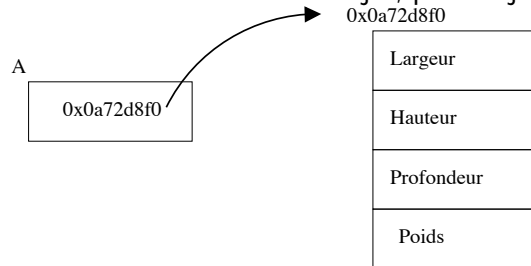
Ici, `ColisPostal` est un objet composite.

15

Valeur & référence

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

- Allocation de la mémoire pour un nouvel objet
- Storage de l'adresse mémoire dans A
→ A contient une *référence* à l'objet, pas l'objet lui-même.



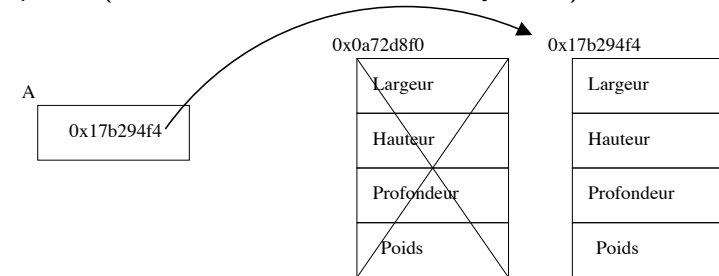
Une référence est une adresse mémoire (ici `0x0a72d8f0`).

16

Si on exécute 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 exécutant `A = null`;

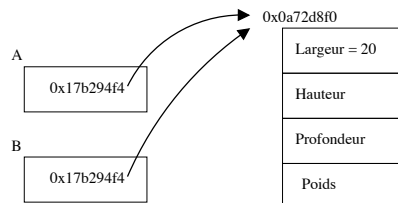
17

Références

Deux variables peuvent référer au même objet...

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

```
A.Largeur=10;  
B = A;  
B.Largeur=20;
```



18

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` ?

→ On doit utiliser des requêtes et commandes.

19

Objet Composite

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;
```

20

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();
```

21

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();
...
```

22

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.
→ erreur plus fréquentes et plus difficiles à corriger

Par exemple, pour garantir que `Largeur` est positif

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

23

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.

24

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 !

25

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.

26

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)

27

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 coordonnees
}
```

Pour utiliser :

```
Point A = new Point();

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

28

Exemple complet : Points

Version *objet non composite* :

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

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

Pour utiliser :

```
Point A = new Point();

A.initialise(10,20);
```

29

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(); A.initialise(10,20);
A.deplace(5,-7); A.affiche();
```

30

Constructeurs

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

Avant :

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

Après :

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

Qu'elle est la différence ?

31

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.

32

Références : copie d'objets

```
Point A = new Point(3,5);
Point B;
```

Si on veut que B soit une copie de A, on peut faire `A=B` ; mais dans ce cas les deux variables réfèrent au même objet. Si on veut deux objets, alors il faut copier explicitement...

```
public class Point {
    ...
    public Point copie() {
        Point p = new Point(x,y);
        return p;
    }
}
```

Utilisation :

```
Point A = new Point(3,5);
Point B = A.copie();
```

33

Références : comparaison d'objets

```
Point A,B;
```

On veut comparer A et B. Si on utilise `A==B` ou `A!=B`, on compare les adresses des objets, pas les objets eux-même !

Comment faire ?

```
public class Point {
    ...
    public boolean estEgal(Point B) {
        return (x == B.X) && (y == B.Y);
    }
}
```

Utilisation :

```
if( A.estEgal(B) ) { ... }
```

34

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 class Point {
    ...
    public static boolean compare(Point A,Point B) {
        return (A.x == B.X) && (A.y == B.Y);
    }
    ...
}
```

Utilisation :

```
if( Point.compare(A,B) ) { ... }
```

35