

Programmation objet

L3Pro SCT – Bases de données et programmation

Mathieu Sassolas

IUT de Sénart Fontainebleau
Département Informatique

Année 2015-2016
Cours 4



- 1 Concept de la programmation orientée objet
- 2 La syntaxe Java
 - Déclarer une classe
 - Le code des opérations
 - Fonction principale, compilation, lancement
- 3 TD/TP

- 1 Concept de la programmation orientée objet
- 2 La syntaxe Java
 - Déclarer une classe
 - Le code des opérations
 - Fonction principale, compilation, lancement
- 3 TD/TP

- ▶ Pouvoir manipuler les données, calculer avec, etc.
- ▶ Les programmes stockent les données dans des **structures de données** : tableau, liste, arbre, pile, file.
- ▶ Une structure de donnée est utile pour garder plusieurs éléments de même forme, pas pour créer des types complexes.
- ▶ Par exemple : un point est formé de deux coordonnées et d'un nom : `Point ∈ float×float×String`.
- ▶ On manipule également des données qui ne seront pas stockées dans une BDD : taille de la fenêtre par exemple.
- ▶ On ne veut pas toujours avoir dans le programme toutes les données de la BDD (qui est souvent grande !)

- ▶ Un conteneur pour des **données** qu'on appelle alors **attributs** ; chaque attribut a un nom.
- ▶ Les données sont soit des types de base (entier, flottant, booléen, caractère...), soit d'autres objets (**String**, Date, Point,...)
- ▶ Quelque chose d'**actif** : l'objet peut lui même manipuler ses données (et d'autres) à l'aide d'**opérations**.

Un objet est construit selon un **modèle** que l'on appelle **classe**.

- ▶ Donne la liste des attributs avec leur type (éventuellement leur valeur par défaut).
- ▶ Donne la liste et le code des opérations que l'objet peut exécuter.
- ▶ Donne le code de l'opération permettant de créer l'objet : un **constructeur** : doit donner une valeur à chaque attribut.

Attributs comme opérations peuvent être **privés** ou **publics**.

Privé N'apparaît pas hors de l'objet ; cela n'empêche pas les opérations de l'objet lui même de le manipuler.

↪ C'est la visibilité à privilégier pour tous les attributs.

Public Est complètement manipulable (en lecture/écriture pour les attributs et en exécution pour les opérations) depuis l'extérieur de l'objet.

↪ C'est la visibilité à privilégier pour toutes les opérations.

- ▶ Il y en a plusieurs, avec chacun leur spécificité syntaxique.
- ▶ Le principe des objets reste le même.
- ▶ Exemples : C++, **Java**.

Programmation
objet

M. Sassolas
L3Pro SCT – M7
Cours 4

Les objets

Java

Les classes
Le code
Compilation etc

TD/TP

1 Concept de la programmation orientée objet

2 La syntaxe Java

- Déclarer une classe
- Le code des opérations
- Fonction principale, compilation, lancement

3 TD/TP

- ▶ Langage compilé vers une **machine virtuelle**
 - portabilité;
 - lenteur;
- ▶ `Programme.java` $\xrightarrow{\text{compilation}}$ `Programme.class`
- ▶ `Programme.class` est du bytecode lu par la machine virtuelle Java.

- ▶ Les entiers : `int`, `long`
- ▶ Les flottants : `float`, `double`
- ▶ Les booléens : `boolean`
- ▶ Les caractères : `char`
- ▶ Les octets : `byte`

Remarques

Le type `String` n'est pas un type de base mais une classe.

Syntaxe

```
<visibilité_de_classe> class <Nom_de_la_classe> {  
    // Déclaration des attributs et opérations  
    /* Remarquez les deux manières de faire  
       des commentaires */  
}
```

Exemple

```
public class Point { /* ... */}
```

Remarque

- ▶ Par convention, les classes commencent par une majuscule.
- ▶ La tradition de nommage habituelle est le « camel case » :
PasEspacesMaisMajusculesAuMilieu.

Syntaxe

```
<visibilité_de_l'attribut> <type>  
    <nom_de_l'attribut> [= <valeur_par_défaut>];  
<visibilité_de_l'attribut> <type>  
    <nom_de_l'attribut1>, <nom_de_l'attribut2>, ...  
    [= <valeur_par_défaut>];
```

Exemple

```
private String name;  
private int score = 42;  
private float x,y;  
private boolean valide,actif = false;
```

Remarque

Par convention, les attributs commencent par une minuscule.

Syntaxe

```
<visibilité> <type_de_retour> <nom_de_l'opération>  
    (<type1> <arg1>, ..., <typeN> <argN>) {  
    // code  
}
```

Exemple

```
public double distance (Point p) { // TODO  
    return 0;} // Pour le type de retour
```

```
public void xShift (float s) {  
    x += s;} // Pas de return car retour "void"
```

Remarque

Par convention, les opérations commencent par une minuscule.

```
public class Point {
    private float x,y;
    private String name;

    /* Accesseurs; pas de modification pour x et y. */
    public float getX() {return x;}
    public float getY() {return y;}
    public String getName() {return name;}
    public void setName(String newName) {
        name = newName;}

    /* Autres opérations */
    public double distance (Point p) {// TODO
        return 0;} // Pour le type de retour
    ...
}
```

- ▶ Un constructeur retourne un objet de la classe donnée.
- ▶ Il doit donner une valeur à tous les attributs qui n'ont pas de valeur par défaut.
- ▶ Le nom de l'opération est celui de la classe (avec la majuscule !); il n'y a pas de type de retour (car c'est forcément un objet de cette classe).

Exemple

```
public Point(float vx, float vy, String vname) {  
    x = vx;  
    y = vy;  
    name= vname;  
}
```

- ▶ Ces méthodes n'utilisent pas les attributs d'un objet.
- ▶ Elles sont définies dans la classe car elles utilisent des objets définis à cet endroit.
- ▶ Elles sont précédées du mot clef `static`

Exemple

```
public static double  
    distanceBis(Point p1, Point p2) {...}
```

- ▶ Chaque instruction se termine par un `;`.
- ▶ La fonction sort une valeur avec le mot clef `return`. Le type de ce qui est retourné doit correspondre au type de retour de la fonction.
- ▶ Une fonction qui ne sort pas de valeur a `void` comme type de retour.
- ▶ Pour afficher sur la console (stdout) :
`System.out.println("Message"+"autre msg");`
- ▶ Chaque variable doit être déclarée avec son type.
- ▶ Un objet peut être déclaré mais non initialisé (ex : `int i;` `Point p;`). Les objets prennent alors la valeur `null`; si on cherche à les utiliser on aura une erreur ! (NullPointerException).

- ▶ Créer une **variable ayant un type de base** :
`typeDeBase maVariable = valeur;`
Par exemple : `int i = 0;`

- ▶ Créer une **variable ayant un type de base** :

```
typeDeBase maVariable = valeur;
```

Par exemple : `int i = 0;`

- ▶ Créer un **objet** :

```
Classe monObjet = new Classe(arguments);
```

Par exemple : `Point origine = new Point(0,0,"0");`

- ▶ Créer une **variable ayant un type de base** :

```
typeDeBase maVariable = valeur;
```

Par exemple : `int i = 0;`

- ▶ Créer un **objet** :

```
Classe monObjet = new Classe(arguments);
```

Par exemple : `Point origine = new Point(0,0,"0");`

- ▶ Utiliser une **opération d'un objet** :

```
monObjet.operation(...)
```

Par exemple : `float xOrig = origine.getX();`

- ▶ Créer une **variable ayant un type de base** :

```
typeDeBase maVariable = valeur;
```

Par exemple : `int i = 0;`

- ▶ Créer un **objet** :

```
Classe monObjet = new Classe(arguments);
```

Par exemple : `Point origine = new Point(0,0,"0");`

- ▶ Utiliser une **opération d'un objet** :

```
monObjet.operation(...)
```

Par exemple : `float xOrig = origine.getX();`

- ▶ Utiliser une **opération de classe (static)** :

```
maClasse.operation(...)
```

Par exemple :

```
double dis = Point.distanceBis(origine,p);
```

- ▶ Créer une **variable ayant un type de base** :
`typeDeBase maVariable = valeur;`
Par exemple : `int i = 0;`
- ▶ Créer un **objet** :
`Classe monObjet = new Classe(arguments);`
Par exemple : `Point origine = new Point(0,0,"0");`
- ▶ Utiliser une **opération d'un objet** :
`monObjet.operation(...)`
Par exemple : `float xOrig = origine.getX();`
- ▶ Utiliser une **opération de classe (static)** :
`maClasse.operation(...)`
Par exemple :
`double dis = Point.distanceBis(origine,p);`
- ▶ Accéder à un objet depuis une fonction qui l'utilise : **this**.
Par exemple : `this.operation();`

- ▶ Java dispose d'un type tableau de base.
- ▶ Pour chaque type de base ou classe, le type `<Type> []` représente un tableau de ce type de variables.
- ▶ Exemple : `int []`, `Point []`.

- ▶ Java dispose d'un type tableau de base.
- ▶ Pour chaque type de base ou classe, le type `<Type> []` représente un tableau de ce type de variables.
- ▶ Exemple : `int []`, `Point []`.
- ▶ Longueur d'un tableau `tab` : `tab.length` (pas de parenthèses car c'est un attribut!)
- ▶ Accès au i -ème élément : `tab[i]`.
- ▶ NB : les éléments sont numérotés de 0 à `tab.length-1`.

- ▶ Java dispose d'un type tableau de base.
- ▶ Pour chaque type de base ou classe, le type `<Type> []` représente un tableau de ce type de variables.
- ▶ Exemple : `int []`, `Point []`.
- ▶ Longueur d'un tableau `tab` : `tab.length` (pas de parenthèses car c'est un attribut!)
- ▶ Accès au *i*-ème élément : `tab[i]`.
- ▶ NB : les éléments sont numérotés de 0 à `tab.length-1`.
- ▶ Créer un tableau : `<Type> [] <nom> = new <Type> [<longueur>];`
- ▶ Il est initialisé à 0 pour les nombres, `false` pour les booléens et `null` pour les objets.
- ▶ Exemples : `int [] tab = new int [42];`
`Point [] hexagone = new Point [6];`

- ▶ Comparaison de nombres : $<$, $<=$, $==$, $!=$, $>=$, $>$.

- ▶ Comparaison de nombres : `<`, `<=`, `==`, `!=`, `>=`, `>`.
- ▶ Comparaison d'objets :
 - `==` et `!=` s'entendent comme égalité des objets : `a == b` ssi les variables `a` et `b` désignent **le même objet**.
 - On peut utiliser la méthode `equals` (implémentée dans toutes les classes standard) qui teste l'égalité de valeur : `a.equals(b)`.

- ▶ Comparaison de nombres : `<`, `<=`, `==`, `!=`, `>=`, `>`.
- ▶ Comparaison d'objets :
 - `==` et `!=` s'entendent comme égalité des objets : `a == b` ssi les variables `a` et `b` désignent **le même objet**.
 - On peut utiliser la méthode `equals` (implémentée dans toutes les classes standard) qui teste l'égalité de valeur : `a.equals(b)`.

```
String a = new String("Bonjour");  
String b = new String("Bonjour");  
System.out.println(a == b);           // false  
System.out.println(a.equals(b));      // true
```

- ▶ Comparaison de nombres : `<`, `<=`, `==`, `!=`, `>=`, `>`.
- ▶ Comparaison d'objets :
 - `==` et `!=` s'entendent comme égalité des objets : `a == b` ssi les variables `a` et `b` désignent **le même objet**.
 - On peut utiliser la méthode `equals` (implémentée dans toutes les classes standard) qui teste l'égalité de valeur : `a.equals(b)`.
 - Savoir si un objet est une instance d'une classe : `<objet> instanceof <Classe>`.

```
String a = new String("Bonjour");  
String b = new String("Bonjour");  
System.out.println(a == b);           // false  
System.out.println(a.equals(b));      // true
```

- ▶ Comparaison de nombres : `<`, `<=`, `==`, `!=`, `>=`, `>`.
- ▶ Comparaison d'objets :
 - `==` et `!=` s'entendent comme égalité des objets : `a == b` ssi les variables `a` et `b` désignent **le même objet**.
 - On peut utiliser la méthode `equals` (implémentée dans toutes les classes standard) qui teste l'égalité de valeur : `a.equals(b)`.
 - Savoir si un objet est une instance d'une classe : `<objet> instanceof <Classe>`.

```
String a = new String("Bonjour");  
String b = new String("Bonjour");  
System.out.println(a == b);           // false  
System.out.println(a.equals(b));      // true
```

```
System.out.println(a instanceof String); // true  
System.out.println(a instanceof Point);  // false
```

- ▶ Affectation simple : `i = 3;`, `b = "Bonjour";`.
- ▶ Affectation avec incrément/décément : `i += 3;` signifie `i = i+3;`. On a aussi `-=`, `*=`, `/=`, `%=`.
- ▶ Incrément/décément : `i++;` signifie `i += 1;` ou `i = i+1;`. On a aussi `i--`.
- ▶ Combinaisons booléennes : conjonction : `&&`; disjonction : `||`; négation : `!`.

- ▶ Affectation simple : `i = 3;`, `b = "Bonjour";`.
- ▶ Affectation avec incrément/décément : `i += 3;` signifie `i = i+3;`. On a aussi `-=`, `*=`, `/=`, `%=`.
- ▶ Incrément/décément : `i++;` signifie `i += 1;` ou `i = i+1;`. On a aussi `i--`.
- ▶ Combinaisons booléennes : conjonction : `&&`; disjonction : `||`; négation : `!`.

Condition

```
if (<condition>) { /* contenu du then */  
  [else if (<condition2>) { /* contenu */}]  
  [else { /* contenu du else */}]
```

```
if (i <= 1) {return 1;} else {return i*fact(i-1);}
```

Boucle « tant que »

```
while (<condition>) {  
    /* Contenu de la boucle */  
}
```

```
int i=99;  
while (i>0) {  
    System.out.println(i+" bottles on the wall...");  
    i = i-1;}  
}
```

Boucle « tant que »

```
while (<condition>) {  
    /* Contenu de la boucle */  
}
```

```
int i=99;  
while (i>0) {  
    System.out.println(i+" bottles on the wall...");  
    i = i-1;}  
}
```

Boucle « pour »

```
for (<initialisation>; <condition>; <incrément>) {  
    /* Contenu de la boucle */  
}
```

```
int i;  
for (i=99; i > 0; i--) {  
    System.out.println(i+" bottles on the wall...");  
}
```

- ▶ Un programme Java est constitué de plusieurs classes, s'appelant les unes les autres (créations d'objets, appels d'opérations de classe...).
- ▶ Chaque classe est définie dans un fichier portant le nom de la classe : `<MaClasse>.java`
- ▶ L'une d'elle sera appelée par le système pour lancer véritablement le programme. Elle porte souvent le nom du programme : par exemple `GeometryForNoobs.java`, qui utilise la classe `Point.java`.
- ▶ La classe principale on doit avoir une fonction `main` dont la signature est :

```
public static void main(String[] args) {...}
```

- ▶ Compilation : chaque fichier .java est compilé avec `javac <Classe>.java`.
- ▶ On peut faire `javac *.java` pour compiler tous les fichiers Java du répertoire courant.
- ▶ Pour le lancement, on utilise `java <ClassePrincipale> arg1 arg2 ...`
- ▶ Cela a pour effet de lancer la fonction `main` avec comme argument le tableau formé de `[arg1; arg2;...]`

Programmation
objet

M. Sassolas

L3Pro SCT – M7

Cours 4

Les objets

Java

Les classes

Le code

Compilation etc

TD/TP

Programmation
objet

M. Sassolas
L3Pro SCT – M7

Cours 4

Les objets

Java

TD/TP

- 1 Concept de la programmation orientée objet
- 2 La syntaxe Java
 - Déclarer une classe
 - Le code des opérations
 - Fonction principale, compilation, lancement
- 3 TD/TP

Programmation
objet

M. Sassolas

L3Pro SCT – M7

Cours 4

Les objets

Java

TD/TP

↳ C'est l'heure du TP ↵