

Programmation Orientée Objet - Licence TIS CM1/9

Lancelot PECQUET

Lancelot.Pecquet@math.univ-poitiers.fr



Poitiers, le 13/01/2006

- 1 Informations pratiques
- 2 Introduction à la POO
- 3 Introduction au langage Java
- 4 Syntaxe élémentaire du langage Java

Contacts et documents

Pour me joindre (Lancelot PECQUET)

- Laboratoire de Mathématiques (SP2MI, Futuroscope)
- Lancelot.Pecquet@math.univ-poitiers.fr

Documents

Sur <http://www-math.univ-poitiers.fr/~pecquet> (RSS)

- les transparents de cours ;
- le photocopié (le consulter pour une bibliographie) ;
- les sujets de TD et TP, projets et examens.

Enseignements et enseignants

- CM : $9 \times 2h$
par Lancelot PECQUET
- TD : $6 \times 2h$
par Bruno MERCIER (TP_I.B & TP_I.C)
et Pierre CALLADINE (TP_M.A & TP_R.D)
- TP : $6 \times 2h$
par Pierre CALLADINE (TP_R.D)
et Yannick DEGARDIN (TP_M.A + TP_I.D & TP_I.C)

Avant vacances de février

mar 03 jan	CM1			introduction, environnement, types de base, notion de classe
ven 13 jan	CM2			classes : structure, encapsulation, constructeurs et destructeurs
ven 20 jan		TD1		classes
lun 23 jan	CM3			héritage simple, redéfinition
lun 23 jan			TP1	environnement, compilation, classes
lun 30 jan	CM4			classes abstraites et interfaces, héritage multiple
lun 30 jan			TP2	pratique de l'héritage simple
ven 03 fev		TD2		héritage simple
lun 06 fev	CM5			exceptions, paquetages, javadoc, conteneurs et itérateurs
lun 06 fev			TP3	classes abstraites, interfaces, héritage multiple
lun 13 fev	CM6			égalité, transtypage, comparabilité, clonage
ven 17 fev		TD3		classes abstraites, interfaces

Après vacances de février

lun 27 fev : **contrôle continu**

lun 27 fev			TP4	exceptions, paquetages, javadoc, conteneurs et itérateurs
ven 03 mar		TD4		égalité, transtypage, clonage
lun 06 mar	CM7			présentation du projet , conception OO, UML, design pattern
ven 10 mar		TD5		préparation du projet
lun 13 mar	CM8			fichiers, sérialisation, threads, applets
lun 13 mar			TP5	égalité, transtypage, clonage, révisions, I/O
lun 20 mar	CM9			révisions autour de String
lun 20 mar		TD6		révisions
lun 27 mar			TP6	finalisation du projet

Questions pratiques ?



Intérêt de la POO

Plusieurs philosophies complémentaires de programmation

- impérative (assembleurs, C, Perl, ocaml...);
- fonctionnelle (lisp, scheme, ocaml...);
- **objet** (C++, Java, ocaml...).

Intérêts principaux de la POO

- 1 objets (données+fonctions) \implies collaboration plus facile (haut niveau);
- 2 encapsulation des données \implies augmentation de la sûreté du logiciel;
- 3 héritage et polymorphisme \implies code réutilisable.

Diagramme de classe UML

Définition

Une **classe** est constituée d'un nom de type, de données (appelées **champs**) et de fonctions (appelées **méthodes**) caractérisant une famille d'objets.

Exemple en UML : une classe voiture

nom de la classe	voiture
champs (données)	modèle vitesse ...
méthodes (fonctions)	freiner() accélérer() atteindre_vitesse(v) ...

Instances d'une classe

Définition

Une **instance** d'une classe *C* est une valeur (= un objet) de type *C* et qui possède donc les champs et méthodes de *C*.

Exemple en UML

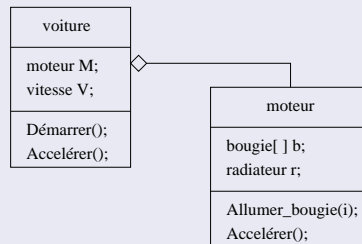
instance 981 AC 33	instance 844 TU 21
modèle = renault_clio	modèle = citroën_2CV
vitesse = 91km/h	vitesse = 54km/h
...	...

Aggrégation

Définition

Lorsqu'une classe C_1 possède un champ de type C_2 , on dit que est C_2 **aggrégée** dans C_1 .

Exemple



Encapsulation

Définition

L'**encapsulation** est un mécanisme limitant l'accès à certains champs ou méthodes d'une classe \implies sécurité et masquage.

Exemple

	Moteur
	...
privé	- Bougie[] b
privé	- Radiateur r
	...
public	+ démarrer()
privé	- allumer_bougie(i)
	...

A la différence de son concepteur, l'utilisateur d'un `moteur` ne peut directement allumer chaque bougie mais peut, par exemple, démarrer.

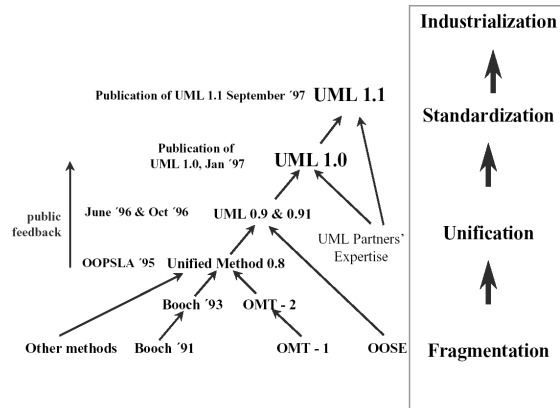
Questions ?



Historique des langages OO

- **60's (Norvège) :** *Simula*
- **70's (USA) :** *Smalltalk (Xérox)* ;, premier système multi-fenêtrage avec écran bitmap et souris
- **80's :** Eiffel (objet pur), extensions OO (C++, Delphi)
- **90's :** Java (1995), OCaml, Python

Historique de la modélisation OO



Avantages de Java

- ① **propre** (typage fort, arithmétique portable) ;
- ② **ramasse-miette** ;
- ③ **sécurisé** (encapsulation, exceptions, security manager, ...) ;
- ④ **multithread** ;
- ⑤ **surcharge des fonctions** (pas de surcharge d'opérateurs) ;
- ⑥ **portable** (JVM, bibliothèques standard, GUI) ;
- ⑦ **international** (Unicode) ;
- ⑧ **adapté à Internet** (applets, réseau) ;
- ⑨ **populaire** (proche de C/C++, utilisé en entreprise).

Inconvénients de Java

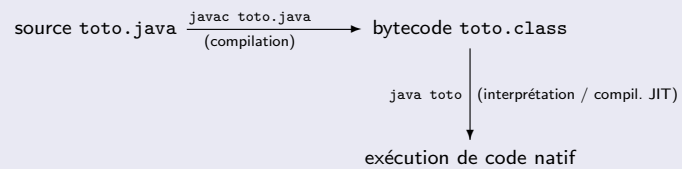
- 1 **lent** (même en mode natif) ;
- 2 **absence d'héritage multiple** (mais interfaces) ;
- 3 **pas de généricité** (pas de *templates*).

Compilation et exécution

Java Virtual Machine

« processeur logiciel » + « exécutables portables »

Compilation de toto.java



Assembleur Java (hackers only)

Désassemblage de toto.class

```
toto.class  $\xrightarrow[\text{(désassemblage)}]{\text{javap toto.class}}$  assembleur portable toto.ksm
```

Assemblage « risqué » donc non prévu officiellement (jasmin).

Premier programme Java

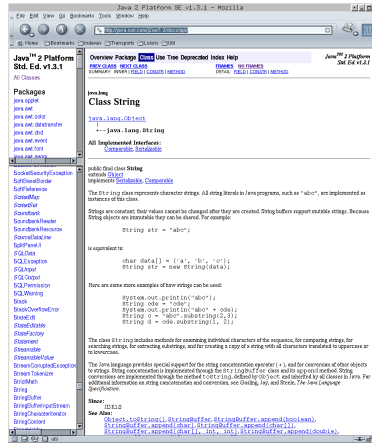
HelloWorld.java

```
1 class HelloWorld{  
2     public static void main(String[] args){  
3         System.out.println("Hello_World");  
4     }  
5 }
```

Commentaires

- **public** : invocation possible de partout ;
- **static** : indépendant de l'instance (méthode de classe).

Aide : <http://java.sun.com/j2se/1.5/docs/api>



Questions ?



Commentaires

```
1 // les commentaires de fin de ligne
2
3 /* Les commentaires susceptibles de
4 s'étendre sur plusieurs lignes */
5
6 /** Les commentaires que le logiciel
7 javadoc pourra utiliser pour générer une
8 documentation HTML
9 */
```

Types primitifs

Nom	représentation
boolean	1 bit
char	16 bits, Unicode UTF-16, 2.1
byte	8 bits, signé
short	16 bits, signé
int	32 bits, signé
long	64 bits, signé
float	32 bits, signé IEEE754/SP/1985
double	64 bits, signé IEEE754/DP/1985

L'arithmétique entière est modulo (pas de débordement)

Précision arbitraire : `java.Math.BigInteger` (non primitif)

Flottants IEEE754 (dont `infinity` ou `NaN`)

Troncature des *calculs* flottants à 64 bits : préfixe `strictfp`
(certains processeurs calculent sur 80 bits puis tronquent à 64)

Opérateurs sur les types primitifs

syntaxe	sémantique
<code>x = y</code>	<code>x</code> prend la même valeur que celle de <code>y</code> (affectation)
<code>x == y</code>	vrai si <code>x = y</code> , faux sinon (test d'égalité)
<code>x != y</code>	faux si <code>x = y</code> , vrai sinon (test de différence)
<code>x < y</code>	<code>x < y</code> (inférieur)
<code>x <= y</code>	<code>x ≤ y</code> (inférieur ou égal)
<code>x + y</code>	<code>x + y</code> (addition)
<code>x - y</code>	<code>x - y</code> (soustraction)
<code>x * y</code>	<code>x × y</code> (multiplication)
<code>x / y</code>	<code>x / y</code> (division)
<code>x % y</code>	<code>x mod y</code> (modulo)
<code>++x</code>	pré-incrémentation de <code>x</code> (incrémente et renvoie la valeur après incr.)
<code>x++</code>	post-incrémentation de <code>x</code> (incrémente et renvoie la valeur avant incr.)
<code>x += y</code>	équivalent à <code>x = x + y</code>
<code>x && y</code>	conjonction logique de booléen
<code>x y</code>	disjonction logique de booléen
<code>!x</code>	négation logique d'un booléen
<code>(T)x</code>	transtype <code>x</code> vers le type <code>T</code> (si possible)

Note sur certains opérateurs

- L'opérateur `=` est associatif à droite et renvoie la valeur de ses opérandes :
dans `int i, j ; i=j=5 ;`, l'instruction `j=5` renvoie 5.
- Sémantique :
 - par valeur pour les types primitifs ;
 - par référence sinon.

Surcharge : exemple de l'affichage

Définition

Le mécanisme de **surcharge** permet d'avoir plusieurs fonctions qui ont le même nom mais pas le même type.

Exemple

```
1 System.out.println(1); // l'argument est un int
2
3 System.out.println(2.5); // l'argument est un double
4
5 System.out.println('A'); // l'argument est un char
```

Portée des variables

- un identificateur est connu dans une zone délimitée par des accolades
- les variables locales sont prioritaires sur les variables globales

Exemple

```
1 class test{
2   public static void main(String[] args){
3     int x = 1;
4     System.out.println(x); // Affiche 1
5     {
6       double x = 2.5;
7       double y = -1;
8       System.out.println(x); // Affiche 2.5
9       System.out.println(y); // Affiche -1
10    }
11    System.out.println(x); // Affiche 1
12    System.out.println(y); // Provoque une erreur
13  }
14 }
```

Instruction conditionnelle if

```
if(test1){  
    partie a executer si test1 est vrai;  
}else if(test2){  
    partie a executer si test1 est faux et test2 est vrai;  
    :  
}else if(testn){  
    partie a executer si testi est faux pour 1 ≤ i < n  
    et si testn est vrai;  
}else{  
    partie a executer si testi est faux pour 1 ≤ i ≤ n;  
}
```

Instruction conditionnelle switch

Si x est de type byte, char, short ou int :

```
switch(x){  
    case a1 :  
        partie a executer si x = a1 ;  
        break;  
    :  
    case an :  
        partie a executer si x = an ;  
        break;  
    default: // Optionnel mais fortement recommande  
        partie a executer si x ∉ {a1, ..., an} ;  
}
```

Boucles while

```
while(test){  
    corps de la boucle a executer si test est vrai;  
}  
  
ou  
  
do{  
    corps de la boucle a executer;  
}while(test);
```

Boucles for

```
for(etat initial de i ; test sur i ; transformation de i){  
    corps de la boucle a executer si test est vrai;  
}
```

La transformation sur *i* est souvent *i++* ou *++i*

On peut définir localement la variable de boucle.

Ex : `for(int i=0;i<10;i++)`.

`for` = cas particulier de `while` ou : la condition initiale, finale et le test ne portent que sur la variable de boucle lorsqu'on sait que la boucle terminera toujours : il faut distinguer `for` de `while` des que possible.

Sortie prématurée : break (version simple)

```
1 boolean egalite(int[] x, int[] y){
2     boolean b = true;
3
4     for(int j=0; j<n;j++){
5         if (x[j] != y[j]){
6             b = false;
7             break;
8         }
9     }
10    return b;
11 }
```

Sortie prématurée : break (version imbriquée : nommage)

```
1 boolean egalite(int[][] A, int[][] B){
2     boolean b = true;
3
4     all_loops:
5     for(int i=0; i<k; i++){
6         for(int j=0; j<n; j++){
7             if (A[i][j] != B[i][j]){
8                 b = false;
9                 break all_loops; // sort des deux boucles en meme temps
10            }
11        }
12    }
13    return b;
14 }
```

Tableaux unidimensionnels de valeurs de type primitif

- Les indices des tableaux commencent à 0. `null` : tableau vide.
- Allocation mémoire avec `new` / désallocation automatique (ramasse-miettes)

```
1 int [] tab = null; // tableau d'entiers vide
2 tab = new int[30]; // tableau d'indices numerotes de 0 a 29
```

- Tout accès hors bornes déclenche une `ArrayIndexOutOfBoundsException`

```
1 System.out.println(tab[33]); // ArrayIndexOutOfBoundsException
```

- Pas de `new` si initialisation explicite :

```
1 int [] tab = {1, 2, 3}; // tab[0] = 1, tab[1] = 2, tab[2] = 3
```

Tableaux multidimensionnels de valeurs de type primitif

```
1 int [][] tab = new int[k][n]; // allocation simultanee de toutes les dimensions
2 for(int i=0; i<k; i++){
3   for(int j=0; j<n; j++){
4     tab[i][j] = 0;
5   }
6 }
```

Questions ?



Conclusion

Aujourd'hui, nous avons vu :

- 1 Informations pratiques
- 2 Introduction à la POO
- 3 Introduction au langage Java
- 4 Syntaxe élémentaire du langage Java

La séance prochaine, nous verrons la structure des classes.