Types de données et objets

VLADY RAVELOMANANA

Licence 1 – S2
Université Denis Diderot (P7).

vlad@liafa.jussieu.fr
LIAFA Bureau 6A13. 175, rue Chevaleret

Plan du cours

1/ Rappels et compléments Java.

2/ Tableaux, boucles et invariants.

3/ Notions élémentaires de complexité.

4/ Récursion.

5/ Structures de données et introduction aux types abstraits de données.

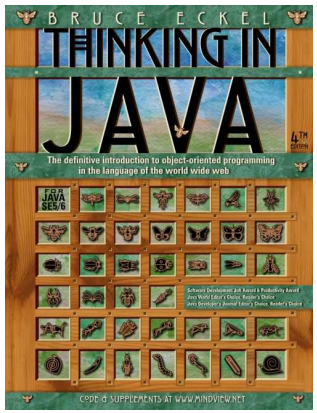
6/ Quelques compléments Java.

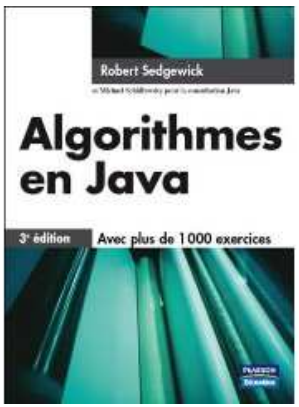
Bibliographie

- La documentation en ligne de SUN
  http://java.sun.com/docs/index.html

- Algorithmes en Java. (de [Bob Sedgewick])

- Un livre complet en ligne et libre: http://mindview.net/Books/TIJ4
  (de [Bruce Eckel]).





1/ Rappels et compléments Java

Plan du chapitre : Rappels et compléments Java

- Structures de contrôles (conditionnelles, boucles, méthodes).
- Structures de données.
  1. Types primitifs.
  2. Tableaux.
  3. Classes-objets (sans héritage).
- Références, créations d'objets, copie d'objets, comparaison.
- Passage de paramètres et références.

Elles sont reprises du langage C. Voici un résumé et des exemples!
L'instruction conditionnelle

La syntaxe générale

```
if (expression boolénne) {
        bloc1
}
else {
        bloc2
}
```

Structures de contrôles

- La condition doit être évaluable en `true` ou `false` et elle est obligatoirement entourée de parenthèses.
- La partie commençant par `else` est facultative.
- Les points-virgules sont obligatoires après chaque instruction.
- Si un bloc ne comporte qu'une seule instruction, les accolades qui l'entourent peuvent être omises.
- Les conditionnelles peuvent bien sûr s'imbriquer.

Exemples

ExempleIf1.java

```
// les entiers a, b, c sont definis et affectes
System.out.print("Le plus petit entre "+a+" et "+b +" est : ");
if (b <a ) {
  System.out.println( a);
}
else {
  System.out.println( b);
}
```

On a aussi l'opérateur conditionnel ( ) ? ... : ...

Exemple de ()? ... : ...

System.out.println((b < a) ? b : a); c = (b < a) ? a-b : b-a ;

L'instruction de choix multiples switch

La syntaxe générale

```
switch (expr) {
    case i:
    case j:
    bloc d'intructions
    break;
    case k:
    ...
    default:
    ...
}
```

- Permet d'aiguiller le programme vers un bloc d'instructions dans le cas d'un choix multiple.
- Dans switch(), la variable ou l'expression, de type caractère ou entier (le sélecteur) permet de différencier les cas.
- L'instruction facultative default (à placer à la fin) permet de traiter toutes les autres valeurs.
- A noter la présence indispensable de break après chaque bloc de cas, pour ne pas traiter les autres cas.

L'exemple des années bissextiles

```
int mois, nbJours;
switch (mois) {
      case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            nbJours = 31;
            break;
      case 4: case 6: case 9: case 11:
            nbJours = 30;
            break;
      case 2:
           if ( ((annee % 4 == 0) && !(annee % 100 == 0)) || (annee % 400 == 0) )
                     nbJours = 29;
            else
                     nbJours = 28;
            break;
       default nbJours=0;
}
```

Instructions itératives

L'itération while

```
while (expression) {
bloc
}
```

- Elle permet de répéter un bloc d'instructions TANT QUE la condition est vraie.
- La sortie de boucle est effectuée aussitôt que la condition est évalué fausse.
- Le test de cette condition est vérifié au début de chaque boucle, si bien que le bloc peut ne pas être exécuté.

Instructions itératives

L'itération while

```
while (expression) {
bloc
}
```

- Elle permet de répéter un bloc d'instructions TANT QUE la condition est vraie.
- La sortie de boucle est effectuée aussitôt que la condition est évalué fausse.
- Le test de cette condition est vérifié au début de chaque boucle, si bien que le bloc peut ne pas être exécuté.

Exemple : que calcule ce code?

```
int max = 100, i = 0, somme = 0;
while (i <= max) {
        somme += i;                    // somme = somme + i
        i++ ;
}
```

L'itération do ... while

Syntaxe

```
do
        bloc
while (expression)
```

Cette itération est toujours effectuée au moins une fois, car le test est effectué à la fin du bloc.

Le même code autrement

```
int max = 100, i = 0, somme = 0 ;
do {
        somme += i;
i++;
} while ( i <= max );
```

L'itération contrôlée for

Syntaxe

```
for ( expression a; expression b ; expression c) {
        bloc
}
```

- C'est la boucle contrôlée, utilisée pour répéter N fois un même bloc d'instructions
- "expression a" précise la valeur initiale de la variable de contrôle (ou compteur)
- "expression b", la condition à satisfaire pour rester dans la boucle
- "expression c", une action à réaliser à la fin de chaque boucle (en général, l'actualisaton du compteur).

L'itération contrôlée for

Syntaxe

```
for ( expression a; expression b ; expression c) {
        bloc
}
```

- C'est la boucle contrôlée, utilisée pour répéter N fois un même bloc d'instructions
- "expression a" précise la valeur initiale de la variable de contrôle (ou compteur)
- "expression b", la condition à satisfaire pour rester dans la boucle
- "expression c", une action à réaliser à la fin de chaque boucle (en général, l'actualisaton du compteur).

Ecrire la même sommation

Que toute à l'heure!

Les opérateurs Java

1. De façon générale, les opérateurs ont la même signification qu'en C (ou langages de script dérivés). On les distingue suivant le nombre de valeurs (les opérandes) sur lesquels ils agissent.
   Opérateurs unaires : $++i$; $i--$;
   Opérateurs binaires : $a+3$; $a+=3$;
   opérateur ternaire : $a<b?b-a:a-b$; (opérateur conditionnel)
2. les opérateurs arithmétiques $+$, $-$, * , % reste de la division entière / division entière si un opérande (diviseur ou dividende) est entier, division décimale sinon
3. Les opérateurs de comparaison Ces opérateurs permettent de comparer deux valeurs et fournissent un résultat de type booléen.
4. $==$ (égal) , $>$, $<$, $>=$, $<=$, $!=$ (différent) Les opérateurs logiques (comme && ET logique, || OU logique) qui agissent sur des valeurs ou des expressions booléennnes, et donnent un résultat vrai ou faux.

opérateurs ... (suite)

5. Les opérateurs d'incrémentation/décrémentation $++$ (incrémentation) , $--$ (décrémentation) attention : pré ou post-incrémentation selon la position de l'opérateur par rapport à la variable
6. Les opérateurs d'affectation L'affectation est $=$ qui permet d'attribuer à une variable.
7. L'opérateur $+$ sur chaînes de caractères. Il n'existe qu'un seul opérateur sur les chaînes de caractères, il s'agit de l'opérateur concaténation dont le symbole est $+$ (ou son dérivé $+=$).
8. Opérateur instanceof Il s'agit de l'opérateur d'appartenance d'un objet à une classe. Il permet de tester si un objet est une instance de la classe passée en paramètre (ou de l'une de ses sous-classes). Exemple :

```
class Personne { ...}
        void estceUnePersonne() {
                if ( martin instanceof Personne ) ....
        }
```

Structures de données

- Types primitifs
- Tableaux
- Classes-objets

Types primitifs: les constantes

Les types de constantes en Java sont :

- constantes numériques On peut forcer l'attribution d'un type de base (voir plus loin) : 4 est un int par défaut; pour avoir un entier long, par exemple, écrire 4L ou (long) 4. 2.45 est un double par défaut; pour obtenir un flottant écrire 2.45f ou (float) 2.45 voir plus loin la conversion de types ou cast
- constantes booléennes : true , false
- constantes caractères : une lettre entourée d'un simple guillemet 'A' elles sont codées sur 2 octets, en unicode (ex 'A' est codé ŏ041)
- constantes chaines : suite de caractères entourées de " ". Ce sont des objets de la classe String (et non pas des tableaux de char) Il s'agit d'un cas particulier : nous sommes dispensés de déclarer et de construire la chaine : Java le fait implicitement à notre place lorsqu'on crée une constante chaine. Les chaines peuvent renfermer des caractères de controle (
  n retour à la ligne,
  t tabulation,
  b retour arrière..)

Les variables

- Comme toujours, une variable posséde un nom (ou identificateur), un type et une valeur. Les autres types de données sont des types par référence vers des objets ou des tableaux. En fait, ce sont de variables destinées à contenir des adresses d'objets ou de tableaux, ce que l'on exprime par : les données non primitives sont manipulées "par référence" alors que les données primitives sont manipulées par valeur.
- Il y a deux catégories de variables en Java.
  1. les types de base ou primitifs, manipulés par valeur et
  2. les types par référence vers des objets (y compris les chaines et les tableaux). Les variables contiennent des adresses d'objets.
  3. Il y a la même différence entre les paramètres transmis par les méthodes. Par exemple, lors d'un appel de méthode avec un paramètre tableau, on transmet en fait (comme en C) l'adresse du tableau ; si cette méthode modifie le tableau reçu en paramètre, elle modifie directement le contenu du tableau dont on lui a transmis l'adresse. De même pour les attributs des objets transmis en paramètres lors d'appels de méthodes.
- En Java, toute variable doit être explicitement typée et initialisée avant d'être utilisée.

Tableaux Java

- Le type des éléments d'un tableau est unique, et est fixé dès sa définition. Si le type des éléments du tableau est simple, Java stocke directement les valeurs dans les éléments du tableau. Mais les éléments peuvent être des objets construits à partir d'une classe standard ou définie par le programmeur. Dans ce cas, on dit que le type des objets est structuré, et les éléments du tableau contiennent toujours des références (pointeurs) aux objets à stocker et non pas les objets eux-mêmes.
- Un tableau en Java a toujours une taille fixée. Si on a besoin d'une stucture de taille variable, il faut utiliser un objet Vector.

Déclaration et construction des tableaux

- Les tableaux ne sont pas véritablement des objets, car ils présentent la particularité d'être des instances d'une classe Array spéciale et cachée. Ceci nous décharge d'une construction explicite. Il suffira d'invoquer le type des éléments du tableau et le mot-clé new pour que le compilateur crée le tableau.
- Déclaration. Pour créer un tableau, on utilise l'opérateur [], comme en C. Dans les déclarations qui suivent, typeTab est le type des éléments, type simple ou type classe déjà définie. Il n'est bien sûr par possible de "mélanger" les types dans un même tableau.

  typeTab[] tab ; déclare une variable pour référencer un tableau d'objets de type typeTab
  typeTab tab[] ; autre notation équivalente.
  new typeTab[taille] ; construit (instancie) un objet tableau de taille fixée, sans l'identifier
  typeTab[] tab = new typeTab[taille] ; le tableau est directement nommé (référencé par l'identificateur tab) et construit avec taille éléments de type typeTab.
  Par exemple String tableau[] = new String[50]; construit un tableau de 50 chaines, numérotées de 0 à 49.

Déclaration et construction des tableaux (suite ...)

- Les éléments du tableau sont indicés (numérotés) de 0 à taille -1, ce qui permet une référence directe à un élément. La dimension (ou taille) du tableau n'est pas forcément fixée immédiatement à l'identification. Mais elle est obligatoirement fixée et figée lors de la construction. On peut fixer la taille avec toute variable (ou expression) entière valorisée. Par exemple :
  typeTab[] tab1, tab2 ; int taille = 10; tab1 = new typeTab[taille] ;
  tab2 = new typeTab[tab1.length*2 + 5] ;
  La propriété length permet de connaitre la taille du tableau, très utile pour parcourir le tableau à l'aide d'une boucle :

Déclaration et construction des tableaux (suite ...)

Le type des éléments est fixé à la définition (pas de mélanges de type permis) Lors de la création du tableau, tous les éléments sont obligatoirement initialisés. Si le programmeur n'effectue pas cette initialisation, Java le fait à sa place, en attribuant des valeurs par défaut : 0 pour les nombres, false pour les booléens, 'slash 0' pour les char, et null pour les objets.

On peut initialiser le tableau en l'affectant directement par une liste de valeurs entre .

Dans ce cas on n'utilise pas new, et le nombre d'éléments est fixé par cette affectation

Exemple:

```
class Tableau {
public static void main(String[] args) {
        int tabEntier[] = {1,5,9};
       String[] joursOuvrables
="lundi","mardi","mercredi","jeudi","vendredi";
        boolean tableau[]={true,false,true};
        System.out.println("sommes nous le "+joursOuvrables[3]+
        tabEntier[0]+" ? c'est "+tableau[1]);
} }
```

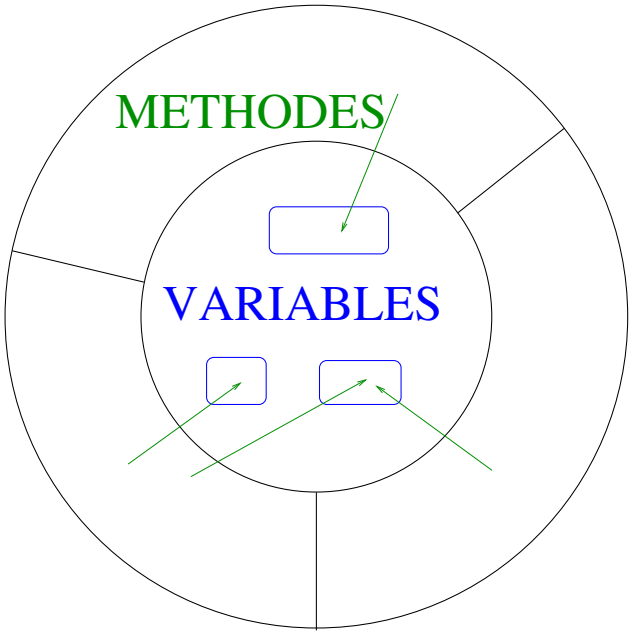
Tableaux d'objets

C'est une erreur fréquente d'oublier de construire tous les éléments du tableau. En cas de tableaux d'objets (prédéfinis ou construits par le programmeur), on sait que chaque élément est initialisé pour pointer sur null, ce qui provoque une erreur du type java.lang.NullPointerException. Par exemple, voici la construction d'un tableau de 100 points :

```
// construire et initialiser le tableau :
Point[] points = new Point[100];
// construire chaque élément du tableau
for (int i=0; i<100 ; i++)
        points[i]= new Point();
// initialiser les coordonnées x et y des points
for (int i=0; i<100 ; i++) {
        points[i].x = (int) (Math.random()*100);
        points[i].y = (int) (Math.random()*100);
}
```

Le principe de la POO : qu'est-ce un "objet"?

- Un objet est un ***ensemble de données*** (variables) et de ***fonctions*** ou ***méthodes*** qui lui sont associées:



- Un objet ***se charge*** des états et valeurs de ses données.
- Le comportement d'un objet est dicté par ses ***méthodes***.

Interface

Autrement dit,

- Afin de s'"assumer", les objets doivent communiquer et avoir des moyens pour cela. Ces moyens incluent
  1. Les données.
  2. Les méthodes.
- Un exemple : l'objet **Complexe** contient la méthode `representationSpherique(double x, double y)`.
- Avec les paramètres dans la méthode
  1. L'objet sait ce qu'il faut faire.
  2. Il sait aussi se servir des informations pour accomplir cette fonction.

Interface

Autrement dit,

- Afin de s'"assumer", les objets doivent communiquer et avoir des moyens pour cela. Ces moyens incluent
  1. Les données.
  2. Les méthodes.
- Un exemple : l'objet **Complexe** contient la méthode `representationSpherique(double x, double y)`.
- Avec les paramètres dans la méthode
  1. L'objet sait ce qu'il faut faire.
  2.

Programmation Objet : un peu de terminologie

- ***Classe.*** On appelle Classe toute description d'objet. C'est un patron pour l'objet. Elle définit ses données et ses méthodes.

  Un classe est donc un type de donnée particulier (par exemple la Classe Complexe)

- ***Instance.*** On appele instance, un objet créé à partir d'une classe (par exemple l'Instance $1 + \overline{2i}$ est une instance des complexes).

Programmation Objet : un peu de terminologie II

- ***Classe*** = définitions pour des données (variables) + fonctions (méthodes) agissant sur ces données
- ***Objet*** = élément d'une classe (instance) avec un état
- une méthode ou une variable peut être
  1. de classe = commune à la classe ou
  2. d'instance = dépendant de l'instance.

La définition d'une classe

La définition d'une classe comporte donc un certain nombre d'étapes.

- la déclaration de variables. Ces variables peuvent être initialisées explicitement et si elles ne le sont pas, elle reÃ§oivent une valeur par défaut.
  Parmi ces variables, on peut distinguer :
  1. les variables de classe : elles portent la qualification ***static*** dans leur définition. De telles varaibles seront caractéristiques de la classe, pas des objets l'instanciant. Toutes les instances de la classe partageront une telle variable.
  2. les variables d'instances (ou d'objets) : ne portant pas le qualificatif précédent, chaque exemplaire construit sur le modèle fourni par la classe, possédera un exemplaire personnel de chacune d'elles. L'ensemble des valeurs de ces différentes variables pour un objet donné en constitueront l' état ;
- idem pour les méthodes : méthodes de classe (static) et méthodes d'instance.

Données et méthodes statiques

- Normalement, rien n'existe réellement avant de créer un objet avec ***new***

- il peut être intéressant d'avoir une zone de stockage pour les données spécifiques qui soit indépendant du nombre d'instances créées. d'où les données ***static***.

Exemple

```
public class Cercle {
        double static PI= 3.14116 ;
        double rayon ;
        public Cercle(double r) { this.rayon = r ;}
        public double aire() {
                return PI*rayon*rayon ;
        }
        ...
}
```

Plus qu'une définition

La définition d'une classe reflète :

Plus qu'une définition

La définition d'une classe reflète :

plusieurs aspects

1/ elle définit un ***nouveau type***;

2/ elle réalise une implantation en définissant à la fois la ***structure d'un objet*** et son ***comportement*** et ses ***fonctionnalités*** ;

3/ elle définit un ***modèle*** permettant de créer de nouveaux objets qui en constituent des instances.

Programmation Objet : la philosophie

- Tout est objet.
- Modélisation naturelle et bien hierarchisée.
- Architecture créateur/utilisateur de classe.
- Très bien adaptée pour les grandes applications à condition de bien penser (et donc définir) les objets et leurs fonctionnalités.

L'Objet Date comme exemple d'architecture créateur/utilisateur

Du point de vue créateur de la classe.

```
class Date {
        int j,m,a;
        void afficherDate(){
                System.out.print(j+”.”+m+”.”+”a”);
        }
}
```

L'Objet Date comme exemple d'architecture créateur/utilisateur

Du point de vue créateur de la classe.

```
class Date {
        int j,m,a;
        void afficherDate(){
                System.out.print(j+”.”+m+”.”+”a”);
        }
}
```

Du point de vue utilisateur de la classe

```
public static void main(String []args){
        Date d = new Date();
        d.j = 31;
        d.m = 2;  /⋆ Erreur donc ⋆/
        d.a = 2010;
        d.afficherDate();
}
```

Etapes de développement d'un projet orienté Objet

Il existe 3 étapes fondamentales :

- ***La phase conceptuelle.*** Le cahier de charges (donné en amont) est étudié. On définit les objets (et leurs responsabilités).

- ***La phase spécification.*** On définit les interfaces entre les objets : les objets doivent communiquer entre eux; on définit les méthodes.

- ***La phase implémentation.*** On développe, on code, on cherche les ***bugs*** et on corrige.

Etapes de développement d'un projet orienté Objet

Il existe 3 étapes fondamentales :

- ***La phase conceptuelle.*** Le cahier de charges (donné en amont) est étudié. On définit les objets (et leurs responsabilités).
- ***La phase spécification.*** On définit les interfaces entre les objets : les objets doivent communiquer entre eux; on définit les méthodes.
- ***La phase implémentation.*** On développe, on code, on cherche les ***bugs*** et on corrige.

Important.

Une seule phase mal conçue entraînera plusieurs aller-retours entre les phases (perte de temps).

Java et les Objets

- Les objets sont manipulés par des références.
  `Eleve Toto;`

- Les identificateurs des objets sont des références. Elle est ici créée mais on peut pas s'en servir

- Les objets sont instanciés par l'opérateur ***new*** (dynamiquement). A ce moment, on a donc créé une instance de l'objet.
  `Toto = new Eleve();`
- On a associé cette référence à une instance de l'objet. On peut maintenant l'utiliser.

- Bien entendu, plusieurs instances d'un même objet peuvent co-exister.