Cours n° 2

Données en Java

Sommaire

1. **Types**
   - Notion de types
   - Types natifs/prédéfinis/primitifs
2. **Variables**
   - Notion de variables
   - Extraits de règles BNF
   - Déclaration de variables
   - Variables et constantes
   - Visibilité et masquage
   - Assignations
3. **Expressions**
   - Opérateur et expressions
   - Règles d'évaluation
   - Précédence/priorité des opérateurs
   - Opérateurs arithmétiques/relationnels/logique/assignation

Notion de type dans les langages

Information sémantique associée à une donnée

Le type d'une donnée est le moyen de définir dans un langage informatique

- le codage à utiliser/utilisé pour la représentation interne de la donnée
- un ensemble de valeurs possibles pour la donnée
- un ensemble d'opérations applicables aux données de ce type

Exemple

short : type des entiers signés (positifs et négatifs)

- représentation interne sur 16 bits en **Java** permettant de coder resp. 2^8=256 (resp. 2^{16}=65 5366) valeurs distinctes construites par un codage i) binaire pur pour les entiers positifs et ii) « par complément à 2 » pour les entiers négatifs
- ensemble des valeurs entières de -128..127 (resp. -32768..32767)
- opérations arithmétiques (+, -, *, /, …)

Remarque : Dans les LOO, une classe est conceptuellement un type de données

Types natifs/prédéfinis/primitifs (1/4)

Dynamique de codage des types entiers

Taille (octet)	Valeurs min..max
1	-128..127
2	-32 768..+32 767
4	-2 147 483 648 .. +2 147 483 647
8	-9 223 372 036 854 775 808 .. +9 223 372 036 854 775 808

TENTIER*_MIN** (valeur entière minimum) du type entier ***TENTIER
TENTIER*_MAX** (valeur entière maximum) du type entier ***TENTIER

Types natifs/prédéfinis/primitifs (2/4)

Précision des types flottants et valeurs absolues minimum et maximum

Identificateur	Taille (octet)	Précision (chiffres)	Valeurs absolues min..max
float	4	7	1.40299846 x10^{-45} 3.40282347x10^{38}
double	8	15	4.9406564584124654 x10^{-324} 1.7976931348623160 x10^{308}

Valeurs données (à titre indicatif) pour un codage en virgule flottante **norme IEEE 754**

Types natifs/prédéfinis/primitifs (3/4)

Type des caractères

Identificateur	Taille (octet)	Nb valeurs distinctes	Norme
char	2	65 536	Unicode (\u0000 à \uffff)

Remarque : Tables de transcodage ASCII/ANSI ou Unicode des caractères

Unicode

- Objectif : représenter tous les systèmes d'écriture
- Intérêt : universalité en Gestion Electronique des Documents (GED) et pour l'édition de programmes
- Inclut les 256 codes ASCII/ANSI (1er octet nul)

Types natifs/prédéfinis/primitifs (4/4)

Type des booléens

Identificateur	Taille (octet)	Valeurs logiques
boolean	1	false true

Notion de variable (1/2)

Moyen informatique de manipuler une donnée/information : associer la donnée à une variable

Une **variable** est une **entité symbolique** représentant :

i) sur un plan conceptuel, une donnée

ii) sur un plan informatique, un **emplacement de stockage** (une ou plusieurs cellules) en **mémoire** repéré par **l'adresse** de la **première cellule** mémoire **utilisée**

La **valeur** est l'**information stockée** dans cet **emplacement**
L'**affectation** est l'opération (=) par laquelle une valeur est attribuée à une variable

Les 3 attributs (informatiques) d'une variable

- **Identificateur** désignant la ***donnée*** dans le code source auquel est associé une ***adresse*** de stockage en mémoire
- **Type** définissant les modalités de ***représentation interne*** et les ***opérations*** applicables
- **Valeur** (littérale)

Notion de variable (2/2)

Etapes de création d'une variable (implantation en mémoire)

1) **Déclaration** : du type de la variable
   Effet : **réservation** d'un **emplacement libre** en mémoire
   (nécessaire à la représentation interne d'une variable du type donné)
   La variable, par son emplacement mémoire, correspond à une **adresse mémoire** de **début** d'implantation (@)

   Remarque :
   Selon les langages, cette adresse mémoire est manipulable ou non
   En Java**,** les adresses des variables ne sont pas manipulables. La mémoire n'est pas accessible/modifiable par ces adresses

2) **Initialisation** : association/assignation effective de la valeur à la variable (valeur littérale exprimée dans un format autorisé)
   Effet : **mise à jour** de l'emplacement mémoire réservé à la variable par la **représentation interne** de la **valeur** correspondant au codage lié au type de la variable

Extraits de règles syntaxiques BNF

variable_declaration ::= { modifier } type variable_declarator { "," variable_declarator } ";"

type ::= type_specifier { "[" "]" }

type_specifier ::= "boolean" | "byte" | "char" | "short" | "int" | "float" | "long" | "double" | class_name | interface_name

variable_declarator ::= identifier { "[" "]" } ["=" variable_initializer]

modifier ::= "public" | "private" | "protected" | "static" | "final" | "native" | "synchronized" | "abstract" | "threadsafe" | "transient"

variable_initializer ::= expression | ("{" [variable_initializer { "," variable_initializer } [","]] "}")

expression ::= numeric_expression | testing_expression | logical_expression | string_expression | bit_expression | casting_expression | creating_expression | literal_expression | "null" | "super" | "this" | identifier | ("(" expression ")") | (expression (("(" [arglist] ")") | ("[" expression "]") | ("." expression) | ("," expression) | ("instanceof" (class_name | interface_name))))

identifier ::= "a..z,$,_" { "a..z,$,_,0..9,unicode character over 00C0" } *etc…*

Limitation à la présentation des règles syntaxiques les plus usuelles

Déclaration de variable (1/2)

<déclarationVariable> ::= <type> <identificateur>[=<valeur>] {, <déclarationVariable>} ;

type de la variable déclarée
identificateur de la variable
valeur : valeur **littérale** ou **expression** de même type que la variable

Exemples :

```
byte age;
float totalTHT;
char c;
int nbClients=0, nbFournisseurs;
```

Remarque :
La déclaration est **obligatoire** avant toute utilisation de la variable

Sans déclaration préalable :
erreur signalée par le compilateur

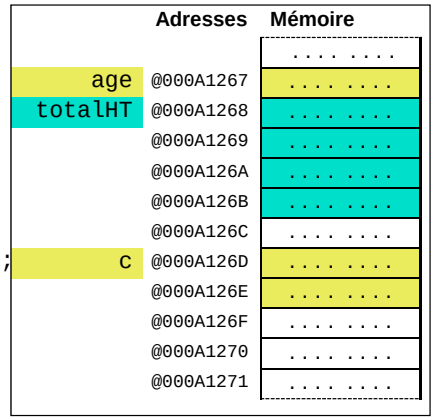


Illustration des déclarations

.... état courant de la mémoire

Déclaration de variable (2/2)

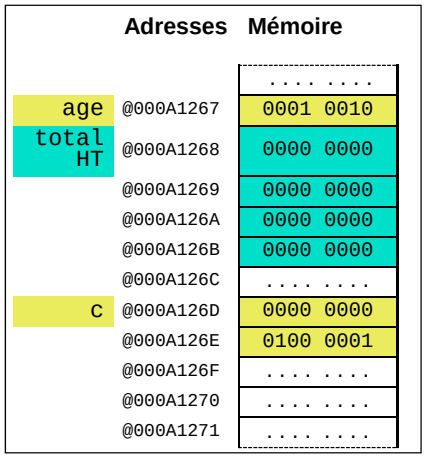
Exemples :

```
byte age=18;
float totalTHT=0.0;
char c='A';
…
float pi=22./7.;
```

Conseil

Initialiser explicitement les variables

même en cas d'initialisation automatique (usuellement à 0) des compilateurs



Sémantique des identificateurs

Le langage informatique comme tout autre langage comporte les aspects : lexical, syntaxique et **sémantique**.

La variable a le rôle sémantique de la donnée/information qu'elle représente.

- Le **rôle sémantique** sera **précisé** par **l'identificateur** qui lui est attribué
- Le **choix** de l'identificateur est donc **très important**, il nécessite un **temps de réflexion**

Conseil

L'identificateur *doit être* ***explicite*** *et de* ***longueur acceptable***
Exemple de règle d'écriture des identificateurs :
Commencer par une minuscule, passage en majuscule en début de mot
`nbClients, noCompteClient, tauxTVA, prixUnitaire`
Les variables de boucles (usuelles) : `i, j, k,…`

Valeurs littérales

Entiers – Notations applicables aux variables de type **entier**

- **Notation décimale**

```
i=0; i=-128; i=128; …
```

- **Notation hexadécimale** (0xH…)

Symboles hexadécimaux : 0..9, A, B, C, D, E, F (minuscules autorisées)

Notation débutant par **0x** suivie des symboles hexadécimaux

```
short nbMois=0xC;      // ≡ short nbMois=12;
```

Flottants – Notations applicables aux variables de type **flottant**

- **Notation décimale**

```
r=0.; s=.0;            // ≡ r=0.0; s=0.0;
f=.234; g=234.;        // ≡ f=0.234; et g=234.0;
x=39.276;
```

- **Notation exponentielle**

```
f = 24.3e-4;           // f= 0.00243  (24.3*10-4 )(E accepté)
g = 1.5123E3;          // g=1512.3
```


Variables et constantes

Variable : donnée de nature **modifiable**
Constante : donnée de nature **non modifiable**

Dans les langages, les **modifieurs** permettent de définir certaines **caractéristiques** des **données** lors de leur déclaration

Modifieurs des règles de modification

Modifieur	Exemple
const	`const double g=9.81;`
final	`final double g=9.81;`

Remarque :

L'initialisation de la donnée constante ne peut être différée de sa déclaration. Une fois déclarée et initialisée, la constante ne peut plus être modifiée par programme.

Erreur signalée par le compilateur en cas de tentative de modification.

Portée/visibilité des variables/constantes

C'est la portion de code dans laquelle une variable est utilisable/accessible/référençable (par son nom)

Variables locales à un bloc

- Variables **déclarées** à l'**intérieur** de ce **bloc**
- Elles ne sont **accessibles** qu'à l'**intérieur** de ce **bloc**

Variables globales

- Variables **déclarées** en **dehors** de **tout bloc**
- Elles sont **accessibles** dans **l'ensemble** du **programme**

Conseil

Pour des raisons d'intégrité des données :
ne pas utiliser de variables globales

Visibilité et masquage des variables

```
// Déclarations globales
int z=0;
...
int main()
{  // Bloc1
   // Déclarations locales Bloc1
   int x,y;
   …
   {  // Bloc2
      // Déclarations locales
      int y;
      ... x ...
      ... y ...
      ... z ...
   }  // Fin Bloc2
   ... y ...
   ... z ...
} //  Fin Bloc1
```

```
// MonProgramme.cpp
int z=0;
...
int main()
{
   // Bloc1 (main)
   int x,y;
   …
   {
      // Bloc2 (imbriqué)
      int y;
      ... x ...
      ... y ...
      ... z ...
   }
   ... y ...
   ... z ...
}
```


Affectation

<Affectation> ::= <identificateur> = <valeur> ;

identificateur : identificateur de la variable (***l-value****/left-value*)
valeur : valeur **littérale** ou **expression** de même type que la variable (***r-value****/right-value*)

Exemples :

```
int age;
float totalTHT;
char c;
age=18;
```

Remarque :
L'initialisation est **essentielle** avant toute utilisation de la variable

Sans initialisation préalable :
- *Généralement, pas d'erreur signalée par le compilateur*
- *Bug a priori inévitable (initialisation avec l'état de la mémoire)*

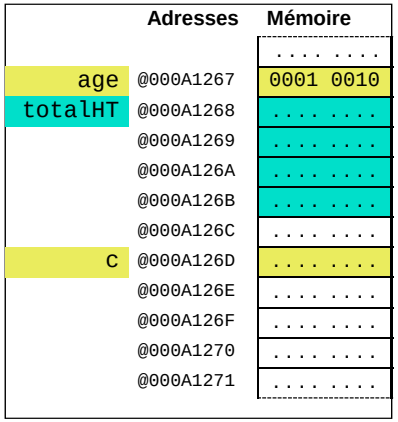


Illustration de l'affectation

Opérateurs et expressions

Opérateur

Symbole indiquant une **opération** à effectuer entre 1, 2 ou plusieurs **opérandes** et retournant un **résultat typé** dépendant de l'opérateur et du type des opérandes.

Un **opérateur** est caractérisé par son **arité** (nombre d'arguments)

Lorsque l'arité est supérieure à 1, l'opération est en notation *infixée*

Exemples : moins *unaire* : `-10`
moins *binaire* : `10-7`

Expression

Combinaison de **littéraux**, de **variables**, de **constantes**, de **fonctions**, d'**expressions** et d'**opérateurs**.

L'expression exprime un calcul, une manipulation de caractères ou un test de données

Exemples :
```
2+3*x+1
"C++ " + "et Java"
((a<b && a>0) || (a>b  && a==0))
```


Classification usuelle des opérateurs

Suivant les types des opérandes

- **Arithmétiques** associés à une/des opérandes de type entiers/réels
  Résultat : entier/réel
- **Relationnels** associés à deux opérandes de même type natif
  Résultat : booléen
- **Logiques** associés à une/des opérandes de type booléen
  Résultat : booléen

Exemples :
```
2 + 3 * x   // avec x de type flottant
(b+c != 0)
(b1 || (i==0)) // avec b1 de type booléen et i de type entier
```

Règle d'évaluation des expressions (1/2)

Interprétation d'une expression
2 + 3 * x // ≡ (2+3) * x ? ou ≡ 2 + (3*x) ?

Règle d'évaluation

i) Fondée sur l'**ordre de précédence/priorité** des **opérateurs**
un nombre arbitraire fixant pour deux précédence distinctes l'ordre d'évaluation
l'expression correspondant à l'opérateur de plus forte précédence est évalué en premier

ii) Fondée sur l'**associativité de l'opérateur** : gauche (G) ou droite (D)
Associativité gauche : a ♥ b ♥ c = (a ♥ b) ♥
Associativité droite : a ♥ b ♥ c = a ♥ (b ♥ c)

iii) Règle d'évaluation en cas d'**égalité de précédence** : de gauche à droite
a ♥ b ♣ c ≡ (a ♥ b) ♣ c // ♥ et ♣, deux opérateurs de même précédence
// Ordre d'évaluation : 1) ♥ et 2) ♣

iv) Pour un opérateur (binaire) donné, l'**ordre d'évaluation** est (généralement) celui : 1) du premier opérande et 2) du deuxième opérande

Règle d'évaluation des expressions (1/2)

Parenthésage explicite

Pour une **évaluation** des expressions **différente** des **règles d'évaluation** on utilise le **parenthésage explicite** (qui force l'ordre d'évaluation de l'expression)
Les expressions entre parenthèses sont évaluées en premier

Remarque :

La **règle d'évaluation** n'est **pas un principe standard** dans les **différents langages**

i) Les **précédences**, ii) les règles **d'associativité** et iii) la règle **d'évaluation** des opérateur **binaires** doivent être :
apprises et **connues** pour **chaque langage** et lorsque ceux-ci ne sont pas normalisés pour **chaque compilateur**

Règle d'évaluation des expressions

Conseil

- N'utiliser que les **précédences** d'opérateurs les « plus **usuelles** »
  Définir explicitement la **précédence** des opérateurs
  par les **parenthèses** appropriées, même si elles sont inutiles

Objectif en génie logiciel

- Code source clair
- Plus facile à lire
- Plus sûr
  (moins de risques d'erreurs lorsque l'on modifie des expressions
  pour une maintenance par exemple)

Précédence des opérateurs (1/2)

Niveau de priorité	Opérateur	Rôle
17	::	Sélection de contexte global
17	::	Sélection de contexte de classe
16	.	Sélecteurs de membres
16	[]	Index de tableau
16	()	Appel de fonction
15	instanceof	Appartenance d'un objet à une classe
15	++, --	Incrémentation, décrémentation
15	~	NOT binaire, bit à bit
15	!	NOT logique
15	+, -	Signe plus, signe moins
14	()	Conversion de type, casting
13	*, /, %	Opérateurs multiplication, division, modulo
12	+, -	Opérateurs arithmétiques
11	<<, >>, >>>	Décalage bit à bit
10	<, <=, >, >=	Opérateurs relationnels

Précédence/priorité des opérateurs (2/2)

Niveau de priorité	Opérateur	Rôle
9	==, !=	Egalité, Inégalité
8	&	AND bit à bit
7	^	XOR bit à bit
6	|	OR bit à bit
5	&&	AND logique
4	||	OU logique
3	?:	IF arithmétique
2	=, *=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=	Opérateurs d'assignation
1	,	Opérateur virgule (succession)

Opérateurs arithmétiques

Par ordre décroissant de précédence

Opérateur	Opération
*	multiplication
/	division
%	modulo
+	addition
-	soustraction

modulo : reste de la division entière
10 % 3 égal 1

Exemple :

```
int x=20, y;
y = 2 + 3 * x;
```

Evaluation. Quelle est la valeur de y ?

// ≡ `(2 + 3) * x` ou ≡ `2 + (3 * x)` ?

Opérateurs relationnels

Par ordre décroissant de précédence

Opérateur	Opération
<	infériorité
<=	infériorité ou égalité
>	supériorité
>=	supériorité ou égalité
==	égalité
!=	inégalité

Exemple : `((a<b && a>b) || (a>b && a==0)`

Evaluation ? Pour `a=2` et `b=10`

Opérateurs logiques

Par ordre décroissant de précédence

Opérateur	Opération
!	NON logique
&&	ET logique
||	OU logique

^ : xor ou exclusif

Tables de vérités

x	y	! x	x & y	x | y	x ^ y
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Exemple : `(a==0) && ((a<b && a>0) || (a>b && a==0))`
Evaluation ? Pour `a=2` et `b=16`

Opérateurs d'assignation

Par ordre décroissant de précédence

Opérateur	Opération
=	assignation
♥=	assignation composée

Définition de l'assignation composée

```
a ♥= b; // ≡ a = a ♥ b
```

Exemple :

```
int i, a, b;
i = a = b = 3;    // associativité droite
                  // de l'opérateur =
                  // ≡ i = (a = (b = 3));
```


Exemple d'évaluation d'expression

Exemples

Evaluer l'expression (syntaxiquement correcte) 2 * 5 + 20 % 7 / 5 - 12