Cours d'introduction à JAVA
UE LXTMI - Section IPCM

Philippe ESLING - Pascal MANOURY

Université Pierre et Marie Curie
UFR d'ingénierie et informatique

1er semestre - 2009 - 2010

Chapter 1

Introduction à Java

1.1 De l'esprit de la programmation

Lorsque nous programmons, nous utilisons l'outil à la fois le plus performant et le plus dénué d'intelligence à notre disposition, l'ordinateur. Car si l'on regarde de plus près, l'ordinateur n'est en fait qu'une sorte de calculatrice extrêmement puissante qui attends nos instructions. Le coeur de l'ordinateur est en effet un processeur ou Central Processing Unit (CPU) qui n'est en mesure de réaliser que les opérations arithmétiques de base (addition, soustraction, multiplication, division) et d'inscrire ou modifier des données dans la mémoire de l'ordinateur. Lorsqu'on parle de processeur de 3 GHz, cela signifie en fait qu'il est capable d'effectuer 3 milliards de ces opérations de base par seconde. Cependant l'ordinateur n'effectuera jamais une opération qui ne lui a pas été explicitement demandée. Nous avons donc à notre disposition un enfant avec un cerveau démesuré auquel il faut apprendre en lui donnant les instructions à suivre étape par étape.

Imaginons que nous demandions à un ordinateur de boire un verre de jus d'orange. Cet exemple permet déja de saisir un point clé de l'art de la programmation qui consiste à savoir imaginer les étapes successives permettant la réalisation d'une opération.

Deux mécanismes essentiel :

1. imaginer quelles étapes successives pour réaliser l'opération (ouvrir la main, prendre le verre, le lever, l'approcher, l'incliner, etc...)

2. prévoir quelles sont les variables nécessaires. (un verre et du jus d'orange)

Cet exemple permet également de saisir le concept de fonctions qui permettent de regrouper de multiples opérations dans un bloc que l'on pourra appeller de manière pratique à n'importe quel moment. Par exemple les instructions précédentes peuvent être regroupées dans une fonction *boire()* que l'on pourra alors appeller à n'importe quel moment sans avoir besoin de réécrire l'intégralité du code et l'ordinateur effectuera à nouveau toutes les instructions contenues dans

cette fonction. Les autres instructions sont également des fonctions et l'on peut descendre dans leur code. Un autre concept fondamental de la programmation sont par la même les variables, celles-ci permettent de conserver des données dans la mémoire de l'ordinateur. Car rappellons-nous que l'ordinateur est une calculatrice extrêmement puissante et c'est donc cette puissance calculatoire que nous essayerons essentiellement de mettre à profit. Grâce aux variables, on peut ainsi conserver une trace de certaines valeurs mais également effectuer des opérations sur celles-ci et les re-stocker à nouveau.

1.2 La syntaxe

1.2.1 Les commentaires

Trois manières de créer des commentaires en JAVA

```
/* ...  */
/* ceci est un commentaire
sur plusieurs lignes
...  qui se termine ici */
//...
int i // ceci est une variable entière
/** et */
```

Ce commentaire est à réserver dans les déclarations en vue d'une documentation automatique

bonnes habitudes Il faut absolument commenter les programmes car ceux-ci sont

- écrit une seule fois
- relu des dizaines de fois

Les commentaires permettent une reformulation explicite de certaines commandes et donc une meilleure compréhension du programme à la relecture.

mauvais commentaire :

```
int zoom=2 // zoom à 2
```

aucun renseignement sur le rôle de zoom et pourquoi 2

commentaire OK :

```
int zoom=2 // valeur par défaut du zoom au démarrage
```

1.2.2 Les identificateurs

Comment nommer :

- les variables
- les méthodes
- les classes

identifiant = "dans {a..z, A..Z, $, _}" < " dans {a..z,$,_,0..9,unicode character over 00C0}" >

- des caractères a..z ou A..Z
- des chiffres de 0 à 9
- les _ et $
- et les caractères Unicode supérieure à 0X00C0. caractères nationaux tels que: Ç, ü, ...

exemples d'identificateur valides :

```
dateDeNaissance
$valeur_system
ISO9000
```

exemples d'identificateur non-valides:

```
ça // Ç comme premier caractère
9neuf // 9 comme premier caractère
note# // # pas au dessus de 0X00C0
long // OK mais c'est un mot réservé !
```

listes des mots réservés:

abstract	boolean	break	byte	byvalue	case	catch
char	class	const	continue	default	do	double
double	else	extends	false	finally	float	for
goto	if	implements	import	instanceof	int	interface
long	native	new	null	package	private	protected
public	return	short	static	super	switch	synchronized
threadsafe	throw	transient	true	try	void	while

Donc un identificateur ne doit pas être un mot de cette liste.

bonnes habitudes

ne pas utiliser le $ et le _ si vous devez utiliser des librairies en C

ne pas utiliser le $ en première position

séparer les noms composés en capitalisant la première lettre des noms à partir du deuxième mot.

acceptable	**conseillé**	**déconseillé**
nom_de _methode	nomDeMethode	$init
ceci_est_une_variable	ceciEstUneVariable	_type

1.2.3 Les litéraux

Les litéraux définissent explicitement les valeurs sur lesquelles travaillent les programmes JAVA. trois catégories de litéraux:

- les booléens

- les nombres,
- les caractères.

1.2.3.1 booléens

Deux valeurs possibles pour les booléens :

- false
- true

initialiser des variables booléennes (Attention: ne peut être assimilée à un 0 ou 1 comme dans le cas des langages C ou C++)

```
boolean resultatAtteint = false;
boolean continuer = true;
```

1.2.3.2 entiers

trois formats:

- décimal
- octal
- hexadécimal

Attention ! en décimal ne commence jamais par un zéro

```
int nbrDeMois = 12;
```

Pour en octal, précédé d'un zéro.

```
int nbrDeDoigts = 012; // ==10 en décimal
```

en hexadécimal, précédé d'un 0x ou 0X.

```
int dixHuit= 0x12;
```

entiers déclarés littéralement = int sur 32 bits.
entier long en lui faisant succéder un L (64 bits).

1.2.3.3 flottants

- une mantisse
- éventuellement exposant en puissance de 10.
- obligatoirement un point décimal ou un exposant

Exemple de flottants sans partie exposant:
2.52 52.0 .001
Exemple de flottants avec partie exposant:
2E02E32E-3.2E32.5E-3
Les nombres flottant déclarés = float sur 32 bits.
flottant double précision, succéder un D (64 bits)
flottant float , succéder un F (32 bits)
3.14F3.14159D

1.2.3.4 caractères

caractère / deux apostrophes (quotation simple):
`'x''a''4'`
l'ensemble des caractères Unicode (16bits)

caractère	séquence
continuation	\
nouvelle ligne	\n
tabulation	\t
retour arrière	\b
retour chariot	\t
saut de page	\f
backslash	\\
apostrophe	\'
guillement	\"
caractère octal	\377
caractère héxadécimal	\xFF
caractère unicode	\uFFFF

1.2.3.5 chaînes de caractères

- suite de caractères entourée de guillemets
- type String.
- constitue une classe
- $\neq$ un tableau de caractères

la chaine de caractère	le résultat si on l'imprime
""	
"\""	"
"texte sur 1 ligne"	texte sur 1 ligne
"texte sur 1 ligne \défini sur 2 lignes"	texte sur 1 ligne défini sur 2 lignes
"texte sur 1 ligne \n écrit sur 2 lignes"	"texte sur 1 ligne écrit sur 2 lignes"

1.2.4 La déclaration des variables

obligatoirement une déclaration

Le compilateur : vérifications de compatibilité de type dans les expressions, visibilité de la variable, etc.

augmenter la qualité des programmes :

- détecter des erreurs à la compilation

- détecter au moment de l'exécution

préfixer le nom de la variable par son type
```
int i;
int i, j, k;
```
modifier la définition de la variable
final = une constante.
```
final int NbreDeRoues = 4;
final float pi = 3.14159;
```

1.2.4.1 Type simple vs composé

Les types simples :

- les booléens
- les entiers
- ...

types composés, construits à partir d'autres types

- les vecteurs,
- matrices,
- classes,
- interfaces
- ...

1.2.4.2 booléens

définit une variable dont le contenu sera vrai ou faux.
assignée

- false
- true
- résultat d'une expression logique

```
boolean voitureArretee = true;
```

1.2.4.3 entiers

quatre types d'entier:
byte
short
int
long

type entier	nombre de bits	exemples
byte	8	byte nbrEnfant;
short	16	short volumeSon;
int	32	int nbrCheveux;
long	64	long detteSecu;

assignée

- les littéraux entiers
- le résultat d'une expression entière.

1.2.4.4 flottants

deux types de flottant:
float
double

type flottant	nombre de bits	exemples
float	32	float ageMoyen;
double	64	double pi;

assignée

- les littéraux flottant
- résultat d'une expression flottante.

1.2.4.5 caractères

une valeur de l'ensemble Unicode de caractères
assignée

- les littéraux caractère
- le résultat d'une expression caractère

// le séparateur est un tabulateur
char separateur="\t";
Le caractère en JAVA est représenté sur 16 bits.
Il ne sert qu'à conserver la valeur d'un seul caractère.
Les chaînes sont représentées par la classe String.

1.2.4.6 vecteurs et matrices

post-fixer le type ou la variable par [ ] :

```
int i[]; // vecteur d'entiers
int[] j; // j à le même type que i
char motsCroises[][]; // une matrice de caractères
```

pas contraints au moment de la déclaration.
L'allocation au moyen d'une méthode new

1.2.5 visibilité des variables

La variable est visible à l'intérieur du bloc où elle est définie. un bloc est définit comme l'ensemble des instructions comprises entre deux accolades { }

```
class Test { // debut de test
public static void principal(String args[])
{ // debut de principal
float x
...
} // fin de principal
public void methodeA ()
{ // debut de methodeA
char c
...
} // fin de methodeA
} // fin de principal
```

1.2.5.1 redéfinition des variables

si dans un bloc on redéfinit une variable existant dans un bloc supérieur, cette nouvelle variable masque la variable supérieur à l'intérieur de ce bloc (donc aussi pour ses sous-blocs)

Algorithm 1.1 - Redéfinir des variables

```
class Test
{
    public static void testEchange()
    {
        float echange;
        int a,b;
        ...
        if (a < b) then
        {
                int echange;
                echange = a;
                a = b;
                b = echange;
        }
    }
}
```

Test principal a,b,echange bloc du if echange

- réservée à des variables temporaires
- indice d'une boucle.

1.2.6 Opérateurs

regroupés par type d'opération:

- numérique,
- de comparaison,
- logique,
- sur les chaînes de caractères,
- de manipulations binaires.

le nombre d'opérandes :

- unaire,
- binaire
- ternaire.

évaluées de la gauche vers la droite,
une table de précédence, la priorité entre les opérations.
x=z+w-y/(3*y^2)
on a = + / () par ordre de lecture

La table de précédence => () / + - =.
Dans la (), on a * ^ par odre de lecture,
l'ordre d'exécution est aussi *^.
table de précédence (de la plus haute à la plus basse).
Les opérateurs de même niveau depuis la gauche.

.	[]	()		
++	–	!	~	instanceof
*	/	%		
+	-			
<<	>>	>>>		
<	>	<=	>=	
==	!=			
&				
^				
&&				
||				
?:				
=	op=			
,				

1.2.6.1 l'assignation

L'assignation assigne l'expression de droite, à l'expression de gauche (une variable).

opérateur binaire qui modifie son opérande gauche.

```
j = 2; // expression d'assignation d'un litéral
i = j * 3; // expression d'assignation d'une expression
```

Cette dernière expression a pour résultat i=6
effets de bord: ~, ++ , –
dans une même expression rend confus la lecture.
décomposer les expressions un seul opérateur avec effet de bord.

opérateurs unaires	action	exemple	équivalence
-	négation	i = -j;	
++	incrémentation de 1	i++;	i = i + 1;
–	décrémentation de 1	i–;	i = i - 1;

++ et – peuvent préfixer ou postfixer la variable.
++i; // est équivalent à i++;

1.2.6.2 expression numérique

opérateurs binaires	action	exemple
+	addition	i = j + k;
+=		i += 2; ⇔ i = i + 2;
-	soustraction	i = j - k;
-=		i -= j; ⇔ i = i - j;
*	multiplication	i = j * k;
*=		i *= i; ⇔ i = i * i;
/	division	i = j / k;
/=	(tronque si entiers)	i /= 10; ⇔ i = i / 10;
%	modulo	i = j % k
%=		i %= 2; ⇔ i = i % 2;

la division par 0 et le modulo par 0 génère une exception à l'exécution. Les opérations dont la représentation dépasse le maximun du type de l'entier débordent dans les entiers négatifs. Pour les flottant, les opérations même sémantique. ++ et – on ajoute 1.0. Le % modulo appliqué au flottant prend le sens de la division dans les entiers. la division par 0 et le modulo par 0 génère la valeur inf. Les opérations dont la représentation dépasse le maximun du type génère la valeur inf. Les débordements vers l'infinement petit génère 0.

Le programme TestNumber

Algorithm 1.2 - TestNumber

```
class TestNumber
{
    public static void main (String args[])
    {
        int i=1000, j=1000;
        float x=1, y=1;
        for (int k = 0; k < 100; k++)
        {
            i *= 10;
            j /= 10;
            x *= 10000;
            y /= 10000;
            System.out.println("\ni="+i+" j="+j+" x="+x+" y="+y);
        }
    }
}
```

i=10000 j=100 x=10000 y=0.0001
i=100000 j=10 x=1e+08 y=1e-08
i=1000000 j=1 x=1e+12 y=1e-12
i=10000000 j=0 x=1e+16 y=1e-16
i=100000000 j=0 x=1e+20 y=1e-20

i=1000000000 j=0 x=1e+24 y=1e-24
i=1410065408 j=0 x=1e+28 y=1e-28
i=1215752192 j=0 x=1e+32 y=1e-32
i=-727379968 j=0 x=1e+36 y=1e-36
i=1316134912 j=0 x=Inf y=9.99995e-41
i=276447232 j=0 x=Inf y=9.80909e-45
i=-1530494976 j=0 x=Inf y=0
i=1874919424 j=0 x=Inf y=0
...

1.2.6.3 Opérateurs relationnels

tester tous les types avec une relation d'ordre:
entiers,
réels,
caractères, ...

opérateurs relationnels	action	exemple
<	plus petit que	x < i;
>	plus grand que	i > 100;
<=	plus petit ou égal à	j <= k;
>=	plus grand ou égal à	c >= 'a';
==	égal à	i == 20;
!=	différent de	c != 'z';

ATTENTION relation d'ordre sur les flottants,
x==y peut être vrai
y<x || y>x soit forcement vrai.
l'opérateur == confondu avec =

1.2.6.4 opérateurs logiques

opérandes booléennes.
Les trois opérateurs de base sont:

- ! la négation
- && le ET
- || le OU

p	q	!p	p && q	p || q
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

L'évaluation de l'expression logique est stoppée dès que sa valeur de vérité peut être assurée.

p1 && p2 && ... & pn

l'évaluation est stoppée si pi est évaluée à faux.

Dans le cas:

p1 || p2 || ... || pn

l'évaluation est stoppée si pi est évaluée à vrai.

permet d'éviter une erreur

i!=0 && x > (y/i);// y/i n'est pas évalué si i égal 0

JAVA possède aussi une expression ternaire de la forme:

```
p?e1:e2;
```

Si p est vrai alors l'expression e1 est évaluée sinon e2 est évaluée.

l'assignation étant une expression, il est possible de l'utiliser comme une instruction if.

```
if (p) e1; else e2; // est équivalent à p?e1:e2;
```

1.2.6.5 opérateurs sur les chaînes de caractères

Concaténation +

assignation +=

Les opérandes sont automatiquement converties.

```
System.out.println("\ni="+i+" j="+j+" x="+x+" y="+y);
```

opérateurs concaténation	action	exemple
+	concaténation	"conca" + "ténation";
+=	ajoute à la fin	s += " à la fin";

1.2.6.6 opérateurs de manipulation binaire

manipulations &, |, ^

décalages binaires (deux opérandes):

- valeur binaire sur laquelle on effectue le décalage
- le nombre de décalages à effectuer. On a donc

i>>k; // decaler i vers la droite de k bits avec son signe

i<<k; // decaler i vers la gauche de k bits

i>>>k; // decaler i vers la droite de k bits sans signe

version assignée.

opérateurs de manipulation	action	exemple
<<	décalage à gauche	i << n
>>	décalage à droite signé	i >> 4
>>>	décalage à droite non signé	i >>> 2
<<=	décalage à gauche assigné	k <<= n
>>=	décalage à droite signé assigné	k >>= n
>>>=	décalage à droite non signé assigné	k >>>= n

1.2.6.7 opérateur d'allocation

L'opérateur d'allocation new

- créer un objet instance d'une classe
- créer et de fixer les dimensions des vecteurs

déclarer un vecteur, pour fixer sa taille:

```
// i un vecteur de 100 entiers
int i[] = new int [100];
// c une matrice de 10 par 10
int c[][] = new char[10][10];
```

Les indices des vecteurs commencent à 0

```
i[0] = 1; //la première position de i initialisée à 1
i[9] = 10; //la dernière position de i initialisée à 10
c[0][0]='a'; // première case de c
c[9][9]='z'; // dernière case de c
```

Seule la première dimension doit être contrainte pour un type ayant plusieurs dimensions.

```
int c[][] = new char[10][]; //c une matrice de 10 par ?
```

Les autres dimensions déclarée à l'exécution:

```
c[0] = new char [24]; // première ligne de c = 24 caractères
```

Les dimensions variables:

```
c[1] = new char [12]; // deuxième ligne de c = 12 caractères
```

D'une manière générale, une allocation peut être d'une variable de dimension n peut être vue comme une serie de n-1 boucles imbriquées effectuant chaque allocation.

```
int c[][] = new char[10][10];
// est équivalent à:
int c[][] = new char[10][];
for (int i = 0; i < c.length; i++) //allocation de chaque ligne
de c
c[i] = new char [10];
```

1.2.6.8 conversion de type

- un caractère=> sa valeur entière. ?
- un entier=> sa valeur comme un ?
- convertir les valeurs des types

```
int i;
char c = (char) i; // assigne à c un caractère de valeur i
i = (int) c; //assigne à i un entier de la valeur de c
```

faites explicitement en JAVA

possibilité de perdre de l'information:

```
int i;
i == (int) ((float) i); // n'est pas toujours vrai
```

tableau des conversions sans perte

	byte	short	int	long	float	double	char
byte	oui	oui	oui	oui	oui	oui	oui
short		oui	oui	oui	oui	oui	
int			oui	oui	précision	oui	
long				oui	précision	précision	
float					oui	oui	
double						oui	
char			oui	oui	oui	oui	oui

Les autres conversions peuvent être effectuées sans perte d'information dans certaines conditions, généralement celles qui restreignent le type le plus large aux valeurs du type le plus étroit. byte short int long float double char byte oui oui oui oui oui oui oui short oui oui oui oui oui int oui oui perte de précisio n oui long oui perte de précisio n perte de précisio n float oui oui double oui char oui oui oui oui oui

1.3 Structures de contrôle

Différentes stuctures de contrôle:
le si-alors,
le faire-tantque,
le tantque-faire, dans-le-cas

1.3.1 Le bloc d'instructions

- contenu entre {}.
- définit la visibilité des variables.
- instruction = bloc d'instructions

1.3.2 Exécution conditionelle

Il existe deux formes d'exécution conditionnel. le si_alors_sinon et le dans_le_cas.

1.3.2.1 Si ... alors ... sinon

structure de base de la programmation
L'expression de test booléene (vraie ou fausse)

```
if (e) S1;
if (i==1) s=" i est égal à 1 ";
if (e) S1 else S2;
```

```
if (i==1) s=" i est égal à 1 ";
else s=" i est différent de 1 ";
```

De ces deux formes, nous pouvons dériver:

```
if (e) {B1};
if (i==1)
{
s=" i est égal à 1 ";
i=j;
}
if (e) {B1} else {B2};
```

Algorithm 1.3 - L'alternative complète

```
if (i==1)
{
    s=" i est égal à 1 ";
    i=j;
}
else
{
    s=" i est différent de 1 ";
    i=1515;
}
```

1.3.2.2 cascader les conditions

```
if (e1) S1 else if (e2) S2 ...  else Sn;
```

Algorithm 1.4 - Cascade de conditions

```
if (i==1)
    s = "i est égal à 1";
else if (i==2)
    s = "i est égal à 2";
else if (i==3)
    s = "i est égal à 3";
else
    s = "i est différent de 1,2 et 3";
```

L'utilisation de {} est nécessaire
marquer le début est la fin des instructions;
EXEMPLE :

```
i=0; j=3;
if (i==0)
if (j==2) s=" i est égal à 2 ";
else i=j;
System.out.println(i);
```

Que va imprimer le programme suivant 0 ou 3 ?
Il va imprimer 3! car i=0 et j!=2 alors on exécute i=j
Le else se rapporte en effet au if le plus imbriqué.

1.3.2.3 au cas où

Les cascades de conditions peuvent alourdir grandement le code, switch = un aiguillage. limité à char, byte, short ,int.

- évalue l'expression qui lui est liée
- compare aux case.
- commence à exécuter le code du switch OK
- exécute donc tous les instructions des cas suivants

isoler chaque cas, il faut le terminer avec un break.
Si aucun cas , clause default
case et default sont considérés comme des étiquettes.

```
switch (e)
{
case c1:  S1;
case c2:  S2;
...
case cn:  Sn;
default:  Sd
};
```

Examinons la forme générale de switch avec break;

```
switch (e)
{
case c1:  S1 break;
case c2:  S2 break;
...
case cn:  Sn break;
default:  Sd
};
```

Algorithm 1.5 - L'utilisation de switch

```
class TestSwitch
{
    public static void main (String args[])
    {
        int i = 3;
        switch (i)
        {
            case 1:  System.out.println("I"); break;
            case 2:  System.out.println("II"); break;
            case 3:  System.out.println("III"); break;
            case 4:  System.out.println("IV"); break;
            case 5:  System.out.println("V"); break;
            default:  System.out.println( "pas de traduction");
        }
    }
}
```

1.3.3 Les boucles

1.3.3.1 tant que ... faire ...

L'expression e doit être du type booléen.

```
while (e) S1;
while (e) {B1};
```

Algorithm 1.6 - L'utilisation de while

```
class TestWhile
{
    public static void main (String args[])
    {
        int i = 100, sommeI = 0, j = 0;
        // boucle 1:  expression sans effet de bord
        while (j <= i)
        {
            sommeI += j;
            ++j;
        }
        System.out.println("boucle 1:" + sommeI);
        // boucle 2:  expression avec effet de bord
        sommeI = 0;
        j = 0;
        while (++j <= i)
            sommeI += j;
        System.out.println("boucle 2:" + sommeI);
    }
}
```

1.3.3.2 faire ... tant que ...

L'expression e doit être du type booléen.

```
do S1 while(e);
do {B1} while(e);
```

Algorithm 1.7 - L'utilisation de while

```
class TestDo
{
    public static void main (String args[])
    {
        int i = 100, sommeI = 0, j = 0;
        // boucle 1:  expression sans effet de bord
        do
        {
            sommeI += j;
            ++j;
        } while (j<=i);
        System.out.println("boucle 1:" + sommeI);
        // boucle 2:  expression avec effet de bord
        sommeI = 0;
        j = 0;
        do
            sommeI += j;
        while (++j <= i);
        System.out.println("boucle 2:" + sommeI);
    }
}
```

1.3.3.3 pour ... faire ...

basé sur un itérateur qui est contrôlé dans l'instruction

- expression initialise l'itérateur et le déclare
- expression teste si la condition d'achèvement
- expression modifie l'itérateur.

`for (e1;e2;e3) B1;`
Les expressions e1 et e3 sont optionnelles.
Le bloc B1 doit alors

- modifier l'itérateur
- un break pour quitter la boucle.

for est équivalent au while
`e1; while(e2) {S1; e3};`

Algorithm 1.8 - L'utilisation de while

```
class TestFor
{
    public static void main (String args[])
    {
        int n[] = new int[100];
        // boucle 1:  calculer la somme des entiers pour indice
du vecteur
        for (int i = 1; i < 100; i++)
            n[i] = n[i - 1] + i;
        // boucle 2:  imprimer le résultat par ordre décroissant
        for (int i=99; i>=0;)
        {
            // modification de i dans le bloc d'instructions
            System.out.println("somme(" + i + ")=" + n[i]);
            i--;
        }
    }
}
```