Programmation en Java

IPA 1999

Michel Mauny

PULV & INRIA-Rocquencourt
Michel.Mauny@devinci.fr

Qui ? Quand ?

Cours : MM, TD : MM + Laleh Ghadakpour

vendredi	22/10 :	14h00–17h30	Cours
mercredi	27/10 :	9h30–13h00	TD
mercredi	27/10 :	14h00–17h30	Cours
vendredi	29/10 :	14h00–17h30	TD
mardi	02/11 :	14h00–17h30	Cours
mercredi	03/11 :	9h30–13h00	TD
mercredi	03/11 :	14h00–17h30	Cours
mercredi	10/11 :	14h00–18h45	TD

Comment ?

Cours :
- avec transparents (disponibles — après le cours — sur `http://...`)
- exemples en temps-réel
- exercices à faire, pendant le cours, ou bien chez vous
- passages « au tableau », comme à l'école.
- Questions et interruptions bienvenues.

TD :
- sur PC/Linux, avec
  - `emacs` : éditeur
  - le *Java Developer's Kit* (JDK 1.1.6, Sun Microsystems)
- programmes à faire en 3 heures
- ne jamais rester bloqué(e) sur un problème : demander et redemander ! !

Quoi ?

La programmation
- qu'est-ce qu'un programme ?
- sémantique
- style de programmation
- ce qu'il faut faire et ne pas faire. . .

Le langage Java
1 le langage de base : types et structures de contrôle
2 algorithmes, complexité
3 structuration des programmes I : packages, classes et objets
4 structuration des programmes II : abstraction et héritage

Un programme est une « boîte » qui,
- étant donné quelque chose (son entrée),
- produit un résultat,
- en ayant produit certains effets.

Par exemple, un navigateur Web
- lit une URL
- va chercher le document correspondant, et le stocke sur le disque de la machine (dans le cache)
- et affiche le document à l'écran.

La programmation est la fabrication de telles « boîtes ».

Un programme est un assemblage de nombreux éléments, qui peuvent eux-mêmes être décomposés en éléments plus petits, *etc.*

Un programme a un sens : il est possible de savoir ce qu'il fait dans chacune des situations possibles. On comprend le sens d'un programme en fonction du sens des différents éléments qui le composent.

Ce domaine s'appelle la sémantique des langages de programmation.

La programmation est en fait une discipline où beaucoup de liberté est laissée au programmeur.

La programmation, comme l'écriture ou la peinture peut être réalisée selon des styles différents.

Un programme est comme une œuvre d'art : le style et l'esthétique sont importants,
- pour votre confiance en sa correction et sa fiabilité ;
- pour sa compréhension par un être humain ;
- pour sa maintenance ou ses évolutions futures.

Il faut :
- réfléchir avant d'écrire
- avoir un plan d'ensemble (sur le papier, de préférence) avant de programmer.
- en particulier, bien comprendre ce qui est demandé, connaître (ou chercher) ce qui existe déjà et qui peut être réutilisé, et définir ce que vous devez faire
- de ce plan d'ensemble, on doit en comprendre (imaginer) les différents éléments, et être capable d'en exprimer le rôle
- utiliser des noms assez longs dans vos programmes
- mettre des commentaires : ni trop, ni trop peu

Sinon :

- on se dit à la moitié du programme : *comment je vais m'en sortir ?*
- on se dit vers la fin du programme : *oh là là, j'aurais dû faire autrement ! !*
- on lève la tête et on demande : *siou plaît, qu'est-ce qu'il fait, mon programme ?*
- on se gratte le crâne en disant : *voyons,* `x121`*, c'est le prix de ça,* `x212` *c'est le nombre de trucs, donc quand je fais*

  `f432 = y323(x121,f345(x212,x121)) ;`

  je . . . vais prendre un café.
- . . .

- petits programmes : écrire, compiler, exécuter.
- entiers


```
public class Hello {                          // debut de la classe
  public static void main (String[] args) {   // debut de la methode principale
    System.out.println("Coucou");             // imprime le message
  }                                           // fin methode
}                                             // fin classe
```

Le texte ci-dessus est sauvé dans le fichier `Hello.java`.

Compilation : `javac Hello.java`

Produit le fichier `Hello.class`

Exécution : `java Hello`


```
public class Hello {
  public static void main (String[] args) {
    int i;
    System.out.print("Coucou");
    for (i = 0; i < args.length; i = i+1) {
      System.out.print(" " + args[i]);
    };
    System.out.println();
  }
}
```

Remarques

Une classe est une sorte de modèle (sauf dans les exemples très simples comme ci-dessus). `String` est une classe, par exemple. Le type des tableaux en est une autre.

Une classe (modèle) impose un certain nombre de « champs » (variables ou méthodes) à ses membres.

Un objet est une donnée conforme à un modèle, obtenue à partir de celui-ci, par le mot-clé `new`, en général. `"Coucou"` et `args` sont deux objets. `args` dispose d'une méthode `length` qui rend le nombre d'éléments du tableau.

Remarques (suite)

Une application est une classe, dotée d'une méthode dite statique, c'est-à-dire qui est complètement définie et exécutable sans même que des objets de cette classe ait été créés. Cette méthode est nommée `main`.

Lorsqu'une application est lancée, c'est la méthode `main` qui est lancée.

Remarques (suite et fin)

Conventions de nommage (non-obligatoires, mais conseillées) :
- Classes : commencent par une majuscule (`HelloWorld`)
- Méthodes, variables sont en minuscules (`helloWorld`)
- lorsqu'un nom est constitué de plusieurs mots, on fait commencer les mots suivant le premier par une majuscule.

Dans le reste du cours, on s'intéressera au langage de base, c'est-à-dire qu'on parlera le moins possible de classes et objets.

Commentaires

Les commentaires en Java s'écrivent :
- ou bien `// comme ça (sur une seule ligne)`
- ou alors
```
/* comme ça, et là, ça peut
continuer sur les
lignes suivantes */
```

Les instructions

System.out.println("Hello") ;

sont terminées par un point-virgule.

Il est facile de l'oublier !

Type	Description	Nombre d'octets
byte	entier	1
short	entier	2
int	entier	4
long	entier	8
float	décimal (flottant)	4
double	flottant	8
boolean	booléen	1
char	caractère	2

- **byte** : de -128 à 127
- **short** : de -32 768 à 32 767
- **int** : de -2 147 483 648 à 2 147 483 647
- **long** : de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Declaration :

```
int numberOfPeople = 4;
int count, i;
byte age;
float average;
double xDimension = 23.51;
```


Le type `float` varie de $\pm 3.4 \times 10^{38}$ à $\pm 1.4 \times 10^{-45}$.

Le type `double` varie de $\pm 1.7 \times 10^{308}$ à $\pm 4.9 \times 10^{-324}$.

Exemple :

```
double smallVariation = -3.4E-12;
    if (smallVariation < 1) {
        ...
    }
```

2 valeurs possibles : `true` ou `false`

Exemples :

```
boolean same = (count == i);
boolean older = true;

    if (count <= i) {
        ...
    }
```

Une méthode ou un opérateur calculant des booléens est appelé un prédicat.

Les booléens ne sont pas des entiers.

Opérateurs :

Opérateur	Opération
&&	Et logique, vrai si les 2 conditions sont vraies
||	Ou inclusif, vrai si l'une des 2 cond. au moins est vraie
^	Ou exclusif, vrai si exactement l'une des 2 conditions est vraie
!	Négation

```
x && y          x || y          x ^ y          !(x)
```


Les caractères en Java (composant les String et identificateurs) sont codés sur 16 bits et utilisent le codage Unicode, permettant d'écrire pratiquement tous les jeux de caractères du monde.

Nous ignorerons cette caractéristique, qui est transparente pour le monde occidental.

Caractères spéciaux :

Code	Description
\n	newline
\r	retour-chariot
\t	tabulation
\b	backspace
\\	backslash (contre-oblique)
\"	le caractère "
\'	le caractère '

Exemples :

```
char a = 'a';
char hé = 'é';
char x = '\'';
char y = '"';
char z='\"';

    System.out.print("H" + hé);

=> Hé
```

Chaînes de caractères

Les chaînes ne sont pas vraiment un type de base, mais une classe.

Cependant, son utilisation est naturelle :

```
String hello = "bonjour";
```

Concaténation :

```
System.out.println(hello + " mon garçon");
System.out.println(" vous avez acheté " + count + " bonbons");
```

L'opérateur de concaténation change ses arguments en String lorsque c'est nécessaire (et possible).

Arithmétique

Java dispose des opérateurs arithmétiques classiques :

Opérateur	Opération
+	addition
-	soustraction
*	multiplication
/	division
%	reste de la division

Ces opérations s'appliquent aussi bien sur les entiers que sur les flottants.

La division par zéro provoque une erreur.

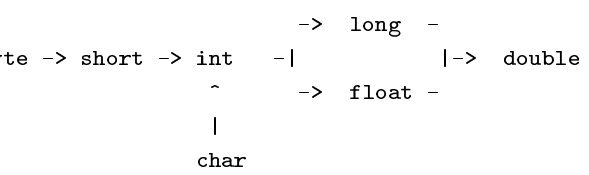
Arithmétique (suite)

Syntaxiquement, il faut faire attention :

`1+2+3` est lu comme `(1+2)+3`
et
`1+2*3` est lu comme `1+(2*3)`

Les opérateurs arithmétiquent associent à gauche, et
les opérateurs multiplicatifs ont priorité sur les opérateurs additifs.

Conversions implicites



Lorsqu'une opération arithmétique attendant des arguments du même type, ceux-ci sont convertis automatiquement en leur plus petit commun majorant.

Il est souvent préférable (plus lisible) d'effectuer les conversions à la main.

```
public class KmToMile {
  public static void main (String[] args) {
    double kilometre = 3.2, mile;   // on déclare mile domme un double
    mile = kilometre * 5 / 8;
    System.out.println(kilometre + " km valent " + mile + " miles.");
 }
}
```

imprime `3.2 km valent 2.0 miles`

Que se passe-t-il si on écrit :

```
    mile = 5 / 8 * kilometre;
```



```
float temperature = (float) 37.2;
float gallons;
double litres = 4.5;
gallons = (float) litres * (float) 0.22;
```

Note : `litres` reste un double. La conversion n'est faite que pour le calcul de `gallons`.

L'opérateur d'affectation est le signe « = ».

Il en existe des formes dérivées :

Opération	Signification
`c += d`	`c = c+d`
`c -= d`	`c = c-d`
`c *= d`	`c = c*d`
`c /= d`	`c = c/d`
`c++`	`c = c+1`
`c--`	`c = c-1`

Imprimer dans la sortie standard : `System.out.print`, `System.out.println`

Imprimer dans la sortie d'erreur (`stderr` sous Unix) : `System.err.print`, `System.err.println`

La lecture est plus compliquée, et on l'abordera dans une leçon suivante. Nous utiliserons le tableau d'arguments, et la méthode `Integer.parseInt` pour lire des arguments entiers.

Exemple :

```
int count;
...
count = Integer.parseInt(args[i])
```

Tableaux

Les tableaux sont des structures de données de taille fixe, contenant des éléments du même type. On accède à un élément du tableau à l'aide d'un entier (indice).

Création :

```
String[] rvb = { "rouge", "vert", "bleu" };
boolean[][] ouExclusif = { {true, false}, {false, false} };
```


Tableaux (suite)

```
byte octetBuffer[];

octetBuffer = new byte[1024];
Button buttons[] = new Button[10];
```

Dans les deux derniers cas, l'allocation du tableau est dynamique.

`octetBuffer` est une référence vers un tableau.

Noter les deux syntaxes différentes pour le type des tableaux.

Tableaux (suite)

La mémoire occuppée par les tableaux est récupérée automatiquement par un *Garbage Collector* (ou GC).

Les tableaux multidimensionnels ne sont que des tableaux de tableaux.

Accès aux éléments d'un tableau :

```
int[] tab = new int[100];

tab[0] = 0;

for (int i=1; i < tab.length; i++)
  tab[i] = i + tab[i-1];
```

La validité des accès aux éléments des tableaux est vérifiée à l'exécution.

Le flot de contrôle : if ... else

```
if (condition) statement;

if (condition) statement1; else statement2;
```

où `statement` est ou bien une instruction simple (affectation, par exemple) ou alors une instruction composée :

```
{ stat1; stat2; ...; statN; }
```

Exemple :

```
public void checkSpeed() {
  if (speed < 200) System.out.println("trop lent"); else
  if (speed < 300) System.out.println("c'est bon");
  else System.out.println("vraiment top!");
}
```

Permet de tester la valeur d'une expression contre des constantes :

```
switch (expression) {
  case (valeur1): ...; break;
  ...
  case (valeurN): ...; break;
  default: ...;
}
```


Exemple :

```
public void checkMemory() {
  String m;
  switch (memory) {
    case(8):
    case(16): m = "trop peu"; break;
    case(32): m = "tout juste"; break;
    case(64): m = "suffisant"; break;
    default: m = "ouah!";
  }
}
```


Répéter :

```
for (count=0; count < 10; count++) {
      ...
}
```

Tant que :

```
while (condition) {
  ...
}
```

Tant que, mais corps exécuté au moins une fois :

```
do {
  ...
} while (condition)
```


`break` (déjà vu dans le `switch`), permet de sortir de la construction courante (switch, for, while, do...while).

`continue` stoppe l'itération courante, et passe à l'étape suivante (pour les boucles).

Donner une valeur de retour à une méthode

Le mot-clé `return` permet de sortir de la méthode courante avec un résultat donné.

```
public static int minimum2(int m, int n) {
   if (m < n)
      return m;
   else
      return n;
}
```

Ici, la méthode `minimum2` a 2 paramètres entiers `m` et `n`, et retourne un entier : le plus petit des 2.

Exercices

1. Écrire un programme calculant et imprimant le minimum d'un tableau d'entiers. Les entiers sont donnés sur la ligne de commande :

   ```
   % java Min 3 2 4 1 5 6 -2
   => -2
   ```

2. Modifier ce programme pour qu'il calcule et imprime la moyenne des valeurs du tableau.
3. Le modifier encore pour qu'il imprime le tableau trié en ordre croissant (tri par insertion).