Chapitre 1
Introduction à Java

Plan

- Références bibliographiques
- Les paradigmes de programmation
- Présentation du langage
- Mise en route
- Programmation procédurale en Java
- Identificateurs
- Données primitives
- Opérateurs
- Structures de contrôle
- Méthodes
- Commentaires et documentation

Références bibliographiques

Référence 1

Bruce Eckel
Thinking in Java
2ème édition
www.BruceEckel.com
850 pages
édition électronique

Référence 2

Cay S. Horstmann et Gary Cornell
Au coeur de Java 2
Volume 1 - notions fondamentales
www.campuspress.fr
774 pages
37,96 €

Référence 3

Cay S. Horstmann et Gary Cornell
Au coeur de Java 2
Volume 2 - notions avancées
www.campuspress.fr
990 pages
37,96 €

Référence 4

Deitel et Deitel
Comment programmer en Java
4ème édition
www.goulet.ca
1350 pages
89,95 $
ISBN: 2-89377-254-4

Référence 5

Ivor Horton
Maîtrisez Java 2
www.wrox.fr
1300 pages
50 €

Référence 6

Clavel, Mirouze, Pichon et Soukal

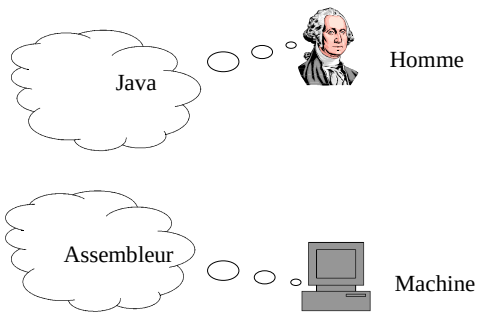
Java, la synthèse

InterEditions

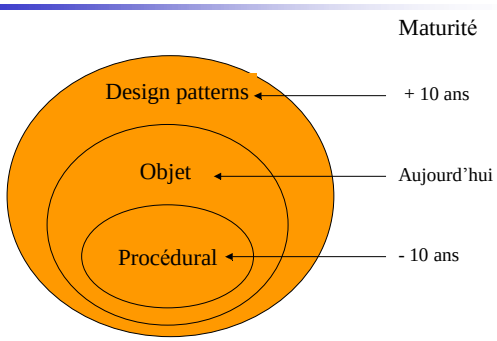
220 pages

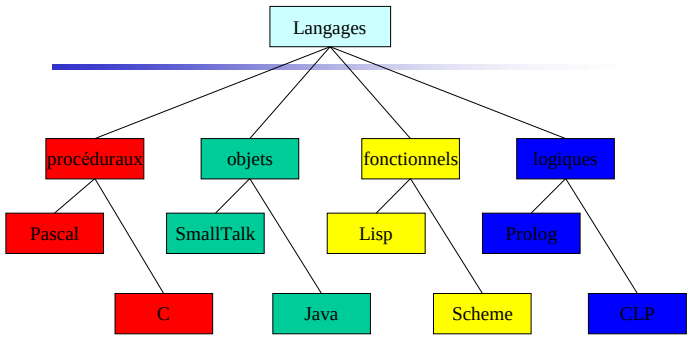
Les paradigmes de programmation

Où en est-on ?



Les principaux paradigmes





Quel type de programmation ?

- Programmation procédurale
  - représentation d'une solution algorithmique en termes de procédures.
- Programmation objet
  - représentation d'une solution algorithmique en termes d'objets s'échangeant des messages.

Quel type de programmation ?

- Programmation fonctionnelle
  - représentation d'une solution algorithmique en termes de fonctions.
- Programmation logique
  - représentation d'une solution algorithmique en termes de prédicats.

Programmation procédurale

- Langages : Basic, Fortran, Pascal, C...
- Ils permettent de structurer les instructions (sous forme de procédures) et les données (sous forme de types).
- Ils peuvent être utilisés sans trop de problèmes pour de petites applications...
- Ils sont pénalisants pour de grosses applications...

Programmation objet

- Langages : SmallTalk, C++, Eiffel, Java...
- Ils permettent de structurer les objets (sous forme de classes).
- Ils sont adaptés à la conception de grosses applications car ils offrent une modélisation plus naturelle et plus aisée...

Programmation fonctionnelle

- Langages Lisp, Scheme, ...
- Langages utilisant essentiellement la structure de Listes.
- Langages assez déclaratifs utilisés surtout en intelligence artificielle (Lisp dans les années 70-80).

Programmation logique

- Langages Prolog, CLP, ...
- Langages basées sur la logique des prédicats (Prolog) et l'intégration de contraintes (CLP).
- Langages purement déclaratifs utilisés surtout en intelligence artificielle (Prolog dans les années 89-90 et CLP/CSP aujourd'hui).

Présentation du langage

Java

- Langage de programmation
  - orienté objet
  - simple
  - portable
  - efficace pour le développement
  - complet
  - gratuit
- Développé par Sun Microsystems (www.sun.com)
- Début 1999 : Java 2 SDK Platform

Java est orienté objet

- Tout est classe excepté quelques types primitifs.
  - toute classe dérive de java.lang.Object
- Modèle plus pur que celui de C++
  - pas de variables et fonctions en dehors d'une classe

Java est simple

- Java suit quelque peu la syntaxe de C/C++
  - mêmes instructions
  - mêmes structures de contrôle
- Mais
  - pas de fichiers en-tête
  - pas d'arithmétique de pointeurs
  - pas de structures et unions
  - pas de pointeurs de fonction

Java est portable

- Le compilateur java produit un code intermédiaire universel (le bytecode).
- Il suffit de posséder un interpréteur pour le bytecode sur chaque type de OS/UC.
- Les types de données élémentaires sont clairement spécifiées.

Java est efficace pour le développement

- Développer une application en Java est au moins 2 fois plus rapide qu'en C++
- Exécuter une application en Java est 4 fois moins rapide qu'en C++ mais
  - le temps d'exécution est peu important pour nombre d'applications.
  - il est toujours possible de traduire en C/C++ l'application (qui est alors un prototype).

Java est complet

- Compilateur : javac
- Interpréteur : java
- Générateur de documentation : javadoc
- Testeur pour applets : appletviewer
- Documentation :
  - spécifications, APIs
  - tutorial

Java est gratuit

- Java 2 SDK, Standard Edition Version 1.3.0
  - outils de base permettant de créer des applications (et applets) Java
- Integrated Development Environnement
  - environnement permettant de développer visuellement des applications Java

Mise en route

Structure d'un programme Java

- Nécessite la définition d'au moins une classe dans laquelle apparaît une fonction main.
- Cette classe doit apparaître dans un fichier portant le même nom comme préfixe et java comme suffixe.
- Convention : le nom de la classe (et le nom du fichier) commence par une majuscule.

Un premier programme

- Afficher le message "hello world" à l'écran.
  - une classe de nom Hello dans un fichier de nom Hello.java
  - une fonction main dans le corps de cette classe en n'oubliant pas l'argument (String[] args).
  - une instruction System.out.println(...)
- Hello.java

Hello.java


```
class Hello
{
   static void main(String[] args)
   {
      System.out.println("Hello world");
   }
}
```


Compilation

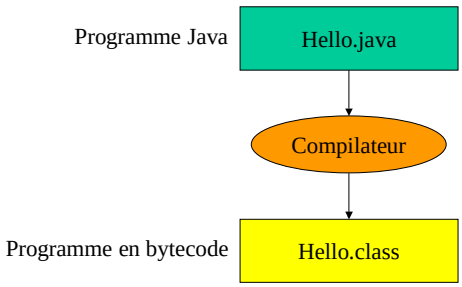
- Un programme Java n'est pas directement traduit dans le langage natif de l'ordinateur.
- Il est d'abord traduit par une opération de compilation dans un langage appelé « bytecode ».
- Ce langage est indépendant de la machine sur laquelle est exécuté le programme.

Commande de compilation

javac NomClasse.java

- Résultat de la commande
  - une erreur de compilation si la classe est mal écrite
  - un fichier NomClasse.class sinon
- Le fichier généré NomClasse.class
  - contient du bytecode
  - se situe dans le même répertoire que le fichier source (NomClasse.java)

Compilateur



Exécution

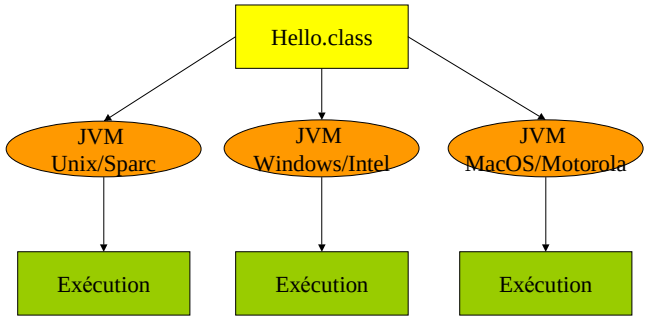
- Le bytecode doit être exécuté par une JVM (Java Virtual Machine).
- Une JVM est un interpréteur qui traduit chaque instruction du bytecode dans le langage de la machine où s'exécute le programme.
- Tout système doit fournit sa JVM.

Commande d'exécution

java NomClasse

- Résultat de la commande
  - le bytecode de la fonction main de la classe NomClasse est interprété par la JVM
- Problèmes pouvant survenir
  - le fichier NomClasse.class ne peut être trouvé
    Java.lang.NoClassDefFoundError: NomClasse
  - la fonction main ne peut être trouvée
    java.lang.NoSuchMethodError: main

JVM



Le bytecode

- Avantages
  - portabilité : tout fichier .class peut être exporté vers une autre machine.
  - sécurité : la JVM contrôle les instructions du bytecode.
- Inconvénients
  - lenteur d'exécution
    - la JVM effectue de nombreuses vérifications
    - la JVM doit interpréter les instructions

Programmation procédurale en Java

Programmation procédurale

- Il est possible en Java d'ignorer le concept objet.
- Il suffit de tout déclarer sous une forme *static*
- Mais peu d'intérêt d'utiliser un langage objet pour écrire du procédural (hormis pour quelque petites applications).

Principe

- Il suffit de décomposer le problème en plusieurs procédures (et fonctions).
- Les procédures et fonctions
  - sont placées dans une classe
  - sont codées en précisant « static » devant leur en-tête
- Les procédures et fonctions peuvent être appelées à partir de n'importe quelle autre.

Exemple

- On désire afficher la racine carrée d'une valeur donnée par l'utilisateur.
- On peut décomposer (de façon simpliste) ce problème en trois sous-problèmes :
  - lire la valeur
  - calculer la racine carrée
  - afficher la racine carrée

Racine.java (1/2)

```
class Racine
{
   static int lireParametre()
   {
        System.out.print("Entrez une valeur entiere : ");
        int v = Mediator.readInt();
        return v;
   }

   static double calculerRacine(int v)
   {
        double d = Math.sqrt(v);
        return d;
   }
```


Racine.java (2/2)

```
   static void afficherRacine(double r)
   {
        System.out.println("La racine est égale à " + r);
   }

   static void main(String[] args)
   {
        int i=lireParametre();
        double r = calculerRacine(i);
        afficherRacine(r);
   }
}
```


Identificateurs

Définition d'un identificateur

- Un identificateur est un nom pouvant être donné à tout élément d'un programme Java:
  - variables (et constantes)
  - classes
  - méthodes
  - Packages
- Un identificateur peut être aussi long que souhaité.

Règle de nommage

- La première lettre d'un identificateur est :
  - soit une lettre [a..z,A..Z]
  - soit un tiret-bas _
  - soit un dollar $
- Pour les autres lettres, il est possible d'utiliser les caractères Unicode dont le code est supérieur à 0x00C0 tel que ç, à, ü, …

v Mais en pratique, on évitera ces caractères.

Conventions de nommage

- Un nom de classe commence par une majuscule.
- Un nom de variable (champ) ou de fonction (méthode) commence par une minuscule. Si le nom est composé, chaque mot (autre que le premier) commence par une majuscule.
- Pour en savoir plus, Conventions

Intérêt des conventions

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

  http://java.sun.com/docs/codeconv/

Mots réservés

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

Données « primitives »

Les données primitives

- On appellera donnée primitive toute donnée définie à partir d'un type suivant :

  boolean char

  byte short int long

  float double
- On peut définir des variables primitives à partir de ces types primitifs.

Les types primitifs

- Tous les types primitifs numériques sont signés. Il n'existe pas comme en C de spécificateur « unsigned ».
- Tous les types primitifs sont invariants en taille quelque soit l'architecture de la machine.
  - (gros) Avantage : portabilité
  - (léger) Inconvénient : perte d'efficacité

Les types primitifs en C/C++ : L

- Sur un processeur 16 bits (tel que 8086)
  - le type int est codé sur 2 octets
- Sur un processeur 32 bits (tel que SPARC)
  - le type int est codé sur 4 octets
- Sur un Pentium Intel
  - le type int est codé sur 2 octets sous Dos et Windows 3.1 et sur 4 octets sous Windows 95 et NT.

boolean

- Type défini sur 1 bit (1 octet ?)
- Valeurs possibles :

  {false, true}
- Déclarations possibles de variables:

  boolean b;

  boolean fini=false;
- Valeur par défaut d'une variable champ de classe : false

Les booléens en C/C++ : L

- En C, il n'existe pas de type booléen. On utilise la valeur entière 0 pour faux et toute valeur différente de 0 pour vrai.
- En C++, le type bool a été ajouté à la norme mais il est toujours possible d'utiliser des pointeurs et des entiers comme expressions booléennes.

char

- Type défini sur 16 bits.
- Valeurs possibles :
  {' \u0000' , …, ' \uFFFF' }
- Déclarations possibles de variables:
  char c;
  char espace=' ';
- Valeur par défaut d'une variable champ de classe : ' \u0000'

Unicode versus ASCII

- Le jeu Unicode (codé sur 2 octets) permet de gérer tous les caractères de toutes les langues écrites
- Le jeu ASCII/ANSI (codé sur 1 octet) est un sous-ensemble du jeu Unicode : les 255 premiers caractères.
- http://www.unicode.org/

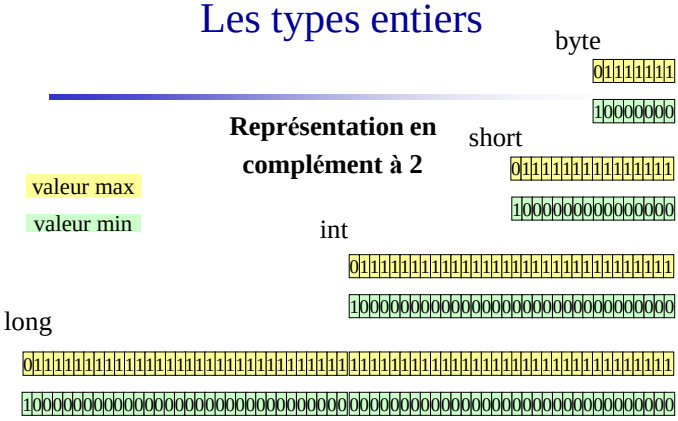
Les littéraux du type char

- Caractères imprimables
  - entre simples quotes (apostrophes) : 'a'
  - mais pas entre doubles quotes (guillemets) : "a"
- Caractères spéciaux
  - entre simples quotes et en utilisant une séquence d'échappement (\ et un caractère)
- Tous les caractères
  - entre simples quotes et en utilisant une notation hexadécimale de la forme \u????

Exemples

backspace	'\b'	'\u0008'
tabulation	'\t'	'\u0009'
new line	'\n'	'\u000a'
carriage return	'\r'	'\u000d'
form feed	'\f'	'\u000c'
Simple quote	'\''	'\u0027'
Double quote	'\"'	'\u0022'
Antislash	'\\'	'\u005c'

Les types entiers



byte

- Type défini sur 8 bits.
- Valeurs possibles :
  $\{-2^7, \ldots, 2^7-1\}$ soit de -128 à 127
- Déclarations possibles de variables:
  byte b;
  byte octetLu=-1;
- Valeur par défaut d'une variable champ de classe : (byte)0

short

- Type défini sur 16 bits.
- Valeurs possibles :
  $\{-2^{15}, \ldots, 2^{15}-1\}$ soit de –32768 à 32767
- Déclarations possibles de variables:
  short s;
  short longueur=100;
- Valeur par défaut d'une variable champ de classe : (short)0

int

- Type défini sur 32 bits.
- Valeurs possibles :
  $\{-2^{31}, \ldots, 2^{31}-1\}$ soit de –2 147 483 648 à …
- Déclarations possibles de variables:
  int i;
  int nbLignes=10;
- Valeur par défaut d'une variable champ de classe : 0

long

- Type défini sur 64 bits.
- Valeurs possibles :
  {-2^{63}, …, 2^{63}-1} soit –9 223 372 036 854 775 808L à
- Déclarations possibles de variables:
  long l;
  long capacite=34500;
- Valeur par défaut d'une variable champ de classe : 0L

Les littéraux entiers

- Il est possible d'utiliser une notation hexadécimale pour coder une valeur entière.
  byte maxByte = 0x7F;
  short maxShort = 0x7FFF;
  int i = 0x2a11bc10;
  long n = 2011L;
- Par défaut, un littéral entier est de type int.
- Un suffixe L permet d'identifier un littéral long.
- Un préfixe 0 permet d'utiliser une notation octale

Les types réels

float **Norme IEEE 754**

+/- 3.4E38 avec environ 7 chiffres de précision

double

+/- 1.7E308 avec environ 15 chiffres de précision

float

- Type défini sur 32 bits.
- Valeurs possibles :
  Définies par la norme IEEE 754
- Déclarations possibles de variables:
  float f;
  float volume=34.5;
- Valeur par défaut d'une variable champ de classe : 0.0F

double

- Type défini sur 64 bits.
- Valeurs possibles :
  Définies par la norme IEEE 754
- Déclarations possibles de variables:
  double d;
  double surface=100;
- Valeur par défaut d'une variable champ de classe : 0.0D

Les littéraux réels

- Par défaut, un littéral réel est de type double.
- Un suffixe F permet d'identifier un littéral float.
- Un suffixe D permet d'identifier un littéral double.

Approximation des valeurs réelles

- Il est nécessaire d'être très prudent avec les valeurs réelles.
- Si celles-ci sont utilisées au niveau de conditions de structures de contrôle, un problème peut survenir.
- Approximation.java
- Utiliser dans ce cas des valeurs entières.

Opérateurs

Opérateurs arithmétiques

- Addition : + Soustraction : -
- Multiplication : * Division : /
- Incrémentation : ++ Décrémentation : --
- Reste de division entière : %
- Une division est entière si les 2 opérandes représentent des valeurs entières, réelle sinon.

Incrémentation et décrémentation

- Ces deux opérateurs (++ et --) :
  - Ne peuvent être appliquées qu'à une variable
  - Peuvent être placées en position préfixe ou postfixe
- La différence entre les deux positions n'apparaît que dans certaines expressions. Par exemple :
  - int m=7, n=7;
  - int a = 2* ++m; // a vaut 16 et m vaut 8
  - int b = 2* n++; // b vaut 14 et n vaut 8

v A utiliser avec précaution.

Opérateurs relationnels

- Égal à : == Différent de : !=
- Inférieur à : < Inférieur ou égal à : <=
- Supérieur à : > Supérieur ou égal à : >=
- On ne peut utiliser = à la place de ==

  if (x = 0) est admis en C/C++

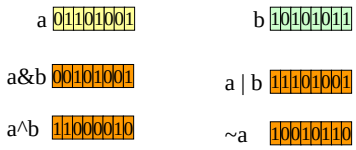
  if (x = 0) n'est pas admis en Java.

Opérateurs logiques

- Et : && Ou : || Non : !
- Les opérateurs && et || sont évalués de manière optimisée (en court-circuit) :
  - Dans A && B, B n'est évaluée que si A a été évaluée à vrai.
  - Dans A || B, B n'est évaluée que si A a été évaluée à faux.

Opérateurs bit à bit

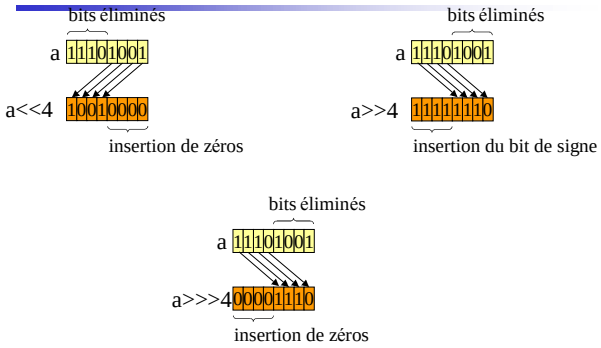
- Et : & Complément : ~
- Ou : | Ou exclusif : ^
- Ces opérateurs s'utilisent avec des entiers.



Opérateurs de décalage

- Décalage à gauche :
  << (insertion de zéros)
- Décalage à droite :
  >> (insertion du bit de signe)
  >>> (insertion de zéros)
- Ces opérateurs s'utilisent avec des entiers.
- Décaler d'une position à gauche (resp. à droite) une valeur entière revient à la multiplier (resp. à la diviser) par 2.

Opérateurs de décalage



Opérateurs d'affectation

- Opérateur simple d'affectation : =
- Opérateurs d'affectation combinés avec un opérateur arithmétique :
  += -= *= /= %=
- Opérateurs d'affectation combinés avec un opérateur bit à bit :
  &=|= ~= ^= <<= >>= >>>=
- Equivalence
  lhs op= rhs est un raccourci de lhs = lhs op (rhs);

Opération de transtypage

- Transtypage (cast) : (typeCast)expression
- Cette opération permet de convertir le type de l'expression en typeCast.
- L'opération peut diminuer la généralité (downcasting) du type avec un risque de perte d'informations. Elle peut également augmenter la généralité (upcasting) du type (sans aucun risque).
- Du moins général au plus général (pour les types primitifs), on a :

  byte → short → int → long → float → double

Cast implicite

- Lors d'une affectation, une opération de upcasting implicite peut avoir lieu (lorsque la partie droite de l'affectation est d'un type moins général).
- Par exemple, lors de l'évaluation de :

  double x = 15;

  la valeur entière 15 est transformée automatiquement en la valeur réelle (double) 15.0 avant d'être affectée à la variable x.

Cast explicite

- Lors de certaines opérations, il est parfois nécessaire d'utiliser explicitement une opération de transtypage.
- Par exemple, l'exécution de :

  int x=3, y=2;

  double z = x/y;

  affecte la valeur 1.0 à z
- Pour que la valeur 1.5 soit affectée à z, il faut :

  double z = (double)x/y;

  afin de convertir x en double et que la division soit ainsi une division réelle.

Opérateur ternaire

- L'opérateur ternaire ?: permet d'évaluer une alternative.
- Par exemple :

  int max = (x > y ? x : y);

  Est équivalent à :

  int max;

  if (x > y) max=x

  else max=y;

v A utiliser avec parcimonie.

Autres opérateurs

- () : appel de fonction
- [] : accès à un tableau
- . : déréférençage
- new : création d'un objet
- instanceof

Priorités des opérateurs

() [] .	gauche
+ (unaire) - (unaire) ++ -- ! ~ (type) new	droite
* / %	gauche
+ -	gauche
<< >> >>>	gauche
< <= > >= instanceof	gauche
== !=	gauche
&	gauche
^	gauche
|	gauche
&&	gauche
||	gauche
?:	gauche
= += -= *= /= %= <<= >>= >>>= &= |= ^=	droite

Structures de contrôle

Instruction

- Une instruction peut-être :
  - Une affectation

    Format : variable = expression;
  - Une structure de contrôle
    - Conditionnelle (if, switch)
    - Itérative (while, for, do… while)
  - L'appel à la méthode d'un objet

    Format : objet.méthode(…);
  - L'appel à la méthode d'une classe

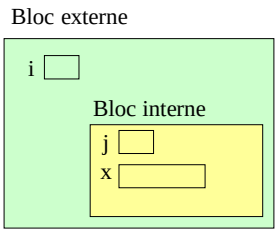
    Format : classe.méthode(…);

Bloc d'instructions

- Un bloc d'instructions est une séquence d'instructions délimitée par une paire d'accolades.
- Il est possible de déclarer des variables dans un bloc.
- La portée (visibilité) d'une variable commence à sa déclaration jusqu'à la fin du bloc.

Exemple

```
{
   int i;
   …
   {
        int j=0;
        …
        double x = 3.5;
        …
   }
}
```



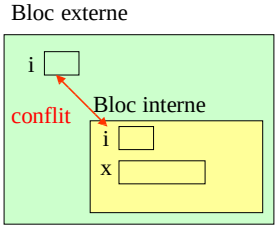
Variables homonymes

- En Java, il n'est pas possible déclarer des variables homonymes dans deux blocs imbriqués.
- En C/C++, il est possible de le faire :
  - La variable interne cache la variable externe
  - Cela constitue une source d'erreur **L**

Exemple

```
{
   int i;
   …
   {
        int i=0;
        …
        double x = 3.5;
        …
   }
}
```

conflit



Statement

- Appelons statement :
  - toute instruction
  - tout bloc d'instructions

if

- La structure conditionnelle if se présente sous deux formes.
- Forme 1

  If (expression-booléenne) statement
- Forme 2

  If (expression-booléenne) statement

  else statement

L

Exemple

```
int comparer(int valeur1, int valeur2)
{
   int resultat;
   if (valeur1 < valeur2) resultat=-1;
   else if (valeur1 > valeur2) resultat=+1;
   else resultat=0;
   return resultat;
}
```


Exemple

```
int comparer(int valeur1, int valeur2)
{
   int resultat;
   if (valeur1 < valeur2) resultat=-1;
   else if (valeur1 > valeur2) resultat=+1;

   return resultat;
}
```

int resultat =0;

Erreur à la compilation
Java exige que resultat soit initialisé

J

Exemple

```
int comparer(int valeur1, int valeur2)
{
   if (valeur1 < valeur2) return -1;
   if (valeur1 > valeur2) return +1;
   return 0;
}
```


switch

- La structure conditionnelle switch se présente sous la forme suivante :

```
switch (expression)
{
   case constante : statement
   ...
   case constante : statement
   default : statement
}
```


switch

- Une instruction break est nécessaire au niveau de chaque cas (sauf le dernier) si on ne désire pas exécuter le reste des instructions.
- Le choix par défaut est facultatif.
- Il est possible de grouper plusieurs cas.
- Le type de l'expression doit être entier ou caractère.

L

Exemple

```
void afficheMedaillePour(int position)
{
   switch (position)
   {
        case 1 : System.out.println("or »); break;
        case 2 : System.out.println("argent"); break;
        case 3 : System.out.println("bronze"); break;
        default : System.out.println("non classé");
   }
}
```


Intérêt du switch ?

- Structure plutôt lourde qui peut être parfois remplacée avantageusement par des if en cascade.
- Par ailleurs, le concept de polymorphisme réduit l'emploi de structures conditionnelles.

Exemple

```
void afficheMedaillePour(int position)
{
   if (position == 1) System.out.println("or »);
   else if (position == 2) System.out.println("argent");
   else if (position == 3) System.out.println("bronze");
   else System.out.println("non classé");
}
```


Exemple

```
String retourneMedaillePour(int position)
{
   switch (position)
   {
        case 1 : return "or";
        case 2 : return "argent";
        case 3 : return "bronze";
        default : return "non classé";
   }
}
```


J

Exemple

```
String retourneMedaillePour(int position)
{
   if (position == 1) return "or";
   if (position == 2) return "argent";
   if (position == 3) return "bronze";
   return "non classé";
}
```

while

- La structure itérative while se présente sous la forme suivante :

```
while (expression-booléenne) statement
```


Exemple

```
long factorielDe(int n)
{
   long fact=1;
   int cpt=2;
   while (cpt <= n)
   {
      fact=fact*cpt;
      cpt++;
   }
   return fact;
}
```


for

- La structure itérative for se présente sous la forme suivante :

```
for (initialisation; expression-booléenne; step)
   statement
```


Exemple

```
long factorielDe(int n)
{
   long fact=1;
   for (int i=2; i <= n; i++) fact=fact*cpt;
   return fact;
}
```


do … while

- La structure itérative do … while se présente sous la forme suivante :

```
do statement
while (expression-booléenne);
```


Exemple

```
int lireNote()
{
   int note;
   do
   {
        System.out.print("Entrez une note : ");
        note=Mediator.readInt();
   }
   while (note < 0 || note > 20);
   return note;
}
```


break

- L'instruction break permet de sortir d'une boucle sans
  - exécuter les instructions restant dans le tour
  - tester la condition de la boucle
- A utiliser dans de rares occasions

L

Exemple

```
boolean chercherZeroDans(int[] tableau)
{
   int cpt=0;
   boolean trouve=false;
   while (!trouve && cpt < tableau.length)
   {
        if (tableau[cpt] == 0) trouve=true;
        cpt++;
   }
   return trouve;
}
```

L

Exemple

```
boolean chercherZeroDans(int[] tableau)
{
   boolean trouve=false;
   for (int cpt=0; !trouve && cpt<tableau.length; cpt++)
        if (tableau[cpt] == 0) trouve=true;
   return trouve;
}
```


Exemple

```
boolean chercherZeroDans(int[] tableau)
{
   int cpt;
   for (cpt=0; cpt<tableau.length; cpt++)
        if (tableau[cpt] == 0) break;
   return (cpt < tableau.length);
}
```


J

Exemple

```
boolean chercherZeroDans(int[] tableau)
{
   for (cpt=0; cpt<tableau.length; cpt++)
        if (tableau[cpt] == 0) return true;
   return false;
}
```


Continue

- L'instruction continue permet de terminer le tour d'une boucle sans exécuter les instructions restant dans le tour.
- A utiliser avec précaution et parcimonie.

Exemple

```
int cpt=0;
for (int i=idRow-1; i<=idRow+1; i++) {
   if (i<0 || i>=cells.length) continue;
   for (int j=idColumn-1; j<=idColumn+1; j++) {
        if (j<0 || j>=cells[i].length) continue;
        cpt++;
   }
}
```


Méthodes

Méthodes

- Les fonctions en Java s'appellent des méthodes.
- Il existe deux types de méthodes :
  - Les méthodes d'instances
  - Les méthodes de classes

Format des méthodes

- Le format est le suivant :
  type-retour nom-fonction(paramètres)
- Lorsque la fonction ne retourne rien, il faut utiliser le type de retour void.
- Si il n'y a aucun paramètres, on ne place rien entre parenthèses : on ne peut placer void.
- Les paramètres sont séparés par des virgules.

Paramètres

- Les paramètres sont toujours passés par valeur.
- Règle importante : il ne faut jamais chercher à modifier les paramètres.
- Cette règle ne peut être appliquée que dans le cadre d'une approche objet :
  - l'objet qui exécute la méthode peut être modifié

Méthodes de classe

- Fonctions déclarées comme "static".
- Il est possible de développer une approche procédurale classique en définissant static toutes les méthodes.
- CalculateurV1.java

Méthodes d'instance

- Fonctions non déclarées "static".
- Il est nécessaire de créer un objet pour pouvoir appeler de telles méthodes.
- CalculateurV2.java
- Comme il n'y a aucun champ dans cette classe, cette approche n'est pas dans ce cas précis indispensable.

Commentaires et documentation

Commentaires

Deux façons de procéder :

- à la C : sur 1 ou plusieurs lignes

```
/* ceci est un commentaire
sur deux lignes */
```

- à la C++ : sur 1 seule ligne

```
// ceci est un commentaire sur 1 seule ligne
```


Documentation

Gestion automatique par javadoc

- Les commentaires peuvent porter sur :
  - une classe
  - un attribut
  - une méthode
- Les commentaires peuvent intégrer
  - des commandes HTML
  - des tags

Documentation

```
/** un commentaire de classe */
public class Essai
{
    /** un commentaire d'attribut */
    public int i;
    /** un commentaire de méthode
    *    sur plusieurs lignes
    */
    public void f() { ... }
}
```


Tags

@version version-information
@author author-information
@param parameter-name description
@return description
@exception class-name description
@see class-name
@see class-name#method-name

Exemple

- Commentons la classe Calculateur.
- CalculateurV3.java
- Pour générer la documentation :
  javadoc CalculateurV3.java
  options : -flag et –author
- Résultat : une documentation au format html
- index.html