Types et instructions de base Java

Certains de ces transparents sont une reprise des transparents du cours *"Comment JAVA ? Java bien !"* de P. Itey

Identificateurs

- Nommer les classes, les variables, les méthodes, ...
- Un identificateur Java
  - *est de longueur quelconque*
  - *commence par une lettre Unicode (caractères ASCII recommandés)*
  - *peut ensuite contenir des lettres ou des chiffres ou le caractère souligné « _ »*
  - *ne doit pas être un mot réservé du langage (mot clé) (`if, for, true, …`)*

```
[a..z, A..Z, $, _]{a..z, A..Z, $, _, 0..9, Unicode}
```

Conventions pour les identificateurs

- Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes) :
  - **`Visage, Object`**
- Les mots contenus dans un identificateur commencent par une majuscule :
  - **`UneClasse, uneMethode, uneVariable`**
  - On préférera **`ageDuCapitaine`** à **`ageducapitaine`** ou **`age_du_capitaine`**
- Les constantes sont en majuscules et les mots sont séparés par le caractère souligné « _ » :
  - **`UNE_CONSTANTE`**

Mots-clés Java

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

Commentaires

- Sur une ligne

```
// Comme en "C++", après un slash-slash
int i; // commentaire jusqu'à la fin de la ligne
```

- Sur plusieurs lignes

```
/* Comme en "C", entre un slash-étoile et
   un étoile-slash, sur plusieurs lignes */
```

- Commentaires documentants pour l'outil **javadoc**

```
/**
 * pour l'utilisation de Javadoc
 * à réserver pour la documentation automatique
 * avec javadoc
 */
```


Commentaires

- Commenter le plus possible et judicieusement
- Chaque déclaration (variable, mais aussi méthode, classe)
- Commenter clairement (utiliser au mieux les 3 possibilités)

Types de données en Java

- 2 grands groupes de types de données :
  - *types primitifs*
  - *objets (instances de classe)*
- Java manipule différemment les valeurs des types primitifs et les objets : les variables contiennent
  - *des valeurs de types primitifs*
  - *ou des références aux objets*

Types primitifs

- Valeur logique
  - ***boolean*** *(true/false)*
- Nombres entiers
  - ***byte*** *(1 octet),* ***short*** *(2octets),* ***int*** *(4 octets),* ***long*** *(8 octets)*
- Nombres non entiers (à virgule flottante)
  - ***float*** *(4 octets),* ***double*** *(8 octets).*
- Caractère (un seul)
  - ***char*** *(2 octets) ; codé par le codage Unicode (et pas ASCII)*

- types indépendants de l'architecture
  - *En C/C++, représentation dépendante de l'architecture (compilateur, système d'exploitation, processeur)*
    ex: int = 32 bits sous x86, mais 64 bits sous DEC alpha
    Portage difficile, types numériques signés/non signés

Types primitifs et valeurs

Type	Taille	Valeurs
`boolean`	1	`true`, `false`
`byte`	8	-2^7 à $+2^7-1$
`char`	16	0 à 65535
`short`	16	-2^{15} à $+2^{15}-1$
`int`	32	-2^{31} à $+2^{31}-1$
`long`	64	-2^{63} à $+2^{63}-1$
`float`	32	1.40239846e-45 à 3.40282347e38
`double`	64	4.94065645841246544e-324 à 1.79769313486231570e308

Constantes nombres

- Une constante «entière» est de type **long** si elle est suffixée par «**`L`**» et de type **int** sinon
- Une constante «flottante» est de type **float** si elle est suffixée par «**`F`**» et de type **double** sinon

- **Exemples**
  - *`35`*
  - *`2589L // constante de type long`*
  - *`4.567e2 // 456,7 de type double`*
  - *`.123587E-25F // de type float`*

Constantes de type caractère

- Un caractère Unicode entouré par 2 simples quotes "'"
- Exemples :
  - `'A' 'a' 'ç' '1' '2'`
  - `\` *caractère d'échappement pour introduire les caractères spéciaux*
    - `'\t'` tabulation
    - `'\n'` nouvelle ligne
    - `'\r'` retour chariot, retour arrière
    - `'\f'` saut de page
    - …
    - `'\\' '\`' '\"'`
  - *`'\u03a9'` (`\u` suivi du code hexadécimal à 4 chiffres d'un caractère Unicode)*
    - `'α'`

Autres constantes

- Type booléen
  - ***`false`***
  - ***`true`***
- Référence inexistante (indique qu'une variable de type non primitif ne référence rien)
  - ***`null`***

Forcer un type en Java

- Java langage fortement typé
  - *le type de donnée est associé au nom de la variable, plutôt qu'à sa valeur. (Avant de pouvoir être utilisée une variable doit être déclarée en associant un type à son identificateur).*
  - *la compilation ou l'exécution peuvent détecter des erreurs de typage*
- Dans certains cas, nécessaire de forcer le compilateur à considérer une expression comme étant d'un type qui n'est pas son type réel ou déclaré
  - *On utilise le* cast ou transtypage*:* (type-forcé) expression
- **Exemple**
  - `int i = 64;`
  - `char c = (char)i;`

Casts entre types primitifs

- Un *cast* entre types primitifs peut occasionner une perte de données
  - *Par exemple, la conversion d'un **int** vers un **short** peut donner un nombre complètement différent du nombre de départ.*

```
int i = 32768;
short s = (short) i;
System.out.println(s);   → -32767;
```

- Un *cast* peut provoquer une simple perte de précision
  - *Par exemple, la conversion d'un **long** vers un **float** peut faire perdre des chiffres significatifs mais pas l'ordre de grandeur*

```
long l1 = 928999999L;
float f = (float) l1;
System.out.println(f); → 9.29E8
long l2 = (long) f;
System.out.println(l2); → 929000000
```

Casts entre types primitifs

- Les affectations entre types primitifs peuvent utiliser un *cast* implicite si elles ne peuvent provoquer qu'une perte de précision (ou, encore mieux, aucune perte)

```
int i = 130;
double x = 20 * i;
```

- Sinon, elles doivent comporter un *cast* explicite

```
short s = 65; // cas particulier affectation int "petit"
s = 1000000; // provoque une erreur de compilation
int i = 64;
byte b = (byte)(i + 2); // b = 66
char c = i; // caractère dont le code est 64 → '@'
b = (byte)128; // b = -128 !
```


Casts entre entiers et caractères

- La correspondance **char** → **int**, **long** s'obtient par *cast* implicite
- Les correspondances **char** → **short**, **byte** , et **long**, **int**, **short** ou **byte** → **char** nécessitent un *cast* explicite (entiers sont signés et pas les **char**)

```
int i = 80;
char c = 68; // caractère dont le code est 68
c = (char)i
i = c;
short s = (short)i;
char c2 = s; // provoque une erreur
```

Opérateurs

- Les plus utilisés
  - *Arithmétiques*
    - `+ - * /`
    - `%` **(modulo)**
    - `++ --` (pré ou post décrementation)
  - *Logiques*
    - `&&` (et) `||` (ou) `!` (négation)
  - *Relationnels*
    - `== != < > <= >=`
  - *Affectations*
    - `= += -= *=` ...

Opérateurs

Ordre de priorité des opérateurs

Postfixés	`[] . (params) expr++ expr--`
Unaires	`++expr --expr +expr -expr ~ !`
Création et cast	`new (type)expr`
Multiplicatifs	`* / %`
Additifs	`+ -`
Décalages bits	`<< >>`
Relationnels	`< > <= >= instanceof`
Egalité	`== !=`
Bits, et	`&`
Bits, ou exclusif	`^`
Bits, ou inclusif	`|`
Logique, et	`&&`
Logique, ou	`||`
Conditionnel	`? :`
Affectation	`= += -= *= /= %= &= ^= |= <<= >>=`

Les opérateurs d'égales priorités sont évalués de gauche à droite, sauf les opérateurs d'affectation, évalués de droite à gauche

03/10/2002 © P. Itey - INRIA Programmation Java - page 165

Déclarations

- Avant toute utilisation dans un programme une variable doit être déclarée
- syntaxe: ***type identificateur***
  - type : un type primitif ou un nom de classe
- Exemples

```
byte age;
boolean jeune;
float poids;
double x, y ,z;
```

- Une variable est accessible (visible) depuis l'endroit où elle est déclarée jusqu'à la fin du bloc où sa déclaration a été effectuée

Affectation

- Syntaxe : *lvalue* **=** *expression*

  lvalue est une expression qui doit délivrer une variable (par exemple un identificateur de variable, élément de tableau...., mais pas une constante)

- Exemples

```
int age;
age = 10;
boolean jeune = true;
float poids = 71.5f;
float taille = 1.75f;
float poidsTaile = poids / taille;
```

- Attention en JAVA comme en C, l'affectation est un opérateur. L'affectation peut donc être utilisée comme une expression dont la valeur est la valeur affectée à la variable

```
i = j = 10;
```

bloc d'instructions - instruction composée

- *permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction*

- syntaxe:

```
{
   séquence d'énoncés
}
```

- exemple

```
int k;
{
        int i = 1;
        int j = 12;
        j = i+1;
        k = 2 * j - i;
}
```


Instruction conditionnelle - instruction **if**

- Syntaxe

if (*expression booléenne* **)** *instruction1*

ou bien

if (*expression booléenne* **)**
instruction1
else
instruction2

- exemple

```
if (i==j){
   j = j -1;
   i = 2 * j;
}
else
   i = 1;
```

Un bloc car *instruction1* est composée de deux instructions

boucle tantque ... faire - instruction **while ()**

- Syntaxe **while (** *expression booléenne* **)**
  instruction

- Exemple

```
int i = 0;
int somme = 0;
while (i <= 10){
    somme += i;
    i++;
}
System.out.println("Somme des 10 premiers entiers" + somme);
```


boucle répeter ... jusqu'à – instruction **do while ()**

- Syntaxe **do**
  instruction
  while (*expression booléenne* **) ;**

- Exemple

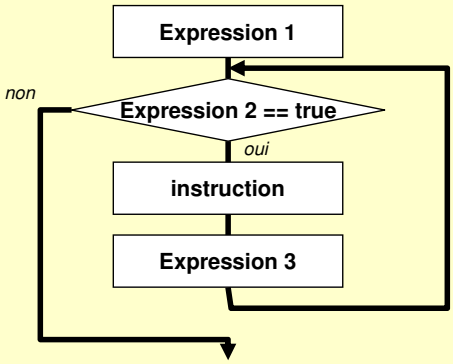
```
int i = 0;
int somme = 0;
do
{
    somme += i;
    i++;
}  while (i <= 10);
System.out.println("Somme des 10 premiers entiers" + somme);
```

Flot de contrôle

boucle pour – instruction **for**

- Syntaxe **for (***expression1* **;** *expression2***;** *expression3***)**
  instruction

- Exemple




```
int  i;
int somme = 0;
for (i = 0; i <= 10; i++)
   somme += i;

System.out.println("Somme des 10 premiers entiers" + somme);
```


Entrées/sorties sur console

Affichage sur la console

System.out.print(*chaîne de caractères à afficher***)**
System.out.println(*chaîne de caractères à afficher***)**

- chaîne de caractères peut être :
  - *une constante chaîne de caractères (String)*
    `System.out.println("coucou");`
  - *une expression de type String*
    `System.out.println(age);`

    Ici `age` est une variable de type **int**
    Elle est **automatiquement** convertie en **String**
  - *une combinaison (concaténation) de constantes et d'expressions de type String. La concaténation est exprimée à l'aide de l'opérateur +*

```
System.out.println("L'age de la personne est " +
    age + " son poids " + poids);
```

`age` (int) et `poids` (float) sont **automatiquement** converties en **String**

Entrées/sorties sur console

- Lecture de valeurs au clavier
  - *classe* **`LectureClavier`** *facilitant la lecture de données à partir du clavier. Définit une méthode de lecture pour les types de base les plus couramment utilisés (int, float, double, boolean, String)*

```
System.out.print("entrez un entier : ");
int i = LectureClavier.lireEntier();
System.out.println("entier lu : " + i);

String s = LectureClavier.lireChaine("entrez une chaine :");
System.out.println("chaine lue : " + s);

double d = LectureClavier.lireDouble("entrez un réel (double) : ");
System.out.println("réel (double) lu : " + d);


boolean b = LectureClavier.lireOuiNon("entrez une réposne O/N : ");
System.out.println("booleen lu : " + b);
```

`LectureClavier` n'est pas une classe standard de java. Pour l'utiliser vous devrez la récupérer sur le site Web de cet enseignement et l'intégrer à vos programme. Sa raison d'être est que dans les versions initiales de Java il n'y avait pas de moyen "simple" de faire ces opération. Ce n'est plus le cas, depuis la version 5 de Java et l'introduction de la classe `Scanner` (du package java.util).

Mon premier programme Java

Le code de la classe doit être enregistré dans un fichier de même nom (casse comprise) que la classe

`HelloWorld.java`

(3)

(1) Tout code java doit être défini à l'intérieur d'une classe

```
public class HelloWorld {

   public static void main(String[] args) {

      int i = 0;

      while (i < 5) {
         System.out.println("Hello World !");
         i++;
      }

   }
}
```

(4) Le point d'entrée pour l'exécution est la méthode `main()`

(2) La description de la classe est effectuée à l'intérieur d'un bloc { }

Compilation :
`javac HelloWorld.java`

`HelloWorld.java` → javac → `HelloWorld.class`

Exécution :
`java HelloWorld` → java →

```
Hello World !
Hello world !
Hello World !
Hello World !
Hello World
```