Types de base

Type	Déclarateur	Constantes	Opérateurs principaux
Booléens	**boolean**	true false	&& *(et)* || *(ou)* ! *(non)*
Entiers	**int**	132 -67 3e8	- *(unaire, binaire)* + * / % *(modulo)* -- ++ > >= < <= == *(égalité)* != *(inégalité)*
Octets	**byte**	-32 0xA8	- + * / % -- ++ > >= < <= == !=
Entiers courts	**short**	252 756	- + * / % -- ++ > >= < <= == !=
Entiers longs	**long**	-25244L 756443L	- + * / % -- ++ > >= < <= == !=
Flottants double précision	**double**	2.23 -4.32e-12	- + * / > >= < <= == !=
Flottants simple précision	**float**	8,42f -6,63e-4f	- + * / > >= < <= == !=
Caractères (unicode)	**char**	'a' 'Z' \0xABCE *(unicode)*	> >= < <= == !=

Variables

Déclaration : **<type> <identificateur> [= <expression>];**
où <expression> est la valeur d'initialisation.
```
int monEntier; // valeur par défaut : 0
double monFlottant = 2.212;
char monChar = 'a';
```
Affectation : **<variable> = <expression>;**
```
monEntier = 8;
```
Variantes : += -= etc. *(comme en CC++)*

Tableaux

Déclaration : **<type>[] <identificateur> [= <elt0>, <elt1>, ...];**
où <elt$_i$> sont des constantes.
```
int[] monTableau;
double[] monTableau2 = 3.12, 4.11, 8.3;
double[][] maMatrice = new double[3][3];
maMatrice[1][2]=12; // ligne 1 , colonne 2
```
Allocation : **<tableau> = new <type>[<taille>];**
```
monTableau = new int[20];
```
Affectation : **<tableau>[<index>] = <expression>;**
```
monTableau[5] = 8;
```
Accès : **<tableau>[<index>]**
```
System.out.println(monTableau[5]);
```
Remarques : *Les <index> vont de 0 à <taille>-1.*
Les tableaux ont une taille statique.
```
monTableau2[4]=1.5; //=> génère une exception ArrayOutOfBoundsException
```
Taille : **<tableau>.length**
```
System.out.println("taille = "+monTableau.length);
```

Tableaux Dynamiques : class `ArrayList`

Import : **import java.util.ArrayList;**
Déclaration : **ArrayList<type objet> <nom tabdyn>;**
```
ArrayList<Integer> monTabdyn;
ArrayList<String> monTabdyn2;
```
Allocation : **<tabdyn> = new ArrayList<type>();**
```
monTabdyn2 = new ArrayList<String>();
```
Ajout : **<tabdyn>.add(<expression>);**
```
monTabdyn.add(8); monTabdyn2.add("pingouin");
```
Accès : **<tabdyn>.get(<index entier>)**
```
System.out.println(monTabdyn.get(0));
```
Remarque : *Les <index> vont de 0 à <taille>-1.*
```
monTabDyn.get(5); //=> génère une exception IndexOutOfBoundsException
```
Taille : **<tabdyn>.size()**
```
System.out.println("taille = "+monTabdyn.size());
```


Chaînes de caractères

Chaînes constantes : **java.lang.String**
```
String maChaine="This is a nice string";
```
Taille : `maChaine.length()`
Comparaison : `maChaine.equals(autreChaine)`
Chaînes modifiables : **java.lang.StringBuffer**

Affichage console

Sortie standard :
```
System.out.print("Bonjour"); // sans retour de ligne
System.out.println(" le monde "+2+"!"); // avec retour de ligne
```
Sortie d'erreur :
```
System.err.println("Problème!");
```

Blocs

{ <séquence d'instructions> }
Exécuter <séquence d'instructions> (qui elle-même peut contenir des blocs imbriqués)

Portée des variables : Toute variable déclarée dans le bloc est visible uniquement dans le bloc (tant qu'elle n'est pas cachée). Sa portée s'étend de l'instruction qui suit la déclaration jusqu'à l'accolade fermante du bloc courant.
Cas particulier : Le corps d'une méthode ou d'un constructeur est un bloc lexical.

```
 1: public void methode(int a, int b) {
 2:   int c = a + b;
 3:   if (c+a>b) {
 4:     int a = 2*a-b;
 5:     c = a+b;
 6:   } else {
 7:     long b = 8;
 8:     c = a + b;
 9:   }
10: }
```

Portée de a : [1-10] (mais masqué entre 4 et 6)
Portée de b : [1-10] (mais masqué entre 7 et 9)
Portée de c : [2-10]
Portée du second a : [4-6]
Portée du second b : [7-9]

Structures de Contrôle

Alternatives

if(<condition booléenne>) <conséquent> [else <alternant>]
Si la condition est varie alors exécuter le <conséquent>, sinon exécuter l'<alternant> (version avec else) ou ne rien faire (version sans else).

Remarque : <conséquent> et <alternant> sont ou bien une instruction <instr> ;
ou bien un bloc { <instr1> ; <instr2> ; ... }.

```
if (monEntier>12)
  // conséquent
  System.out.print("grand ");
else
  // alternant
  System.out.println("petit ");
// (attention: ce n'est pas l'alternant !)
System.out.println(monEntier);
```

```
int a = 8; int b = 9;
if (b>a) {
 int temp = a;
 a = b;
 b = temp;
}
System.out.println("a="+a+", b="+b);
```

Traitements par cas

```
switch(<expression int char byte short>) {
case <valeur1> : <sequence1>; break;
case <valeur2> : <sequence2>; break;
...
default : <instructions par défaut>;
}
```
En fonction de la valeur de l'<expression>, exécuter <sequence1> (une séquence d'instructions) si <valeur 1>, <sequence1> si <valeur 2>, etc. Le cas par défaut (default) est inconditionnel.

Remarque : si on ne met pas de `break` alors plusieurs cas (en particulier le `default`) peuvent s'exécuter (tous les cas suivants sont évalués, et s'ils sont vrais les séquences correspondantes sont exécutées, jusqu'au premier `break` (ou à la fin du bloc).

```
int monEntier = 1;
String chaine = "";
switch(monEntier) {
  case 0 : chaine="zéro"; break;
  case 1 : chaine="un"; break;
  case 2 : chaine="deux"; break;
  default : chaine="plutôt grand";
}
System.out.println(monEntier+" est "+chaine);
```

Boucles `while`

`while (<condition>) <corps>`
Remarque : `<corps>` est une instruction ou un bloc.

```
int i= 0;
// on suppose un tableau tab
while(i<tab.length)
  System.out.println(tab[i++]);
```

```
int j = 10; int total = 0;
while(j>0) {
 total += total*j;
 j--;
}
System.out.println("total = "+total);
```

Variante : **`do <corps> while (<condition>)`**
Permet de s'assurer que le `<corps>` est exécuté au moins une fois.

```
int j=0, total=1;
do {
  total+=total*j;
  j--;
} while(j>0);
System.out.println("total (dw)="+total);
```

```
j=0; total=1;
while (j>0) {
  total+=total*j;
  j--;
}
System.out.println("total (w)="+total);
```

Boucles `for`

`for(<init>; <condition>; <increment>) <corps>`
Tant que la condition de poursuite est vraie, `<corps>` est exécuté. Après exécution de `<corps>`, `<increment>` est exécuté à son tour. Les variables déclarées et initialisées dans `<init>` sont visibles dans `<corps>`.

Remarques : Déclarations et mises à jour multiples : séparées par des virgules. `<corps>` est une instruction simple ou un bloc.

```
for(int i=0;i<tab.length;i++)
// on suppose un tableau tab
  System.out.println(tab[i]);
```

```
int total1=0, total2=0;
for(int j=10;j>0;j--) {
 total1 += total1*j;
 total2 += total2+j;
}
System.out.println("total1 = "+total1);
System.out.println("total2 = "+total2);
```

Boucles `for-each` (depuis java 1.5)

`for(<type> <variable> : <collection>) <corps>`
Pour chaque élément de la `<collection>` (tableau ou collection ArrayList, HashSet, etc.), exécuter `<corps>` dans lequel `<variable>`, de type `<type>` est visible et reçoit l'élément courant.

```
int taille = 0;
for(Object obj : tab) {
  System.out.println("L'élément courant est: "+obj);
  taille++;
}
System.out.println("La taille du tableau est : "+taille);

ArrayList<String> maListe = new ArrayList<String>();
maListe.add("Kant"); maListe.add("Aristote"); maListe.add("Platon");
for(String philosophe : maListe)
 System.out.println(philosophe+" est un philosophe connu");
```


Exceptions : `try/catch`

```
try {
// code levant Exception1, Exception2, etc.
} catch(Exception1 e1) { /* code si e1 */
} catch(Exception2 e2) { /* code si e2 */
} // etc.
```

```
try { // on suppose tabdyn un tableau dynamique et nb un entier
  for(int i=0;i<nb;i++)
    System.out.println(tabdyn.get(i)); // throws IndexOutOfBoundsException
} catch(IndexOutOfBoundsException e) {
    System.err.println("indice nb hors tableau: "+nb);
    e.printStackTrace(System.err);  // trace de pile sur la sortie d'erreur
    System.exit(1); // quitter le programme avec une erreur
}
```

Méthodes statiques

```
package matrices;            /* matrices/Matrices.java */
public class Matrices {
   // exemple de fonction;  appel: mat2=Matrices.mul(x,mat)
   public static double[][] mul(double x, double[][] mat) {
     double[][] res = new double[mat.length][mat[0].length];
     for(int i=0;i<mat.length;i++)
        for(int j=0;j<mat[i].length;j++) res[i][j]=x*mat[i][j];
     return mat;
   }
   // exemple de procédure;  appel: Matrices.Print(mat);
   public static void print(double[][] mat) {
     for(int i=0;i<mat.length;i++) {
       System.out.print("[");
       for(int j=0;j<mat[i].length;i++) System.out.print("\t"+mat[i][j]);
       System.out.println("\t]");
     }
    }
}
```

Point d'entrée de programme

```
package matrices;            /* matrices/MatricesMain.java */
public class MatricesMain {
  public static void main(String[] args) {
    double x = 0.0;
    try {
      x = Double.parseDouble(args[0]); // throws NumberFormatException
    } catch(NumberFormatException e) {
      System.out.println("Syntaxe:matrices.MatricesMain <x> avec <x> flottant");
      System.exit(1);
    }
    double[][] mat1 = { { 1,2,3 }, { 4,5,6 }, { 7,8,9 } };
    Matrices.print(mat1);
    System.out.println("* "+x);
    System.out.println("=");
    double[][] mat2 = Matrices.mul(x,mat1);
    Matrices.print(mat2);
  }
}
```

Compilation : **`javac matrices/*;`**
Lancement : **`java matrices.MatricesMain;`**