Chapitre 2

Éléments de base du langage Java

Règles de base

- Java est sensible à la casse
- Les blocs de code sont encadrés par des accolades **{}**
- Chaque instruction se termine par un point virgule **;**
- Une instruction peut tenir sur plusieurs lignes:
  - Exemple :
    - l'instruction « int x=3; » peut être écrite :

```
int
x
=
3;
```

Identificateurs

- Chaque objet, classe, programme ou variable est associé à un nom : un identificateur.
- Un identificateur peut se composer de tous les caractères alphanumériques et des caractères _ et $.
- Le premier caractère doit être une lettre, _ ou $.
- Un identificateur ne peut pas être un mot réservé du langage Java (ex: abstract, new, return, try, …)

Commentaires

- Ils sont sauté par le compilateur (pas de ';' à la fin)
- Il existe trois types de commentaires en Java :
  1. **Monoligne**. *Exemple* : int N=1; **//** déclaration du compteur
  2. **Multiligne**. *Exemple* : **/*** commentaires ligne 1
     commentaires ligne 2 ***/**
  3. **De documentation automatique.**
     Exemple :

```
/**
* commentaire de la méthode
* @param val la valeur a traiter
* @return Rien
* @deprecated Utiliser la nouvelle méthode XXX
*/
```

Variables

- Une variable possède un **nom**, un **type** et une **valeur**.
- Pour utiliser une variable il faut la déclarer pour lui réserver de la mémoire avant de lui affecter une valeur.
- Exemples :
  - int x, y, somme=0;
  - String nom;
  - Date dateNaissance;
- Remarque :
  - Une variable n'est visible que dans le bloc ou elle est déclarée.
  - Elle peut être simple (int, long,..) ou objet (String, Date,…)

Types élémentaires

- Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution ce qui permet à Java d'être indépendant de la plate-forme sur lequel le code s'exécute.

Entiers

Type	Désignation	Longueur	Valeurs
byte	octet signé	8 bits	-128 à 127
short	entier court signé	16 bits	-32768 à 32767
int	entier signé	32 bits	-2147483648 à 2147483647
long	entier long	64 bits	-9223372036854775808 à 9223372036854775807

Types élémentaires

Réels

Type	Désignation	Longueur	Valeurs
float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308

Autre

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique	1 bit	true ou false	pas de conversion possible vers un autre type
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java

Format des types élémentaires

- **Le format des nombres entiers :**
  - Un nombre de type **byte**, **short**, **int** et **long** peuvent être codés en **décimal**, **hexadécimal** (commence par **0x**) ou **octal** (commencer par **0**).
  - Le suffixe **l** ou **L** permet de spécifier que c'est un entier **long**.
  - Exemple: long x=584756L
- **Le format des nombres décimaux :**
  - Les types **float** et **double** stockent des nombres flottants.
  - Ils doivent posséder soit un point, un exposant ou l'un des suffixes **f**, **F**, **d**, **D**.
  - Exemples : **float** pi = 3.14**f**; **double** v = 3**d** **;**
    float f = +.1**f** , d = 5e3**f**; //5 x 10^3

Format des types élémentaires

- Remarque :
  - Il est possible de préciser des nombres qui n'ont pas le partie entière ou décimale.
  - Par défaut un littéral est de type double : pour définir un float il faut le suffixer par la lettre f ou F.
  - Exemple :double w = 1.1;
  - Attention ! float pi = 3.14; // provoque une **erreur** à la compilation
  - Solution : float pi = 3.14f;

Format des types élémentaires

- **Le format des caractères :**
  - Un caractère Unicode est codé sur **16 bits**
  - Il doit être **entouré** par des **apostrophes**.
  - Définition d'un caractère : char touche = '%';
  - Une valeur de type char peut être considérée comme un **entier** non négatif de **0** à **65535**.
  - Les caractères 0 à 255 correspondent à ASCII
  - La conversion implicite par affectation n'est pas possible.
- Liste des caractères Unicode:

```
for(int i=0;i<Character.MAX_VALUE;i++)
     System.out.println(i+" : "+(char)i);
```

Format des types élémentaires

- Extrait de l'exécution :
  - ...
  - ا : 1575
  - ب : 1576
  - ....
  - ج : 1580
  - ح : 1581
  - ...
- Exemple : la lettre arabe JIM (ج)
  - System.out.println((char)1580); //décimal
  - System.out.println("\u062C"); // hexadécimal

Format des types élémentaires

- Les caractères spéciaux dans les chaînes:
  - Apostrophe : \'
  - Guillemet : \"
  - Anti-slash : \\
  - Tabulation : \t
  - retour arrière (backspace) : \b
  - retour chariot : \r
  - saut de page (form feed) : \f
  - saut de ligne (newline) : \n
  - caractère ASCII ddd (octal) : \0ddd
  - caractère ASCII dd (hexadécimal) : \xdd
  - caractère Unicode dddd (hexadécimal) : \udddd

Format des types élémentaires

- Exemples:
  - Pour afficher Bonjour "Ali"
    - ➔ System.out.println("Bonjour \"Ali\" ");
  - Pour afficher Bonjour
    Ali
    - ➔ System.out.println("Bonjour \n Ali ");
  - Pour afficher c:\programmes\java
    - ➔ System.out.println("c:\\programmes\\java");

Chaînes de caractères

- Une chaîne de caractères stocke une séquence de caractères alphanumériques.
- Définition d'une chaîne : String texte = "bonjour";
- Le compilateur Java remplace les constantes chaînes par des objets de type String.
- Exemple : String texte = "tata".**replace**('a','o');
- Remarque :
  - La classe String, est immuable, c'est-à-dire que son contenu ne peut pas être modifié une fois créé.
  - Pour toute modification un autre chaine doit être crée

Chaînes de caractères

- Quelques méthodes de la classe String:
  - int **length**() : Retourne la longueur de la chaine.
  - String **toUpperCase**() : transforme notre chaine en majuscule.
  - String **toLowerCase**() : transforme notre chaine en miniscule.
  - int **compareTo**(String chaine2) : Compare deux chaines alphabetiquement.
    - Si le résultat **<** 0 : notre chaine précède chaine2
    - Si le résultat **>** 0 : notre chaine2 précède notre chaine
    - Si le résultat **=** 0 : les deux chaines sont égales
- Exemple : String ch1="Ali", ch2="Bachir"
  - ch1.compareTo(ch2) retourne un entier négatif car « Ali » précède « Bachir » en ordre alphabétique.

Chaînes de caractères

- Pour modifier le contenu d'une chaine de caractère sans créer une nouvelle, utilisez la classe StringBuider
  - Exemple :

```
StringBuilder sb = new StringBuilder("Bonjour ");
sb.append("monde");  // Bonjour monde
sb.insert(7, " le");  // Bonjour le monde
sb.delete(8, 16);  // Bonjour
sb.reverse();  //ruojnob
System.out.println(sb.toString());
```

- Remarque :
  - Dans le cas de traitements parallèles concurrents (threads) utiliser la classe StringBuffer qui contient des méthodes semblables à StringBuilder.

Opérateurs

- Les opérateurs arithmétiques se notent :
  - **+** (addition)
  - **-** (soustraction)
  - ***** (multiplication)
  - **/** (division)
  - **%** (reste de la division)
- Ils peuvent se combiner à l'opérateur d'affectation.

Affectation

- le signe **=** est l'opérateur d'affectation et s'utilise avec une expression de la forme ***variable = expression***.
- L'opération d'affectation est associatif de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire : ***x = y = z = 0***;
- Liste d'opérateurs abrégée d'affectation :

Opérateur	Exemple	Signification
=	a=10	
+=	a+=10	a = a + 10
-=	a-=10	a = a - 10
*=	a*=10	a = a * 10
/=	a/=10	a = a / 10
%=	a%=10	Reste de la division
^=	a^=10	a = a ^ 10

Comparaisons

- Java propose des opérateurs pour toutes les comparaisons :
  - **>** (strictement supérieur)
  - **<** (strictement inférieur)
  - **>=** (supérieur ou égal)
  - **<=**(inférieur ou égal)
  - **==** (égalité)
  - **!=** (diffèrent de)
  - **&&** (ET logique)
  - || (OU logique)
  - **!** (Négation)
  - & (ET binaire)
  - ^ (OU exclusif binaire)
  - | (OU binaire)
  - ?: (opérateur conditionnel)
    - **Exemple** : ***a ? b : c*** équivalent à : si a alors b sinon c
    - **Remarque** : b et c doivent retourner le même type

Comparaisons

- Priorités des opérateurs du plus au moins prioritaire :
  1) les parenthèses : **( )**
  2) les opérateurs d'incrémentation : **++** , **--**
  3) les opérateurs de multiplication, division, et modulo :*****, **/** , **%**
  4) les opérateurs d'addition et soustraction :**+**, **-**
  5) les opérateurs de décalage : **<<** et **>>**
  6) les opérateurs de comparaison : **<** , **>**,**<=**,**>=**
  7) les opérateurs d'égalité : **==**, **!=**
  8) l'opérateur OU exclusif : **^**
  9) l'opérateur ET : **&**
  10) l'opérateur OU : **|**

Comparaisons

11) l'opérateur ET logique : **&&**

12) l'opérateur OU logique : **||**

13) les opérateurs d'assignement : **=**, **+=** ,**-=**

- Remarque 1:
  - Les parenthèses ayant une forte priorité, l'ordre d'interprétation des opérateurs peut être modifié par des parenthèses.

Comparaisons

- Remarque 2 :
  - L'opérateur **==** teste l'égalité entre les types primitifs comme les entiers, réels,...
    - Exemple : int x,y;

```
....
if (x==y) {.....}
```

  - Pour les classes la méthode equals() est à utiliser
    - Exemple : String **ch1,ch2**;

```
....
if (ch1.equals(ch2)){....}
```

Lecture & Ecriture des variables

- Ecriture : System.out.println(expression)
  - Exemple : System.out.println(" le résultat est " + R)
- Lecture : Se fait grâce à la classe **Scanner**
  - next() : permet de lire un mot de type **String**
  - nextLine() : permet de lire une chaine de caractère
  - nextInt() : permet de lire un **int**
  - nextLong() : permet de lire un **long**
  - nextDouble() : permet de lire un **double**
  - nextFloat() : permet de lire un **float**

Lecture & Ecriture des variables

- Utilisation :
  1) La classe doit être importée du package java.util
  2) Elle doit être ensuite instanciée (création et initialisation de la variable)
  3) Finalement ses méthodes de lecture peuvent être appelées

Lecture & Ecriture des variables

- Exemple:

```
import java.util.Scanner;
public class SaisieClavier{
    public static void main(String [] args){
      Scanner s = new Scanner(System.in);
      System.out.println("Entrez un entier:");
      int n = s.nextInt();
      System.out.println(" Le carré de :"+ n +" est :" + n*n);
       s.close();
    }
}
```


Opérations arithmétiques

- Remarque :
  - L'opérateur + permet de concaténer une chaine de caractères avec d'autres chaines de caractères ou types élémentaires.
  - Exemple :
    - 1+2 donne 3
    - "1" + "2" donne "12"
- On distingue entre deux arithmétiques :
  - L'arithmétique entière
  - L'arithmétique en virgule flottante

Opérations arithmétiques

- **Arithmétique entière**
  - Conversion implicite vers le type **int** des types numériques (promotion entière) pour renforcer la sécurité du code
  - **Exemple** :

```
short z , x = 5 , y = 15;
z = x + y ; //provoque une erreur à la compilation
Incompatible type for =. Explicit cast needed to
convert int to short.
z = x + y ; //erreur à la compilation ^ 1 error
```

  - **Explication** :
    - Le résultat est promus en type int (32 bits) est affecté dans un type short (16 bits): incompatibilité !!
  - **Solution** : cast explicite : z = (short) (x + y);

Opérations arithmétiques

- La division par zéro pour les types entiers lève l'exception ArithmeticException
- **Exemple**

```
/* test sur la division par zero de nombres entiers */
class test3 {
      public static void main (String args[]) {
        int valeur=10;
        double resultat = valeur / 0; // provoque exeption
        System.out.println("Resultat = " + resultat);
      }
}
```

Opérations arithmétiques

- **L'arithmétique en virgule flottante**
  - la **division par zéro** des float ou double, ne produit pas d'exception mais une des 3 valeurs spéciales:
    1. **NaN** (**not a number** **:** nombre non défini):
       - ***Float.NaN*** ou ***Double.NaN***
    2. **Infinity** (**+∞ :** valeur positive supérieure au plafond du type):
       - ***Float.POSITIVE_INFINITY*** ou ***Double.POSITIVE_INFINITY***
    3. **- Infinity** (**-∞ :** valeur négative supérieure au plafond du type):
       - ***Float.NEGATIVE_INFINITY*** ou ***Double.NEGATIVE_INFINITY***

Opérations arithmétiques

- Exemple:
  - Dans ce programme nous divisons délibérément un float par zéro. Remarquez le résultat !

Tableau récapitulatif

X	Y	X/Y	X%Y
valeur finie	0	+/-∞	NaN
valeur finie	+/-∞	0	x
0	0	NaN	NaN
+/- ∞	valeur finie	+/-∞	NaN
+/- ∞	+/-∞	NaN	NaN

Solution : /* test sur la division par zéro de nombres flottants */

```
class test2 {
        public static void main (String args[]) {
                float valeur=10f;
                double résultat = valeur / 0;
                System.out.println(" Resultat = " + résultat);
        }
}
```

Opérations arithmétiques

- L'incrémentation et la décrémentation
  - Les opérateurs d'incrémentation et de décrémentation sont : ++n, n++ , --n , n--
  - L'opérateur ++ (resp. --) renvoie la valeur avant incrémentation s'il est postfixé et après incrémentation s'il est préfixé.
  - Exemple :

    y=x++ ⇔ y=x; x = x + 1;

    y=++x ⇔ x = x + 1; y=x;

Opérations arithmétiques

- **Exercice** : Quel sera le résultat de l'exécution du programme :

```
public class test4 {
 public static void main (String args[]) {
  int a=0,b=0;
  System.out.println("a = " + a + " b = " + b);
  a=b++;
  System.out.println("a = " + a + " b = " + b);
  a=++b;
  System.out.println("a = " + a + " b = " + b);
  a=a++;
  System.out.println("a = " + a + " b = " + b);

 }
}
```

Opérations arithmétiques

- Résultat:
  - int a=0; int b=0; ⇨ a=0 b=0
  - a=b++; ⇨ a=0 b=1
  - a=++b; ⇨ a=2 b=2
  - a=a++; ⇨ a=2 b=2

   attention : a ne change pas de valeur
  - Par contre l'instruction "a=++a" incrémente la valeur de a

Structures de contrôles

- Sont similaires aux autres langages de programmation.
- Sont composés de :
  1. **boucles** : permettent d'itérer sur un certain nombre d'instructions (while, do..while, for, for pour collections)
  2. **branchements conditionnels** : Permettent d'effectuer des tests (if, opérateur ternaire, switch)

Structures de contrôles

- Boucle WHILE:

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

- Boucle DO..WHILE:

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

  - **Similitudes** : Dans les deux boucles le code est exécuté tant que la condition est vraie. Lorsque la condition passe à faux le bloc du **While** est sauté.
  - **Différence** : Dans **do**…**While**, la boucle est au moins exécutée une fois quelque soit la valeur de la condition;

Structures de contrôles

- Boucle FOR normale :
  - Syntaxe : for (initialisation; condition; modification) {…}
  - Exemple :

```
for (int i =0; i<10; i++) {
  System.out.println(" le carré de"+i+ "est :"+i*i);
}
```

- Boucle for des collections :
  - Syntaxe : for (type variable : collection)
- Exemple :

```
int[] t= {23,1,12,33,41,57,16,27,81,19};
for (int x : t) {
  System.out.println(x);
}
```

Structures de contrôles

- Les branchements conditionnels Se composent :
  - Du mot clé **if**
  - De l'opérateur ternaire
  - Du mot clé **switch**

Structures de contrôles

- Tests avec IF :
  - Forme :

```
if (condition) { ... }
else  if (condition) { ... }
        else { ... }
```

  - Exemple :

```
if (moy>10)
    resultat= "V";
else
     resultat = "RAT";
```

- L'opérateur ternaire:
  - Forme :

```
condition ? valeur-vrai : valeur-faux
```

  - Facilité d'écriture pour un if avec un else.
  - Exemple :

```
resultat = moy>=10 ? "V" : "RAT";
```

```
System.out.println(moy>=10 ? "V" : "RAT");
```

Structures de contrôles

- Remarque :
  - Il ne faut jamais déclarer une variable à l'intérieur d'une boucle ou d'une condition !!

Structures de contrôles

- Tests avec Switch :
  - Switch permet d'effectuer des tests pour des valeurs constantes de type (byte, short, int, char, String).
  - Forme :

```
switch (expression)
{   case constante1 : instr11; instr12; break;
    case constante2 : ...
    default : ...
}
```

  - En l'absence du break, l'exécution passe au case suivant.
  - Il est possible d'imbriquer des switch

Structures de contrôles

- Exemple :

```
import java.util.Scanner;
public class SwitchDemo1 {
 public static void main(String[] args) {
     Scanner s = new Scanner(System.in);
     System.out.println("Fournir un nombre de 1 à 12 !");
     int n = s.nextInt();
     String mois;
     switch (n) {
            case 1: mois="Janvier"; break;
            case 2: mois="Février"; break;
            …
            case 12: mois="Décembre"; break;
            default: mois="Mois Invalide";break;
      }
     System.out.println(mois);
 }
}
```


Structures de contrôles

- Remarque : il est possible de regrouper des valeurs qui ont les mêmes traitement.
- Exemple :
  - Transformer l'exemple précédent pour indiquer la saison du mois entré.

Structures de contrôles

```
import java.util.Scanner;
public class SwitchDemo2 {
 public static void main(String[] args) {
     Scanner s = new Scanner(System.in);
     System.out.println("Fournir un nombre de 1 à 12 !");
     int n = s.nextInt();
     String saison;
     switch (n) {
             case 12 :
             case 1:
             case 2: saison ="Hiver"; break;
             case 3:
             case 4:
             case 5 : saison ="Printemps"; break;
             case 6:
             case 7:
             case 8 : saison ="Eté"; break;
             case 9:
             case 10:
             case 11 : saison ="Automne"; break;
             default: saison="Mois Invalide";
    }
   System.out.println(saison);
 }
}
```


Structures de contrôles

- Nouvelle syntaxe de switch
  - Utilisation d'une flèche « **->** » et suppression du « **break** »
  - Utilisation de la virgule « **,** » en cas de plusieurs valeurs
  - Permet d'assigner directement le résultat de chaque cas à une variable.
  - Renvoi de valeur automatique dans le cas d'une seule instruction
  - Renvoi avec le mot clé « **yield** » dans le cas de plusieurs instructions

Structures de contrôles

- **Exemple 1** :

```
import java.util.Scanner;
public class SwitchDemo1 {
 public static void main(String[] args) {
     Scanner s = new Scanner(System.in);
     System.out.println("Fournir un nombre de 1 à 12 !");
     int n = s.nextInt();
     String mois = switch (n) {
             case 1 ->"Janvier"; //retourne janvier
             case 2 ->"Février";
             …
             case 12->"Décembre";
             default: "Mois Invalide";
    }
    System.out.println(mois);
 }
}
```


Structures de contrôles

- **Exemple 1 v2**:

```
import java.util.Scanner;
public class SwitchDemo1 {
 public static void main(String[] args) {
     Scanner s = new Scanner(System.in);
     System.out.println("Fournir un nombre de 1 à 12 !");
     int n = s.nextInt();
     System.out.println(switch (n) {
             case 1 ->"Janvier";
             case 2 ->"Février";
             …
             case 12->"Décembre";
             default: "Mois Invalide";
     } );
 }
}
```

Structures de contrôles

- **Exemple 2** : regroupement des valeurs qui ont les mêmes traitement.

```
import java.util.Scanner;
public class SwitchDemo2 {
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      System.out.println("Fournir un nombre de 1 à 12 !");
      int n = s.nextInt();
      System.out.println(switch (n) {
         case 12, 1, 2 -> "Hiver"; //renvoi Hiver
         case 3, 4, 5 -> "Printemps";
         case 6, 7, 8 -> "Eté";
         case 9, 10, 11 -> "Automne";
         default -> "Mois Invalide";
      });
   }
}
```


Structures de contrôles

- Remarque :
  - Dans le cas de plusieurs instructions après la flèche :
    - Mettre des accolades
    - Utiliser le mot clé « yield » pour renvoyer le résultat
  - Exemple :

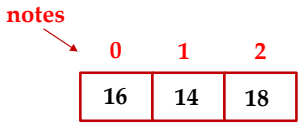
    case valeur -> {instructions;…; yield resulat;}

Tableaux

- Les tableaux sont des structures de données regroupant plusieurs valeurs de même type.
- On peut accéder à n'importe quelle valeur à partir de son rang ou indice.
- Exemples de déclarations:
  - float[] notes = new float[3] // tableau de 3 réels
  - int[] t=new int[10] // tableau de 10 entiers
  - String[] noms= new String[20] // tableau de 20 strings
- Remarque :
  - Une fois fixée, la taille d'un tableau ne peut changer.

Tableaux

- Exemple d'affectation :
  - notes[0]=16
  - notes[1]=14
  - notes[2]=18



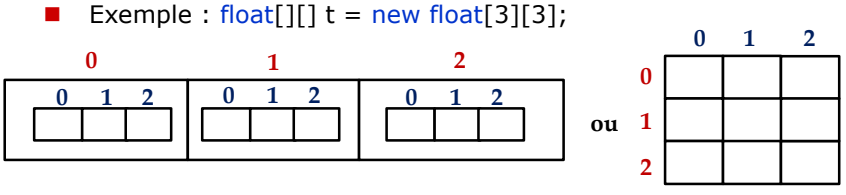
- Taille du tableau : propriété **length**
  - Exemple : **notes.length** renvoie **3**
- Parcours :
  - Requière une boucle avec un seul indice de 0 jusqu'à length-1

Tableaux : Exemple

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Combien d'éléments voulez vous ?");
        int n = s.nextInt(); //lecture de la taille souhaitée
        int[] t = new int[n]; // déclaration et réservation
        for (int i = 0; i < t.length; i++) {
            System.out.println("Entrer Element " + i + " : ");
            t[i] = s.nextInt();
        }
        System.out.println("Affichage du Contenu du tableau :");
        for (int i = 0; i < t.length; i++) {
            System.out.println("t[" + i + "] = " + t[i]);
        }
    }
}
```


Tableaux

- Les tableaux à deux dimensions sont déclarés comme étant un tableau de tableaux.
  - Exemple : float[][] t = new float[3][3];



- Taille du tableau : **length**
  - **t.length** renvoie ta taille de la 1ère dimension
  - **t[0].length, t[1].length,...**renvoient la taille de la 2eme dimension

Tableaux

- Parcours :
  - Une première boucle pour parcourir les cases de la première dimension (indice de 0 jusqu'à length-1)
  - Une deuxième boucles pour parcourir les contenus des cases qui sont des tableaux

t[0][0]; t[0][1]; t[0][2];
t[1][0]; t[1][1]; t[1][2];
t[2][0]; t[2][1]; t[2][2];

  - Forme :

```
for (int i = 0; i < t.length; i++) {
 for (int j = 0; j < t[i].length; j++) {
   // traitement sur t[i][j]
 }
}
```


Tableaux

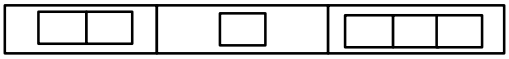
```
import java.util.Scanner;
public class Main {
 public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.println(" Taille de la première dimension?");
    int n = s.nextInt(); //lecture de la taille de la 1ere dim
    System.out.println(" Taille de la deuxième dimension ?");
    int m = s.nextInt(); //lecture de la taille de la 2eme dim
    int[][] t = new int[n][m]; // déclaration et réservation
    // parcours et remplissage du tableau
    for (int i = 0; i < t.length; i++) {
     for (int j = 0; j < t[i].length; j++) {
      System.out.println("Entrer Element " + i + " , "+ j + " : ");
      t[i][j] = s.nextInt();
     }
    }
```

Tableaux

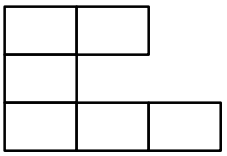
```
    // parcours et affichage du contenu du tableau
    System.out.println(" Affichage du contenu du tableau");
    for (int i = 0; i < t.length; i++) {
     for (int j = 0; j < t[i].length; j++) {
      System.out.println("t["+i+"][" +j+"] = "+ t[i][j]);
     }
    }
 } // fin main
}//fin classe
```


Tableaux

- Remarque : La taille des tableaux de la seconde dimension peut ne pas être identique pour chaque occurrence.



- Exemple :
  - int[][] t= new int[3][];
  - t[0] = new int[2];
  - t[1] = new int[1];
  - t[2] = new int[3];



- Parcours :

```
for (int i = 0; i < t.length; i++) {
 for (int j = 0; j < t[i].length; j++) {
   // traitement sur t[i][j]
 }
}
```

Tableaux

- Chaque élément du tableau est initialisé selon son type par l'instruction new :
  - 0 pour les numériques,
  - '\0' pour les caractères,
  - false pour les booléens et
  - null pour les chaînes de caractères et les autres objets.
- L'initialisation explicite d'un tableau. Exemple :
  - int[] tableau = {10,20,30,40,50};
  - int[][] tableau = {{5,1},{6,2},{7,3}};
  - int[][] tableau = {{4,2},{9},{6,3,4}};
- La taille du tableau est déduite automatiquement

Tableaux

- La boucle for des collections : permet d'itérer sur un tableau ou une collection par leurs contenus
- Exemple :

```
public class TestForArray {
 public static void main(String[] args) {
      int[] t1= {23,1,12,33,41,57,16,27,81,19};
      for (int x: t1) { System.out.println(x); }
      int[][] t2={{8,3,1},{4,5,7,12},{7,6},{9,15,87,19}};
      for (int[] tab : t2) {
        for (int x : tab) {
            System.out.print(x+" ");
        }
        System.out.println();
      }
}
```

Tableaux

- Exemple de déclaration de tableaux dans une méthode :
  - En paramètre : public void afficher(int[] t){ ... }
  - En retour : public int[] calculer(){ ... }
- Exemple d'appel :
  - e.afficher(new int[] {5,2,9});
- Remarque :
  - Les tableaux sont toujours transmis par référence puisque ce sont des objets.

Conversions de types

- On peut avoir besoin de convertir une variable d'un type vers un autre type.
- Deux cas de figure existent :
  - Conversion vers un type de taille plus grande : se font sans perte d'information, et sans besoin d'être spécifié.
  - Conversion vers un type de taille plus petite: peut entraîner une perte d'information, et doit être spécifiées par un opérateur de *cast*.
    - Exemple:
      - int entier = 5;
      - float flottant = (float) entier;

Conversions de types

- La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires :
  - **String** : pour les chaines de caractères Unicode
  - **Integer** : pour les valeurs entières (int)
  - **Long** : pour les entiers long signés (long)
  - **Float** : pour les nombres à virgules flottante (float)
  - **Double** : pour les nombres à virgule flottante en double précision (double)

Conversions de types

- Conversion d'une chaine de caractère en :
  - int : méthode **Integer.parseInt(chaine-à-convertir)**
  - long : méthode **Long.parseLong(chaine-à-convertir)**
  - float : méthode **Float.parseFloat(chaine-à-convertir)**
  - double: méthode **Double.parseDouble(chaine-à-convertir)**
- Exemple :
  - String ch="10";
  - int x=Integer.parseInt(ch)
- Remarque : L'instruction de conversion doit être obligatoirement être mise dans un bloc try/catch

Conversions de types

- Exemple :

```
try {
    x=Integer.parseInt(ch);
}
catch (Exception e) {
    System.out.println("erreur  :" + e.getMessage());
}
```


Conversions de types

- Conversion d'un **int/long/float/double** en chaîne de caractère String:
  - Première méthode : le concaténer avec une chaine vide
    - Exemple :
      - int x=10;
      - String ch=""+x;
  - Deuxième méthode : utiliser la méthode valueOf de String
    - Exemple :
      - int i = 10;
      - String montexte = new String();
      - montexte =montexte.**valueOf**(i);

Méthodes

- Une méthode est un sous-programme qui permet d'effectuer une tâche.
- Elle est composée d'une signature et d'un corps.
  - La signature est constituée de:
    - Une visibilité
    - Un type de retour
    - Un nom
    - En ensemble de paramètres
  - Le corps est un bloc de code délimité par des accolades

Méthodes

- Une méthode retourne le résultat grâce à ***return***
- Les instructions après return sont ignorées.
- Exemple 1:

```
public class Main {
    // déclaration
     public static int carré(int x) {    return x*x;      }

     public static void main(String[] args) {
       int nombre=3;
       //appel de la méthode
       int résultat = carré(nombre);
       System.out.println("le carré de "+nombre+" est :"+ résultat);

     }
}
```

Méthodes

- Si la méthode ne retourne rien, alors on utilise ***void***.
- Exemple 2:

```
public class Main {
    // déclaration
     public static void carré(int x) {
        int résultat = x*x;
        System.out.println("le carré de "+x+" est :"+ résultat) ;
     }

     public static void main(String[] args) {
        int nombre=3;
        //appel de la méthode
       carré(nombre);
     }
}
```

- **Remarque** :
  - l'instruction *return* dans une méthode de type *void* permet de quitter la méthode.

Méthode main

- Constitue le point d'entrée pour l'exécution
- Signature : public static void main (String args[]) {..}
- Cette signature doit être respectée sinon la machine virtuelle ne la reconnaitra pas.
- Le tableau « args » permet de recueillir des arguments en ligne de commande.

Méthode main

- Faire un programme qui calcule la moyenne de notes fournies en ligne de commande.
- Remarques:
  - Pour exécuter cet exemple , il faut se mettre en mode commande (ms dos)
  - Et taper par exemple :
    - *java Moyenne 10 12 14 14 10*
  - Le résultat sera affiché sous dos aussi comme suit :
    - *la moyenne est : 12.0*

Méthode main

```
public class Moyenne{
    public static void main(String[] args) {
      double somme=0,moyenne,note;
      try{
          for (String arg : args){
            note=Double.parseDouble(arg);
            somme=somme+note;
          }
         moyenne=somme/args.length;
         System.out.println("la moyenne est : "+moyenne);
      }
      catch(FormatException e){
         System.out.println("Erreur : "+e.getMessage());
      }
    }
}
```

Enumérations

- Lorsqu'une variable varie dans un ensemble de valeurs prédéfinis, il faut créer un type énumération.
- Une énumération est un ensemble fini de constantes numériques (correspondant à 0, 1, 2, …)
- Exemple :
  - public **enum** Jours { *LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE* }
  - Jours **jr**=Jours.*VENDREDI*; // exemple de variable
- Le nom de l'énumération précède toujours ses constantes (sauf dans Switch)

Enumérations : Exemple

```
import java.util.Scanner;
public class Main {
  public enum Saisons { automne, hiver, printemps, été}
  public static void main(String[] args) {
    Scanner s=new Scanner(System.in);
    System.out.println("saisir une saison :");
    String rep=s.next().toLowerCase();
    Saisons sz = Saisons.valueOf(rep);//peut générer exception
    switch (sz) {
      case automne : System.out.println("Début"); break;
      case hiver : System.out.println(" Froid"); break;
      case printemps : System.out.println(" Beau"); break;
      case été : System.out.println(" Chaud"); break;
    }
  }
}
```

Dates & Temps

- Les classes gérant la date et le temps font partie du package **java.time**
- Principales Classes :
  - **LocalDate** : Représente une date sans heure.
  - **LocalTime** : Représente une heure sans date.
  - **LocalDateTime** : Combine date et heure.
  - **ZonedDateTime** : Intègre un fuseau horaire pour des opérations à l'échelle mondiale.
- La classe **DateTimeFormatter** du package **java.time.format** permet de formater ou parser des objets des classes précédentes.

Dates & Temps : classe LocalDate

- Méthodes de la classe LocalDate :
  - now() : Renvoie l'instant actuel sous forme de LocalDate.
    - **Exemple** : LocalDate date = LocalDate.now();
  - of(int year, int month, int dayOfMonth) : Crée une instance de LocalDate avec l'année, le mois et le jour spécifiés.
    - Exemple : LocalDate date = LocalDate.of(2024, 2, 18);
  - plusDays(long daysToAdd) / minusDays(long daysToSubtract) : Ajoute/soustrait un nombre spécifié de jours à/from une date.
    - **Exemple** : LocalDate futureDate = date.plusDays(7);
    - LocalDate pastDate = date.minusDays(7);

Dates & Temps : classe LocalDate

- isBefore(LocalDate other) / isAfter(LocalDate other) : Compare deux dates pour déterminer si l'une est antérieure ou postérieure à l'autre.
  - Exemple : boolean isBefore = date.isBefore(futureDate);
  - Exemple : boolean isAfter = date.isAfter(pastDate);

Dates & Temps : classe LocalTime

- Méthodes de la classe java.time.LocalTime :
  - now() : Renvoie l'instant actuel sous forme de LocalTime.
    - **Exemple** : LocalTime time = LocalTime.now();
  - of(int hour, int minute) : Crée une instance de LocalTime avec l'heure et la minute spécifiées.
    - **Exemple** : LocalTime time = LocalTime.of(12, 30);
  - plusHours(long hoursToAdd) / minusHours(long hoursToSubtract) : Ajoute/soustrait un nombre spécifié d'heures à/from une heure.
    - **Exemple** : LocalTime futureTime = time.plusHours(3);
    - **Exemple** : LocalTime pastTime = time.minusHours(2);

Dates & Temps : classe LocalTime

- Méthodes de la classe java.time.LocalDateTime :
  - now() : Renvoie l'instant actuel sous forme de LocalDateTime.
    - Exemple : LocalDateTime dateTime = LocalDateTime.now();
  - of(int year, int month, int dayOfMonth, int hour, int minute) : Crée une instance de LocalDateTime avec l'année, le mois, le jour, l'heure et la minute spécifiés.
    - Exemple : LocalDateTime dateTime = LocalDateTime.of(2022, 1, 18, 12, 30);

Dates & Temps : classe LocalTime

- plusDays(long daysToAdd) / minusDays(long daysToSubtract) : Ajoute/soustrait un nombre spécifié de jours à/from une date et heure.
  - Exemple : LocalDateTime futureDateTime = dateTime.plusDays(7);
  - Exemple : LocalDateTime pastDateTime = dateTime.minusDays(7);

Dates & Temps : classe ZonedDateTime

- Méthodes de la classe java.time.ZonedDateTime
  - now() : Renvoie l'instant actuel sous forme de ZonedDateTime en tenant compte du fuseau horaire système.
    - Exemple : ZonedDateTime zonedDateTime = ZonedDateTime.now();
  - of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond, ZoneId zone) : Crée une instance de ZonedDateTime avec l'année, le mois, le jour, l'heure, la minute, la seconde, le nanoseconde et le fuseau horaire spécifiés.

Dates & Temps : classe ZonedDateTime

    - Exemple : ZonedDateTime zonedDateTime = ZonedDateTime.of(2022, 1, 18, 12, 30, 0, 0, ZoneId.of("Africa/Casablanca"));
  - withZoneSameInstant(ZoneId zone) / withZoneSameLocal(ZoneId zone) : Ajuste le fuseau horaire d'une ZonedDateTime tout en maintenant l'instant (withZoneSameInstant) ou en gardant la date et l'heure locales (withZoneSameLocal).
    - Exemple : ZonedDateTime newZoneInstant = zonedDateTime.withZoneSameInstant(ZoneId.of("America/New_York"));
    - Exemple : ZonedDateTime newZoneLocal = zonedDateTime.withZoneSameLocal(ZoneId.of("Asia/Tokyo"));

Dates & Temps : classe DateTimeFormatter

- Méthodes de la classe DateTimeFormatter :
  - ofPattern(String pattern) : Crée un formateur de date personnalisé en utilisant un motif spécifié.
    - Exemple : DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
  - format(TemporalAccessor temporal) : Convertit un objet LocalDate (resp. LocalDateTime, ZonedDateTime) en une chaîne de caractères
    - Exemple : LocalDateTime dateTime = LocalDateTime.now();
      String formattedDateTime = formatter.format(dateTime);

Dates & Temps : classe DateTimeFormatter

- parse(CharSequence text) : Convertit une chaîne de caractères en un objet TemporalAccessor en utilisant le format spécifié par le DateTimeFormatter.
  - Exemple :

```
String dateString = "18-01-2022 14:30:00";
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
LocalDateTime parsedDateTime = LocalDateTime.parse(dateString, formatter);
```

Dates & Temps : Exemple

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class GestionTempsExemple {
    public static void main(String[] args) {
        // Obtention de la date et de l'heure actuelles
        LocalDateTime maintenant = LocalDateTime.now();
        // Création d'un événement futur dans 7 jours
        LocalDateTime evenementFutur = maintenant.plusDays(7);
        // Formatage de la date et de l'heure actuelles
        DateTimeFormatter formateur = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
        String maintenantFormate = formateur.format(maintenant);
        System.out.println("Date et heure actuelles : " + maintenantFormate);
        // Formatage de l'événement futur
        String evenementFuturFormate = formateur.format(evenementFutur);
        System.out.println("Événement futur dans 7 jours : " + evenementFuturFormate);
        // Calcul de la différence entre la date actuelle et l'événement futur
        long joursJusquaEvenement = maintenant.until(evenementFutur).toDays();
        System.out.println("Jours jusqu'à l'événement : " + joursJusquaEvenement + " jours");
    }
}
```