LANGAGE JAVA : LES BASES DU LANGAGE

TABLE DES MATIÈRES

LA STRUCTURE DE BASE D'UN PROGRAMME EN JAVA

La structure générale d'un programme est la suivante :

```
/** commentaire d'entête permettant d'expliquer
      1. Le rôle du programme
      2. L'auteur du programme
      3. La date de création
      4. La date de mise à jour
      Etc..
*/

//Zone d'importation des classes utilisées dans le fichier
import … ;
```

```
public class Appli {

    public static void main(String[] args) {

            /* zone de déclaration des variables locales */
            Declaration1;
            Declaration2;

            /* CORPS DU PROGRAMME */
            Instruction1;
            Instruction2;

    }
} /* FIN */
```

Un programme JAVA s'écrit dans un fichier ayant comme extension **.JAVA**. Le nom du fichier doit porter le nom de la classe. Dans l'exemple ci-dessus, la classe s'appelle Appli, donc ce programme doit obligatoirement se trouver dans un fichier nommé « **Appli.java** »

LES COMMENTAIRES

Les commentaires permettent une relecture de programme pour le programmeur lui-même, mais aussi pour ceux qui vont relire le programme (travail en équipe). Un commentaire d'entête est nécessaire pour indiquer le nom du programmeur, la date, le nom du fichier, descriptif du programme. Les commentaires sont du texte simple et ne sont donc pas traités par le compilateur.

2 types de commentaires :

- // indiquent au compilateur que le reste de la ligne est un commentaire.
- /* …. */ indiquent au compilateur qu'à partir de /* le texte est un commentaire jusqu'au symbole */ permet des commentaires sur plusieurs lignes.

Netbeans ajoute automatiquement un commentaire d'entête, comme son nom l'indique en début de fichier.

```
/*
 * File:   Appli.java
 * Author: sgibert
 *
 * Created on 14 septembre 2018, 20:19
 *
 * Description du programme : explique ici ce que fait ton programme
 */
```

FONCTION MAIN()

Les applications console sont faites pour être exécutée dans la console Windows (cmd)

Elles commencent toujours par la fonction main () (principale). C'est le point d'entrée du programme. Les accolades { et } délimitent un bloc d'instructions.

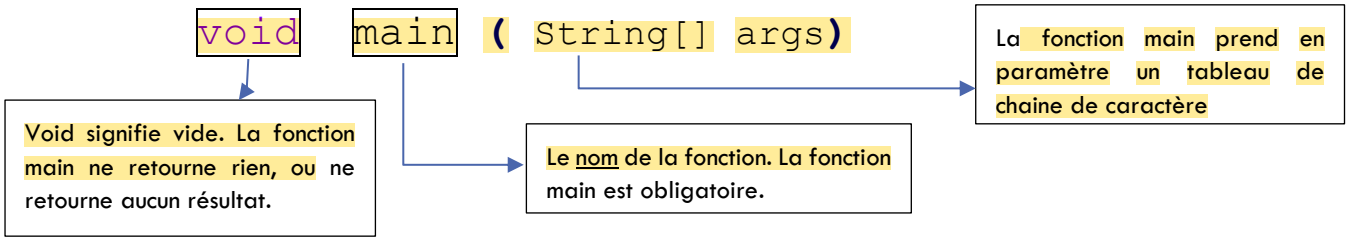
```
public static void main(String[] args) {   //cette ligne s'appelle la signature
      //ici début du bloc d'instruction attaché au main

      . . . //bloc d'instructions = plusieurs d'instructions

}     //ici fin du bloc d'instructions main
```

SIGNATURE DE LA FONCTION MAIN

Comment interpréter la signature ?



LES DÉCLARATIONS ET INSTRUCTIONS

Une instruction ou une déclaration se terminant toujours par un point-virgule (;)

LES DÉCLARATIONS DE VARIABLES

Rappel du cours d'algorithmique :

En résumé : **variable = NOM + TYPE + VALEUR**

Nous ajouterons en langage JAVA une notion supplémentaire : l'espace mémoire utilisé.

- L'espace mémoire utilisé par une variable dépend de son type. L'espace mémoire c'est la place que prennent en mémoire vos données. Par exemple, une variable de type **int** prend en mémoire 4 octets (1 octet = 8 bits) donc 32 bits.

CHOIX DU NOM OU DE L'IDENTIFIANT

Le nom d'une variable accepte les caractères suivants :

- a..z, A..Z
- 0..9 ➔ interdit au début du nom
- _ (le caractère souligné ou underscore) ➔ autorisé au début du nom

On utilise l'alternance de minuscule, majuscule, ou _ (underscore) pour améliorer la lecture du nom. Rappel : pour la relecture du programme, choisir un nom qui caractérise au mieux la variable.

Exemples:

nbrDeCaisses, poste_1, poste_123, etc…

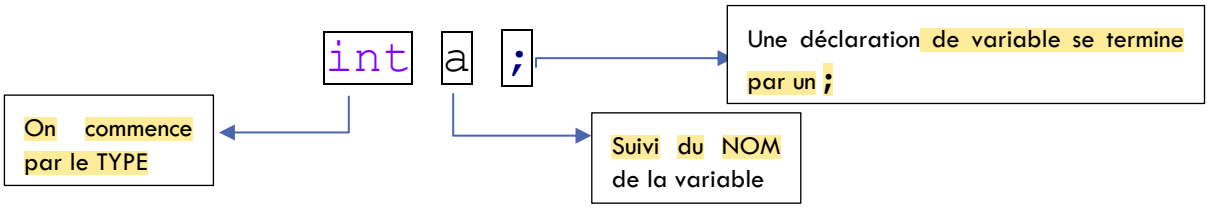
Pour éviter les confusions, un identificateur ne doit pas être similaire à un des mots-clés du langage dont voici la liste (en souligner les mots-clés du langage pour les types de données):

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

DÉCLARATION D'UNE VARIABLE

Pour pouvoir être utilisable, une variable doit d'abord être déclarée. On parle de déclaration de variable.

Pour déclarer une variable, il faut indiquer au minimum son type, son nom et terminer par un point-virgule. D'autres syntaxes sont possibles, voir exemple. La syntaxe de base est :



Exemple :

```
public static void main(String[] args) {
   int a;           //déclaration d'une variable a de type int sans valeur affectée.
   int b,c;         //déclaration de 2 variables b et c de type int sans valeur
   int d=10,e=3;    //déclaration et affectation, d vaut 10 et e vaut 3

   b = d + e ;       // calcul de « d + e » (10 + 3). Le résultat est affecté à « b »
   c = a + b ;       // Erreur : calcul impossible. « a » n'a pas de valeur
}
```

Dans le dernier exemple, on remarque que l'utilisation d'une variable sans valeur initiale n'est pas possible.

LES DIFFÉRENTS TYPES ET LEUR PLAGE DE LA VALEUR

Tableau récapitulatif:

Type JAVA	Type Algo	Taille	Plage de valeurs
byte	Entier	1 octet	-128 .. 0 .. 127 ➔ (-2^8)/2 .. 0 .. ((2^8)/2)-1
short	Entier	2 octets	-32768 à 32767 ➔ (-2^16)/2 .. 0 .. ((2^16)/2)-1
int	Entier	4 octets	-2147483648 à 2147483647 ➔ (-2^32)/2 .. 0 .. ((2^32)/2)-1
long	Entier	8 octets	(-2^64)/2 .. 0 .. ((2^64)/2)-1
boolean	Booléen	1 octet	true ou false.
float	Réel	4 octets	1.2e-38 à 3,4e38. Précisions : 6 chiffres après la virgule
double	Réel	8 octets	2.2e-308 à 1.8e308. Précisions: 15 chiffres après la virgule
char		2 octets	Permets de représenter un caractère Unicode (65536 caractères)

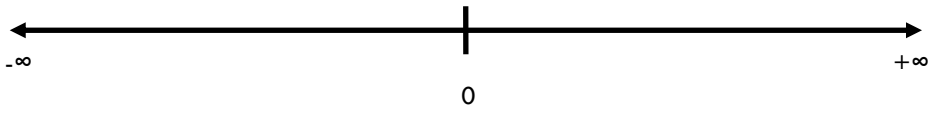
DÉTAILS SUR LES DIFFÉRENTS TYPES

- **byte, short, int et long:** représente un entier. Par défaut, ces types de données peuvent recevoir des valeurs négatives. Dans le cas d'une valeur négative, il faut ajouter le signe (-) devant la valeur. Pour une valeur positive, il est possible, mais inutile, d'ajouter le signe (+) devant la valeur.
  Exemple :

```
short a=-10 ;
```

Principe de la représentation des entiers en mathématique :

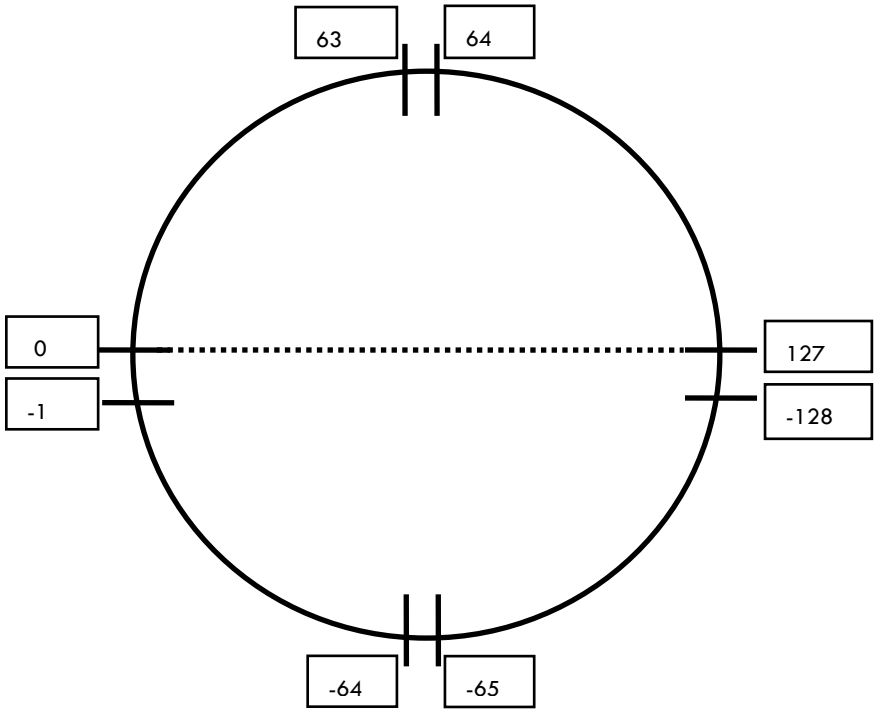
En mathématique existe la notion abstraite de +∞ et -∞. Ainsi tous les nombres peuvent se représenter selon le graphique :



Principe de la représentation des entiers en informatique :

Les machines informatiques ne permettent pas de représenter les nombres entiers de +∞ à -∞. Pour chaque type de donnée correspond une limite. Par exemple, un « byte » est sur 1 octet. La représentation est donc celle d'un cercle. 0 à 255. La plage de valeur possible va de [-128 ,127]

On note que (127+1) = -128.



Ce principe illustré sur un byte vaut aussi pour les autres types de données (short, int et long). La formule générale pour connaitre la plage de valeurs est : $-(2^{n-1})..0..(2^{n-1}-1)$

Conclusion : le choix du type de donnée lorsque l'on veut représenter un entier est important. Par exemple

- Pour représenter le nombre d'étudiants d'une classe, un byte suffit.
- Si on veut représenter tous les élèves d'un lycée, on prendra un short.
- Si on veut représenter tous les élèves d'un pays, on prendra un int.

- **Le type char** : ce type fait partie de la famille des entiers. Mais il sert avant tout à la représentation de caractères. Rien d'étonnant à cela, car un caractère possède une représentation graphique, et une valeur entière. Le caractère 'a' a pour valeur entière 97, la valeur 'A' a pour valeur décimale 65 (voir table Unicode). Sur 2 octets ce type permet la représentation de 65536 caractères.

```
char c=0x50;                    // 0x ➔ valeur Hexadécimal
c = 80;                         // Valeur décimale
c='P';                                 // Solution à privilégié
c='\u0050';                     // nombre dans l'Unicode
System.out.println("" +c);
```

Dans le programme précédent, nous voyons les 4 façons de définir un caractère. Dans les 4 cas, « c » vaut le caractère 'P'. Mais la troisième est bien entendu la plus compréhensible.

Dans certains cas on veut représenter un caractère qui ne se trouve pas sur le clavier. Par exemple comment représenter le caractère ✅. Vous pouvez chercher sur votre clavier vous ne trouverez pas cette touche. Donc dans ce cas-là la meilleure façon est d'utiliser le code Unicode. En l'occurrence il s'agit de '\u2705'.

Voici une liste de certains caractères spéciaux. Il commence par un le caractère \ (anti-slash) qui dans de nombreux langages de programmation sert d'échappement. Cela signifie que le caractère qui suit le \ sera traité de façon « spéciale ».

'\n'	saut de ligne (LF line feed)
'\t'	tabulation (HT horizontale tabulation)
'\r'	retour charriot (CR Carriage Return)
'\f'	saut de page
'\b'	espace arrière (BS Back Space)
'\\'	le caractère \
'\"	le caractère '

Exemple :

```
System.out.println("cou\ncou");      Affiche cou retour à la ligne puis cou
```

- Particularité avec le type long, float, et double : Afin d'informer que le type utilisé est **long**, **float** ou **double** vous **DEVEZ** ajouter respectivement un L, F ou D à la fin de votre nombre. Par exemple dans le cas d'un long si vous ne mettez pas la mention "L" à la fin de votre nombre le compilateur essaiera d'allouer ce dernier dans une taille d'espace mémoire de type entier et votre code ne compilera pas si votre nombre est trop grand.

Exemple :

```
byte a=256;           //256 est trop grand pour un byte : erreur de compilation
long b=12222222;      //ce nombre est un "int" affecté à un long pas de soucis
long c=12000000000;   //ce nombre est trop grand pour être un "int" donc erreur de compilation
long d=120000000000L; //avec l'ajout du "L" ce nombre est maintenant un long. Plus d'erreurs.
```

Dans le cas de nombres réels même chose. Sans précision le nombre est considéré comme un double.

```
float a=3.5;  // sans le F erreur de compilation. On cherche à affecter une valeur double à
              // (a) considéré comme float
double b=3.5D;
```

- **Le type boolean :** ce type de donnée n'autorise que 2 valeurs **true** ou **false.**

- **Le type float et double :** permets de représenter des nombres à virgules flottantes. Cette notion de virgule flottante est propre à la représentation des nombres en informatique. Pour faire simple il faut retenir que ce type de représentation entraine dans les calculs des erreurs d'arrondi.

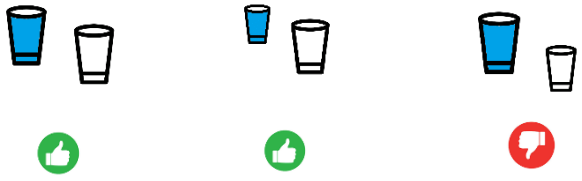
LE MÉCANISME DU CAST

Imaginons que vous cherchiez à transférer la valeur d'une variable entière dans une autre variable entière.

Plusieurs cas de figure :

- Même type de données : toujours possible
- Les types sont différents :
  - Possible si la capacité est plus grande
  - Impossible dans les autres cas

➔ Mécanisme du cast.



Prenons l'exemple suivant. On a un int (a), et un byte(b).

```
byte b=127;
int a=b;          // possible, car capacité int >= capacité de b
b=a;              // Erreur : impossible, car capacité byte < int
```

Il est possible de transformer une valeur d'un type dans un autre type grâce au mécanisme du **cast.**

```
byte b=127;
int a=1500;
b=(byte)a;      // a est convertie en byte et affectée à b.
                // entraine une perte à la conversion puisque cette valeur
                // dépasse la valeur max d'un byte.

System.out.println("" + b);  // affiche -36
```

L'ordre de capacité est le suivant : **byte < short < int < long < float < double.**

Voici un autre exemple :

```
int i = 2;
float fa = 5.56f;

i = (int) fa;          // fa est convertie en int, i vaut 5
fa = (float) i;        // i est converti en float, fa vaut 5.0
```

AUTRES PARTICULARITÉS

Comme le langage Java est en perpétuelle évolution, chaque version ajoute ses améliorations. Ainsi pour rendre plus lisible la lecture d'un nombre il est possible de séparer les chiffres avec le caractère (underscore) "_", ce qui donne par exemple :

```
int a=1_000_000;  //la lecture du code est ici simplifiée
System.out.println("a="+a);
```

Il est également possible d'affecter des valeurs hexadécimale (0x) ou binaire(0b) aux variables.

```
int a=0xFF;              // 0xFF en hexa vaut 255 en décimal
int b=0b1111_1111;       // 0b11111111 vaut 255 en décimal
System.out.println("a="+a +"et b=" + b);    //affiche 255 pour a et b.
```

LES CONSTANTES

Le mot-clé **final** permet de définir une constante. Comme pour une variable classique il faut une valeur à la différence qu'il ne pourra y avoir qu'une seule affectation.

À noter que ce mot-clé est utilisé dans d'autres contextes en Java, dans ce cas-là il a une autre signification.

```
final int a=50;    // 50 est la valeur final de (a), donc non modifiable.
final int b;
b=30;              // 30 est la valeur final de (b), donc non modifiable.
a=30;              // impossible a est final est à déjà une valeur (ici 50)
```

LES OPÉRATEURS

Il y a 3 grandes catégories d'opérateurs.

LES OPÉRATEURS ARITHMÉTIQUES

Il existe une certaine priorité des opérateurs, comme en mathématique. Pour s'éviter tous problèmes avec ces priorités il suffit de mettre des parenthèses.

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division entière)

- Cas particulier pour l'opérateur +, -, * , / , % : Pour les types entiers (byte, short, int et long), Java met en œuvre un mécanisme de conversion implicite vers le type int appelé promotion entière. Cela oblige à effectuer des cast pour les types byte et short.

```
byte a=5,b=127,c=0;
c = a + b;   //erreur : le résultat de l'opération + est converti automatiquement en int.
```

Dans cet exemple (a) et (b) sont de type « byte ». L'opération d'addition ne donne pas un **byte**, mais un **int** du fait de la conversion implicite. Cela provoque donc une erreur de compilation. Pour corriger ce problème, il faut effectuer un **cast** au niveau de l'opération. On écrira :

```
c = (byte)(a + b);   // Attention : affiche ???
```

- Cas particulier du modulo (%) : il n'accepte comme diviseur uniquement un entier.

```
int a=5;
float c,b=2;
c=a%b;       //ici b est le diviseur. b est un réel: erreur de compilation
```

- Cas particulier de la division : Il ne faut jamais faire de division par 0. À l'exécution le programme génère une Exception du type : « Exception in thread "main" java.lang.ArithmeticException: / by zero »

  Exemple :

```
int a=10,b=0;
c=a/b;    //ici b vaut 0, donc division par 0

int a=10,b=2;
c=a/b;    //ici b vaut 2, aucun problème
```

L'opérateur / sert à la fois pour diviser des entiers (byte, short, int, long) et des réels (float et double). En fonctions du type, le comportement est différent. Cela pose parfois des problèmes de résultat.
Exemple:

```
int a=5,b=2;
float c;          // c est un réel
c = a / b;        // comme a et b sont des entiers, le résultat  de la division est un entier
                  // donc c ne vaut pas 2.5, mais 2.0
```

Pour pallier à ce problème, il existe 2 solutions:

- Mettre b en float, ce qui revient à diviser un entier (a) par un réel (b), dans ce cas (c) vaut 2.5

```
int a=5;
float c,b=2;
c=a/b;  // division d'un entier par un réel. Le résultat est un réel
```

- Faire ce qu'on appelle un **CAST** ou conversion de type. On laisse (b) en entier, mais au moment de la division (b) est convertie en float:

```
int a=5,b=2;
float c;
c=a/(float)b;     // avec(float)b, b est converti en réel. C vaut bien 2.5
```

LES OPÉRATEURS LOGIQUES

Ce type d'opérateur dans les tests des conditions (if, else if) ou des boucles (for, while, do while). Le résultat d'une opération logique est une valeur booléenne (true ou flase)

Opérateur	Signification
!	Non logique
&&	Et logique
||	Ou logique
== (double égal)	Égal à
!=	Différent de
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à

Exemple :

```
int i=4,j=3 ;
boolean v ;
v=(i<=j) ;                // v vaut : false
v=(i==4 && i > 2);        // v vaut ?
```

LES OPÉRATEURS D'AFFECTATION ET D'INCRÉMENTATION

Opérateur	Signification
=	Affectation
+=	Ajout de l'opérande de droite
-=	Suppression de l'opérande de droite
*=	Multiplication de l'opérande de droite
/=	Division de l'opérande de droite
%=	Modulo de l'opérande de droite
? :	Affectation conditionnelle
++variable	Préincrémentation (avant)
variable++	Postincrémentation (après)
--variable	Prédécrémentation
variable--	Postdécrémentation

Quelques exemples simples:

```
int i=3;
i++ ;    // i=i+1 ;
i-- ;    // i=i-1 ;
i+=1 ;   //i=i+1 ;
i*=5 ;   //i=i*5 ;
```

Attention, certains opérateurs sont dits à effet de bord, et s'ils sont mal utilisé (ou mal maitrisé) peuvent donner des résultats inattendus. Exemple

```
int a=5,b=1,c=0;
c=++b + a++;    //c vaut ?

int a=5,b=1,c=0;
c=b++ + a++;    //c vaut ?
```

Conseils : Programmez au début en utilisant une façon simple d'incrémenter et d'affecter.

L'affectation conditionnelle :

Syntaxe : **<opérande> = <condition> ?<expression1> :<expression2> ;**

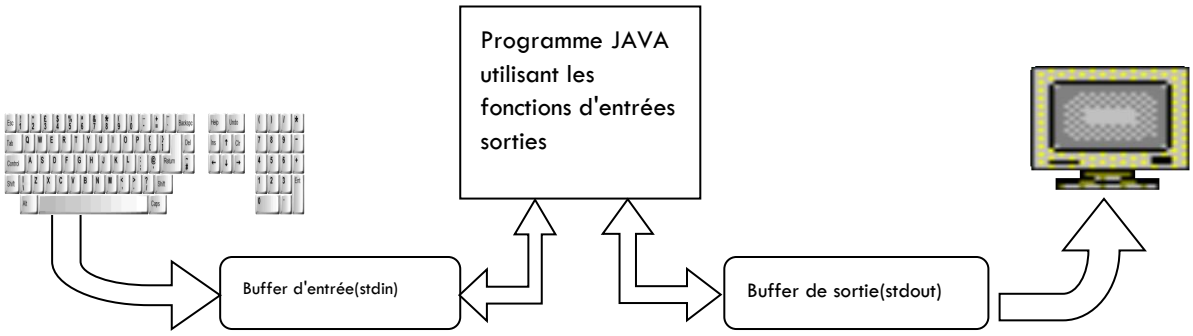
Si condition vaut true alors on affecte à opérande expression1, si false expresson2.

Exemple :

```
int i=3,j=4,k ;
k= i>j ?10 :-10 ;
```

LES ENTRÉES/SORTIES

Les fonctions d'entrée-sortie permettent de "dialoguer" avec un périphérique d'entrée (bien souvent un clavier) et un périphérique de sortie (bien souvent un écran). En réalité, ces fonctions ne dialoguent pas directement avec les périphériques, mais au travers de "buffer". Un buffer est une zone de stockage de l'information qui fait le "tampon" entre votre périphérique et le programme.



PREMIÈRE NOTION DE PACKAGE

Tout ce qui est nécessaire pour la programmation de base en java est inclus dans un package automatiquement importé.

```
import java.lang.*;    //cette ligne n'est pas obligatoire, car automatiquement ajouté
```

Un package est un ensemble de classes. En fait, c'est un ensemble de dossiers et de sous-dossiers contenant une ou plusieurs classes. La ligne `import java.lang.*` indique que dans le package java, il y a le package lang, et dans ce package on importe toutes les classes. Charger un package nous permet d'utiliser les classes qu'il contient. Le package lang contient entre autres les classes enveloppes (Integer,Double, Character, …), mais aussi d'autres classes très utile comme Math, Number, String, System.

LA CLASSE SYSTEM ➔ POUR LA SORTIE

C'est une classe qui ne peut pas être instanciée. Elle contient notamment un objet nommé "out", qui représente la sortie standard, à partir duquel on accède à de nombreuses méthodes permettant l'affichage.

Exemple : la méthode "println" sans paramètre qui affiche un saut de ligne.

```
java.lang.System.out.println();  // ln  pour new line
// on note ici l'enchainement : package.package.Class.objet.methode
// package.package n'est pas obligatoire, car importé automatiquement
```

Parmi les méthodes pour l'affichage on a : print(), printf() comme en langage C , et bien d'autres encore. Il faut savoir fouiller dans une classe. C'est un peu comme une boite à outils ! il n'y a pas que le tournevis !

LA CLASSE SCANNER ➔ POUR LA LECTURE

Une lecture de donnée se fait sur l'entrée standard en Java qui correspond à System.in

Mais pour faire cette lecture, il nous faut un objet de type Scanner. C'est une classe définie dans le package java.util

Exemple :

```
import java.util.Scanner; //ici on importe juste la classe Scanner
import java.util.*; //ici on importe toutes les classes de java.util


Scanner sc = new Scanner (System.in);   //sc est un objet de type Scanner. Il va lire sur l'entrée standard.
```

Maintenant que nous avons notre objet, il est possible d'effectuer des lectures au clavier grâce à différentes méthodes.

Par exemple :

```
System.out.println("Saisir une phrase");
String str=sc.nextLine();
System.out.println(""+str);

System.out.println("Saisir un nombre");
int a= sc.nextInt();
System.out.println("votre nombre "  +a);
```

Attention toutefois, dans le dernier exemple la méthode nextInt() peut provoquer une erreur, notamment si l'entrée au clavier ne peut être convertie en entier. Nous verrons plus tard comment gérer ce type de problème au travers du mécanisme d'exceptions.