

Cours de Programmation Orientée Objet JAVA

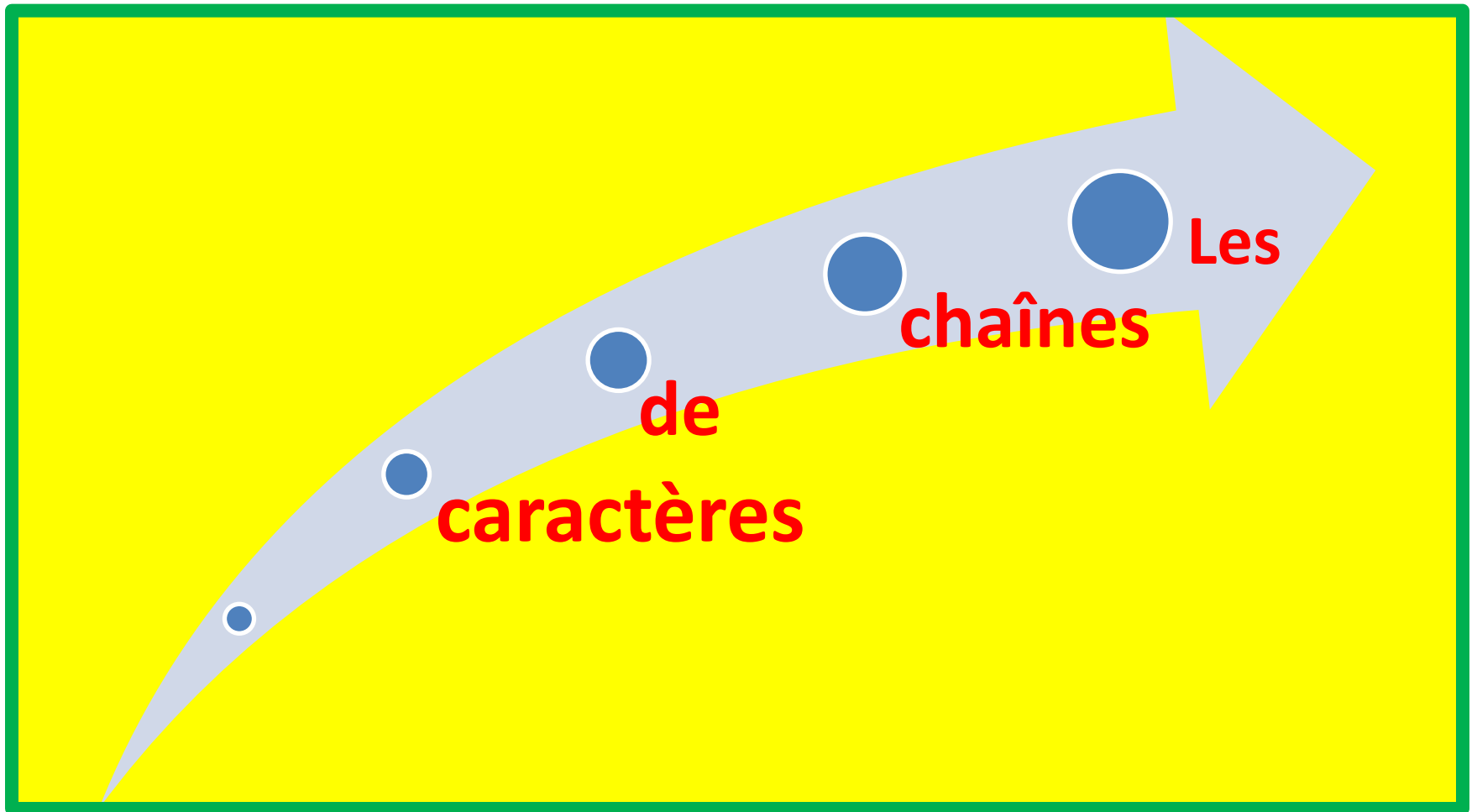
Demba SOW

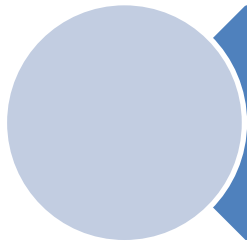
*Docteur en Codage
Cryptologie Algèbre et
Applications*

L.A.C.G.A.A.

F.S.T. / U.C.A.D.







La classe String

La classe **String** permet de manipuler « les chaînes de caractères ».

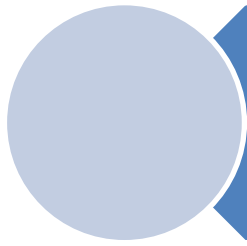
String chaine ; // déclaration d'une référence chaine
// à un objet de type String

chaine = "ça c'est un objet chaine de caractère" ; //un objet
//chaine référencé par la variable chaine

La classe String possède plusieurs constructeurs dont:

String(); // pour construire une chaine vide

String (String original); // crée un objet contenant la chaine original



Fonctionnalités

Un objet de type String n'est pas **modifiable**.

```
String chaine1= " salut ";
```

chaine1



salut

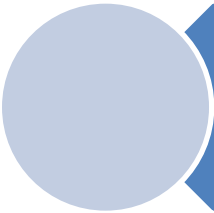
Sera détruit
s'il n'est plus
référéncé.

```
chaine1 += " le monde ";
```

salut le monde

ATTENTION

L'objet n'a pas été modifié, c'est simplement la référence qui change.



Les méthodes de la classe String (1/3)

*/*retourne la longueur de l'objet **String**.*/*

int length ()

*/*retourne un nouvel objet **String** résultant du remplacement de toutes les occurrences d'un caractère donnée par un autre caractère. */*

String replace(char oldChar, char newChar)

*/*remplace dans une chaîne de caractères, chaque sous-chaîne qui correspondent à l'expression régulière fournie, par une chaîne de caractères de remplacement*/*

String replaceAll (String origine, String remplacement)

*/*teste si l'objet **String** démarre au préfixe spécifié. */*

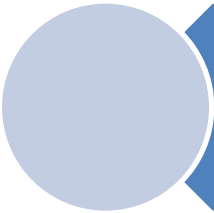
boolean startsWith (String prefix)

*/*retourne une nouvelle chaîne de caractères qui est une sous-chaîne de l'objet **String** par l'intermédiaire d'un intervalle commençant à l'index spécifié jusqu'à la fin. */*

String substring (int beginIndex)

*/*retourne une nouvelle chaîne de caractères qui est une sous-chaîne de l'objet **String** par l'intermédiaire d'un intervalle spécifié */*

String substring(int beginIndex, int endIndex)



Les méthodes de la classe String (2/3)

*/*retourne le caractère positionné à l'index spécifié. */*

char charAt (int index)

*/*compare l'objet String à un autre objet o. Retourne 0 en cas d'égalité*

*-1 en cas d'infériorité et 1 en cas de supériorité */*

int compareTo (Object o)

*/*comparaison lexicographique de deux chaînes. Retourne 0 en cas d'égalité*

*-1 en cas d'infériorité et 1 en cas de supériorité */*

int compareTo(String anotherString)

*/*compare deux chaînes lexicographiquement en ignorant la casse de caractères*/*

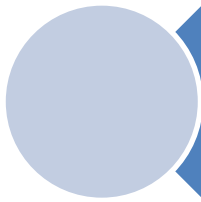
int compareToIgnoreCase (String str)

*/*concatène l'objet String à une autre chaîne de caractères. */*

String concat (String str)

*/*retourne true si et seulement si l'objet String représente la même séquence de caractères comme l'objet StringBuffer spécifié. */*

boolean contentEquals (StringBuffer buff)



Les méthodes de la classe String (3/3)

*/*teste si la fin de l'objet String correspond au suffixe spécifié. */*

boolean endsWith (String suffix)

*/*compare le contenu de deux chaînes de caractères entre elles. */*

boolean equals (Object anObject)

*/*compare l'objet String à un autre objet de même type en ignorant la casse de caractères. */*

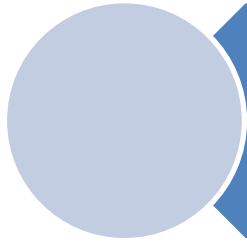
boolean equalsIgnoreCase (String anotherString)

*/*retourne l'index à l'intérieur de l'objet String de la première occurrence du caractère spécifié */*

int indexOf (int ch)

*/*retourne l'index à l'intérieur de l'objet String de la première occurrence du caractère spécifié à partir d'une certaine position */*

int indexOf(int ch, int fromIndex)



Utilisation de quelques méthodes (1/2)

```
String ch = " Le langage Java est vraiment très puissant ";
```

```
ch.length ( ); // longueur de la chaine est 43 (espaces compris)  
ch.substring ( 11);
```

"Java est vraiment très puissant ";

```
ch.substring ( 0,16); //attention: le caractère à l'indice 16 n'est pas extrait
```

"Le langage Java ";

```
ch.toUpperCase ( ); // il existe aussi toLowerCase
```

"LE LANGUAGE JAVA EST VRAIMENT TRÈS PUISSANT"

```
ch = " bonjour \n" ;
```

```
ch.trim( ) //supprime les espaces de début et de fin dans la chaine
```

"bonjour"

Utilisation de quelques méthodes (2/2)

```
String s= " java " ;  
String ch = " java " ;  
s.equals (ch); // ici renvoie true car equals est redéfinie dans String
```

true

```
s.charAt (0); // renvoie le caractère à la position 0 donc j
```

j

```
char c [ ] = s.toCharArray( ); // renvoie un tableau de caractères
```

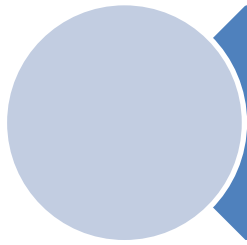
c vaut { 'j', 'a', 'v', 'a' }

```
ch.indexOf ('a'); // l'indice de la 1ère occurrence trouvée
```

renvoie la valeur 1

```
ch.indexOf ( 'a',2); // l'indice de la 1ère occurrence trouvée à partir de 2
```

renvoie la valeur 3



La méthode toUpperCase ()

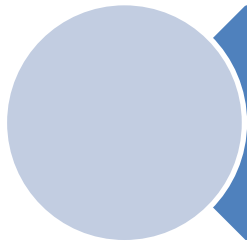
La méthode **toUpperCase** de la classe **String** permet de convertir tous les caractères d'une chaîne en majuscules.

Pour convertir certains caractères d'une chaîne en majuscules, utilisez la méthode **public static char toUpperCase (char c)** de la classe **Character**.

Par exemple:

```
String s = "bonjour ";  
for (int j = 0; j < s.length ( ) ;j++)  
    if (s.charAt ( j ) == 'o')  
    {  
        char c = Character.toUpperCase (s.charAt (j) ) ;  
        s = s.replace (s.charAt (j), c ) ;  
    }  
System.out .print (" affichage: " +s) ; // affichage : "bOnjOur "
```

*Il existe aussi la méthode **public static char toLowerCase (char c)***



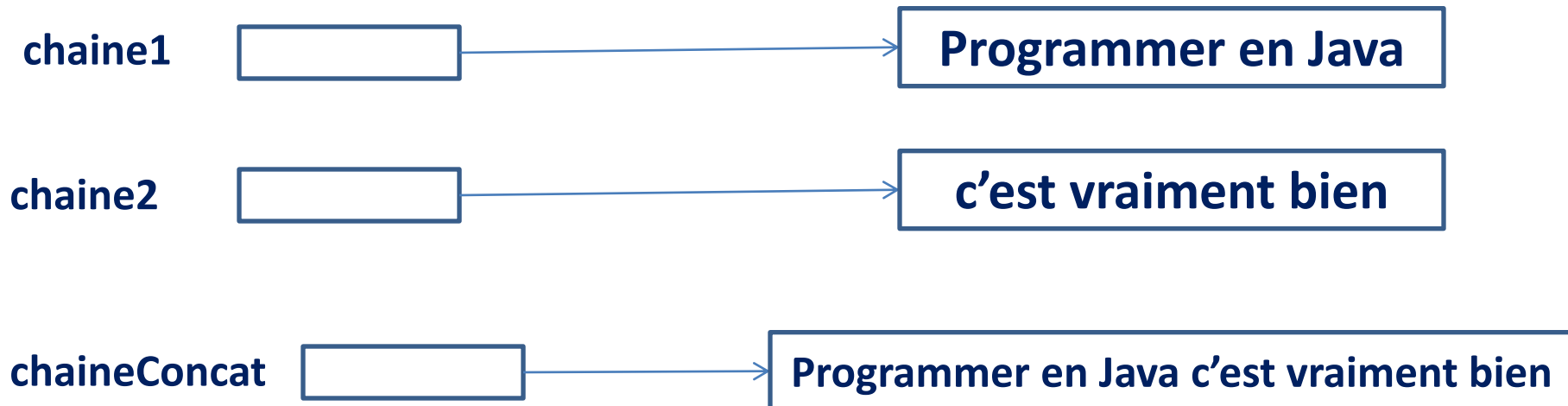
L'opérateur +

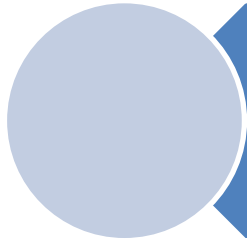
L'opérateur + permet de concaténer deux chaînes de caractères. Il est défini lorsque ses deux opérandes sont des chaînes. Il renvoie un résultat qui est la Concaténation des deux chaînes.

```
String chaine1 = "Programmer en Java" ;
```

```
String chaine2 = "c'est vraiment bien";
```

```
String chaineConcat = chaine1 + chaine2;
```





L'opérateur +

L'opérateur + est utilisé lorsque ses deux opérandes sont de type String. Mais, il est possible de mélanger des expressions de type chaîne et de type primitif. **Dans ce cas, il y a conversion (formatage) de la valeur de type primitif en chaîne.**

```
int p = 100;  
System.out.println (" la valeur de p est: " +p); // la valeur en binaire de p est  
// representee en chaine
```

En définitive, lorsque l'opérateur + possède un opérande de type String, l'autre est automatiquement converti en String.

Lorsque l'opérateur + possède deux opérandes, l'un de type String, l'autre peut être de n'importe quel type primitif, mais aussi de **type objet**. Dans ce dernier cas, il y a conversion de la valeur de l'objet en chaîne et ceci est réalisé grâce à la méthode **toString** de la classe de l'objet qu'il faut souvent redéfinir.

L'opérateur +=

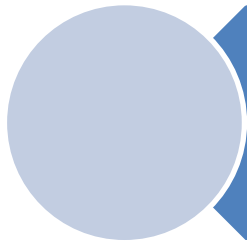
L'opérateur += défini dans le cadre des opérations arithmétiques binaires est également défini pour les chaînes. *Cet opérateur n'est pas défini si son deuxième opérande est chaîne alors que le premier ne l'est pas.*

```
String ch = "note";  
for (int i = 0; i < 5; i++)  
    ch += i;  
System.out.println (ch);
```

i = 0	ch = "note 0"
i = 1	ch = "note 01"
i = 2	ch = "note 012"
i = 3	ch = "note 0123"
i = 4	ch = "note 01234"

Seront
candidat
au
ramasse
miettes.

*Cette façon de procéder pour créer la chaîne "note 01234" est médiocre elle pourra être supplantée par l'utilisation de la classe **StringBuffer** qui permet de modifier directement la valeur d'une chaîne de caractères.*



= = et equals

```
String ch = "note";
```

```
String s = "note";
```

```
System.out.print (ch == s) ; // affiche la valeur true
```

// == teste les références des chaînes

Une chaîne n'étant pas modifiable, une seule chaîne est créée et référencée par ch et s. On parle ici d'une fusion des chaînes identiques.

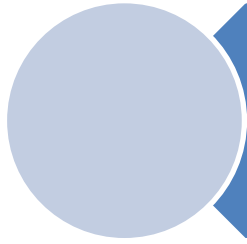
```
String ch = " bonjour ";
```

```
String s = " bon ";
```

```
s += " jour " ;
```

```
System.out.print (ch == s) ; // affiche la valeur false
```

Vu l'utilisation non optimisée de ==, pour comparer deux chaînes il faut utiliser la méthode **equals** qui compare le contenu de deux chaînes. Cette méthode est celle de la classe Object mais redéfinie dans la classe String.

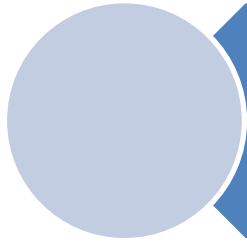


= = et equals

```
String ch = "note";  
String s = "note";  
System.out.print (ch.equals (s)) ; // affiche la valeur true  
// equivalent à ch.equals("note " );
```

```
String ch = " bonjour " ;  
String s = " bon " ;  
s += " jour " ;  
System.out.print (ch.equals( s)) ; // affiche la valeur true
```

```
String ch = "NoTe";  
String s = "note";  
System.out.print (ch.equalsIgnoreCase (s)) ; // affiche la valeur true
```



Conversions chaînes et types primitifs

La classe `String` possède une méthode statique **`valueOf`** surdéfinie avec un argument de chaque type primitif qui permet de convertir n'importe quel type primitif en chaîne de caractères.

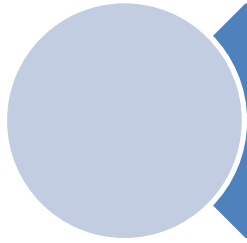
```
int n = 541;  
double d = 120.56;
```

```
String intCh = String.valueOf ( n); //intCh contient la chaîne " 541"
```

```
String doubleCh = String.valueOf ( d); //doubleCh contient la chaîne " 120.56 "
```

L'écriture `intCh = String.valueOf (n);` est équivalent à celle-ci:

```
intCh = " " +n; //on utilise une chaîne vide pour recourir à  
// l'opérateur + qui convertit n en String
```

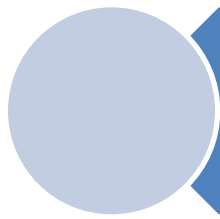
Conversions chaînes et types primitifs

Il est possible de convertir une chaîne dans un type primitif en recourant aux classes enveloppes définies pour chaque type primitif.

```
String ch = " 3521 " ;  
int n = Integer.parseInt (ch); // n contient la valeur entiere 3521
```

On dispose aussi des méthodes:

- Byte.parseByte**
- Short.parseShort**
- Integer.parseInt**
- Long.parseLong**
- Double.parseDouble**
- Float.parseFloat**



chaînes et tableaux

En Java, on peut convertir un tableau en chaîne et vice versa:

```
char [ ] mot = { 'b','o','n','j','o','u','r'};
```

/*on construit une chaine à partir d'un tableau de caractères*/

```
String ch = new String (mot); // ch = " bonjour "
```

/*constructeur avec le premier caractère et le nombre de caractères*/

```
String ch2 = new String (mot, 3,4); // ch = " jour "
```

```
String ch = " bonjour " ;
```

```
char [ ] mot = ch.toCharArray ( );
```

```
mot = { 'b','o','n','j','o','u','r'};
```

La classe StringBuffer

Les objets de la classe String ne sont pas modifiables.

La modification d'une chaîne n'est possible qu'en créant une nouvelle chaîne, ce qui n'est pas optimale lorsqu'on manipule des chaînes assez intenses.

C'est pourquoi, Java propose la classe **StringBuffer** qui permet de manipuler des chaînes tout en ayant la possibilité d'en modifier la valeur sans créer de nouvelles chaînes.

/*pour optimiser la création précédente de la chaine "note 01234 " */

String ch = "note";

StringBuffer sb = new StringBuffer (ch) ; // on transmit ici un objet String

for (int i =0; i < 5;i++)

sb.append (i); //on rajoute à la fin du StringBuffer vide

ch = sb.toString(); // on convertit le StringBuffer en String

System.out.println (ch);

La classe StringTokenizer

Cette classe n' a aucun lien direct avec la classe String, elle se trouve d'ailleurs dans le paquetage **java.util** . Elle apporte un rôle dans la manipulation des chaînes en facilitant la division de chaînes en sous-chaînes selon un nombre de « **délimiteurs** ».

Cette méthode divise une chaîne en différents éléments appelés tokens.

```
String s = " Java, est: un .langage ;interessant";
StringTokenizer st = new StringTokenizer (s, " ,;. " );
while (st.hasMoreTokens ( )) //tant qu'il y a des tokens
{ st.nextToken( ) ; // renvoie le premier élément et se positionne sur le suivant
// cette méthode renvoie un objet String.
}
```

On trouve 5 tokens : **Java** | **est** | **un** | **langage** | **interessant**

