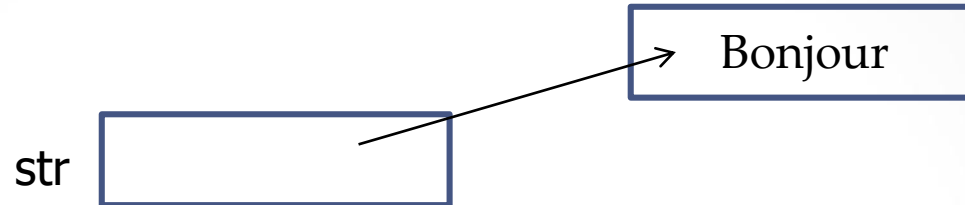


Les chaînes de caractères

Fonctionnalités de base de la classe String

Déclaration :

```
String str ;  
str = "bonjour" ;
```



La classe *String* dispose de deux constructeurs:

- l'un sans argument créant une chaîne vide,
- l'autre avec un argument de type *String* qui en crée une copie :

```
String ch1 = new String () ; // ch1 contient la référence a une chaîne vide  
String ch2 = new String("hello") ; // ch2 contient la référence a une chaîne  
// contenant la suite "hello "
```

```
String ch3 = new String(ch2) ; // ch3 contient la référence a une chaîne  
// copie de ch2, donc contenant "hello"
```

Fonctionnalités de base de la classe String

Un objet de type **String** n'est pas modifiable : Il n'existera donc aucune méthode permettant d'en modifier la valeur.

Mais il ne faut pas perdre de vue qu'on manipule en fait des références à des objets et que celles-ci peuvent voir leur valeur évoluer au fil du programme,

Fonctionnalités de base de la classe String

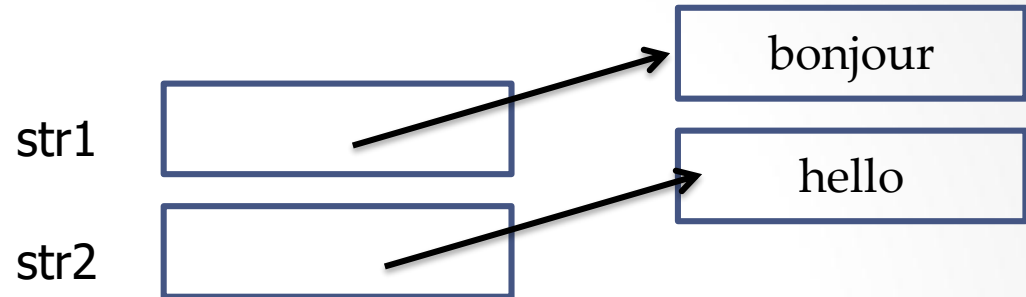
Exemple:

```
String str1, str2, str ;
```

```
str1 = "bonjour" ;
```

```
str2 = "hello" ;
```

1- Après leur exécution

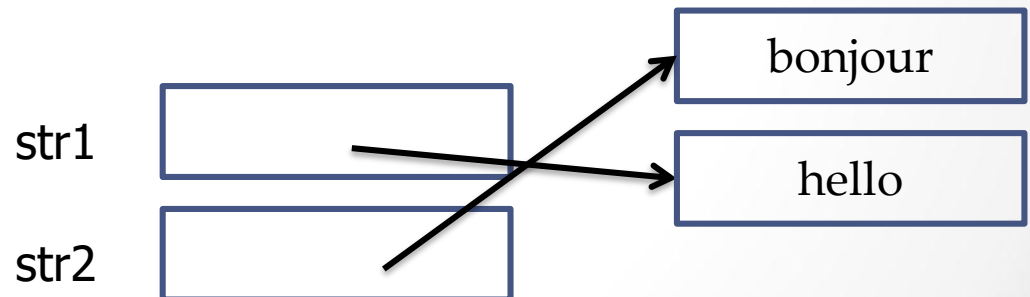


Exécutons maintenant ces instructions :

```
str = str 1 ;
```

```
str 1 = str 2 ;
```

```
str 2 = str ;
```



Les deux objets de type chaîne n'ont pas été modifiés, mais les références *str1* et *str2* l'ont été.

Entrées-sorties de chaînes

```
System.out.println ("bonjour") ;
```

Cette méthode reçoit la référence à une chaîne. Elle peut donc aussi être utilisée de cette manière :

```
String ch ;
```

```
.....
```

```
System.out.println (ch) ;
```

Longueur d'une chaîne : length

il n'existe pas de méthode standard permettant de lire une chaîne au clavier.

Classe utilitaire *Clavier* d'une méthode (statique) nommée *lireString*,

```
String ch ;
```

```
.....
```

```
ch = Clavier.lireString() ; // cree un objet de type String contenant la  
                           // reference a une chaine lue au clavier
```

Fonctionnalités de base de la classe String

Longueur d'une chaîne : **length**

La méthode **length** permet d'obtenir la longueur d'une chaîne, c'est-à-dire le nombre de caractères qu'elle contient

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

```
int n = ch.length() ;           // n contient 7  
ch    = "hello" ; n = ch.length () ; // n contient 5  
ch    = "" ; n = ch.length () ;   // n contient 0
```

Accès aux caractères d'une chaîne : **charAt**

La méthode **charAt** de la classe **String** permet d'accéder à un caractère de rang donné d'une chaîne (le premier caractère porte le rang 0):

Exemple:

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

```
ch.charAt(0)    correspond au caractère 'b' ;
```

```
ch.charAt(2)    correspond au caractère 'n'.
```

Classe String

Voici un exemple:

```
public class MotCol
{
    public static void main (String args[])
    {
        String mot ;
        System.out.print ("donnez un mot : ") ;
        mot = Clavier.lireString() ;
        System.out.println ("voici votre mot en colonne :") ;
        for (int i=0 ; i<mot.length() ; i++)
            System.out.println (mot.charAt(i)) ;
    }
}
```

Résultat:

donnez un mot : Langage
voici votre mot en colonne :

L
a
n
g
a
g
e

Classe String

Écrire un programme qui lit une chaîne au clavier et qui en affiche :

- un caractère sur trois (le premier étant affiché),
- le premier et le dernier caractère.

```
public class Calcul
{
    public static void main (String args[])
    {
        System.out.print ("donnez une chaine : ") ;
        String ch = Clavier.lireString() ;

        System.out.print ("un caractere sur trois : ") ;

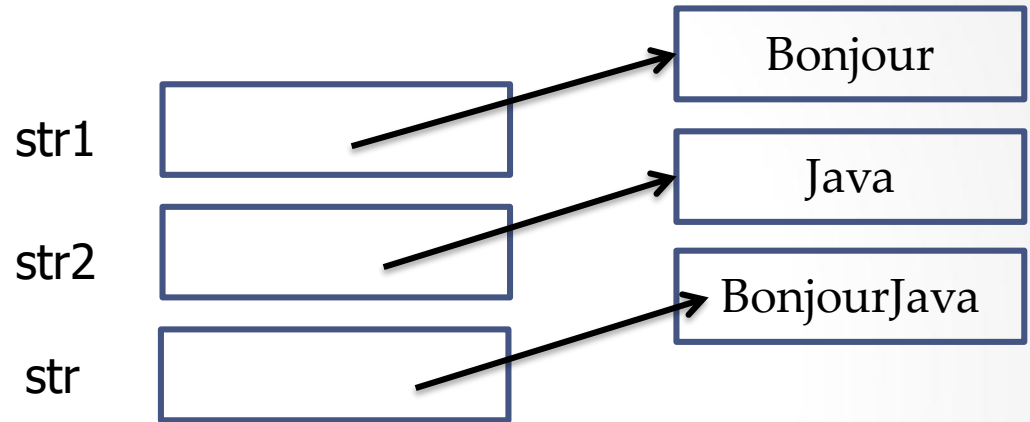
        for (int i = 0 ; i<ch.length() ; i+=3)
            System.out.print (ch.charAt(i)) ;
        System.out.println () ;

        System.out.println ("Premier caractere = " +
            ch.charAt(0)) ;
        System.out.println ("Dernier caractere = " +
            ch.charAt(ch.length()-1)) ;
    }
}
```

Classe String

Concaténation de chaînes

```
String str1 = " Bonjour" ;  
String str2 = "Java" ;  
String str = str1 + str2 ;
```



L'opérateur +=

```
String ch = "bonjour" ;  
ch += " tout le monde" ;
```



// ch désigne la chaîne "bonjour tout le monde"

```
String ch = "chiffres = " ;  
for (int i = 0 ; i<=9 ; i++)  
ch += i ;  
System.out.println (ch) ;
```



// affiche : chiffres = 0123456789

Classe String

Exemple:

Quels résultats fournit le programme suivant ?

```
public class Chaine
{ public static void main (String args[])
{ String ch1 = new String();
System.out.println ("A - ch1 =:" + ch1 + ":") ;
String ch2 = "hello" ;
System.out.println ("B - ch2 =:" + ch2 + ":") ;
String ch3 = new String ("bonjour") ;
System.out.println ("C - ch3 =:" + ch3 + ":") ;

String ch4 = new String (ch3) ;
System.out.println ("D - ch4 =:" + ch4 + ":") ;
ch3 = "bonsoir" ;
System.out.println ("E - ch4 =:" + ch4 + ": ch3 =:" + ch3 + ":") ;
ch4 = ch3 ;
ch3 = "au revoir" ;
System.out.println ("F - ch4 =:" + ch4 + ": ch3 =:" + ch3 + ":") ;
}
}
```

Classe String

String ch1 = new String();

crée une chaîne vide et place sa référence dans *ch1*.

String ch2 = "hello" ;

crée une chaîne formée des cinq caractères *h*, *e*, *l*, *l* et *o* et place sa référence dans *ch2*.

String ch3 = new String ("bonjour") ;

crée une chaîne contenant les sept caractères *b*, *o*, *n*, *j*, *o*, *u* et *r* et place sa référence dans *ch3*.

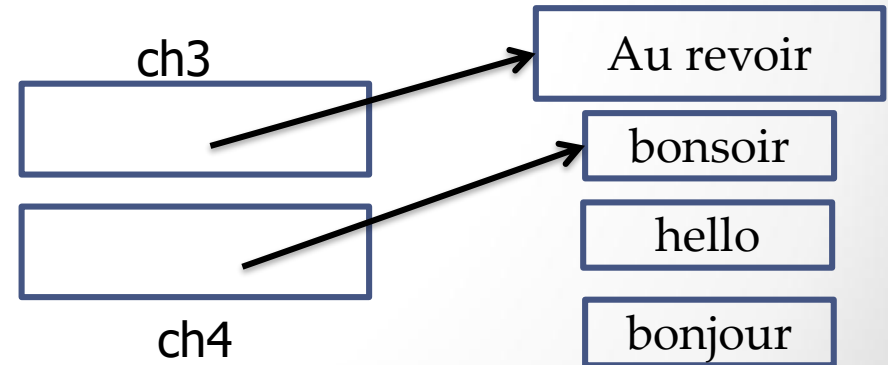
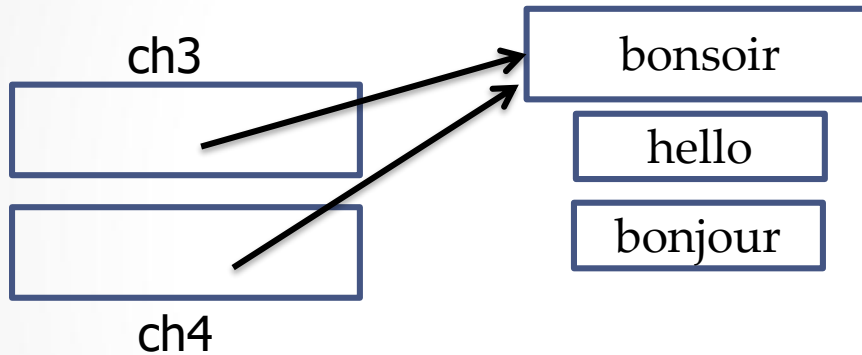
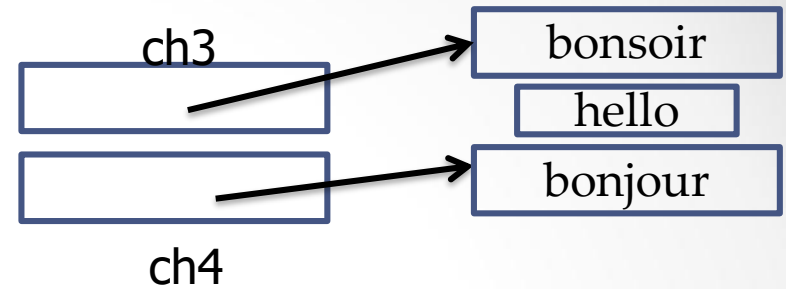
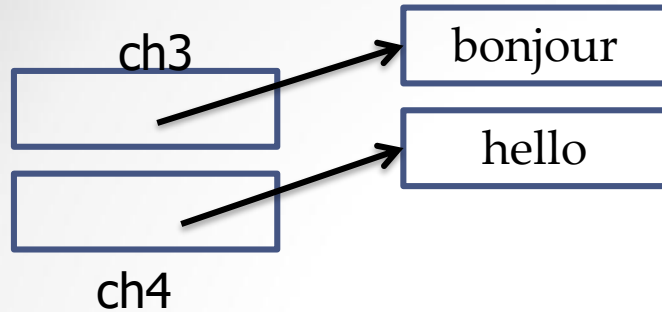
String ch4 = new String (ch3) ;

crée une chaîne par copie de la valeur de la chaîne de référence *ch3* et place sa référence dans *ch4*.

Affectation

ch4 = ch3 ;

Classe String



A - `ch1 =::`
B - `ch2 =:hello:`
C - `ch3 =:bonjour:`
D - `ch4 =:bonjour:`
E - `ch4 =:bonjour: ch3 =:bonsoir:`
F - `ch4 =:bonsoir: ch3 =:au revoir:`

Classe String

Écriture des constantes chaînes

l'utilisation la plus courante est celle du caractère de fin de ligne au sein d'une constante chaîne

```
String ch = "bonjour\nmonsieur" ;  
System.out.println (ch) ;
```

Elles affichent les deux lignes suivantes :

```
bonjour  
Monsieur
```

ch.length() ----> 16

Classe String

Recherche dans une chaîne

La méthode *indexOf* surdéfinie dans la *classe String* permet de rechercher, à partir du début d'une chaîne ou d'une position donnée :

Voyez ces instructions :

```
String mot = "anticonstitutionnellement" ;  
int n ;  
  
n = mot.indexOf ('t') ;           // n vaut 2  
n = mot.lastIndexOf ('t') ;      // n vaut 24  
n = mot.indexOf ("ti") ;         // n vaut 2  
n = mot.lastIndexOf ("ti") ;     // n vaut 12  
n = mot.indexOf ('x') ;          // n vaut -1
```

Classe String

Voici un exemple de programme ou la méthode *indexOf* pour compter le nombre de caractères *e* présents dans un mot entré au clavier :

```
public class Compte1
{ public static void main (String args[])
{ final char car = 'e' ;
int i, posCar ;
int nbCar = 0 ;
String ch ;
System.out.print ("donnez un mot : ") ;
ch = Clavier.lireString() ;
i = 0 ;
do
{ posCar = ch.indexOf(car, i) ; // recherche a partir du caractère de rang i
if (posCar>=0) { nbCar++ ;
i = posCar+1 ;
}
}
while (posCar>=0) ;
System.out.println ("votre mot comporte " + nbCar
+ " fois le caractere " + car) ;
}}
```

donnez un mot : exceptionnelle
votre mot comporte 4 fois le
caractere e

Classe String

Une deuxième version plus simple:

```
public class Compte2
{
    public static void main (String args[])
    {
        final char car = 'e' ;
        String ch ;
        System.out.print ("donnez un mot : ") ;
        ch = Clavier.lireString() ;

        int nbCar = 0 ;
        for (int i=0 ; i<ch.length() ; i++)
            if (ch.charAt(i) == car) nbCar++ ;

        System.out.println ("votre mot comporte " + nbCar
            + " fois le caractere " + car) ;
    }
}
```

Classe String

Comparaisons de chaînes

la classe *String* dispose d'une méthode ***equals*** qui compare le contenu de deux chaînes :

```
String ch1 = "hello" ;  
String ch2 = "bonjour" ;  
.....  
ch1.equals(ch2)           // cette expression est fausse  
ch1.equals("hello")       // cette expression est vraie
```

La méthode ***equalsIgnoreCase*** effectue la même comparaison, mais sans distinguer les majuscules des minuscules :

```
String ch1 = "HeLlo" ;  
String ch2 = "hello" ;  
.....  
ch1.equalsIgnoreCase(ch2) // cette expression est vraie  
ch1.equalsIgnoreCase("hello") // cette expression est vraie
```

Classe String

La méthode **compareTo** : On peut effectuer des *comparaisons lexicographiques* de chaînes pour savoir laquelle de deux chaînes apparaît avant une autre, en se fondant sur l'ordre des caractères.

chaîne1	chaîne2	chaîne1.compareTo(chaîne2)
"bonjour"	"monsieur"	négatif
"bonj"	"bonjour"	négatif
"paris2"	"paris10"	positif (car '2' > '1')
"Element"	"element"	négatif (car 'E' < 'e')
"Element"	"élément"	négatif (car 'E' < 'é')
"element"	"élément"	égatif (car 'e' < 'é')

Classe String

Modification de chaînes

- Remplacement de caractères:

La méthode ***replace*** crée une chaîne en remplaçant toutes les occurrences d'un caractère donné par un autre

```
String ch = "bonjour" ;  
String ch1 = ch.replace('o', 'a') ;      // ch n'est pas modifiée  
                                           // ch1 contient "banjour"
```

Classe String

Modification de chaînes

Extraction de sous-chaîne

La méthode ***substring*** permet de créer une nouvelle chaîne en extrayant de la chaîne courante :

- soit tous les caractères depuis une position donnée

```
String ch = "anticonstitutionnellement" ;  
String ch1 = ch.substring (5) ;           // ch n'est pas modifiée  
                                           // ch1 contient "onstitutionnellement "
```

Soit tous les caractères compris entre deux positions données
(la première incluse, la seconde exclue) :

```
String ch = "anticonstitutionnellement" ;  
String ch1 = ch.substring (4, 16) ;      // ch n'est pas modifiée  
                                           // ch1 contient "constitution"
```

Classe String

Passage en majuscules ou en minuscules

La méthode ***toLowerCase*** crée une nouvelle chaîne en remplaçant toutes les majuscules par leur équivalent en minuscules (lorsque celui-ci existe. La méthode ***toUpperCase*** crée une nouvelle chaîne en remplaçant toutes les minuscules par leur équivalent en majuscules.

```
String ch = "LanGaGE_3" ;  
String ch1 = ch.toLowerCase() ;    // ch est inchangée  
                                   // ch1 contient "language_3"  
.....  
String ch2 = ch.toUpperCase() ;    // ch n'est pas modifiée  
                                   // ch2 contient "LANGUAGE_3"
```

Classe String

Suppression des séparateurs de début et de fin

La méthode *trim* crée une nouvelle chaîne en supprimant les éventuels séparateurs de début et de fin (espace, tabulations, fin de ligne) :

```
String ch = " \ndes separateurs avant, pendant\t\n et apres \n    " ;
```

```
String ch1 = ch.trim() ;           // ch n'est pas modifiée, ch1 contient la chaîne :
```

```
"des separateurs avant, pendant\t\n et apres"
```

Classe String

Tableaux de chaînes

un exemple de programme qui effectue un tri lexicographique de chaînes :

```
public class TriCh
{ public static void main (String args[])
{ String tabCh [] = {"java", "c", "pascal", "c++", "ada", "basic", "fortran" } ;

String temp ;    // pour l'Echange de deux references
int i, j ;
int nbCh = tabCh.length ;
for (i=0 ; i<nbCh-1 ; i++)
for (j=i ; j<nbCh ; j++)
if ( (tabCh[i].compareTo(tabCh[j])) > 0)
{ temp = tabCh [i] ;
  tabCh [i] = tabCh [j] ;
  tabCh [j] = temp ;
}
System.out.println ("chaines trie'es : ");
for (i=0 ; i<nbCh ; i++) System.out.println (tabCh[i]) ;
}
}
```

chaines trie'es :

ada
basic
c
c++
fortran
java
pascal

Classe String

Exercice à faire:

Écrire un programme qui lit un mot au clavier et qui indique combien de fois sont présentes chacune des voyelles *a, e, i, o, u* ou *y*, que celles-ci soient écrites en majuscules ou en minuscules.

Par exemple:

Donnez un mot : **Anticonstitutionnellement**

il comporte

1 fois la lettre a

3 fois la lettre e

3 fois la lettre i

2 fois la lettre o

1 fois la lettre u

0 fois la lettre y

Classe String

```
public class TriVoyelles
{ public static void main (String args[])
{
char voy[] = {'a', 'e', 'i', 'o', 'u', 'y'} ;
int nVoy [] = new int [voy.length] ;
for (int i=0 ; i<nVoy.length ; i++) nVoy[i]
= 0 ;
System.out.print ("donnez un mot : ") ;
String mot = Clavier.lireString() ;
mot = mot.toLowerCase() ;
for (int i=0 ; i<mot.length() ; i++)
for (int j=0 ; j<voy.length ; j++)
if (mot.charAt(i) == voy[j]) nVoy[j]++ ;
System.out.println ("il comporte : ") ;
for (int i=0 ; i<voy.length ; i++)
System.out.println(nVoy[i] + " fois la lettre
" + voy[i]) ;
}
}
```

Donnez un mot :

Anticonstitutionnellement

il comporte

1 fois la lettre a
3 fois la lettre e
3 fois la lettre i
2 fois la lettre o
1 fois la lettre u
0 fois la lettre y

Classe String

Conversions entre chaînes et types primitifs

Voici un programme qui lit des entiers au clavier et les convertit en chaîne (on s'interrompt lorsque l'utilisateur fournit un entier nul) :

```
public class ConvICH
{
    public static void main (String args[])
    { int n ;
      while (true)                // on s'arrêtera quand n == 0
      { System.out.print ("donnez un entier (0 pour finir) : ") ;
        n = Clavier.lireInt() ;
        if (n==0) break ;
        String ch = String.valueOf(n) ;
        System.out.println (" chaine correspondante,
de longueur "+ ch.length() + " : " + ch) ;
      }
    }
}
```

```
donnez un entier (0 pour finir) : 427
chaine correspondante, de longueur 3 : 427
donnez un entier (0 pour finir) : -4351
chaine correspondante, de longueur 5 : -4351
donnez un entier (0 pour finir) : 123456789
chaine correspondante, de longueur 9 :
123456789
donnez un entier (0 pour finir) : 0
```

conversions d'un int en String

conversions d'un double en String

```
public class ConvDCh
{ public static void main (String args[])
{ double x ;
while (true) // on s'arretera quand n == 0
{ System.out.print ("donnez un double (0 pour finir) : ") ;
x = Clavier.lireDouble() ;
if (x==0.) break ;
String ch = String.valueOf(x) ;
System.out.println (" chaine correspondante, de longueur "
+ ch.length() + " : " + ch) ;
}}}
```

```
donnez un double (0 pour finir) : 51
chaine correspondante, de longueur 4 : 51.0
donnez un double (0 pour finir) : -23
chaine correspondante, de longueur 5 : -23.0
donnez un double (0 pour finir) : 12.345e+2
chaine correspondante, de longueur 6 : 1234.5
donnez un double (0 pour finir) : 12.345e30
chaine correspondante, de longueur 9 : 1.2345E31
donnez un double (0 pour finir) : 12.345e-30
chaine correspondante, de longueur 10 : 1.2345E-29
donnez un double (0 pour finir) : 0
```

Les conversions d'une chaîne en un type primitif

On peut réaliser les conversions inverses des précédentes. Il faut alors recourir à une méthode de la classe enveloppe associée au type primitif : ***Integer*** pour ***int*** ou ***Float*** pour ***float***

on utilisera la méthode statique ***parseInt*** de la classe enveloppe *Integer*, comme ceci :

```
String ch = "3587" ;  
int n = Integer.parseInt(ch) ;
```

D'une manière générale, on dispose des méthodes suivantes :

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

Classe String

Un exemple de programme qui lit des chaînes au clavier et les convertit en entiers

```
public class ConvChI
{ public static void main (String args[])
{ String ch ;
while (true) // on s'arrêtera quand chaîne vide
{ System.out.print ("donnez une chaîne (vide pour finir) : ") ;
ch = Clavier.lireString() ;
if (ch.length()==0) break ;
int n = Integer.parseInt(ch) ;
System.out.println (" entier correspondant " + n) ;
}}}
```

conversion d'une chaîne en int

```
donnez une chaîne (vide pour finir) : 1234
entier correspondant 1234
donnez une chaîne (vide pour finir) : -789
entier correspondant -789
donnez une chaîne (vide pour finir) : 0123
entier correspondant 123
donnez une chaîne (vide pour finir) :
00123456789
entier correspondant 123456789
donnez une chaîne (vide pour finir) :
```

Classe String

Conversions entre chaînes et tableaux de caractères

On peut construire une chaîne à partir d'un tableau de caractères :

```
char mot [] = {'b', 'o', 'n', 'j', 'o', 'u', 'r'} ; // tableau de 7 caracteres
```

```
String ch = new String (mot) ; // ch est construite a partir du tableau mot  
// et contient maintenant la chaine "bonjour"
```

Il existe même un constructeur permettant de n'utiliser qu'une partie des caractères d'un tableau

```
char mot [] = {'b', 'o', 'n', 'j', 'o', 'u', 'r'} ; // tableau de 7 caracteres  
String ch = new String (mot, 2, 4) ; // ch est construite en prelevant 4 carac  
// du tableau mot, a partir de celui de rang 2 ;  
ch contient la chaine "onjo"
```

Classe String

De façon symétrique, on peut transformer un objet chaîne en un tableau de caractères, grâce à la méthode ***toCharArray*** :

```
String ch = "bonjour" ;  
char mot [] ;  
mot = ch.toCharArray() ; // mot référence maintenant un tableau de 7 éléments  
                          // (les 7 caracteres de bonjour)
```

La classe **StringBuffer**

Les objets de type *String* ne sont pas modifiables mais nous avons vu qu'il était possible de les employer pour effectuer la plupart des manipulations de chaînes:



En créant une nouvelle chaîne



- La perte de temps qui en résulte peut devenir gênante



Java dispose d'une classe ***StringBuffer*** destinée elle aussi à la manipulation de chaînes, mais dans laquelle les objets sont modifiables.

La classe ***StringBuffer*** dispose de fonctionnalités classiques mais elle n'a aucun lien d'héritage avec *String* et ses méthodes ne portent pas toujours le même nom,

La classe `StringBuffer`

On peut créer un objet de type *StringBuffer* à partir d'un objet de type *String*. Il existe des méthodes :

- de modification d'un caractère de rang donné : *setCharAt*,
- d'accès à un caractère de rang donné : *charAt*,
- d'ajout d'une chaîne en fin : la méthode *append* accepte des arguments de tout type primitif et de type *String*,
- d'insertion d'une chaîne en un emplacement donné : *insert*,
- de remplacement d'une partie par une chaîne donnée : *replace*,
- de conversion de *StringBuffer* en *String* : *toString*.

La classe StringBuffer

Exemple d'utilisation de la classe StringBuffer

Voici un programme utilisant ces différentes possibilités :

```
class TestclassStringBuffer
{ public static void main (String args[])
{ String ch = "la java" ;
StringBuffer chBuf = new StringBuffer (ch) ;
System.out.println (chBuf) ;
chBuf.setCharAt (3, 'J'); System.out.println (chBuf) ;
chBuf.setCharAt (1, 'e') ; System.out.println (chBuf) ;
chBuf.append (" 2") ; System.out.println (chBuf) ;
chBuf.insert (3, "langage ") ; System.out.println (chBuf) ;
}
}
```

```
la java
la Java
le Java
le Java 2
le langage Java 2
```

Arrays et ArrayList

La classe Arrays

Java fournit une classe Arrays, qui contient de nombreuses méthodes utiles pour manipuler des tableaux.

Méthode	Description
<code>static int binarySearch(type[] a, type key)</code>	Rechercher dans le tableau spécifié la valeur de key spécifiée à l'aide de l'algorithme de recherche binaire.
<code>static boolean equals(type[] a, type[] a2)</code>	Renvoyer true si les deux tableaux du même type spécifiés sont égaux.
<code>static void fill(type[] a, type val)</code>	Affecter la valeur spécifiée à chaque élément du tableau spécifié
<code>static void sort(type[] a)</code>	Trier le tableau spécifié en ordre croissant
<code>static void sort(type[] a, int fromIndex, int toIndex)</code>	Trier la plage spécifiée du tableau en ordre croissant
<code>static void parallelSort(type[] a)</code>	Trier le tableau spécifié en ordre croissant à l'aide d'un tri en parallèle.
<code>static void parallelSort(type[] a, int fromIndex, int toIndex)</code>	Trier la plage spécifiée du tableau en ordre croissant en utilisant le tri en parallèle
<code>static List asList(type ... a)</code>	Renvoyer une liste de taille fixe contient les éléments des tableaux spécifiés.
<code>static type[] copyOf(type[] originalArray, int newLength)</code>	Copier le tableau spécifié, en tronquant ou en remplissant avec la valeur par défaut (si nécessaire) afin que la copie ait la longueur spécifiée.
<code>static type[] copyOfRange(type[] originalArray, int fromIndex, int endIndex)</code>	Copier la plage spécifiée du tableau spécifié dans un nouveau tableau.
<code>static String toString(type[] originalArray)</code>	renvoyer une représentation sous forme de chaîne du contenu du tableau spécifié.

La classe Arrays

- Il existe une version de la méthode **sort()** pour trier les tableaux **int, double, char, byte, float, long, short et Object**.
- Les méthodes de la classe Arrays sont des méthodes statiques, ce qui signifie que vous les utilisez avec le nom de la classe sans instancier un objet Arrays.
- La classe Arrays est située dans le package java.util. Vous pouvez donc utiliser l'instruction `import java.util. *;` pour y accéder.

La classe Arrays

Exemple 1: binarySearch

```
1  import java.util.Arrays;
2
3  public class Test {
4      public static void main(String args[]) {
5          int[] T1 = { 2, 4, 5, 9, 5, 9 };
6          int[] T2 = { 25, 30, 45, 50, 52, 60 };
7
8          System.out.println(Arrays.binarySearch(T2, 45));
9      }
10 }
```

2

Exemple 2: equals

```
1  import java.util.Arrays;
2
3  public class Test {
4      public static void main(String args[]) {
5          int[] T1 = { 2, 4, 5, 9, 5, 9 };
6          int[] T2 = { 25, 30, 45, 50, 52, 60 };
7
8          System.out.println(Arrays.equals(T1, T2));
9      }
10 }
```

false

La classe Arrays

Exemple 3 : toString

```
1  import java.util.*;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T1 = { 2, 4, 5, 9, 5, 9 };
7
8          System.out.println(Arrays.toString(T1));
9      }
10 }
```

[2, 4, 5, 9, 5, 9]

Exemple 4 : fill

```
1  import java.util.Arrays;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T3 = new int[4];
7          Arrays.fill(T3, 2);
8          System.out.println(Arrays.toString(T3));
9      }
10 }
```

[2, 2, 2, 2]

La classe Arrays

Exemple 5 : sort

```
1  import java.util.Arrays;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T1 = { 2, 4, 5, 9, 5, 9 };
7          Arrays.sort(T1);
8          System.out.println(Arrays.toString(T1));
9      }
10 }
```

[2, 4, 5, 5, 9, 9]

Exemple 6 : parallelSort

```
1  import java.util.Arrays;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T1 = { 2, 4, 5, 9, 5, 9 };
7          Arrays.parallelSort(T1);
8          System.out.println(Arrays.toString(T1));
9      }
10 }
```

[2, 4, 5, 5, 9, 9]

La classe Arrays

Exemple 5 : sort

```
1  import java.util.Arrays;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T1 = { 2, 4, 5, 9, 5, 9 };
7          Arrays.sort(T1);
8          System.out.println(Arrays.toString(T1));
9      }
10 }
```

[2, 4, 5, 5, 9, 9]

Exemple 6 : parallelSort

```
1  import java.util.Arrays;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          int[] T1 = { 2, 4, 5, 9, 5, 9 };
7          Arrays.parallelSort(T1);
8          System.out.println(Arrays.toString(T1));
9      }
10 }
```

[2, 4, 5, 5, 9, 9]

La classe ArrayList

La classe ArrayList

Java fournit une classe **ArrayList** qui peut être utilisée pour créer des conteneurs qui stockent des listes d'objets.

La classe ArrayList offre certains avantages par rapport à la classe Arrays.

- une **ArrayList** est redimensionnable dynamiquement, ce qui signifie que sa taille peut changer pendant l'exécution du programme.

Cela signifie que :

- Vous pouvez ajouter un élément à tout moment dans un conteneur **ArrayList**.
- La taille du tableau se développe automatiquement pour s'adapter au nouvel élément.
- Vous pouvez supprimer un élément à tout moment dans un conteneur **ArrayList** et la taille du tableau se contracte automatiquement.

Pour utiliser la classe **ArrayList**, vous devez utiliser l'une des instructions d'importation suivantes:

```
import java.util.ArrayList;
```

```
import java.util.*;
```

La classe ArrayList

pour déclarer une **ArrayList**, vous pouvez utiliser le constructeur par défaut, comme dans l'exemple suivant qui déclare une liste de String

```
ArrayList< String> names = new ArrayList< String>();
```

Une **ArrayList** peut contenir n'importe quel type d'objet ; L'ajout d'un type de données entre crochets fait que Java vérifie que vous affectez les types appropriés à une liste.

Le constructeur par défaut crée une **ArrayList** avec une capacité de 10 éléments.

Vous pouvez également spécifier une capacité si vous le souhaitez. L'exemple suivant déclare une **ArrayList** qui peut contenir 20 noms.

```
ArrayList< String> noms = new ArrayList< String>(20);
```

La classe ArrayList

Le tableau suivant résume quelques méthodes utiles ArrayList.

Méthode	Description
<code>public void add(Object)</code>	Ajouter un élément à une ArrayList; la version par défaut ajoute un élément au prochain emplacement disponible; une version surchargée vous permet de spécifier une position à laquelle nous voulons ajouter l'élément
<code>public void add(int, Object)</code>	
<code>public void remove(int)</code>	Supprimer un élément d'une ArrayList à un emplacement spécifié
<code>public void set(int, Object)</code>	Modifier un élément à un emplacement spécifié dans une ArrayList
<code>Object get(int)</code>	Récupérer un élément d'un emplacement spécifié dans une ArrayList
<code>public int size()</code>	Renvoyer la taille actuelle de ArrayList

La classe ArrayList

Exemple 1 : add

```
1  import java.util.ArrayList;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          ArrayList< Integer> liste = new ArrayList< Integer>();
7          liste.add(4);
8          liste.add(5);
9          liste.add(2);
10         liste.add(0, 1);
11         System.out.println(liste);
12     }
13 }
```

[1, 4, 5, 2]

La classe ArrayList

Exemple 2 : remove

```
1  import java.util.ArrayList;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          ArrayList< Integer> liste = new ArrayList< Integer>();
7          liste.add(4);
8          liste.add(5);
9          liste.add(2);
10         System.out.println(liste);
11         liste.remove(1);
12         System.out.println(liste);
13     }
14 }
```

[4, 5, 2]

[[4, 2]

La classe ArrayList

Exemple 3 : set

```
1  import java.util.ArrayList;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          ArrayList< Integer> liste = new ArrayList< Integer>();
7          liste.add(4);
8          liste.add(5);
9          liste.add(2);
10         System.out.println(liste);
11         liste.set(1, 15);
12         System.out.println(liste);
13     }
14 }
```

[4, 5, 2]

[4, 15, 2]

La classe ArrayList

Exemple 4 : get

```
1  import java.util.ArrayList;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          ArrayList< Integer> liste = new ArrayList< Integer>();
7          liste.add(4);
8          liste.add(5);
9          liste.add(2);
10         System.out.println("liste[1] : " + liste.get(1));
11     }
12 }
```

La classe ArrayList

Exemple 5 : size

```
1  import java.util.ArrayList;
2
3  public class Test {
4
5      public static void main(String args[]) {
6          ArrayList< Integer> liste = new ArrayList< Integer>();
7          liste.add(4);
8          liste.add(5);
9          liste.add(2);
10         System.out.println("la taille est : " + liste.size());
11     }
12 }
```

la taille est : 3

Les types énumérés

Les types énumérés

Un type énuméré permet de définir un ensemble d'identificateurs pour des constantes ; il peut permettre par exemple de définir l'ensemble : LUNDI, MARDI, ... pour les jours de la semaine.

On peut définir des types énumérés à l'aide de classes. Ces classes diffèrent des classes traditionnelles ; en particulier, leur en-tête commence par le mot **enum** à la place du mot **class**.

Exemple 1

La **classe Jour** consiste uniquement en une énumération d'identificateurs pour les jours de la semaine.

Cette énumération doit nécessairement être en première ligne de la définition de la classe. Une méthode main utilise la classe Jour ; on y utilise une instruction par cas selon les jours de la semaine.

Les types énumérés

```
public enum Jour {  
LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE;  
}
```

```
class EssaiJour {  
    public static void main(String[] arg) {  
        Jour jour = Jour.SAMEDI;  
        if (jour == Jour.SAMEDI) System.out.print("fin de semaine : ");  
        switch(jour) {  
            case SAMEDI :  
            case DIMANCHE :  
                System.out.println("se reposer");  
                break;  
            default :  
                System.out.println("travailler");  
                break;  
        }  
    }  
}
```

fin de semaine : faire les courses

Les types énumérés

```
public enum Jour {  
    LUNDI, MARDI, ..., DIMANCHE ;  
}
```

Dans sa forme la plus simple un type énuméré définit un ensemble de valeurs

```
public enum Mois {  
    JANVIER, FEVRIER, ..., DECEMBRE ;  
}
```

```
public class RendezVous {  
    private Jour j;  
    private Mois m;  
    ...  
  
    public RendezVous(Jour j, Mois m, ...) {  
        this.j = j;  
        this.m = m;  
        ...  
    }  
    ...  
    String toString() {  
        return "Rendez-vous : " + j + ... ;  
    }  
}
```

```
...  
RendezVous rv = new RendezVous(  
    Jour.LUNDI, Mois.DECEMBRE, ...);  
...  
System.out.println(rv);  
...
```

```
...  
Rendez-vous : LUNDI ...  
...
```

Les types énumérés étant définis comme des classes il est possible de définir des types beaucoup plus riches

Les types énumérés

```
public enum Semaine_jours {  
    LUNDI(8), MARDI(8), MERCREDI(5), JEUDI(8), VENDREDI(8), SAMEDI(0), DIMANCHE(0);  
  
    private int nbHeures;  
  
    public Semaine_jours(int nb) {  
        nbHeures = nb;  
    }  
  
    public int getNbHeures() {  
        return nbHeures;  
    }  
  
    public String action() {  
        switch(this) {  
            case SAMEDI : return « repos";  
            case DIMANCHE : return « Jardin ";  
            default : return « Bosser";  
        }  
    }  
}
```

numero de SAMEDI : 5
SAMEDI : repos
nombre d'heures de LUNDI : 8
nombre d'heures de MARDI : 8
nombre d'heures de MERCREDI : 5
nombre d'heures de JEUDI : 8
nombre d'heures de VENDREDI : 8
nombre d'heures de SAMEDI : 0
nombre d'heures de DIMANCHE : 0

```
class principal {  
    public static void main(String[] arg) {  
        Semaine_jours unJour = Semaine_jours.SAMEDI;  
        System.out.println("numero de " + unJour + " : " + unJour.ordinal()); // la méthode ordinal est une méthode définie  
                                                //dans la classe java.lang.Enum et qui renvoie le numéro  
                                                // d'ordre dans l'énumération, en commençant à la valeur 0.  
  
        System.out.println(unJour + " : " + unJour.action());  
        for (Semaine_jours jour : Semaine_jours.values())  
            System.out.println("nombre d'heures de " + jour + " : " + jour.getNbHeures());  
    }  
}
```