Chapitre 7

Les chaînes de caractères en JAVA

Description et utilisation des classes **String**, **StringTokenizer** et **StringBuffer** de JAVA.

1. Presentation

Une chaîne de caractères en Java est un objet de la classe **java.lang.String**.

Les objets de type **String** sont des objets fixes c'est à dire qu'il n'est pas possible de changer le contenu d'une chaîne de caractères: Fini les débordements sur les chaînes de caractères!!

Cela n'est pas une difficulté, comme on va le voir, car Java peut à tout moment allouer une nouvelle chaîne de caractères qui est le résultat d'un traitement sur les chaînes.

Pour des raisons de performance, quand il est nécessaire de gérer de gros volumes de chaîne de caractère (éditeur de texte par exemple), Java définit la classe **java.lang.StringBuffer** qui implémente les chaînes modifiables et de taille variable.

De plus, il existe une autre classe très utile permettant de gérer les éléments d'une chaîne de caractère: **StringTokenizer**.

2. La classe String

2.1. Présentation

La classe String est dans le package java.lang.

Elle contient plus de 60 méthodes.

Cette classe est utilisée pour créer des chaînes de caractère simples, usage courant dans l'informatique. Nous verrons que pour utiliser de longues chaîne de caractères, on préférera utiliser la classe StringBuffer.

2.2. La création d'une chaîne de caractère avec String

Nous avons vu que pour créer une chaîne de caractères, il fallait faire :

```
String str = "TOTO";
```

Il existe une autre façon de créer une chaîne de caractère :

```
String str = new String("TOTO");
```

La différence est que cette nouvelle instruction fait appel à un constructeur d'objet via l'instruction **new**.

Il existe un autre constructeur de chaîne qui accepte un tableau de caractère en entrée :

```
char[] tab = { 't', 'o', 't', 'o' };
String str = new String(tab);
```

Ce constructeur est utilisé pour convertir un tableau de caractère en String. Il est utilisé conjointement avec la méthode **toCharArray** qui permet de convertir une chaîne en un tableau de caractère :

```
String str = "ceci est un exemple";

char[] tab = str.toCharArray();
```

On peut ainsi traiter une chaîne sous la forme d'un tableau de caractère, modifier les caractères du tableau si besoin, puis convertir le tableau en chaîne.

Exemple:
Ecrire la méthode qui change tous les caractères majuscules en minuscules d'une chaînes de caractère et retourne la nouvelle chaîne.

```
public class Exemple
{
    public static void main(String args[])
    {
        String str = Terminal.lireString();

        str = convertirMajEnMin(str);

        Terminal.ecrireStringln(str);
    }

    static String convertirMajEnMin(String a_str)
    {
        char[] tab_str = a_str.toCharArray();

        for(int i=0;i<tab_str.length;i++)
            {
                if (Character.isUpperCase(tab_str[i]))
                    tab_str[i] = Character.toLowerCase(tab_str[i]);
            }

        String res_str = new String(tab_str);

        return(res_str);
    }
}
```

Remarque : On utilise deux nouvelles méthodes de la classe **Character** :
Character.isUpperCase
teste si un caractère est une majuscule
Character.toLowerCase
convertit un caractère minuscule en majuscule

2.3. Les méthodes importantes de la classe String

`char` **`charAt`**`(int index)`
Returns the `char` value at the specified index.
Returns:
the char value at the specified index of this string. The first char value is at index 0.

`int` **`compareTo`**`(String anotherString)`
Compares two strings lexicographically.
Returns:
the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

`int` **`compareToIgnoreCase`**`(String str)`

Compares two strings lexicographically, ignoring case differences.

```
public class Exemple
{
    public static void main(String args[])
    {
        Terminal.ecrireStringln("- avec compareTo  -----------------");
        test1("Lafont","Laforgue");
        test1("Laforgue","Lafont");
        test1("Dupont","Dupont");
        test1("Dupont","dupont");

        Terminal.ecrireStringln("- avec compareToIgnoreCase  -------");
        test2("Lafont","Laforgue");
        test2("Laforgue","Lafont");
        test2("Dupont","Dupont");
        test2("Dupont","dupont");

    }

    static void test1(String str1,String str2)
    {
        Terminal.ecrireStringln("---------------------------------");
        Terminal.ecrireStringln("Comparer " + str1 + " " + str2);
        Terminal.ecrireIntln(str1.compareTo(str2));
    }

    static void test2(String str1,String str2)
    {
        Terminal.ecrireStringln("---------------------------------");
        Terminal.ecrireStringln("Comparer " + str1 + " " + str2);
        Terminal.ecrireIntln(str1.compareToIgnoreCase(str2));
    }
}

- avec compareTo  -----------------
---------------------------------
Comparer Lafont Laforgue
-4
---------------------------------
Comparer Laforgue Lafont
4
---------------------------------
Comparer Dupont Dupont
0
---------------------------------
Comparer Dupont dupont
-32
- avec compareToIgnoreCase  -------
---------------------------------
Comparer Lafont Laforgue
-4
---------------------------------
Comparer Laforgue Lafont
4
---------------------------------
Comparer Dupont Dupont
0
---------------------------------
Comparer Dupont dupont
0
```

```
String concat(String str)
    Concatenates the specified string to the end of this string.

boolean equalsIgnoreCase(String anotherString)

int indexOf(int ch)
    Returns the index within this string of the first occurrence of the specified character.
int indexOf(int ch, int fromIndex)
    Returns the index within this string of the first occurrence of the specified character,
starting the search at the specified index.
int indexOf(String str)
    Returns the index within this string of the first occurrence of the specified substring.
int indexOf(String str, int fromIndex)
    Returns the index within this string of the first occurrence of the specified substring,
starting at the specified index.
```

```
import java.util.*;

public class Exemple
{
    public static void main(String args[])
    {
        String str = "Ceci est un exemple, encore un, pour trouver le mot
un";

        int index;

        index = str.indexOf("un",0);
        while (index!=-1)
            {
                Terminal.ecrireIntln(index);
                index = str.indexOf("un",index+1);
            }

        // Pour tester quand l'index est en dehors de la chaine
        Terminal.ecrireStringln("-Test ------------");
        index = str.indexOf("un",1000);
        Terminal.ecrireIntln(index);
    }
}

java Exemple
9
28
52
-Test ------------
-1
```

```
int length()
    Returns the length of this string.

String replaceAll(String regex, String replacement)
    Replaces each substring of this string that matches the given regular expression
with the given replacement.
```

```
public class Exemple
{
    public static void main(String args[])
    {
```

```
        String str = "un:deux:trois/deux/cinq:six[eu]";
        String res;
        Terminal.ecrireStringln(str);

        Terminal.ecrireStringln("------------------------");
        Terminal.ecrireStringln("remplacer deux par 2222");
        res = str.replaceAll("deux","2222");
        Terminal.ecrireStringln(res);

        Terminal.ecrireStringln("------------------------");
        Terminal.ecrireStringln("remplacer [eu] par Z");
        res = str.replaceAll("[eu]","Z");
        Terminal.ecrireStringln(res);

        Terminal.ecrireStringln("------------------------");
        Terminal.ecrireStringln("remplacer [eu] par Z");
        res = str.replaceAll("\\[eu\\]","Z");
        Terminal.ecrireStringln(res);
    }
}

un:deux:trois/deux/cinq:six[eu]
------------------------
remplacer deux par 2222
un:2222:trois/2222/cinq:six[eu]
------------------------
remplacer [eu] par Z
Zn:dZZx:trois/dZZx/cinq:six[ZZ]
------------------------
remplacer [eu] par Z
un:deux:trois/deux/cinq:sixZ
```

String **replaceFirst**(String regex, String replacement)
Replaces the first substring of this string that matches the given regular expression with the given replacement.

String[] **split**(String regex)
Splits this string around matches of the given regular expression.

```
public class Exemple
{
    public static void main(String args[])
    {
        String str = "un:deux:trois/quatre/cinq:six";

        Terminal.ecrireStringln("Avec [:/]");
        String[] tab_str = str.split("[:/]");
        for(String e:tab_str)
            Terminal.ecrireStringln(e);

        Terminal.ecrireStringln("Avec :");
        tab_str = str.split(":");
        for(String e:tab_str)
            Terminal.ecrireStringln(e);
    }
}

Avec [:/]
un
deux
trois
```

```
quatre
cinq
six

Avec :
un
deux
trois/quatre/cinq
six
```

Attention s'il y a plusieurs caractères séparateurs successifs, la méthode split ne les saute pas. Nous verrons qu'avec la classe StringTokenizer cela n'est pas le cas.

Exemple:

```
public class Exemple
{
    public static void main(String args[])
    {
        String str ="Ceci est   une    phrase composes de mots";

        String[] tab_str = str.split(" ");
        for(String s : tab_str)
            Terminal.ecrireStringln(s);
    }
}

java Exemple
Ceci
est


une



phrase
composes
de
mots
```

`boolean` **`startsWith`**`(String prefix)`
Tests if this string starts with the specified prefix.
`boolean` **`startsWith`**`(String prefix, int toffset)`
Tests if this string starts with the specified prefix beginning a specified index.

`String` **`substring`**`(int beginIndex)`
Returns a new string that is a substring of this string.
`String` **`substring`**`(int beginIndex, int endIndex)`
Returns a new string that is a substring of this string.

`char[]` **`toCharArray`**`()`
Converts this string to a new character array.

`String` **`toLowerCase`**`()`
Converts all of the characters in this `String` to lower case using the rules of the default locale
`String` **`toUpperCase`**`()`
Converts all of the characters in this `String` to upper case using the rules of the default locale.

static String **valueOf**(boolean b)
Returns the string representation of the boolean argument.
static String **valueOf**(char c)
Returns the string representation of the char argument.
static String **valueOf**(double d)
Returns the string representation of the double argument.
static String **valueOf**(float f)
Returns the string representation of the float argument.
static String **valueOf**(int i)
Returns the string representation of the int argument.
static String **valueOf**(long l)
Returns the string representation of the long argument.

Ces méthodes permettent de convertir des types primitifs en chaine de caractère :
Ces méthodes sont statiques.

```
import java.util.*;

public class Exemple
{
    public static void main(String args[])
    {
        int x=100;
        double y = 2.34e-3;

        String str1 = String.valueOf(x);
        Terminal.ecrireStringln("valeur de x en chaine : "+str1);
        String str2 = String.valueOf(y*1000);
        Terminal.ecrireStringln("valeur de y en chaine : "+str2);
    }
}

java Exemple
valeur de x en chaine : 100
valeur de y en chaine : 2.34
```

2.4. Conclusion

Nous avons déjà déterminé que la classe String est une classe constante et qu'il était donc impossible de modifier une chaîne de caractère et qu'il fallait donc créer une nouvelle chaîne de caractère quitte à re-affecter le résultat sur la chaîne elle-même.

Par exemple insérer une chaine au milieu d'une chaîne de caractère n'est pas chose facile :

```
import java.util.*;

public class Exemple
{
    public static void main(String args[])
    {
        String str = "Ceci est un exemple";
        String mot = "un";
        String ins = " autre";

        // On veut insérer le mot "autre" apres le mot un :

        int index = str.indexOf(mot);

        str = str.substring(0,index) +
```

```
            mot +
            ins +
            str.substring(index+mot.length(),str.length());

        Terminal.ecrireStringln("Resultat:");
        Terminal.ecrireStringln(str);

        // pour comparer, avec la classe StringBuffer :

        StringBuffer strb = new StringBuffer("Ceci est un exemple");

        strb.insert(strb.indexOf(mot)+mot.length(),ins);

        Terminal.ecrireStringln("Resultat:");
        Terminal.ecrireStringln(strb.toString());
    }
}

java Exemple
Resultat:
Ceci est un autre exemple
Resultat:
Ceci est un autre exemple
```

De plus, l'utilisation de la classe String n'est pas performante quand il est nécessaire de traiter des longues chaînes de caractère. Pour cela, on utilise la classe StringBuffer.

3. La classe StringBuffer

3.1. Présentation

La classe StringBuffer est dans le package java.lang.

La classe *StringBuffer* est utilisée pour gérer des chaînes de caractères de grande dimension (éditeur de texte, …).

A la différence de la classe *String*, le contenu d'un objet de type *StringBuffer* peut évolué.

Il est possible à tout instant de convertir une *String* en *StringBuffer* et vice-versa.

```
String str = "Ceci est un exemple";

StringBuffer strBuff = new StringBuffer(str);

String str2 = strbuff.toString();
```

3.2. Création d'une StringBuffer

La création d'un StringBuffer se fait grâce à l'utilisation de deux constructeurs :

```
StringBuffer strbuff1 = new StringBuffer();

String str1 = "Ceci est un autre exemple";
StringBuffer strbufff3 = new StringBuffer(str1);
```

Le premier crée une chaîne vide dans laquelle il sera possible ensuite d'y ajouter des caractères.

Le deuxième crée une chaine avec une valeur initiale de type String, dans laquelle il sera possible ensuite d'y ajouter des caractères.

3.3. Les méthodes importantes de StringBuffer

3.3.1. Ajout en fin de StringBuffer

La méthode **append** permet d'ajouter en fin de la chaîne un élément Java.

```
      StringBuffer str1;

      str1=new StringBuffer("Ceci est un exemple");

      str1.append(" : ");
      str1.append(123);
      str1.append(1.45);
      str1.append(true);
      Terminal.ecrireStringln(str1.toString());



Ceci est un exemple : 1231.45true
```

3.3.2. Longueur

On peut accéder et changer la longueur d'une *StringBuffer*: :
complétude avec des blancs ou
troncature.

```
      n=str1.length();
      Terminal.ecrireStringln("Longueur de Str1: "+n); //36

      str1.setLength(40);
      n=str1.length();
      Terminal.ecrireStringln ("Longueur de str1: "+n); //40
      Terminal.ecrireStringln (str1.toString()+"$");
              //Ceci est un exemple : 123 1.45 true    $

      str1.setLength(10);
      n=str1.length();
      Terminal.ecrireStringln ("Longueur de Str1: "+n); //10
      Terminal.ecrireStringln (str1.toString()); //Ceci est u
```

3.3.3. Insertion

Opération impossible de faire avec une *String*, on peut insérer une chaîne dans une *StringBuffer*. Les caractères sont décalés d'autant.

```
      // Insertion de "user/" avant jl

      str1 = new StringBuffer("/home/jl/bin");
```

```
      n=str1.indexOf("jl");

      if (n!=-1) str1.insert(n,"user/");

      Terminal.ecrireStringln (str1.toString()); // /home/user/jl/bin
```

3.3.4. Conclusion

Les fonctions append et insert s'utilise avec tous les types primitifs.

Il existe de nombreuses méthodes prédéfinies permettant de gérer une *StringBuffer*. Pour cela, voir la classe **java.lang.StringBuffer**.

3.3.5. Lire un fichier texte

Afin de réaliser des exercices sur les fichiers textes, j'ai ajouté une méthode dans la classe Terminal : **lireFichier** qui retourne un StringBuffer contenant le texte du fichier.

```
  import java.util.*;

  public class Exemple
  {
      public static void main(String args[])
      {
          StringBuffer str = Terminal.lireFichier("Exemple.java");

          Terminal.ecrireStringln(str.toString());
      }
  }

  Résultat de l'exécution :
  $java Exemple
  import java.util.*;

  public class Exemple
  {
      public static void main(String args[])
      {
          StringBuffer str = Terminal.lireFichier("Exemple.java");

          Terminal.ecrireStringln(str.toString());
      }
  }
```

Vous trouvez la classe Terminal sur le site dans le chapitre des cours.

A titre d'information voici le code de la méthode lireFichier :

```
      public static StringBuffer lireFichier(String nomFichier)
      {
          try{
              File fichier = new File(nomFichier);
              FileInputStream     fis     =     new     FileInputStream(new
File(nomFichier));

              byte[] buffer = new byte[(int)fichier.length()];
```

```
            fis.read(buffer);
            fis.close();
            return(new StringBuffer(new String(buffer)));
        }
        catch(Exception ex)
            {
                return(null);
            }
    }
```

Si on veut traiter chacune des lignes du fichier lu par la méthode lireFichier on peut stocker dans un tableau de chaîne chacune des lignes de la manière suivante car le caractère séparateur de ligne est le caractère '\n'.

```
import java.util.*;

public class Exemple
{
    public static void main(String args[])
    {
        StringBuffer str = Terminal.lireFichier("Exemple.java");

        String[] tab_str = str.toString().split("\n");

        for(String s:tab_str)
            Terminal.ecrireStringln(">>" + s);
    }
}

java Exemple
>>import java.util.*;
>>
>>public class Exemple
>>{
>>    public static void main(String args[])
>>    {
>>        StringBuffer str = Terminal.lireFichier("Exemple.java");
>>
>>        String[] tab_str = str.toString().split("\n");
>>
>>        for(String s:tab_str)
>>            Terminal.ecrireStringln(">>"+s);
>>    }
>>}
```

4. La classe StringTokenizer

4.1. Présentation

La classe StringTokenizer est dans le package **java.util**.

Cette classe permet de parcourir une chaîne de caractères afin d'isoler des "tokens" (ou "mots"). Le caractère séparateur de ces tokens est par défaut le caractère blanc mais il peut prendre n'importe quel caractère.

4.2. La création et l'utilisation d'une StringTokenizer

On peut utiliser deux constructeurs.

Utilisation du constructeur par défaut :

```
  import java.util.*;

  public class Exemple
  {
      public static void main(String args[])
      {
          StringTokenizer strtok1 = new StringTokenizer("   Ceci   est un
exemple   ");

          while (strtok1.hasMoreTokens())
          {
             String s = strtok1.nextToken();
             Terminal.ecrireStringln ("["+s+"]");
          }
  }

  [Ceci]
  [est]
  [un]
  [exemple]
```

La méthode hasMoreTokens permet de tester s'il existe encore un "token".

La méthode nextToken retourne le prochain token.

Le package java.lang n'est pas un package prédéfini java.

Il est donc indispensable de mettre la commande d'import des classes du package :
import java.util.*;

Sauf si on fait précéder le classe StringTokenizer de l'accès au package.

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

  public class Exemple
  {
      public static void main(String args[])
      {
        java.util.StringTokenizer strtok1 = new java.util.StringTokenizer("
Ceci  est un   exemple   ");

        while (strtok1.hasMoreTokens())
        {
            String s = strtok1.nextToken();
            Terminal.ecrireStringln ("["+s+"]");
        }
  }
```

Utilisation du constructeur qui permet de définir le caractère séparateur

```
  strtok1 = new StringTokenizer("/home/user/jl/fichier","/");
```

```
    while (strtok1.hasMoreTokens())
      {
          String s = strtok1.nextToken();
           Terminal.ecrireStringln ("->"+s);
      }

Résultat de l'exécution :
->home
->user
->jl
->fichier
```

La chaîne de définition du caractère séparateur est une "regexp".

```
public class Exemple
{
    public static void main(String args[])
    {
        String slue = "un:deux trois,quatre cinq/six:sept huit neuf";

        StringTokenizer str = new StringTokenizer(slue," ,:/\t");
        while (str.hasMoreTokens())
            {
                String s = str.nextToken();
                Terminal.ecrireStringln ("->"+s);
            }

    }
}

java Exemple
->un
->deux
->trois
->quatre
->cinq
->six
->sept
->huit
->neuf
```

Si on veut stocker les éléments extraits dans un tableau de chaîne, la méthode **countTokens** permet de connaître le nombre d'éléments sans pour cela parcourir au préalable la chaîne.

```
  public class Exemple
  {
      public static void main(String args[])
      {
          String slue = "un:deux trois,quatre cinq/six:sept huit neuf";

          StringTokenizer str = new StringTokenizer(slue," ,:/\t");

          String[] tab_str = new String[str.countTokens()];

          int nb=0;
          while (str.hasMoreTokens())
              {
                  String s = str.nextToken();
                  tab_str[nb] = s;
                  nb++;
              }

          for(String s:tab_str)
              Terminal.ecrireStringln("["+s+"]");
      }
  }

  java Exemple
  ->un
  ->deux
  ->trois
  ->quatre
  ->cinq
  ->six
  ->sept
  ->huit
  ->neuf
```

Si plusieurs caractères séparateurs se suivent alors ils sont vus comme un seul :

```
  import java.util.*;

  public class Exemple
  {
      public static void main(String args[])
      {
          String str;

          str ="Ceci est   une    phrase composes de mots";

          Terminal.ecrireStringln("----Avec split ----------------------");
          String[] tab_str = str.split(" ");
          for(String s : tab_str)
              Terminal.ecrireStringln("["+s+"]");

          StringTokenizer strtok;

          Terminal.ecrireStringln("----Avec StringTokenizer (blanc) -------
--------------");
          strtok = new StringTokenizer(str);
          while (strtok.hasMoreTokens())
              {
                  String s = strtok.nextToken();
                  Terminal.ecrireStringln("["+s+"]");
              }
```

```
          Terminal.ecrireStringln("----Avec StringTokenizer (/) -----------
----------");
          str ="Ceci/est///une//phrase/composes//de/mots";
          strtok = new StringTokenizer(str,"/");
          while (strtok.hasMoreTokens())
              {
                  String s = strtok.nextToken();
                  Terminal.ecrireStringln("["+s+"]");
              }
      }
  }

  java Exemple
  ----Avec split ---------------------
  [Ceci]
  [est]
  []
  []
  [une]
  []
  []
  []
  [phrase]
  [composes]
  [de]
  [mots]
  ----Avec StringTokenizer (blanc) ---------------------
  [Ceci]
  [est]
  [une]
  [phrase]
  [composes]
  [de]
  [mots]
  ----Avec StringTokenizer (/) ---------------------
  [Ceci]
  [est]
  [une]
  [phrase]
  [composes]
  [de]
  [mots]
```

L'instruction nextToken retourne une exception s'il n'y a pas de prochain token :

```
  import java.util.*;

  public class Exemple
  {
      public static void main(String args[])
      {
          String slue = "un:deux";

          StringTokenizer str = new StringTokenizer(slue,":");

          String s1 = str.nextToken();
          String s2 = str.nextToken();
          String s3 = str.nextToken();  // Exception
```

```
    }
}


java Exemple
Exception in thread "main" java.util.NoSuchElementException
 at java.util.StringTokenizer.nextToken(Unknown Source)
 at Exemple.main(Exemple.java:13)
```

5. Exercices

5.1. Exercice 1

Faire la méthode Java : **String[] isolerMots(String phrase)** qui retourne un tableau de chaîne contenant chacun des mots de la phrase. Si le mot existe plusieurs fois dans la phrase, il ne doit exister qu'une fois dans le tableau retourné.

Dans la phrase, les mots peuvent être séparés par plusieurs blancs.

Ecrire le programme principal qui saisie la phrase à l'écran, appelle la méthode *isolerMots* et affiche le tableau à l'écran (utilisez la boucle énumérative Java).

Faite l'algorithme du programme avant d'écrire le code.

(Sujet d'examen de Février 2011)

5.2. Exercice 2

En utilisant les méthodes de la classe StringTokenizer, on se propose de faire le programme Java qui détermine le mot le plus long. Les mots sont espacés par un ou plusieurs caractères blancs.

5.3. Exercice 3

Soit le fichier texte "Annuaire.txt" dont chaque ligne est structurée de la manière suivante :

<nom>;<prenom>;<adresse>;<telephone>

Exemple :
lafont;paul;10 rue du rosier;06 12 34 56 78
laforde;eric;1 rue pasteur;05 12 21 12 21
lafont;jules;3 rue de cominges; 89 78 67 45
zoe;eric;123 bd de sebastopol;07 08 09 89

On se propose de faire le programme Java qui permet de rechercher l'adresse ou le numéro de téléphone d'une personne en fonction de son nom et/ou de son prénom. S'il existe plusieurs fois alors on affiche toutes les personnes trouvées.

Le programme demande la saisie du nom et/ou du prenom.
Le programme boucle tant que l'on veut rechercher quelqu'un.

Pour lire le fichier vous utilisez la méthode lireFichierTexte de la classe Terminal.

La classe Terminal est sur le site http://jacques.laforgue.free.fr dans Cour NFA031 et NFA032 > Outils > **Outil00a_TerminalSansPackage**

6. Corrections des exercices

6.1. Exercice 1

Algorithme du programme principal :

```
  Debut
     Saisir la phrase;
     Appeler le traitement qui retourne un tableau contenant les mots de la
phrase sans redondance;
     Afficher le tableau résultat du traitement;
  Fin.
```

Algorithme du traitement :

```
  Debut
     Créer les tokens de la phrase avec le caractère blan comme caractère
séparateur;
     Créer une liste de chaine vide de chaine TMP;
     Pour chaque token de tokens faire
        Si token n'est pas dans TMP alors
          Ajouter token dans TMP
        Finsi
     Finpour
     Retourner TMP sous la forme d'un tableau de chaine;
  Fin
```

```
  import java.util.*;

  public class Exercice1
  {
      // Programme principal
      //
      static public void main(String args[])
      {
          Terminal.ecrireString("Saisir la phrase: ");
          String phrase = Terminal.lireString();   // Saisi
          String[] mots = isolerMots(phrase);      // Traitement
          for(String m:mots)                       // Affichage
              Terminal.ecrireStringln(m);
      }

      // Méthode qui isole de maniere unique chacun des mots d'une phrase
      //
      static String[] isolerMots(String phrase)
      {
          //  On isole les mots
          StringTokenizer strtok = new StringTokenizer(phrase);
          int n = strtok.countTokens();

          // Tableau qui contiendra les mots
          //   au maximume ce tableau est de la dimension des mots de la
phrase
          ArrayList<String> tabMots = new ArrayList<String>();

          // On parcourt les mots de la phrase et on l'ajoute dans le
          //  tableau de mot s'il n'existe pas deja
          //
          while(strtok.hasMoreTokens())
```

```
        {
            String mot = strtok.nextToken();

            // On recherche s'il n'existe pas deja
            if (! tabMots.contains(mot))
                tabMots.add(mot);
        }
      String[] t = new String[tabMots.size()];
      tabMots.toArray(t);
      return(t);
    }
}
```

6.2. Exercice 2

```
import java.util.*;

public class Exercice2
{
    public static void main(String args[])
    {
        String str;

        str ="Ceci est   une    phrase composes de mots";

        StringTokenizer strtok = new StringTokenizer(str);
        String[] tab_str = new String[strtok.countTokens()];

        int nb=0;
        while (strtok.hasMoreTokens())
            {
                String s = strtok.nextToken();
                tab_str[nb]=s;
                nb++;
            }

        String mot_max = "";
        for(String s : tab_str)
            if (s.length()>mot_max.length())
                mot_max = s;
        Terminal.ecrireStringln("Le mot le plus long est : " +mot_max);
    }
}
```

6.3. Exercice 3

```
import java.util.*;

public class Exercice3
{
    public static void main(String args[])
    {
        // Chaque ligne est stockee dans un tableau
        String[] tab_str = Terminal.lireFichierTexte("Annuaire.txt");


        // On verifie le contenu du tableau
        for(String e:tab_str)
            Terminal.ecrireStringln(">>"+e);


        boolean fini=false;
        while(! fini)
            {
                // Saisie du nom et/ou du prenom
                //
                Terminal.ecrireStringln("====================");
                Terminal.ecrireString("Nom    : ");
                String nom = Terminal.lireString();
                Terminal.ecrireString("Prenom : ");
                String prenom = Terminal.lireString();

                // Si saisie de deux haines vides alors on sort
                // du programme
                if ( (nom.equals("")) && (prenom.equals("")) )
                    fini = true;
                else
                    {
                        // Recherche
                        //
```

```
                        for(String s:tab_str)
                            {
                                // On extrait les informations
                                 //de la ligne
                                String[] infos    = s.split(";");
                                String nomc       = infos[0];
                                String prenomc    = infos[1];
                                String adressec   = infos[2];
                                String numeroc    = infos[3];

                                // Si le nom est vide
                                if (nom.equals(""))
                                    {
                                        if (prenom.equals(prenomc))
                                            {
                                                Terminal.ecrireStringln(s);
                                            }
                                    }
                                else
                                    {
                                        // Si le prenom est vide
                                        if (prenom.equals(""))
                                            {
                                                if (nom.equals(nomc))
                                                    {
                                                        Terminal.ecrireStringln(s);
                                                    }
                                            }
                                        else // nom et prenom non vides
                                            {
                                                if ( (nom.equals(nomc))&&(prenom.equals(prenomc)) )
                                                    {
                                                        Terminal.ecrireStringln(s);
                                                    }
                                            }
                                    }
                            }
                    }
            }
```

```
    }
}


java Exemple
>>lafont;paul;10 rue du rosier;06 12 34 56 78
>>laforde;eric;1 rue pasteur;05 12 21 12 21
>>lafont;jules;3 rue de cominges; 89 78 67 45
>>zoe;eric;123 bd de sebastopol;07 08 09 89
====================
Nom    : lafont
Prenom :
lafont;paul;10 rue du rosier;06 12 34 56 78
lafont;jules;3 rue de cominges; 89 78 67 45
====================
Nom    : zoe
Prenom :
zoe;eric;123 bd de sebastopol;07 08 09 89
====================
Nom    : lafont
Prenom : pauk
====================
Nom    : lafont
Prenom : paul
lafont;paul;10 rue du rosier;06 12 34 56 78
====================
Nom    :
Prenom : paul
lafont;paul;10 rue du rosier;06 12 34 56 78
====================
Nom    :
Prenom : eric
laforde;eric;1 rue pasteur;05 12 21 12 21
zoe;eric;123 bd de sebastopol;07 08 09 89
====================
Nom    :
Prenom :
```

Les corrections de ces 3 exercices sont sur le site : **Exercice03_String**