Chapitre IV

Chaines de Caractères et Enumérations

Introduction

Ce chapitre décrit brièvement deux types spécifiques d'objets qui sont les chaines de caractères et les énumérations. Les chaines de caractères sont créées et manipulées à travers les classes String, StringBuilder ou StringBuffer comportant des méthodes dédiées aux traitements spécifiques sur les chaines de caractères tels que la comparaison, la concaténation, l'extraction de sous-chaines, etc. Les énumérations sont des classes à objets dénombrables décrites à l'aide du mot clé *enum* et pouvant être manipulées via des méthodes spécifiques.

1. Les chaînes de caractères

En java, les chaînes de caractères sont des objets de la classe **String** (ou d'autres classes StringBuffer, StringBuilder) définie dans le package java.lang importé par défaut. Ainsi, à un objet de la classe **String**, on peut appliquer des méthodes prédéfinies de cette classe.

1.1 Création et manipulation de chaînes de caractères

Une chaîne de caractères peut être déclarée par : **String nom_chaîne;**
Elle peut être initialisée dès sa création de la manière suivante :

nom de la chaîne = "valeur de la chaîne" ;

On pourra écrire par exemple, **String s = "cours de java";**

1.2 Lecture d'une chaîne de caractères

Pour la lecture d'une chaîne de caractères au clavier, on pourra utiliser les classes **InputStreamReader** et **BufferedReader** (afin de convertir le codage de caractères vers un codage java) définies dans le package **java.io**. Par exemple :

```
String chaine;
InputStreamReader e = new InputStreamReader (System.in)
BufferedReader entree = new BufferedReader (e);
chaine = entree.readLine();
```

On peut utiliser la classe **Scanner** (définie dans le package **java.util**) ne nécessitant pas une conversion de caractères, on écrira alors :

```
java.util.Scanner entree = new java.util.Scanner(System.in);
System.out.println (" entrez la chaîne :") ;
String chaine = entree.next() ;  // ou entree.nextLine() ;
```

1.3 Concaténation de chaînes de caractères

Le langage java définit un opérateur de concaténation de chaînes de caractères, l'opérateur +. Le tableau suivant résume les résultats de concaténation selon les types des opérandes à concaténer.

Opérande 1	Opérande 2	Concaténation
String	String	Concaténer la 2ème chaîne à la première.
String	Boolean	Concaténer la chaîne "true" ou "false" à la chaîne de caractères.
Boolean	String	Concaténer la chaîne de caractères à chaîne "true" ou "false".
String	Nombre	Concaténer la chaîne représentant le nombre à la chaîne de caractères.
Nombre	String	Concaténer la chaîne de caractères à la chaîne représentant le nombre.
String	Char	Concaténer le caractère à la chaîne de caractères.
Char	String	Concaténer la chaîne de caractères au caractère.
Objet	String	Convertir l'objet en chaîne de caractères en utilisant la méthode ***toString*** et concaténer ensuite la chaîne de caractères.
String	Objet	Concaténer l'objet converti à la chaîne de caractères.

Remarque : la méthode ***toString*** est définie dans la super-classe ***Object.*** Cette méthode est utilisée par la machine Java toutes les fois où elle a besoin de représenter un objet sous forme d'une chaîne de caractères. Par exemple, il est correct d'écrire le code suivant :

```
Etudiant E = new Etudiant(20130007, "Belkadi", "Islam", ... ) ;
System.out.println("L'étudiant E : " + E);
```

L'objet **E** sera ainsi converti en objet de type **String** et le programme va simplement afficher le nom de l'objet suivi de son adresse (hexadécimale) en mémoire. Il faut noter que la méthode **toString** peut être redéfinie dans toute classe implémentée pour afficher la description de l'objet que le programmeur veut faire apparaître à l'écran.

La classe *String* est une classe spéciale :

- les chaînes de caractères peuvent se concaténer à l'aide de l'opérateur +, ou à l'aide de la méthode *concat*. Ces deux concaténations ne sont pas équivalentes lorsque l'opérande de droite vaut *null*.
- les instances peuvent ne pas être créées explicitement *String s = "abc" ;* au lieu de *String s = **new** String("abc") ;*

Exemple
Voici un exemple qui illustre les règles de concaténation montrées dans le tableau précédent
class Concat_chaines

```
{ public static void main (String [ ] arg)
    { String  s1= " le ciel ";
      double  x= 135.2 ;
      System.out.println (" longueur de s1= " + s1.length() );
      System.out.println (" Exact = " +  (2= = 1+1) );
      System.out.println ("System.out.println (" Valeur de x = " + x );
      System.out.println (" Poids = " + new Long (10) + "Kilos" );
      System.out.println ( "10 + 5" + "font" + 1+5 );
      String  S2 = S1.concat("et la terre");
      System.out.println (S1);  System.out.println (S2);      } }
```

Question: commenter ce programme et donner les différents résultats d'affichage.

1.4 Comparaison de chaînes de caractères

Pour comparer des chaînes de caractères, on peut utiliser les méthodes equals et compareTo définies dans la classe **String**.

Exemple : l'exemple suivant illustre l'utilisation des deux méthodes.

```
class  Compare_chaine
{ public static void main (String [ ] arg)
        { String r = "essai";
          String s = "es" + "sai";
          String t = "essais";
          String u = "ESSAI".toLowerCase ();
          System.out.println (" r.equals(u) vaut :  " + r.equals(u));
          System.out.println (" r.compareTo (u) =  " + r.compareTo(u));
          System.out.println (" r.compareTo (t) =  " + r.compareTo(t));
          System.out.println (" t.compareTo (r) =  " + t.compareTo(r));
       }
        }
```

Question: commenter ce programme et donner les différents résultats d'affichage.

1.5 Méthodes pour la manipulation de chaînes de caractères

Le tableau suivant énumère les méthodes usuelles de manipulation de chaînes de caractères, objets de la classe **String**.

Signature de la méthode	Description
public **String**() public **String**(String s) public int **length**() public char **charAt**(int index)	Constructeur Constructeur Longueur de la chaîne Retourne le caractère à la position index

public String **substring**(int dbt,int fin)	Extrait la chaîne entre les positions dbt et fin
public boolean **equals**(Object o)	Test d'égalité
public boolean **startsWith**(String s)	Teste si deux chaines sont égales
public boolean **endsWith**(String s)	Teste si le début de la chaîne est égal à la chaîne s
public int **compareTo**(String s)	Comparaison des 2 chaînes, (0 si égalité, <0 si elle est inférieure, >0 sinon)
public int **indexOf**(char c)	Retourne la position du caractère c dans la chaîne Retourne
public int **lastIndexOf**(char c)	la dernière position du caractère c
public int **indexOf**(char c, int i)	Retourne la position de c à partir de i
public int **indexOf**(String s)	Retourne la position de la sous-chaîne s
public String **replace**(char c, char d)	Remplace toute occurrence de c par d
public String **toLowerCase**()	Conversion en minuscules
public String **toUpperCase**()	Conversion en majuscules
public char[] **toCharArray**()	Conversion de la chaîne en tableau de caractères
public String **trim**()	Suppression des espaces en début et fin de la chaîne
public static String **valueOf**(char t[])	Conversion d'un tableau de caractères en String

Autres méthodes

int compareTo(Object o)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0
int compareTo(String anotherString)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0. La comparaison est une comparaison lexicographique.
int compareToIgnoreCase(String str)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0. La comparaison est une comparaison lexicographique, ignorant la casse.
boolean equals(Object anObject)	Compare la chaîne a un objet et retourne *true* en cas d'égalité et *false* sinon
boolean equalsIgnoreCase(Object anObject)	Compare la chaîne a un objet et retourne *true* en cas d'égalité et *false* sinon (on ignore la casse)
boolean regionMatchesIgnoreCase (**int** toffset, String other, **int** ooffset, **int** len)	Teste l'égalité de deux parties de chaînes en ignorant la casse
boolean regionMatches(**int** toffset, String other, **int** ooffset, **int** len)	Teste l'égalité de deux parties de chaînes

1.6 Les classes StringBuffer et StringBuilder

Un objet chaîne de caractères de la classe ***String*** ne permet pas de modifier le contenu de la chaîne créée (on parle d'objet immuable). Afin de donner la possibilité de modifier le contenu d'une chaîne, on utilise les classes **StringBuffer** ou **StringBuilder** définies dans le paquetage **java.lang** ; ces classes possèdent les mêmes fonctionnalités. Par exemple, y sont définies les méthodes :
charAt(int i) qui renvoie le ième caractère de la chaîne concernée ; et
setCharAt(int i, char c) qui remplace le ième caractère de la chaîne par le caractère c.

Remarque

- Les méthodes de la classe *StringBuffer* sont synchronisées, et peuvent être utilisées par plusieurs *thread* sur une même chaîne de caractères.

- Les méthodes de la classe *StringBuilder* ne sont pas synchronisées, elles sont donc plus rapides que celle de la classe *StringBuffer*, mais ne sont pas «thread safe».

Exemple
Cet exemple permet de vérifier si une chaîne de caractères est un palindrôme, en créant une chaine inverse via la classe StringBuilder.

```
class Chaine
      { String ch ;
        Chaine (String s) { ch = s;}   /* constructeur */
        /* methode inverse qui retourne la chaîne inverse */
        String inverse ()
        { StringBuilder envers = new StringBuilder (ch);
          int i, taille = ch.length();
          for (i = 0; i< taille; i++)
          envers.setCharAt (i, ch.charAt (taille-i-1));     // mettre le car de la position i à la
                                                              position taille-i-1
        return new String (envers); }   // on obtient une chaîne contenant la chaîne envers

       /* methode  qui vérifie si la chaîne est palindrôme */
        boolean Palind ()
        { return ch.equals (inverse())}          // vérifier si la chaîne est égale à son inverse

class  TestPalindrome
 { public static void main (String [ ] arg)
      { String s = "radar";
        Chaine chaine = new Chaine (s);
        System.out.println (" Inverse de :  " + s + " :" + chaine. inverse ());
        System.out.println ( s + " est un palindrome :" + chaine. Palind ());
        s = "classeur";
   System.out.println ( s + " est un palindrome :" + (new Chaine (s). Palind ());
    } }
```

Le tableau suivant décrit quelques unes des méthodes de la classe **StringBuffer**

Signature de la méthode	Description
public int **length**()	Longueur de la chaîne
public char **charAt**(int index)	Retourne le caractère à la position index
public void **getChars**(int dbt, int fin, char dst[ ],int index)	Recopie la sous-chaîne entre les positions dbt et fin dans le tableau dst, à partir de l'indice index
public int **capacity**()	Retourne la capacité courante de la chaîne
public void **setCharAt**(int index, char c)	Met le caractère c à l'indice index
public StringBuffer **append**(Object obj)	Concatène la représentation textuelle de l'objet obj
StringBuffer append(**boolean** b)	Concatène la représentation en chaîne du booléen.
StringBuffer append(**char** c)	Concatène la représentation en chaîne du caractère.
StringBuffer append(**char**[] str)	Concatène la représentation en chaîne du du tableau de caractères.

StringBuffer append(**char**[] str, **int** offset, **int** len)	Concatène la représentation en chaîne du tableau de caractères.
StringBuffer append(**double** d)	Concatène la représentation en chaîne du double.
StringBuffer append(**float** f)	Concatène la représentation en chaîne du float.
StringBuffer append(**int** i)	Concatène la représentation en chaîne du int.
StringBuffer append(**long** l)	Concatène la représentation en chaîne du long.
StringBuffer append(Object obj)	Concatène la représentation en chaîne de l'objet
StringBuffer append(String str)	Concatène la représentation en chaîne de la chaîne.

D'autres méthodes existent pour la conversion en types primitifs, formatage des chaines, etc.

2. Les Types Enumérés

Un type énuméré permet de définir un ensemble d'identificateurs pour des constantes, comme par exemple les jours de la semaine, les mois de l'année, le statut social d'une personne,…

2.1 Déclaration d'un type énuméré

A partir du JDK 5.0, le langage java offre la possibilité de définir un type énuméré à l'aide de classes en utilisant le mot **enum** au lieu de class.

Exemple

```
enum Saison { HIVER, PRINTEMPS, ETE, AUTOMNE }
class TestEnum
        { public static void main( String arg[ ])
         { Scanner e= new Scanner (System.in);
                Saison s = Saison.valueOf (e.next());
                   /* s est un objet de la classe Saison */
           switch (s)
            { case HIVER :
                 System.out.Println ("c'est le premier trimestre de l'année");
                 break ;
                 case PRINTEMPS: System.out.Println ("c'est le second trimestre"); break ;
                 case ETE: System.out.Println ("c'est le troisième trimestre");
                       break;
                 case AUTOMNE: System.out.Println ("c'est le quatrième
                              trimestre") ; break ;
                 default : System.out.Println (" ne correspond à aucune saison") ;}
           }     }
```

2.2 Méthodes pour la manipulation de types énumérés

valueOf : renvoie la valeur d'un objet de l'énumération en ayant un argument de type String.
Exemple : Sting s = "hiver" Saison.**valueOf** (s.toUpperCase()) = HIVER

ordinal () : renvoie le numéro d'ordre d'un objet dans l'énumération.

Exemple : AUTOMNE.**ordinal**() renvoie la valeur 3

values() : parcourt toutes les valeurs du type énuméré, peut être utilisé dans une boucle for.

```
for (Saison s: Saison.values())      // parcourir les éléments de l'énumération
        System.out.println (s);
```

2.3 Définition d'un constructeur pour un type énuméré

Dans une classe déclarée enum, il est possible de définir un constructeur de la classe ainsi que d'autres méthodes.

Exemple

On énumère les semestres d'étude pour un cycle de licence en précisant le nombre de modules par semestre. S1, S2, …S6 représentent les valeurs principales de l'énumération et les valeurs 6, 5,…0 représentent les valeurs secondaires qu'il faut faire correspondre à un nom d'attribut.

```
enum SemEtude { S1 (6), S2(5); S3(6), S4(5), S5(4), S6(0)
        int nbmodules;
        SemEtude (int nb)        // constructeur de la classe
            {nbmodules = nb ;}

          String ChargeEtude()      // définition d'une méthode
           {switch (this)
              { case S1:
                case  S3 :  return ("Trop chargé") ;
                case  S2 :
                case  S4 :
                case S5 :  return ("Moins chargé") ;
                case  S6:   return ("Pas de modules") ;

            } }
          }

class TestCharge
 { public static void main( String arg[ ])
        { SemEtude  S = SemEtude.S3;
        System.out.println ("numéro de  " + S + "est : "+ S.ordinal());
        S.ChargeEtude();          // Invocation de la méthode ChargeEtude
        for (SemEtude  s: SemEtude.values())
                // parcourir les éléments de l'énumération
              System.out.println (" En " + s + "le nombre de modules est  : "+
                                  s.nbmodules);
        }}
```

Question : Quels sont les différents affichages du programme ?

Remarque : Il est possible d'avoir plusieurs valeurs secondaires séparées par des virgules pour la même valeur principale. A chaque valeur secondaire, il faut faire correspondre un nom d'attribut. Par exemple, on pourrait envisager la classe d'énumération Mois définie comme suit

```
enum Mois
        {Janvier (Jan, 31), Février (Fev, 28), ..., Décembre (Dec, 31) ;
                //attributs
         String abrev ;
         int nbjour ;
                 //constructeur
         Mois (String abrev, int nb) {this.abrev= abrev; nbjour = nb); }
        void Afficher ()
          { for Mois m: Mois.values())
                System.out.println( m + " :"   + m.abrev  + " :"   + m.nbjour); }
       }
```