Tema 5

Arreglos y Cadenas en Java

Arreglos

Un arreglo es una lista de variables del mismo tipo que se encuentran en localidades contiguas de memoria y que comparten el mismo nombre. Cada una de las variables de la lista, llamadas **elementos**, puede ser accedida por el nombre del arreglo y su posición en él. Dependiendo de la distribución lógica (no física) de sus elementos, los arreglos pueden clasificarse en:

- Unidimensionales, si visualizamos a los elementos acomodados en una fila (o columna).
- Bidimensionales, si los visualizamos acomodados en una tabla.
- Tridimensionales, si los visualizamos acomodados en varias tablas formando una estructura tridimensional.
- Etc. No hay restricciones en el número de dimensiones que un arreglo pueda tener.

Arreglos Unidimensionales

En un arreglo unidimensional, la posición de un elemento en el arreglo está dada por un valor entero llamado **índice** que representa la distancia de ese elemento con respecto al primer elemento del arreglo. Por ejemplo si deseamos almacenar las calificaciones de un grupo de alumnos podemos tener un arreglo llamado califs formado de 50 variables:

`califs[0]` representa la calificación del primer alumno,
`califs[1]` representa la calificación del segundo alumno, etc.

El número entre corchetes es el índice. Note que el primer elemento tiene un índice igual a 0 y el último un índice igual a n-1, donde n es el número de elementos del arreglo.

Referencia a un Arreglo Unidimensional

Al igual que con los objetos, el nombre de un arreglo es una referencia al arreglo, esto es, utilizamos el nombre para acceder a los elementos del arreglo.

La sintaxis para declarar la referencia a un arreglo unidimensional es la siguiente:

```
tipo nomArreglo[];
```

ó

```
tipo[] nomArreglo;
```

`tipo` es el **tipo base** del arreglo, el tipo de cada una de las variables que forman el arreglo, y puede ser cualquier tipo de datos: primitivos o clases. `nomArreglo` es un identificador, el nombre de la referencia al arreglo.

Por ejemplo:

```
int califs[];
double[] precios;
Cancion canciones[];
```

Creación de un Arreglo Unidimensional

Al igual que con los objetos, los arreglos deben crearse antes de emplearse, esto reserva memoria para el arreglo. La sintaxis para crear un arreglo es la siguiente:

```
nomArreglo = new tipo[tamArreglo];
```

`tamArreglo` es una expresión de tipo entero que indica el tamaño del arreglo, el número de variables de la lista. Los corchetes encerrando a `tamArreglo` no indican que éste sea opcional. Aquí forman parte de la sintaxis de la creación de un arreglo.

Por ejemplo:

```
califs = new int[50];
precios = new double[100];
canciones = new Cancion[20];
```

La declaración de la referencia a un arreglo y el reservado de memoria para el mismo pueden combinarse en una sentencia cuya sintaxis es:

```
tipo nomArreglo[] = new tipo[tamArreglo];
```

ó

```
tipo[] nomArreglo = new tipo[tamArreglo];
```

Por ejemplo:

```
int califs[] = new int[50];
double[] precios = new double[100];
Cancion canciones[] = new Cancion[20];
```

Inicialización de Arreglos Unidimensionales

Los arreglos al igual que las variables de los tipos básicos pueden inicializarse al momento de crearse. La sintaxis para inicializar un arreglo es

```
tipo nomArreglo[] = {cte1 [, cte2] ... }
```

ó

```
tipo[] nomArreglo = {cte1 [, cte2] ... }
```

Donde `cte1 [, cte2] ...` es una lista de constantes encerrada entre llaves, { y }, que son los valores a los que se inicializan los elementos del arreglo. El tamaño del arreglo es el número de constantes. Por ejemplo, en la declaración

```
int x[] = {0, 1, 2, 3, 4};
```

estamos declarando al arreglo x y lo estamos inicializando a:

x	0	1	2	3	4

Acceso a los elementos de un Arreglo Unidimensional

Para acceder a un elemento de un arreglo unidimensional se utiliza el nombre del elemento, tal como lo haríamos con cualquier variable. El nombre de cada elemento de un arreglo está formado por el nombre del arreglo y su posición en el arreglo. Esta posición se conoce como el **índice** del elemento. El índice del primer elemento de un arreglo siempre es cero. El índice de un elemento puede expresarse mediante cualquier expresión entera que al evaluarse nos indica el elemento en particular al que queremos acceder. Por ejemplo

```
califs[4] = 8;
```

le asigna al quinto elemento del arreglo `califs` el valor de 8.

```
califs[i]
```

accede al i-ésimo elemento del arreglo `califs`, mientras que

```
mats[j + 3]
```

accede al j-ésimo + 3 elemento del arreglo `mats`.

Por ejemplo supongamos que deseamos acceder en forma secuencial a los elementos del arreglo `califs` para sacar el promedio de las calificaciones. El código para hacer esto sería

```
suma = 0;
for(i = 0; i < nAlumnos; i++) suma += califs[i];
promedio = (double)suma/nAlumnos;
```

donde `nAlumnos` es el número de calificaciones que están almacenadas en el arreglo. Este valor puede ser menor o igual al tamaño del arreglo `califs`.

Cada vez que se le asigna un valor u objeto a un elemento de un arreglo, el sistema de ejecución de Java verifica que el tipo del arreglo sea compatible con el tipo del valor u objeto asignársele. Si no son compatibles ocurre una excepción del tipo: `java.lang.ArrayStoreException`.

Cada vez que se accede a un elemento de un arreglo, el sistema de ejecución de Java verifica que el valor del índice es un valor válido, esto es, que es número comprendido entre 0 y *tamArreglo* – 1. De no ser así ocurre una excepción del tipo: `java.lang.ArrayIndexOutOfBoundsException`.

Arreglos Unidimensionales y Métodos

En Java los arreglos son objetos. Podemos obtener el tamaño del arreglo en elementos de la siguiente manera:

```
nomArreglo.length
```

En Java podemos pasarle como parámetro a un método, una referencia a un arreglo. No se le está pasando una copia del arreglo, sólo la referencia. La sintaxis de un parámetro que representa la referencia a un arreglo unidimensional es la siguiente:

```
tipo nomParArreglo[]
```

donde *`tipo`* es el **tipo base** del arreglo y *`nomParArreglo`* es el nombre del parámetro que representa la referencia al arreglo. Note que los corchetes están vacíos.

Al llamar al método, el argumento será la referencia al arreglo, el cual es el nombre del arreglo:

```
nomArreglo
```

También es posible que un método nos regrese una referencia a un arreglo. La sintaxis para declarar que el método regresa una referencia a un arreglo unidimensional es:

```
tipo[] nomMetodo() {
   ...
}
```

donde *tipo* es el **tipo base** del arreglo y *nomMetodo* es el nombre del método que nos regresa la referencia a un arreglo.

El ciclo `for-each`

La versión 5 de java extendió la sintaxis del ciclo `for` para facilitar la iteración sobre los elementos de un arreglo o de una colección. Las colecciones se verán en el Tema 6. A esta nueva variante del ciclo `for` se le conoce como ciclo `for-each`.

Si deseamos iterar sobre los elementos de un arreglo usando ciclo `for` normal, la sintaxis sería:

```
for(int i = 0; i < nomArreglo.length; i++) {
    tipo nomVariable = nomArreglo[i];
    cuerpo del ciclo
}
```

El mismo ciclo reescrito usando el ciclo `for-each` tendría la siguiente sintaxis:

```
for(tipo nomVariable: nomArreglo) {
    cuerpo del ciclo
}
```

Haciendo más claro el código. Por ejemplo el siguiente código que utiliza el `for` normal:

```
suma = 0;
for(i = 0; i < califs.lenght; i++) suma += califs[i];
promedio = (double)suma/ califs.lenght;
```

podría rescribirse usando el `for-each` de la siguiente manera:

```
suma = 0;
for(int calif: califs) suma += calif;
promedio = (double)suma/ califs.lenght;
```

Limitaciones del ciclo for-each

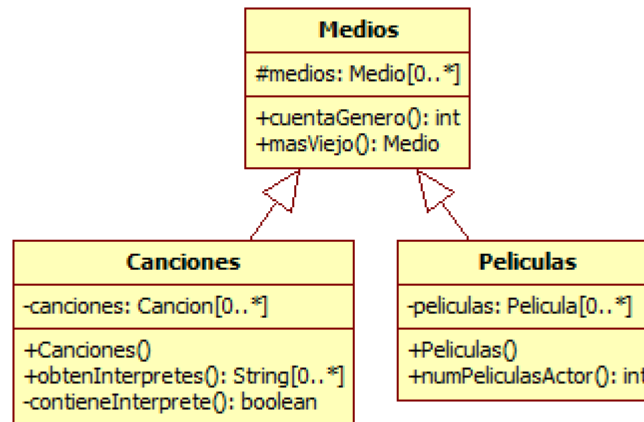
El ciclo for-each tiene ciertas limitaciones en su uso con arreglos:

- Sólo podemos acceder a los elementos del arreglo. No podemos asignarle un valor a los elementos de un arreglo.

- Sólo se puede trabajar con un solo arreglo. No podemos, por ejemplo comparar dos arreglos.
- Sólo se puede accede un elemento a la vez. No podemos comparar, por ejemplo elementos consecutivos.
- Sólo se puede recorrer la colección hacia adelante y en incrementos de uno.
- No puede iterarse sobre sólo parte de la colección. El ciclo `for-each` recorre toda la colección.
- No se use si se desea compatibilidad con código previo a la versión 5 de java.

Ejemplos Sobre Arreglos Unidimensionales

Como ejemplos de arreglos unidimensionales, supongamos que deseamos implementar varias operaciones sobre un conjunto de canciones o películas almacenadas en arreglos unidimensionales. Para ello implementamos las clases del siguiente diagrama de clases:



En la clase `Medios` el atributo `medios` es una referencia al arreglo `canciones` o `peliculas` creados en las clases `Canciones` o `Peliculas`.

Los métodos de la clase `Medios` trabajan tanto con canciones como con películas.

- `int cuentaGenero(String nombre)` nos regresa el número de canciones o películas del género cuyo nombre está dado por el parámetro ya sea en el arreglo canciones o películas.
- `Medio masViejo()` nos regresa la canción o película más vieja ya sea en el arreglo canciones o películas.

Medios.java

```
/*
 * Medios.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */
```

```
package catalogos;

import objetosNegocio.Medio;

/**
 * Esta clase permite realizar algunas operaciones con objetos de tipo medio:
 * Canciones o peliculas, almacenadas en un arreglo.
 *
 * @author mdomitsu
 */
public class Medios {

    protected Medio medios[];

    /**
     * Cuenta los medios del mismo genero que el parámetro
     * @param nombre Nombre del género de los medios a contar
     * @return Numero de medios del mismo genero que el parámetro
     */
    public int cuentaGenero(String nombre) {
        int cuenta = 0;

        // Recorre el arreglo
        for (Medio medio : medios) {
            // Si es el género especificado
            if (nombre.equals(medio.getGenero().getNombre())) {
                // Incrementa el contador de generos
                cuenta++;
            }
        }

        return cuenta;
    }

    /**
     * Regresa el medio de mayor antiguedad
     * @return El medio de mayor antiguedad
     */
    public Medio masViejo() {
        // Suponemos que el medio de mayor antiguedad es el primero
        Medio medioMasViejo = medios[0];

        // Recorre el arreglo
        for (Medio medio : medios) {
            // Si hay otro de mayor antiguedad, substituyelo
            if (medio.getFecha().before(medioMasViejo.getFecha())) {
                medioMasViejo = medio;
            }
        }

        return medioMasViejo;
    }
}
```

Los métodos de la clase `Canciones` solo operan sobre canciones.

- `Canciones(Cancion canciones[])` Inicializa el arreglo `canciones` de la clase al arreglo de su parámetro.
- `String[] obtenInterpretes()` nos regresa en un arreglo los diferentes interpretes de las canciones del catálogo.
- `boolean contieneInterprete(String interpretes[], String interprete, int numInterpretes)` regresa verdadero si el arreglo `interpretes` con `numInterpretes` contiene al intérprete `interprete`.

Canciones.java

```
/*
 * Canciones.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */
package catalogos;


import objetosNegocio.Cancion;

/**
 * Esta clase permite realizar algunas operaciones con objetos de tipo
 * Cancion almacenadas en un arreglo.
*
 * @author mdomitsu
 */
public class Canciones extends Medios {
  // Arreglo para almacenar las canciones
    private Cancion canciones[];

    /**
     * Inicializa el atributo de la clase y hace que la referencia medios
     * apunte al arreglo canciones
     * @param canciones El arreglo de canciones
     */
    public Canciones(Cancion canciones[]) {
        this.canciones = canciones;
        medios = canciones;
    }

    /**
     * Regresa un arreglo con los interpretes de las canciones del catalogo
     * @return
     */
    public String[] obtenInterpretes() {
        String interpretesTemp[] = new String[canciones.length];
        String interpretes[] = new String[canciones.length];
        int numInterpretes = 0;

        // Recorre el arreglo
        for(Cancion cancion: canciones) {
            // Si el interprete de la cancion no esta en el arreglo de
            // interpretes
            if(!contieneInterprete(interpretesTemp, cancion.getInterprete(),
                numInterpretes)) {
```

```
                // Agrega al interprete al arreglo de interpretes
                interpretesTemp[numInterpretes] = cancion.getInterprete();
                numInterpretes++;
            }

            // Ajusta el tamaño del arreglo de interpretes al número de
            // interpretes
            interpretes = new String[numInterpretes];
            System.arraycopy(interpretesTemp, 0, interpretes, 0,
                             numInterpretes);
        }

        return interpretes;
    }

    /**
     * Determina si el interprete del parametro se encuentra en el arreglo
     * del parametro
     * @param interpretes Arreglo de interpretes
     * @param interprete Interprete a buscar en el arreglo
     * @param numInterpretes numero de interpretes en el arreglo
     * @return true si el interprete del parametro se encuentra en el arreglo
     * del parametro, false en caso contrario,
     */
    private boolean contieneInterprete(String interpretes[],
                       String interprete, int numInterpretes) {
        // Recorre el arreglo
        for(int i = 0; i < numInterpretes; i++) {
            // Si el interprete esta en el arreglo, regresa verdadero
            if(interprete.equals(interpretes[i])) return true;
        }

        return false;
    }
}
```

Los métodos de la clase `Peliculas` solo operan sobre películas.

- `Peliculas (Pelicula peliculas [])` Inicializa el arreglo `peliculas` de la clase al arreglo de su parámetro.
- `int numPeliculasActor(String actor)` Regresa el número de películas en la que aparece el actor del parámetro:

Peliculas.java

```
/*
 * Peliculas.java
 *
 * Creada el 15 de septiembre de 2007, 12:21 PM
 */
package catalogos;


import objetosNegocio.Pelicula;
```

```
/**
 * Esta clase permite realizar algunas operaciones con objetos de tipo
 * Pelicula, almacenadas en un arreglo.
 *
 * @author mdomitsu
 */
public class Peliculas extends Medios {
  // Arreglo para almacenar las películas
  private Pelicula peliculas[];

  /**
   * Inicializa el atributo de la clase y hace que la referencia medios
   * apunte al arreglo peliculas
   * @param peliculas El arreglo de peliculas
   */
  public Peliculas(Pelicula peliculas[]) {
      this.peliculas = peliculas;
      medios = peliculas;
  }

  /**
   * Regresa el numero de peliculas en las que actua el actor del parametro
   * @param actor que actua en las peliculas
   * @return Numero de peliculas en las que actua el actor del parametro
   */
  public int numPeliculasActor(String actor) {
    int nPeliculasActor = 0;

    // Recorre el arreglo
    for(Pelicula pelicula: peliculas) {
      // Si es el actor especificado
      if(pelicula.getActor1().equals(actor) ||
         pelicula.getActor2().equals(actor))
        // incrementa el contador
        nPeliculasActor++;
    }

    return nPeliculasActor;
  }
}
```

Arreglos Multidimensionales

En un arreglo bidimensional, la posición de un elemento en el arreglo está dada dos valores que representan el renglón y la columna del elemento en la tabla. En un arreglo tridimensional, la posición de un elemento en el arreglo está dada por tres valores que representan la tabla, el renglón y la columna del elemento en el arreglo, etc.

Referencia a Arreglos Multidimensionales

Las sintaxis para declarar referencias a arreglos en dos, tres, ... dimensiones son las siguientes:

```
tipo nomArreglo[][];
tipo nomArreglo[][][];
...
```

ó

```
tipo[][] nomArreglo;
tipo[][][] nomArreglo;
...
```

tipo es el **tipo base** del arreglo y puede ser es cualquier tipo primitivo o clase. . *nomArreglo* es un identificador, el nombre de la referencia al arreglo.

Por ejemplo:

```
int calPars[][];
double[][][] ventas;
```

Creación de Arreglos Multidimensionales

Para crear arreglos en dos, tres, ... dimensiones se usa las siguientes sintaxis:

```
nomArreglo = new tipo[numFilas][numCols];
nomArreglo = new tipo[numTablas][numFilas][numCols];
...
```

numTablas, numFilas, numCols son expresiones enteras que representan los números de tablas, filas y columnas de los arreglos.

Por ejemplo:

```
calPars = new int[4][50];
ventas = new double[3][4][12];
```

La declaración de referencias a arreglos multidimensionales y el reservado de memoria para ellos mismos pueden combinarse. Por ejemplo la sintaxis para declarar y crear arreglos bidimensionales y tridimensionales es:

```
tipo nomArreglo[][] = new tipo[numFilas][numCols];
tipo nomArreglo[][][] = new tipo[numTablas][numFilas][numCols];
```

ó

```
tipo[][] nomArreglo = new tipo[numFilas][numCols];
tipo[][][] nomArreglo = new tipo[numTablas][numFilas][numCols];
```

Por ejemplo:

```
int calPars[][] = new int[4][50];
double[][][] ventas = new double[3][4][12];
```

Inicialización de Arreglos Multidimensionales

Los arreglos multidimensionales también pueden inicializarse al momento de declararse. Por ejemplo la siguiente declaración

```
int x[][] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
```

declara al arreglo x y lo inicializa a

x

0	1	2
3	4	5
6	7	8

Un ejemplo de la inicialización de un arreglo en tres dimensiones sería:

```
int y[][][] = {{{0, 1}, {2, 3}, {4, 5}},
               {{6, 7}, {8, 9}, {10, 11}}
               {{12, 13}, {14, 15}, {16, 17}}
               {{18, 19}, {20, 21}, {22, 23}}};
```

Acceso a los elementos de un Arreglo Multidimensional

El acceso a los elementos de un arreglo multidimensional es similar al visto para los arreglos en una dimensión, nada más que en lugar de usar un sólo índice utilizaremos índices múltiples, dos índices para un arreglo en dos dimensiones, tres para un arreglo en tres dimensiones, etc. Por ejemplo

```
calPars[4][2] = 9;
```

le asigna al elemento que se encuentra en la quinta fila, tercera columna del arreglo calPars el valor de 9.

```
ventas[2][3][1] = 1252.30
```

le asigna al elemento que se encuentra en el tercer plano, cuarta fila, segunda columna del arreglo ventas el valor de 1252.30.

Por ejemplo el siguiente código nos permite calcular el promedio de las calificaciones de cada alumno y la calificación promedio del grupo:

```
int i, j, suma;
double sumaT, promedioT;
int calPars[][] = new int[50][4];
double promedios[] = new double[50];

...
sumaT = 0;
for(i = 0; i < nAlums; i++)
{
   suma = 0;
   for(j = 0; j < 4; j++) suma += calPars[i][j];
   promedios[i] = (double)suma/4;
   sumaT += promedios[i];
}
promedioT = sumaT/nAlums;
```

En realidad los arreglos multidimensionales son arreglos de arreglos. Un arreglo bidimensional es un arreglo donde cada elemento del arreglo es a su vez un arreglo unidimensional. Un arreglo tridimensional es un arreglo donde cada elemento del arreglo es a su vez un arreglo bidimensional, etc.

Cada uno de los arreglos individuales de un arreglo multidimensional puede ser referenciado independientemente. Por ejemplo, para las declaraciones:

```
int calPars[][] = new int[4][50];
double[][][] ventas = new double[3][4][12];
```

`calPars[i]` es una referencia al i-ésimo arreglo unidimensional del arreglo bidimensional `calPars`, esto es, el i-ésimo renglón de la tabla `calPars`; `ventas[i]` es una referencia al i-ésimo arreglo bidimensional del arreglo tridimensional `ventas`, esto es, la i-ésima tabla del arreglo tridimensional `ventas`; y `ventas[i][j]` es una referencia al i-ésimo, j-ésimo arreglo unidimensional del arreglo tridimensional `ventas`, esto es, al j-ésimo renglón de la i-ésima tabla del arreglo tridimensional `ventas`.

Arreglos Multidimensionales y Métodos

Al igual que con los arreglos unidimensionales, en Java podemos pasarle como parámetro a un método, una referencia a un arreglo multidimensional. La sintaxis de los parámetros que representan las referencias a arreglos en dos y tres dimensiones es, respectivamente:

```
tipo nomParArreglo[][]
tipo nomParArreglo[][][]
```

donde *`tipo`* es el **tipo base** del arreglo y *`nomParArreglo`* es el nombre del parámetro que representa la referencia al arreglo. Note que los corchetes están vacíos.

Al llamar al método, el argumento será la referencia al arreglo, el cual es el nombre del arreglo:

```
nomArreglo
```

También es posible que un método nos regrese una referencia a un arreglo multidimensional. La sintaxis para declarar métodos que regresan referencias a arreglos en dos y tres dimensiones es, respectivamente:

```
tipo[][] nomMetodo() {
   ...
}

tipo[][][] nomMetodo() {
   ...
}
```

donde `tipo` es el **tipo base** del arreglo y `nomMetodo` es el nombre del método que nos regresa la referencia a un arreglo.

Cadenas

Junto con el procesamiento numérico, una de las tareas que más frecuentemente realiza la computadora es el procesamiento de textos. Algunas de las aplicaciones en las que se utiliza el procesamiento de texto son:

- Bases de datos como directorios telefónicos, nóminas de una empresa, expedientes de alumnos de una escuela, etc.
- Procesadores de palabras y programas para publicación
- Compiladores e intérpretes.

Todos los programas anteriores requieren de la capacidad de almacenar y manipular texto, es decir, secuencias de caracteres. A estas secuencias de caracteres se les conoce como **cadenas**. En una computadora sólo se pueden almacenar números y por lo tanto cada uno de los caracteres de una cadena es codificado mediante un número. El esquema de codificación más empleado en las computadoras es el esquema de codificación ASCII (por las siglas en inglés del comité que lo creo: American Standard Code for Information Interchange), el cual codifica cada carácter mediante un número en el rango 0 – 127, por lo que este esquema sólo permite codificar 128 caracteres, lo que es insuficiente para representar todos los caracteres empleados en los diferentes lenguajes empleados en los diferentes países. Debido a esto, Java utiliza para codificar los caracteres un nuevo esquema de codificación llamado Unicode. **Unicode** provee un código único para cada carácter, sin importar la plataforma, programa o lenguaje. Para lograr esto Unicode utiliza dos bytes para codificar cada carácter, lo que le permite representar 65,536 caracteres diferentes. Por compatibilidad con el código ASCII, los primeros 128 códigos de Unicode se utilizan para representar los mismos caracteres que los representados por el código ASCII.

Para el manejo de cadenas, la API de Java nos proporciona varias clases entre ellas las clases `String` y `StringBuffer`.

Clase `String`

La clase `String` representa una cadena de caracteres. Las instancias de la clase `String` son constantes. Su valor no puede cambiarse después de ser creadas.

Todas las literales de tipo cadena en Java se implementan como instancias de la clase String. Por ejemplo, la siguiente literal:

```
"Hola"
```

crea una instancia de tipo `String` con la secuencia de caracteres: ‘H’, ‘o’, ‘l’, ‘a’.

Al igual que con cualquier otro objeto, podemos declarar una referencia a `String` y al mismo tiempo crear un objeto y asociárselo. Por ejemplo:

```
String mensaje = "Hola";
```

El lenguaje Java define el operador (+) para concatenación cadenas y el operador (+=) que combina la concatenación con la asignación. Por ejemplo, siguiendo la declaración anterior, podemos tener la siguiente sentencia:

```
mensaje = mensaje + " Juan";
```

la cual es equivalente a:

```
mensaje += " Juan";
```

Las sentencias anteriores podrían interpretarse como que la cadena mensaje ha cambiado su contenido al agregársele la cadena `" Juan"`. Sin embargo, lo que ha pasado es que al concatenarse las cadenas `mensaje` y `" Juan"`, se crea un nuevo objeto de tipo `String` con la cadena resultante y este nuevo objeto se asigna a la referencia `mensaje`.

El lenguaje Java también define la conversión de objetos a cadenas mediante el método `toString()` definido para la clase `Object` de la cual heredan todas las clases de Java.

La Clase String posee un conjunto muy rico de métodos. La tabla 5.1 muestra algunos de esos métodos:

Tabla 5.1 Algunos Métodos de la Clase `String`

<table>
<tr><td>

`public String(char[] value)`

Construye una cadena con la secuencia de caracteres contenida en el arreglo dado por el parámetro `value`.

Lanza:
`NullPointerException` – Si el arreglo `value` es nulo.

</td></tr>
<tr><td>

`public String(StringBuffer buffer)`

Construye una cadena con la secuencia de caracteres contenida en el objeto de tipo `StringBuffer` dado por el parámetro.

Lanza:
`NullPointerException` – Si `buffer` es nulo.

</td></tr>
<tr><td>

`public char charAt(int index)`

Regresa el carácter cuyo índice está dado por el parámetro `index`. El primer carácter tiene índice 0.

Lanza:
`IndexOutOfBoundsException` – Si el índice es negativo o mayor que la longitud de esta cadena.

</td></tr>
<tr><td>

`public int compareTo(String str)`
`public int compareToIgnoreCase(String str)`

Comparan lexicográficamente esta cadena con la cadena dada por el parámetro `str`. El segundo método no hace diferencia entre minúsculas y mayúsculas. Una cadena es menor lexicográficamente que otra si apareciera listada en un diccionario antes que la otra. Regresa:

- El valor de 0 si esta cadena es igual a la cadena dada por el parámetro.
- Un valor menor que 0 si esta cadena es menor lexicográficamente que la cadena dada por el parámetro.
- Un valor mayor que 0 si esta cadena es mayor lexicográficamente que la cadena dada por el parámetro.

Lanza:
`NullPointerException` – Si la cadena dada por el parámetro es `null`.

</td></tr>
<tr><td>

`public String concat(String` *`str`***`)`**

Concatena la cadena del parámetro `str` al final de esta cadena. Si la longitud de la cadena del parámetro es cero regresa esta cadena. Si no, regresa una nueva cadena con la concatenación de esta cadena y la cadena del parámetro. Regresa una cadena con la concatenación de esta cadena y la cadena del parámetro.

</td></tr>
<tr><td>

`public String contains(String` *`str`***`)`**
`public String contains(StringBuffer` *`str`***`)`**

Regresa `true` verdadero si y sólo sí esta cadena contiene la secuencia de caracteres dada por el parámetro *`str`*.

Lanza:
`NullPointerException` – Si *`str`* es `null`.

</td></tr>
</table>

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

`public boolean contentEquals(String str)` **`public boolean contentEquals(StringBuffer str)`** Regresa `true` si y sólo sí esta cadena representa la misma secuencia de caracteres que el objeto del parámetro *`str`*, `false` en caso contrario. **Lanza:** `NullPointerException` – Si *`str`* es `null`.
`public boolean endsWith(String suffix)` Regresa `true` si y sólo sí esta cadena termina con la misma secuencia de caracteres que la cadena dada por el parámetro `suffix`, `false` en caso contrario. **Lanza:** `NullPointerException` – Si la cadena dada por el parámetro es `null`.
`public boolean equals(Object object)` Regresa `true` si y sólo sí el objeto dado por el parámetro `object` no es `null` y es una cadena que representa la misma secuencia de caracteres que esta cadena, `false` en caso contrario.
`public boolean equalsIgnoreCase(String str)` Regresa `true` si y sólo sí esta cadena es igual a la cadena dada por el parámetro `str`, sin tomar en cuenta las mayúsculas, `false` en caso contrario.
`public int indexOf(int ch)` Regresa el índice de la primera ocurrencia dentro de esta cadena del carácter `ch` dado por el parámetro. Si `ch` no está en esta cadena regresa -1.
`public int indexOf(int ch,` `int fromIndex)` Regresa el índice de la primera ocurrencia dentro de esta cadena del carácter `ch` a partir de la posición dada por `fromIndex`. Si `ch` no está en esta cadena regresa -1.
`public int indexOf(String str)` Regresa el índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1.
`public int indexOf(String str,` `int fromIndex)` Regresa el índice de la primera ocurrencia dentro de esta cadena de la subcadena `str` a partir de la posición dada por `fromIndex`. Si `str` no está en esta cadena regresa -1.
`public int lastIndexOf(int ch)` Regresa el índice de la última ocurrencia dentro de esta cadena del carácter `ch` dado por el parámetro. Si `ch` no está en esta cadena regresa -1. La búsqueda se hace de atrás hacia adelante.
`public int lastIndexOf(int ch,` `int fromIndex)` Regresa el índice de la última ocurrencia dentro de esta cadena del carácter `ch` haciendo una búsqueda de atrás hacia adelante a partir de la posición dada por `fromIndex`. Si `ch` no está en esta cadena regresa -1.
`public int lastIndexOf(String str)` Regresa el índice de la última ocurrencia dentro de esta cadena de la subcadena dada por el parámetro `str`. Si `str` no está en esta cadena regresa -1. La búsqueda se hace de atrás hacia adelante.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

public int lastIndexOf(String str, int fromIndex**)** Regresa el índice de la última ocurrencia dentro de esta cadena de la subcadena `str` haciendo una búsqueda de atrás hacia adelante a partir de la posición dada por `fromIndex`. Si `str` no está en esta cadena regresa -1.
public int length() Regresa la longitud de esta cadena, esto es, el número de caracteres.
public String **replace**(char oldChar, char newChar) Regresa una nueva cadena con el resultado de reemplazar todas las ocurrencias del carácter `oldChar` de esta cadena por el carácter `newChar`.
public String **replace**(**String** target, **String** replacement) public String **replace**(**String** target, **StringBuffer** replacement) public String **replace**(**StringBuffer** target, **String** replacement) public String **replace**(**StringBuffer** target, **StringBuffer** replacement) Regresa una nueva cadena con el resultado de reemplazar todas las ocurrencias de la secuencia de caracteres dada por `target` de esta cadena por la secuencia de caracteres dada por `replacement`.
public boolean startsWith(String prefix) Regresa `true` si y sólo sí esta cadena empieza con la misma secuencia de caracteres que la cadena dada por el parámetro `prefix`, `false` en caso contrario. **Lanza:** `NullPointerException` – Si la cadena dada por el parámetro es `null`.
public String substring(int beginIndex) Regresa una nueva cadena que es un subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro `beginIndex` y se extiende hasta el final de esta cadena. **Lanza:** `IndexOutOfBoundsException` – Si `beginIndex` es negativo o mayor que la longitud de esta cadena.
public String substring(int beginIndex, int endIndex) Regresa una nueva cadena que es un subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro `beginIndex` y se extiende hasta el carácter en el índice `endIndex` **-** 1. **Lanza:** `IndexOutOfBoundsException` – Si `beginIndex` es negativo o **end**`Index` es mayor que la longitud de esta cadena o `beginIndex` es mayor que **end**`Index`.
public char[] toCharArray() Convierte esta cadena a un arreglo de caracteres. Regresa un arreglo cuya longitud es el tamaño de esta cadena y cuyo contenido se inicializa a la secuencia de caracteres de esta cadena.
public String toLowerCase() Convierte los caracteres de esta cadena a minúsculas. Regresa una nueva cadena con los caracteres de esta cadena convertidos a minúsculas.

Tabla 5.1 Algunos Métodos de la Clase `String`. Continuación.

`public String toUpperCase()` Convierte los caracteres de esta cadena a mayúsculas. Regresa una nueva cadena con los caracteres de esta cadena convertidos a mayúsculas.
`public String trim()` Regresa una copia de esta cadena eliminando los caracteres blancos iniciales y finales. Un carácter blanco es aquel cuyo código Unicode (o ASCII) es menor o igual a 32,

Clase `StringBuffer`

Los objetos de tipo `StringBuffer` son cadenas como las del tipo `String` pero su contenido puede modificarse invocando una serie de métodos.

Tabla 5.2 Algunos Métodos de la Clase `StringBuffer`

`public StringBuffer()` Construye una instancia de `StringBuffer` vacío y con una capacidad inicial de 16 caracteres.
`public StringBuffer(int length)` Construye una instancia de `StringBuffer` vacío y con una capacidad inicial dada por el parámetro `length`. **Lanza:** `NegativeArraySizeException` – Si el parámetro es negativo.
`public StringBuffer(String str)` Construye una instancia de `StringBuffer` con la secuencia de caracteres contenida en el parámetro `str`. **Lanza:** `NullPointerException` – Si `str` es nulo.
`public StringBuffer append(boolean b)` **`public StringBuffer append(char c)`** **`public StringBuffer append(char [] str)`** **`public StringBuffer append(double)`** **`public StringBuffer append(float f)`** **`public StringBuffer append(int i)`** **`public StringBuffer append(long l)`** **`public StringBuffer append(Object o)`** **`public StringBuffer append(String str)`** **`public StringBuffer append(StringBuffer sb)`** Le agrega a esta cadena, una cadena con la representación del parámetro. Regresa una referencia a esta cadena
`public char charAt(int index)` Regresa el carácter cuyo índice está dado por el parámetro `index`. El primer carácter tiene índice 0. **Lanza:** `IndexOutOfBoundsException` – Si el índice es negativo o mayor que la longitud de esta cadena.

Tabla 5.2 Algunos Métodos de la Clase `StringBuffer`. Continuación

<table>
<tr><td>public StringBuffer delete(int start, int end)

Elimina los caracteres de una subcadena de esta cadena. La subcadena empieza con el carácter en el índice dado por el parámetro start y se extiende hasta el carácter en el índice end - 1. Regresa una referencia a esta cadena.

Lanza:
StringIndexOutOfBoundsException – Si start es negativo, mayor que la longitud de esta cadena o mayor que end.</td></tr>
<tr><td>public StringBuffer deleteCharAt(int index)

Elimina de esta cadena el carácter con índice dado por el parámetro index. Regresa una referencia a esta cadena.

Lanza:
StringIndexOutOfBoundsException – Si index es negativo, mayor o igual que la longitud de esta cadena.</td></tr>
<tr><td>public int indexOf(String str)

Regresa el índice de la primera ocurrencia dentro de esta cadena de la subcadena dada por el parámetro str. Si str no está en esta cadena regresa -1.

Lanza:
NullPointerException – Si la cadena dada por el parámetro es null.</td></tr>
<tr><td>public StringBuffer insert(int offset, boolean b)
public StringBuffer insert(int offset, char c)
public StringBuffer insert(int offset, char[] str)
public StringBuffer insert(int offset, double d)
public StringBuffer insert(int offset, float f)
public StringBuffer insert(int offset, int i)
public StringBuffer insert(int offset, long l)
public StringBuffer insert(int offset, Object obj)
public StringBuffer insert(int offset, String str)

Inserta en esta cadena una subcadena con la representación del segundo parámetro en la posición dada por el parámetro offset. Regresa una referencia a esta cadena.

Lanza:
StringIndexOutOfBoundsException – Si start es inválido.</td></tr>
<tr><td>public int lastIndexOf(String str)

Regresa el índice de la última ocurrencia dentro de esta cadena de la subcadena dada por el parámetro str. Si str no está en esta cadena regresa -1. La búsqueda se hace de atrás hacia adelante.

Lanza:
NullPointerException – Si la cadena dada por el parámetro es null.</td></tr>
</table>

Tabla 5.2 Algunos Métodos de la Clase StringBuffer. Continuación

public int lastIndexOf(String str, int fromIndex**)**

Regresa el índice de la última ocurrencia dentro de esta cadena de la subcadena `str` haciendo una búsqueda de atrás hacia adelante a partir de la posición dada por `fromIndex`. Si `str` no está en esta cadena regresa -1.

Lanza:

`NullPointerException` – Si la cadena dada por el parámetro es `null`.

---

public int length()

Regresa la longitud de esta cadena, esto es, el número de caracteres.

---

public StringBuffer reverse()

Invierte los caracteres de la cadena. Se intercambian los caracteres con índice 0 y `length()` – 1, los caracteres con índice 1 y `length()` – 2, etc. Regresa una referencia a esta cadena.

---

public void setCharAt(int index, char ch)

Reemplaza el carácter de esta cadena con índice dado por el parámetro `index`, por el carácter dado por el parámetro ch.

Lanza:

`IndexOutOfBoundsException` – Si `index` es negativo o mayor o igual que la longitud de esta cadena.

---

public String substring(int start)

Regresa una nueva cadena de tipo `String` con la secuencia de caracteres de esta cadena a partir del carácter con índice `start` y hasta el final de esta cadena.

Lanza:

`StringIndexOutOfBoundsException` – Si `start` es negativo o mayor que la longitud de esta cadena.

---

public String substring(int start, int end)

Regresa una nueva cadena de tipo `String` con la secuencia de caracteres de esta cadena a partir del carácter con índice `start` y hasta el carácter con índice `end` – 1 de esta cadena.

Lanza:

`StringIndexOutOfBoundsException` – Si `start` o `end` son negativos, o mayores que la longitud de esta cadena o `start` es mayor que `end`.

---

public String **toString**()

Regresa una cadena de tipo `String` con la secuencia de caracteres de esta cadena.