10. Algunas clases estándar de Java (I)

Objetivos:
a) Presentar algunas de las clases predefinidas en Java
b) Interpretar el código fuente de una aplicación Java donde aparecen las clases anteriores
c) Construir una aplicación Java sencilla, convenientemente especificada, que emplee estas clases.

Uno de los puntos fuertes de Java es la gran cantidad de clases **predefinidas** que posee. Dichas clases pueden ser utilizadas por los programadores sin necesidad de *reinventar la rueda*. En este capítulo se presentan algunas de las clases predefinidas de Java que se suelen utilizar más en la construcción de programas.

10.1. Clases contenedoras o wrappers

En Java existen una serie de clases predefinidas equivalentes a los tipos primitivos denominadas *wrappers*, clases contenedoras o envoltorios. Como muestra la Tabla 10.1 el identificador de cada una de estas clases es el mismo que el del tipo primitivo correspondiente pero con la letra inicial en mayúsculas (salvo `int` - `Integer` y `char` - `Character`). Cada una de estas clases declaran un conjunto de métodos de gran utilidad.

Tabla 10.1. Tipos primitivos y clases contenedoras correspondientes

Tipo primitivo	Clase contenedora
`boolean`	`Boolean`
`char`	`Character`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

El uso de estas clases puede ser especialmente interesante para realizar determinadas operaciones mediante los métodos que implementan. En la siguiente sección se analiza el uso de la clase `Character`.

10.2. Objetos de la clase `Character`

La clase predefinida `Character` permite trabajar con instancias a las que se les puede asociar un único carácter Unicode. Esta clase incluye un conjunto de métodos (Tabla 10.2) que facilitan la manipulación de datos de tipo primitivo `char`.

Tabla 10.2. Métodos de la clase `Character`

Método	Significado
`boolean isUpperCase(char)` `boolean isLowerCase(char)`	Determina si el carácter es mayúsculas o minúsculas, respectivamente

`char toUpperCase(char)` `char toLowerCase(char)`	Devuelve el carácter en mayúsculas o minúsculas correspondiente.

`boolean isLetter(char)`
`boolean isDigit(char)`
`boolean isLetterOrDigit(char)`
	Determina si el carácter es una letra, un dígito, o una letra o un dígito, respectivamente.

`boolean isWhitespace(char)`	Determina si el carácter es un carácter de espacio en blanco.
`boolean isSpaceChar(char)`	Determina si el carácter es un carácter de espacio en blanco de acuerdo a Unicode.
`boolean isJavaIdentifierStart(char)` `boolean isJavaIdentifierPart(char)`	Determina si el carácter puede ser el primer carácter legál en un identificador o parte de él.

El siguiente código muestra como crear referencias e instancias de la clase `Character`:

```
Character a1;
a1 = new Character('A');
Character a2 = new Character('B');
```

Tabla 10.3. Otros métodos de la clase `Character`

Método	Significado
`int compareTo(Character otroCter)`	Compara dos objetos `Character` con un resultado numérico: el valor es 0 si el parámetro y la instancia son iguales, es menor que 0 si la instancia es menor que el parámetro y es mayor que 0 si la instancia es mayor que el parámetro.

10.3. Objetos de la clase `String`

`String` es una clase predefinida y especial de Java definida en la librería o paquete `java.lang` y orientada a manejar cadenas constantes de caracteres. Una instancia de la clase `String` es inmutable, es decir, una vez que se ha creado y se le ha asignado un valor, éste no puede modificarse (añadiendo, eliminando o cambiando caracteres). El siguiente código muestra diferentes ejemplos de creación de referencias e instancias de la clase `String`:

```
String cortesia = new String("Buenos dias");
// O bien, al ser una clase muy habitual en la forma abreviada:
String saludo = "Hola";
// O tambien:
String despedida;
despedida = "Adios";
```

Como se ha visto anteriormente, las constantes de la clase `String` o cadenas literales se indican entre comillas dobles. Estas comillas no se consideran parte de la cadena.

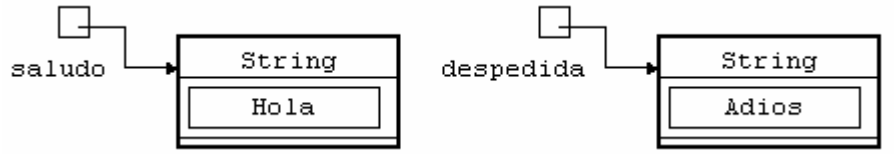

Figura 10.1. Representacion gráfica de los objetos de la clase `String` en memoria

10.4. Operaciones con instancias de la clase `String`

Al ser un objeto, una instancia de la clase `String` no sigue las normas de manipulación de los datos de tipo primitivo con excepción del operador concatenación. El operador `+` realiza una concatenación cuando, al menos, un operando es un `String`. El otro operando puede ser de un tipo primitivo. El resultado es una nueva instancia de tipo `String`. Por ejemplo:

```
"casa" + "blanca"  // Genera "casablanca"
"capitulo" + 5     // Genera "indice5"
5 + "capitulo"     // Genera "5capitulo"
"x" + 2 + 3        // Genera "x23"
2 + 3 + "x"        // Genera "5x": cuidado con la prioridad de los operadores
2 + (3 + "x")      // Genera "23x"
```

Tambien puede emplearse el operador `+=`, de forma que la sentencia `a+=b` es equivalente a la sentencia `a = a+b`.

La comparación de dos objetos `String` no se realiza con el operador igualdad (`==`), ya que se compararían las referencias, sino que se realiza con el método `equals`, de forma que `cadena1.equals(cadena2)` devuelve `true` si `cadena1` y `cadena2` hacen referencia a un mismo valor. Los métodos más importantes de la clase `String` se resumen en la Tabla 10.4.

Tabla 10.4. Métodos de la clase `String`

Método	Significado
`length()`	Devuelve la longitud de la cadena
`indexOf('caracter')`	Devuelve la posición de la primera aparición de caracter
`lastIndexOf('caracter')`	Devuelve la posición de la última aparición de caracter
`charAt(n)`	Devuelve el carácter que está en la posición `n`
`substring(n1,n2)`	Devuelve la subcadena comprendida entre las posiciones `n1` y `n2` ambas incluidas
`toUpperCase()`	Devuelve la cadena convertida a maýusculas
`toLowerCase()`	Devuelve la cadena convertida a minúsculas
`equals("cad")`	Compara dos cadenas y devuelve `true` si son iguales
`equalsIgnoreCase("cad")`	Igual que `equals` pero sin considerar mayúsculas y minúsculas
`valueOf(n)`	Convierte el valor entero n a cadena. Existen otros métodos con el mismo identificador, `valueOf`, para la conversión del resto de los tipos primitivos a cadena.

Para visualizar por pantalla el contenido de un objeto `String` pueden emplearse los métodos `print` y `println` de la clase estándar `System.out`.

```
if (saludo.equals(despedida)) System.out.println(saludo);
   else System.out.println(despedida);
```

La longitud de una cadena puede obtenerse con el método `length` perteneciente a la clase `String` que devuelve un valor entero que indica el número de caracteres que componen la cadena:

```
String despedida = "Adios";
int longitud = despedida.length();       // longitud toma el valor 5
```

```
longitud = "Hasta luego".length();       // longitud toma el valor 11
```

Una cadena de longitud igual a `0`, que no contiene ningún carácter, se denomina cadena vacía y se representa como `""`. Por otro lado, el método `charAt` devuelve el carácter cuya posición indica el parámetro de la llamada, teniendo en cuenta que las posiciones se indican con valores enteros y que el `0` corresponde al primer carácter de la cadena.

```
char c = despedida.charAt(2);             // c toma el valor 'i'
```

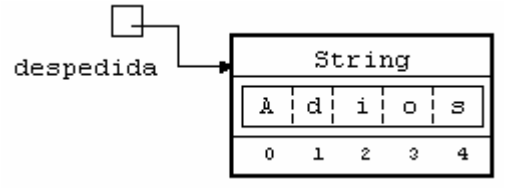

Figura 10.2. Posiciones correspondientes a los caracteres que componen una cadena

En el siguiente programa se muestra un pequeño ejemplo completo de código:

```
/**
 *  Cadena: Ejemplo de uso de string
 *  A. Garcia-Beltran, noviembre de 2007
 */

public class Cadena {
  public static void main (String [] args) {
     String saludo = "Hola";
     String despedida;
     despedida = "Adios";
     String cortesia = saludo + " y " + despedida;
     System.out.print(cortesia + " tiene ");
     System.out.println(cortesia.length() + " caracteres");
     for (int i=cortesia.length()-1 ; i>=0; i--)
         System.out.print(cortesia.charAt(i));
     }
}
```

La salida por pantalla en la ejecución es la siguiente:

```
$>java Cadena↵

Hola y Adios tiene 12 caracteres
soidA y aloH
```

10.5. Otros métodos para trabajar con objetos de la clase `String`

El método `substring`, con los parámetros enteros *`inicio`* y *`fin`*, devuelve una nueva subcadena contenida en la de origen. Si sólo se le indica el primer parámetro, devuelve la cadena a partir del carácter indicado hasta el final de la cadena original. Por ejemplo:

```
String despedida = "Adios";
String s1 = despedida.substring(1,3);      // s1 toma el valor "di"
String s2 = despedida.substring(1);        // s2 toma el valor "dios"
```

El método `toString` permite convertir cualquier valor de un tipo primitivo en un objeto de tipo `String`.

```
String s;
int a = 78;
s = Integer.toString(a);
```

Los métodos `parseInt`, `parseDouble`... pertenecientes a diversas clases predefinidas de Java permiten realizar la tarea inversa, es decir, convertir un objeto de tipo `String` en un dato de tipo primitivo.

```
String s;
s = "156.24";
double t = Double.parseDouble(s);
```

El siguiente programa resume las operaciones básicas que pueden realizarse con objetos de la clase `String`:

```
/**
 *  Cadenas: Ejemplo de uso de la clase String
 *  A. Garcia-Beltran - abril, 2007
 */
public class Cadenas {
  public static void main (String [] args) {
     String saludo = "Hola";
     String despedida;
     despedida = "Adios";
     if (saludo.equals(despedida)) System.out.println(saludo);
        else System.out.println(despedida);
     String cortesia = saludo + " y " + despedida;
     System.out.print(cortesia + " tiene ");
     System.out.println(cortesia.length() + " caracteres");
     for (int i=cortesia.length()-1 ; i>=0; i--)
         System.out.print(cortesia.charAt(i));
     System.out.println();
     System.out.println(cortesia.substring(1,4));
     String s1, s2;
     double x = -14.25;
     s1  = Double.toString(x);
     System.out.println(s1);
     s2 = "156";
     int n = Integer.parseInt(s2);
     System.out.println(n);
  }
}
```

La salida por pantalla en la ejecución es la siguiente:

```
$>java Cadenas↵

Adios
Hola y Adios tiene 12 caracteres
soidA y aloH
ola
-14.25
156
```