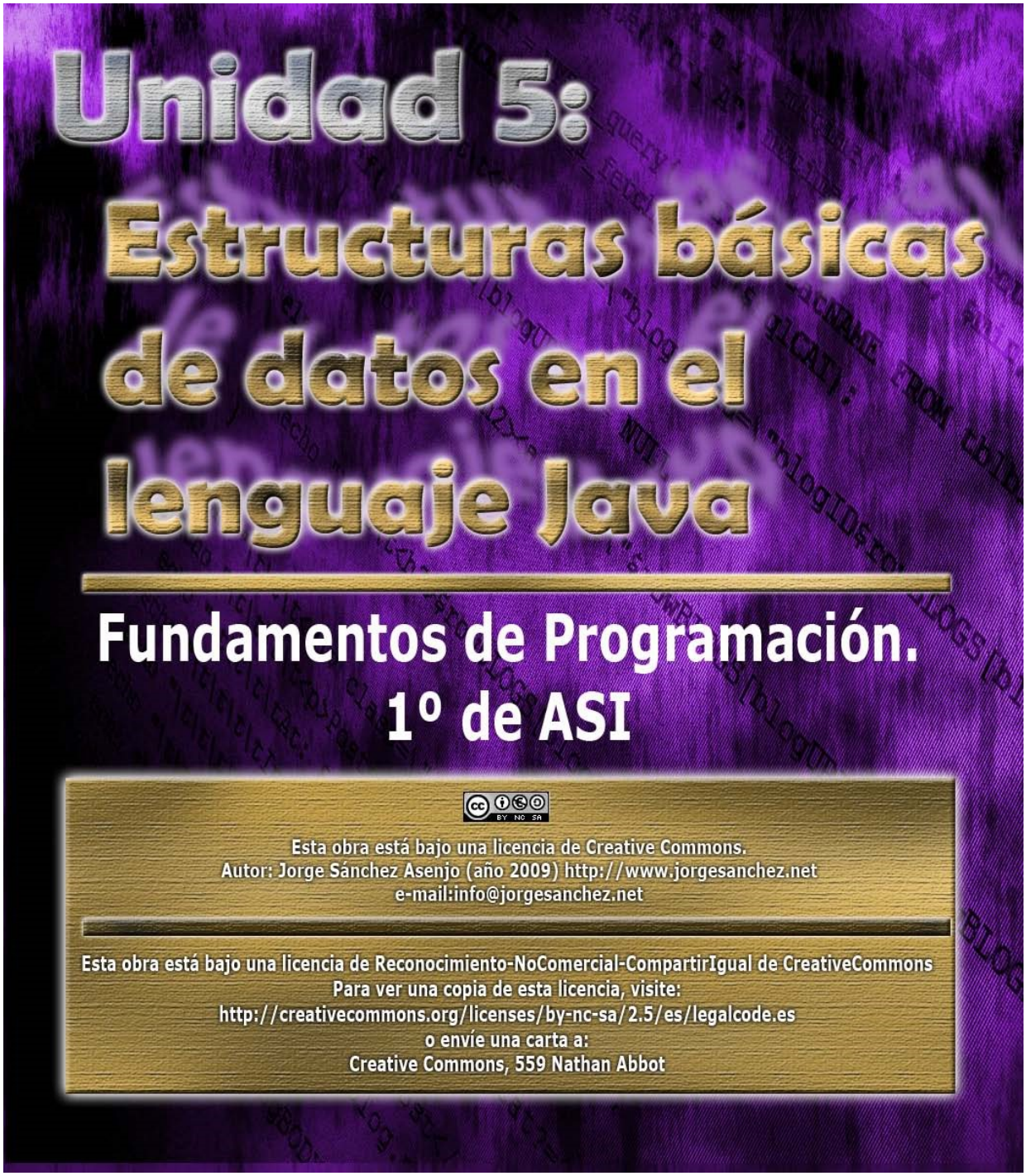
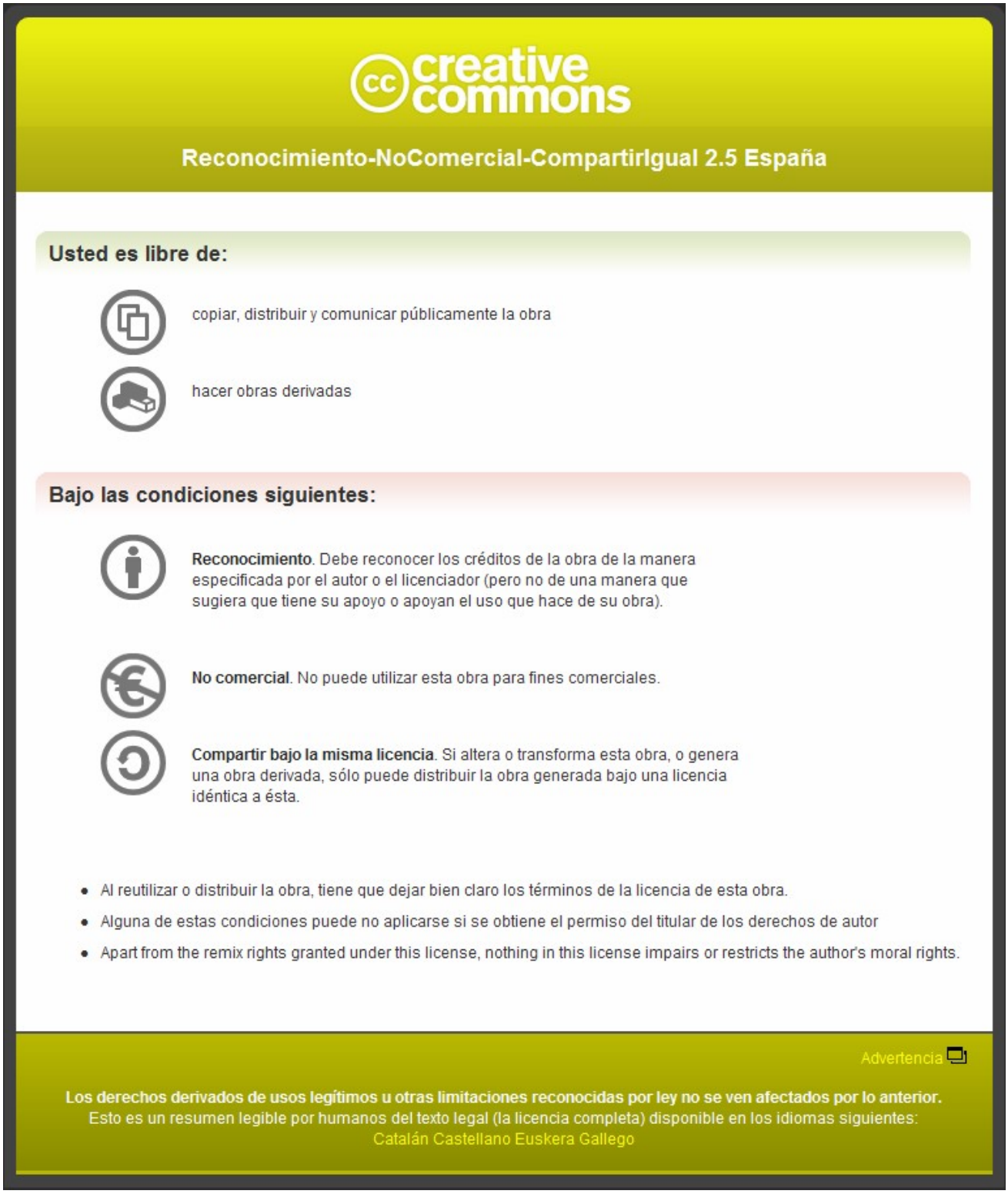


fundamentos de programación
(unidad 4) programación estructurada en Java



Para ver una copia completa de la licencia, acudir a la dirección http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es

---

(3)

(5) estructuras básicas de datos en Java

esquema de la unidad

(5.2) clase String

(5.2.1) introducción. Declarar e iniciar textos

Una de las carencias más grandes que teníamos hasta ahora al programar en Java, era la imposibilidad de almacenar textos en variables. El texto es uno de los tipos de datos más importantes y por ello Java le trata de manera especial.

Para Java las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo **String** (String se suele traducir como cadena; cadena de texto).

Declarar e iniciar un texto se suele hacer de esta forma:

```
String texto1 = "¡Prueba de texto!";
```

Las cadenas pueden ocupar varias líneas utilizando el operador de concatenación "+".

```
String texto2 ="Este es un texto que ocupa " +
                "varias líneas, no obstante se puede "+
                "perfectamente encadenar";
```

Otras formas de También se pueden crear objetos String sin utilizar constantes entrecomilladas, usando otros constructores.

Por ejemplo podemos utilizar un array de caracteres, como por ejemplo:

```
char[] palabra = {'P','a','l','a','b','r','a'};//palabra es un array de caracteres
                                                //no es lo mismo que un String
```

a partir de ese array de caracteres podemos crear un String:

```
String cadena = new String(palabra); //mediante el operador new
                //podemos convertir el array de caracteres en
                //un String
```

De forma similar hay posibilidad de convertir un array de bytes en un String. En este caso se tomará el array de bytes como si contuviera códigos de caracteres, que serán traducidos por su carácter correspondiente al ser convertido a String. Hay que indicar la tabla de códigos que se utilizará (si no se indica se entiende que utilizamos el código ASCII:

```
String codificada = new String (datos, "8859_1");
```

En el último ejemplo la cadena *codificada* se crea desde un array de tipo byte que contiene números que serán interpretados como códigos Unicode. Al asignar, el valor **8859_1** indica la tabla de códigos a utilizar.

(5.2.2) comparación entre objetos String

Los objetos **String** (como ya ocurría con los arrays) no pueden compararse directamente con los operadores de comparación. En su lugar se deben utilizar estas expresiones:

cadena1.**equals**(cadena2). El resultado es **true** si la *cadena1* es igual a la *cadena2*. Ambas cadenas son variables de tipo **String**.

cadena1.**equalsIgnoreCase**(cadena2). Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas. Por cierto en cuestiones de mayúsculas y minúsculas, los hispanohablantes (y el resto de personas del planeta que utilice otro idioma distinto al inglés) podemos utilizar esta función sin temer que no sea capaz de manipular correctamente caracteres como la eñe, las tildes,… Funciona correctamente con cualquier símbolo alfabético de cualquier lengua presente en el código **Unicode**.

s1.**compareTo**(s2). Compara ambas cadenas, considerando el orden alfabético. Si la primera cadena es mayor en orden alfabético que la segunda devuelve *1*, si son iguales devuelve *0* y si es la segunda la mayor devuelve *-1*. Hay que tener en cuenta que el orden no es el del alfabeto español, sino que usa la tabla ASCII, en esa tabla la letra *ñ* es mucho mayor que la *o*.

s1.**compareToIgnoreCase**(s2). Igual que la anterior, sólo que además ignora las mayúsculas (disponible desde Java 1.2)

(5.2.3) String.valueOf

Este método pertenece no sólo a la clase String, sino a otras y siempre es un método que convierte valores de una clase a otra. En el caso de los objetos String, permite convertir valores que no son de cadena a forma de cadena. Ejemplos:

```
String numero = String.valueOf(1234);
String fecha = String.valueOf(new Date());
```

No vale para cualquier tipo de datos, pero sí para casi todos los vistos hasta ahora. No valdría por ejemplo para los arrays.

(5.2.4) métodos de los objetos String

Cada nuevo String que creemos posee, por el simple hecho de ser un String, una serie de métodos que podemos utilizar para facilitar nuestra manipulación de los textos. Para utilizarlos basta con poner el nombre del objeto (de la variable) String, un punto y seguido el nombre del método que deseamos utilizar junto con los parámetros que necesita. Método y sus parámetros después del nombre de la variable String. Es decir:

```
variableString.método(argumentos)
```

length

Permite devolver la longitud de una cadena (el número de caracteres de la cadena):

```
String texto1=”Prueba”;
System.out.println(texto1.length());//Escribe 6
```

concatenar cadenas

Se puede hacer de dos formas, utilizando el método **concat** o con el operador suma (**+**). Desde luego es más sencillo y cómodo utilizar el operador suma. Ejemplo:

```
String s1=”Buenos ”, s2=” días”, s3, s4;
s3 = s1 + s2;              //s3 vale "Buenos días"
s4 = s1.concat(s2);        //s4 vale "Buenos días"
```

charAt

Devuelve un carácter de la cadena. El carácter a devolver se indica por su posición (el primer carácter es la posición 0) Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución (se pararía el programa), una excepción tipo **IndexOutOfBoundsException**. Ejemplo:

```
String s1=”Prueba”;
char c1=s1.charAt(2); //c1 valdrá ‘u’
```

substring

Da como resultado una porción del texto de la cadena. La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final). Si las posiciones indicadas no son válidas ocurre una excepción de tipo **IndexOutOfBoundsException**. Se empieza a contar desde la posición 0. Ejemplo:

```
String s1=”Buenos días”;
String s2=s1.substring(7,10); //s2 = "día"
```

indexOf

Devuelve la primera posición en la que aparece un determinado texto en la cadena. En el caso de que la cadena buscada no se encuentre, devuelve -1. El texto a buscar puede ser **char** o **String**. Ejemplo:

```
String s1=”Quería decirte que quiero que te vayas”;
System.out.println(s1.indexOf(“que”)); //Escribe: 15
```

Se puede buscar desde una determinada posición. En el ejemplo anterior:

```
System.out.println(s1.indexOf("que",16)); //Ahora escribe: 26
```

lastIndexOf

Devuelve la última posición en la que aparece un determinado texto en la cadena. Es casi idéntica a la anterior, sólo que busca desde el final. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";
System.out.println(s1.lastIndexOf("que"); //Escribe: 26
```

También permite comenzar a buscar desde una determinada posición.

endsWith

Devuelve **true** si la cadena termina con un determinado texto. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";
System.out.println(s1.endsWith("vayas"); //Escribe true
```

startsWith

Devuelve **true** si la cadena empieza con un determinado texto.

replace

Cambia todas las apariciones de un carácter por otro en el texto que se indique y lo almacena como resultado. El texto original no se cambia, por lo que hay que asignar el resultado de **replace** en otro String para no perder el texto cambiado:

```
String s1="Mariposa";                          //Da Meripose
System.out.println(s1.replace('a','e'));       //Sigue valiendo
System.out.println(s1);                        Mariposa
```

replaceAll

Modifica en un texto cada entrada de una cadena por otra y devuelve el resultado. El primer parámetro es el texto que se busca (que puede ser una expresión regular), el segundo parámetro es el texto con el que se reemplaza el buscado. La cadena original no se modifica.

```
String s1="Cazar armadillos";
System.out.println(s1.replaceAll("ar","er"));//Escribe: Cazer ermedillos
System.out.println(s1);                      //Sigue valiendo Cazar armadilos
```

toUpperCase

Obtiene la versión en mayúsculas de la cadena. Es capaz de transformar todos los caracteres nacionales:

```
String s1 = "Batallón de cigüeñas";
  System.out.println(s1.toUpperCase());//Escribe: BATALLÓN DE CIGÜEÑAS
System.out.println(s1); //Escribe: Batallón de cigüeñas
```

toLowerCase

Obtiene la versión en minúsculas de la cadena.

toCharArray

Consigue un array de caracteres a partir de una cadena. De esa forma podemos utilizar las características de los arrays para manipular el texto, lo cual puede ser interesante para manipulaciones complicadas.

```
String s="texto de prueba";
char c[]=s.toCharArray();
```

matches

Es una función muy interesante disponible desde la versión 1.4. Examina la expresión regular que recibe como parámetro (en forma de String) y devuelve verdadero si el texto que examina cumple la expresión regular.

Una expresión regular es una expresión textual que utiliza símbolos especiales para hacer búsquedas avanzadas.

Las expresiones regulares pueden contener:

- **Caracteres**. Como *a, s, ñ*,... y les interpreta tal cual. Si una expresión regular contuviera sólo un carácter, **matches** devolvería verdadero si el texto contiene sólo ese carácter. Si se ponen varios, obliga a que el texto tenga exactamente esos caracteres.
- **Caracteres de control** (\n,\\,....)
- **Opciones de caracteres**. Se ponen entre corchetes. Por ejemplo *[abc]* significa *a*, *b* ó *c*.
- **Negación de caracteres**. Funciona al revés impide que aparezcan los caracteres indicados. Se pone con corchetes dentro de los cuales se pone el carácter circunflejo (^). *[^abc]* significa ni *a* ni *b* ni *c*.
- **Rangos**. Se ponen con guiones. Por ejemplo *[a-z]* significa: cualquier carácter de la a *a* la *z*.

- **Intersección**. Usa **&&**. Por ejemplo *[a-x&&r-z]* significa de la *r* a la *x* (intersección de ambas expresiones.
- **Substracción**. Ejemplo *[a-x&&[^cde]]* significa de la a *a* la *x* excepto la *c*, *d* ó *e*.
- **Cualquier carácter. Se hace con el símbolo punto (.)**
- **Opcional**. El símbolo **?** sirve para indicar que la expresión que le antecede puede aparecer una o ninguna veces. Por ejemplo *a?* indica que puede aparecer la letra *a* o no.
- **Repetición**. Se usa con el asterisco (*****). Indica que la expresión puede repetirse varias veces o incluso no aparecer.
- **Repetición obligada**. Lo hace el signo **+**. La expresión se repite una o más veces (pero al menos una).
- **Repetición un número exacto de veces**. Un número entre llaves indica las veces que se repite la expresión. Por ejemplo *\d{7}* significa que el texto tiene que llevar siete números (siete cifras del 0 al 9). Con una coma significa *al menos*, es decir *\d{7,}* significa *al menos* siete veces (podría repetirse más veces). Si aparece un segundo número indica un máximo número de veces *\d{7,10}* significa de siete a diez veces.

lista completa de métodos

método	descripción
char **charAt**(int index)	Proporciona el carácter que está en la posición dada por el entero *index*.
int **compareTo**(String s)	Compara las dos cadenas. Devuelve un valor menor que cero si la cadena *s* es mayor que la original, devuelve *0* si son iguales y devuelve un valor mayor que cero si *s* es menor que la original.
int **compareToIgnoreCase**(String s)	Compara dos cadenas, pero no tiene e cuenta si el texto es mayúsculas o no.
String **concat**(String s)	Añade la cadena *s* a la cadena original.
String **copyValueOf**(char[] data)	Produce un objeto **String** que es igual al array de caracteres *data*.
boolean **endsWith**(String s)	Devuelve **true** si la cadena termina con el texto *s*
boolean **equals**(String s)	Compara ambas cadenas, devuelve **true** si son iguales
boolean **equalsIgnoreCase**(String s)	Compara ambas cadenas sin tener en cuenta las mayúsculas y las minúsculas.
byte[] **getBytes**()	Devuelve un array de bytes que contiene los códigos de cada carácter del String

método	descripción
void **getBytes**(int srcBegin, int srcEnd, char[] dest, int dstBegin);	Almacena el contenido de la cadena en el array de caracteres *dest*. Toma los caracteres desde la posición *srcBegin* hasta la posición *srcEnd* y les copia en el array desde la posición *dstBegin*
int **indexOf**(String s)	Devuelve la posición en la cadena del texto *s*
int **indexOf**(String s, int primeraPos)	Devuelve la posición en la cadena del texto s, empezando a buscar desde la posición *PrimeraPos*
int **lastIndexOf**(String s)	Devuelve la última posición en la cadena del texto s
int **lastIndexOf**(String s, int primeraPos)	Devuelve la última posición en la cadena del texto s, empezando a buscar desde la posición *PrimeraPos*

int **length()**	Devuelve la longitud de la cadena
boolean **matches(**String expReg**)**	Devuelve verdadero si el String cumple la expresión regular
String **replace(**char carAnterior, char ncarNuevo)	Devuelve una cadena idéntica al original pero que ha cambiando los caracteres iguales a *carAnterior* por *carNuevo*
String **replaceFirst(**String str1, String str2**)**	Cambia la primera aparición de la cadena str1 por la cadena str2
String **replaceFirst(**String str1, String str2**)**	Cambia la primera aparición de la cadena uno por la cadena dos
String **replaceAll(**String str1, String str2**)**	Cambia la todas las apariciones de la cadena uno por la cadena dos
String **startsWith(**String s**)**	Devuelve **true** si la cadena comienza con el texto *s*.
String **substring(**int primeraPos, int segundaPos**)**	Devuelve el texto que va desde *primeraPos* a *segunaPos*.
char[] **toCharArray()**	Devuelve un array de caracteres a partir de la cadena dada
String **toLowerCase()**	Convierte la cadena a minúsculas
String **toLowerCase(**Locale local**)**	Lo mismo pero siguiendo las instrucciones del argumento *local*
String **toUpperCase()**	Convierte la cadena a mayúsculas
String **toUpperCase(**Locale local**)**	Lo mismo pero siguiendo las instrucciones del argumento *local*
String **trim()**	Elimina los blancos que tenga la cadena tanto por delante como por detrás
static String **valueOf(***tipo* elemento**)**	Devuelve la cadena que representa el valor *elemento*. Si elemento es booleano, por ejemplo devolvería una cadena con el valor **true** o **false**

(5.3) arrays de Strings

Los arrays se aplican a cualquier tipo de datos y eso incluye a los Strings. Es decir, se pueden crear arrays de textos. El funcionamiento es el mismo, sólo que en su manejo hay que tener en cuenta el carácter especial de los Strings. Ejemplo:

```
String[] texto=new String[4]; texto[0]="Hola"; texto[1]=new String(); texto[2]="Adiós"; texto[3]=texto[0];
for (int i = 0; i < texto.length; i++) {
        System.out.println(texto[i]); }
/*se escribirá: Hola

   Adiós
   Hola
*/
```

(5.4) parámetros de la línea de comandos

Uno de los problemas de trabajar con entornos de desarrollo es que nos abstrae quizá en exceso de la realidad. En esa realidad un programa java se ejecuta gracias a la orden **java programa**; orden que hay que ejecutar en la línea de comandos del sistema.

En cualquier momento podemos hacerlo. Pero es más, podemos añadir parámetros al programa. Por ejemplo supongamos que hemos creado un programa capaz de escribir un texto un número determinado de veces. Pero en lugar de leer el texto y el número por teclado, queremos que lo pasen como parámetro desde el sistema. Eso significa añadir esa información en el sistema operativo, por ejemplo con:

```
java Veces Hola 10
```

Eso podría significar que invocamos al programa llamado *Veces* y le pasamos dos parámetros: el primero es el texto a escribir () y el segundo el número de veces.

El método **main** es el encargado de recoger los parámetros a través de la variable **args**. Esta variable es un array de Strings que recoge cada uno de los parámetros. De tal modo que **args[0]** es un String que recoger el primer parámetro, **args[1]** el segundo, etc. SI no hay parámetros, **args.length** devuelve cero.

(22)

Así el programa **Veces** comentado antes quedaría:

```
public class Veces {
  public static void main(String[] args) {          if(args.length>=2) {
                int tam=Integer.parseInt(args[1]);                  f o r
(int i = 1; i <=tam; i++) {
                        System.out.println(args[0]);
                }               }
        else {
                 System.out.println("Faltan parámetros");
```

```
            }
        }
    }
```

En casi todos los IDE se pueden configurar los parámetros sin necesidad de tener que ejecutar el programa desde la línea de comandos. Por ejemplo en Eclipse se realiza desde el menú **Ejecutar-Configuración de ejecución**: