APÉNDICE

C Algunas rutinas de librerías

Este apéndice enumera algunas de las clases de la librería de Java utilizadas en este libro. Se da una breve descripción de cada clase, así como algunas de las componentes más utilizadas. Una descripción completa de estas clases se puede encontrar en la *JDK API Documentation*.

C.1 Clases del paquete `java.lang`

Éste es el paquete estándar que importan automáticamente todos los programas Java.

C.1.1 `Character`

La clase `Character` oculta en un objeto un valor del tipo primitivo `char`. Un objeto de tipo `Character` tiene un único campo cuyo tipo es `char`. Esta clase también proporciona varios métodos para determinar el tipo de un carácter y la conversión de caracteres entre mayúsculas y minúsculas, y viceversa. Es una clase final.

Métodos importantes

`Character( char value )`

Construye un objeto `Character` y lo inicializa de tal forma que represente el valor primitivo dado como argumento.

`char charValue( )`

Devuelve el valor del objeto `Character`.

`boolean equals( Object obj )`

Compara el objeto `Character` con el objeto especificado. El resultado es `true`, si y sólo si el argumento es distinto de `null` y es un objeto `Character` que representa el mismo valor `char` que el objeto actual.

`String toString( )`

Devuelve un objeto `String` que representa el valor del objeto `Character`. El resultado es un valor de tipo `String` cuya longitud es 1. La única componente del `String` es el valor primitivo `char` que representa el objeto.

static boolean isDigit(char ch)
Devuelve `true` si y sólo si el carácter especificado es un dígito.

static boolean isDefined(char ch)
Devuelve `true` si y sólo si el carácter especificado tiene sentido en Unicode.

static boolean isLetter(char ch)
Devuelve `true` si y sólo si el carácter especificado es una letra.

static boolean isLetterOrDigit(char ch)
Devuelve `true` si y sólo si el carácter especificado es una letra o un dígito.

static boolean isLowerCase(char ch)
Devuelve `true` si y sólo si el carácter especificado es una letra en minúscula.

static boolean isUpperCase(char ch)
Devuelve `true` si y sólo si el carácter especificado es una letra en mayúscula.

static boolean isWhiteSpace(char ch)
Devuelve `true` si y sólo si el carácter especificado es un carácter de espacio en blanco en Java.

static char toLowerCase(char ch)
Devuelve, si existe, el carácter en minúsculas equivalente al carácter dado; si no, devuelve el mismo carácter.

static char toUpperCase(char ch)
Devuelve, si existe, el carácter en mayúsculas equivalente al carácter dado; si no, devuelve el mismo carácter.

static boolean isJavaIdentifierStart(char ch)
Devuelve `true` si y sólo si el carácter especificado puede ser el primer carácter de un identificador Java. (Incorporado en Java 1.1.)

static boolean isJavaIdentifierPart(char ch)
Devuelve `true` si y sólo si el carácter especificado puede ser parte de un identificador Java. (Incorporado en Java 1.1.)

C.1.2 Integer

La clase `Integer` oculta un valor del tipo primitivo `int` en un objeto. Un objeto de tipo `Integer` contiene un único campo cuyo tipo es `int`. Esta clase tiene varios métodos para convertir un valor de tipo `int` en un valor de tipo `String` y viceversa, así como constantes y métodos útiles cuando se trabaja con enteros. Es una clase final.

Constantes importantes

static final const int MAX_VALUE
El mayor valor de tipo `int`.

static final const int MIN_VALUE
El menor valor de tipo `int`.

Métodos importantes

`Integer( int val )`
Construye un nuevo objeto `Integer` que representa el valor de tipo primitivo `int` del argumento.

`int intValue( )`
Devuelve el valor representado por el objeto `Integer` al que se aplica.

`boolean equals( Object obj )`
Devuelve `true`, si y sólo si el argumento es distinto de `null` y es un objeto `Integer` que representa el mismo valor `int` que el objeto.

`static int parseInt( String str )`
Analiza el valor de tipo `String` dado como argumento devolviendo un entero decimal con signo. Puede lanzar una excepción `NumberFormatException`.

`static int parseInt( String str, int radix )`
Analiza el valor de tipo `String` dado como argumento devolviendo un entero con signo en la base especificada como segundo argumento. Puede lanzar una excepción `NumberFormatException`.

`String toString( )`
Devuelve un valor de tipo `String` que representa el valor de tipo `Integer` al que se aplica.

`static String toString( int theInt )`
Devuelve un valor de tipo `String` que representa el entero especificado.

`static String toString( int theInt, int radix )`
Crea un objeto `String` que representa el primer argumento en la base especificada en el segundo argumento.

C.1.3 `Object`

La clase `Object` es la raíz de la jerarquía de clases, es decir, todas las clases tienen a `Object` como superclase. Todos los objetos, incluyendo los vectores, implementan los métodos de esta clase.

Métodos importantes

`Object( )`
Construye un objeto; rara vez se invoca directamente.

`boolean equals ( Object obj )`
Compara dos objetos de tipo `Object`. Este método implementa el test más discriminante posible. Es decir, para cualquier par de referencias `x` e `y`, `x.equals(y)` devuelve `true` si y sólo si `x` e `y` son referencias al mismo objeto (`x == y` vale `true`).

String toString( )

Devuelve una cadena que representa el objeto. El resultado debería ser una representación concisa pero informativa, que sea fácil de leer.

C.1.4 String

La clase `String` representa cadenas de caracteres. Todas las constantes literales de cadenas en programas Java, como `"abc"`, se implementan como instancias de esta clase. Los valores de tipo `String` son constantes, es decir, sus valores no pueden cambiar después de su creación. En la Sección C.1.5 veremos cadenas que pueden cambiar. La clase `String` incluye multitud de métodos, algunos de los cuales describiremos a continuación.

El lenguaje Java proporciona un soporte especial para la operación de concatenación de cadenas (+) y para conversión de otros objetos en cadenas. Las conversiones a cadenas se realizan a través del método `toString`, definido en la clase `Object`, y heredado por todas las clases Java.

Métodos importantes

String( )

Crea una cadena vacía.

String(String anotherStr)

Crea una nueva cadena que contiene la misma secuencia de caracteres que la cadena dada como argumento.

char charAt(int index)

Devuelve el carácter en la posición especificada. El índice varía desde 0 hasta `length( )-1`. Se lanza `StringIndexOutOfBoundsException` si la posición está fuera de rango.

boolean equals(Object obj)

Compara la cadena con el objeto especificado. El resultado es `true`, si y sólo si el argumento es distinto de `null` y es un objeto `String` que representa la misma secuencia de caracteres que el objeto.

int compareTo(String str)

Compara dos cadenas respecto al orden lexicográfico. La comparación se basa en el valor Unicode de cada carácter en las cadenas. Devuelve el valor 0 si las cadenas son iguales; un valor menor que 0 si la cadena que recibe el mensaje es menor lexicográficamente que la cadena argumento; y un valor mayor que 0, en otro caso.

int length( )

Devuelve la longitud de la cadena.

boolean startsWith(String prefix)

Devuelve `true`, si y sólo si la cadena comienza con el prefijo especificado como argumento.

`boolean endsWith( String prefix )`
Devuelve `true`, si y sólo si la cadena termina con el sufijo especificado como argumento.

`String subsString( int beginIdx, int endIdx )`
Devuelve una nueva cadena que es subcadena de aquello sobre lo que se aplica. La subcadena empieza en el valor especificado `beginIdx` y termina en la posición `endIdx-1`. Se lanza `StringIndexOutOfBoundsException` si `beginIdx` o `endIdx` están fuera de rango.

C.1.5 StringBuffer

Ya que las cadenas no se pueden modificar, pequeños cambios en una cadena pueden resultar costosos. Concretamente, el operador `+=` es muy ineficiente. Por ejemplo, `str += 'A'` se implementa como `str = str + 'A'`. Esto implica que se crea una nueva cadena cuyo valor es el resultado de `str + 'A'`, y `str` referencia a esta nueva cadena. Como resultado, el coste de la concatenación de un solo carácter no es constante (como parece lógico que sea). Por el contrario, es proporcional a la longitud de la cadena sobre la que se trabaja. Si hay concatenaciones repetidas a la misma cadena, el coste puede ser prohibitivo.

Un ejemplo es el método trivial de la Figura C.1 que genera una cadena con N caracteres A. Añadir el i-ésimo carácter requiere un coste proporcional a i, ya que, como hemos visto, se crea una nueva cadena de longitud i. El coste total del método viene dado por $1 + 2 + 3 + ... + N$, lo cual es cuadrático. Como caso extremo, para ejecutar este método con $N = 64.000$ en un Pentium 100 se necesitan unos 5 minutos.

Para que las operaciones sean eficientes, los caracteres se deberían manipular directamente, construyendo finalmente una sola cadena conteniendo el resultado deseado, tras completar las operaciones. Esto es lo que hace la clase `StringBuffer`.

Un valor del tipo `StringBuffer` puede construirse a partir de una cadena cualquiera o bien representar la cadena vacía. Sobre ellos se pueden aplicar diversas operaciones que manipulan el `StringBuffer`, como `append`, que añade por el final; `setChar`, que cambia un carácter concreto; e `insert`, que añade por el medio desplazando caracteres si ello es necesario. El compilador utiliza los `StringBuffer` para generar código eficiente para las concatenaciones triviales. Para operaciones no triviales, tales como el ejemplo de la Figura C.1, lo tiene que hacer uno mismo. La Figura C.2 muestra cómo se puede utilizar un `StringBuffer` para generar de forma eficiente un `String` de N caracteres A. Este método es lineal y requiere 1,5 segundos para $N = 64.000$.

```
1    // Método cuadrático que genera una cadena con n Aes
2  static String cadenaLargaMala( int n )
3  {
4      String resultado = "";
5      for( int i = 0; i < n; i++ )
6          resultado += 'A';
7      return resultado;
8  }
```

Figura C.1 Método ineficiente para generar un `String` de N caracteres A.

```
1    // Método lineal que genera una cadena con n Aes
2  static String cadenaLargaBuena( int n )
3  {
4      StringBuffer resultado = new StringBuffer( );
5
6      for( int i = 0; i < n; i++ )
7          resultado.append( 'A' );
8
9      return new String( resultado );
10 }
```

Figura C.2 Método eficiente para generar un `String` de *N* caracteres *A*.

`StringBuffer` utiliza duplicación de vectores, para así asegurar que su capacidad puede expandirse, si ello es necesario. El tamaño inicial es de 16 caracteres.

Métodos importantes

`StringBuffer( )`

Construye un valor `StringBuffer` sin caracteres y con una capacidad inicial de 16 caracteres.

`StringBuffer( String str )`

Construye un valor `StringBuffer` que representa la misma cadena de caracteres que la cadena dada como argumento. La capacidad inicial es de 16 caracteres más que la longitud del argumento.

`int length( )`

Devuelve la longitud (número de caracteres) del `StringBuffer`.

`char charAt( int index )`

Devuelve el carácter en la posición indicada. El primer carácter se encuentra en la posición 0, el siguiente en la posición 1, y así sucesivamente. Se lanza la excepción `StringIndexOutOfBoundsException` si la posición no es válida.

`void setCharAt( int index, char ch )`

Asigna `ch` al carácter en la posición especificada. Lanza la excepción `StringIndexOutOfBoundsException` si la posición no es válida.

`StringBuffer append( Object obj )`

Añade al buffer la representación como `String` del argumento `Object`. El argumento se convierte en una cadena, y los caracteres de esta cadena se añaden al buffer. Esto es lo que se devuelve, por lo que puede ser usado en una ristra de operaciones. Hay otras operaciones `append` disponibles para todos los tipos primitivos, así como para vectores de caracteres.

`StringBuffer insert( int offset, Object obj )`

Añade al buffer la representación como `String` del argumento `Object`. El segundo argumento se convierte en una cadena, y los caracteres de esta cadena se inserta en la posición indicada. Esto es lo que se devuelve, por lo que puede ser usado en una ristra de operaciones. Hay otras operaciones `insert` disponibles para los

tipos primitivos, así como para vectores de caracteres. Se lanza la excepción `StringIndexOutOfBoundsException` si la posición no es válida.

`StringBuffer reverse( )`
La secuencia de caracteres que contiene el buffer se reemplaza por su inversa. Esto es lo que se devuelve, por lo que puede ser usado en una ristra de operaciones.

`String toString( )`
Convierte el contenido en un valor `String`. Se crea un nuevo objeto de tipo `String` y se inicializa con la secuencia de caracteres representada en ese momento por el buffer. Cambios posteriores del buffer no afectan al contenido de la cadena.

C.1.6 System

La clase `System` contiene varios atributos y métodos útiles. No puede ser instanciada. Entre las facilidades proporcionadas por la clase `System` se encuentran los canales de entrada y salida estándar, el canal de errores, un método `exit`, y muchos métodos de sistema que no se utilizan en este texto.

Constantes importantes

`static final InputStream in`
El canal de entrada estándar. El canal está ya abierto y dispuesto para ser leído. Típicamente, este canal corresponde a la entrada por teclado u otra fuente de entrada especificada por el entorno o el usuario.

`static final PrintStream out`
El canal de salida estándar. El canal está ya abierto y dispuesto para ser leído. Típicamente, este canal corresponde a la salida por pantalla u otro destino de salida especificado por el entorno o el usuario.

`static final PrintStream err`
El canal de salida estándar de errores. El canal está ya abierto y dispuesto para ser leído. Típicamente, este canal corresponde a la salida por pantalla u otro destino de salida especificado por el entorno o el usuario. Por convenio, este canal de salida se utiliza para mostrar mensajes de error u otra información que debería llamar la atención inmediata del usuario incluso si el canal de la salida estándar, esto es, el valor de la variable `out`, se ha direccionado a un fichero u otro destino que no se muestra continuamente.

Métodos importantes

`static long currentTimeMillis( )`
Devuelve la hora actual en milisegundos.

`static String getProperty( String key )`
Devuelve la propiedad del sistema indicada por la clave especificada.

`static void exit( int status )`

Concluye la ejecución del programa. El argumento sirve para indicar el estado de terminación; por convenio, un código distinto de cero indica una terminación anormal. Este método no termina de forma normal, y no puede ser llamado desde un applet.

`static void gc( )`

Ejecuta el recolector de basura. La llamada a este método solicita al sistema de ejecución que invierta esfuerzos en reciclar objetos que ya no pueden ser utilizados, para hacer que la memoria que ocupan vuelva a estar disponible. Cuando el método termina, el sistema ha hecho todo lo posible por liberar el espacio de los objetos no utilizados.

C.1.7 Thread

Una hebra es un flujo de ejecución. En particular, un programa. Una aplicación puede tener simultáneamente diversas hebras en ejecución. Las hebras se necesitan en las animaciones. Este uso de las hebras se discute en las Secciones D.4 y D.5.5. Aquí sólo se enumeran los métodos que están relacionados con aplicaciones de animación o que se han utilizado en otro punto de este libro.

Métodos importantes

`Thread ( Runnable target )`

Crea un nuevo objeto `Thread`. `target` es el objeto cuyo método `run` es ejecutado.

`static void sleep( long milliseconds )`

Hace que se duerma (cese su ejecución) la hebra que está actualmente en ejecución, durante el número especificado de milisegundos. Lanza la excepción `InterruptedException` si otra hebra ya ha interrumpido a ésta. Esta excepción debería ser capturada.

`void start( )`

Hace que comience la ejecución de la hebra, lo que se consigue llamando al método `run` de la hebra. Lanza la excepción `IllegalThreadStateException` si la hebra ya se está ejecutando. Esta excepción no tiene por qué ser capturada.

`void stop( )`

Detiene la ejecución de la hebra. Se puede parar una hebra que todavía no ha empezado a ejecutarse. Si la hebra empieza a ejecutarse más tarde, termina inmediatamente.

`boolean isAlive( )`

Comprueba si la hebra está viva. Una hebra está viva si ha empezado a ejecutarse y todavía no ha terminado. Devuelve `true` si y sólo si la hebra está viva.

`void suspend( )`

Suspende la hebra. Si la hebra está viva, es suspendida y su ejecución no progresa hasta que se reanude explícitamente.

`void resume( )`

Reanuda la ejecución de una hebra suspendida. Si la hebra está viva pero suspendida, la ejecución es reanudada permitiéndose que haga progresos en ella.

C.1.8 Throwable

La clase `Throwable` es la superclase de todos los errores y excepciones en el lenguaje Java. Sólo los objetos que son instancias de esta clase (o de alguna de sus subclases) pueden ser lanzados. De forma similar, sólo esta clase o alguna de sus subclases puede ser el tipo del argumento de la cláusula `catch`. Un objeto `Throwable` contiene una instantánea de la pila de ejecución de su hebra en el momento en que ésta fue creada. También puede contener un mensaje que dé más información sobre el error detectado.

Métodos importantes

`Throwable( )`
Construye un nuevo objeto `Throwable` sin mensaje. La traza de la pila se rellena automáticamente.

`Throwable( String message )`
Construye un nuevo objeto `Throwable` con el mensaje especificado. La traza de la pila se rellena automáticamente.

`String getMessage( )`
Devuelve el mensaje del objeto.

`String getLocalizedMessage( )`
Devuelve una descripción localizada del objeto. Las subclases pueden sobrescribir este método para producir un mensaje específico. Para las subclases que no lo sobrescriban, la implementación por defecto devuelve el mismo resultado que `getMessage`.

`String toString( )`
Devuelve una breve descripción del objeto.

`void printStackTrace( )`
Imprime el objeto y su traza en el canal de errores estándar.

C.2 Clases del paquete `java.io`

Este paquete contiene clases para manipular ficheros y canales de entrada/salida. Las clases estudiadas aquí son nuevas en Java 1.1, excepto la clase `File`.

C.2.1 BufferedReader

Lee texto desde un canal de entrada de caracteres, almacenando los caracteres en un buffer de tal forma que la lectura de caracteres, vectores y líneas sea eficiente. Se puede especificar el tamaño del buffer, o se puede utilizar un tamaño por defecto. En la mayoría de las ocasiones el valor por defecto es suficiente. En general, cada petición de lectura sobre un `Reader` provoca la correspondiente lectura

en el canal de entrada. Por tanto, cuando la operación de lectura pueda ser costosa, es aconsejable envolver el `Reader` con un `BufferedReader`, como un `FileReader` o `InputStreamReader`.

Métodos importantes

`BufferedReader( Reader in )`
Crea un almacén para un secuencia de caracteres de entrada, utilizando el tamaño por defecto.

`int read( )`
Lee un solo carácter. Lanza la excepción `IOException` si se produce un error de E/S. El carácter se devuelve como un entero. Por tanto, un uso típico requerirá una conversión a `char`.

`String readLine( )`
Lee una línea de texto. Se considera que una línea termina con un carácter de fin de línea (`'\n'`), un salto de carro (`'\r'`) o un salto de carro seguido inmediatamente por un fin de línea. Devuelve una cadena que contiene la línea, sin incluir los caracteres de terminación, o `null` si se ha llegado al final del flujo. Lanza la excepción `IOException` si se produce un error de E/S.

`boolean ready( )`
Informa sobre si el canal está preparado para ser leído. Una secuencia de caracteres en un buffer está lista si el buffer no es vacío o si la secuencia de caracteres subyacente está lista. Lanza la excepción `IOException` si se produce un error de E/S.

`void close( )`
Cierra el canal. Lanza la excepción `IOException` si se produce un error de E/S.

C.2.2 `File`

La clase `File` proporciona una abstracción que trata de forma independiente de la máquina la mayoría de las complejidades dependientes de la máquina sobre ficheros y rutas de acceso.

Constante importante

`static final char separatorChar`
El carácter separador de directorios dependiente del sistema. Este carácter separa el directorio del nombre del fichero.

Métodos importantes

`File( String name )`
Crea un objeto `File` que representa el fichero cuyo nombre se ha dado en `name`.

`String getName( )`
Devuelve el nombre del fichero representado por el objeto. El nombre del fichero es todo aquello después del separador de directorios en el nombre completo.

String getPath( )
Devuelve el nombre de la ruta del fichero representado por el objeto.

boolean exists( )
Devuelve `true` si y sólo si el fichero especificado existe.

boolean isDirectory( )
Devuelve `true` si y sólo si el objeto al que se aplica es un directorio.

long length( )
Devuelve la longitud del fichero representado por el objeto.

String [ ] list( )
Devuelve un vector con los nombres de los ficheros en el directorio representado por este fichero. Esta lista no incluye ni el directorio actual ni el directorio padre («.» y «..»).

C.2.3 FileReader

`FileReader` es una clase cuyo uso resulta conveniente para leer ficheros de caracteres. Esta clase extiende `Reader`, pero sus métodos no deberían ser utilizados directamente. Para hacerlo adecuadamente, envuelva el objeto con uno de la clase `BufferedReader`, tanto por eficiencia como para así poder llamar a `readLine`.

Métodos importantes

FileReader(String filename)
Construye un objeto `FileReader` para el nombre de fichero dado. Lanza la excepción `FileNotFoundException` si el fichero no se puede abrir.

C.2.4 InputStreamReader

Un `InputStreamReader` es un puente entre las secuencias de bytes y las secuencias de caracteres: lee bytes y los traduce a caracteres. Su uso más importante es la construcción con `System.ini` como parámetro. Esta clase extiende la clase abstracta `Reader`, pero sus métodos no deberían usarse directamente. Para hacerlo adecuadamente, envuelva el objeto con uno de la clase `BufferedReader`, tanto por eficiencia como para así poder llamar a `readLine`.

Métodos importantes

InputStreamReader(InputStream in)
Crea un objeto `InputStreamReader` a partir del `InputStream` dado. Un `InputStream` típico es `System.ini`.

C.2.5 PushbackReader

`PushbackReader` implementa un lector de secuencias de caracteres que permite devolver caracteres a la secuencia.

Métodos importantes

`PushbackReader( Reader in )`

Crea un nuevo lector con un buffer de retroceso de un carácter.

`PushbackReader( Reader in, int bufferSize )`

Crea un nuevo lector con un buffer de retroceso cuya capacidad viene especificada por el segundo argumento.

`int read( )`

Lee un solo carácter. Devuelve el carácter leído, o -1 si se ha alcanzado el final del fichero. Lanza la excepción `IOException` si ocurre algún error de E/S. El carácter se devuelve como un entero. Este uso típico requiere una conversión de tipos.

`void unread( int ch )`

Devuelve al flujo el carácter `ch`. El parámetro que recibe es un entero. Este uso típico requiere una conversión de tipos. Lanza la excepción `IOException` si el buffer de retroceso está lleno u ocurre algún error de E/S.

`void unread( char [ ] cbuf )`

Devuelve al flujo un vector de caracteres copiandolos al principio del buffer de retroceso. Cuando el método termina, el siguiente carácter a leer tendrá el valor `cbuf[0]`, el carácter después de éste tendrá el valor `cbuf[1]`, y así sucesivamente. Lanza una excepción `IOException` si no hay suficiente espacio en el buffer o si ocurre algún error de E/S.

C.3 Clases del paquete `java.util`

Este paquete define varias clases útiles, incluyendo algunas estructuras de datos. También proporciona soporte para fechas y horas. Se describen a continuación las tres clases que se utilizan en el texto.

C.3.1 `Random`

Se utiliza una instancia de esta clase para generar una secuencia de números pseudoaleatorios. La clase utiliza una semilla de 48 bits, la cual se modifica utilizando una fórmula de congruencia lineal (dada al final de la Sección 9.2). Si se crean con la misma semilla dos instancias de la clase `Random` y se realiza sobre cada una la misma secuencia de operaciones, generarán y devolverán la misma secuencia de números.

Métodos importantes

`Random( )`

Crea un nuevo generador de números aleatorios. La semilla se inicializa con un valor basado en la hora actual.

`Random( long seed )`

Crea un nuevo generador de números aleatorios con la semilla dada.

`int nextInt( )`

Devuelve el próximo entero pseudoaleatorio y uniformemente distribuido de la secuencia del generador.

`long nextLong( )`

Devuelve el siguiente entero `long` pseudoaleatorio y uniformemente distribuido de la secuencia del generador.

`double nextDouble( )`

Devuelve el siguiente valor `double` pseudoaleatorio y uniformemente distribuido con media 0,0 y desviación típica 1,0 de la secuencia del generador.

`double nextGaussian( )`

Devuelve el próximo valor pseudoaleatorio, como valor de tipo `double` distribuido según la distribución gausiana de media 0,0 y desviación típica 1,0, a partir de la secuencia de este generador de números aleatorios.

C.3.2 `StringTokenizer`

La clase `StringTokenizer` permite que una aplicación pueda descomponer una cadena en tokens. Los separadores (los caracteres que separan tokens) pueden especificarse en el momento de la creación o de uno en uno. Una instancia de la clase `StringTokenizer` se comporta de dos posibles formas, dependiendo de si al crearse, se le dio a `returnTokens` el valor `true` o `false`. Si es `false`, los caracteres separadores sirven para separar tokens, y un token es una secuencia máxima de caracteres consecutivos que no sean separadores. Si `returnTokens` es `true`, se considera que los caracteres separadores son tokens. Un token es o bien un carácter separador o una secuencia maximal de caracteres consecutivos que no sean separadores.

Métodos importantes

`StringTokenizer( String str )`

Construye un `StringTokenizer` para la cadena especificada. Utiliza el conjunto de separadores por defecto, el cual es `" \t\n\r"`: el espacio en blanco, el carácter tabulador, el carácter de nueva línea y el carácter de salto de línea.

`StringTokenizer( String str, String delim )`

Construye un `StringTokenizer` para la cadena especificada. Toma como separadores los caracteres en el argumento `delim`.

`StringTokenizer( String str, String delim, boolean returnTokens )`

Construye un `StringTokenizer` para la cadena especificada. Los caracteres en el argumento `delim` son los separadores. Si `returnTokens` es `true`, los caracteres separadores también se devuelven como tokens; cada separador se devuelve como una cadena de longitud uno. Si es `false`, los caracteres separadores se saltan y sirven, únicamente, para separar tokens.

boolean hasMoreTokens( )
Devuelve `true` si y sólo si hay más tokens disponibles en la cadena.

String nextToken( )
Devuelve el siguiente token. Lanza la excepción `NoSuchElementException` si no hay más tokens.

String nextToken(String delim)
Devuelve el siguiente token utilizando `delim` como conjunto de separadores. Este conjunto se mantiene como nuevo conjunto de separadores por defecto tras la llamada. Se lanza la excepción `NoSuchElementException` si no hay más tokens.

int countTokens( )
Devuelve el número de tokens que quedan en la cadena utilizando el conjunto actual de separadores.

C.3.3 Vector

La clase `Vector` implementa vectores de objetos de tamaño ajustable.

Métodos importantes

Vector( )
Construye un vector vacío.

Vector(int initialCapacity)
Construye un vector vacío con la capacidad inicial especificada.

Object elementAt(int index)
Devuelve el elemento en la posición indicada. Lanza la excepción `ArrayIndexOutOfBoundsException` si la posición no es válida.

void setElementAt(Object obj, int index)
Hace que la componente del vector en la posición indicada sea el objeto dado. La componente anterior en esa posición es descartada. La posición debe ser un valor mayor o igual que 0 y menor que el tamaño actual del vector. En otro caso, se lanza la excepción `ArrayIndexOutOfBoundsException`.

void setSize(int newSize)
Establece el tamaño del vector. Si el nuevo tamaño es mayor que el actual, se añaden nuevos elementos `null` al final del vector. Si el nuevo tamaño es menor que el actual, se descartan todas las posiciones desde `newSize`.

int capacity( )
Devuelve la capacidad actual del vector.

int size( )
Devuelve el número de componentes en el vector.

En Internet

TestString.java Contiene la versión en inglés de las Figuras C.1 y C.2 y una rutina `main` que hace las llamadas. Se encuentra en el directorio **AppendixC**.