Tema 11

Expresiones Regulares en Java

Una **expresión regular** es una secuencia de caracteres y símbolos, un patrón, que define un conjunto de cadenas. Las expresiones regulares pueden usarse para comparar (y reemplazar) texto en un archivo, datos suministrados a una aplicación, datos capturados en un formulario, análisis lexicográfico, etc.

Si se usa una expresión para analizar o modificar un texto, se dice que “la expresión regular se aplica al texto”. El patrón definido por una expresión regular se aplica a una cadena de izquierda a derecha. Una vez que un carácter de la cadena se ha usado, no puede reutilizarse. Por ejemplo la expresión regular “aba” se ajusta a la cadena match "ababababa"sólo dos veces (aba_aba__).

Los constructores de una expresión regular incluyen caracteres, clases de caracteres, cuantificadores y metacaracteres.

Caracteres

En una expresión regular se usan caracteres alfanuméricos normales y caracteres especiales. Los caracteres especiales se escriben como secuencias de escape. Los diferentes caracteres que se pueden emplear en una expresión regular se muestran en la tabla 11.1

Tabla 11.1 Caracteres

Caracter	Descripción
Alfa-numérico	Todos los caracteres alfanuméricos coinciden con sí mismos. Por ejemplo, la expresión regular “2 casas” coincide con la cadena “2 casas”.
\\	Coincide con el carácter ‘\’
\n	Coincide con el carácter de salto de línea.
\f	Coincide con el carácter de salto de forma.
\r	Coincide con el carácter retorno de carro.
\t	Coincide con el carácter tabulador
\v	Coincide con el carácter tabulador vertical.
\cX	Coincide con el carácter de control x
\0n	Coincide con el carácter cuyo código es el valor octal 0n (0 <= n <= 7)
\0nn	Coincide con el carácter cuyo código es el valor octal 0nn (0 <= n <= 7)
\0mnn	Coincide con el carácter cuyo código es el valor octal 0mnn (0 <= m <= 3, 0 <= n <= 7)
\xhh	Coincide con el carácter cuyo código es el valor hexadecimal xhh.
\xhhh	Coincide con el carácter cuyo código es el valor hexadecimal xhhh.

Nota: En Java las secuencias de escape se tienen que escribir con dos \\ ya que en sí el carácter \ se representa por la secuencia de escape ‘\\’: ‘\\0’, ‘\\n’, ‘\\f’, etc.

Clases de Caracteres

Las clases de caracteres se utilizan para definir el contenido de un patrón. Esto es, lo que el patrón debe buscar. Las diferentes clases de caracteres que se pueden emplear en una expresión regular se muestran en la tabla 11.2

Tabla 11.2 Clases de Caracteres

Símbolo	Descripción	Ejemplo
[*xyz*]	Coincide con un solo de los caracteres del conjunto de caracteres encerrado entre los corchetes.	[ABc] coincide con A o con B o con c pero no con D o con 7.
[^*xyz*]	Negación. Coincide con un solo de los caracteres que no esté en el conjunto de caracteres encerrado entre los corchetes. Se pueden usar un guión para denotar un rango.	[^AB] coincide con C, F y H pero no con A o B.
[*a-z*]	Coincide con un solo de los caracteres del conjunto de caracteres dado por el rango.	[*a-zA-Z*] coincide con una de las letras minúsculas o mayúsculas.
[^*a-z*]	Coincide con un solo de los caracteres del conjunto de caracteres que no esté dado por el rango.	[^*a-zA-Z*] coincide con un carácter que no sea una letra minúscula o mayúscula.
[a-d[m-p]]	Unión. Coincide con cualquier carácter que este ya sea en el primer o segundo grupo	[a-d[m-p]] coincide con uno de los caracteres a a d o m a p.
[a-z&&[def]]	Intersección. Coincide con cualquier carácter que esté en ambos grupos	[a-z&&[def]] coincide con uno de los caracteres d, e o f.
[a-z&&[^bc]]	Substracción. Coincide con cualquier carácter que esté en el primer grupo pero no en el segundo.	[a-z&&[^bc]] coincide con uno de los caracteres a, d a z.
.	Coincide con cualquier carácter excepto el carácter salto de línea o cualquier carácter UNICODE terminador de línea.	Coincide con a, b, -, etc. pero no con \n
\w	Coincide con cualquier carácter alfanumérico, incluyendo al guión bajo. Es equivalente a [a-zA-Z0-9_].	Coincide con a, b, C, _, etc. pero no con % o &
\W	Coincide con cualquier carácter no alfanumérico. Es equivalente a [^a-zA-Z0-9_].	Coincide con %, #, &, etc. pero no con a, b, C, o _.
\d	Coincide con cualquier dígito. Equivale a [0-9].	Coincide con 3, 4, 2, etc. pero no con N, o p
\D	Coincide con cualquier carácter que no sea un dígito. Equivale a [^0-9].	Coincide con N, p, #, etc, pero no con 3, 4, o 2.
\s	Coincide con cualquier carácter de espacio. Equivale a [\t\r\n\v\f].	
\S	Coincide con cualquier carácter que no sea un carácter de espacio. Equivale a [^\t\r\n\v\f].	

Nota: En Java las secuencias de escape se tienen que escribir con dos \\ ya que en sí el carácter \ se representa por la secuencia de escape '\\w': '\\W', '\\d', '\\D', etc.

Operadores Lógicos

Los operadores lógicos de la tabla 11.3 nos permiten combinar caracteres o clases de caracteres para formar expresiones regulares más complejas.

Tabla 11.3 Operadores Lógicos

Operador	Descripción	Ejemplo
XY	Coincide con la expresión regular X seguida por la expresión regular Y.	aBc coincide con aBc. a-\d coincide con a-0, a-1, a-2, etc. [a-f][^a-f] coincide con un carácter en el rango a-f seguido de un carácter fuera del rango a-f .
X|Y	Coincide con la expresión regular X o con la expresión regular Y.	a|Bc coincide con ac, o con Bc.

Cuantificadores

Los cuantificadores definen cuantas veces aparecerá un elemento en una expresión. Los diferentes cuantificadores que se pueden emplear en una expresión regular se muestran en la tabla 11.4

Tabla 11.4 Cuantificadores

Símbolo	Descripción	Ejemplo
X{n}	Coincide exactamente con n ocurrencias de la expresión regular X.	\d{5} coincide exactamente con 5 dígitos
X{n,}	Coincide con n o más ocurrencias de una expresión regular X.	\s{2,} coincide con al menos a dos caracteres de espacio.
X{m,n}	Coincide con m a n ocurrencias de una expresión regular X.	\d{2,4} coincide con al menos 2 pero no más de cuatro dígitos.
X?	Coincide con cero o una ocurrencias de una expresión regular X. Equivale a {0,1}	a\s?b coincide con "ab" o "a b".
X*	Coincide con cero o más ocurrencias de una expresión regular X. Equivale a {0,}	/ab*c/ coincide con "ac", "abc", "abbc", "abbbc", etc.
X+	Coincide con una o más ocurrencias de una expresión regular X. Equivale a {1,}	ab+c coincide con "abc", a "abbc", a "abbbc", etc.

Metacaracteres de Frontera

Los metacaracteres de frontera nos permiten especificar en qué parte de una cadena buscar por una coincidencia con la expresión regular. Algunos de estos metacaracteres se muestran en la tabla 11.5.

Tabla 11.5. Metacaracteres de Frontera

Patrón	Descripción	Ejemplo
^	Sólo busca coincidencias al inicio de una línea	^hola coincide con "hola" pero no con "dijo hola"
$	Sólo busca coincidencias al final de una línea	$ola coincide con "hola" pero no a "olas"

Tabla 11.5. Metacaracteres de Frontera. Cont.

Patrón	Descripción	Ejemplo
\b	Busca coincidencias en cualquier frontera de palabra (principio o final de una palabra).	Ola\b coincide sólo con la primera "ola" en "hola le dijo a las olas"
\B	Busca coincidencias en cualquier lugar excepto en las frontera de palabra	\Bola coincide sólo con la primera "ola" en "hola le dijo a las olas"
\A	Sólo busca coincidencias al inicio de la secuencia completa	
\z	Sólo busca coincidencias al final de la secuencia completa	
\G	Busca coincidencias al final de de la coincidencia anterior	\Gola coincide dos veces en "olaolas"

Captura de Grupos

La captura de grupos nos permite tratar múltiples caracteres como si fueran una unidad simple. Para capturar un grupo se encierran los caracteres que forman el grupo entre paréntesis, (), tabla 11.6.

Tabla 11.6 Alternado o Agrupado

Símbolo	Descripción	Ejemplo
( )	Agrupa caracteres para crear una cláusula. Pueden ir anidados	(abc)+(def) coincide con una o más ocurrencias de "abc", seguidas de una ocurrencia de "def".

La porción de la cadena que coincide al grupo capturado se almacena para futura referencia. Los grupos capturados se enumeran contando el número de paréntesis izquierdos de izquierda a derecha. Por ejemplo en la expresión ((A)(B(C))) hay cuatro grupos:

1. ((A)(B(C)))
2. (A)
3. (B(C))
4. (C)

Hay un grupo especial llamado grupo 0 que representa la expresión completa, pero que no se incluye en la cuenta de grupos.

Referencias

Para hacer referencia a la porción de la cadena que coincide al grupo capturado dentro de una expresión regular se usa el carácter diagonal invertida '\' seguido del número del grupo capturado. Por ejemplo, la expresión regular (\d\d)\1 nos indica que se va a capturar un grupo formado por dos dígitos contiguos, y que el patrón a ajustar serán esos dos dígitos seguidos de los mismos dos dígitos.

La expresión regular anterior coincide con "1212" pero no con "1234"

Búsquedas Hacia Adelante y Hacia Atrás

Las operaciones de búsqueda hacia adelante nos permiten crear expresiones regulares para determinar si un patrón va seguido o no de otro patrón en una cadena.

Las operaciones de búsqueda hacia atras nos permiten crear expresiones regulares para determinar si un patrón termina o no en otro patrón en una cadena.

En la tabla 11.7 se muestran las diferentes operaciones de búsqueda hacia adelante y hacia atrás. Estas operaciones no crean una referencia numerada para los caracteres que coinciden con la expresión regular dentro de los paréntesis.

Tabla 11.7 Alternado o Agrupado

Símbolo	Descripción	Ejemplo
`(?:X)`	Coincide con la expresión regular X pero no la captura.	`(?:.d){2}` coincide pero no captura a `cdad"`.

`X(?=Y)`	Coincide con la expresión regular X si y sólo sí va seguida de la expresión regular Y. Y no forma parte del patrón al que se va a ajustar la cadena, sólo es una condición para el ajuste. .	`Juan (?=Perez)` coincide con `"Juan"` en `"Juan Perez"` pero no en `"Juan Ramos"` o en `"Juan Lopez".`
		
		`Juan (?=Perez|Lopez)` coincide con `"Juan"` en `"Juan Perez"` o en `"Juan Lopez"` pero no en `"Juan Ramos".`

`x(?!y)`	Coincide con la expresión regular X si y sólo sí no va seguida de la expresión regular Y. Y no forma parte del patrón al que se va a ajustar la cadena, sólo es una condición para el ajuste.	`^\d+ (?!meses)` coincide con `"5"` en `"5 dias"` o en `"5 horas"`, pero no en `"5 meses".`

`X(?<=Y)`	Coincide con la expresión regular X si y sólo sí termina en la expresión regular Y. Y no forma parte del patrón al que se va a ajustar la cadena, sólo es una condición para el ajuste. .	`Juan (?<=Perez)` coincide con `"Juan Perez"` en `"Juan Perez"` pero no en `"Juan Ramos"` o en `"Juan Lopez".`
		
		`Juan (?=Perez|Lopez)` coincide con `"Juan Perez"` en `"Juan Perez"` o en `"Juan Lopez" "` en `"Juan Lopez"` pero no en `"Juan Ramos".`

`x(?<!y)`	Coincide con la expresión regular X si y sólo sí no termina de la expresión regular Y. Y no forma parte del patrón al que se va a ajustar la cadena, sólo es una condición para el ajuste.	`^\d+ (?!meses)` coincide con `"5 dias"` en `"5 dias"` o en `"5 horas"`, en `"5 horas"` pero no en `"5 meses".`

Ejemplos Sobre Expresiones Regulares

1. La siguiente expresión regular define un patrón que se ajusta a una cadena vacía o que contiene sólo caracteres blancos:

```
^\\s*$
```

^ Significa que para que una cadena se ajuste al patrón, debe coincidir desde el inicio de la cadena.
\\s* indica una secuencia de 0, 1 o más caracteres blancos.
$ Significa que para que una cadena se ajuste al patrón, debe coincidir hasta el final de la cadena.

El patrón anterior se ajusta a cadenas como:

```
""
" "
"\n"
"\t"
" \n"
```

Si omitiéramos el metacaracter ^, el patrón \\s*$, aparte de ajustarse a las cadenas anteriores, hallaría coincidencias en cadenas que tuvieran 0 o más caracteres blancos al final como en:

```
"xxx"
"xxx "
"xxx\n"
"xx yy "
```

En el último ejemplo, el patrón coincidiría con los dos últimos caracteres de espacio y no con los intermedios ya que esos no ocurren al final de la cadena. Por otro lado, si en el patrón original omitiéramos el metacaracter $, el patrón ^\\s*, aparte de ajustarse a las cadenas a las que se ajusta el patrón original, hallaría coincidencias en cadenas que tuvieran 0 o más caracteres blancos al inicio como en:

```
"xxx"
" xx "
"\nxxx\n"
"  xx yy"
```

En el último ejemplo, el patrón coincidiría con los dos primeros caracteres de espacio y no con los intermedios ya que esos no ocurren al inicio de la cadena. Por último si omitiéramos los metacaracteres ^ y $, el patrón ^\\s*, hallaría coincidencias en todas las cadenas, tengan espacios o no.

2. La siguiente expresión regular define un patrón formado sólo por 3 letras mayúsculas seguidas de 4 dígitos:

```
^[A-Z]{3}\\d{4}$
```

[A-Z]{3} indica una secuencia de 3 caracteres mayúsculos seguidos.
\\d{4} indica una secuencia de 4 dígitos seguidos.

El patrón anterior se ajusta a las siguientes cadenas:

```
"ABC0123"
"DFR8953".
```

3. La siguiente expresión regular define un patrón para una fecha en el formato dd/mm/aaaa:

```
"^(([0-2]?\\d)|([3][0-1]))\\/(([0]?\\d)|([1][0-2]))\\/(\\d{4})$"
```

`(([0-2]?\\d)|([3][0-1]))` Indica que podemos tener un dígito; un cero, un uno o un dos seguido de un dígito; o un tres seguido de un cero o un uno.
`\\/` Indica una diagonal.
`(([0]?\\d)|([1][0-2]))` Indica un dígito; un cero seguido de un dígito; o un 1 seguido por un cero, un uno o un dos.
`(\\d{4})` Indica cuatro dígitos.

El paquete `java.uti.regex`

La API de Java contiene, en el paquete , una serie de clases que nos permiten determinar si una secuencia de caracteres se ajusta a un patrón definido por una expresión regular. Esas clases se muestran en el diagrama de clases de la figura 11.1.

La Interfaz `MatchResult`

interfaz `MatchResult` declara métodos que nos permiten consultar el resultado de un ajustar una secuencia de caracteres contra una expresión regular. Los métodos de esta interfaz se muestran en la figura 11.8.

Tabla 11.8 Métodos de la Interfaz `MatchResult`

`int` **`start`**`()` Regresa la posición, dentro de la secuencia de caracteres, del primer carácter de una coincidencia. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló.

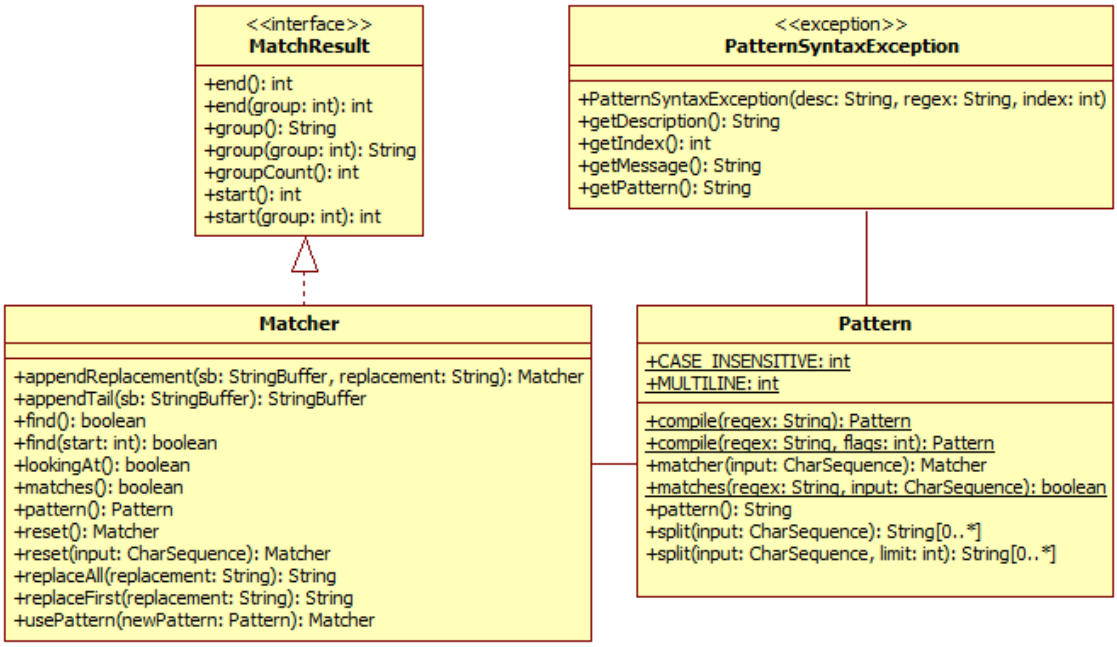

Figura 11.1 Diagrama de Clases de las Clases de Expresiones Regulares

Tabla 11.8 Métodos de la Interfaz `MatchResult`. Cont.

`int` **`start`**`(int group)` Regresa la posición, dentro de la secuencia de caracteres capturada por el grupo del parámetro, del primer carácter de una coincidencia. Si hubo una coincidencia pero no con el grupo mismo, el método regresa -1. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló. `IndexOutOfBoundsException` – Si no hay un grupo capturado en el patrón con el índice del parámetro.
`int` **`end`**`()` Regresa la posición, dentro de la secuencia de caracteres, después del último carácter de una coincidencia. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló.

Tabla 11.8 Métodos de la Interfaz `MatchResult`. Cont.

`int` **`end`**`(int group)` Regresa la posición, dentro de la secuencia de caracteres capturada por el grupo del parámetro, después del último carácter de una coincidencia. Si hubo una coincidencia pero no con el grupo mismo, el método regresa -1. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló. `IndexOutOfBoundsException` – Si no hay un grupo capturado en el patrón con el índice del parámetro.
`String` **`group`**`()` Regresa la subsecuencia de la secuencia de caracteres que se ajusta al patrón. Si el parámetro se ajusta a una cadena vacía, el método regresa una cadena vacía. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló.
`String` **`group`**`(int group)` Regresa la subsecuencia de la secuencia de caracteres capturada por el grupo del parámetro. Si hubo una coincidencia pero no con el grupo mismo, el método regresa `null`. Si el parámetro se ajusta a una cadena vacía, el método regresa una cadena vacía. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló. `IndexOutOfBoundsException` – Si no hay un grupo capturado en el patrón con el índice del parámetro.
`int` **`groupCount`**`()` Regresa el número de grupos en el patrón. No incluye el grupo cero.

La Clase `Matcher`

La clase `Matcher` es la máquina que realiza las operaciones de ajuste de una secuencia de caracteres a un patrón definido por la clase `Pattern`. Un ajustador, que es una instancia de la clase `Matcher` se crean invocando al método `matcher()` de un patrón, que es una instancia de la clase `Pattern`. Algunos de los métodos de la clase `Matcher` y que podemos usar para realizar tres diferentes tipos de operaciones de ajuste, se muestran en la tabla 11.9:

Tabla 11.9 Métodos de la Clase `Matcher`.

`public boolean` **`matches`**`()` Intenta ajustar la secuencia de caracteres completa al patrón. Regresa `true` si y sólo sí la secuencia completa se ajusta al patrón. Si el ajuste tiene éxito, se puede obtener más información usando los métodos: `start()`, `end()`, y `group()`.
`public boolean` **`find`**`()` Intenta encontrar la siguiente subsecuencia de la secuencia de caracteres que se ajusta al patrón. Regresa `true` si y sólo sí hay una subsecuencia de la secuencia de caracteres que se ajusta al patrón. Este método inicia al principio de la secuencia o si la invocación previa tuvo éxito, y el ajustador no fue restaurado, al primer carácter que no coincidió en el ajuste previo. Si el ajuste tiene éxito, se puede obtener más información usando los métodos: `start()`, `end()`, y `group()`.
`public boolean` **`find`**`(int start)` Restablece este ajustador y luego intenta encontrar la siguiente subsecuencia de la secuencia de caracteres que se ajusta al patrón, a partir del índice especificado por el parámetro `start`. Regresa `true` si y sólo sí hay una subsecuencia de la secuencia de caracteres que se ajusta al patrón, a partir del índice especificado. Las siguientes invocaciones del método `find()` iniciarán al primer carácter que no coincidió en este ajuste. Si el ajuste tiene éxito, se puede obtener más información usando los métodos: `start()`, `end()`, y `group()`. Lanza: `IndexOutOfBoundsException` – Si el parámetro `start` < 0 o si es mayor a la longitud de la secuencia de caracteres.
`public boolean` **`lookingAt`**`()` Intenta ajustar la secuencia de caracteres al patrón, a partir del inicio de la secuencia. Regresa `true` si y sólo sí un prefijo de la secuencia se ajusta al patrón. Si el ajuste tiene éxito, se puede obtener más información usando los métodos: `start()`, `end()`, y `group()`.
`public Pattern` **`pattern`**`()` Regresa el patrón que es interpretado por este ajustador.
`public Matcher` **`reset`**`()` Restablece este ajustador. Al restablecerlo se descarta toda su información explícita sobre su estado y se establece la posición de agregado a cero. El método regresa este ajustador.

Tabla 11.9 Métodos de la Clase `Matcher`. Cont.

`public Matcher` **`reset`**`(CharSequence input)` Restablece este ajustador con una nueva secuencia de caracteres. Al restablecerlo se descarta toda su información explícita sobre su estado y se establece la posición de agregado a cero. El método regresa este ajustador.
`public Matcher` **`usePattern`**`(Pattern newPattern)` Cambia el patron usado por este ajustador para encontrar coincidencias al patrón del parámetro. El método regresa este ajustador. Este método hace que el ajustador pierda la información acerca de los grupos del último ajuste. La posición del ajustador en la secuencia de caracteres se mantiene y la última posición de agregado no se ve afectada. Lanza: `IllegalArgumentException` – Si el parámetro es `null`.
`public String` **`replaceAll`**`(String replacement)` Regresa una cadena en la que cada subsecuencia de la secuencia de caracteres que se ajusta al patrón es reemplazada por la cadena del parámetro. Por ejemplo, dada la expresión regular `a*b`, la secuencia de caracteres `"aabfooaabfooabfoob"` y la cadena de reemplazo `"-"`, la invocación a este método regresa la cadena `"-foo-foo-foo-"`.
`public String` **`replaceFirst`**`(String replacement)` Regresa una cadena en la que la primera subsecuencia de la secuencia de caracteres que se ajusta al patrón es reemplazada por la cadena del parámetro. Por ejemplo, dada la expresión regular `can`, la secuencia de caracteres `"zzzcanzzzcanzzz"` y la cadena de reemplazo `"gato"`, la invocación a este método regresa la cadena `"zzzgatozzzcanzzz"`.
`public Matcher` **`appendReplacement`**`(StringBuffer sb, String replacement)` Este método hace las siguientes acciones: 1. Le agrega al `StringBuffer` del parámetro `sb`, la subsecuencia de la secuencia de caracteres desde el punto de agregado hasta antes del primer carácter de la subsecuencia que se ajusta al patrón. 2. Le agrega al `StringBuffer` del parámetro `sb`, la cadena de reemplazo dada por el parámetro `replacement`. 3. Establece el valor del punto de reemplazo a la siguiente posición del último carácter de la subsecuencia que se ajusta al patrón. Lanza: `IllegalStateException` - Si no se ha intentado aún ajustar la secuencia de caracteres al patrón o si no la operación de ajuste previa falló.

Tabla 11.9 Métodos de la Clase `Matcher`. Cont.

`public StringBuffer` **`appendTail`**`(StringBuffer sb)`
 	
Este método lee caracteres de la secuencia de entrada a partir de la posición de agregado y los agrega al `StringBuffer` del parámetro `sb`.
 	
Este método se usa normalmente junto con los métodos `find()` y `appendReplacement()` para hacer una búsqueda y reemplazo. Por ejemplo, el siguiente código le agrega al `StringBuffer sb`, la secuencia `"un gato, dos gatos en el patio"`:
 	
`Pattern p = Pattern.compile("gato");`
`Matcher m = p.matcher("un perro, dos perros en el patio ");`
`StringBuffer sb = new StringBuffer();`
`while (m.find()) {`
`  m.appendReplacement(sb, "dog");`
`}`
`m.appendTail(sb);`

La Clase `Pattern`

La clase `Pattern` es una representación compilada de una expresión regular.

Una cadena con una expresión regular debe compilarse a una instancia de esta clase. El patrón resultante puede crearse para crear un ajustador que pueda ajustar secuencias de caracteres a expresiones regulares.

Algunos de los atributos y métodos de la clase `Pattern` y que podemos usar para realizar tres diferentes tipos de operaciones de ajuste, se muestran en laa tablas 11.10 y 11.11:

Tabla 11.10 Atributos de la Clase `Pattern`.

`public static final int` **`CASE_INSENSITIVE`** Establece que el ajuste se hará sin hacer distinción entre minúsculas y mayúsculas.
`public static final int` **`MULTILINE`** Activa el modo multilínea. En el modo multilínea los metacaracteres ^ y $ se ajustan hasta después o hasta antes de un carácter terminador de línea o el final de la secuencia de entrada, respectivamente. Por omisión los metacaracteres sólo se ajustan al principio y final de la secuencia de caracteres completa.
`public static Pattern` **`compile`**`(String regex)` Compila la expresión regular del parámetro a un patrón Lanza: `PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

Tabla 11.1 Métodos de la Clase `Pattern`.

`public static Pattern` **`compile`**`(String regex, int flags)` Compila la expresión regular del parámetro `regex` a un patrón con las banderas dadas por el parámetro `flags`. Dos de las banderas son: `CASE_INSENSITIVE` y `MULTILINE`. Las banderas pueden combinarse usando el operador or para bits `|`. Lanza: `PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida. `IllegalArgumentException` – Si una de las banderas no es válida.
`public Matcher` **`matcher`**`(CharSequence input)` Crea y regresa un ajustador que ajustará la secuencia de caracteres del parámetro a este patrón.

`public static boolean` **`matches`**`(String regex, CharSequence input)`
 	
Compila la expresión regular del parámetro `regex` a un patrón e intenta ajustarlo a la secuencia de caracteres del parámetro `input`.
 	
Regresa `true` si y sólo sí la secuencia completa se ajusta al patrón.
 	
La invocación a este método produce el mismo resultado que la expresión:
 	
`Pattern.compile(regex).matcher(input).matches()`
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

`public Pattern` **`pattern`**`()` Regresa el patrón que es interpretado por este ajustador.
`public Matcher` **`reset`**`()` Restablece este ajustador. Al restablecerlo se descarta toda su información explícita sobre su estado y se establece la posición de agregado a cero. El método regresa este ajustador.
`public String` **`pattern`**`()` Regresa la expresión regular de la que se compiló este patrón.

Tabla 11.11 Métodos de la Clase `Pattern`. Cont.

`public String[] **split**(CharSequence input)`

Separa la secuencia de caracteres del parámetro en subcadenas tomando las coincidencias con este patrón como separadores. El método regresa un arreglo con las subcadenas obtenidas. Las subcadenas en el arreglo aparecen en el orden en que ocurrieron en la secuencia de caracteres. Si este patrón no se ajusta a ninguna subsecuencia de la secuencia de caracteres, entonces el arreglo sólo tendrá un elemento, la secuencia de caracteres completa como una cadena. Las cadenas vacías al final del arreglo serán descartadas.

Por ejemplo, la secuencia de caracteres `"boo:and:foo"` produce los siguiente resultados con esas expresiones regulares:

Expresión Regular	Contenido del Arreglo
`:`	`{"boo", "and", "foo"}`
`o`	`{"b", "", ":and:f"}`

`public String[] **split**(CharSequence input, int limit)`

Separa la secuencia de caracteres del parámetro en subcadenas tomando las coincidencias con este patrón como separadores. El método regresa un arreglo con las subcadenas obtenidas. Las subcadenas en el arreglo aparecen en el orden en que ocurrieron en la secuencia de caracteres. Si este patrón no se ajusta a ninguna subsecuencia de la secuencia de caracteres, entonces el arreglo sólo tendrá un elemento, la secuencia de caracteres completa como una cadena.

El parámetro `limit` controla el número de veces que se aplica el patrón y por lo tanto afecta al tamaño del arreglo. Si el valor de `limit` < 0, entonces el patrón se aplicará cuando mucho `limit` -1 veces, la longitud del arreglo no será mayor a `limit`, y el último elemento del arreglo contendrá la subsecuencia con todos los caracteres después de la última coincidencia. Si `limit` es negativo, el patrón se aplicará tantas veces como sea posible y el arreglo tendrá esa longitud. Si `limit` es cero, el patrón se aplicará tantas veces como sea posible, el arreglo podrá tener cualquier longitud y las cadenas vacías al final del arreglo serán descartadas.

Por ejemplo, la secuencia de caracteres `"boo:and:foo"` produce los siguiente resultados con esas expresiones regulares:

Expresión Regular	`limit`	Contenido del Arreglo
`:`	2	`{"boo", "and:foo"}`
`:`	5	`{"boo", "and", "foo"}`
`:`	-2	`{"boo", "and", "foo"}`
`o`	5	`{"b", "", ":and:f", "", ""}`
`o`	-2	`{"b", "", ":and:f", "", ""}`
`o`	0	`{"b", "", ":and:f"}`

La Excepción `PatternSyntaxException`

Esta excepción no verificada se utiliza para indicar errores en los patrones de expresiones regulares. Esta excepción hereda de `IllegalArgumentException` que a su vez hereda de RuntimeException. Los métodos adicionales de esta clase se muestran en la tabla 11.12

Tabla 11.12 Excepción `PatternSyntaxException`.

`public` **`PatternSyntaxException`**`(String desc, String regex, int index)` Construye una nueva instancia de esta clase con la descripción del error, la expresión regular con el error y el índice aproximado del error o -1 si no se conoce la posición.
`public int` **`getIndex`**`()` Regresa el índice aproximado del error o -1 si no se conoce la posición.
`public String` **`getDescription`**`()` Regresa la descripción del error.
`public String` **`getPattern`**`()` Regresa la expresión regular con el error.
`public String` **`getMessage`**`()` Regresa una cadena multilínea con la descripción del error de sintaxis y su índice. La expresión regular inválida y una indicación visual del índice del error en la expresión. Este método sobrescribe el método `getMessage()` de la clase `Throwable`.

La clase `String` y Las Expresiones Regulares

La clase `String` de la API de Java tiene un conjunto de métodos que nos permite realizar algunas de las operaciones con expresiones regulares. Esos métodos se muestran en el diagrama de clases de la figura 11.2 y en la tabla 11.13.

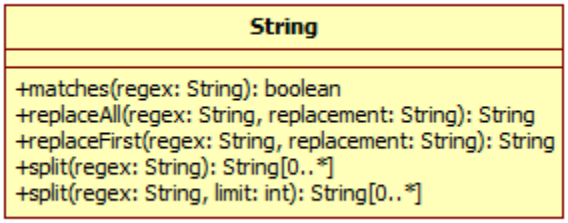

Figura 11.2. Métodos de la Clase `String` para Expresiones Regulares.

Tabla 11.13 Métodos de la Clase `String` para Expresiones Regulares.

`public boolean` **`matches`**`(String regex)`
 	
Regresa `true` si y sólo sí esta cadena se ajusta a la expresión regular de su parámetro.
 	
La invocación a este método produce el mismo resultado que la expresión:
 	
`Pattern.matches(regex, str)`
 	
Donde `str` es la cadena de esta clase.
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

`public String` **`replaceAll`**`(String regex, String replacement)`
 	
Regresa una cadena en la que cada subcadena de esta cadena que se ajusta a la expresión regular del parámetro `regex` es reemplazada por la cadena del parámetro `replacement`.
 	
La invocación a este método produce el mismo resultado que la expresión:
 	
`Pattern.compile(`*`regex`*`).matcher(`*`str`*`).replaceAll(replacement)`
 	
Donde `str` es la cadena de esta clase.
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

`public String` **`replaceFirst`**`(String regex, String replacement)`
 	
Regresa una cadena en la que la primera subcadena de esta cadena que se ajusta a la expresión regular del parámetro `regex` es reemplazada por la cadena del parámetro `replacement`.
 	
La invocación a este método produce el mismo resultado que la expresión:
 	
`Pattern.compile(`*`regex`*`).matcher(`*`str`*`).replaceFirst(replacement)`
 	
Donde `str` es la cadena de esta clase.
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

Tabla 11.13 Métodos de la Clase `String` para Expresiones Regulares.

`public String[] split(String regex)`
 	
Separa la secuencia de caracteres del parámetro en subcadenas tomando las coincidencias con este patrón como separadores. El método regresa un arreglo con las subcadenas obtenidas. Las subcadenas en el arreglo aparecen en el orden en que ocurrieron en la secuencia de caracteres. Si este patrón no se ajusta a ninguna subsecuencia de la secuencia de caracteres, entonces el arreglo sólo tendrá un elemento, la secuencia de caracteres completa como una cadena. Las cadenas vacías al final del arreglo serán descartadas.
 	
La invocación a este método produce el mismo resultado que la expresión:
 	
`Pattern.compile(regex).split(str)`
 	
Donde `str` es la cadena de esta clase.
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

`public String[] split(String regex, int limit)`
 	
Separa la secuencia de caracteres del parámetro en subcadenas tomando las coincidencias con este patrón como separadores. El método regresa un arreglo con las subcadenas obtenidas. Las subcadenas en el arreglo aparecen en el orden en que ocurrieron en la secuencia de caracteres. Si este patrón no se ajusta a ninguna subsecuencia de la secuencia de caracteres, entonces el arreglo sólo tendrá un elemento, la secuencia de caracteres completa como una cadena.
 	
El parámetro `limit` controla el número de veces que se aplica el patrón y por lo tanto afecta al tamaño del arreglo. Si el valor de `limit` < 0, entonces el patrón se aplicará cuando mucho `limit` -1 veces, la longitud del arreglo no será mayor a `limit`, y el último elemento del arreglo contendrá la subsecuencia con todos los caracteres después de la última coincidencia. Si `limit` es negativo, el patrón se aplicará tantas veces como sea posible y el arreglo tendrá esa longitud. Si `limit` es cero, el patrón se aplicará tantas veces como sea posible, el arreglo podrá tener cualquier longitud y las cadenas vacías al final del arreglo serán descartadas.
 	
`Pattern.compile(regex).split(str, limit)`
 	
Donde `str` es la cadena de esta clase.
 	
Lanza:
`PatternSyntaxException` - Si la sintaxis de la expresión regular es inválida.

Ejemplos sobre las Clases para Expresiones Regulares

1. La siguiente clase tiene métodos que nos permiten leer un archivo texto con código Java, guardarlo en una cadena, borrarle los comentarios y escribirlo a otro archivo texto.

BorraComentarios.java

```
/*
 * BorraComentarios.java
 */
package borraComentarios;

import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Esta clase lee un archivo de código Java, le quita los comentarios y
 * escribe el resultado en otro archivo.
 *
 * @author Manuel Domitsu
 */
public class BorraComentarios {
  String nomArchivoEntrada;
  String nomArchivoSalida;

  /**
   * Este constructor inicializa los atributos de la clase e invoca a los
   * metodos que abren el archivo fuente, borran los comentarios y
   * escriben el resultado en otro archivo.
   *
   * @param nomArchivoEntrada Nombre del archivo con el codigo fuente
   * en java
   * @param nomArchivoSalida Nombre del archivo con los resultados
   */
  public BorraComentarios(String nomArchivoEntrada,
                          String nomArchivoSalida) {
    String fuenteCC, fuenteSC;

    this.nomArchivoEntrada = nomArchivoEntrada;
    this.nomArchivoSalida = nomArchivoSalida;

    // Lee el archivo con el codigo fuente
    fuenteCC = leeArchivo();

    // Borra los comentarios
    fuenteSC = borraComentarios(fuenteCC);

    // Escribe los resultados en un archivo texto
    escribeArchivo(fuenteSC);
  }

  /**
   * Este metodo abre el archivo con el codigo fuente, lo lee linea por
   * linea, las guarda en una cadena de tipo String.
   * @return Una cadena con el contenido del archivo de entrada
   */
  public String leeArchivo() {
    FileReader fileReader;
    BufferedReader bufferedReader;
    String linea;
    StringBuffer fuente = new StringBuffer();
```

```
    // Abre el archivo con el codigo fuente
    try {
      fileReader = new FileReader(nomArchivoEntrada);
    } catch (FileNotFoundException fnfe) {
      System.out.println("Error: No existe el archivo "
                         + nomArchivoEntrada);
      return null;
    }

    // Se envuelve el archivo con el codigo fuente con un archivo
    // del tipo BufferedReader para eficientar su acceso
    bufferedReader = new BufferedReader(fileReader);

    try {
      while (true) {
        // Obten la siguiente linea del archivo con el codigo fuente
        linea = bufferedReader.readLine();

        // Si ya no hay lineas, termina el ciclo
        if (linea == null) {
          break;
        }

        // Agrega la línea al Stringbuffer
        fuente.append(linea);
        // Agrega un salto de línea al Stringbuffer
        fuente.append("\n");
      }
      bufferedReader.close();
      fileReader.close();

      return fuente.toString();
    } catch (IOException ioe) {
      System.out.println("Error al procesar el archivo"
                         + nomArchivoEntrada);
      return null;
    }
  }

  /**
   * Este metodo escribe en un archivo texto, el contenido del archivo
   * fuente de Java de entrada sin los comentarios
   * @param fuenteSC Archivo fuente de Java de entrada sin los
   * comentarios
   */
  public void escribeArchivo(String fuenteSC) {
    FileWriter fileWriter;
    BufferedWriter bufferedWriter;

    // Abre el archivo donde se escribiran los resultados
    try {
      fileWriter = new FileWriter(nomArchivoSalida);
    } catch (IOException ioe) {
      System.out.println("Error, no se pudo crear el archivo"
          + nomArchivoSalida);
      return;
```

```
    }

    // Se envuelve el archivo donde se escribiran los resultados con un
    // archivo del tipo BufferedWriter para eficientar su acceso
    bufferedWriter = new BufferedWriter(
                                    (OutputStreamWriter) (fileWriter));

    try {
      bufferedWriter.write(fuenteSC);

      bufferedWriter.close();
      fileWriter.close();
    } catch (IOException ioe) {
      System.out.println("Error al escribir al archivo"
                         + nomArchivoSalida);
    }
  }

  /**
   * Este metodo borra los comentarios de línea, multilínea y de
   * documentación de archivos con código Java.
   * Falla si una cadena literal contiene dos diagonales seguidas
   */
  public String borraComentarios(String fuenteCC) {
    // Crea una expresión regular que elimine los comentarios
    String exReg = "(?:/\\*(?:[^*]|(?:\\*+[^*/]))*\\*+/)|(?://.*$)";

    // Compila la expression regular en un patron
    Pattern p = Pattern.compile(exReg, Pattern.MULTILINE);

    // Crea un ajustador para el texto del archivo
    Matcher m = p.matcher(fuenteCC);

    // Reemplaza los comentarios por una cadena vacia
    return m.replaceAll("");
  }
}
```

Para probar el código de la clase anterior se tiene la siguiente clase de prueba:

PruebaBorraComentarios.java

```
/*
 * PruebaBorraComentarios.java
 */
package borraComentarios;

/**
 * Esta clase sirve para probar la clase BorraComentarios que lee un
 * archivo de código Java, le quita los comentarios y
 * escribe el resultado en otro archivo.
 *
 * @author Manuel Domitsu Kono
```

```
 */
public class PruebaBorraComentarios {

  /**
   * Metodo main(). Invoca a los metodos de la clase BorraComentarios
   * @param args Argumentos en la linea de comando
   */
  public static void main(String[] args) {
    PruebaBorraComentarios pruebaBorraComentarios = new
                                          PruebaBorraComentarios();

    // Crea un objeto del tipo BorraComentarios y le pasa los
    // nombres de los archivos con el codigo fuente y el codigo
    // fuente sin comentarios
    BorraComentarios borraComentarios = new BorraComentarios(
                        "src\\borraComentarios\\BorraComentarios.java",
                        "src\\borraComentarios\\sinCometarios.java");
  }
}
```

2. La siguiente clase tiene métodos que nos permiten leer un archivo texto con código Java, contar el número de veces que ocurre cada palabra reservada y escribirlo los resultados a otro archivo texto.

Estadísticas2.java

```
/*
 * Estadisticas2.java
 */
package estadisticas;

import java.io.*;
import java.util.Map;
import java.util.TreeMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Este programa lee un archivo texto que contiene el codigo fuente de
 * una clase de Java y cuenta el numero de veces que aparecen las
 * palabras reservadas. Los resultados los almacena en otro archivo
 * texto.
 */
public class Estadisticas2 {
  String nomArchivoEntrada;
  String nomArchivoSalida;

  /**
   * Este constructor inicializa los atributos de la clase e invoca a los
```

```
   * metodos que abren el archivo fuente, cuentan las palabras reservadas
   * y escriben el resultado en otro archivo.
   *
   * @param nomArchivoEntrada Nombre del archivo con el codigo fuente en
   * java
   * @param nomArchivoSalida Nombre del archivo con los resultados
   * @param palabras lista de las palabras reservadas a buscar.
   */
  public Estadisticas2(String nomArchivoEntrada, String nomArchivoSalida,
                       String palabras[]) {
    String fuente;
    Map<String, Integer> cuentaPalabras;

    this.nomArchivoEntrada = nomArchivoEntrada;
    this.nomArchivoSalida = nomArchivoSalida;

    // Lee el archivo con el codigo fuente
    fuente = leeArchivo();

    // Cuenta las palabras reservadas
    cuentaPalabras = procesaArchivo(fuente, palabras);

    // Escribe los resultados en un archivo texto
    escribeResultados(cuentaPalabras);
  }

  /**
   * Este metodo abre el archivo con el codigo fuente, lo lee linea por
   * linea, las guarda en una cadena de tipo String.
   *
   * @return Una cadena con el contenido del archivo de entrada
   */
  public String leeArchivo() {
    FileReader fileReader;
    BufferedReader bufferedReader;
    String linea;
    StringBuffer fuente = new StringBuffer();

    // Abre el archivo con el codigo fuente
    try {
      fileReader = new FileReader(nomArchivoEntrada);
    } catch (FileNotFoundException fnfe) {
      System.out.println("Error: No existe el archivo "
                         + nomArchivoEntrada);
      return null;
    }

    // Se envuelve el archivo con el codigo fuente con un archivo del
    // tipo BufferedReader para eficientar su acceso
```

```
    bufferedReader = new BufferedReader(fileReader);

    try {
      while (true) {
        // Obten la siguiente linea del archivo con el codigo fuente
        linea = bufferedReader.readLine();

        // Si ya no hay lineas, termina el ciclo
        if (linea == null) {
          break;
        }

        // Agrega la línea al Stringbuffer
        fuente.append(linea);
        // Agrega un salto de línea al Stringbuffer

        fuente.append("\n");
      }
      bufferedReader.close();
      fileReader.close();

      return fuente.toString();
    } catch (IOException ioe) {
      System.out.println("Error al procesar el archivo"
                         + nomArchivoEntrada);
      return null;
    }
  }

  /**
   * Este metodo escribe en un archivo texto, las ocurrencias de cada
   * palabra reservada en un archivo con codigo fuente de Java.
   *
   * @param cuentaPalabras Mapa con las ocurrencias de cada palabra
   * reservada
   */
  public void escribeResultados(Map<String, Integer> cuentaPalabras) {
    FileWriter fileWriter = null;
    BufferedWriter bufferedWriter;

    // Abre el archivo donde se escribiran los resultados
    try {
      fileWriter = new FileWriter(nomArchivoSalida);
    } catch (IOException ioe) {
      System.out.println("Error, no se pudo crear el archivo"
                         + nomArchivoSalida);
      return;
    }
```

```
    // Se envuelve el archivo donde se escribiran los resultados con un
    // archivo del tipo BufferedWriter para eficientar su acceso
    bufferedWriter = new BufferedWriter(
                                (OutputStreamWriter) (fileWriter));

    try {
      bufferedWriter.write("Frecuencia de las palabras reservadas");
      bufferedWriter.newLine();
      bufferedWriter.write("en la clase: " + nomArchivoEntrada);
      bufferedWriter.newLine();
      bufferedWriter.newLine();

      // Para cada palabra reservada
      for (Map.Entry<String, Integer> palabra :
                                       cuentaPalabras.entrySet()) {
        // escribe en el archivo la palabra reservada y su ocurrencia
        bufferedWriter.write(palabra.getKey() + ": "
                             + palabra.getValue());
        // Escribe en el archivo un salto de linea
        bufferedWriter.newLine();
      }

      bufferedWriter.close();
      fileWriter.close();
    } catch (IOException ioe) {
      System.out.println("Error al cerrar el archivo"
                         + nomArchivoSalida);
    }
  }

  /**
   * Este metodo cuenta las ocurrencias de cada palabra clave y las
   * almacena en un mapa.
   *
   * @param fuente Cadena con el codigo fuente del archivo de entrada
   * @param palabras Arreglo con las palabras reservadas
   * @return Mapa con las ocurrencias de cada palabra reservada
   */
  public Map<String, Integer> procesaArchivo(String fuente,
                                             String palabras[]) {
    Map<String, Integer> cuentaPalabras = new TreeMap();
    String palabra;
    StringBuffer exReg = new StringBuffer();

    // Para cada palabra reservada
    for (int i = 0; i < palabras.length; i++) {
      // Agrégala a la expresión regular
      exReg.append("(?:");
      exReg.append("\\b");
```

```
      exReg.append(palabras[i]);
      exReg.append("\\b");
      exReg.append(")");
      if (i < palabras.length - 1) {
        exReg.append("|");
      }
    }

   // Compila la expression regular en un patron
    Pattern p = Pattern.compile(exReg.toString(), Pattern.MULTILINE);

    // Crea un ajustador para el texto del archivo
    Matcher m = p.matcher(fuente);

    // Mientras haya palabras reservadas en el codigo fuente
    while (m.find()) {
      // Obten la palabra reservada encontrada
      palabra = m.group();

      // Si la palabra reservada es nueva
      if (!cuentaPalabras.containsKey(palabra)) {
        cuentaPalabras.put(palabra, 1);
      } // Si la palabra reservada esta repetida
      else {
        int n = cuentaPalabras.get(palabra);
        cuentaPalabras.put(palabra, n + 1);
      }
    }

    return cuentaPalabras;
  }
}
```

Para probar el código de la clase anterior se tiene la siguiente clase de prueba:

PruebaEstadisticas2.java

```
/*
 * PuebaEstadisticas2.java
 */
package estadisticas;

/**
 * Esta clase sirve para probar la clase Estadisticas2 que lee un archivo
 * texto que contiene el codigo fuente de una clase de Java y cuenta el
 * numero de veces que aparecen las palabras reservadas. Los resultados
 * los almacena en otro archivo texto.
 */
```

```
public class PuebaEstadisticas2 {

  /**
   * Metodo main(). Invoca a los metodos de la clase Estadisticas2
   * @param args Argumentos en la linea de comando
   */
  public static void main(String[] args) {
    PuebaEstadisticas2 puebaEstadisticas = new PuebaEstadisticas2();
    String palabrasReservadas[] = {"if", "switch", "while", "for", "do"};

    // Crea un objeto del tipo Estadisticas2 y le pasa los nombres de los
    // archivos con el codigo fuente, los resultados y la lista de
    // palabras reservadas a buscar
    Estadisticas2 estadisticas =
        new Estadisticas2("src\\estadisticas\\Estadisticas2.java",
                          "estadisticas.txt", palabrasReservadas);
  }
}
```

3. Otra de las aplicaciones de las expresiones regulares es la de validar las entradas de un programa. Estas validaciones son importantes ya que una entrada errónea puede hacer que el programa arroje resultados erróneos. La siguiente clase contiene algunas funciones que utilizan expresiones regulares para probar si la cadena de su parámetro se ajusta a un tipo de entrada particular.

Validadores.java

```
/*
 * Validadores.java
 *
 * Created on 23 de abril de 2012, 03:25 PM
 *
 * @author mdomitsu
 */
package validores;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Esta clase provee de metodos para validar que las cadenas de sus
 * parámetros representan a datos de los tipos especificados por el
 * metodo
 */
public class Validadores {

  /**
   * Valida si la cadena s está vacia.
   *
   * @param s Cadena a verificar
   * @return true si la cadena es vacia o nula, false en caso contrario.
   */
```

```
  public boolean cadenaVacia(String s) {
    CharSequence cadena = s;

    // Define una expresión regular para una cadena vacía
    String reCadena = "^\\s*$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }

  /**
   * Valida si la cadena s no excede la longitud longMax
   *
   * @param longMax Longitud máxima de la cadena
   * @param s Cadena a verificar
   * @return true si la cadena no excede la longitud longMax, false en
   * caso contrario
   */
  public boolean validaCadena(int longMax, String s) {
    CharSequence cadena = s;

    // Define una expresión regular para una cadena de longitud máxima
    // dada por el parámetro longMax
    String reCadena = "^\\w{1," + longMax + "}$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }

  /**
   * Valida una clave. Una clave esta formada de 3 letras mayúsculas
   * y 4 digitos
   *
   * @param s Clave a validar
   * @return true si es una clave valida, false en caso contrario
   */
  public boolean validaClave(String s) {
```

```
    CharSequence cadena = s.trim();

    // Define una expresión regular para una clave
    String reCadena = "^[A-Z]{3}[0-9]{4}$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }

  /**
   * Valida si la cadena s sólo contiene digitos.
   *
   * @param s Cadena a verificar
   * @return true si la cadena representa un entero, false en caso
   * contrario.
   */
  public boolean validaEntero(String s) {
    CharSequence cadena = s.trim();

    // Define una expresión regular para una cadena con puros digitos
    String reCadena = "^\\d+$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }

  /**
   * Valida si la cadena s representa un entero positivo de no más de
   * numDigitos de longitud.
   *
   * @param s Cadena a verificar
   * @return true si la cadena representa un entero, false en caso
   * contrario.
   */
  public boolean validaEnteroMaxDigitos(int numDigitos, String s) {
    CharSequence cadena = s.trim();
```

```
    // Define una expresión regular para una cadena con cuando mucho
    // numDigitos
    String reCadena = "^\\d{1," + numDigitos + "}$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }

  /**
   * Valida un entero positivo en el rango [min, max].
   *
   * @param min Limite inferior del entero
   * @param max Limite superior del entero
   * @param s Cadena a verificar
   * @return true si la cadena representa un entero, false en caso
   * contrario.
   */
  public boolean validaEnteroRango(int min, int max, String s) {
    // valida que solo tenga digitos
    if (!validaEntero(s)) {
      return false;
    }

    // Convierte la cadena a un entero
    int n = Integer.parseInt(s);

    // Si la cadena no está en el rango dado
    if (n < min || n > max) {
      return false;
    }

    return true;
  }

  /**
   * Valida si la cadena s representa una fecha en el formato dd/mm/aaaa.
   *
   * @param s Cadena a verificar
   * @return true si la cadena representa una fecha en el formato
   * dd/mm/aaaa,
   * false en caso contrario.
   */
  public boolean validaFecha(String s) {
    CharSequence cadena = s.trim();

    // Define una expresión regular para una fecha
    String reCadena = "^(([0-2]?\\d)|([3][0-1]))\\/(([0]?\\d)|([1][0-
```

```
2]))\\/(\\d{4})$";

    // Compila la expresión regular a un patrón
    Pattern pattern = Pattern.compile(reCadena);

    // Crea un comparador para comparar la cadena contra el patrón
    Matcher matcher = pattern.matcher(cadena);

    // Si la cadena se ajusta al patrón
    if (matcher.matches()) {
      return true;
    }

    return false;
  }
}
```

Para probar los métodos de validación de la clase anterior se tiene la siguiente clase de pruebas unitarias:

ValidadoresTest.java

```
/*
 * ValidadoresTest.java
 */
package validores;

import org.junit.AfterClass;
import static org.junit.Assert.*;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * Esta es la clase de prueba de la clase Validadores.java
 *
 * @author Manuel Domitsu
 */
public class ValidadoresTest {

  public ValidadoresTest() {
  }

  @BeforeClass
  public static void setUpClass() throws Exception {
  }

  @AfterClass
  public static void tearDownClass() throws Exception {
  }

  /**
   * Prueba unitaria del metodo cadenaVacia de la clase Validadores.
   */
  @Test
  public void testCadenaVacia() {
    Validadores instance = new Validadores();
    boolean result;
```

```
    String s;

    System.out.println("Prueba unitaria del metodo cadenaVacia");

    // Prueba si acepta una cadena vacia
    s = "";
    result = instance.cadenaVacia(s);
    assertTrue(result);

    // Prueba si acepta una cadena con espacios
    s = " ";
    result = instance.cadenaVacia(s);
    assertTrue(result);

    // Prueba si acepta una cadena con caracter tabulador
    s = " \t";
    result = instance.cadenaVacia(s);
    assertTrue(result);

    // Prueba si acepta una cadena con caracter tabulador
    s = " \n";
    result = instance.cadenaVacia(s);
    assertTrue(result);

    // Prueba si rechaza una cadena no vacia
    s = "no\nno";
    result = instance.cadenaVacia(s);
    assertFalse(result);
  }

  /**
   * Prueba unitaria del metodo validaCadena de la clase Validadores.
   */
  @Test
  public void testValidaCadena() {
    Validadores instance = new Validadores();
    boolean result;
    String s;
    int longMax;

    System.out.println("Prueba unitaria del metodo validaCadena");

    // Prueba si acepta una cadena de longitud igual a longMax
    longMax = 5;
    s = "12345";
    result = instance.validaCadena(longMax, s);
    assertTrue(result);

    // Prueba si acepta una cadena de longitud menor a longMax
    longMax = 5;
    s = "123";
    result = instance.validaCadena(longMax, s);
    assertTrue(result);

    // Prueba si rechaza una cadena de longitud mayor a longMax
    longMax = 5;
    s = "1234567";
```

```
    result = instance.validaCadena(longMax, s);
    assertFalse(result);

    // Prueba si rechaza una cadena vacia
    longMax = 5;
    s = "";
    result = instance.validaCadena(longMax, s);
    assertFalse(result);
  }

  /**
   * Prueba unitaria del metodo validaClave de la clase Validadores.
   */
  @Test
  public void testValidaClave() {
    Validadores instance = new Validadores();
    boolean result;
    String s;

    System.out.println("Prueba unitaria del metodo validaClave");

    // Prueba si acepta una clave valida
    s = "ABC0123";
    result = instance.validaClave(s);
    assertTrue(result);

    // Prueba si rechaza una clave con minusculas
    s = "abc0123";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con otros caracteres
    s = "AB+;0123";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave vacia
    s = "";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con espacios
    s = " ";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con solo caracteres
    s = "ABC";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con solo digitos
    s = "0123";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con mas caracteres
```

```
    s = "ABCD0123";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con menos caracteres
    s = "AB0123";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con mas digitos
    s = "ABC01234";
    result = instance.validaClave(s);
    assertFalse(result);

    // Prueba si rechaza una clave con menos digitos
    s = "ABC012";
    result = instance.validaClave(s);
    assertFalse(result);
  }

  /**
   * Prueba unitaria del metodo validaEntero de la clase Validadores.
   */
  @Test
  public void testValidaEntero() {
    Validadores instance = new Validadores();
    boolean result;
    String s;

    System.out.println("Prueba unitaria del metodo validaEntero");

    // Prueba si acepta un entero valido
    s = "1234";
    result = instance.validaEntero(s);
    assertTrue(result);

    // Prueba si rechaza una cadena con letras
    s = "12a3";
    result = instance.validaEntero(s);
    assertFalse(result);


    // Prueba si rechaza una cadena con otros caracteres
    s = "A12+;0";
    result = instance.validaEntero(s);
    assertFalse(result);

    // Prueba si rechaza una cadena vacia
    s = "";
    result = instance.validaEntero(s);
    assertFalse(result);

    // Prueba si rechaza una cadena con espaciosa
    s = " ";
    result = instance.validaEntero(s);
    assertFalse(result);
  }
```

```
  /**
   * Prueba unitaria del metodo validaEnteroMaxDigitos de la clase
Validadores.
   */
  @Test
  public void testValidaEnteroMaxDigitos() {
    Validadores instance = new Validadores();
    boolean result;
    String s;
    int numDigitos;

    System.out.println(
                 "Prueba unitaria del metodo validaEnteroMaxDigitos");

    // Prueba si acepta un entero valido
    numDigitos = 1;
    s = "1";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertTrue(result);

    // Prueba si acepta un entero valido
    numDigitos = 5;
    s = "12345";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertTrue(result);

    // Prueba si acepta un entero valido
    numDigitos = 5;
    s = "1234";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertTrue(result);

    // Prueba si rechaza un entero muy largo
    numDigitos = 5;
    s = "124567";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertFalse(result);

    // Prueba si rechaza un entero con letras
    numDigitos = 5;
    s = "12a3";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertFalse(result);

    // Prueba si rechaza una cadena con otros caracteres
    numDigitos = 5;
    s = "A12+;0";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertFalse(result);

    // Prueba si rechaza una cadena vacia
    numDigitos = 5;
    s = "";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertFalse(result);
```

```
    // Prueba si rechaza una cadena con espaciosa
    numDigitos = 5;
    s = " ";
    result = instance.validaEnteroMaxDigitos(numDigitos, s);
    assertFalse(result);
  }

  /**
   * Prueba unitaria del metodo validaEnteroRango de la clase
   * Validadores.
   */
  @Test
  public void testValidaEnteroRango() {
    Validadores instance = new Validadores();
    boolean result;
    String s;
    int min, max;

    System.out.println("Prueba unitaria del metodo validaEnteroRango");

    min = 10;
    max = 100;

    // Prueba si acepta un entero con rango valido
    s = "50";
    result = instance.validaEnteroRango(min, max, s);
    assertTrue(result);

    // Prueba si acepta un entero con rango valido
    s = "10";
    result = instance.validaEnteroRango(min, max, s);
    assertTrue(result);

    // Prueba si acepta un entero con rango valido
    s = "100";
    result = instance.validaEnteroRango(min, max, s);
    assertTrue(result);

    // Prueba si rechaza un entero con por debajo del rango
    s = "5";
    result = instance.validaEnteroRango(min, max, s);
    assertFalse(result);

    // Prueba si rechaza un entero con por encima del rango
    s = "500";
    result = instance.validaEnteroRango(min, max, s);
    assertFalse(result);
  }

  /**
   * Prueba unitaria del metodo validaFecha de la clase Validadores.
   */
  @Test
  public void testValidaFecha() {
    Validadores instance = new Validadores();
    boolean result;
    String s;
```

```
    System.out.println("Prueba unitaria del metodo validaFecha");

    // Prueba si acepta una fecha valida
    s = "1/1/2001";
    result = instance.validaFecha(s);
    assertTrue(result);

    // Prueba si acepta una fecha valida
    s = "01/01/2010";
    result = instance.validaFecha(s);
    assertTrue(result);

    // Prueba si acepta una fecha valida
    s = "15/12/2001";
    result = instance.validaFecha(s);
    assertTrue(result);

    // Prueba si acepta una fecha valida
    s = "21/08/2000";
    result = instance.validaFecha(s);
    assertTrue(result);

    // Prueba si acepta una fecha valida
    s = "31/1/2001";
    result = instance.validaFecha(s);
    assertTrue(result);

    // Prueba si rechaza una fecha invalida
    s = "11/2001";
    result = instance.validaFecha(s);
    assertFalse(result);

    // Prueba si rechaza una fecha invalida
    s = "51/13/2001";
    result = instance.validaFecha(s);
    assertFalse(result);

    // Prueba si rechaza una fecha valida
    s = "0/0/201";
    result = instance.validaFecha(s);
    assertFalse(result);

    // Prueba si rechaza una fecha valida
    s = "12/ago/2001";
    result = instance.validaFecha(s);
    assertFalse(result);
  }
}
```

El siguiente código ilustra el uso de los validadores anteriores. Es el código parcial de un cuadro de diálogo empleado para capturar los datos de una canción del programa del amante a la música. Cada campo se valida en forma individual cuando el usuario pasa al siguiente campo y al final cuando

el usuario hace clic en el botón Aceptar. Para validar un campo de texto se usa un método oyente del tipo `ActionPerformed` para ese campo de texto.

DlgCancion.java

```
/*
 * DlgCancion.java
 */
package interfazUsuario;

import java.awt.Dimension;
import java.awt.Point;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JOptionPane;
import objetosNegocio.Cancion;
import objetosNegocio.Genero;
import objetosServicio.Fecha;
import validores.Validadores;

/**
 * este cuadro de dialogo captura los datos de una cancion del programa
 * del amante a la musica y los regresa en el parametro cancion
 *
 * @author Manuel Domitsu
 */
public class DlgCancion extends javax.swing.JDialog {
  ...

  /**
   * Método oyente del botón botonAceptar. Valida los campos de
   * entrada. Si no hubo errores llena un objeto de tipo cancion con
   * los valores capturados. En caso contrario despliega un mensaje con
   * los errores detectados y luego cierra el cuadro de dialogo.
   *
   * @param evt Evento al que escucha
   */
  private void botonAceptarActionPerformed(java.awt.event.ActionEvent
                                           evt) {
    StringBuffer msjError = new StringBuffer();
    String titulo;
    String interprete;
    String autor;
    String album;
    String sDuracion;
    String sFecha;
    int posError = 0;

    // Si la operación es Agregar o Actualizar
```

```
    if ((operacion == ConstantesGUI.AGREGAR)
            || (operacion == ConstantesGUI.ACTUALIZAR)) {
      // Obtiene los valores de los campos de texto
      titulo = campoTextoTitulo.getText();
      interprete = campoTextoInterprete.getText();
      autor = campoTextoAutor.getText();
      album = campoTextoAlbum.getText();
      sDuracion = campoTextoDuracion.getText();
      sFecha = campoTextoFecha.getText();

      // Si no hay titulo
      if (validadores.cadenaVacia(titulo)) {
        // Agrega mensaje de error
        msjError.append("Faltó el título. ");
        // Establece la posicion del primer error
        posError = 1;
      } // Si el titulo es muy largo
      else if (!validadores.validaCadena(35, titulo)) {
        // Agrega mensaje de error
        msjError.append("Título muy largo. ");
        // Establece la posicion del primer error
        posError = 1;
      }

      // Si no hay interprete
      if (validadores.cadenaVacia(interprete)) {
        // Agrega mensaje de error
        msjError.append("Faltó el intérprete. ");
        // Establece la posicion del primer error
        if (posError == 0) {
          posError = 2;
        }
      } // Si el interprete es muy largo
      else if (!validadores.validaCadena(35, interprete)) {
        // Agrega mensaje de error
        msjError.append("Intérprete muy largo. ");
        // Establece la posicion del primer error
        if (posError == 0) {
          posError = 2;
        }
      }

      // Si hay autor y es muy largo
      if (!validadores.cadenaVacia(autor)
              && !validadores.validaCadena(35, autor)) {
        // Agrega mensaje de error
        msjError.append("Autor muy largo. ");
        // Establece la posicion del primer error
        if (posError == 0) {
```

```
          posError = 3;
        }
      }

      // Si hay album y es muy largo
      if (!validadores.cadenaVacia(album)
              && !validadores.validaCadena(35, album)) {
        // Agrega mensaje de error
        msjError.append("Album muy largo. ");
        // Establece la posicion del primer error
        if (posError == 0) {
          posError = 4;
        }
      }

      // Si hay duracion y es muy larga
      if (!validadores.cadenaVacia(sDuracion)
              && !validadores.validaEnteroMaxDigitos(3, sDuracion)) {
        // Agrega mensaje de error
        msjError.append("Entero muy largo. ");
        // Establece la posicion del primer error
        if (posError == 0) {
          posError = 5;
        }
      }

      // Si hay fecha y es invalida
      if (!validadores.cadenaVacia(sFecha)
              && !validadores.validaFecha(sFecha)) {
        // Agrega mensaje de error
        msjError.append("Fecha inválida. ");
        // Establece la posicion del primer error
        if (posError == 0) {
          posError = 6;
        }
      }

      // Si hubo al menos un error despliega sus mensajes
      if (posError > 0) {
        JOptionPane.showMessageDialog(this, msjError,
                                      "Error de entrada.",
                JOptionPane.ERROR_MESSAGE);

        // Has que el campo de texto con el primer error obtenga el
        // foco
        switch (posError) {
          case 1:
            campoTextoTitulo.requestFocus();
            break;
```

```
          case 2:
            campoTextoInterprete.requestFocus();
            break;
          case 3:
            campoTextoAutor.requestFocus();
            break;
          case 4:
            campoTextoAlbum.requestFocus();
            break;
          case 5:
            campoTextoDuracion.requestFocus();
            break;
          case 6:
            campoTextoFecha.requestFocus();
            break;
        }
        return;
      }

      // Toma los valores capturados en los campos de texto y en la
      // caja combinada y almacénalos en el parámetro cancion.
      cancion.setTitulo(titulo);
      cancion.setInterprete(interprete);
      cancion.setAutor(autor);
      cancion.setGenero((Genero)
                      cajaCombinadaGenerosCanciones.getSelectedItem());
      cancion.setAlbum(album);
      cancion.setDuracion(Integer.parseInt(sDuracion));
      cancion.setFecha(new Fecha(sFecha));
    }

    // Borra el contenido de respuesta
    respuesta.delete(0, respuesta.length());

    // Establece que se presionó el botón botonAceptar
    respuesta.append(ConstantesGUI.ACEPTAR);

    // Destruye el cuadro de díalogo
    dispose();
  }

  /**
   * Método oyente del botón botonRestaurar. Restaura los valores de
   * los campos de entrada y caja combinada a sus valores originales.
   *
   * @param evt Evento al que escucha
   */
  private void botonRestaurarActionPerformed(java.awt.event.ActionEvent
                                             evt) {
```

```
    // Restaura el contenido de los campos de texto a su valor original
    // Si la operación es Agregar
    if (operacion == ConstantesGUI.AGREGAR) {
      campoTextoTitulo.setText("");
      campoTextoInterprete.setText("");
      campoTextoAutor.setText("");
      cajaCombinadaGenerosCanciones.setSelectedIndex(0);
      campoTextoAlbum.setText("");
      campoTextoDuracion.setText("");
      campoTextoFecha.setText("");
    }

    // Si la operación es Actualizar
    if (operacion == ConstantesGUI.ACTUALIZAR) {
      campoTextoTitulo.setText(cancion.getTitulo());
      campoTextoInterprete.setText(cancion.getInterprete());
      campoTextoAutor.setText(cancion.getAutor());
      cajaCombinadaGenerosCanciones
                                .setSelectedItem(cancion.getGenero());
      campoTextoAlbum.setText(cancion.getAlbum());
      campoTextoDuracion.setText(new Integer(cancion.getDuracion())
                                .toString());
      campoTextoFecha.setText(cancion.getFecha().toString());
    }
  }

  /**
   * Método oyente del botón botonCancelar. cierra el cuadro de
   ** dialogo.
   *
   * @param evt Evento al que escucha
   */
  private void botonCancelarActionPerformed(java.awt.event.ActionEvent
                                            evt) {
    // Destruye el cuadro de díalogo
    dispose();
  }

  /**
   * Metodo oyente del campo de texto del titulo. Valida el titulo.
   * Debe haber un titulo y no debe exceder a los 35 caracteres Si hubo
   * errores despliega un mensaje de error
   *
   * @param evt Evento al que escucha
   */
  private void
    campoTextoTituloActionPerformed(java.awt.event.ActionEvent evt) {
    String titulo = campoTextoTitulo.getText();
    String msjError = "";
```

```
    // Si no hay titulo
    if (validadores.cadenaVacia(titulo)) {
      // crea el mensaje de error
      msjError = "Faltó el título";
    } // Si el titulo es muy largo
    else if (!validadores.validaCadena(35, titulo)) {
      // crea el mensaje de error
      msjError = "Título muy largo";
    }

    // Si hay un mensaje de error
    if (!msjError.equals("")) {
      // Despliega el mensaje de error
      JOptionPane.showMessageDialog(this, msjError,
                                    "Error de entrada.",
              JOptionPane.ERROR_MESSAGE);
      // Has que este campo de texto obtenga el foco
      campoTextoTitulo.requestFocus();
    } else {
      // Has que la siguiente componente obtenga el foco
      campoTextoInterprete.requestFocus();
    }
  }

  /**
   * Metodo oyente del campo de texto del interprete. Valida el
   * interprete. Debe haber un interprete y no debe exceder a los 35
   * caracteres Si hubo errores despliega un mensaje de error
   *
   * @param evt Evento al que escucha
   */
  private void
  campoTextoInterpreteActionPerformed(java.awt.event.ActionEvent evt) {
    String interprete = campoTextoInterprete.getText();
    String msjError = "";

    // Si no hay interprete
    if (validadores.cadenaVacia(interprete)) {
      // crea el mensaje de error
      msjError = "Faltó el intérprete";
    } // Si el interprete es muy largo
    else if (!validadores.validaCadena(35, interprete)) {
      // crea el mensaje de error
      msjError = "Intérprete muy largo";
    }

    // Si hay un mensaje de error
    if (!msjError.equals("")) {
```

```
      // Despliega el mensaje de error
      JOptionPane.showMessageDialog(this, msjError,
                                    "Error de entrada.",
              JOptionPane.ERROR_MESSAGE);
      // Has que este campo de texto obtenga el foco
      campoTextoInterprete.requestFocus();
    } else {
      // Has que la siguiente componente obtenga el foco
      campoTextoAutor.requestFocus();
    }
  }

  /**
   * Metodo oyente del campo de texto del autor. Valida el autor. Si
   * hay un autor, no debe exceder a los 35 caracteres Si hubo errores
   * despliega un mensaje de error
   *
   * @param evt Evento al que escucha
   */
  private void
       campoTextoAutorActionPerformed(java.awt.event.ActionEvent evt) {
    String autor = campoTextoAutor.getText();

    // Si hay un autor y es muy largo
    if (!validadores.cadenaVacia(autor)
            && !validadores.validaCadena(5, autor)) {
      // Despliega el mensaje de error
      JOptionPane.showMessageDialog(this, "Autor muy largo",
                                    "Error de entrada.",
              JOptionPane.ERROR_MESSAGE);
      // Has que este campo de texto obtenga el foco
      campoTextoAutor.requestFocus();
    } else {
      // Has que la siguiente componente obtenga el foco
      cajaCombinadaGenerosCanciones.requestFocus();
    }
  }

  /**
   * Metodo oyente de la caja combinada del genero. Pasa el foco al
   * siguiente campo de texto
   *
   * @param evt Evento al que escucha
   */
  private void
cajaCombinadaGenerosCancionesActionPerformed(java.awt.event.ActionEvent
evt) {
      // Has que la siguiente componente obtenga el foco
    campoTextoAlbum.requestFocus();
```

```
  }

  /**
   * Metodo oyente del campo de texto del album. Valida el album. Si
   * hay un album, no debe exceder a los 35 caracteres Si hubo errores
   * despliega un mensaje de error
   *
   * @param evt Evento al que escucha
   */
  private void
     campoTextoAlbumActionPerformed(java.awt.event.ActionEvent evt) {
    String album = campoTextoAlbum.getText();

    // Si hay un album y es muy largo
    if (!validadores.cadenaVacia(album)
            && !validadores.validaCadena(5, album)) {
      JOptionPane.showMessageDialog(this, "Album muy largo",
                                    "Error de entrada.",
              JOptionPane.ERROR_MESSAGE);
      // Has que este campo de texto obtenga el foco
      campoTextoAlbum.requestFocus();
    } else {
       // Has que la siguiente componente obtenga el foco
     campoTextoDuracion.requestFocus();
    }
  }

  /**
   * Metodo oyente del campo de texto de la duracion. Valida la
   * duracion. Si existe no debe exceder a los tres digitos. Si hubo
   * errores despliega un mensaje de error
   *
   * @param evt Evento al que escucha
   */
  private void
    campoTextoDuracionActionPerformed(java.awt.event.ActionEvent evt) {
    String sDuracion = campoTextoDuracion.getText();

    // Si hay una duracion y es muy larga
    if (!validadores.cadenaVacia(sDuracion)
            && !validadores.validaEnteroMaxDigitos(3, sDuracion)) {
      // Despliega el mensaje de error
      JOptionPane.showMessageDialog(this, "Entero muy largo",
                                    "Error de entrada.",
              JOptionPane.ERROR_MESSAGE);
      // Has que este campo de texto obtenga el foco
      campoTextoDuracion.requestFocus();
    } else {
      // Has que la siguiente componente obtenga el foco
```

```
        campoTextoFecha.requestFocus();
      }
    }

    /**
     * Metodo oyente del campo de texto de lA fecha. Valida la fecha.
     * Debe estar en el formato dd/mm/aaaa. Si hubo errores despliega un
     * mensaje de error
     * @param evt Evento al que escucha
     */
    private void
         campoTextoFechaActionPerformed(java.awt.event.ActionEvent evt) {
      String sFecha = campoTextoFecha.getText();

      // Si hay una fecha y es invalida
      if (!validadores.cadenaVacia(sFecha)
              && !validadores.validaFecha(sFecha)) {
        // Despliega el mensaje de error
        JOptionPane.showMessageDialog(this, "Fecha inválida",
                                      "Error de entrada.",
                JOptionPane.ERROR_MESSAGE);
        // Has que este campo de texto obtenga el foco
        campoTextoFecha.requestFocus();
      }
    }

    ...

    private Cancion cancion;
    private DefaultComboBoxModel listaGenerosCanciones;
    private int operacion;
    private StringBuffer respuesta;
    private Validadores validadores = new Validadores();
}
```