29

Todas las noticias que puede imprimir.
—Adolph S. Ochs

¿Qué loca persecución?
¿De qué aprieto escapar?
—John Keats

No elimines el punto de referencia en el límite de los campos.
—Amenehope

Salida con formato

OBJETIVOS

En este capítulo aprenderá a:

- Trabajar con los flujos de entrada y salida.
- Usar el formato `printf`.
- Imprimir con anchuras de campo y precisiones.
- Usar banderas de formato en la cadena de formato de `printf`.
- Imprimir con un índice como argumento.
- Imprimir literales y secuencias de escape.
- Dar formato a la salida con la clase `Formatter`.

Plan general

29.1 Introducción

Una parte importante de la solución a cualquier problema es la presentación de los resultados. En este capítulo, hablaremos sobre las características de formato del método **`printf`** y la clase `Formatter` (paquete `java.util`). El método `printf` aplica formato a los datos y los imprime en el flujo de salida estándar (`System.out`). La clase **`Formatter`** aplica formato a los datos y los imprime en un destino especificado, como una cadena o un flujo de salida de archivo.

Anteriormente en este libro hablamos sobre muchas de las características de `printf`. En este capítulo se sintetizan esas características y se presentan otras, como mostrar datos de fecha y hora en varios formatos, reordenar la salida con base en el índice del argumento, y mostrar números y cadenas con varias banderas.

29.2 Flujos

Por lo general, las operaciones de entrada y salida se llevan a cabo con flujos, los cuales son secuencias de bytes. En las operaciones de entrada, los bytes fluyen de un dispositivo (como un teclado, una unidad de disco, una conexión de red) a la memoria principal. En las operaciones de salida, los bytes fluyen de la memoria principal a un dispositivo (como una pantalla, una impresora, una unidad de disco, una conexión de red).

Cuando empieza la ejecución de un programa, tres flujos se conectan a éste de manera automática. Por lo común, el flujo de entrada estándar se conecta al teclado, y el flujo de salida estándar se conecta a la pantalla. Un tercer flujo, el **flujo de error estándar** (`System.err`), se conecta generalmente a la pantalla y se utiliza para imprimir mensajes de error en ésta, de manera que puedan verse de inmediato; aún y cuando el flujo de salida estándar esté escribiendo en un archivo. Generalmente, los sistemas operativos permiten redirigir estos flujos a otros dispositivos. En el capítulo 14, Archivos y flujos, y en el capítulo 24, Redes, se describen los flujos con detalle.

29.3 Aplicación de formato a la salida con `printf`

Con `printf` podemos lograr un formato preciso en la salida. [*Nota:* Java SE 5 tomó prestada esta característica del lenguaje de programación C]. El método `printf` cuenta con las siguientes herramientas de formato, cada una de las cuales se verá en este capítulo:

1. Redondeo de valores de punto flotante a un número indicado de posiciones decimales.
2. Alineación de una columna de números con puntos decimales, que aparezcan uno encima de otro.
3. Justificación a la derecha y justificación a la izquierda de los resultados.

4. Inserción de caracteres literales en posiciones precisas de una línea de salida.
5. Representación de números de punto flotante en formato exponencial.
6. Representación de enteros en formato octal y hexadecimal (vea el apéndice E, Sistemas numéricos, para obtener más información acerca de los valores octales y hexadecimales).
7. Visualización de todo tipo de datos con anchuras de campo de tamaño fijo y precisiones.
8. Visualización de fechas y horas en diversos formatos.

Cada llamada a `printf` proporciona como primer argumento una **cadena de formato**, la cual describe el formato de salida. La cadena de formato puede consistir en **texto fijo** y **especificadores de formato**. El texto fijo se imprime mediante `printf` de igual forma que como se imprimiría mediante los métodos `print` o `println` de `System.out`. Cada especificador de formato es un receptáculo para un valor y especifica el tipo de datos a imprimir. Los especificadores de formato también pueden incluir información de formato opcional.

En su forma más simple, cada especificador de formato empieza con un signo de porcentaje (`%`) y va seguido de un **carácter de conversión** que representa el tipo de datos del valor a imprimir. Por ejemplo, el especificador de formato `%s` es un receptáculo para una cadena, y el especificador de formato `%d` es un receptáculo para un valor `int`. La información de formato opcional se especifica entre el signo de porcentaje y el carácter de conversión. La información de formato opcional incluye un índice como argumento, banderas, anchura de campo y precisión. A lo largo de este capítulo, definiremos cada uno de estos elementos, y mostraremos ejemplos.

29.4 Impresión de enteros

Un entero es un número completo, como `776`, `0` o `-52`, que no contiene punto decimal. Los valores enteros se muestran en uno de varios formatos. En la figura 29.1 se describen los **caracteres de conversión integrales**.

En la figura 29.2 se imprime un entero usando cada una de las conversiones integrales. En las líneas 9 a 10, observe que el signo positivo no se muestra de manera predeterminada, pero el signo negativo sí. Más adelante en este capítulo (figura 29.14) veremos cómo forzar a que se impriman los signos positivos.

El método `printf` tiene la forma

```
printf( cadena-de-formato, lista-de-argumentos );
```

en donde *cadena-de-formato* describe el formato de salida, y *lista-de-argumentos* contiene los valores que corresponden a cada especificador de formato en *cadena-de-formato*. Puede haber muchos especificadores de formato en una cadena de formato.

Cada cadena de formato en las líneas 8 a 10 especifica que `printf` debe imprimir un entero decimal (`%d`) seguido de un carácter de nueva línea. En la posición del especificador de formato, `printf` sustituye el valor del primer argumento después de la cadena de formato. Si la cadena de formato contiene varios especificadores de formato, en cada posición del siguiente especificador de formato, `printf` sustituirá el valor del siguiente argumento en la lista de argumentos. El especificador de formato `%o` en la línea 11 imprime el entero en formato octal. El especificador de formato `%x` en la línea 12 imprime el entero en formato hexadecimal. El especificador de formato `%X` en la línea 13 imprime el entero en formato hexadecimal, con letras mayúsculas.

Carácter de conversión	Descripción
`d`	Muestra un entero decimal (base 10).
`o`	Muestra un entero octal (base 8).
`x` o `X`	Muestra un entero hexadecimal (base 16). `X` hace que se muestren los dígitos del 0 al 9 y las letras `A` a la `F`, y `x` hace que se muestren los dígitos del 0 al 9 y las letras de la a a la f.

Figura 29.1 | Caracteres de conversión de enteros.

```
1  // Fig. 29.2: PruebaConversionEnteros.java
2  // Uso de los caracteres de conversión integrales.
3
4  public class PruebaConversionEnteros
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%d\n", 26 );
9        System.out.printf( "%d\n", +26 );
10       System.out.printf( "%d\n", -26 );
11       System.out.printf( "%o\n", 26 );
12       System.out.printf( "%x\n", 26 );
13       System.out.printf( "%X\n", 26 );
14    } // fin de main
15 } // fin de la clase PruebaConversionEnteros
```

```
26
26
-26
32
1a
1A
```

Figura 29.2 | Uso de caracteres de conversión de enteros.

29.5 Impresión de números de punto flotante

Un valor de punto flotante contiene un punto decimal, como en `33.5`, `0.0` o `-657.983`. Los valores de punto flotante se muestran en uno de varios formatos. En la figura 29.3 se describen las conversiones de punto flotante. Los **caracteres de conversión e** y `E` muestran valores de punto flotante en **notación científica computarizada** (también conocida como **notación exponencial**). La notación exponencial es el equivalente computacional de la notación científica que se utiliza en las matemáticas. Por ejemplo, el valor `150.4582` se representa en notación científica matemática de la siguiente manera:

1.504582×10^2

y se representa en notación exponencial como

```
1.504582e+02
```

en Java. Esta notación indica que `1.504582` se multiplica por `10` elevado a la segunda potencia (`e+02`). La **e** representa al "exponente".

Carácter de conversión	Descripción
`e` o `E`	Muestra un valor de punto flotante en notación exponencial. Cuando se utiliza el carácter de conversión `E`, la salida se muestra en letras mayúsculas.
`f`	Muestra un valor de punto flotante en formato decimal.
`g` o `G`	Muestra un valor de punto flotante en el formato de punto flotante `f` o en el formato exponencial `e`, con base en la magnitud del valor. Si la magnitud es menor que 10^{-3}, o si es mayor o igual que 10^7, el valor de punto flotante se imprime con `e` (o `E`). En cualquier otro caso, el valor se imprime en el formato `f`. Cuando se utiliza el carácter de conversión `G`, la salida se muestra en letras mayúsculas.
`a` o `A`	Muestra un número de punto flotante en formato hexadecimal. Cuando se usa el carácter de conversión `A`, la salida se muestra en letras mayúsculas.

Figura 29.3 | Caracteres de conversión de punto flotante.

Los valores que se imprimen con los caracteres de conversión `e`, `E` y `f` se muestran con seis dígitos de precisión en el lado derecho del punto decimal de manera predeterminada (por ejemplo, `1.045921`); otras precisiones se deben especificar de manera explícita. Para los valores impresos con el carácter de conversión `g`, la precisión representa el número total de dígitos mostrados, excluyendo el exponente. El valor predeterminado es de seis dígitos (por ejemplo, `12345678.9` se muestra como `1.23457e+07`). El **carácter de conversión f** siempre imprime por lo menos un dígito a la izquierda del punto decimal. Los caracteres de conversión `e` y `E` imprimen una `e` minúscula y una `E` mayúscula antes del exponente, y siempre imprimen sólo un dígito a la izquierda del punto decimal. El redondeo ocurre si el valor al que se está dando formato tiene más dígitos significativos que la precisión.

El **carácter de conversión g** (o **G**) imprime en formato `e` (`E`) o `f`, dependiendo del valor de punto flotante. Por ejemplo, los valores `0.0000875`, `87500000.0`, `8.75`, `87.50` y `875.0` se imprimen como `8.750000e-05`, `8.750000e+07`, `8.750000`, `87.500000` y `875.000000` con el carácter de conversión `g`. El valor `0.0000875` utiliza la notación `e` ya que la magnitud es menor que 10^{-3}. El valor `87500000.0` utiliza la notación `e`, debido a que la magnitud es mayor que 10^{7}. En la figura 29.4 se muestra cada uno de los caracteres de conversión de punto flotante.

```
1  // Fig. 29.4: PruebaPuntoFlotante.java
2  // Uso de los caracteres de conversión de punto flotante.
3
4  public class PruebaPuntoFlotante
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%e\n", 12345678.9 );
9        System.out.printf( "%e\n", +12345678.9 );
10       System.out.printf( "%e\n", -12345678.9 );
11       System.out.printf( "%E\n", 12345678.9 );
12       System.out.printf( "%f\n", 12345678.9 );
13       System.out.printf( "%g\n", 12345678.9 );
14       System.out.printf( "%G\n", 12345678.9 );
15    } // fin de main
16 } // fin de la clase PruebaPuntoFlotante
```

```
1.234568e+07
1.234568e+07
-1.234568e+07
1.234568E+07
12345678.900000
1.23457e+07
1.23457E+07
```

Figura 29.4 | Uso de los caracteres de conversión de punto flotante.

29.6 Impresión de cadenas y caracteres

Los caracteres de conversión `c` y `s` se utilizan para imprimir caracteres individuales y cadenas, respectivamente. El carácter de conversión `s` también puede imprimir objetos con los resultados de las llamadas implícitas al método `toString`. Los **caracteres de conversión c** y **C** requieren un argumento `char`. Los **caracteres de conversión s** y **S** pueden recibir un objeto `String` o cualquier objeto `Object` (se incluyen todas las subclases de `Object`) como argumento. Cuando se pasa un objeto al carácter de conversión `s`, el programa utiliza de manera implícita el método `toString` del objeto para obtener la representación `String` del objeto. Cuando se utilizan los caracteres de conversión `C` y `S`, la salida se muestra en letras mayúsculas. El programa que se muestra en la figura 29.5 imprime en pantalla caracteres, cadenas y objetos con los caracteres de conversión `c` y `s`. Observe que se realiza una conversión autoboxing en la línea 10, cuando se asigna una constante `int` a un objeto `Integer`. En la línea 15 se asocia un objeto `Integer` como argumento para el carácter de conversión `s`, el cual invoca de manera implícita al método `toString` para obtener el valor entero. Observe que también puede imprimir en pantalla un objeto `Integer` mediante el uso del especificador de formato `%d`. En este caso, se realizará una conversión unboxing con el valor `int` en el objeto `Integer` y se imprimirá en pantalla.

```
1  // Fig. 29.5: ConversionCadenasChar.java
2  // Uso de los caracteres de conversión de cadenas y caracteres.
3
4  public class ConversionCadenasChar
5  {
6     public static void main( String args[] )
7     {
8        char caracter= 'A';  // inicializa el char
9        String cadena = "Esta tambien es una cadena"; // objeto String
10       Integer entero = 1234;  // inicializa el entero (autoboxing)
11
12       System.out.printf( "%c\n", caracter );
13       System.out.printf( "%s\n", "Esta es una cadena" );
14       System.out.printf( "%s\n", cadena );
15       System.out.printf( "%S\n", cadena );
16       System.out.printf( "%s\n", entero ); // llamada implícita a toString
17    } // fin de main
18 } // fin de la clase ConversionCadenasChar
```

```
A
Esta es una cadena
Esta tambien es una cadena
ESTA TAMBIEN ES UNA CADENA
1234
```

Figura 29.5 | Uso de los caracteres de conversión de cadenas y caracteres.

Error común de programación 29.1

El uso de %c para imprimir una cadena produce una excepción `IllegalFormatConversionException`; una cadena no se puede convertir en un carácter.

29.7 Impresión de fechas y horas

Con el **carácter de conversión** `t` o `T`, podemos imprimir fechas y horas en diversos formatos. El carácter de conversión `t` o `T` siempre va seguido de un **carácter de sufijo de conversión** que especifica el formato de fecha y/o de hora. Cuando se utiliza el carácter de conversión `T`, la salida se muestra en letras mayúsculas. En la figura 29.6 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las **composiciones de fecha y hora** que muestran tanto la fecha como la hora. En la figura 29.7 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las fechas. En la figura 29.8 se listan los caracteres de sufijo de conversión comunes para aplicar formato a las horas. Para ver la lista completa de caracteres de sufijo de conversión, visite el sitio Web `java.sun.com/javase/6/docs/api/java/util/Formatter.html`.

Carácter de sufijo de conversión	Descripción
`c`	Muestra la fecha y hora con el formato `dia mes fecha hora:minuto:segundo zona-horaria año` con tres caracteres para `dia` y `mes`, dos dígitos para `fecha`, `hora`, `minuto` y `segundo`, y cuatro dígitos para `año`; por ejemplo, `Mié Mar 03 16:30:25 GMT –05:00 2004`. Se utiliza el reloj de 24 horas. En este ejemplo, `GMT –05:00` es la zona horaria.
`F`	Muestra la fecha con el formato `año-mes-dia` con cuatro dígitos para el `año` y dos dígitos para el `mes` y la `fecha` (por ejemplo, `2004-05-04`).

Figura 29.6 | Caracteres de sufijo de conversión de composiciones de fecha y hora. (Parte 1 de 2).

Carácter de sufijo de conversión	Descripción
`D`	Muestra la fecha con el formato `mes/dia/año`, con dos dígitos para el `mes`, `dia` y `año` (por ejemplo, `03/03/04`).
`r`	Muestra la hora con el formato `hora:minuto:segundo AM|PM`, con dos dígitos para la `hora`, `minuto` y `segundo` (por ejemplo, `04:30:25 PM`). Se utiliza el reloj de 12 horas.
`R`	Muestra la hora con el formato `hora:minuto`, con dos dígitos para la `hora` y el `minuto` (por ejemplo, `16:30`). Se utiliza el reloj de 24 horas.
`T`	Muestra la hora con el formato `hora:minuto:segundo`, con dos dígitos para la `hora`, `minuto` y `segundo` (por ejemplo, `16:30:25`). Se utiliza el reloj de 24 horas.

Figura 29.6 | Caracteres de sufijo de conversión de composiciones de fecha y hora. (Parte 2 de 2).

Carácter de sufijo de conversión	Descripción
`A`	Muestra el nombre completo del día de la semana (por ejemplo, `Miércoles`).
`a`	Muestra el nombre corto de tres caracteres del día de la semana (por ejemplo, `Mié`).
`B`	Muestra el nombre completo del mes (por ejemplo, `Marzo`).
`b`	Muestra el nombre corto de tres caracteres del mes (por ejemplo, `Mar`).
`d`	Muestra el día del mes con dos dígitos, rellenando con ceros a la izquierda si es necesario (por ejemplo, `03`).
`m`	Muestra el mes con dos dígitos, rellenando con ceros a la izquierda si es necesario (por ejemplo, `07`).
`e`	Muestra el día del mes sin ceros a la izquierda (por ejemplo, `3`).
`Y`	Muestra el año con cuatro dígitos (por ejemplo, `2004`).
`y`	Muestra los dos últimos dígitos del año con ceros a la izquierda, según sea necesario (por ejemplo, `04`).
`j`	Muestra el día del año con tres dígitos, rellenando con ceros a la izquierda según sea necesario (por ejemplo, `016`).

Figura 29.7 | Caracteres de sufijo de conversión para aplicar formato a las fechas.

Carácter de sufijo de conversión	Descripción
`H`	Muestra la hora en el reloj de 24 horas, con un cero a la izquierda si es necesario (por ejemplo, `16`).
`I`	Muestra la hora en el reloj de 12 horas, con un cero a la izquierda si es necesario (por ejemplo, `04`).
`k`	Muestra la hora en el reloj de 24 horas sin ceros a la izquierda (por ejemplo, `16`).
`l`	Muestra la hora en el reloj de 12 horas sin ceros a la izquierda (por ejemplo, `4`).

Figura 29.8 | Caracteres de sufijo de conversión para aplicar formato a las horas. (Parte 1 de 2).

Carácter de sufijo de conversión	Descripción
`M`	Muestra los minutos con un cero a la izquierda, si es necesario (por ejemplo, `06`).
`S`	Muestra los segundos con un cero a la izquierda, si es necesario (por ejemplo, `05`).
`Z`	Muestra la abreviación para la zona horaria (por ejemplo, `GMT -05:00`, que representa a la Hora estándar occidental, la cual se encuentra a 5 horas de retraso de la Hora del Meridiano de Greenwich).
`p`	Muestra las iniciales de mañana o tarde en minúsculas (por ejemplo, `pm`).
`P`	Muestra el marcador de mañana o tarde en mayúsculas (por ejemplo, `PM`).

Figura 29.8 | Caracteres de sufijo de conversión para aplicar formato a las horas. (Parte 2 de 2).

En la figura 29.9 se utiliza el carácter de conversión `t` con los caracteres de sufijo de conversión para mostrar fechas y horas en diversos formatos. El carácter de conversión `t` requiere que su correspondiente argumento sea de tipo `long`, `Long`, `Calendar` o `Date` (ambas clases se encuentran en el paquete `java.util`); los objetos de cada una de estas clases pueden representar fechas y horas. La clase `Calendar` es la preferida para este propósito, ya que ciertos constructores y métodos en la clase `Date` se sustituyen por los de la clase `Calendar`. En la línea 10 se invoca el método `static` `getInstance` de `Calendar` para obtener un calendario con la fecha y hora actuales. En las líneas 13 a 17, 20 a 22 y 25 a 26 se utiliza este objeto `Calendar` en instrucciones `printf` como el valor al que se aplicará formato con el carácter de conversión `t`. Observe que en las líneas 20 a 22 y 25 a 26 se utiliza el **índice como argumento** opcional (`"1$"`) para indicar que todos los especificadores de formato en la cadena de formato utilizan el primer argumento después de la cadena de formato en la lista de argumentos. En la sección 29.11 aprenderá más acerca de los índices como argumentos. Al usar el índice como argumento, se elimina la necesidad de listar repetidas veces el mismo argumento.

```
1  // Fig. 29.9: PruebaFechaHora.java
2  // Aplicación de formato a fechas y horas con los caracteres de conversión t y T.
3  import java.util.Calendar;
4
5  public class PruebaFechaHora
6  {
7     public static void main( String args[] )
8     {
9        // obtiene la fecha y hora actuales
10       Calendar fechaHora = Calendar.getInstance();
11
12       // impresión con caracteres de conversión para composiciones de fecha/hora
13       System.out.printf( "%tc\n", fechaHora );
14       System.out.printf( "%tF\n", fechaHora );
15       System.out.printf( "%tD\n", fechaHora );
16       System.out.printf( "%tr\n", fechaHora );
17       System.out.printf( "%tT\n", fechaHora );
18
19       // impresión con caracteres de conversión para fechas
20       System.out.printf( "%1$tA, %1$tB %1$td, %1$tY\n", fechaHora );
21       System.out.printf( "%1$TA, %1$TB %1$Td, %1$TY\n", fechaHora );
22       System.out.printf( "%1$ta, %1$tb %1$te, %1$ty\n", fechaHora );
23
24       // impresión con caracteres de conversión para horas
25       System.out.printf( "%1$tH:%1$tM:%1$tS\n", fechaHora );
```

Figura 29.9 | Aplicación de formato a fechas y horas con los caracteres de conversión `t`. (Parte 1 de 2).

```
26         System.out.printf( "%1$tZ %1$tI:%1$tM:%1$tS %Tp", fechaHora );
27      } // fin de main
28   } // fin de la clase PruebaFechaHora
```

```
mié nov 07 11:54:30 CST 2007
2007-11-07
11/07/07
11:54:30 AM
11:54:30
miércoles, noviembre 07, 2007
MIÉRCOLES, NOVIEMBRE 07, 2007
mié, nov 7, 07
11:54:30
CST 11:54:30 AM
```

Figura 29.9 | Aplicación de formato a fechas y horas con los caracteres de conversión `t`. (Parte 2 de 2).

29.8 Otros caracteres de conversión

El resto de los caracteres de conversión son **b**, **B**, **h**, **H**, **%** y **n**. Éstos se describen en la figura 29.10.

En las líneas 9 y 10 de la figura 29.11 se utiliza `%b` para imprimir el valor de los valores booleanos `false` y `true`. En la línea 11 se asocia un objeto `String` a `%b`, el cual devuelve `true` debido a que no es `null`. En la línea 12 se asocia un objeto `null` a `%B`, el cual muestra `FALSE` ya que `prueba` es `null`. En las líneas 13 y 14 se utiliza `%h` para imprimir las representaciones de cadena de los valores de código hash para las cadenas `"hola"` y `"Hola"`. Estos valores se podrían utilizar para almacenar o colocar las cadenas en un objeto `Hashtable` o `HashMap` (los cuales vimos en el capítulo 19, Colecciones). Observe que los valores de código hash para estas dos cadenas difieren, ya que una cadena empieza con letra minúscula y la otra con letra mayúscula. En la línea 15 se utiliza `%H` para imprimir `null` en letras mayúsculas. Las últimas dos instrucciones `printf` (líneas 16 y 17) utilizan `%%` para imprimir el carácter `%` en una cadena, y `%n` para imprimir un separador de línea específico de la plataforma.

Error común de programación 29.2

Tratar de imprimir un carácter de porcentaje literal mediante el uso de % en vez de %% en la cadena de formato podría provocar un error lógico difícil de detectar. Cuando aparece el % en una cadena de formato, debe ir seguido de un carácter de conversión en la cadena. El signo de por ciento individual podría ir seguido accidentalmente de un carácter de conversión legítimo, con lo cual se produciría un error lógico.

Carácter de conversión	Descripción
`b` o `B`	Imprime `"true"` o `"false"` para el valor de un `boolean` o `Boolean`. Estos caracteres de conversión también pueden aplicar formato al valor de cualquier referencia. Si la referencia no es `null`, se imprime `"true"`; en caso contrario, se imprime `"false"`. Cuando se utiliza el carácter de conversión `B`, la salida se muestra en letras mayúsculas.
`h` o `H`	Imprime la representación de cadena del valor de código hash de un objeto en formato hexadecimal. Si el correspondiente argumento es `null`, se imprime `"null"`. Cuando se utiliza el carácter de conversión `H`, la salida se muestra en letras mayúsculas.
`%`	Imprime el carácter de por ciento.
`n`	Imprime el separador de línea específico de la plataforma (por ejemplo, `\r\n` en Windows o `\n` en UNIX/LINUX).

Figura 29.10 | Otros especificadores de conversión.

```
1  // Fig. 29.11: OtrasConversiones.java
2  // Uso de los caracteres de conversión b, B, h, H, % y n.
3
4  public class OtrasConversiones
5  {
6     public static void main( String args[] )
7     {
8        Object prueba = null;
9        System.out.printf( "%b\n", false );
10       System.out.printf( "%b\n", true );
11       System.out.printf( "%b\n", "Prueba" );
12       System.out.printf( "%B\n", prueba );
13       System.out.printf( "El codigo hash de \"hola\" es %h\n", "hola" );
14       System.out.printf( "El codigo hash de \"Hola\" es %h\n", "Hola" );
15       System.out.printf( "El codigo hash de null es %H\n", prueba );
16       System.out.printf( "Impresion de un %% en una cadena de formato\n" );
17       System.out.printf( "Impresion de una nueva linea %nla siguiente linea empieza
         aqui" );
18    } // fin de main
19 } // fin de la clase OtrasConversiones
```

```
false
true
true
FALSE
El codigo hash de "hola" es 30f4bc
El codigo hash de "Hola" es 2268dc
El codigo hash de null es NULL
Impresion de un % en una cadena de formato
Impresion de una nueva linea
la siguiente linea empieza aquí
```

Figura 29.11 | Uso de los caracteres de conversión `b`, `B`, `h`, `H`, `%` y `n`.

29.9 Impresión con anchuras de campo y precisiones

El tamaño exacto de un campo en el que se imprimen datos se especifica mediante una **anchura de campo**. Si la anchura de campo es mayor que los datos que se van a imprimir, éstos se justificarán a la derecha dentro de ese campo, de manera predeterminada. (En la sección 29.10 demostraremos la justificación a la izquierda). El programador inserta un entero que representa la anchura de campo entre el signo de porcentaje (`%`) y el carácter de conversión (por ejemplo, `%4d`) en el especificador de formato. En la figura 29.12 se imprimen dos grupos de cinco números cada uno, y se justifican a la derecha los números que contienen menos dígitos que la anchura de campo. Observe que la anchura de campo se incrementa para imprimir valores más anchos que el campo, y que el signo menos para un valor negativo utiliza una posición de carácter en el campo. Además, si no se especifica la anchura del campo, los datos se imprimen en todas las posiciones que sean necesarias. Las anchuras de campo pueden utilizarse con todos los especificadores de formato, excepto el separador de línea (`%n`).

Error común de programación 29.3

Si no se proporciona una anchura de campo suficientemente extensa como para manejar un valor a imprimir, se pueden desplazar los demás datos que se impriman, con lo que se producirán resultados confusos. ¡Debe conocer sus datos!

El método `printf` también proporciona la habilidad de especificar la precisión con la que se van a imprimir los datos. La precisión tiene distintos significados para los diferentes tipos. Cuando se utiliza con los caracteres de conversión de punto flotante `e` y `f`, la precisión es el número de dígitos que aparecen después del punto decimal.

```
1  // Fig. 29.12: PruebaAnchuraCampo.java
2  // Justificación a la derecha de enteros en campos.
3
4  public class PruebaAnchuraCampo
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%4d\n", 1 );
9        System.out.printf( "%4d\n", 12 );
10       System.out.printf( "%4d\n", 123 );
11       System.out.printf( "%4d\n", 1234 );
12       System.out.printf( "%4d\n\n", 12345 ); // datos demasiado extensos
13
14       System.out.printf( "%4d\n", -1 );
15       System.out.printf( "%4d\n", -12 );
16       System.out.printf( "%4d\n", -123 );
17       System.out.printf( "%4d\n", -1234 ); // datos demasiado extensos
18       System.out.printf( "%4d\n", -12345 ); // datos demasiado extensos
19    } // fin de main
20 } // fin de la clase PruebaAnchuraCampo
```

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

Figura 29.12 | Justificación a la derecha de enteros en campos.

Cuando se utiliza con el carácter de conversión `g`, la precisión es el número máximo de dígitos significativos a imprimir. Cuando se utiliza con el carácter de conversión `s`, la precisión es el número máximo de caracteres a escribir de la cadena. Para utilizar la precisión, se debe colocar entre el signo de porcentaje y el especificador de conversión un punto decimal (`.`), seguido de un entero que representa la precisión. En la figura 29.13 se muestra el uso de la precisión en las cadenas de formato. Observe que, cuando se imprime un valor de punto flotante con una precisión menor que el número original de posiciones decimales en el valor, éste se redondea. Además, observe que el especificador de formato `%.3g` indica que el número total de dígitos utilizados para mostrar el valor de punto flotante es `3`. Como el valor tiene tres dígitos a la izquierda del punto decimal, se redondea a la posición de las unidades.

La anchura de campo y la precisión pueden combinarse, para lo cual se coloca la anchura de campo, seguida de un punto decimal, seguido de una precisión entre el signo de porcentaje y el carácter de conversión, como en la siguiente instrucción:

```
printf( "%9.3f", 123.456789 );
```

la cual muestra `123.457` con tres dígitos a la derecha del punto decimal, y se justifica a la derecha en un campo de nueve dígitos; antes del número se colocarán dos espacios en blanco en su campo.

29.10 Uso de banderas en la cadena de formato de `printf`

Pueden usarse varias banderas con el método `printf` para suplementar sus herramientas de formato de salida. Hay siete banderas disponibles para usarlas en las cadenas de formato (figura 29.14).

```
1  // Fig 29.13: PruebaPrecision.java
2  // Uso de la precisión para números de punto flotante y cadenas.
3  public class PruebaPrecision
4  {
5     public static void main( String args[] )
6     {
7        double f = 123.94536;
8        String s = "Feliz Cumpleanios";
9
10       System.out.printf( "Uso de la precision para numeros de punto flotante\n" );
11       System.out.printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
12
13       System.out.printf( "Uso de la precision para las cadenas\n" );
14       System.out.printf( "\t%.11s\n", s );
15    } // fin de main
16 } // fin de la clase PruebaPrecision
```

```
Uso de la precision para numeros de punto flotante
         123.945
         1.239e+02
         124

Uso de la precision para las cadenas
         Feliz Cumpl
```

Figura 29.13 | Uso de la precisión para los números de punto flotante y las cadenas.

Bandera	Descripción
- (signo negativo)	Justifica a la izquierda la salida dentro del campo especificado.
+ (signo positivo)	Muestra un signo positivo antes de los valores positivos, y un signo negativo antes de los valores negativos.
espacio	Imprime un espacio antes de un valor positivo que no se imprime con la bandera +.
#	Antepone un `0` al valor de salida cuando se utiliza con el carácter de conversión octal `o`. Antepone `0x` al valor de salida cuando se usa con el carácter de conversión hexadecimal `x`.
0 (cero)	Rellena un campo con ceros a la izquierda.
, (coma)	Usa el separador de miles específico para la configuración regional (es decir, `','` para los EUA), para mostrar números decimales y de punto flotante.
(	Encierra los números negativos entre paréntesis.

Figura 29.14 | Banderas de la cadena de formato.

Para usar una bandera en una cadena de formato, coloque la bandera justo a la derecha del signo de porcentaje. Pueden usarse varias banderas en el mismo especificador de formato. En la figura 29.15 se muestra la justificación a la derecha y la justificación a la izquierda de una cadena, un entero, un carácter y un número de punto flotante. Observe que la línea 9 sirve como mecanismo de conteo para la salida en la pantalla.

En la figura 29.16 se imprime un número positivo y un número negativo, cada uno con y sin la **bandera +**. Observe que el signo negativo se muestra en ambos casos, pero el signo positivo se muestra sólo cuando se utiliza la bandera +.

```
1  // Fig. 29.15: PruebaBanderaMenos.java
2  // Justificación a la derecha y justificación a la izquierda de los valores
3
4  public class PruebaBanderaMenos
5  {
6     public static void main( String args[] )
7     {
8        System.out.println( "Columnas:" );
9        System.out.println( "01234567890123456789012345678901234567890123456789\n" );
10       System.out.printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
11       System.out.printf(
12          "%-10s%-10d%-10c%-10f\n", "hola", 7, 'a', 1.23 );
13    } // fin de main
14 } // fin de la clase PruebaBanderaMenos
```

```
Columnas:
01234567890123456789012345678901234567890123456789

     hola         7         a  1.230000

hola      7         a         1.230000
```

Figura 29.15 | Justificación a la derecha y justificación a la izquierda de los valores.

En la figura 29.17 se antepone un espacio al número positivo mediante la **bandera de espacio**. Esto es útil para alinear números positivos y negativos con el mismo número de dígitos. Observe que el valor `-547` no va precedido por un espacio en la salida, debido a su signo negativo. En la figura 29.18 se utiliza la **bandera #** para anteponer un `0` al valor octal, y **`0x`** para el valor hexadecimal.

En la figura 29.19 se combinan la bandera `+`, la **bandera `0`** y la bandera de espacio para imprimir `452` en un campo con una anchura de 9, con un signo **`+`** y ceros a la izquierda; después se imprime `452` en un campo de anchura 9, usando sólo la bandera `0`, y después se imprime `452` en un campo de anchura 9, usando sólo la bandera de espacio.

```
1  // Fig. 29.16: PruebaBanderaMas.java
2  // Impresión de números con y sin la bandera +.
3
4  public class PruebaBanderaMas
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%d\t%d\n", 786, -786 );
9        System.out.printf( "%+d\t%+d\n", 786, -786 );
10    } // fin de main
11 } // fin de la clase PruebaBanderaMas
```

```
786     -786
+786    -786
```

Figura 29.16 | Impresión de números con y sin la bandera +.

En la figura 29.20 se utiliza la bandera de coma (`,`) para mostrar un número decimal y un número de punto flotante con el separador de miles. En la figura 29.21 se encierran números negativos entre paréntesis, usando la bandera **`(`**. Observe que el valor `50` no se encierra entre paréntesis en la salida, ya que es un número positivo.

```
1  // Fig. 29.17: PruebaBanderaEspacio.java
2  // Impresión de un espacio antes de valores no negativos.
3
4  public class PruebaBanderaEspacio
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "% d\n% d\n", 547, -547 );
9     } // fin de main
10 } // fin de la clase PruebaBanderaEspacio
```

```
 547
-547
```

Figura 29.17 | Uso de la bandera de espacio para imprimir un espacio antes de los valores no negativos.

```
1  // Fig. 29.18: PruebaBanderaLibras.java
2  // Uso de la bandera # con los caracteres de conversión o y x.
3
4  public class PruebaBanderaLibras
5  {
6     public static void main( String args[] )
7     {
8        int c = 31;       // inicializa c
9
10       System.out.printf( "%#o\n", c );
11       System.out.printf( "%#x\n", c );
12    } // fin de main
13 } // fin de la clase PruebaBanderaLibras
```

```
037
0x1f
```

Figura 29.18 | Uso de la bandera # con los caracteres de conversión o y x.

```
1  // Fig. 29.19: PruebaBanderaCero.java
2  // Impresión con la bandera 0 (cero) para rellenar con ceros a la izquierda.
3
4  public class PruebaBanderaCero
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%+09d\n", 452 );
9        System.out.printf( "%09d\n", 452 );
10       System.out.printf( "% 9d\n", 452 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaCero
```

```
+00000452
000000452
      452
```

Figura 29.19 | Impresión con la bandera 0 (cero) para rellenar con ceros a la izquierda.

```
1  // Fig. 29.20: PruebaBanderaComa.java
2  // Uso de la bandera de coma (,) para mostrar números con el separador de miles.
3
4  public class PruebaBanderaComa
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%,d\n", 58625 );
9        System.out.printf( "%,.2f\n", 58625.21 );
10       System.out.printf( "%,.2f", 12345678.9 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaComa
```

```
58,625
58,625.21
12,345,678.90
```

Figura 29.20 | Uso de la bandera de coma (,) para mostrar números con el separador de miles.

```
1  // Fig. 29.21: PruebaBanderaParentesis.java
2  // Uso de la bandera ( para colocar paréntesis alrededor de números negativos.
3
4  public class PruebaBanderaParentesis
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf( "%(d\n", 50 );
9        System.out.printf( "%(d\n", -50 );
10       System.out.printf( "%(.1e\n", -50.0 );
11    } // fin de main
12 } // fin de la clase PruebaBanderaParentesis
```

```
50
(50)
(5.0e+01)
```

Figura 29.21 | Uso de la bandera (para colocar paréntesis alrededor de números negativos.

29.11 Impresión con índices como argumentos

Un **índice como argumento** es un entero opcional, seguido de un signo de $, el cual indica la posición del argumento en la lista de argumentos. Por ejemplo, en las líneas 20 a 21 y 24 a 25 de la figura 29.9 se utiliza el índice como argumento "1$" para indicar que todos los especificadores de formato utilizan el primer argumento en la lista de argumentos. Los índices como argumentos permiten a los programadores reordenar la salida, de manera que los argumentos en la lista de argumentos no necesariamente se encuentren en el orden de sus especificadores de formato correspondientes. Los índices como argumentos también ayudan a evitar argumentos duplicados. En la figura 29.22 se muestra cómo imprimir argumentos en la lista de argumentos en orden inverso, mediante el uso del índice como argumento.

```
1  // Fig. 29.22: PruebaIndiceArgumento
2  // Reordenamiento de la salida con los índices como argumentos.
3
4  public class PruebaIndiceArgumento
```

Figura 29.22 | Reordenamiento de la salida con los índices como argumentos. (Parte 1 de 2).

```
5  {
6     public static void main( String args[] )
7     {
8        System.out.printf(
9           "Lista de parametros sin reordenar: %s %s %s %s\n",
10          "primero", "segundo", "tercero", "cuarto" );
11       System.out.printf(
12          "Lista de parametros despues de reordenar: %4$s %3$s %2$s %1$s\n",
13          "primero", "segundo", "tercero", "cuarto" );
14    } // fin de main
15 } // fin de la clase PruebaIndiceArgumento
```

```
Lista de parametros sin reordenar: primero segundo tercero cuarto
Lista de parametros despues de reordenar: cuarto tercero segundo primero
```

Figura 29.22 | Reordenamiento de la salida con los índices como argumentos. (Parte 2 de 2).

29.12 Impresión de literales y secuencias de escape

La mayoría de los caracteres literales que se imprimen en una instrucción `printf` sólo necesitan incluirse en la cadena de formato. Sin embargo, hay varios caracteres "problemáticos", como el signo de comillas dobles (") que delimita a la cadena de formato en sí. Varios caracteres de control, como el carácter de nueva línea y el de tabulación, deben representarse mediante secuencias de escape. Una secuencia de escape se representa mediante una barra diagonal inversa (\), seguida de un carácter de escape. En la figura 29.23 se listan las secuencias de escape y las acciones que producen.

Error común de programación 29.4

Tratar de imprimir un carácter de comillas dobles o un carácter de barra diagonal inversa como datos literales en una instrucción `printf`, sin anteponer al carácter una barra diagonal inversa para formar una secuencia de escape apropiada, podría producir un error de sintaxis.

Secuencia de escape	Descripción
`\'` (comilla sencilla)	Imprime el carácter de comilla sencilla (`'`).
`\"` (doble comilla)	Imprime el carácter de doble comilla (`"`).
`\\` (barra diagonal inversa)	Imprime el carácter de barra diagonal inversa (\).
`\b` (retroceso)	Desplaza el cursor una posición hacia atrás en la línea actual.
`\f` (nueva página o avance de página)	Desplaza el cursor al principio de la siguiente página lógica.
`\n` (nueva línea)	Desplaza el cursor al principio de la siguiente línea.
`\r` (retorno de carro)	Desplaza el cursor al principio de la línea actual.
`\t` (tabulador horizontal)	Desplaza el cursor hacia la siguiente posición del tabulador horizontal.

Figura 29.23 | Secuencias de escape.

29.13 Aplicación de formato a la salida con la clase `Formatter`

Hasta ahora, hemos visto cómo mostrar salida con formato en el flujo de salida estándar. ¿Qué deberíamos hacer si quisiéramos enviar salidas con formato a otros flujos de entrada o dispositivos, como un objeto `JTextArea` o un archivo? La solución recae en la clase `Formatter` (en el paquete `java.util`), la cual proporciona las mismas herramientas de formato que `printf`. `Formatter` es una clase utilitaria que permite a los programadores impri-

mir datos con formato hacia un destino especificado, como un archivo en el disco. De manera predeterminada, un objeto `Formatter` crea una cadena en la memoria. En la figura 29.24 se muestra cómo usar un objeto `Formatter` para crear una cadena con formato, la cual después se muestra en un cuadro de diálogo de mensaje.
En la línea 11 se crea un objeto `Formatter` mediante el uso del constructor predeterminado, por lo que este objeto creará una cadena en la memoria. Se incluyen otros constructores para que el programador pueda especificar el destino hacia el que se deben enviar los datos con formato. Para obtener más información, consulte la página `java.sun.com/javase/6/docs/api/java/util/Formatter.html`

En la línea 12 se invoca el método `format` para dar formato a la salida. Al igual que `printf`, el método `format` recibe una cadena de formato y una lista de argumentos. La diferencia es que `printf` envía la salida con formato directamente al flujo de salida estándar, mientras que `format` envía la salida con formato al destino especificado por su constructor (en este programa, una cadena en la memoria). En la línea 15 se invoca el método `toString` de `Formatter` para obtener los datos con formato, como una cadena que luego se muestra en un cuadro de diálogo de mensaje.

Observe que la clase `String` también proporciona un método de conveniencia `static` llamado `format`, el cual nos permite crear una cadena en la memoria, sin necesidad de crear primero un objeto `Formatter`. Podríamos haber sustituido las líneas 11 a 12 y la línea 15 de la figura 29.24 por:

```
String s = String.format( "%d = %#o = %#^x", 10, 10, 10 );
JOptionPane.showMessageDialog( null, s );
```

```
1  // Fig. 29.24: PruebaFormatter.java
2  // Cadena de formato con la clase Formatter.
3  import java.util.Formatter;
4  import javax.swing.JOptionPane;
5
6  public class PruebaFormatter
7  {
8     public static void main( String args[] )
9     {
10       // crea un objeto Formatter y aplica formato a la salida
11       Formatter formatter = new Formatter();
12       formatter.format( "%d = %#o = %#X", 10, 10, 10 );
13
14       // muestra la salida en el componente JOptionPane
15       JOptionPane.showMessageDialog( null, formatter.toString() );
16    } // fin de main
17 } // fin de la clase PruebaFormatter
```

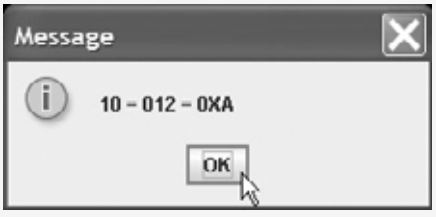

Figura 29.24 | Aplicar formato a la salida con la clase `Formatter`.

29.14 Conclusión

En este capítulo vimos un resumen acerca de cómo aplicar formato a la salida mediante los diversos caracteres y banderas de formato. Mostramos números decimales mediante el uso de los caracteres de formato `d`, `o`, `x` y `X`. Mostramos números de punto flotante usando los caracteres de formato `e`, `E`, `f`, `g` y `G`. Mostramos la fecha y la hora en diversos formatos, usando los caracteres de formato `t` y `T` junto con sus caracteres de sufijo de conversión. Aprendió a mostrar la salida con anchuras de campo y precisiones. Presentamos las banderas `+`, `-`, espacio, `#`, `0`, coma y `(` que se utilizan en conjunto con los caracteres de formato para producir la salida. También demostramos

cómo aplicar formato a la salida con la clase `Formatter`. En el siguiente capítulo hablaremos sobre los métodos de la clase `String` para manipular cadenas. También presentaremos las expresiones regulares y demostraremos cómo validar la entrada del usuario mediante éstas.

Resumen

Sección 29.2 Flujos

- Por lo general, las operaciones de entrada y salida se llevan a cabo con flujos, los cuales son secuencias de bytes. En las operaciones de entrada, los bytes fluyen de un dispositivo a la memoria principal. En las operaciones de salida, los bytes fluyen de la memoria principal a un dispositivo.
- Por lo común, el flujo de entrada estándar se conecta al teclado, y el flujo de salida estándar se conecta a la pantalla de la computadora.

Sección 29.3 Aplicación de formato a la salida con `printf`

- La cadena de formato de `printf` describe los formatos en los que aparecen los valores de salida. El especificador de formato consiste en un índice como argumento, banderas, anchuras de campo, precisiones y caracteres de conversión.

Sección 29.4 Impresión de enteros

- Los enteros se imprimen con los caracteres de conversión `d` para enteros decimales, `o` para enteros en formato octal y `x` (o `X`) para los enteros en formato hexadecimal. Cuando se utiliza el carácter de conversión `X`, la salida se muestra en letras mayúsculas.

Sección 29.5 Impresión de números de punto flotante

- Los valores de punto flotante se imprimen con los caracteres de conversión `e` (o `E`) para la notación exponencial, `f` para la notación de punto flotante regular, y `g` (o `G`) para la notación `e` (o `E`) o `f`. Cuando se indica el especificador de conversión `g`, se utiliza el carácter de conversión `e` si el valor es menor que 10^{-3}, o mayor o igual a 10^{7}; en caso contrario, se utiliza el carácter de conversión `f`. Cuando se utilizan los caracteres de conversión `E` y `G`, la salida se muestra en letras mayúsculas.

Sección 29.6 Impresión de cadenas y caracteres

- El carácter de conversión `c` imprime un carácter.
- El carácter de conversión `s` (o `S`) imprime una cadena de caracteres. Cuando se utiliza el carácter de conversión `S`, la salida se muestra en letras mayúsculas.

Sección 29.7 Impresión de fechas y horas

- El carácter de conversión `t` (o `T`) seguido de un carácter de sufijo de conversión imprime la fecha y la hora en diversos formatos. Cuando se utiliza el carácter de conversión `T`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `t` (o `T`) requiere que el argumento sea de tipo `long`, `Long`, `Calendar` o `Date`.

Sección 29.8 Otros caracteres de conversión

- El carácter de conversión `b` (o `B`) imprime la representación de cadena de un valor `boolean` o `Boolean`. Estos caracteres de conversión también imprimen `"true"` para las referencias que no son `null`, y `"false"` para las referencias `null`. Cuando se utiliza el carácter de conversión `B`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `h` (o `H`) devuelve `null` para una referencia `null`, y una representación `String` del valor de código hash (en base 16) del objeto. Los códigos de hash se utilizan para almacenar y obtener objetos que se encuentran en objetos `Hashtable` y `HashMap`. Cuando se utiliza el carácter de conversión `H`, la salida se muestra en letras mayúsculas.
- El carácter de conversión `n` imprime el separador de línea específico de la plataforma.
- El carácter de conversión `%` se utiliza para mostrar un `%` literal.

Sección 29.9 Impresión con anchuras de campo y precisiones

- Si la anchura de campo es mayor que el objeto a imprimir, éste se justifica a la derecha en el campo.

- Las anchuras de campo se pueden usar con todos los caracteres de conversión, excepto la conversión con el separador de línea.
- La precisión que se utiliza con los caracteres de conversión de punto flotante `e` y `f` indica el número de dígitos que aparecen después del punto decimal. La precisión que se utiliza con el carácter de conversión de punto flotante `g` indica el número de dígitos significativos que deben aparecer.
- La precisión que se utiliza con el carácter de conversión `s` indica el número de caracteres a imprimir.
- La anchura de campo y la precisión se pueden combinar, para lo cual se coloca la anchura de campo, seguida de un punto decimal, seguido de la precisión entre el signo de porcentaje y el carácter de conversión.

Sección 29.10 Uso de banderas en la cadena de formato de `printf`

- La bandera – justifica a la izquierda su argumento en un campo.
- La bandera + imprime un signo más para los valores positivos, y un signo menos para los valores negativos.
- La bandera de espacio imprime un espacio antes de un valor positivo. La bandera de espacio y la bandera + no se pueden utilizar juntas en un carácter de conversión integral.
- La bandera # antepone un `0` a los valores octales, y `0x` a los valores hexadecimales.
- La bandera `0` imprime ceros a la izquierda para un valor que no ocupa todo su campo.
- La bandera de coma (`,`) utiliza el separador de miles específico para la configuración regional (por ejemplo, `','` para los EUA), para mostrar números enteros y de punto flotante.
- La bandera `(` encierra un número negativo entre paréntesis.

Sección 29.11 Impresión con índices como argumentos

- Un índice como argumento es un entero decimal opcional, seguido de un signo `$` que indica la posición del argumento en la lista de argumentos.
- Los índices como argumentos permiten a los programadores reordenar la salida, de manera que los argumentos en la lista de argumentos no estén necesariamente en el orden de sus correspondientes especificadores de formato. Los índices como argumentos también ayudan a evitar los argumentos duplicados.

Sección 29.13 Aplicación de formato a la salida con la clase `Formatter`

- La clase `Formatter` (en el paquete `java.util`) proporciona las mismas herramientas de formato que `printf`. `Formatter` es una clase utilitaria que permite a los programadores imprimir salida con formato hacia varios destinos, incluyendo componentes de GUI, archivos y otros flujos de salida.
- El método `format` de la clase `Formatter` imprime los datos con formato al destino especificado por el constructor de `Formatter`.
- El método `static format` de la clase `String` aplica formato a los datos y devuelve los datos con formato, como un objeto `String`.

Terminología

`#`, bandera
`%`, carácter de conversión
`(`, bandera
`+` (más), bandera
`-` (menos), bandera
`,` (coma), bandera
`0` (cero), bandera
`A`, carácter de sufijo de conversión
`a`, carácter de sufijo de conversión
alineación
anchura de campo
`b`, carácter de conversión
`B`, carácter de conversión
`B`, carácter de sufijo de conversión
`b`, carácter de sufijo de conversión
bandera
bandera de espacio
`c`, carácter de conversión
`C`, carácter de sufijo de conversión
cadena de formato
carácter de conversión
conversión de enteros
conversión de números de punto flotante
`d`, carácter de conversión
`D`, carácter de sufijo de conversión
`e`, carácter de conversión
`E`, carácter de conversión
`e`, carácter de sufijo de conversión
especificador de formato
`f`, carácter de conversión
`F`, carácter de sufijo de conversión
flujo
flujo de entrada estándar
flujo de error estándar

flujo de salida estándar
`format`, método de `Formatter`
`format`, método de `String`
formato de punto flotante exponencial
formato hexadecimal
formato octal
`Formatter`, clase
`g`, carácter de conversión
`G`, carácter de conversión
`h`, carácter de conversión
`H`, carácter de conversión
`H`, carácter de sufijo de conversión
`I`, carácter de sufijo de conversión
índice como argumento
`j`, carácter de sufijo de conversión
justificación a la derecha
justificación a la izquierda
`K`, carácter de sufijo de conversión
`m`, carácter de sufijo de conversión
`M`, carácter de sufijo de conversión
`n`, carácter de conversión
notación científica
`o`, carácter de conversión
`P`, carácter de sufijo de conversión
`p`, carácter de sufijo de conversión
precisión
`printf`, método
punto flotante
`r`, carácter de sufijo de conversión
redirigir un flujo
redondeo
`s`, carácter de conversión
`S`, carácter de conversión
`S`, carácter de sufijo de conversión
`t`, carácter de conversión
`T`, carácter de conversión
`T`, carácter de sufijo de conversión
`toString`, método de `Formatter`
`x`, carácter de conversión
`y`, carácter de sufijo de conversión
`Y`, carácter de sufijo de conversión
`Z`, carácter de sufijo de conversión

Ejercicios de autoevaluación

29.1 Complete los enunciados:

a) Todas las operaciones de entrada y salida se manejan en forma de _____.
b) El flujo _____ se conecta generalmente al teclado.
c) El flujo _____ se conecta generalmente a la pantalla de la computadora.
d) El método _____ de `System.out` se puede utilizar para aplicar formato al texto que se muestra en la salida estándar.
e) El carácter de conversión _____ puede utilizarse para imprimir en pantalla un entero decimal.
f) Los caracteres de conversión _____ y _____ se utilizan para mostrar enteros en formato octal y hexadecimal, respectivamente.
g) El carácter de conversión _____ se utiliza para mostrar un valor de punto flotante en notación exponencial
h) Los caracteres de conversión `e` y `f` se muestran con _____ dígitos de precisión a la derecha del punto decimal, si no se especifica una precisión.
i) Los caracteres de conversión _____ y _____ se utilizan para imprimir cadenas y caracteres, respectivamente.
j) El carácter de conversión _____ y el carácter de sufijo de conversión _____ se utilizan para imprimir la hora para el reloj de 24 horas, como `hora:minuto:segundo`.
k) La bandera _____ hace que la salida se justifique a la izquierda en un campo.
l) La bandera _____ hace que los valores se muestren con un signo más o con un signo menos.
m) El índice como argumento _____ corresponde al segundo argumento en la lista de argumentos.
n) La clase _____ tiene la misma capacidad que `printf`, pero permite a los programadores imprimir salida con formato en varios destinos, además del flujo de salida estándar.

29.2 Encuentre el error en cada uno de los siguientes enunciados, y explique cómo se puede corregir.

a) La siguiente instrucción debe imprimir el carácter `'c'`.
```
System.out.printf( "%c\n", "c" );
```
b) La siguiente instrucción debe imprimir `9.375%`.
```
System.out.printf( "%.3f%", 9.375 );
```

c) La siguiente instrucción debe imprimir el tercer argumento en la lista de argumentos:
```
System.out.printf( "%2$s\n", "Lun", "Mar", "Mie", "Jue", "Vie" );
```
d)
```
System.out.printf( ""Una cadena entre comillas"");
```
e)
```
System.out.printf( %d %d, 12, 20 );
```
f)
```
System.out.printf( "%s\n", 'Richard' );
```

29.3 Escriba una instrucción para cada uno de los siguientes casos:

a) Imprimir `1234` justificado a la derecha, en un campo de 10 dígitos.
b) Imprimir `123.456789` en notación exponencial con un signo (`+` o `-`) y 3 dígitos de precisión.
c) Imprimir `100` en formato octal, precedido por `0`.
d) Dado un objeto `Calendario` de la clase `Calendar`, imprima una fecha con formato de `mes/día/año` (cada uno con dos dígitos).
e) Dado un objeto `Calendar` llamado `calendario`, imprimir una hora para el reloj de 24 horas como `hora:minuto:segundo` (cada uno con dos dígitos), usando un índice como argumento y caracteres de sufijo de conversión para aplicar formato a la hora.
f) Imprimir `3.333333` con un signo (`+` o `-`) en un campo de 20 caracteres, con una precisión de 3.

Respuestas a los ejercicios de autoevaluación

29.1 a) Flujos. b) de entrada estándar. c) de salida estándar. d) `printf`. e) `d`. f) `o`, `x` o `X`. g) `e` o `E`. h) 6. i) `s` o `S`, `c` o `C`. j) `t`, `T`. k) – (menos). l) `+` (más). m) `2$`. n) `Formatter`.

29.2 a) Error: el carácter de conversión `c` espera un argumento del tipo primitivo `char`.
Corrección: para imprimir el carácter `'c'`, cambie `"c"` a `'c'`.
b) Error: está tratando de imprimir el carácter literal **%** sin usar el especificador de formato **%%**.
Corrección: use **%%** para imprimir un carácter **%** literal.
c) Error: el índice como argumento no empieza con `0`; por ejemplo, el primer argumento es `1$`.
Corrección: para imprimir el tercer argumento, use `3$`.
d) Error: está tratando de imprimir el carácter literal " sin usar la secuencia de escape \".
Corrección: sustituya cada comilla en el conjunto interno de comillas con \".
e) Error: la cadena de formato no va encerrada entre comillas dobles.
Corrección: encierre `%d %d` entre comillas dobles.
f) Error: la cadena a imprimir está encerrada entre comillas.
Corrección: use dobles comillas en vez de comillas sencillas para representar una cadena.

29.3 a)
```
System.out.printf( "%10d\n", 1234 );
```
b)
```
System.out.printf( "%+.3e\n", 123.456789 );
```
c)
```
System.out.printf( "%#o\n", 100 );
```
d)
```
System.out.printf( "%tD\n", calendario );
```
e)
```
System.out.printf( "%1$tH:%1$tM:%1$tS\n", calendario );
```
f)
```
System.out.printf( "%+20.3f\n", 3.333333 );
```

Ejercicios

29.4 Escriba una o más instrucciones para cada uno de los siguientes casos:

a) Imprimir el entero `40000` justificado a la derecha en un campo de 15 dígitos.
b) Imprimir `200` con y sin un signo.
c) Imprimir `100` en formato hexadecimal, precedido por `0x`.
d) Imprimir `1.234` con tres dígitos de precisión en un campo de nueve dígitos con ceros a la izquierda.

29.5 Muestre lo que se imprime en cada una de las siguientes instrucciones. Si una instrucción es incorrecta, indique por qué.

a)
```
System.out.printf( "%-10d\n", 10000 );
```
b)
```
System.out.printf( "%c\n", "Esta es una cadena" );
```
c)
```
System.out.printf( "%8.3f\n", 1024.987654 );
```
d)
```
System.out.printf( "%#o\n%#X\n", 17, 17 );
```
e)
```
System.out.printf( "% d\n%+d\n", 1000000, 1000000 );
```

f) `System.out.printf( "%10.2e\n", 444.93738 );`
g) `System.out.printf( "%d\n", 10.987 );`

29.6 Encuentre el(los) error(es) en cada uno de los siguientes segmentos de programa. Muestre la instrucción corregida.

a) `System.out.printf( "%s\n", 'Feliz cumpleanios' );`
b) `System.out.printf( "%c\n", 'Hola' );`
c) `System.out.printf( "%c\n", "Esta es una cadena" );`
d) La siguiente instrucción debe imprimir `"Buen viaje"` con las dobles comillas:
```
System.out.printf( ""%s"", "Buen viaje" );
```
e) La siguiente instrucción debe imprimir `"Hoy es viernes"`:
```
System.out.printf( "Hoy es %s\n", "Lunes", "Viernes" );
```
f) `System.out.printf( 'Escriba su nombre: ' );`
g) `System.out.printf( %f, 123.456 );`
h) La siguiente instrucción debe imprimir la hora actual en el formato `"hh:mm:ss"`:
```
Calendar fechaHora = Calendar.getInstance();
System.out.printf( "%1$tk:1$%tl:%1$tS\n", fechaHora );
```

29.7 *(Impresión de fechas y horas)* Escriba un programa que imprima fechas y horas en los siguientes formatos:

```
GMT-05:00 04/30/04 09:55:09 AM
GMT-05:00 Abril 30 2004 09:55:09
2004-04-30 dia-del-mes:30
2004-04-30 dia-del-anio: 121
Vie Abr 30 09:55:09 GMT-05:00  2004
```

[*Nota:* dependiendo de su ubicación, tal vez tenga una zona horaria distinta de GMT-05:00].

29.8 Escriba un programa para probar los resultados de imprimir el valor entero `12345` y el valor de punto flotante `1.2345` en campos de varios tamaños.

29.9 *(Redondeo de números)* Escriba un programa que imprima el valor `100.453267` redondeado al dígito, a la decena, centena, múltiplo de mil y de diez mil más cercanos.

29.10 Escriba un programa que reciba como entrada una palabra del teclado y determine su longitud. Imprima la palabra usando el doble de la longitud como anchura del campo.

29.11 *(Conversión de temperatura en grados Fahrenheit a Centígrados)* Escriba un programa que convierta temperaturas enteras en grados Fahrenheit de `0` a `212` grados, a temperaturas en grados Centígrados de punto flotante con tres dígitos de precisión. Use la siguiente fórmula:

```
centigrados = 5.0 / 9.0 * ( fahrenheit - 32 );
```

para realizar el cálculo. La salida debe imprimirse en dos columnas justificadas a la derecha de 10 caracteres cada una, y las temperaturas en grados Centígrados deben ir precedidas por un signo, tanto para los valores positivos como para los negativos.

29.12 Escriba un programa para probar todas las secuencias de escape en la figura 29.23. Para las secuencias de escape que desplazan el cursor, imprima un carácter antes y después de la secuencia de escape, de manera que se pueda ver con claridad a dónde se ha desplazado el cursor.

29.13 Escriba un programa que utilice el carácter de conversión `g` para imprimir el valor `9876.12345`. Imprima el valor con precisiones que varíen de `1` a `9`.