**Complete Printf for Java Format String Specification**

**1. especificadores de formato**

La sintaxis general de un especificador de formato es

%[ *flags* ] [ *ancho* ] [ .*precisión* ] [ *argsize* ] *typechar*

Especificadores de formato comienzan con un carácter de porcentaje (%) y terminan con un "carácter de tipo", que indica el tipo de datos ( int, float, etc.) que se convirtió a la forma básica en la que estarán representados los datos (decimal, hexadecimal, etc.) Por ejemplo, la cadena de formato

"I\_can\_lift\_%f\_tons".

indica que la salida formateada será la concatenación de la subcadena " I\_can\_lift\_ " , una representación de cadena de un float valor y la subcadena final " \_tons " . El vector de argumento correspondiente para esta cadena de formato contendría un número, que se interpretaría como el float.

Dependiendo del tipo de conversión, se pueden insertar ciertos indicadores y restricciones de formato *entre* el carácter de porcentaje y el carácter de tipo. Por ejemplo, al cambiar " % f " por " % .3f " en la cadena de formato anterior, se floatdebe formatear con una precisión de tres decimales después del punto decimal.

Los componentes de un especificador de formato se definen en las siguientes secciones.

**1.1. tipo de conversión**

*typechar* es un solo caracter que identifica el tipo de conversión. Los tipos de conversión soportados se enumeran a continuación, junto con sus significados, y los argumentos correspondientes esperados en el vector argumento:

|  |  |  |
| --- | --- | --- |
| **tipo personaje** | **Entrada** | **resultado de cadena** |
| %d | signo int | entero decimal con signo |
| %u | no Signo int | entero decimal sin signo |
| %o | no Signo int | entero sin signo octal |
| %x, %X | no Signo int | entero hexadecimal sin signo, minúscula o mayúscula |
| %z[*n*] , %Z[*n*] | No signo int | entero sin signo base *n* , con *n* codificado en decimal; incluir corchetes |
| %f | Float | número real, notación estándar |
| %e, %E | Float | número real, notación científica (marcador de exponente minúsculo o mayúscula) |
| %g, %G | Float | mismo formato que %fo %e, dependiendo del valor. La notación científica se usa solo si el exponente es mayor que la precisión o menor que -4. |
| %s | String | cuerda |
| %c | Char | personaje |
| %p | Object | código hash de identidad de objeto (es decir, valor de puntero), en hexadecimal sin signo |
| **especificadores de formato adicionales que no dan lugar a conversiones de argumento** | | |
| %\n | *(ninguna)* | separador de línea independiente de la plataforma (ver §3) |
| %n | *(nulo)* | cuenta caracteres (ver §4) |

El tipo de conversión no solo indica el tipo de cadena generada a partir de la entrada, sino también el tipo de entrada requerida.

|  |
| --- |
| **Ejemplo:** El especificador de formato " %o" significa que un intargumento se convertirá en una cadena octal sin signo. |

**1.2. modificador de tamaño de entrada**

Cada carácter de tipo de conversión indica que se proporciona un argumento de un tipo específico. Si no se proporciona el tipo de datos correcto para un especificador de formato, se produce un error, generalmente a ClassCastException. Sin embargo, es posible cambiar el tipo de datos requerido para ciertos especificadores de formato. Esto se hace precediendo al carácter de tipo con un "modificador de tamaño de entrada".

La siguiente tabla enumera los modificadores de tamaño de entrada permitidos:

|  |  |  |
| --- | --- | --- |
| **tamaño de entrada predeterminado** | **modificador de tamaño de entrada** | **tamaño de entrada real utilizado** |
| int | *(ninguna)* | int |
| int | h | short |
| int | b | byte |
| int | B | BigInteger |
| int | l | long |
| float | *(ninguna)* | float |
| float | l | double |
| float | B | BigDecimal |

|  |
| --- |
| **Ejemplo** : El especificador de formato " %d" indica que intse tratará como un número decimal con signo de 32 bits. Sin embargo, el especificador de formato " %hd" significa que a shortse convertirá en su lugar. |

**1.2.1. Formato BigInteger**

El %u, %xy %oespecificadores de formato típicamente representan *sin firmar* conversiones de números enteros. Esto es posible con tipos de datos primitivos, como int, porque los tamaños de los tipos son fijos y conocidos, lo que permite que cada bit se trate como un bit de magnitud.BigIntegers, sin embargo, representan patrones de bits "infinitamente largos", por lo que no es posible tratarlos como valores sin signo. Debido a esto, los especificadores de formato %Bx, %Boy %Buproducirán *firmaron* resultados. (Tenga en cuenta que tiene poco sentido usar %Bu, ya que es lo mismo que %Bd).

**1.3. especificador de ancho**

Un "especificador de ancho" opcional, si está presente, indica el campo con, o el número mínimo de caracteres en la salida que abarcará el argumento formateado. Si la representación de cadena del valor no llena la longitud mínima, el campo se rellenará con espacios. Sin embargo, si el valor convertido *excede* la longitud mínima, el resultado convertido no se truncará.

|  |
| --- |
| **Ejemplo:** si el especificador de formato es " %6d", y el intargumento proporcionado es 52, entonces la salida será "     52 " (cuatro espacios a la izquierda). |

Si no se especifica el ancho, entonces no hay un tamaño mínimo para el resultado convertido. El campo solo será tan grande como sea necesario para mostrar el resultado.

**1.3.1. ancho variable**

Es posible especificar un ancho de campo *variable* , donde uno de los argumentos en el vector argumento se usa para determinar el ancho del campo. Esto se indica en la cadena de formato utilizando un carácter de asterisco (' \*') en lugar del valor de ancho real. El valor correspondiente en la lista de argumentos debe ser an int, y debe preceder al argumento real que se está formateando.

|  |
| --- |
| **Ejemplo:** si el especificador de formato es " %\*hd", entonces hay dos argumentos en la lista de argumentos correspondientes a este especificador de formato. El primer argumento es un int, que indica el ancho del campo, y el segundo argumento se convierte como un firmado short. |

**1.4. especificador de precisión**

Se puede incluir un "especificador de precisión" opcional en un especificador de formato para indicar la precisión con la que convertir los datos. El significado de la palabra "precisión" depende del tipo de conversión que se realiza:

|  |  |  |
| --- | --- | --- |
| **tipo de conversión** | **significado de "precisión"** | **valor predeterminado  (si se omite)** |
| %d, %o, %u, %x, %X, %z[*N*] , %Z[*n*] (conversiones entero) | número mínimo de dígitos El valor convertido se antepondrá con ceros si es necesario. Tenga en cuenta que si la precisión es 0 y el valor es cero, no se imprimirá nada, o todo el campo no será nada más que relleno. | 1 |
| %f, %e, %E, %g, %G(Conversiones reales) | número de dígitos fraccionarios después del punto decimal. El valor convertido se redondeará si es necesario. | 6 |
| %s (instrumentos de cuerda) | la cantidad máxima de caracteres Si la cadena es demasiado larga, se truncará. | infinito |

Si no se especifica precisión, se usará la precisión predeterminada. La precisión no se puede especificar para tipos de conversión distintos de los enumerados anteriormente.

**1.4.1. precisión variable**

Al igual que el campo de ancho, el campo de precisión también se puede hacer *variable* reemplazando el número con un asterisco. El argumento esperado es un int. (Consulte la sección sobre "ancho variable" para más detalles.) Si se usan tanto el ancho variable como la precisión variable en el mismo especificador de formato, el argumento de ancho aparece primero en la lista de argumentos.

**1.5. banderas**

Los indicadores son caracteres únicos que indican excepciones al comportamiento predeterminado del tipo de conversión. Un especificador de formato puede tener múltiples indicadores, pero algunos indicadores son mutuamente excluyentes. Múltiples banderas pueden aparecer en cualquier orden. La siguiente tabla enumera los indicadores de formato admitidos por las clases de printf de Lava:

|  |  |  |
| --- | --- | --- |
| **bandera** | **sentido** | **tipos de conversión aplicables** |
| ' -' | El resultado se alinea a la izquierda en el campo. Esta bandera no tiene sentido si no se especifica un ancho de campo obligatorio. | %d, %u, %o, %x, %X, %z[*N*] , %Z[*n*] , %f, %e, %E, %g, %G, %s, %c,%p |
| ' ^' | El resultado se centra en el campo. Esta bandera no tiene sentido si no se especifica un ancho de campo obligatorio. |
| ' +' | Los valores no negativos comienzan con un carácter más ('+'). | %d, %f |
| '  ' | Los valores no negativos comienzan con un carácter de espacio (''). Este indicador solo es útil para los resultados de conversión firmados (% d y% f). |
| ' #' | El argumento se representa en una "forma alternativa". Esto depende del tipo de conversión:   |  |  | | --- | --- | | %o | Los valores octales no negativos se anteponen con un cero ('0'). | | %x, %X | Los valores hexadecimales se anteponen con el prefijo "0x" o "0X". | | %e, %E,%f | La parte entera del resultado siempre termina con un punto decimal ('.'), Incluso si la porción fraccional es cero. | | %g, %G | La porción fraccionaria siempre aparece, incluso si es cero. | | %c | Si el carácter es especial o no imprimible, se imprime en forma de escape. El resultado puede estar rodeado por comillas simples para formar un literal de carácter Java sintácticamente válido. |   No hay forma alternativa para %s, %d, %u, %z[*n*] , y %Z[*n*] . | |

**2. vectores de argumento**

Un vector de argumento es una lista de elementos de datos que se convertirán en cadenas y se insertarán en la salida de impresión, de acuerdo con los criterios de formato especificados en la cadena de formato. Cada especificador de formato en la cadena de formato requiere un número específico y el tipo de argumentos correspondientes en la lista. Dado que Java permite métodos con listas de argumentos de longitud variable o de tipo variable, todos los argumentos deben estar encapsulados en una Objectmatriz genérica , que luego se pasa como un único parámetro.

Desafortunadamente, las Objectmatrices no pueden contener tipos de datos primitivos, como into long. Por lo tanto, para incluir primitivas en una lista de argumentos, las Objectencapsulaciones deben usarse en su lugar. La siguiente tabla enumera los tipos de entrada requeridos por los especificadores de formato y las clases que se deben usar para incluirlos en la lista de argumentos:

|  |  |  |  |
| --- | --- | --- | --- |
| **tipo de entrada** | **tipo de objeto recomendado** | **tipo de objeto requerido** | **método llamado para obtener entrada** |
| byte | java.lang.Byte | java.lang.Number | byteValue() |
| short | java.lang.Short | java.lang.Number | shortValue() |
| int | java.lang.Integer | java.lang.Number | intValue() |
| long | java.lang.Long | java.lang.Number | longValue() |
| float | java.lang.Float | java.lang.Number | floatValue() |
| double | java.lang.Double | java.lang.Number | doubleValue() |
| char | java.lang.Character | java.lang.Character | charValue() |
| java.lang.BigInteger | java.lang.BigInteger | java.lang.BigInteger | n / A |
| java.lang.BigDecimal | java.lang.BigDecimal | java.lang.BigDecimal | n / A |
| java.lang.String | java.lang.String | java.lang.Object | toString() |

|  |
| --- |
| **Ejemplo:** si el especificador de formato es " %f", entonces el argumento correspondiente en la Objectmatriz debe ser un Floatobjeto. |

|  |
| --- |
| **Ejemplo:** si la cadena de formato es  " My friend %s is %d years old."  entonces la lista de argumentos correspondiente debe consistir en a Stringy an int. Suponiendo que namees a Stringy agees un int, entonces la lista de argumentos correspondiente podría ser creada usando la expresión  new Object [] {name, new Integer (age)}  En esta expresión, el primer elemento de la Objectmatriz es a String, que corresponde al primer especificador de formato, " %s". El segundo elemento de la matriz es an Integer, que corresponde al segundo especificador de formato, " %d". |

**2.1. tipos numéricos abstractos**

A pesar de las recomendaciones dadas en la sección anterior con respecto a la encapsulación de tipo primitivo, en realidad es legal usar cualquier instancia de java.lang.Numberdonde los tipos de entrada

int, long, short, byte, float, Odouble

son esperados. Esto se debe a que el motor de formateo llamará correctamente al método abstracto apropiado,

intValue(), longValue(), shortValue(), byteValue(), floatValue(), O doubleValue(),

respectivamente, para obtener los datos necesarios. Este comportamiento proporciona una medida de flexibilidad que pueden disfrutar los programadores avanzados, incluida la capacidad de inventar y formatear los propios tipos de datos numéricos.

**2.2. conversión automática de cadenas**

Siempre que Stringse necesita una instancia de la lista de argumentos, es seguro usar *cualquier* tipo de objeto. Esto se debe a que el motor de formateo llamará automáticamente al toString()método del objeto (que, para un Stringobjeto, simplemente devuelve una autorreferencia). Esto facilita el formateo de representaciones de cadenas de todo tipo de objetos. También hace que sea legal proporcionar a StringBuffer, en lugar de a String, el parámetro correspondiente para %s.

**3. separadores de línea independientes de la plataforma**

En situaciones donde el software Java se ejecutará en más de una plataforma, es aconsejable evitar el uso de caracteres de control dependientes del sistema, como '\ n'. Las clases de impresión de Lava admiten la noción de un "separador de línea independiente de plataforma", representado en cadenas de formato por el " %\n" especificador de formato. Dondequiera que %\n ocurra en una cadena de formato, las clases de printf insertarán automáticamente el separador de línea nativo del sistema host.

Tenga en cuenta que si la compatibilidad multiplataforma no es un problema, y ​​el software se ejecutará principalmente en hosts basados ​​en Unix y Windows, el uso del carácter tradicional de nueva línea \nprobablemente produzca resultados aceptables.

**4. conteo de personajes**

Un especificador de formato especial aún no mencionado, " %n", no requiere ningún dato de la lista de argumentos ni produce ninguna salida. En cambio, cuando este especificador aparece en la cadena de formato, se calcula el número de caracteres formateados hasta el punto donde aparece y el Integervalor resultante se escribe en el vector de argumento proporcionado. Por lo tanto, al compilar la lista de argumentos para una cadena de formato que contiene este especificador, se debe reservar un espacio adicional para el resultado. Esta es la única instancia en la que el vector argumento será modificado por una llamada printf.

|  |
| --- |
| **Ejemplo:** Este programa imprime la cadena " Mi amigo Sharky tiene 27 años " . Utiliza %npara obtener posiciones de personajes para que pueda subrayar el número 27.  // Estos son los parámetros:  String name = "Sharky";  int edad = 27;  // Este es el argumento vectorial. Observe que nos estamos yendo  // slots para que el especificador de formato% n coloque sus resultados:  Object [] args = new Object [] {name, null, new Integer (age), null};  // Ahora produciremos la primera línea de salida:  Stdio.printf ("Mi amigo% s es% n% d% n años viejo. \ N", args);  // A continuación, usaremos los resultados de% n para formar una cadena de subrayado:  int start = ((Entero) args [1]). intValue ();  int stop = ((Entero) args [3]). intValue ();  String underline\_string =  StringToolbox.repeat ('', inicio) +  StringToolbox.repeat ('-', stop - start);  // Por último, imprimimos la cadena de subrayado. Tenga en cuenta que, a diferencia del  // arriba de la operación de formateo, esta vez formamos el vector de argumento  // en línea. Esto se debe a que no necesitaremos acceder a su contenido después  // lo hemos usado, como hicimos anteriormente:  Stdio.printf ("% s \ n", nuevo Object [] {underline\_string});  **Resultados:**  Mi amigo Sharky tiene 27 años.  - |