| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DecimalFormat.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/text/DateFormatSymbols.html)   [**NEXT CLASS**](http://docs.google.com/java/text/DecimalFormatSymbols.html) | [**FRAMES**](http://docs.google.com/index.html?java/text/DecimalFormat.html)    [**NO FRAMES**](http://docs.google.com/DecimalFormat.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#tyjcwt) | [FIELD](#1t3h5sf) | [CONSTR](#4d34og8) | [METHOD](#2s8eyo1) | DETAIL: FIELD | [CONSTR](#lnxbz9) | [METHOD](#2jxsxqh) |

## **java.text**

Class DecimalFormat

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.text.Format](http://docs.google.com/java/text/Format.html)  
 [java.text.NumberFormat](http://docs.google.com/java/text/NumberFormat.html)  
 **java.text.DecimalFormat**

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html)

public class **DecimalFormat**extends [NumberFormat](http://docs.google.com/java/text/NumberFormat.html)

DecimalFormat is a concrete subclass of NumberFormat that formats decimal numbers. It has a variety of features designed to make it possible to parse and format numbers in any locale, including support for Western, Arabic, and Indic digits. It also supports different kinds of numbers, including integers (123), fixed-point numbers (123.4), scientific notation (1.23E4), percentages (12%), and currency amounts ($123). All of these can be localized.

To obtain a NumberFormat for a specific locale, including the default locale, call one of NumberFormat's factory methods, such as getInstance(). In general, do not call the DecimalFormat constructors directly, since the NumberFormat factory methods may return subclasses other than DecimalFormat. If you need to customize the format object, do something like this:

NumberFormat f = NumberFormat.getInstance(loc);  
 if (f instanceof DecimalFormat) {  
 ((DecimalFormat) f).setDecimalSeparatorAlwaysShown(true);  
 }

A DecimalFormat comprises a *pattern* and a set of *symbols*. The pattern may be set directly using applyPattern(), or indirectly using the API methods. The symbols are stored in a DecimalFormatSymbols object. When using the NumberFormat factory methods, the pattern and symbols are read from localized ResourceBundles.

#### Patterns

DecimalFormat patterns have the following syntax:

*Pattern:*  
 *PositivePattern*  
 *PositivePattern* ; *NegativePattern*  
 *PositivePattern:*  
 *Prefixopt* *Number* *Suffixopt*  
 *NegativePattern:*  
 *Prefixopt* *Number* *Suffixopt*  
 *Prefix:*  
 any Unicode characters except \uFFFE, \uFFFF, and special characters  
 *Suffix:*  
 any Unicode characters except \uFFFE, \uFFFF, and special characters  
 *Number:*  
 *Integer* *Exponentopt*  
 *Integer* . *Fraction* *Exponentopt*  
 *Integer:*  
 *MinimumInteger*  
 #  
 # *Integer*  
 # , *Integer*  
 *MinimumInteger:*  
 0  
 0 *MinimumInteger*  
 0 , *MinimumInteger*  
 *Fraction:*  
 *MinimumFractionopt* *OptionalFractionopt*  
 *MinimumFraction:*  
 0 *MinimumFractionopt*  
 *OptionalFraction:*  
 # *OptionalFractionopt*  
 *Exponent:*  
 E *MinimumExponent*  
 *MinimumExponent:*  
 0 *MinimumExponentopt*

A DecimalFormat pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional; if absent, then the positive subpattern prefixed with the localized minus sign ('-' in most locales) is used as the negative subpattern. That is, "0.00" alone is equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0#;(#)" produces precisely the same behavior as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the positive and negative prefixes or the suffixes must be distinct for DecimalFormat.parse() to be able to distinguish positive from negative values. (If they are identical, then DecimalFormat will behave as if no negative subpattern was specified.) Another example is that the decimal separator and thousands separator should be distinct characters, or parsing will be impossible.

The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands. The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last one and the end of the integer is the one that is used. So "#,##,###,####" == "######,####" == "##,####,####".

#### Special Pattern Characters

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as literals.

The characters listed here are used in non-localized patterns. Localized patterns use the corresponding characters taken from this formatter's DecimalFormatSymbols object instead, and these characters lose their special status. Two exceptions are the currency sign and quote, which are not localized.

| Symbol | Location | Localized? | Meaning |
| --- | --- | --- | --- |
| 0 | Number | Yes | Digit |
| # | Number | Yes | Digit, zero shows as absent |
| . | Number | Yes | Decimal separator or monetary decimal separator |
| - | Number | Yes | Minus sign |
| , | Number | Yes | Grouping separator |
| E | Number | Yes | Separates mantissa and exponent in scientific notation. *Need not be quoted in prefix or suffix.* |
| ; | Subpattern boundary | Yes | Separates positive and negative subpatterns |
| % | Prefix or suffix | Yes | Multiply by 100 and show as percentage |
| \u2030 | Prefix or suffix | Yes | Multiply by 1000 and show as per mille value |
| ¤ (\u00A4) | Prefix or suffix | No | Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator. |
| ' | Prefix or suffix | No | Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock". |

#### Scientific Notation

Numbers in scientific notation are expressed as the product of a mantissa and a power of ten, for example, 1234 can be expressed as 1.234 x 10^3. The mantissa is often in the range 1.0 <= x < 10.0, but it need not be. DecimalFormat can be instructed to format and parse scientific notation *only via a pattern*; there is currently no factory method that creates a scientific notation format. In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation. Example: "0.###E0" formats the number 1234 as "1.234E3".

* The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum. Negative exponents are formatted using the localized minus sign, *not* the prefix and suffix from the pattern. This allows patterns such as "0.###E0 m/s".
* The minimum and maximum number of integer digits are interpreted together:
  + If the maximum number of integer digits is greater than their minimum number and greater than 1, it forces the exponent to be a multiple of the maximum number of integer digits, and the minimum number of integer digits to be interpreted as 1. The most common use of this is to generate *engineering notation*, in which the exponent is a multiple of three, e.g., "##0.#####E0". Using this pattern, the number 12345 formats to "12.345E3", and 123456 formats to "123.456E3".
  + Otherwise, the minimum number of integer digits is achieved by adjusting the exponent. Example: 0.00123 formatted with "00.###E0" yields "12.3E-4".
* The number of significant digits in the mantissa is the sum of the *minimum integer* and *maximum fraction* digits, and is unaffected by the maximum integer digits. For example, 12345 formatted with "##0.##E0" is "12.3E3". To show all digits, set the significant digits count to zero. The number of significant digits does not affect parsing.
* Exponential patterns may not contain grouping separators.

#### Rounding

DecimalFormat provides rounding modes defined in [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) for formatting. By default, it uses [RoundingMode.HALF\_EVEN](http://docs.google.com/java/math/RoundingMode.html#HALF_EVEN).

#### Digits

For formatting, DecimalFormat uses the ten consecutive characters starting with the localized zero digit defined in the DecimalFormatSymbols object as digits. For parsing, these digits as well as all Unicode decimal digits, as defined by [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), are recognized.

#### Special Values

NaN is formatted as a string, which typically has a single character \uFFFD. This string is determined by the DecimalFormatSymbols object. This is the only value for which the prefixes and suffixes are not used.

Infinity is formatted as a string, which typically has a single character \u221E, with the positive or negative prefixes and suffixes applied. The infinity string is determined by the DecimalFormatSymbols object.

Negative zero ("-0") parses to

* BigDecimal(0) if isParseBigDecimal() is true,
* Long(0) if isParseBigDecimal() is false and isParseIntegerOnly() is true,
* Double(-0.0) if both isParseBigDecimal() and isParseIntegerOnly() are false.

#### Synchronization

Decimal formats are generally not synchronized. It is recommended to create separate format instances for each thread. If multiple threads access a format concurrently, it must be synchronized externally.

#### Example

**// Print out a number using the localized number, integer, currency,  
 // and percent format for each locale**  
 Locale[] locales = NumberFormat.getAvailableLocales();  
 double myNumber = -1234.56;  
 NumberFormat form;  
 for (int j=0; j<4; ++j) {  
 System.out.println("FORMAT");  
 for (int i = 0; i < locales.length; ++i) {  
 if (locales[i].getCountry().length() == 0) {  
 continue; // Skip language-only locales  
 }  
 System.out.print(locales[i].getDisplayName());  
 switch (j) {  
 case 0:  
 form = NumberFormat.getInstance(locales[i]); break;  
 case 1:  
 form = NumberFormat.getIntegerInstance(locales[i]); break;  
 case 2:  
 form = NumberFormat.getCurrencyInstance(locales[i]); break;  
 default:  
 form = NumberFormat.getPercentInstance(locales[i]); break;  
 }  
 if (form instanceof DecimalFormat) {  
 System.out.print(": " + ((DecimalFormat) form).toPattern());  
 }  
 System.out.print(" -> " + form.format(myNumber));  
 try {  
 System.out.println(" -> " + form.parse(form.format(myNumber)));  
 } catch (ParseException e) {}  
 }  
 }

**See Also:**[Java Tutorial](http://java.sun.com/docs/books/tutorial/i18n/format/decimalFormat.html), [NumberFormat](http://docs.google.com/java/text/NumberFormat.html), [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html), [ParsePosition](http://docs.google.com/java/text/ParsePosition.html), [Serialized Form](http://docs.google.com/serialized-form.html#java.text.DecimalFormat)

| **Nested Class Summary** | |
| --- | --- |

| **Nested classes/interfaces inherited from class java.text.**[**NumberFormat**](http://docs.google.com/java/text/NumberFormat.html) |
| --- |
| [NumberFormat.Field](http://docs.google.com/java/text/NumberFormat.Field.html) |

| **Field Summary** | |
| --- | --- |

| **Fields inherited from class java.text.**[**NumberFormat**](http://docs.google.com/java/text/NumberFormat.html) |
| --- |
| [FRACTION\_FIELD](http://docs.google.com/java/text/NumberFormat.html#FRACTION_FIELD), [INTEGER\_FIELD](http://docs.google.com/java/text/NumberFormat.html#INTEGER_FIELD) |

| **Constructor Summary** | |
| --- | --- |
| [**DecimalFormat**](http://docs.google.com/java/text/DecimalFormat.html#DecimalFormat())()            Creates a DecimalFormat using the default pattern and symbols for the default locale. |
| [**DecimalFormat**](http://docs.google.com/java/text/DecimalFormat.html#DecimalFormat(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Creates a DecimalFormat using the given pattern and the symbols for the default locale. |
| [**DecimalFormat**](http://docs.google.com/java/text/DecimalFormat.html#DecimalFormat(java.lang.String,%20java.text.DecimalFormatSymbols))([String](http://docs.google.com/java/lang/String.html) pattern, [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) symbols)            Creates a DecimalFormat using the given pattern and symbols. |

| **Method Summary** | |
| --- | --- |
| void | [**applyLocalizedPattern**](http://docs.google.com/java/text/DecimalFormat.html#applyLocalizedPattern(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Apply the given pattern to this Format object. |
| void | [**applyPattern**](http://docs.google.com/java/text/DecimalFormat.html#applyPattern(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Apply the given pattern to this Format object. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**clone**](http://docs.google.com/java/text/DecimalFormat.html#clone())()            Standard override; no change in semantics. |
| boolean | [**equals**](http://docs.google.com/java/text/DecimalFormat.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Overrides equals |
| [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) | [**format**](http://docs.google.com/java/text/DecimalFormat.html#format(double,%20java.lang.StringBuffer,%20java.text.FieldPosition))(double number, [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) result, [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) fieldPosition)            Formats a double to produce a string. |
| [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) | [**format**](http://docs.google.com/java/text/DecimalFormat.html#format(long,%20java.lang.StringBuffer,%20java.text.FieldPosition))(long number, [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) result, [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) fieldPosition)            Format a long to produce a string. |
| [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) | [**format**](http://docs.google.com/java/text/DecimalFormat.html#format(java.lang.Object,%20java.lang.StringBuffer,%20java.text.FieldPosition))([Object](http://docs.google.com/java/lang/Object.html) number, [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) toAppendTo, [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) pos)            Formats a number and appends the resulting text to the given string buffer. |
| [AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) | [**formatToCharacterIterator**](http://docs.google.com/java/text/DecimalFormat.html#formatToCharacterIterator(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Formats an Object producing an AttributedCharacterIterator. |
| [Currency](http://docs.google.com/java/util/Currency.html) | [**getCurrency**](http://docs.google.com/java/text/DecimalFormat.html#getCurrency())()            Gets the currency used by this decimal format when formatting currency values. |
| [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) | [**getDecimalFormatSymbols**](http://docs.google.com/java/text/DecimalFormat.html#getDecimalFormatSymbols())()            Returns a copy of the decimal format symbols, which is generally not changed by the programmer or user. |
| int | [**getGroupingSize**](http://docs.google.com/java/text/DecimalFormat.html#getGroupingSize())()            Return the grouping size. |
| int | [**getMaximumFractionDigits**](http://docs.google.com/java/text/DecimalFormat.html#getMaximumFractionDigits())()            Gets the maximum number of digits allowed in the fraction portion of a number. |
| int | [**getMaximumIntegerDigits**](http://docs.google.com/java/text/DecimalFormat.html#getMaximumIntegerDigits())()            Gets the maximum number of digits allowed in the integer portion of a number. |
| int | [**getMinimumFractionDigits**](http://docs.google.com/java/text/DecimalFormat.html#getMinimumFractionDigits())()            Gets the minimum number of digits allowed in the fraction portion of a number. |
| int | [**getMinimumIntegerDigits**](http://docs.google.com/java/text/DecimalFormat.html#getMinimumIntegerDigits())()            Gets the minimum number of digits allowed in the integer portion of a number. |
| int | [**getMultiplier**](http://docs.google.com/java/text/DecimalFormat.html#getMultiplier())()            Gets the multiplier for use in percent, per mille, and similar formats. |
| [String](http://docs.google.com/java/lang/String.html) | [**getNegativePrefix**](http://docs.google.com/java/text/DecimalFormat.html#getNegativePrefix())()            Get the negative prefix. |
| [String](http://docs.google.com/java/lang/String.html) | [**getNegativeSuffix**](http://docs.google.com/java/text/DecimalFormat.html#getNegativeSuffix())()            Get the negative suffix. |
| [String](http://docs.google.com/java/lang/String.html) | [**getPositivePrefix**](http://docs.google.com/java/text/DecimalFormat.html#getPositivePrefix())()            Get the positive prefix. |
| [String](http://docs.google.com/java/lang/String.html) | [**getPositiveSuffix**](http://docs.google.com/java/text/DecimalFormat.html#getPositiveSuffix())()            Get the positive suffix. |
| [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) | [**getRoundingMode**](http://docs.google.com/java/text/DecimalFormat.html#getRoundingMode())()            Gets the [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) used in this DecimalFormat. |
| int | [**hashCode**](http://docs.google.com/java/text/DecimalFormat.html#hashCode())()            Overrides hashCode |
| boolean | [**isDecimalSeparatorAlwaysShown**](http://docs.google.com/java/text/DecimalFormat.html#isDecimalSeparatorAlwaysShown())()            Allows you to get the behavior of the decimal separator with integers. |
| boolean | [**isParseBigDecimal**](http://docs.google.com/java/text/DecimalFormat.html#isParseBigDecimal())()            Returns whether the [parse(java.lang.String, java.text.ParsePosition)](http://docs.google.com/java/text/DecimalFormat.html#parse(java.lang.String,%20java.text.ParsePosition)) method returns BigDecimal. |
| [Number](http://docs.google.com/java/lang/Number.html) | [**parse**](http://docs.google.com/java/text/DecimalFormat.html#parse(java.lang.String,%20java.text.ParsePosition))([String](http://docs.google.com/java/lang/String.html) text, [ParsePosition](http://docs.google.com/java/text/ParsePosition.html) pos)            Parses text from a string to produce a Number. |
| void | [**setCurrency**](http://docs.google.com/java/text/DecimalFormat.html#setCurrency(java.util.Currency))([Currency](http://docs.google.com/java/util/Currency.html) currency)            Sets the currency used by this number format when formatting currency values. |
| void | [**setDecimalFormatSymbols**](http://docs.google.com/java/text/DecimalFormat.html#setDecimalFormatSymbols(java.text.DecimalFormatSymbols))([DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) newSymbols)            Sets the decimal format symbols, which is generally not changed by the programmer or user. |
| void | [**setDecimalSeparatorAlwaysShown**](http://docs.google.com/java/text/DecimalFormat.html#setDecimalSeparatorAlwaysShown(boolean))(boolean newValue)            Allows you to set the behavior of the decimal separator with integers. |
| void | [**setGroupingSize**](http://docs.google.com/java/text/DecimalFormat.html#setGroupingSize(int))(int newValue)            Set the grouping size. |
| void | [**setMaximumFractionDigits**](http://docs.google.com/java/text/DecimalFormat.html#setMaximumFractionDigits(int))(int newValue)            Sets the maximum number of digits allowed in the fraction portion of a number. |
| void | [**setMaximumIntegerDigits**](http://docs.google.com/java/text/DecimalFormat.html#setMaximumIntegerDigits(int))(int newValue)            Sets the maximum number of digits allowed in the integer portion of a number. |
| void | [**setMinimumFractionDigits**](http://docs.google.com/java/text/DecimalFormat.html#setMinimumFractionDigits(int))(int newValue)            Sets the minimum number of digits allowed in the fraction portion of a number. |
| void | [**setMinimumIntegerDigits**](http://docs.google.com/java/text/DecimalFormat.html#setMinimumIntegerDigits(int))(int newValue)            Sets the minimum number of digits allowed in the integer portion of a number. |
| void | [**setMultiplier**](http://docs.google.com/java/text/DecimalFormat.html#setMultiplier(int))(int newValue)            Sets the multiplier for use in percent, per mille, and similar formats. |
| void | [**setNegativePrefix**](http://docs.google.com/java/text/DecimalFormat.html#setNegativePrefix(java.lang.String))([String](http://docs.google.com/java/lang/String.html) newValue)            Set the negative prefix. |
| void | [**setNegativeSuffix**](http://docs.google.com/java/text/DecimalFormat.html#setNegativeSuffix(java.lang.String))([String](http://docs.google.com/java/lang/String.html) newValue)            Set the negative suffix. |
| void | [**setParseBigDecimal**](http://docs.google.com/java/text/DecimalFormat.html#setParseBigDecimal(boolean))(boolean newValue)            Sets whether the [parse(java.lang.String, java.text.ParsePosition)](http://docs.google.com/java/text/DecimalFormat.html#parse(java.lang.String,%20java.text.ParsePosition)) method returns BigDecimal. |
| void | [**setPositivePrefix**](http://docs.google.com/java/text/DecimalFormat.html#setPositivePrefix(java.lang.String))([String](http://docs.google.com/java/lang/String.html) newValue)            Set the positive prefix. |
| void | [**setPositiveSuffix**](http://docs.google.com/java/text/DecimalFormat.html#setPositiveSuffix(java.lang.String))([String](http://docs.google.com/java/lang/String.html) newValue)            Set the positive suffix. |
| void | [**setRoundingMode**](http://docs.google.com/java/text/DecimalFormat.html#setRoundingMode(java.math.RoundingMode))([RoundingMode](http://docs.google.com/java/math/RoundingMode.html) roundingMode)            Sets the [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) used in this DecimalFormat. |
| [String](http://docs.google.com/java/lang/String.html) | [**toLocalizedPattern**](http://docs.google.com/java/text/DecimalFormat.html#toLocalizedPattern())()            Synthesizes a localized pattern string that represents the current state of this Format object. |
| [String](http://docs.google.com/java/lang/String.html) | [**toPattern**](http://docs.google.com/java/text/DecimalFormat.html#toPattern())()            Synthesizes a pattern string that represents the current state of this Format object. |

| **Methods inherited from class java.text.**[**NumberFormat**](http://docs.google.com/java/text/NumberFormat.html) |
| --- |
| [format](http://docs.google.com/java/text/NumberFormat.html#format(double)), [format](http://docs.google.com/java/text/NumberFormat.html#format(long)), [getAvailableLocales](http://docs.google.com/java/text/NumberFormat.html#getAvailableLocales()), [getCurrencyInstance](http://docs.google.com/java/text/NumberFormat.html#getCurrencyInstance()), [getCurrencyInstance](http://docs.google.com/java/text/NumberFormat.html#getCurrencyInstance(java.util.Locale)), [getInstance](http://docs.google.com/java/text/NumberFormat.html#getInstance()), [getInstance](http://docs.google.com/java/text/NumberFormat.html#getInstance(java.util.Locale)), [getIntegerInstance](http://docs.google.com/java/text/NumberFormat.html#getIntegerInstance()), [getIntegerInstance](http://docs.google.com/java/text/NumberFormat.html#getIntegerInstance(java.util.Locale)), [getNumberInstance](http://docs.google.com/java/text/NumberFormat.html#getNumberInstance()), [getNumberInstance](http://docs.google.com/java/text/NumberFormat.html#getNumberInstance(java.util.Locale)), [getPercentInstance](http://docs.google.com/java/text/NumberFormat.html#getPercentInstance()), [getPercentInstance](http://docs.google.com/java/text/NumberFormat.html#getPercentInstance(java.util.Locale)), [isGroupingUsed](http://docs.google.com/java/text/NumberFormat.html#isGroupingUsed()), [isParseIntegerOnly](http://docs.google.com/java/text/NumberFormat.html#isParseIntegerOnly()), [parse](http://docs.google.com/java/text/NumberFormat.html#parse(java.lang.String)), [parseObject](http://docs.google.com/java/text/NumberFormat.html#parseObject(java.lang.String,%20java.text.ParsePosition)), [setGroupingUsed](http://docs.google.com/java/text/NumberFormat.html#setGroupingUsed(boolean)), [setParseIntegerOnly](http://docs.google.com/java/text/NumberFormat.html#setParseIntegerOnly(boolean)) |

| **Methods inherited from class java.text.**[**Format**](http://docs.google.com/java/text/Format.html) |
| --- |
| [format](http://docs.google.com/java/text/Format.html#format(java.lang.Object)), [parseObject](http://docs.google.com/java/text/Format.html#parseObject(java.lang.String)) |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### DecimalFormat

public **DecimalFormat**()

Creates a DecimalFormat using the default pattern and symbols for the default locale. This is a convenient way to obtain a DecimalFormat when internationalization is not the main concern.

To obtain standard formats for a given locale, use the factory methods on NumberFormat such as getNumberInstance. These factories will return the most appropriate sub-class of NumberFormat for a given locale.

**See Also:**[NumberFormat.getInstance()](http://docs.google.com/java/text/NumberFormat.html#getInstance()), [NumberFormat.getNumberInstance()](http://docs.google.com/java/text/NumberFormat.html#getNumberInstance()), [NumberFormat.getCurrencyInstance()](http://docs.google.com/java/text/NumberFormat.html#getCurrencyInstance()), [NumberFormat.getPercentInstance()](http://docs.google.com/java/text/NumberFormat.html#getPercentInstance())

### DecimalFormat

public **DecimalFormat**([String](http://docs.google.com/java/lang/String.html) pattern)

Creates a DecimalFormat using the given pattern and the symbols for the default locale. This is a convenient way to obtain a DecimalFormat when internationalization is not the main concern.

To obtain standard formats for a given locale, use the factory methods on NumberFormat such as getNumberInstance. These factories will return the most appropriate sub-class of NumberFormat for a given locale.

**Parameters:**pattern - A non-localized pattern string. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if pattern is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given pattern is invalid.**See Also:**[NumberFormat.getInstance()](http://docs.google.com/java/text/NumberFormat.html#getInstance()), [NumberFormat.getNumberInstance()](http://docs.google.com/java/text/NumberFormat.html#getNumberInstance()), [NumberFormat.getCurrencyInstance()](http://docs.google.com/java/text/NumberFormat.html#getCurrencyInstance()), [NumberFormat.getPercentInstance()](http://docs.google.com/java/text/NumberFormat.html#getPercentInstance())

### DecimalFormat

public **DecimalFormat**([String](http://docs.google.com/java/lang/String.html) pattern,  
 [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) symbols)

Creates a DecimalFormat using the given pattern and symbols. Use this constructor when you need to completely customize the behavior of the format.

To obtain standard formats for a given locale, use the factory methods on NumberFormat such as getInstance or getCurrencyInstance. If you need only minor adjustments to a standard format, you can modify the format returned by a NumberFormat factory method.

**Parameters:**pattern - a non-localized pattern stringsymbols - the set of symbols to be used **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if any of the given arguments is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given pattern is invalid**See Also:**[NumberFormat.getInstance()](http://docs.google.com/java/text/NumberFormat.html#getInstance()), [NumberFormat.getNumberInstance()](http://docs.google.com/java/text/NumberFormat.html#getNumberInstance()), [NumberFormat.getCurrencyInstance()](http://docs.google.com/java/text/NumberFormat.html#getCurrencyInstance()), [NumberFormat.getPercentInstance()](http://docs.google.com/java/text/NumberFormat.html#getPercentInstance()), [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html)

| **Method Detail** |
| --- |

### format

public final [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) **format**([Object](http://docs.google.com/java/lang/Object.html) number,  
 [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) toAppendTo,  
 [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) pos)

Formats a number and appends the resulting text to the given string buffer. The number can be of any subclass of [Number](http://docs.google.com/java/lang/Number.html).

This implementation uses the maximum precision permitted.

**Overrides:**[format](http://docs.google.com/java/text/NumberFormat.html#format(java.lang.Object,%20java.lang.StringBuffer,%20java.text.FieldPosition)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**number - the number to formattoAppendTo - the StringBuffer to which the formatted text is to be appendedpos - On input: an alignment field, if desired. On output: the offsets of the alignment field. **Returns:**the value passed in as toAppendTo **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if number is null or not an instance of Number. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if toAppendTo or pos is null [ArithmeticException](http://docs.google.com/java/lang/ArithmeticException.html) - if rounding is needed with rounding mode being set to RoundingMode.UNNECESSARY**See Also:**[FieldPosition](http://docs.google.com/java/text/FieldPosition.html)

### format

public [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) **format**(double number,  
 [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) result,  
 [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) fieldPosition)

Formats a double to produce a string.

**Specified by:**[format](http://docs.google.com/java/text/NumberFormat.html#format(double,%20java.lang.StringBuffer,%20java.text.FieldPosition)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**number - The double to formatresult - where the text is to be appendedfieldPosition - On input: an alignment field, if desired. On output: the offsets of the alignment field. **Returns:**The formatted number string **Throws:** [ArithmeticException](http://docs.google.com/java/lang/ArithmeticException.html) - if rounding is needed with rounding mode being set to RoundingMode.UNNECESSARY**See Also:**[FieldPosition](http://docs.google.com/java/text/FieldPosition.html)

### format

public [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) **format**(long number,  
 [StringBuffer](http://docs.google.com/java/lang/StringBuffer.html) result,  
 [FieldPosition](http://docs.google.com/java/text/FieldPosition.html) fieldPosition)

Format a long to produce a string.

**Specified by:**[format](http://docs.google.com/java/text/NumberFormat.html#format(long,%20java.lang.StringBuffer,%20java.text.FieldPosition)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**number - The long to formatresult - where the text is to be appendedfieldPosition - On input: an alignment field, if desired. On output: the offsets of the alignment field. **Returns:**The formatted number string **Throws:** [ArithmeticException](http://docs.google.com/java/lang/ArithmeticException.html) - if rounding is needed with rounding mode being set to RoundingMode.UNNECESSARY**See Also:**[FieldPosition](http://docs.google.com/java/text/FieldPosition.html)

### formatToCharacterIterator

public [AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) **formatToCharacterIterator**([Object](http://docs.google.com/java/lang/Object.html) obj)

Formats an Object producing an AttributedCharacterIterator. You can use the returned AttributedCharacterIterator to build the resulting String, as well as to determine information about the resulting String.

Each attribute key of the AttributedCharacterIterator will be of type NumberFormat.Field, with the attribute value being the same as the attribute key.

**Overrides:**[formatToCharacterIterator](http://docs.google.com/java/text/Format.html#formatToCharacterIterator(java.lang.Object)) in class [Format](http://docs.google.com/java/text/Format.html) **Parameters:**obj - The object to format **Returns:**AttributedCharacterIterator describing the formatted value. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if obj is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - when the Format cannot format the given object. [ArithmeticException](http://docs.google.com/java/lang/ArithmeticException.html) - if rounding is needed with rounding mode being set to RoundingMode.UNNECESSARY**Since:** 1.4

### parse

public [Number](http://docs.google.com/java/lang/Number.html) **parse**([String](http://docs.google.com/java/lang/String.html) text,  
 [ParsePosition](http://docs.google.com/java/text/ParsePosition.html) pos)

Parses text from a string to produce a Number.

The method attempts to parse text starting at the index given by pos. If parsing succeeds, then the index of pos is updated to the index after the last character used (parsing does not necessarily use all characters up to the end of the string), and the parsed number is returned. The updated pos can be used to indicate the starting point for the next call to this method. If an error occurs, then the index of pos is not changed, the error index of pos is set to the index of the character where the error occurred, and null is returned.

The subclass returned depends on the value of [isParseBigDecimal()](http://docs.google.com/java/text/DecimalFormat.html#isParseBigDecimal()) as well as on the string being parsed.

* If isParseBigDecimal() is false (the default), most integer values are returned as Long objects, no matter how they are written: "17" and "17.000" both parse to Long(17). Values that cannot fit into a Long are returned as Doubles. This includes values with a fractional part, infinite values, NaN, and the value -0.0. DecimalFormat does *not* decide whether to return a Double or a Long based on the presence of a decimal separator in the source string. Doing so would prevent integers that overflow the mantissa of a double, such as "-9,223,372,036,854,775,808.00", from being parsed accurately.  
  Callers may use the Number methods doubleValue, longValue, etc., to obtain the type they want.
* If isParseBigDecimal() is true, values are returned as BigDecimal objects. The values are the ones constructed by [BigDecimal.BigDecimal(String)](http://docs.google.com/java/math/BigDecimal.html#BigDecimal(java.lang.String)) for corresponding strings in locale-independent format. The special cases negative and positive infinity and NaN are returned as Double instances holding the values of the corresponding Double constants.

DecimalFormat parses all Unicode characters that represent decimal digits, as defined by Character.digit(). In addition, DecimalFormat also recognizes as digits the ten consecutive characters starting with the localized zero digit defined in the DecimalFormatSymbols object.

**Specified by:**[parse](http://docs.google.com/java/text/NumberFormat.html#parse(java.lang.String,%20java.text.ParsePosition)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**text - the string to be parsedpos - A ParsePosition object with index and error index information as described above. **Returns:**the parsed value, or null if the parse fails **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if text or pos is null.**See Also:**[NumberFormat.isParseIntegerOnly()](http://docs.google.com/java/text/NumberFormat.html#isParseIntegerOnly()), [Format.parseObject(java.lang.String, java.text.ParsePosition)](http://docs.google.com/java/text/Format.html#parseObject(java.lang.String,%20java.text.ParsePosition))

### getDecimalFormatSymbols

public [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) **getDecimalFormatSymbols**()

Returns a copy of the decimal format symbols, which is generally not changed by the programmer or user.

**Returns:**a copy of the desired DecimalFormatSymbols**See Also:**[DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html)

### setDecimalFormatSymbols

public void **setDecimalFormatSymbols**([DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) newSymbols)

Sets the decimal format symbols, which is generally not changed by the programmer or user.

**Parameters:**newSymbols - desired DecimalFormatSymbols**See Also:**[DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html)

### getPositivePrefix

public [String](http://docs.google.com/java/lang/String.html) **getPositivePrefix**()

Get the positive prefix.

Examples: +123, $123, sFr123

### setPositivePrefix

public void **setPositivePrefix**([String](http://docs.google.com/java/lang/String.html) newValue)

Set the positive prefix.

Examples: +123, $123, sFr123

### getNegativePrefix

public [String](http://docs.google.com/java/lang/String.html) **getNegativePrefix**()

Get the negative prefix.

Examples: -123, ($123) (with negative suffix), sFr-123

### setNegativePrefix

public void **setNegativePrefix**([String](http://docs.google.com/java/lang/String.html) newValue)

Set the negative prefix.

Examples: -123, ($123) (with negative suffix), sFr-123

### getPositiveSuffix

public [String](http://docs.google.com/java/lang/String.html) **getPositiveSuffix**()

Get the positive suffix.

Example: 123%

### setPositiveSuffix

public void **setPositiveSuffix**([String](http://docs.google.com/java/lang/String.html) newValue)

Set the positive suffix.

Example: 123%

### getNegativeSuffix

public [String](http://docs.google.com/java/lang/String.html) **getNegativeSuffix**()

Get the negative suffix.

Examples: -123%, ($123) (with positive suffixes)

### setNegativeSuffix

public void **setNegativeSuffix**([String](http://docs.google.com/java/lang/String.html) newValue)

Set the negative suffix.

Examples: 123%

### getMultiplier

public int **getMultiplier**()

Gets the multiplier for use in percent, per mille, and similar formats.

**See Also:**[setMultiplier(int)](http://docs.google.com/java/text/DecimalFormat.html#setMultiplier(int))

### setMultiplier

public void **setMultiplier**(int newValue)

Sets the multiplier for use in percent, per mille, and similar formats. For a percent format, set the multiplier to 100 and the suffixes to have '%' (for Arabic, use the Arabic percent sign). For a per mille format, set the multiplier to 1000 and the suffixes to have '\u2030'.

Example: with multiplier 100, 1.23 is formatted as "123", and "123" is parsed into 1.23.

**See Also:**[getMultiplier()](http://docs.google.com/java/text/DecimalFormat.html#getMultiplier())

### getGroupingSize

public int **getGroupingSize**()

Return the grouping size. Grouping size is the number of digits between grouping separators in the integer portion of a number. For example, in the number "123,456.78", the grouping size is 3.

**See Also:**[setGroupingSize(int)](http://docs.google.com/java/text/DecimalFormat.html#setGroupingSize(int)), [NumberFormat.isGroupingUsed()](http://docs.google.com/java/text/NumberFormat.html#isGroupingUsed()), [DecimalFormatSymbols.getGroupingSeparator()](http://docs.google.com/java/text/DecimalFormatSymbols.html#getGroupingSeparator())

### setGroupingSize

public void **setGroupingSize**(int newValue)

Set the grouping size. Grouping size is the number of digits between grouping separators in the integer portion of a number. For example, in the number "123,456.78", the grouping size is 3.

The value passed in is converted to a byte, which may lose information.

**See Also:**[getGroupingSize()](http://docs.google.com/java/text/DecimalFormat.html#getGroupingSize()), [NumberFormat.setGroupingUsed(boolean)](http://docs.google.com/java/text/NumberFormat.html#setGroupingUsed(boolean)), [DecimalFormatSymbols.setGroupingSeparator(char)](http://docs.google.com/java/text/DecimalFormatSymbols.html#setGroupingSeparator(char))

### isDecimalSeparatorAlwaysShown

public boolean **isDecimalSeparatorAlwaysShown**()

Allows you to get the behavior of the decimal separator with integers. (The decimal separator will always appear with decimals.)

Example: Decimal ON: 12345 -> 12345.; OFF: 12345 -> 12345

### setDecimalSeparatorAlwaysShown

public void **setDecimalSeparatorAlwaysShown**(boolean newValue)

Allows you to set the behavior of the decimal separator with integers. (The decimal separator will always appear with decimals.)

Example: Decimal ON: 12345 -> 12345.; OFF: 12345 -> 12345

### isParseBigDecimal

public boolean **isParseBigDecimal**()

Returns whether the [parse(java.lang.String, java.text.ParsePosition)](http://docs.google.com/java/text/DecimalFormat.html#parse(java.lang.String,%20java.text.ParsePosition)) method returns BigDecimal. The default value is false.

**Since:** 1.5 **See Also:**[setParseBigDecimal(boolean)](http://docs.google.com/java/text/DecimalFormat.html#setParseBigDecimal(boolean))

### setParseBigDecimal

public void **setParseBigDecimal**(boolean newValue)

Sets whether the [parse(java.lang.String, java.text.ParsePosition)](http://docs.google.com/java/text/DecimalFormat.html#parse(java.lang.String,%20java.text.ParsePosition)) method returns BigDecimal.

**Since:** 1.5 **See Also:**[isParseBigDecimal()](http://docs.google.com/java/text/DecimalFormat.html#isParseBigDecimal())

### clone

public [Object](http://docs.google.com/java/lang/Object.html) **clone**()

Standard override; no change in semantics.

**Overrides:**[clone](http://docs.google.com/java/text/NumberFormat.html#clone()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Returns:**a clone of this instance.**See Also:**[Cloneable](http://docs.google.com/java/lang/Cloneable.html)

### equals

public boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) obj)

Overrides equals

**Overrides:**[equals](http://docs.google.com/java/text/NumberFormat.html#equals(java.lang.Object)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**obj - the reference object with which to compare. **Returns:**true if this object is the same as the obj argument; false otherwise.**See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### hashCode

public int **hashCode**()

Overrides hashCode

**Overrides:**[hashCode](http://docs.google.com/java/text/NumberFormat.html#hashCode()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Returns:**a hash code value for this object.**See Also:**[Object.equals(java.lang.Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### toPattern

public [String](http://docs.google.com/java/lang/String.html) **toPattern**()

Synthesizes a pattern string that represents the current state of this Format object.

**See Also:**[applyPattern(java.lang.String)](http://docs.google.com/java/text/DecimalFormat.html#applyPattern(java.lang.String))

### toLocalizedPattern

public [String](http://docs.google.com/java/lang/String.html) **toLocalizedPattern**()

Synthesizes a localized pattern string that represents the current state of this Format object.

**See Also:**[applyPattern(java.lang.String)](http://docs.google.com/java/text/DecimalFormat.html#applyPattern(java.lang.String))

### applyPattern

public void **applyPattern**([String](http://docs.google.com/java/lang/String.html) pattern)

Apply the given pattern to this Format object. A pattern is a short-hand specification for the various formatting properties. These properties can also be changed individually through the various setter methods.

There is no limit to integer digits set by this routine, since that is the typical end-user desire; use setMaximumInteger if you want to set a real value. For negative numbers, use a second pattern, separated by a semicolon

Example "#,#00.0#" -> 1,234.56

This means a minimum of 2 integer digits, 1 fraction digit, and a maximum of 2 fraction digits.

Example: "#,#00.0#;(#,#00.0#)" for negatives in parentheses.

In negative patterns, the minimum and maximum counts are ignored; these are presumed to be set in the positive pattern.

**Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if pattern is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given pattern is invalid.

### applyLocalizedPattern

public void **applyLocalizedPattern**([String](http://docs.google.com/java/lang/String.html) pattern)

Apply the given pattern to this Format object. The pattern is assumed to be in a localized notation. A pattern is a short-hand specification for the various formatting properties. These properties can also be changed individually through the various setter methods.

There is no limit to integer digits set by this routine, since that is the typical end-user desire; use setMaximumInteger if you want to set a real value. For negative numbers, use a second pattern, separated by a semicolon

Example "#,#00.0#" -> 1,234.56

This means a minimum of 2 integer digits, 1 fraction digit, and a maximum of 2 fraction digits.

Example: "#,#00.0#;(#,#00.0#)" for negatives in parentheses.

In negative patterns, the minimum and maximum counts are ignored; these are presumed to be set in the positive pattern.

**Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if pattern is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the given pattern is invalid.

### setMaximumIntegerDigits

public void **setMaximumIntegerDigits**(int newValue)

Sets the maximum number of digits allowed in the integer portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of newValue and 309 is used. Negative input values are replaced with 0.

**Overrides:**[setMaximumIntegerDigits](http://docs.google.com/java/text/NumberFormat.html#setMaximumIntegerDigits(int)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**newValue - the maximum number of integer digits to be shown; if less than zero, then zero is used. The concrete subclass may enforce an upper limit to this value appropriate to the numeric type being formatted.**See Also:**[NumberFormat.setMaximumIntegerDigits(int)](http://docs.google.com/java/text/NumberFormat.html#setMaximumIntegerDigits(int))

### setMinimumIntegerDigits

public void **setMinimumIntegerDigits**(int newValue)

Sets the minimum number of digits allowed in the integer portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of newValue and 309 is used. Negative input values are replaced with 0.

**Overrides:**[setMinimumIntegerDigits](http://docs.google.com/java/text/NumberFormat.html#setMinimumIntegerDigits(int)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**newValue - the minimum number of integer digits to be shown; if less than zero, then zero is used. The concrete subclass may enforce an upper limit to this value appropriate to the numeric type being formatted.**See Also:**[NumberFormat.setMinimumIntegerDigits(int)](http://docs.google.com/java/text/NumberFormat.html#setMinimumIntegerDigits(int))

### setMaximumFractionDigits

public void **setMaximumFractionDigits**(int newValue)

Sets the maximum number of digits allowed in the fraction portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of newValue and 340 is used. Negative input values are replaced with 0.

**Overrides:**[setMaximumFractionDigits](http://docs.google.com/java/text/NumberFormat.html#setMaximumFractionDigits(int)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**newValue - the maximum number of fraction digits to be shown; if less than zero, then zero is used. The concrete subclass may enforce an upper limit to this value appropriate to the numeric type being formatted.**See Also:**[NumberFormat.setMaximumFractionDigits(int)](http://docs.google.com/java/text/NumberFormat.html#setMaximumFractionDigits(int))

### setMinimumFractionDigits

public void **setMinimumFractionDigits**(int newValue)

Sets the minimum number of digits allowed in the fraction portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of newValue and 340 is used. Negative input values are replaced with 0.

**Overrides:**[setMinimumFractionDigits](http://docs.google.com/java/text/NumberFormat.html#setMinimumFractionDigits(int)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**newValue - the minimum number of fraction digits to be shown; if less than zero, then zero is used. The concrete subclass may enforce an upper limit to this value appropriate to the numeric type being formatted.**See Also:**[NumberFormat.setMinimumFractionDigits(int)](http://docs.google.com/java/text/NumberFormat.html#setMinimumFractionDigits(int))

### getMaximumIntegerDigits

public int **getMaximumIntegerDigits**()

Gets the maximum number of digits allowed in the integer portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of the return value and 309 is used.

**Overrides:**[getMaximumIntegerDigits](http://docs.google.com/java/text/NumberFormat.html#getMaximumIntegerDigits()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **See Also:**[setMaximumIntegerDigits(int)](http://docs.google.com/java/text/DecimalFormat.html#setMaximumIntegerDigits(int))

### getMinimumIntegerDigits

public int **getMinimumIntegerDigits**()

Gets the minimum number of digits allowed in the integer portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of the return value and 309 is used.

**Overrides:**[getMinimumIntegerDigits](http://docs.google.com/java/text/NumberFormat.html#getMinimumIntegerDigits()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **See Also:**[setMinimumIntegerDigits(int)](http://docs.google.com/java/text/DecimalFormat.html#setMinimumIntegerDigits(int))

### getMaximumFractionDigits

public int **getMaximumFractionDigits**()

Gets the maximum number of digits allowed in the fraction portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of the return value and 340 is used.

**Overrides:**[getMaximumFractionDigits](http://docs.google.com/java/text/NumberFormat.html#getMaximumFractionDigits()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **See Also:**[setMaximumFractionDigits(int)](http://docs.google.com/java/text/DecimalFormat.html#setMaximumFractionDigits(int))

### getMinimumFractionDigits

public int **getMinimumFractionDigits**()

Gets the minimum number of digits allowed in the fraction portion of a number. For formatting numbers other than BigInteger and BigDecimal objects, the lower of the return value and 340 is used.

**Overrides:**[getMinimumFractionDigits](http://docs.google.com/java/text/NumberFormat.html#getMinimumFractionDigits()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **See Also:**[setMinimumFractionDigits(int)](http://docs.google.com/java/text/DecimalFormat.html#setMinimumFractionDigits(int))

### getCurrency

public [Currency](http://docs.google.com/java/util/Currency.html) **getCurrency**()

Gets the currency used by this decimal format when formatting currency values. The currency is obtained by calling [DecimalFormatSymbols.getCurrency](http://docs.google.com/java/text/DecimalFormatSymbols.html#getCurrency()) on this number format's symbols.

**Overrides:**[getCurrency](http://docs.google.com/java/text/NumberFormat.html#getCurrency()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Returns:**the currency used by this decimal format, or null**Since:** 1.4

### setCurrency

public void **setCurrency**([Currency](http://docs.google.com/java/util/Currency.html) currency)

Sets the currency used by this number format when formatting currency values. This does not update the minimum or maximum number of fraction digits used by the number format. The currency is set by calling [DecimalFormatSymbols.setCurrency](http://docs.google.com/java/text/DecimalFormatSymbols.html#setCurrency(java.util.Currency)) on this number format's symbols.

**Overrides:**[setCurrency](http://docs.google.com/java/text/NumberFormat.html#setCurrency(java.util.Currency)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**currency - the new currency to be used by this decimal format **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if currency is null**Since:** 1.4

### getRoundingMode

public [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) **getRoundingMode**()

Gets the [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) used in this DecimalFormat.

**Overrides:**[getRoundingMode](http://docs.google.com/java/text/NumberFormat.html#getRoundingMode()) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Returns:**The RoundingMode used for this DecimalFormat.**Since:** 1.6 **See Also:**[setRoundingMode(RoundingMode)](http://docs.google.com/java/text/DecimalFormat.html#setRoundingMode(java.math.RoundingMode))

### setRoundingMode

public void **setRoundingMode**([RoundingMode](http://docs.google.com/java/math/RoundingMode.html) roundingMode)

Sets the [RoundingMode](http://docs.google.com/java/math/RoundingMode.html) used in this DecimalFormat.

**Overrides:**[setRoundingMode](http://docs.google.com/java/text/NumberFormat.html#setRoundingMode(java.math.RoundingMode)) in class [NumberFormat](http://docs.google.com/java/text/NumberFormat.html) **Parameters:**roundingMode - The RoundingMode to be used **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if roundingMode is null.**Since:** 1.6 **See Also:**[getRoundingMode()](http://docs.google.com/java/text/DecimalFormat.html#getRoundingMode())

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DecimalFormat.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/text/DateFormatSymbols.html)   [**NEXT CLASS**](http://docs.google.com/java/text/DecimalFormatSymbols.html) | [**FRAMES**](http://docs.google.com/index.html?java/text/DecimalFormat.html)    [**NO FRAMES**](http://docs.google.com/DecimalFormat.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#tyjcwt) | [FIELD](#1t3h5sf) | [CONSTR](#4d34og8) | [METHOD](#2s8eyo1) | DETAIL: FIELD | [CONSTR](#lnxbz9) | [METHOD](#2jxsxqh) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).