| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Font.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/awt/FocusTraversalPolicy.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/FontFormatException.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/Font.html)    [**NO FRAMES**](http://docs.google.com/Font.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#49x2ik5) | [METHOD](#23ckvvd) |

## **java.awt**

Class Font

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

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html) **Direct Known Subclasses:** [FontUIResource](http://docs.google.com/javax/swing/plaf/FontUIResource.html)

public class **Font**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Serializable](http://docs.google.com/java/io/Serializable.html)

The Font class represents fonts, which are used to render text in a visible way. A font provides the information needed to map sequences of *characters* to sequences of *glyphs* and to render sequences of glyphs on Graphics and Component objects.

#### Characters and Glyphs

A *character* is a symbol that represents an item such as a letter, a digit, or punctuation in an abstract way. For example, 'g', LATIN SMALL LETTER G, is a character.

A *glyph* is a shape used to render a character or a sequence of characters. In simple writing systems, such as Latin, typically one glyph represents one character. In general, however, characters and glyphs do not have one-to-one correspondence. For example, the character 'á' LATIN SMALL LETTER A WITH ACUTE, can be represented by two glyphs: one for 'a' and one for '´'. On the other hand, the two-character string "fi" can be represented by a single glyph, an "fi" ligature. In complex writing systems, such as Arabic or the South and South-East Asian writing systems, the relationship between characters and glyphs can be more complicated and involve context-dependent selection of glyphs as well as glyph reordering. A font encapsulates the collection of glyphs needed to render a selected set of characters as well as the tables needed to map sequences of characters to corresponding sequences of glyphs.

#### Physical and Logical Fonts

The Java Platform distinguishes between two kinds of fonts: *physical* fonts and *logical* fonts.

*Physical* fonts are the actual font libraries containing glyph data and tables to map from character sequences to glyph sequences, using a font technology such as TrueType or PostScript Type 1. All implementations of the Java Platform must support TrueType fonts; support for other font technologies is implementation dependent. Physical fonts may use names such as Helvetica, Palatino, HonMincho, or any number of other font names. Typically, each physical font supports only a limited set of writing systems, for example, only Latin characters or only Japanese and Basic Latin. The set of available physical fonts varies between configurations. Applications that require specific fonts can bundle them and instantiate them using the [createFont](http://docs.google.com/java/awt/Font.html#createFont(int,%20java.io.InputStream)) method.

*Logical* fonts are the five font families defined by the Java platform which must be supported by any Java runtime environment: Serif, SansSerif, Monospaced, Dialog, and DialogInput. These logical fonts are not actual font libraries. Instead, the logical font names are mapped to physical fonts by the Java runtime environment. The mapping is implementation and usually locale dependent, so the look and the metrics provided by them vary. Typically, each logical font name maps to several physical fonts in order to cover a large range of characters.

Peered AWT components, such as [Label](http://docs.google.com/java/awt/Label.html) and [TextField](http://docs.google.com/java/awt/TextField.html), can only use logical fonts.

For a discussion of the relative advantages and disadvantages of using physical or logical fonts, see the [Internationalization FAQ](http://java.sun.com/j2se/corejava/intl/reference/faqs/index.html#desktop-rendering) document.

#### Font Faces and Names

A Font can have many faces, such as heavy, medium, oblique, gothic and regular. All of these faces have similar typographic design.

There are three different names that you can get from a Font object. The *logical font name* is simply the name that was used to construct the font. The *font face name*, or just *font name* for short, is the name of a particular font face, like Helvetica Bold. The *family name* is the name of the font family that determines the typographic design across several faces, like Helvetica.

The Font class represents an instance of a font face from a collection of font faces that are present in the system resources of the host system. As examples, Arial Bold and Courier Bold Italic are font faces. There can be several Font objects associated with a font face, each differing in size, style, transform and font features.

The [getAllFonts](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAllFonts()) method of the GraphicsEnvironment class returns an array of all font faces available in the system. These font faces are returned as Font objects with a size of 1, identity transform and default font features. These base fonts can then be used to derive new Font objects with varying sizes, styles, transforms and font features via the deriveFont methods in this class.

#### Font and TextAttribute

Font supports most TextAttributes. This makes some operations, such as rendering underlined text, convenient since it is not necessary to explicitly construct a TextLayout object. Attributes can be set on a Font by constructing or deriving it using a Map of TextAttribute values.

The values of some TextAttributes are not serializable, and therefore attempting to serialize an instance of Font that has such values will not serialize them. This means a Font deserialized from such a stream will not compare equal to the original Font that contained the non-serializable attributes. This should very rarely pose a problem since these attributes are typically used only in special circumstances and are unlikely to be serialized.

* FOREGROUND and BACKGROUND use Paint values. The subclass Color is serializable, while GradientPaint and TexturePaint are not.
* CHAR\_REPLACEMENT uses GraphicAttribute values. The subclasses ShapeGraphicAttribute and ImageGraphicAttribute are not serializable.
* INPUT\_METHOD\_HIGHLIGHT uses InputMethodHighlight values, which are not serializable. See [InputMethodHighlight](http://docs.google.com/java/awt/im/InputMethodHighlight.html).

Clients who create custom subclasses of Paint and GraphicAttribute can make them serializable and avoid this problem. Clients who use input method highlights can convert these to the platform-specific attributes for that highlight on the current platform and set them on the Font as a workaround.

The Map-based constructor and deriveFont APIs ignore the FONT attribute, and it is not retained by the Font; the static [getFont(java.util.Map)](http://docs.google.com/java/awt/Font.html#getFont(java.util.Map)) method should be used if the FONT attribute might be present. See [TextAttribute.FONT](http://docs.google.com/java/awt/font/TextAttribute.html#FONT) for more information.

Several attributes will cause additional rendering overhead and potentially invoke layout. If a Font has such attributes, the [hasLayoutAttributes](http://docs.google.com/java/awt/Font.html#hasLayoutAttributes) method will return true.

Note: Font rotations can cause text baselines to be rotated. In order to account for this (rare) possibility, font APIs are specified to return metrics and take parameters 'in baseline-relative coordinates'. This maps the 'x' coordinate to the advance along the baseline, (positive x is forward along the baseline), and the 'y' coordinate to a distance along the perpendicular to the baseline at 'x' (positive y is 90 degrees clockwise from the baseline vector). APIs for which this is especially important are called out as having 'baseline-relative coordinates.'

**See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.awt.Font)

| **Field Summary** | |
| --- | --- |
| static int | [**BOLD**](http://docs.google.com/java/awt/Font.html#BOLD)            The bold style constant. |
| static int | [**CENTER\_BASELINE**](http://docs.google.com/java/awt/Font.html#CENTER_BASELINE)            The baseline used in ideographic scripts like Chinese, Japanese, and Korean when laying out text. |
| static [String](http://docs.google.com/java/lang/String.html) | [**DIALOG**](http://docs.google.com/java/awt/Font.html#DIALOG)            A String constant for the canonical family name of the logical font "Dialog". |
| static [String](http://docs.google.com/java/lang/String.html) | [**DIALOG\_INPUT**](http://docs.google.com/java/awt/Font.html#DIALOG_INPUT)            A String constant for the canonical family name of the logical font "DialogInput". |
| static int | [**HANGING\_BASELINE**](http://docs.google.com/java/awt/Font.html#HANGING_BASELINE)            The baseline used in Devanigiri and similar scripts when laying out text. |
| static int | [**ITALIC**](http://docs.google.com/java/awt/Font.html#ITALIC)            The italicized style constant. |
| static int | [**LAYOUT\_LEFT\_TO\_RIGHT**](http://docs.google.com/java/awt/Font.html#LAYOUT_LEFT_TO_RIGHT)            A flag to layoutGlyphVector indicating that text is left-to-right as determined by Bidi analysis. |
| static int | [**LAYOUT\_NO\_LIMIT\_CONTEXT**](http://docs.google.com/java/awt/Font.html#LAYOUT_NO_LIMIT_CONTEXT)            A flag to layoutGlyphVector indicating that text in the char array after the indicated limit should not be examined. |
| static int | [**LAYOUT\_NO\_START\_CONTEXT**](http://docs.google.com/java/awt/Font.html#LAYOUT_NO_START_CONTEXT)            A flag to layoutGlyphVector indicating that text in the char array before the indicated start should not be examined. |
| static int | [**LAYOUT\_RIGHT\_TO\_LEFT**](http://docs.google.com/java/awt/Font.html#LAYOUT_RIGHT_TO_LEFT)            A flag to layoutGlyphVector indicating that text is right-to-left as determined by Bidi analysis. |
| static [String](http://docs.google.com/java/lang/String.html) | [**MONOSPACED**](http://docs.google.com/java/awt/Font.html#MONOSPACED)            A String constant for the canonical family name of the logical font "Monospaced". |
| protected  [String](http://docs.google.com/java/lang/String.html) | [**name**](http://docs.google.com/java/awt/Font.html#name)            The logical name of this Font, as passed to the constructor. |
| static int | [**PLAIN**](http://docs.google.com/java/awt/Font.html#PLAIN)            The plain style constant. |
| protected  float | [**pointSize**](http://docs.google.com/java/awt/Font.html#pointSize)            The point size of this Font in float. |
| static int | [**ROMAN\_BASELINE**](http://docs.google.com/java/awt/Font.html#ROMAN_BASELINE)            The baseline used in most Roman scripts when laying out text. |
| static [String](http://docs.google.com/java/lang/String.html) | [**SANS\_SERIF**](http://docs.google.com/java/awt/Font.html#SANS_SERIF)            A String constant for the canonical family name of the logical font "SansSerif". |
| static [String](http://docs.google.com/java/lang/String.html) | [**SERIF**](http://docs.google.com/java/awt/Font.html#SERIF)            A String constant for the canonical family name of the logical font "Serif". |
| protected  int | [**size**](http://docs.google.com/java/awt/Font.html#size)            The point size of this Font, rounded to integer. |
| protected  int | [**style**](http://docs.google.com/java/awt/Font.html#style)            The style of this Font, as passed to the constructor. |
| static int | [**TRUETYPE\_FONT**](http://docs.google.com/java/awt/Font.html#TRUETYPE_FONT)            Identify a font resource of type TRUETYPE. |
| static int | [**TYPE1\_FONT**](http://docs.google.com/java/awt/Font.html#TYPE1_FONT)            Identify a font resource of type TYPE1. |

| **Constructor Summary** | |
| --- | --- |
| protected | [**Font**](http://docs.google.com/java/awt/Font.html#Font(java.awt.Font))([Font](http://docs.google.com/java/awt/Font.html) font)            Creates a new Font from the specified font. |
|  | [**Font**](http://docs.google.com/java/awt/Font.html#Font(java.util.Map))([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)            Creates a new Font with the specified attributes. |
|  | [**Font**](http://docs.google.com/java/awt/Font.html#Font(java.lang.String,%20int,%20int))([String](http://docs.google.com/java/lang/String.html) name, int style, int size)            Creates a new Font from the specified name, style and point size. |

| **Method Summary** | |
| --- | --- |
| boolean | [**canDisplay**](http://docs.google.com/java/awt/Font.html#canDisplay(char))(char c)            Checks if this Font has a glyph for the specified character. |
| boolean | [**canDisplay**](http://docs.google.com/java/awt/Font.html#canDisplay(int))(int codePoint)            Checks if this Font has a glyph for the specified character. |
| int | [**canDisplayUpTo**](http://docs.google.com/java/awt/Font.html#canDisplayUpTo(char%5B%5D,%20int,%20int))(char[] text, int start, int limit)            Indicates whether or not this Font can display the characters in the specified text starting at start and ending at limit. |
| int | [**canDisplayUpTo**](http://docs.google.com/java/awt/Font.html#canDisplayUpTo(java.text.CharacterIterator,%20int,%20int))([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) iter, int start, int limit)            Indicates whether or not this Font can display the text specified by the iter starting at start and ending at limit. |
| int | [**canDisplayUpTo**](http://docs.google.com/java/awt/Font.html#canDisplayUpTo(java.lang.String))([String](http://docs.google.com/java/lang/String.html) str)            Indicates whether or not this Font can display a specified String. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**createFont**](http://docs.google.com/java/awt/Font.html#createFont(int,%20java.io.File))(int fontFormat, [File](http://docs.google.com/java/io/File.html) fontFile)            Returns a new Font using the specified font type and the specified font file. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**createFont**](http://docs.google.com/java/awt/Font.html#createFont(int,%20java.io.InputStream))(int fontFormat, [InputStream](http://docs.google.com/java/io/InputStream.html) fontStream)            Returns a new Font using the specified font type and input data. |
| [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) | [**createGlyphVector**](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20char%5B%5D))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc, char[] chars)            Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. |
| [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) | [**createGlyphVector**](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.text.CharacterIterator))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc, [CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci)            Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping the specified characters to glyphs one-to-one based on the Unicode cmap in this Font. |
| [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) | [**createGlyphVector**](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20int%5B%5D))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc, int[] glyphCodes)            Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. |
| [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) | [**createGlyphVector**](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc, [String](http://docs.google.com/java/lang/String.html) str)            Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**decode**](http://docs.google.com/java/awt/Font.html#decode(java.lang.String))([String](http://docs.google.com/java/lang/String.html) str)            Returns the Font that the str argument describes. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) trans)            Creates a new Font object by replicating the current Font object and applying a new transform to it. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(float))(float size)            Creates a new Font object by replicating the current Font object and applying a new size to it. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(int))(int style)            Creates a new Font object by replicating the current Font object and applying a new style to it. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(int,%20java.awt.geom.AffineTransform))(int style, [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) trans)            Creates a new Font object by replicating this Font object and applying a new style and transform. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(int,%20float))(int style, float size)            Creates a new Font object by replicating this Font object and applying a new style and size. |
| [Font](http://docs.google.com/java/awt/Font.html) | [**deriveFont**](http://docs.google.com/java/awt/Font.html#deriveFont(java.util.Map))([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)            Creates a new Font object by replicating the current Font object and applying a new set of font attributes to it. |
| boolean | [**equals**](http://docs.google.com/java/awt/Font.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Compares this Font object to the specified Object. |
| protected  void | [**finalize**](http://docs.google.com/java/awt/Font.html#finalize())()            Disposes the native Font object. |
| [Map](http://docs.google.com/java/util/Map.html)<[TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html),?> | [**getAttributes**](http://docs.google.com/java/awt/Font.html#getAttributes())()            Returns a map of font attributes available in this Font. |
| [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html)[] | [**getAvailableAttributes**](http://docs.google.com/java/awt/Font.html#getAvailableAttributes())()            Returns the keys of all the attributes supported by this Font. |
| byte | [**getBaselineFor**](http://docs.google.com/java/awt/Font.html#getBaselineFor(char))(char c)            Returns the baseline appropriate for displaying this character. |
| [String](http://docs.google.com/java/lang/String.html) | [**getFamily**](http://docs.google.com/java/awt/Font.html#getFamily())()            Returns the family name of this Font. |
| [String](http://docs.google.com/java/lang/String.html) | [**getFamily**](http://docs.google.com/java/awt/Font.html#getFamily(java.util.Locale))([Locale](http://docs.google.com/java/util/Locale.html) l)            Returns the family name of this Font, localized for the specified locale. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**getFont**](http://docs.google.com/java/awt/Font.html#getFont(java.util.Map))([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)            Returns a Font appropriate to the attributes. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**getFont**](http://docs.google.com/java/awt/Font.html#getFont(java.lang.String))([String](http://docs.google.com/java/lang/String.html) nm)            Returns a Font object from the system properties list. |
| static [Font](http://docs.google.com/java/awt/Font.html) | [**getFont**](http://docs.google.com/java/awt/Font.html#getFont(java.lang.String,%20java.awt.Font))([String](http://docs.google.com/java/lang/String.html) nm, [Font](http://docs.google.com/java/awt/Font.html) font)            Gets the specified Font from the system properties list. |
| [String](http://docs.google.com/java/lang/String.html) | [**getFontName**](http://docs.google.com/java/awt/Font.html#getFontName())()            Returns the font face name of this Font. |
| [String](http://docs.google.com/java/lang/String.html) | [**getFontName**](http://docs.google.com/java/awt/Font.html#getFontName(java.util.Locale))([Locale](http://docs.google.com/java/util/Locale.html) l)            Returns the font face name of the Font, localized for the specified locale. |
| float | [**getItalicAngle**](http://docs.google.com/java/awt/Font.html#getItalicAngle())()            Returns the italic angle of this Font. |
| [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) | [**getLineMetrics**](http://docs.google.com/java/awt/Font.html#getLineMetrics(char%5B%5D,%20int,%20int,%20java.awt.font.FontRenderContext))(char[] chars, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns a LineMetrics object created with the specified arguments. |
| [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) | [**getLineMetrics**](http://docs.google.com/java/awt/Font.html#getLineMetrics(java.text.CharacterIterator,%20int,%20int,%20java.awt.font.FontRenderContext))([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns a LineMetrics object created with the specified arguments. |
| [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) | [**getLineMetrics**](http://docs.google.com/java/awt/Font.html#getLineMetrics(java.lang.String,%20java.awt.font.FontRenderContext))([String](http://docs.google.com/java/lang/String.html) str, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns a [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) object created with the specified String and [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html). |
| [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) | [**getLineMetrics**](http://docs.google.com/java/awt/Font.html#getLineMetrics(java.lang.String,%20int,%20int,%20java.awt.font.FontRenderContext))([String](http://docs.google.com/java/lang/String.html) str, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns a LineMetrics object created with the specified arguments. |
| [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) | [**getMaxCharBounds**](http://docs.google.com/java/awt/Font.html#getMaxCharBounds(java.awt.font.FontRenderContext))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns the bounds for the character with the maximum bounds as defined in the specified FontRenderContext. |
| int | [**getMissingGlyphCode**](http://docs.google.com/java/awt/Font.html#getMissingGlyphCode())()            Returns the glyphCode which is used when this Font does not have a glyph for a specified unicode code point. |
| [String](http://docs.google.com/java/lang/String.html) | [**getName**](http://docs.google.com/java/awt/Font.html#getName())()            Returns the logical name of this Font. |
| int | [**getNumGlyphs**](http://docs.google.com/java/awt/Font.html#getNumGlyphs())()            Returns the number of glyphs in this Font. |
| java.awt.peer.FontPeer | [**getPeer**](http://docs.google.com/java/awt/Font.html#getPeer())()  **Deprecated.** *Font rendering is now platform independent.* |
| [String](http://docs.google.com/java/lang/String.html) | [**getPSName**](http://docs.google.com/java/awt/Font.html#getPSName())()            Returns the postscript name of this Font. |
| int | [**getSize**](http://docs.google.com/java/awt/Font.html#getSize())()            Returns the point size of this Font, rounded to an integer. |
| float | [**getSize2D**](http://docs.google.com/java/awt/Font.html#getSize2D())()            Returns the point size of this Font in float value. |
| [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) | [**getStringBounds**](http://docs.google.com/java/awt/Font.html#getStringBounds(char%5B%5D,%20int,%20int,%20java.awt.font.FontRenderContext))(char[] chars, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns the logical bounds of the specified array of characters in the specified FontRenderContext. |
| [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) | [**getStringBounds**](http://docs.google.com/java/awt/Font.html#getStringBounds(java.text.CharacterIterator,%20int,%20int,%20java.awt.font.FontRenderContext))([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns the logical bounds of the characters indexed in the specified [CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) in the specified FontRenderContext. |
| [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) | [**getStringBounds**](http://docs.google.com/java/awt/Font.html#getStringBounds(java.lang.String,%20java.awt.font.FontRenderContext))([String](http://docs.google.com/java/lang/String.html) str, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns the logical bounds of the specified String in the specified FontRenderContext. |
| [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) | [**getStringBounds**](http://docs.google.com/java/awt/Font.html#getStringBounds(java.lang.String,%20int,%20int,%20java.awt.font.FontRenderContext))([String](http://docs.google.com/java/lang/String.html) str, int beginIndex, int limit, [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)            Returns the logical bounds of the specified String in the specified FontRenderContext. |
| int | [**getStyle**](http://docs.google.com/java/awt/Font.html#getStyle())()            Returns the style of this Font. |
| [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getTransform**](http://docs.google.com/java/awt/Font.html#getTransform())()            Returns a copy of the transform associated with this Font. |
| int | [**hashCode**](http://docs.google.com/java/awt/Font.html#hashCode())()            Returns a hashcode for this Font. |
| boolean | [**hasLayoutAttributes**](http://docs.google.com/java/awt/Font.html#hasLayoutAttributes())()            Return true if this Font contains attributes that require extra layout processing. |
| boolean | [**hasUniformLineMetrics**](http://docs.google.com/java/awt/Font.html#hasUniformLineMetrics())()            Checks whether or not this Font has uniform line metrics. |
| boolean | [**isBold**](http://docs.google.com/java/awt/Font.html#isBold())()            Indicates whether or not this Font object's style is BOLD. |
| boolean | [**isItalic**](http://docs.google.com/java/awt/Font.html#isItalic())()            Indicates whether or not this Font object's style is ITALIC. |
| boolean | [**isPlain**](http://docs.google.com/java/awt/Font.html#isPlain())()            Indicates whether or not this Font object's style is PLAIN. |
| boolean | [**isTransformed**](http://docs.google.com/java/awt/Font.html#isTransformed())()            Indicates whether or not this Font object has a transform that affects its size in addition to the Size attribute. |
| [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) | [**layoutGlyphVector**](http://docs.google.com/java/awt/Font.html#layoutGlyphVector(java.awt.font.FontRenderContext,%20char%5B%5D,%20int,%20int,%20int))([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc, char[] text, int start, int limit, int flags)            Returns a new GlyphVector object, performing full layout of the text if possible. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/awt/Font.html#toString())()            Converts this Font object to a String representation. |

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

| **Field Detail** |
| --- |

### DIALOG

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

A String constant for the canonical family name of the logical font "Dialog". It is useful in Font construction to provide compile-time verification of the name.

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.DIALOG)

### DIALOG\_INPUT

public static final [String](http://docs.google.com/java/lang/String.html) **DIALOG\_INPUT**

A String constant for the canonical family name of the logical font "DialogInput". It is useful in Font construction to provide compile-time verification of the name.

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.DIALOG_INPUT)

### SANS\_SERIF

public static final [String](http://docs.google.com/java/lang/String.html) **SANS\_SERIF**

A String constant for the canonical family name of the logical font "SansSerif". It is useful in Font construction to provide compile-time verification of the name.

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.SANS_SERIF)

### SERIF

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

A String constant for the canonical family name of the logical font "Serif". It is useful in Font construction to provide compile-time verification of the name.

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.SERIF)

### MONOSPACED

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

A String constant for the canonical family name of the logical font "Monospaced". It is useful in Font construction to provide compile-time verification of the name.

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.MONOSPACED)

### PLAIN

public static final int **PLAIN**

The plain style constant.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.PLAIN)

### BOLD

public static final int **BOLD**

The bold style constant. This can be combined with the other style constants (except PLAIN) for mixed styles.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.BOLD)

### ITALIC

public static final int **ITALIC**

The italicized style constant. This can be combined with the other style constants (except PLAIN) for mixed styles.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.ITALIC)

### ROMAN\_BASELINE

public static final int **ROMAN\_BASELINE**

The baseline used in most Roman scripts when laying out text.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.ROMAN_BASELINE)

### CENTER\_BASELINE

public static final int **CENTER\_BASELINE**

The baseline used in ideographic scripts like Chinese, Japanese, and Korean when laying out text.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.CENTER_BASELINE)

### HANGING\_BASELINE

public static final int **HANGING\_BASELINE**

The baseline used in Devanigiri and similar scripts when laying out text.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.HANGING_BASELINE)

### TRUETYPE\_FONT

public static final int **TRUETYPE\_FONT**

Identify a font resource of type TRUETYPE. Used to specify a TrueType font resource to the [createFont(int, java.io.InputStream)](http://docs.google.com/java/awt/Font.html#createFont(int,%20java.io.InputStream)) method.

**Since:** 1.3 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.TRUETYPE_FONT)

### TYPE1\_FONT

public static final int **TYPE1\_FONT**

Identify a font resource of type TYPE1. Used to specify a Type1 font resource to the [createFont(int, java.io.InputStream)](http://docs.google.com/java/awt/Font.html#createFont(int,%20java.io.InputStream)) method.

**Since:** 1.5 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.TYPE1_FONT)

### name

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

The logical name of this Font, as passed to the constructor.

**Since:** JDK1.0 **See Also:**[getName()](http://docs.google.com/java/awt/Font.html#getName())

### style

protected int **style**

The style of this Font, as passed to the constructor. This style can be PLAIN, BOLD, ITALIC, or BOLD+ITALIC.

**Since:** JDK1.0 **See Also:**[getStyle()](http://docs.google.com/java/awt/Font.html#getStyle())

### size

protected int **size**

The point size of this Font, rounded to integer.

**Since:** JDK1.0 **See Also:**[getSize()](http://docs.google.com/java/awt/Font.html#getSize())

### pointSize

protected float **pointSize**

The point size of this Font in float.

**See Also:**[getSize()](http://docs.google.com/java/awt/Font.html#getSize()), [getSize2D()](http://docs.google.com/java/awt/Font.html#getSize2D())

### LAYOUT\_LEFT\_TO\_RIGHT

public static final int **LAYOUT\_LEFT\_TO\_RIGHT**

A flag to layoutGlyphVector indicating that text is left-to-right as determined by Bidi analysis.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.LAYOUT_LEFT_TO_RIGHT)

### LAYOUT\_RIGHT\_TO\_LEFT

public static final int **LAYOUT\_RIGHT\_TO\_LEFT**

A flag to layoutGlyphVector indicating that text is right-to-left as determined by Bidi analysis.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.LAYOUT_RIGHT_TO_LEFT)

### LAYOUT\_NO\_START\_CONTEXT

public static final int **LAYOUT\_NO\_START\_CONTEXT**

A flag to layoutGlyphVector indicating that text in the char array before the indicated start should not be examined.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.LAYOUT_NO_START_CONTEXT)

### LAYOUT\_NO\_LIMIT\_CONTEXT

public static final int **LAYOUT\_NO\_LIMIT\_CONTEXT**

A flag to layoutGlyphVector indicating that text in the char array after the indicated limit should not be examined.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.Font.LAYOUT_NO_LIMIT_CONTEXT)

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

### Font

public **Font**([String](http://docs.google.com/java/lang/String.html) name,  
 int style,  
 int size)

Creates a new Font from the specified name, style and point size.

The font name can be a font face name or a font family name. It is used together with the style to find an appropriate font face. When a font family name is specified, the style argument is used to select the most appropriate face from the family. When a font face name is specified, the face's style and the style argument are merged to locate the best matching font from the same family. For example if face name "Arial Bold" is specified with style Font.ITALIC, the font system looks for a face in the "Arial" family that is bold and italic, and may associate the font instance with the physical font face "Arial Bold Italic". The style argument is merged with the specified face's style, not added or subtracted. This means, specifying a bold face and a bold style does not double-embolden the font, and specifying a bold face and a plain style does not lighten the font.

If no face for the requested style can be found, the font system may apply algorithmic styling to achieve the desired style. For example, if ITALIC is requested, but no italic face is available, glyphs from the plain face may be algorithmically obliqued (slanted).

Font name lookup is case insensitive, using the case folding rules of the US locale.

If the name parameter represents something other than a logical font, i.e. is interpreted as a physical font face or family, and this cannot be mapped by the implementation to a physical font or a compatible alternative, then the font system will map the Font instance to "Dialog", such that for example, the family as reported by [getFamily](http://docs.google.com/java/awt/Font.html#getFamily()) will be "Dialog".

**Parameters:**name - the font name. This can be a font face name or a font family name, and may represent either a logical font or a physical font found in this GraphicsEnvironment. The family names for logical fonts are: Dialog, DialogInput, Monospaced, Serif, or SansSerif. Pre-defined String constants exist for all of these names, eg @see #DIALOG. If name is null, the *logical font name* of the new Font as returned by getName()is set to the name "Default".style - the style constant for the Font The style argument is an integer bitmask that may be PLAIN, or a bitwise union of BOLD and/or ITALIC (for example, ITALIC or BOLD|ITALIC). If the style argument does not conform to one of the expected integer bitmasks then the style is set to PLAIN.size - the point size of the Font**Since:** JDK1.0 **See Also:**[GraphicsEnvironment.getAllFonts()](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAllFonts()), [GraphicsEnvironment.getAvailableFontFamilyNames()](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAvailableFontFamilyNames())

### Font

public **Font**([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)

Creates a new Font with the specified attributes. Only keys defined in [TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html) are recognized. In addition the FONT attribute is not recognized by this constructor (see [getAvailableAttributes()](http://docs.google.com/java/awt/Font.html#getAvailableAttributes())). Only attributes that have values of valid types will affect the new Font.

If attributes is null, a new Font is initialized with default values.

**Parameters:**attributes - the attributes to assign to the new Font, or null**See Also:**[TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html)

### Font

protected **Font**([Font](http://docs.google.com/java/awt/Font.html) font)

Creates a new Font from the specified font. This constructor is intended for use by subclasses.

**Parameters:**font - from which to create this Font. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if font is null**Since:** 1.6

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

### getPeer

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public java.awt.peer.FontPeer **getPeer**()

**Deprecated.** *Font rendering is now platform independent.*

Gets the peer of this Font.

**Returns:**the peer of the Font.**Since:** JDK1.1

### getFont

public static [Font](http://docs.google.com/java/awt/Font.html) **getFont**([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)

Returns a Font appropriate to the attributes. If attributescontains a FONT attribute with a valid Font as its value, it will be merged with any remaining attributes. See [TextAttribute.FONT](http://docs.google.com/java/awt/font/TextAttribute.html#FONT) for more information.

**Parameters:**attributes - the attributes to assign to the new Font **Returns:**a new Font created with the specified attributes **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if attributes is null.**Since:** 1.2 **See Also:**[TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html)

### createFont

public static [Font](http://docs.google.com/java/awt/Font.html) **createFont**(int fontFormat,  
 [InputStream](http://docs.google.com/java/io/InputStream.html) fontStream)  
 throws [FontFormatException](http://docs.google.com/java/awt/FontFormatException.html),  
 [IOException](http://docs.google.com/java/io/IOException.html)

Returns a new Font using the specified font type and input data. The new Font is created with a point size of 1 and style [PLAIN](http://docs.google.com/java/awt/Font.html#PLAIN). This base font can then be used with the deriveFont methods in this class to derive new Font objects with varying sizes, styles, transforms and font features. This method does not close the [InputStream](http://docs.google.com/java/io/InputStream.html).

To make the Font available to Font constructors the returned Font must be registered in the GraphicsEnviroment by calling [registerFont(Font)](http://docs.google.com/java/awt/GraphicsEnvironment.html#registerFont(java.awt.Font)).

**Parameters:**fontFormat - the type of the Font, which is [TRUETYPE\_FONT](http://docs.google.com/java/awt/Font.html#TRUETYPE_FONT) if a TrueType resource is specified. or [TYPE1\_FONT](http://docs.google.com/java/awt/Font.html#TYPE1_FONT) if a Type 1 resource is specified.fontStream - an InputStream object representing the input data for the font. **Returns:**a new Font created with the specified font type. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fontFormat is not TRUETYPE\_FONTorTYPE1\_FONT. [FontFormatException](http://docs.google.com/java/awt/FontFormatException.html) - if the fontStream data does not contain the required font tables for the specified format. [IOException](http://docs.google.com/java/io/IOException.html) - if the fontStream cannot be completely read.**Since:** 1.3 **See Also:**[GraphicsEnvironment.registerFont(Font)](http://docs.google.com/java/awt/GraphicsEnvironment.html#registerFont(java.awt.Font))

### createFont

public static [Font](http://docs.google.com/java/awt/Font.html) **createFont**(int fontFormat,  
 [File](http://docs.google.com/java/io/File.html) fontFile)  
 throws [FontFormatException](http://docs.google.com/java/awt/FontFormatException.html),  
 [IOException](http://docs.google.com/java/io/IOException.html)

Returns a new Font using the specified font type and the specified font file. The new Font is created with a point size of 1 and style [PLAIN](http://docs.google.com/java/awt/Font.html#PLAIN). This base font can then be used with the deriveFont methods in this class to derive new Font objects with varying sizes, styles, transforms and font features.

**Parameters:**fontFormat - the type of the Font, which is [TRUETYPE\_FONT](http://docs.google.com/java/awt/Font.html#TRUETYPE_FONT) if a TrueType resource is specified or [TYPE1\_FONT](http://docs.google.com/java/awt/Font.html#TYPE1_FONT) if a Type 1 resource is specified. So long as the returned font, or its derived fonts are referenced the implementation may continue to access fontFile to retrieve font data. Thus the results are undefined if the file is changed, or becomes inaccessible.

To make the Font available to Font constructors the returned Font must be registered in the GraphicsEnviroment by calling [registerFont(Font)](http://docs.google.com/java/awt/GraphicsEnvironment.html#registerFont(java.awt.Font)).

fontFile - a File object representing the input data for the font. **Returns:**a new Font created with the specified font type. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fontFormat is not TRUETYPE\_FONTorTYPE1\_FONT. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if fontFile is null. [IOException](http://docs.google.com/java/io/IOException.html) - if the fontFile cannot be read. [FontFormatException](http://docs.google.com/java/awt/FontFormatException.html) - if fontFile does not contain the required font tables for the specified format. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if the executing code does not have permission to read from the file.**Since:** 1.5 **See Also:**[GraphicsEnvironment.registerFont(Font)](http://docs.google.com/java/awt/GraphicsEnvironment.html#registerFont(java.awt.Font))

### getTransform

public [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getTransform**()

Returns a copy of the transform associated with this Font. This transform is not necessarily the one used to construct the font. If the font has algorithmic superscripting or width adjustment, this will be incorporated into the returned AffineTransform.

Typically, fonts will not be transformed. Clients generally should call [isTransformed()](http://docs.google.com/java/awt/Font.html#isTransformed()) first, and only call this method if isTransformed returns true.

**Returns:**an [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) object representing the transform attribute of this Font object.

### getFamily

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

Returns the family name of this Font.

The family name of a font is font specific. Two fonts such as Helvetica Italic and Helvetica Bold have the same family name, *Helvetica*, whereas their font face names are *Helvetica Bold* and *Helvetica Italic*. The list of available family names may be obtained by using the [GraphicsEnvironment.getAvailableFontFamilyNames()](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAvailableFontFamilyNames()) method.

Use getName to get the logical name of the font. Use getFontName to get the font face name of the font.

**Returns:**a String that is the family name of this Font.**Since:** JDK1.1 **See Also:**[getName()](http://docs.google.com/java/awt/Font.html#getName()), [getFontName()](http://docs.google.com/java/awt/Font.html#getFontName())

### getFamily

public [String](http://docs.google.com/java/lang/String.html) **getFamily**([Locale](http://docs.google.com/java/util/Locale.html) l)

Returns the family name of this Font, localized for the specified locale.

The family name of a font is font specific. Two fonts such as Helvetica Italic and Helvetica Bold have the same family name, *Helvetica*, whereas their font face names are *Helvetica Bold* and *Helvetica Italic*. The list of available family names may be obtained by using the [GraphicsEnvironment.getAvailableFontFamilyNames()](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAvailableFontFamilyNames()) method.

Use getFontName to get the font face name of the font.

**Parameters:**l - locale for which to get the family name **Returns:**a String representing the family name of the font, localized for the specified locale.**Since:** 1.2 **See Also:**[getFontName()](http://docs.google.com/java/awt/Font.html#getFontName()), [Locale](http://docs.google.com/java/util/Locale.html)

### getPSName

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

Returns the postscript name of this Font. Use getFamily to get the family name of the font. Use getFontName to get the font face name of the font.

**Returns:**a String representing the postscript name of this Font.**Since:** 1.2

### getName

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

Returns the logical name of this Font. Use getFamily to get the family name of the font. Use getFontName to get the font face name of the font.

**Returns:**a String representing the logical name of this Font.**Since:** JDK1.0 **See Also:**[getFamily()](http://docs.google.com/java/awt/Font.html#getFamily()), [getFontName()](http://docs.google.com/java/awt/Font.html#getFontName())

### getFontName

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

Returns the font face name of this Font. For example, Helvetica Bold could be returned as a font face name. Use getFamily to get the family name of the font. Use getName to get the logical name of the font.

**Returns:**a String representing the font face name of this Font.**Since:** 1.2 **See Also:**[getFamily()](http://docs.google.com/java/awt/Font.html#getFamily()), [getName()](http://docs.google.com/java/awt/Font.html#getName())

### getFontName

public [String](http://docs.google.com/java/lang/String.html) **getFontName**([Locale](http://docs.google.com/java/util/Locale.html) l)

Returns the font face name of the Font, localized for the specified locale. For example, Helvetica Fett could be returned as the font face name. Use getFamily to get the family name of the font.

**Parameters:**l - a locale for which to get the font face name **Returns:**a String representing the font face name, localized for the specified locale.**See Also:**[getFamily()](http://docs.google.com/java/awt/Font.html#getFamily()), [Locale](http://docs.google.com/java/util/Locale.html)

### getStyle

public int **getStyle**()

Returns the style of this Font. The style can be PLAIN, BOLD, ITALIC, or BOLD+ITALIC.

**Returns:**the style of this Font**Since:** JDK1.0 **See Also:**[isPlain()](http://docs.google.com/java/awt/Font.html#isPlain()), [isBold()](http://docs.google.com/java/awt/Font.html#isBold()), [isItalic()](http://docs.google.com/java/awt/Font.html#isItalic())

### getSize

public int **getSize**()

Returns the point size of this Font, rounded to an integer. Most users are familiar with the idea of using *point size* to specify the size of glyphs in a font. This point size defines a measurement between the baseline of one line to the baseline of the following line in a single spaced text document. The point size is based on *typographic points*, approximately 1/72 of an inch.

The Java(tm)2D API adopts the convention that one point is equivalent to one unit in user coordinates. When using a normalized transform for converting user space coordinates to device space coordinates 72 user space units equal 1 inch in device space. In this case one point is 1/72 of an inch.

**Returns:**the point size of this Font in 1/72 of an inch units.**Since:** JDK1.0 **See Also:**[getSize2D()](http://docs.google.com/java/awt/Font.html#getSize2D()), [GraphicsConfiguration.getDefaultTransform()](http://docs.google.com/java/awt/GraphicsConfiguration.html#getDefaultTransform()), [GraphicsConfiguration.getNormalizingTransform()](http://docs.google.com/java/awt/GraphicsConfiguration.html#getNormalizingTransform())

### getSize2D

public float **getSize2D**()

Returns the point size of this Font in float value.

**Returns:**the point size of this Font as a float value.**Since:** 1.2 **See Also:**[getSize()](http://docs.google.com/java/awt/Font.html#getSize())

### isPlain

public boolean **isPlain**()

Indicates whether or not this Font object's style is PLAIN.

**Returns:**true if this Font has a PLAIN sytle; false otherwise.**Since:** JDK1.0 **See Also:**[getStyle()](http://docs.google.com/java/awt/Font.html#getStyle())

### isBold

public boolean **isBold**()

Indicates whether or not this Font object's style is BOLD.

**Returns:**true if this Font object's style is BOLD; false otherwise.**Since:** JDK1.0 **See Also:**[getStyle()](http://docs.google.com/java/awt/Font.html#getStyle())

### isItalic

public boolean **isItalic**()

Indicates whether or not this Font object's style is ITALIC.

**Returns:**true if this Font object's style is ITALIC; false otherwise.**Since:** JDK1.0 **See Also:**[getStyle()](http://docs.google.com/java/awt/Font.html#getStyle())

### isTransformed

public boolean **isTransformed**()

Indicates whether or not this Font object has a transform that affects its size in addition to the Size attribute.

**Returns:**true if this Font object has a non-identity AffineTransform attribute. false otherwise.**Since:** 1.4 **See Also:**[getTransform()](http://docs.google.com/java/awt/Font.html#getTransform())

### hasLayoutAttributes

public boolean **hasLayoutAttributes**()

Return true if this Font contains attributes that require extra layout processing.

**Returns:**true if the font has layout attributes**Since:** 1.6

### getFont

public static [Font](http://docs.google.com/java/awt/Font.html) **getFont**([String](http://docs.google.com/java/lang/String.html) nm)

Returns a Font object from the system properties list. nm is treated as the name of a system property to be obtained. The String value of this property is then interpreted as a Font object according to the specification of Font.decode(String) If the specified property is not found, or the executing code does not have permission to read the property, null is returned instead.

**Parameters:**nm - the property name **Returns:**a Font object that the property name describes, or null if no such property exists. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if nm is null.**Since:** 1.2 **See Also:**[decode(String)](http://docs.google.com/java/awt/Font.html#decode(java.lang.String))

### decode

public static [Font](http://docs.google.com/java/awt/Font.html) **decode**([String](http://docs.google.com/java/lang/String.html) str)

Returns the Font that the str argument describes. To ensure that this method returns the desired Font, format the str parameter in one of these ways

* *fontname-style-pointsize*
* *fontname-pointsize*
* *fontname-style*
* *fontname*
* *fontname style pointsize*
* *fontname pointsize*
* *fontname style*
* *fontname*

in which *style* is one of the four case-insensitive strings: "PLAIN", "BOLD", "BOLDITALIC", or "ITALIC", and pointsize is a positive decimal integer representation of the point size. For example, if you want a font that is Arial, bold, with a point size of 18, you would call this method with: "Arial-BOLD-18". This is equivalent to calling the Font constructor : new Font("Arial", Font.BOLD, 18); and the values are interpreted as specified by that constructor.

A valid trailing decimal field is always interpreted as the pointsize. Therefore a fontname containing a trailing decimal value should not be used in the fontname only form.

If a style name field is not one of the valid style strings, it is interpreted as part of the font name, and the default style is used.

Only one of ' ' or '-' may be used to separate fields in the input. The identified separator is the one closest to the end of the string which separates a valid pointsize, or a valid style name from the rest of the string. Null (empty) pointsize and style fields are treated as valid fields with the default value for that field.

Some font names may include the separator characters ' ' or '-'. If str is not formed with 3 components, e.g. such that style or pointsize fields are not present in str, and fontname also contains a character determined to be the separator character then these characters where they appear as intended to be part of fontname may instead be interpreted as separators so the font name may not be properly recognised.

The default size is 12 and the default style is PLAIN. If str does not specify a valid size, the returned Font has a size of 12. If str does not specify a valid style, the returned Font has a style of PLAIN. If you do not specify a valid font name in the str argument, this method will return a font with the family name "Dialog". To determine what font family names are available on your system, use the [GraphicsEnvironment.getAvailableFontFamilyNames()](http://docs.google.com/java/awt/GraphicsEnvironment.html#getAvailableFontFamilyNames()) method. If str is null, a new Font is returned with the family name "Dialog", a size of 12 and a PLAIN style.

**Parameters:**str - the name of the font, or null **Returns:**the Font object that str describes, or a new default Font if str is null.**Since:** JDK1.1 **See Also:**[getFamily()](http://docs.google.com/java/awt/Font.html#getFamily())

### getFont

public static [Font](http://docs.google.com/java/awt/Font.html) **getFont**([String](http://docs.google.com/java/lang/String.html) nm,  
 [Font](http://docs.google.com/java/awt/Font.html) font)

Gets the specified Font from the system properties list. As in the getProperty method of System, the first argument is treated as the name of a system property to be obtained. The String value of this property is then interpreted as a Font object.

The property value should be one of the forms accepted by Font.decode(String) If the specified property is not found, or the executing code does not have permission to read the property, the font argument is returned instead.

**Parameters:**nm - the case-insensitive property namefont - a default Font to return if property nm is not defined **Returns:**the Font value of the property. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if nm is null.**See Also:**[decode(String)](http://docs.google.com/java/awt/Font.html#decode(java.lang.String))

### hashCode

public int **hashCode**()

Returns a hashcode for this Font.

**Overrides:**[hashCode](http://docs.google.com/java/lang/Object.html#hashCode()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a hashcode value for this Font.**Since:** JDK1.0 **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)

### equals

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

Compares this Font object to the specified Object.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**obj - the Object to compare **Returns:**true if the objects are the same or if the argument is a Font object describing the same font as this object; false otherwise.**Since:** JDK1.0 **See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### toString

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

Converts this Font object to a String representation.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a String representation of this Font object.**Since:** JDK1.0

### getNumGlyphs

public int **getNumGlyphs**()

Returns the number of glyphs in this Font. Glyph codes for this Font range from 0 to getNumGlyphs() - 1.

**Returns:**the number of glyphs in this Font.**Since:** 1.2

### getMissingGlyphCode

public int **getMissingGlyphCode**()

Returns the glyphCode which is used when this Font does not have a glyph for a specified unicode code point.

**Returns:**the glyphCode of this Font.**Since:** 1.2

### getBaselineFor

public byte **getBaselineFor**(char c)

Returns the baseline appropriate for displaying this character.

Large fonts can support different writing systems, and each system can use a different baseline. The character argument determines the writing system to use. Clients should not assume all characters use the same baseline.

**Parameters:**c - a character used to identify the writing system **Returns:**the baseline appropriate for the specified character.**Since:** 1.2 **See Also:**[LineMetrics.getBaselineOffsets()](http://docs.google.com/java/awt/font/LineMetrics.html#getBaselineOffsets()), [ROMAN\_BASELINE](http://docs.google.com/java/awt/Font.html#ROMAN_BASELINE), [CENTER\_BASELINE](http://docs.google.com/java/awt/Font.html#CENTER_BASELINE), [HANGING\_BASELINE](http://docs.google.com/java/awt/Font.html#HANGING_BASELINE)

### getAttributes

public [Map](http://docs.google.com/java/util/Map.html)<[TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html),?> **getAttributes**()

Returns a map of font attributes available in this Font. Attributes include things like ligatures and glyph substitution.

**Returns:**the attributes map of this Font.

### getAvailableAttributes

public [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html)[] **getAvailableAttributes**()

Returns the keys of all the attributes supported by this Font. These attributes can be used to derive other fonts.

**Returns:**an array containing the keys of all the attributes supported by this Font.**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**(int style,  
 float size)

Creates a new Font object by replicating this Font object and applying a new style and size.

**Parameters:**style - the style for the new Fontsize - the size for the new Font **Returns:**a new Font object.**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**(int style,  
 [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) trans)

Creates a new Font object by replicating this Font object and applying a new style and transform.

**Parameters:**style - the style for the new Fonttrans - the AffineTransform associated with the new Font **Returns:**a new Font object. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if trans is null**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**(float size)

Creates a new Font object by replicating the current Font object and applying a new size to it.

**Parameters:**size - the size for the new Font. **Returns:**a new Font object.**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) trans)

Creates a new Font object by replicating the current Font object and applying a new transform to it.

**Parameters:**trans - the AffineTransform associated with the new Font **Returns:**a new Font object. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if trans is null**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**(int style)

Creates a new Font object by replicating the current Font object and applying a new style to it.

**Parameters:**style - the style for the new Font **Returns:**a new Font object.**Since:** 1.2

### deriveFont

public [Font](http://docs.google.com/java/awt/Font.html) **deriveFont**([Map](http://docs.google.com/java/util/Map.html)<? extends [AttributedCharacterIterator.Attribute](http://docs.google.com/java/text/AttributedCharacterIterator.Attribute.html),?> attributes)

Creates a new Font object by replicating the current Font object and applying a new set of font attributes to it.

**Parameters:**attributes - a map of attributes enabled for the new Font **Returns:**a new Font object.**Since:** 1.2

### canDisplay

public boolean **canDisplay**(char c)

Checks if this Font has a glyph for the specified character.

**Note:** This method cannot handle  [supplementary characters](http://docs.google.com/java/lang/Character.html#supplementary). To support all Unicode characters, including supplementary characters, use the [canDisplay(int)](http://docs.google.com/java/awt/Font.html#canDisplay(int)) method or canDisplayUpTo methods.

**Parameters:**c - the character for which a glyph is needed **Returns:**true if this Font has a glyph for this character; false otherwise.**Since:** 1.2

### canDisplay

public boolean **canDisplay**(int codePoint)

Checks if this Font has a glyph for the specified character.

**Parameters:**codePoint - the character (Unicode code point) for which a glyph is needed. **Returns:**true if this Font has a glyph for the character; false otherwise. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the code point is not a valid Unicode code point.**Since:** 1.5 **See Also:**[Character.isValidCodePoint(int)](http://docs.google.com/java/lang/Character.html#isValidCodePoint(int))

### canDisplayUpTo

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

Indicates whether or not this Font can display a specified String. For strings with Unicode encoding, it is important to know if a particular font can display the string. This method returns an offset into the String str which is the first character this Font cannot display without using the missing glyph code. If the Font can display all characters, -1 is returned.

**Parameters:**str - a String object **Returns:**an offset into str that points to the first character in str that this Font cannot display; or -1 if this Font can display all characters in str.**Since:** 1.2

### canDisplayUpTo

public int **canDisplayUpTo**(char[] text,  
 int start,  
 int limit)

Indicates whether or not this Font can display the characters in the specified text starting at start and ending at limit. This method is a convenience overload.

**Parameters:**text - the specified array of char valuesstart - the specified starting offset (in chars) into the specified array of char valueslimit - the specified ending offset (in chars) into the specified array of char values **Returns:**an offset into text that points to the first character in text that this Font cannot display; or -1 if this Font can display all characters in text.**Since:** 1.2

### canDisplayUpTo

public int **canDisplayUpTo**([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) iter,  
 int start,  
 int limit)

Indicates whether or not this Font can display the text specified by the iter starting at start and ending at limit.

**Parameters:**iter - a [CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) objectstart - the specified starting offset into the specified CharacterIterator.limit - the specified ending offset into the specified CharacterIterator. **Returns:**an offset into iter that points to the first character in iter that this Font cannot display; or -1 if this Font can display all characters in iter.**Since:** 1.2

### getItalicAngle

public float **getItalicAngle**()

Returns the italic angle of this Font. The italic angle is the inverse slope of the caret which best matches the posture of this Font.

**Returns:**the angle of the ITALIC style of this Font.**See Also:**[TextAttribute.POSTURE](http://docs.google.com/java/awt/font/TextAttribute.html#POSTURE)

### hasUniformLineMetrics

public boolean **hasUniformLineMetrics**()

Checks whether or not this Font has uniform line metrics. A logical Font might be a composite font, which means that it is composed of different physical fonts to cover different code ranges. Each of these fonts might have different LineMetrics. If the logical Font is a single font then the metrics would be uniform.

**Returns:**true if this Font has uniform line metrics; false otherwise.

### getLineMetrics

public [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) **getLineMetrics**([String](http://docs.google.com/java/lang/String.html) str,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns a [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) object created with the specified String and [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html).

**Parameters:**str - the specified Stringfrc - the specified FontRenderContext **Returns:**a LineMetrics object created with the specified String and [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html).

### getLineMetrics

public [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) **getLineMetrics**([String](http://docs.google.com/java/lang/String.html) str,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns a LineMetrics object created with the specified arguments.

**Parameters:**str - the specified StringbeginIndex - the initial offset of strlimit - the end offset of strfrc - the specified FontRenderContext **Returns:**a LineMetrics object created with the specified arguments.

### getLineMetrics

public [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) **getLineMetrics**(char[] chars,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns a LineMetrics object created with the specified arguments.

**Parameters:**chars - an array of charactersbeginIndex - the initial offset of charslimit - the end offset of charsfrc - the specified FontRenderContext **Returns:**a LineMetrics object created with the specified arguments.

### getLineMetrics

public [LineMetrics](http://docs.google.com/java/awt/font/LineMetrics.html) **getLineMetrics**([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns a LineMetrics object created with the specified arguments.

**Parameters:**ci - the specified CharacterIteratorbeginIndex - the initial offset in cilimit - the end offset of cifrc - the specified FontRenderContext **Returns:**a LineMetrics object created with the specified arguments.

### getStringBounds

public [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) **getStringBounds**([String](http://docs.google.com/java/lang/String.html) str,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns the logical bounds of the specified String in the specified FontRenderContext. The logical bounds contains the origin, ascent, advance, and height, which includes the leading. The logical bounds does not always enclose all the text. For example, in some languages and in some fonts, accent marks can be positioned above the ascent or below the descent. To obtain a visual bounding box, which encloses all the text, use the [getBounds](http://docs.google.com/java/awt/font/TextLayout.html#getBounds()) method of TextLayout.

Note: The returned bounds is in baseline-relative coordinates (see [class notes](http://docs.google.com/java/awt/Font.html)).

**Parameters:**str - the specified Stringfrc - the specified FontRenderContext **Returns:**a [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) that is the bounding box of the specified String in the specified FontRenderContext.**Since:** 1.2 **See Also:**[FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html), [createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String))

### getStringBounds

public [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) **getStringBounds**([String](http://docs.google.com/java/lang/String.html) str,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns the logical bounds of the specified String in the specified FontRenderContext. The logical bounds contains the origin, ascent, advance, and height, which includes the leading. The logical bounds does not always enclose all the text. For example, in some languages and in some fonts, accent marks can be positioned above the ascent or below the descent. To obtain a visual bounding box, which encloses all the text, use the [getBounds](http://docs.google.com/java/awt/font/TextLayout.html#getBounds()) method of TextLayout.

Note: The returned bounds is in baseline-relative coordinates (see [class notes](http://docs.google.com/java/awt/Font.html)).

**Parameters:**str - the specified StringbeginIndex - the initial offset of strlimit - the end offset of strfrc - the specified FontRenderContext **Returns:**a Rectangle2D that is the bounding box of the specified String in the specified FontRenderContext. **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if beginIndex is less than zero, or limit is greater than the length of str, or beginIndex is greater than limit.**Since:** 1.2 **See Also:**[FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html), [createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String))

### getStringBounds

public [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) **getStringBounds**(char[] chars,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns the logical bounds of the specified array of characters in the specified FontRenderContext. The logical bounds contains the origin, ascent, advance, and height, which includes the leading. The logical bounds does not always enclose all the text. For example, in some languages and in some fonts, accent marks can be positioned above the ascent or below the descent. To obtain a visual bounding box, which encloses all the text, use the [getBounds](http://docs.google.com/java/awt/font/TextLayout.html#getBounds()) method of TextLayout.

Note: The returned bounds is in baseline-relative coordinates (see [class notes](http://docs.google.com/java/awt/Font.html)).

**Parameters:**chars - an array of charactersbeginIndex - the initial offset in the array of characterslimit - the end offset in the array of charactersfrc - the specified FontRenderContext **Returns:**a Rectangle2D that is the bounding box of the specified array of characters in the specified FontRenderContext. **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if beginIndex is less than zero, or limit is greater than the length of chars, or beginIndex is greater than limit.**Since:** 1.2 **See Also:**[FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html), [createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String))

### getStringBounds

public [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) **getStringBounds**([CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci,  
 int beginIndex,  
 int limit,  
 [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns the logical bounds of the characters indexed in the specified [CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) in the specified FontRenderContext. The logical bounds contains the origin, ascent, advance, and height, which includes the leading. The logical bounds does not always enclose all the text. For example, in some languages and in some fonts, accent marks can be positioned above the ascent or below the descent. To obtain a visual bounding box, which encloses all the text, use the [getBounds](http://docs.google.com/java/awt/font/TextLayout.html#getBounds()) method of TextLayout.

Note: The returned bounds is in baseline-relative coordinates (see [class notes](http://docs.google.com/java/awt/Font.html)).

**Parameters:**ci - the specified CharacterIteratorbeginIndex - the initial offset in cilimit - the end offset in cifrc - the specified FontRenderContext **Returns:**a Rectangle2D that is the bounding box of the characters indexed in the specified CharacterIterator in the specified FontRenderContext. **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if beginIndex is less than the start index of ci, or limit is greater than the end index of ci, or beginIndex is greater than limit**Since:** 1.2 **See Also:**[FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html), [createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String))

### getMaxCharBounds

public [Rectangle2D](http://docs.google.com/java/awt/geom/Rectangle2D.html) **getMaxCharBounds**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc)

Returns the bounds for the character with the maximum bounds as defined in the specified FontRenderContext.

Note: The returned bounds is in baseline-relative coordinates (see [class notes](http://docs.google.com/java/awt/Font.html)).

**Parameters:**frc - the specified FontRenderContext **Returns:**a Rectangle2D that is the bounding box for the character with the maximum bounds.

### createGlyphVector

public [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) **createGlyphVector**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc,  
 [String](http://docs.google.com/java/lang/String.html) str)

Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. This method does no other processing besides the mapping of glyphs to characters. This means that this method is not useful for some scripts, such as Arabic, Hebrew, Thai, and Indic, that require reordering, shaping, or ligature substitution.

**Parameters:**frc - the specified FontRenderContextstr - the specified String **Returns:**a new GlyphVector created with the specified String and the specified FontRenderContext.

### createGlyphVector

public [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) **createGlyphVector**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc,  
 char[] chars)

Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. This method does no other processing besides the mapping of glyphs to characters. This means that this method is not useful for some scripts, such as Arabic, Hebrew, Thai, and Indic, that require reordering, shaping, or ligature substitution.

**Parameters:**frc - the specified FontRenderContextchars - the specified array of characters **Returns:**a new GlyphVector created with the specified array of characters and the specified FontRenderContext.

### createGlyphVector

public [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) **createGlyphVector**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc,  
 [CharacterIterator](http://docs.google.com/java/text/CharacterIterator.html) ci)

Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping the specified characters to glyphs one-to-one based on the Unicode cmap in this Font. This method does no other processing besides the mapping of glyphs to characters. This means that this method is not useful for some scripts, such as Arabic, Hebrew, Thai, and Indic, that require reordering, shaping, or ligature substitution.

**Parameters:**frc - the specified FontRenderContextci - the specified CharacterIterator **Returns:**a new GlyphVector created with the specified CharacterIterator and the specified FontRenderContext.

### createGlyphVector

public [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) **createGlyphVector**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc,  
 int[] glyphCodes)

Creates a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font. This method does no other processing besides the mapping of glyphs to characters. This means that this method is not useful for some scripts, such as Arabic, Hebrew, Thai, and Indic, that require reordering, shaping, or ligature substitution.

**Parameters:**frc - the specified FontRenderContextglyphCodes - the specified integer array **Returns:**a new GlyphVector created with the specified integer array and the specified FontRenderContext.

### layoutGlyphVector

public [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) **layoutGlyphVector**([FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) frc,  
 char[] text,  
 int start,  
 int limit,  
 int flags)

Returns a new GlyphVector object, performing full layout of the text if possible. Full layout is required for complex text, such as Arabic or Hindi. Support for different scripts depends on the font and implementation.

Layout requires bidi analysis, as performed by Bidi, and should only be performed on text that has a uniform direction. The direction is indicated in the flags parameter,by using LAYOUT\_RIGHT\_TO\_LEFT to indicate a right-to-left (Arabic and Hebrew) run direction, or LAYOUT\_LEFT\_TO\_RIGHT to indicate a left-to-right (English) run direction.

In addition, some operations, such as Arabic shaping, require context, so that the characters at the start and limit can have the proper shapes. Sometimes the data in the buffer outside the provided range does not have valid data. The values LAYOUT\_NO\_START\_CONTEXT and LAYOUT\_NO\_LIMIT\_CONTEXT can be added to the flags parameter to indicate that the text before start, or after limit, respectively, should not be examined for context.

All other values for the flags parameter are reserved.

**Parameters:**frc - the specified FontRenderContexttext - the text to layoutstart - the start of the text to use for the GlyphVectorlimit - the limit of the text to use for the GlyphVectorflags - control flags as described above **Returns:**a new GlyphVector representing the text between start and limit, with glyphs chosen and positioned so as to best represent the text **Throws:** [ArrayIndexOutOfBoundsException](http://docs.google.com/java/lang/ArrayIndexOutOfBoundsException.html) - if start or limit is out of bounds**Since:** 1.4 **See Also:**[Bidi](http://docs.google.com/java/text/Bidi.html), [LAYOUT\_LEFT\_TO\_RIGHT](http://docs.google.com/java/awt/Font.html#LAYOUT_LEFT_TO_RIGHT), [LAYOUT\_RIGHT\_TO\_LEFT](http://docs.google.com/java/awt/Font.html#LAYOUT_RIGHT_TO_LEFT), [LAYOUT\_NO\_START\_CONTEXT](http://docs.google.com/java/awt/Font.html#LAYOUT_NO_START_CONTEXT), [LAYOUT\_NO\_LIMIT\_CONTEXT](http://docs.google.com/java/awt/Font.html#LAYOUT_NO_LIMIT_CONTEXT)

### finalize

protected void **finalize**()  
 throws [Throwable](http://docs.google.com/java/lang/Throwable.html)

Disposes the native Font object.

**Overrides:**[finalize](http://docs.google.com/java/lang/Object.html#finalize()) in class [Object](http://docs.google.com/java/lang/Object.html) **Throws:** [Throwable](http://docs.google.com/java/lang/Throwable.html) - the Exception raised by this method

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Font.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/awt/FocusTraversalPolicy.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/FontFormatException.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/Font.html)    [**NO FRAMES**](http://docs.google.com/Font.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#49x2ik5) | [METHOD](#23ckvvd) |

[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).