AWT FONT CLASS

Introduction

The Font class states fonts, which are used to render text in a visible way.

Class declaration

Following is the declaration for **java.awt.Font** class:

```
public class Font
   extends Object
      implements Serializable
```

Field

Following are the fields for **java.awt.geom.Arc2D** class:

- **static int BOLD** -- The bold style constant.
- **static int CENTER_BASELINE** --The baseline used in ideographic scripts like Chinese, Japanese, and Korean when laying out text.
- **static String DIALOG** --A String constant for the canonical family name of the logical font "Dialog".
- **static String DIALOG_INPUT** --A String constant for the canonical family name of the logical font "DialogInput".
- **static int HANGING_BASELINE** -- The baseline used in Devanigiri and similar scripts when laying out text.
- **static int ITALIC** -- The italicized style constant.
- **static int 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** -- 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** -- 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** -- A flag to layoutGlyphVector indicating that text is right-to-left as determined by Bidi analysis.
- **static String MONOSPACED** -- A String constant for the canonical family name of the logical font "Monospaced".
- **protected String name** -- The logical name of this Font, as passed to the constructor.
- **static int PLAIN** --The plain style constant.
- **protected float pointSize** -- The point size of this Font in float.
- **static int ROMAN_BASELINE** --The baseline used in most Roman scripts when laying out text.
- **static String SANS_SERIF** -- A String constant for the canonical family name of the logical font "SansSerif".
- **static String SERIF** -- A String constant for the canonical family name of the logical font "Serif".

- **protected int size** --The point size of this Font, rounded to integer.
- **protected int style** -- The style of this Font, as passed to the constructor.
- **static int TRUETYPE_FONT** -- Identify a font resource of type TRUETYPE.
- **static int TYPE1_FONT** -- Identify a font resource of type TYPE1.

Class constructors

S.N.	Constructor & Description
1	**protected Font** Creates a new Font from the specified font.
2	**Font***Map < ?extendsAttributedCharacterIterator. Attribute, ? > attributes* Creates a new Font from the specified font.
3	**Font***Stringname, intstyle, intsize* Creates a new Font from the specified font.

Class methods

S.N.	Method & Description
1	**boolean canDisplay***charc* Checks if this Font has a glyph for the specified character.
2	**boolean canDisplay***intcodePoint* Checks if this Font has a glyph for the specified character.
3	**int canDisplayUpTo***char[]text, intstart, intlimit* Indicates whether or not this Font can display the characters in the specified text starting at start and ending at limit.
4	**int canDisplayUpTo***CharacterIteratoriter, intstart, intlimit* Indicates whether or not this Font can display the text specified by the iter starting at start and ending at limit.
5	**int canDisplayUpTo***Stringstr* Indicates whether or not this Font can display a specified String.

6

static Font createFont*intfontFormat, FilefontFile*

Returns a new Font using the specified font type and the specified font file.

7

static Font createFont*intfontFormat, InputStreamfontStream*

Returns a new Font using the specified font type and input data.

8

GlyphVector createGlyphVector*FontRenderContextfrc, char[]chars*

Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.

9

GlyphVector createGlyphVector*FontRenderContextfrc, CharacterIteratorci*

Creates a GlyphVector by mapping the specified characters to glyphs one-to-one based on the Unicode cmap in this Font.

10

GlyphVector createGlyphVector*FontRenderContextfrc, int[]glyphCodes*

Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.

11

GlyphVector createGlyphVector*FontRenderContextfrc, Stringstr*

Creates a GlyphVector by mapping characters to glyphs one-to-one based on the Unicode cmap in this Font.

12

static Font decode*Stringstr*

Returns the Font that the str argument describes.

13

Font deriveFont*AffineTransformtrans*

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

14

Font deriveFont*floatsize*

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

15

Font deriveFont*intstyle*

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

16

Font deriveFont*intstyle, AffineTransformtrans*

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

17

Font deriveFont*intstyle, floatsize*

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

18

Font deriveFont*Map < ?extendsAttributedCharacterIterator. Attribute, ? > attributes*

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

19

boolean equals*Objectobj*

Compares this Font object to the specified Object.

20

protected void finalize

Disposes the native Font object.

21

Map<TextAttribute,?> getAttributes

Returns a map of font attributes available in this Font.

22

AttributedCharacterIterator.Attribute[] getAvailableAttributes

Returns the keys of all the attributes supported by this Font.

23

byte getBaselineFor*charc*

Returns the baseline appropriate for displaying this character.

24

String getFamily

Returns the family name of this Font.

25

String getFamily*Localel*

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

26

static Font getFont*Map < ?extendsAttributedCharacterIterator. Attribute, ? > attributes*

Returns a Font appropriate to the attributes.

27

static Font getFont*Stringnm*

Returns a Font object fom the system properties list.

28

static Font getFont*Stringnm, Fontfont*

Gets the specified Font from the system properties list.

29

String getFontName

Returns the font face name of this Font.

30

String getFontName*Localel*

Returns the font face name of the Font, localized for the specified locale.

31

float getItalicAngle

Returns the italic angle of this Font.

32

LineMetrics getLineMetrics*char[]chars, intbeginIndex, intlimit, FontRenderContextfrc*

Returns a LineMetrics object created with the specified arguments.

33

LineMetrics getLineMetrics*CharacterIteratorci, intbeginIndex, intlimit, FontRenderContextfrc*

Returns a LineMetrics object created with the specified arguments.

34

LineMetrics getLineMetrics*Stringstr, FontRenderContextfrc*

Returns a LineMetrics object created with the specified String and FontRenderContext.

35

LineMetrics getLineMetrics*Stringstr, intbeginIndex, intlimit, FontRenderContextfrc*

Returns a LineMetrics object created with the specified arguments.

36

Rectangle2D getMaxCharBounds*FontRenderContextfrc*

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

37

int getMissingGlyphCode

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

38

String getName

Returns the logical name of this Font.

39

int getNumGlyphs

Returns the number of glyphs in this Font.

40

java.awt.peer.FontPeer getPeer

Deprecated. Font rendering is now platform independent.

41

String getPSName

Returns the postscript name of this Font.

42

int getSize

Returns the point size of this Font, rounded to an integer.

43

float getSize2D

Returns the point size of this Font in float value.

44

Rectangle2D getStringBounds*char[]chars, intbeginIndex, intlimit, FontRenderContextfrc*

Returns the logical bounds of the specified array of characters in the specified FontRenderContext.

45

Rectangle2D getStringBounds*CharacterIteratorci, intbeginIndex, intlimit, FontRenderContextfrc*

Returns the logical bounds of the characters indexed in the specified CharacterIterator in the specified FontRenderContext.

46

Rectangle2D getStringBounds*Stringstr, FontRenderContextfrc*

Returns the logical bounds of the specified String in the specified FontRenderContext.

47

Rectangle2D getStringBounds*Stringstr, intbeginIndex, intlimit, FontRenderContextfrc*

Returns the logical bounds of the specified String in the specified FontRenderContext.

48

int getStyle

Returns the style of this Font.

49

AffineTransform getTransform

Returns a copy of the transform associated with this Font.

50

int hashCode

Returns a hashcode for this Font.

51 **boolean hasLayoutAttributes**

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

52 **boolean hasUniformLineMetrics**

Checks whether or not this Font has uniform line metrics.

53 **boolean isBold**

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

54 **boolean isItalic**

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

55 **boolean isPlain**

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

56 **boolean isTransformed**

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

57 **GlyphVector layoutGlyphVector***FontRenderContextfrc, char*[]*text, intstart, intlimit, intflags*

Returns a new GlyphVector object, performing full layout of the text if possible.

58 **String toString**

Converts this Font object to a String representation.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

Font Example

Create the following java program using any editor of your choice in say **D:/ > AWT > com > tutorialspoint > gui >**

AWTGraphicsDemo.java

```
package com.tutorialspoint.gui;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
```

```
public class AWTGraphicsDemo extends Frame {

   public AWTGraphicsDemo(){
      super("Java AWT Examples");
      prepareGUI();
   }

   public static void main(String[] args){
      AWTGraphicsDemo  awtGraphicsDemo = new AWTGraphicsDemo();
      awtGraphicsDemo.setVisible(true);
   }

   private void prepareGUI(){
      setSize(400,400);
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
   }

   @Override
   public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D)g;
      Font plainFont = new Font("Serif", Font.PLAIN, 24);
      g2.setFont(plainFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 70);
      Font italicFont = new Font("Serif", Font.ITALIC, 24);
      g2.setFont(italicFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 120);
      Font boldFont = new Font("Serif", Font.BOLD, 24);
      g2.setFont(boldFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 170);
      Font boldItalicFont = new Font("Serif", Font.BOLD+Font.ITALIC, 24);
      g2.setFont(boldItalicFont);
      g2.drawString("Welcome to TutorialsPoint", 50, 220);
   }
}
```

Compile the program using command prompt. Go to **D:/ > AWT** and type the following command.

```
D:\AWT>javac com\tutorialspoint\gui\AWTGraphicsDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

```
D:\AWT>java com.tutorialspoint.gui.AWTGraphicsDemo
```

Verify the following output