| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Graphics2D.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/Graphics.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/GraphicsConfigTemplate.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/Graphics2D.html)    [**NO FRAMES**](http://docs.google.com/Graphics2D.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#1t3h5sf) | [METHOD](#4d34og8) | DETAIL: FIELD | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

## **java.awt**

Class Graphics2D

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.awt.Graphics](http://docs.google.com/java/awt/Graphics.html)  
 **java.awt.Graphics2D**

public abstract class **Graphics2D**extends [Graphics](http://docs.google.com/java/awt/Graphics.html)

This Graphics2D class extends the [Graphics](http://docs.google.com/java/awt/Graphics.html) class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. This is the fundamental class for rendering 2-dimensional shapes, text and images on the Java(tm) platform.

## Coordinate Spaces

All coordinates passed to a Graphics2D object are specified in a device-independent coordinate system called User Space, which is used by applications. The Graphics2D object contains an [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) object as part of its rendering state that defines how to convert coordinates from user space to device-dependent coordinates in Device Space.

Coordinates in device space usually refer to individual device pixels and are aligned on the infinitely thin gaps between these pixels. Some Graphics2D objects can be used to capture rendering operations for storage into a graphics metafile for playback on a concrete device of unknown physical resolution at a later time. Since the resolution might not be known when the rendering operations are captured, the Graphics2D Transform is set up to transform user coordinates to a virtual device space that approximates the expected resolution of the target device. Further transformations might need to be applied at playback time if the estimate is incorrect.

Some of the operations performed by the rendering attribute objects occur in the device space, but all Graphics2D methods take user space coordinates.

Every Graphics2D object is associated with a target that defines where rendering takes place. A [GraphicsConfiguration](http://docs.google.com/java/awt/GraphicsConfiguration.html) object defines the characteristics of the rendering target, such as pixel format and resolution. The same rendering target is used throughout the life of a Graphics2D object.

When creating a Graphics2D object, the GraphicsConfiguration specifies the default transform for the target of the Graphics2D (a [Component](http://docs.google.com/java/awt/Component.html) or [Image](http://docs.google.com/java/awt/Image.html)). This default transform maps the user space coordinate system to screen and printer device coordinates such that the origin maps to the upper left hand corner of the target region of the device with increasing X coordinates extending to the right and increasing Y coordinates extending downward. The scaling of the default transform is set to identity for those devices that are close to 72 dpi, such as screen devices. The scaling of the default transform is set to approximately 72 user space coordinates per square inch for high resolution devices, such as printers. For image buffers, the default transform is the Identity transform.

## Rendering Process

The Rendering Process can be broken down into four phases that are controlled by the Graphics2D rendering attributes. The renderer can optimize many of these steps, either by caching the results for future calls, by collapsing multiple virtual steps into a single operation, or by recognizing various attributes as common simple cases that can be eliminated by modifying other parts of the operation.

The steps in the rendering process are:

1. Determine what to render.
2. Constrain the rendering operation to the current Clip. The Clip is specified by a [Shape](http://docs.google.com/java/awt/Shape.html) in user space and is controlled by the program using the various clip manipulation methods of Graphics and Graphics2D. This *user clip* is transformed into device space by the current Transform and combined with the *device clip*, which is defined by the visibility of windows and device extents. The combination of the user clip and device clip defines the *composite clip*, which determines the final clipping region. The user clip is not modified by the rendering system to reflect the resulting composite clip.
3. Determine what colors to render.
4. Apply the colors to the destination drawing surface using the current [Composite](http://docs.google.com/java/awt/Composite.html) attribute in the Graphics2D context.

The three types of rendering operations, along with details of each of their particular rendering processes are:

1. **Shape operations**
   1. If the operation is a draw(Shape) operation, then the [createStrokedShape](http://docs.google.com/java/awt/Stroke.html#createStrokedShape(java.awt.Shape)) method on the current [Stroke](http://docs.google.com/java/awt/Stroke.html) attribute in the Graphics2D context is used to construct a new Shape object that contains the outline of the specified Shape.
   2. The Shape is transformed from user space to device space using the current Transform in the Graphics2D context.
   3. The outline of the Shape is extracted using the [getPathIterator](http://docs.google.com/java/awt/Shape.html#getPathIterator(java.awt.geom.AffineTransform)) method of Shape, which returns a [PathIterator](http://docs.google.com/java/awt/geom/PathIterator.html) object that iterates along the boundary of the Shape.
   4. If the Graphics2D object cannot handle the curved segments that the PathIterator object returns then it can call the alternate [getPathIterator](http://docs.google.com/java/awt/Shape.html#getPathIterator(java.awt.geom.AffineTransform,%20double)) method of Shape, which flattens the Shape.
   5. The current [Paint](http://docs.google.com/java/awt/Paint.html) in the Graphics2D context is queried for a [PaintContext](http://docs.google.com/java/awt/PaintContext.html), which specifies the colors to render in device space.

1. **Text operations**
   1. The following steps are used to determine the set of glyphs required to render the indicated String:
      1. If the argument is a String, then the current Font in the Graphics2D context is asked to convert the Unicode characters in the String into a set of glyphs for presentation with whatever basic layout and shaping algorithms the font implements.
      2. If the argument is an [AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html), the iterator is asked to convert itself to a [TextLayout](http://docs.google.com/java/awt/font/TextLayout.html) using its embedded font attributes. The TextLayout implements more sophisticated glyph layout algorithms that perform Unicode bi-directional layout adjustments automatically for multiple fonts of differing writing directions.
      3. If the argument is a [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html), then the GlyphVector object already contains the appropriate font-specific glyph codes with explicit coordinates for the position of each glyph.
   2. The current Font is queried to obtain outlines for the indicated glyphs. These outlines are treated as shapes in user space relative to the position of each glyph that was determined in step 1.
   3. The character outlines are filled as indicated above under [Shape operations](#2et92p0).
   4. The current Paint is queried for a PaintContext, which specifies the colors to render in device space.

1. **Image Operations**
   1. The region of interest is defined by the bounding box of the source Image. This bounding box is specified in Image Space, which is the Image object's local coordinate system.
   2. If an AffineTransform is passed to [drawImage(Image, AffineTransform, ImageObserver)](http://docs.google.com/java/awt/Graphics2D.html#drawImage(java.awt.Image,%20java.awt.geom.AffineTransform,%20java.awt.image.ImageObserver)), the AffineTransform is used to transform the bounding box from image space to user space. If no AffineTransform is supplied, the bounding box is treated as if it is already in user space.
   3. The bounding box of the source Image is transformed from user space into device space using the current Transform. Note that the result of transforming the bounding box does not necessarily result in a rectangular region in device space.
   4. The Image object determines what colors to render, sampled according to the source to destination coordinate mapping specified by the current Transform and the optional image transform.

## Default Rendering Attributes

The default values for the Graphics2D rendering attributes are: *Paint* The color of the Component. *Font* The Font of the Component. *Stroke* A square pen with a linewidth of 1, no dashing, miter segment joins and square end caps. *Transform* The [getDefaultTransform](http://docs.google.com/java/awt/GraphicsConfiguration.html#getDefaultTransform()) for the GraphicsConfiguration of the Component. *Composite* The [AlphaComposite.SRC\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER) rule. *Clip* No rendering Clip, the output is clipped to the Component.

## Rendering Compatibility Issues

The JDK(tm) 1.1 rendering model is based on a pixelization model that specifies that coordinates are infinitely thin, lying between the pixels. Drawing operations are performed using a one-pixel wide pen that fills the pixel below and to the right of the anchor point on the path. The JDK 1.1 rendering model is consistent with the capabilities of most of the existing class of platform renderers that need to resolve integer coordinates to a discrete pen that must fall completely on a specified number of pixels.

The Java 2D(tm) (Java(tm) 2 platform) API supports antialiasing renderers. A pen with a width of one pixel does not need to fall completely on pixel N as opposed to pixel N+1. The pen can fall partially on both pixels. It is not necessary to choose a bias direction for a wide pen since the blending that occurs along the pen traversal edges makes the sub-pixel position of the pen visible to the user. On the other hand, when antialiasing is turned off by setting the [KEY\_ANTIALIASING](http://docs.google.com/java/awt/RenderingHints.html#KEY_ANTIALIASING) hint key to the [VALUE\_ANTIALIAS\_OFF](http://docs.google.com/java/awt/RenderingHints.html#VALUE_ANTIALIAS_OFF) hint value, the renderer might need to apply a bias to determine which pixel to modify when the pen is straddling a pixel boundary, such as when it is drawn along an integer coordinate in device space. While the capabilities of an antialiasing renderer make it no longer necessary for the rendering model to specify a bias for the pen, it is desirable for the antialiasing and non-antialiasing renderers to perform similarly for the common cases of drawing one-pixel wide horizontal and vertical lines on the screen. To ensure that turning on antialiasing by setting the [KEY\_ANTIALIASING](http://docs.google.com/java/awt/RenderingHints.html#KEY_ANTIALIASING) hint key to [VALUE\_ANTIALIAS\_ON](http://docs.google.com/java/awt/RenderingHints.html#VALUE_ANTIALIAS_ON) does not cause such lines to suddenly become twice as wide and half as opaque, it is desirable to have the model specify a path for such lines so that they completely cover a particular set of pixels to help increase their crispness.

Java 2D API maintains compatibility with JDK 1.1 rendering behavior, such that legacy operations and existing renderer behavior is unchanged under Java 2D API. Legacy methods that map onto general draw and fill methods are defined, which clearly indicates how Graphics2D extends Graphics based on settings of Stroke and Transform attributes and rendering hints. The definition performs identically under default attribute settings. For example, the default Stroke is a BasicStroke with a width of 1 and no dashing and the default Transform for screen drawing is an Identity transform.

The following two rules provide predictable rendering behavior whether aliasing or antialiasing is being used.

* Device coordinates are defined to be between device pixels which avoids any inconsistent results between aliased and antaliased rendering. If coordinates were defined to be at a pixel's center, some of the pixels covered by a shape, such as a rectangle, would only be half covered. With aliased rendering, the half covered pixels would either be rendered inside the shape or outside the shape. With anti-aliased rendering, the pixels on the entire edge of the shape would be half covered. On the other hand, since coordinates are defined to be between pixels, a shape like a rectangle would have no half covered pixels, whether or not it is rendered using antialiasing.
* Lines and paths stroked using the BasicStroke object may be "normalized" to provide consistent rendering of the outlines when positioned at various points on the drawable and whether drawn with aliased or antialiased rendering. This normalization process is controlled by the [KEY\_STROKE\_CONTROL](http://docs.google.com/java/awt/RenderingHints.html#KEY_STROKE_CONTROL) hint. The exact normalization algorithm is not specified, but the goals of this normalization are to ensure that lines are rendered with consistent visual appearance regardless of how they fall on the pixel grid and to promote more solid horizontal and vertical lines in antialiased mode so that they resemble their non-antialiased counterparts more closely. A typical normalization step might promote antialiased line endpoints to pixel centers to reduce the amount of blending or adjust the subpixel positioning of non-antialiased lines so that the floating point line widths round to even or odd pixel counts with equal likelihood. This process can move endpoints by up to half a pixel (usually towards positive infinity along both axes) to promote these consistent results.

The following definitions of general legacy methods perform identically to previously specified behavior under default attribute settings:

* For fill operations, including fillRect, fillRoundRect, fillOval, fillArc, fillPolygon, and clearRect, [fill](http://docs.google.com/java/awt/Graphics2D.html#fill(java.awt.Shape)) can now be called with the desired Shape. For example, when filling a rectangle:  
   fill(new Rectangle(x, y, w, h));  
   is called.
* Similarly, for draw operations, including drawLine, drawRect, drawRoundRect, drawOval, drawArc, drawPolyline, and drawPolygon, [draw](http://docs.google.com/java/awt/Graphics2D.html#draw(java.awt.Shape)) can now be called with the desired Shape. For example, when drawing a rectangle:  
   draw(new Rectangle(x, y, w, h));  
   is called.
* The draw3DRect and fill3DRect methods were implemented in terms of the drawLine and fillRect methods in the Graphics class which would predicate their behavior upon the current Stroke and Paint objects in a Graphics2D context. This class overrides those implementations with versions that use the current Color exclusively, overriding the current Paint and which uses fillRect to describe the exact same behavior as the preexisting methods regardless of the setting of the current Stroke.

The Graphics class defines only the setColor method to control the color to be painted. Since the Java 2D API extends the Color object to implement the new Paint interface, the existing setColor method is now a convenience method for setting the current Paint attribute to a Color object. setColor(c) is equivalent to setPaint(c).

The Graphics class defines two methods for controlling how colors are applied to the destination.

1. The setPaintMode method is implemented as a convenience method to set the default Composite, equivalent to setComposite(new AlphaComposite.SrcOver).
2. The setXORMode(Color xorcolor) method is implemented as a convenience method to set a special Composite object that ignores the Alpha components of source colors and sets the destination color to the value:  
    dstpixel = (PixelOf(srccolor) ^ PixelOf(xorcolor) ^ dstpixel);

**See Also:**[RenderingHints](http://docs.google.com/java/awt/RenderingHints.html)

| **Constructor Summary** | |
| --- | --- |
| protected | [**Graphics2D**](http://docs.google.com/java/awt/Graphics2D.html#Graphics2D())()            Constructs a new Graphics2D object. |

| **Method Summary** | |
| --- | --- |
| abstract  void | [**addRenderingHints**](http://docs.google.com/java/awt/Graphics2D.html#addRenderingHints(java.util.Map))([Map](http://docs.google.com/java/util/Map.html)<?,?> hints)            Sets the values of an arbitrary number of preferences for the rendering algorithms. |
| abstract  void | [**clip**](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape))([Shape](http://docs.google.com/java/awt/Shape.html) s)            Intersects the current Clip with the interior of the specified Shape and sets the Clip to the resulting intersection. |
| abstract  void | [**draw**](http://docs.google.com/java/awt/Graphics2D.html#draw(java.awt.Shape))([Shape](http://docs.google.com/java/awt/Shape.html) s)            Strokes the outline of a Shape using the settings of the current Graphics2D context. |
| void | [**draw3DRect**](http://docs.google.com/java/awt/Graphics2D.html#draw3DRect(int,%20int,%20int,%20int,%20boolean))(int x, int y, int width, int height, boolean raised)            Draws a 3-D highlighted outline of the specified rectangle. |
| abstract  void | [**drawGlyphVector**](http://docs.google.com/java/awt/Graphics2D.html#drawGlyphVector(java.awt.font.GlyphVector,%20float,%20float))([GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) g, float x, float y)            Renders the text of the specified [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) using the Graphics2D context's rendering attributes. |
| abstract  void | [**drawImage**](http://docs.google.com/java/awt/Graphics2D.html#drawImage(java.awt.image.BufferedImage,%20java.awt.image.BufferedImageOp,%20int,%20int))([BufferedImage](http://docs.google.com/java/awt/image/BufferedImage.html) img, [BufferedImageOp](http://docs.google.com/java/awt/image/BufferedImageOp.html) op, int x, int y)            Renders a BufferedImage that is filtered with a [BufferedImageOp](http://docs.google.com/java/awt/image/BufferedImageOp.html). |
| abstract  boolean | [**drawImage**](http://docs.google.com/java/awt/Graphics2D.html#drawImage(java.awt.Image,%20java.awt.geom.AffineTransform,%20java.awt.image.ImageObserver))([Image](http://docs.google.com/java/awt/Image.html) img, [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform, [ImageObserver](http://docs.google.com/java/awt/image/ImageObserver.html) obs)            Renders an image, applying a transform from image space into user space before drawing. |
| abstract  void | [**drawRenderableImage**](http://docs.google.com/java/awt/Graphics2D.html#drawRenderableImage(java.awt.image.renderable.RenderableImage,%20java.awt.geom.AffineTransform))([RenderableImage](http://docs.google.com/java/awt/image/renderable/RenderableImage.html) img, [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform)            Renders a [RenderableImage](http://docs.google.com/java/awt/image/renderable/RenderableImage.html), applying a transform from image space into user space before drawing. |
| abstract  void | [**drawRenderedImage**](http://docs.google.com/java/awt/Graphics2D.html#drawRenderedImage(java.awt.image.RenderedImage,%20java.awt.geom.AffineTransform))([RenderedImage](http://docs.google.com/java/awt/image/RenderedImage.html) img, [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform)            Renders a [RenderedImage](http://docs.google.com/java/awt/image/RenderedImage.html), applying a transform from image space into user space before drawing. |
| abstract  void | [**drawString**](http://docs.google.com/java/awt/Graphics2D.html#drawString(java.text.AttributedCharacterIterator,%20float,%20float))([AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) iterator, float x, float y)            Renders the text of the specified iterator applying its attributes in accordance with the specification of the [TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html) class. |
| abstract  void | [**drawString**](http://docs.google.com/java/awt/Graphics2D.html#drawString(java.text.AttributedCharacterIterator,%20int,%20int))([AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) iterator, int x, int y)            Renders the text of the specified iterator applying its attributes in accordance with the specification of the [TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html) class. |
| abstract  void | [**drawString**](http://docs.google.com/java/awt/Graphics2D.html#drawString(java.lang.String,%20float,%20float))([String](http://docs.google.com/java/lang/String.html) str, float x, float y)            Renders the text specified by the specified String, using the current text attribute state in the Graphics2D context. |
| abstract  void | [**drawString**](http://docs.google.com/java/awt/Graphics2D.html#drawString(java.lang.String,%20int,%20int))([String](http://docs.google.com/java/lang/String.html) str, int x, int y)            Renders the text of the specified String, using the current text attribute state in the Graphics2D context. |
| abstract  void | [**fill**](http://docs.google.com/java/awt/Graphics2D.html#fill(java.awt.Shape))([Shape](http://docs.google.com/java/awt/Shape.html) s)            Fills the interior of a Shape using the settings of the Graphics2D context. |
| void | [**fill3DRect**](http://docs.google.com/java/awt/Graphics2D.html#fill3DRect(int,%20int,%20int,%20int,%20boolean))(int x, int y, int width, int height, boolean raised)            Paints a 3-D highlighted rectangle filled with the current color. |
| abstract  [Color](http://docs.google.com/java/awt/Color.html) | [**getBackground**](http://docs.google.com/java/awt/Graphics2D.html#getBackground())()            Returns the background color used for clearing a region. |
| abstract  [Composite](http://docs.google.com/java/awt/Composite.html) | [**getComposite**](http://docs.google.com/java/awt/Graphics2D.html#getComposite())()            Returns the current Composite in the Graphics2D context. |
| abstract  [GraphicsConfiguration](http://docs.google.com/java/awt/GraphicsConfiguration.html) | [**getDeviceConfiguration**](http://docs.google.com/java/awt/Graphics2D.html#getDeviceConfiguration())()            Returns the device configuration associated with this Graphics2D. |
| abstract  [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) | [**getFontRenderContext**](http://docs.google.com/java/awt/Graphics2D.html#getFontRenderContext())()            Get the rendering context of the Font within this Graphics2D context. |
| abstract  [Paint](http://docs.google.com/java/awt/Paint.html) | [**getPaint**](http://docs.google.com/java/awt/Graphics2D.html#getPaint())()            Returns the current Paint of the Graphics2D context. |
| abstract  [Object](http://docs.google.com/java/lang/Object.html) | [**getRenderingHint**](http://docs.google.com/java/awt/Graphics2D.html#getRenderingHint(java.awt.RenderingHints.Key))([RenderingHints.Key](http://docs.google.com/java/awt/RenderingHints.Key.html) hintKey)            Returns the value of a single preference for the rendering algorithms. |
| abstract  [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html) | [**getRenderingHints**](http://docs.google.com/java/awt/Graphics2D.html#getRenderingHints())()            Gets the preferences for the rendering algorithms. |
| abstract  [Stroke](http://docs.google.com/java/awt/Stroke.html) | [**getStroke**](http://docs.google.com/java/awt/Graphics2D.html#getStroke())()            Returns the current Stroke in the Graphics2D context. |
| abstract  [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getTransform**](http://docs.google.com/java/awt/Graphics2D.html#getTransform())()            Returns a copy of the current Transform in the Graphics2D context. |
| abstract  boolean | [**hit**](http://docs.google.com/java/awt/Graphics2D.html#hit(java.awt.Rectangle,%20java.awt.Shape,%20boolean))([Rectangle](http://docs.google.com/java/awt/Rectangle.html) rect, [Shape](http://docs.google.com/java/awt/Shape.html) s, boolean onStroke)            Checks whether or not the specified Shape intersects the specified [Rectangle](http://docs.google.com/java/awt/Rectangle.html), which is in device space. |
| abstract  void | [**rotate**](http://docs.google.com/java/awt/Graphics2D.html#rotate(double))(double theta)            Concatenates the current Graphics2D Transform with a rotation transform. |
| abstract  void | [**rotate**](http://docs.google.com/java/awt/Graphics2D.html#rotate(double,%20double,%20double))(double theta, double x, double y)            Concatenates the current Graphics2D Transform with a translated rotation transform. |
| abstract  void | [**scale**](http://docs.google.com/java/awt/Graphics2D.html#scale(double,%20double))(double sx, double sy)            Concatenates the current Graphics2D Transform with a scaling transformation Subsequent rendering is resized according to the specified scaling factors relative to the previous scaling. |
| abstract  void | [**setBackground**](http://docs.google.com/java/awt/Graphics2D.html#setBackground(java.awt.Color))([Color](http://docs.google.com/java/awt/Color.html) color)            Sets the background color for the Graphics2D context. |
| abstract  void | [**setComposite**](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite))([Composite](http://docs.google.com/java/awt/Composite.html) comp)            Sets the Composite for the Graphics2D context. |
| abstract  void | [**setPaint**](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint))([Paint](http://docs.google.com/java/awt/Paint.html) paint)            Sets the Paint attribute for the Graphics2D context. |
| abstract  void | [**setRenderingHint**](http://docs.google.com/java/awt/Graphics2D.html#setRenderingHint(java.awt.RenderingHints.Key,%20java.lang.Object))([RenderingHints.Key](http://docs.google.com/java/awt/RenderingHints.Key.html) hintKey, [Object](http://docs.google.com/java/lang/Object.html) hintValue)            Sets the value of a single preference for the rendering algorithms. |
| abstract  void | [**setRenderingHints**](http://docs.google.com/java/awt/Graphics2D.html#setRenderingHints(java.util.Map))([Map](http://docs.google.com/java/util/Map.html)<?,?> hints)            Replaces the values of all preferences for the rendering algorithms with the specified hints. |
| abstract  void | [**setStroke**](http://docs.google.com/java/awt/Graphics2D.html#setStroke(java.awt.Stroke))([Stroke](http://docs.google.com/java/awt/Stroke.html) s)            Sets the Stroke for the Graphics2D context. |
| abstract  void | [**setTransform**](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Overwrites the Transform in the Graphics2D context. |
| abstract  void | [**shear**](http://docs.google.com/java/awt/Graphics2D.html#shear(double,%20double))(double shx, double shy)            Concatenates the current Graphics2D Transform with a shearing transform. |
| abstract  void | [**transform**](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Composes an AffineTransform object with the Transform in this Graphics2D according to the rule last-specified-first-applied. |
| abstract  void | [**translate**](http://docs.google.com/java/awt/Graphics2D.html#translate(double,%20double))(double tx, double ty)            Concatenates the current Graphics2D Transform with a translation transform. |
| abstract  void | [**translate**](http://docs.google.com/java/awt/Graphics2D.html#translate(int,%20int))(int x, int y)            Translates the origin of the Graphics2D context to the point (*x*, *y*) in the current coordinate system. |

| **Methods inherited from class java.awt.**[**Graphics**](http://docs.google.com/java/awt/Graphics.html) |
| --- |
| [clearRect](http://docs.google.com/java/awt/Graphics.html#clearRect(int,%20int,%20int,%20int)), [clipRect](http://docs.google.com/java/awt/Graphics.html#clipRect(int,%20int,%20int,%20int)), [copyArea](http://docs.google.com/java/awt/Graphics.html#copyArea(int,%20int,%20int,%20int,%20int,%20int)), [create](http://docs.google.com/java/awt/Graphics.html#create()), [create](http://docs.google.com/java/awt/Graphics.html#create(int,%20int,%20int,%20int)), [dispose](http://docs.google.com/java/awt/Graphics.html#dispose()), [drawArc](http://docs.google.com/java/awt/Graphics.html#drawArc(int,%20int,%20int,%20int,%20int,%20int)), [drawBytes](http://docs.google.com/java/awt/Graphics.html#drawBytes(byte%5B%5D,%20int,%20int,%20int,%20int)), [drawChars](http://docs.google.com/java/awt/Graphics.html#drawChars(char%5B%5D,%20int,%20int,%20int,%20int)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20java.awt.Color,%20java.awt.image.ImageObserver)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20java.awt.image.ImageObserver)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20int,%20int,%20java.awt.Color,%20java.awt.image.ImageObserver)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20int,%20int,%20java.awt.image.ImageObserver)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20int,%20int,%20int,%20int,%20int,%20int,%20java.awt.Color,%20java.awt.image.ImageObserver)), [drawImage](http://docs.google.com/java/awt/Graphics.html#drawImage(java.awt.Image,%20int,%20int,%20int,%20int,%20int,%20int,%20int,%20int,%20java.awt.image.ImageObserver)), [drawLine](http://docs.google.com/java/awt/Graphics.html#drawLine(int,%20int,%20int,%20int)), [drawOval](http://docs.google.com/java/awt/Graphics.html#drawOval(int,%20int,%20int,%20int)), [drawPolygon](http://docs.google.com/java/awt/Graphics.html#drawPolygon(int%5B%5D,%20int%5B%5D,%20int)), [drawPolygon](http://docs.google.com/java/awt/Graphics.html#drawPolygon(java.awt.Polygon)), [drawPolyline](http://docs.google.com/java/awt/Graphics.html#drawPolyline(int%5B%5D,%20int%5B%5D,%20int)), [drawRect](http://docs.google.com/java/awt/Graphics.html#drawRect(int,%20int,%20int,%20int)), [drawRoundRect](http://docs.google.com/java/awt/Graphics.html#drawRoundRect(int,%20int,%20int,%20int,%20int,%20int)), [fillArc](http://docs.google.com/java/awt/Graphics.html#fillArc(int,%20int,%20int,%20int,%20int,%20int)), [fillOval](http://docs.google.com/java/awt/Graphics.html#fillOval(int,%20int,%20int,%20int)), [fillPolygon](http://docs.google.com/java/awt/Graphics.html#fillPolygon(int%5B%5D,%20int%5B%5D,%20int)), [fillPolygon](http://docs.google.com/java/awt/Graphics.html#fillPolygon(java.awt.Polygon)), [fillRect](http://docs.google.com/java/awt/Graphics.html#fillRect(int,%20int,%20int,%20int)), [fillRoundRect](http://docs.google.com/java/awt/Graphics.html#fillRoundRect(int,%20int,%20int,%20int,%20int,%20int)), [finalize](http://docs.google.com/java/awt/Graphics.html#finalize()), [getClip](http://docs.google.com/java/awt/Graphics.html#getClip()), [getClipBounds](http://docs.google.com/java/awt/Graphics.html#getClipBounds()), [getClipBounds](http://docs.google.com/java/awt/Graphics.html#getClipBounds(java.awt.Rectangle)), [getClipRect](http://docs.google.com/java/awt/Graphics.html#getClipRect()), [getColor](http://docs.google.com/java/awt/Graphics.html#getColor()), [getFont](http://docs.google.com/java/awt/Graphics.html#getFont()), [getFontMetrics](http://docs.google.com/java/awt/Graphics.html#getFontMetrics()), [getFontMetrics](http://docs.google.com/java/awt/Graphics.html#getFontMetrics(java.awt.Font)), [hitClip](http://docs.google.com/java/awt/Graphics.html#hitClip(int,%20int,%20int,%20int)), [setClip](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int)), [setClip](http://docs.google.com/java/awt/Graphics.html#setClip(java.awt.Shape)), [setColor](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [setFont](http://docs.google.com/java/awt/Graphics.html#setFont(java.awt.Font)), [setPaintMode](http://docs.google.com/java/awt/Graphics.html#setPaintMode()), [setXORMode](http://docs.google.com/java/awt/Graphics.html#setXORMode(java.awt.Color)), [toString](http://docs.google.com/java/awt/Graphics.html#toString()) |

| **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()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [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)) |

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

### Graphics2D

protected **Graphics2D**()

Constructs a new Graphics2D object. Since Graphics2D is an abstract class, and since it must be customized by subclasses for different output devices, Graphics2D objects cannot be created directly. Instead, Graphics2D objects must be obtained from another Graphics2D object, created by a Component, or obtained from images such as [BufferedImage](http://docs.google.com/java/awt/image/BufferedImage.html) objects.

**See Also:**[Component.getGraphics()](http://docs.google.com/java/awt/Component.html#getGraphics()), [Graphics.create()](http://docs.google.com/java/awt/Graphics.html#create())

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

### draw3DRect

public void **draw3DRect**(int x,  
 int y,  
 int width,  
 int height,  
 boolean raised)

Draws a 3-D highlighted outline of the specified rectangle. The edges of the rectangle are highlighted so that they appear to be beveled and lit from the upper left corner.

The colors used for the highlighting effect are determined based on the current color. The resulting rectangle covers an area that is width + 1 pixels wide by height + 1 pixels tall. This method uses the current Color exclusively and ignores the current Paint.

**Overrides:**[draw3DRect](http://docs.google.com/java/awt/Graphics.html#draw3DRect(int,%20int,%20int,%20int,%20boolean)) in class [Graphics](http://docs.google.com/java/awt/Graphics.html) **Parameters:**x - the x coordinate of the rectangle to be drawn.y - the y coordinate of the rectangle to be drawn.width - the width of the rectangle to be drawn.height - the height of the rectangle to be drawn.raised - a boolean that determines whether the rectangle appears to be raised above the surface or sunk into the surface.**See Also:**[Graphics.fill3DRect(int, int, int, int, boolean)](http://docs.google.com/java/awt/Graphics.html#fill3DRect(int,%20int,%20int,%20int,%20boolean))

### fill3DRect

public void **fill3DRect**(int x,  
 int y,  
 int width,  
 int height,  
 boolean raised)

Paints a 3-D highlighted rectangle filled with the current color. The edges of the rectangle are highlighted so that it appears as if the edges were beveled and lit from the upper left corner. The colors used for the highlighting effect and for filling are determined from the current Color. This method uses the current Color exclusively and ignores the current Paint.

**Overrides:**[fill3DRect](http://docs.google.com/java/awt/Graphics.html#fill3DRect(int,%20int,%20int,%20int,%20boolean)) in class [Graphics](http://docs.google.com/java/awt/Graphics.html) **Parameters:**x - the x coordinate of the rectangle to be filled.y - the y coordinate of the rectangle to be filled.width - the width of the rectangle to be filled.height - the height of the rectangle to be filled.raised - a boolean value that determines whether the rectangle appears to be raised above the surface or etched into the surface.**See Also:**[Graphics.draw3DRect(int, int, int, int, boolean)](http://docs.google.com/java/awt/Graphics.html#draw3DRect(int,%20int,%20int,%20int,%20boolean))

### draw

public abstract void **draw**([Shape](http://docs.google.com/java/awt/Shape.html) s)

Strokes the outline of a Shape using the settings of the current Graphics2D context. The rendering attributes applied include the Clip, Transform, Paint, Composite and Stroke attributes.

**Parameters:**s - the Shape to be rendered**See Also:**[setStroke(java.awt.Stroke)](http://docs.google.com/java/awt/Graphics2D.html#setStroke(java.awt.Stroke)), [setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite))

### drawImage

public abstract boolean **drawImage**([Image](http://docs.google.com/java/awt/Image.html) img,  
 [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform,  
 [ImageObserver](http://docs.google.com/java/awt/image/ImageObserver.html) obs)

Renders an image, applying a transform from image space into user space before drawing. The transformation from user space into device space is done with the current Transform in the Graphics2D. The specified transformation is applied to the image before the transform attribute in the Graphics2D context is applied. The rendering attributes applied include the Clip, Transform, and Composite attributes. Note that no rendering is done if the specified transform is noninvertible.

**Parameters:**img - the specified image to be rendered. This method does nothing if img is null.xform - the transformation from image space into user spaceobs - the [ImageObserver](http://docs.google.com/java/awt/image/ImageObserver.html) to be notified as more of the Image is converted **Returns:**true if the Image is fully loaded and completely rendered, or if it's null; false if the Image is still being loaded.**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawImage

public abstract void **drawImage**([BufferedImage](http://docs.google.com/java/awt/image/BufferedImage.html) img,  
 [BufferedImageOp](http://docs.google.com/java/awt/image/BufferedImageOp.html) op,  
 int x,  
 int y)

Renders a BufferedImage that is filtered with a [BufferedImageOp](http://docs.google.com/java/awt/image/BufferedImageOp.html). The rendering attributes applied include the Clip, Transform and Composite attributes. This is equivalent to:

img1 = op.filter(img, null);  
 drawImage(img1, new AffineTransform(1f,0f,0f,1f,x,y), null);

**Parameters:**op - the filter to be applied to the image before renderingimg - the specified BufferedImage to be rendered. This method does nothing if img is null.x - the x coordinate of the location in user space where the upper left corner of the image is renderedy - the y coordinate of the location in user space where the upper left corner of the image is rendered**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawRenderedImage

public abstract void **drawRenderedImage**([RenderedImage](http://docs.google.com/java/awt/image/RenderedImage.html) img,  
 [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform)

Renders a [RenderedImage](http://docs.google.com/java/awt/image/RenderedImage.html), applying a transform from image space into user space before drawing. The transformation from user space into device space is done with the current Transform in the Graphics2D. The specified transformation is applied to the image before the transform attribute in the Graphics2D context is applied. The rendering attributes applied include the Clip, Transform, and Composite attributes. Note that no rendering is done if the specified transform is noninvertible.

**Parameters:**img - the image to be rendered. This method does nothing if img is null.xform - the transformation from image space into user space**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawRenderableImage

public abstract void **drawRenderableImage**([RenderableImage](http://docs.google.com/java/awt/image/renderable/RenderableImage.html) img,  
 [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) xform)

Renders a [RenderableImage](http://docs.google.com/java/awt/image/renderable/RenderableImage.html), applying a transform from image space into user space before drawing. The transformation from user space into device space is done with the current Transform in the Graphics2D. The specified transformation is applied to the image before the transform attribute in the Graphics2D context is applied. The rendering attributes applied include the Clip, Transform, and Composite attributes. Note that no rendering is done if the specified transform is noninvertible.

Rendering hints set on the Graphics2D object might be used in rendering the RenderableImage. If explicit control is required over specific hints recognized by a specific RenderableImage, or if knowledge of which hints are used is required, then a RenderedImage should be obtained directly from the RenderableImage and rendered using [drawRenderedImage](http://docs.google.com/java/awt/Graphics2D.html#drawRenderedImage(java.awt.image.RenderedImage,%20java.awt.geom.AffineTransform)).

**Parameters:**img - the image to be rendered. This method does nothing if img is null.xform - the transformation from image space into user space**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int)), [drawRenderedImage(java.awt.image.RenderedImage, java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#drawRenderedImage(java.awt.image.RenderedImage,%20java.awt.geom.AffineTransform))

### drawString

public abstract void **drawString**([String](http://docs.google.com/java/lang/String.html) str,  
 int x,  
 int y)

Renders the text of the specified String, using the current text attribute state in the Graphics2D context. The baseline of the first character is at position (*x*, *y*) in the User Space. The rendering attributes applied include the Clip, Transform, Paint, Font and Composite attributes. For characters in script systems such as Hebrew and Arabic, the glyphs can be rendered from right to left, in which case the coordinate supplied is the location of the leftmost character on the baseline.

**Specified by:**[drawString](http://docs.google.com/java/awt/Graphics.html#drawString(java.lang.String,%20int,%20int)) in class [Graphics](http://docs.google.com/java/awt/Graphics.html) **Parameters:**str - the string to be renderedx - the x coordinate of the location where the String should be renderedy - the y coordinate of the location where the String should be rendered **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if str is null**Since:** JDK1.0 **See Also:**[Graphics.drawBytes(byte[], int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#drawBytes(byte%5B%5D,%20int,%20int,%20int,%20int)), [Graphics.drawChars(char[], int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#drawChars(char%5B%5D,%20int,%20int,%20int,%20int))

### drawString

public abstract void **drawString**([String](http://docs.google.com/java/lang/String.html) str,  
 float x,  
 float y)

Renders the text specified by the specified String, using the current text attribute state in the Graphics2D context. The baseline of the first character is at position (*x*, *y*) in the User Space. The rendering attributes applied include the Clip, Transform, Paint, Font and Composite attributes. For characters in script systems such as Hebrew and Arabic, the glyphs can be rendered from right to left, in which case the coordinate supplied is the location of the leftmost character on the baseline.

**Parameters:**str - the String to be renderedx - the x coordinate of the location where the String should be renderedy - the y coordinate of the location where the String should be rendered **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if str is null**See Also:**[setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [Graphics.setFont(java.awt.Font)](http://docs.google.com/java/awt/Graphics.html#setFont(java.awt.Font)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawString

public abstract void **drawString**([AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) iterator,  
 int x,  
 int y)

Renders the text of the specified iterator applying its attributes in accordance with the specification of the [TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html) class.

The baseline of the first character is at position (*x*, *y*) in User Space. For characters in script systems such as Hebrew and Arabic, the glyphs can be rendered from right to left, in which case the coordinate supplied is the location of the leftmost character on the baseline.

**Specified by:**[drawString](http://docs.google.com/java/awt/Graphics.html#drawString(java.text.AttributedCharacterIterator,%20int,%20int)) in class [Graphics](http://docs.google.com/java/awt/Graphics.html) **Parameters:**iterator - the iterator whose text is to be renderedx - the x coordinate where the iterator's text is to be renderedy - the y coordinate where the iterator's text is to be rendered **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if iterator is null**See Also:**[setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawString

public abstract void **drawString**([AttributedCharacterIterator](http://docs.google.com/java/text/AttributedCharacterIterator.html) iterator,  
 float x,  
 float y)

Renders the text of the specified iterator applying its attributes in accordance with the specification of the [TextAttribute](http://docs.google.com/java/awt/font/TextAttribute.html) class.

The baseline of the first character is at position (*x*, *y*) in User Space. For characters in script systems such as Hebrew and Arabic, the glyphs can be rendered from right to left, in which case the coordinate supplied is the location of the leftmost character on the baseline.

**Parameters:**iterator - the iterator whose text is to be renderedx - the x coordinate where the iterator's text is to be renderedy - the y coordinate where the iterator's text is to be rendered **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if iterator is null**See Also:**[setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### drawGlyphVector

public abstract void **drawGlyphVector**([GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) g,  
 float x,  
 float y)

Renders the text of the specified [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html) using the Graphics2D context's rendering attributes. The rendering attributes applied include the Clip, Transform, Paint, and Composite attributes. The GlyphVector specifies individual glyphs from a [Font](http://docs.google.com/java/awt/Font.html). The GlyphVector can also contain the glyph positions. This is the fastest way to render a set of characters to the screen.

**Parameters:**g - the GlyphVector to be renderedx - the x position in User Space where the glyphs should be renderedy - the y position in User Space where the glyphs should be rendered **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if g is null.**See Also:**[Font.createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String)), [GlyphVector](http://docs.google.com/java/awt/font/GlyphVector.html), [setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### fill

public abstract void **fill**([Shape](http://docs.google.com/java/awt/Shape.html) s)

Fills the interior of a Shape using the settings of the Graphics2D context. The rendering attributes applied include the Clip, Transform, Paint, and Composite.

**Parameters:**s - the Shape to be filled**See Also:**[setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### hit

public abstract boolean **hit**([Rectangle](http://docs.google.com/java/awt/Rectangle.html) rect,  
 [Shape](http://docs.google.com/java/awt/Shape.html) s,  
 boolean onStroke)

Checks whether or not the specified Shape intersects the specified [Rectangle](http://docs.google.com/java/awt/Rectangle.html), which is in device space. If onStroke is false, this method checks whether or not the interior of the specified Shape intersects the specified Rectangle. If onStroke is true, this method checks whether or not the Stroke of the specified Shape outline intersects the specified Rectangle. The rendering attributes taken into account include the Clip, Transform, and Stroke attributes.

**Parameters:**rect - the area in device space to check for a hits - the Shape to check for a hitonStroke - flag used to choose between testing the stroked or the filled shape. If the flag is true, the Stroke oultine is tested. If the flag is false, the filled Shape is tested. **Returns:**true if there is a hit; false otherwise.**See Also:**[setStroke(java.awt.Stroke)](http://docs.google.com/java/awt/Graphics2D.html#setStroke(java.awt.Stroke)), [fill(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#fill(java.awt.Shape)), [draw(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#draw(java.awt.Shape)), [transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [clip(java.awt.Shape)](http://docs.google.com/java/awt/Graphics2D.html#clip(java.awt.Shape)), [Graphics.setClip(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#setClip(int,%20int,%20int,%20int))

### getDeviceConfiguration

public abstract [GraphicsConfiguration](http://docs.google.com/java/awt/GraphicsConfiguration.html) **getDeviceConfiguration**()

Returns the device configuration associated with this Graphics2D.

**Returns:**the device configuration of this Graphics2D.

### setComposite

public abstract void **setComposite**([Composite](http://docs.google.com/java/awt/Composite.html) comp)

Sets the Composite for the Graphics2D context. The Composite is used in all drawing methods such as drawImage, drawString, draw, and fill. It specifies how new pixels are to be combined with the existing pixels on the graphics device during the rendering process.

If this Graphics2D context is drawing to a Component on the display screen and the Composite is a custom object rather than an instance of the AlphaComposite class, and if there is a security manager, its checkPermission method is called with an AWTPermission("readDisplayPixels") permission.

**Parameters:**comp - the Composite object to be used for rendering **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a custom Composite object is being used to render to the screen and a security manager is set and its checkPermission method does not allow the operation.**See Also:**[Graphics.setXORMode(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setXORMode(java.awt.Color)), [Graphics.setPaintMode()](http://docs.google.com/java/awt/Graphics.html#setPaintMode()), [getComposite()](http://docs.google.com/java/awt/Graphics2D.html#getComposite()), [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html), [SecurityManager.checkPermission(java.security.Permission)](http://docs.google.com/java/lang/SecurityManager.html#checkPermission(java.security.Permission)), [AWTPermission](http://docs.google.com/java/awt/AWTPermission.html)

### setPaint

public abstract void **setPaint**([Paint](http://docs.google.com/java/awt/Paint.html) paint)

Sets the Paint attribute for the Graphics2D context. Calling this method with a null Paint object does not have any effect on the current Paint attribute of this Graphics2D.

**Parameters:**paint - the Paint object to be used to generate color during the rendering process, or null**See Also:**[Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color)), [getPaint()](http://docs.google.com/java/awt/Graphics2D.html#getPaint()), [GradientPaint](http://docs.google.com/java/awt/GradientPaint.html), [TexturePaint](http://docs.google.com/java/awt/TexturePaint.html)

### setStroke

public abstract void **setStroke**([Stroke](http://docs.google.com/java/awt/Stroke.html) s)

Sets the Stroke for the Graphics2D context.

**Parameters:**s - the Stroke object to be used to stroke a Shape during the rendering process**See Also:**[BasicStroke](http://docs.google.com/java/awt/BasicStroke.html), [getStroke()](http://docs.google.com/java/awt/Graphics2D.html#getStroke())

### setRenderingHint

public abstract void **setRenderingHint**([RenderingHints.Key](http://docs.google.com/java/awt/RenderingHints.Key.html) hintKey,  
 [Object](http://docs.google.com/java/lang/Object.html) hintValue)

Sets the value of a single preference for the rendering algorithms. Hint categories include controls for rendering quality and overall time/quality trade-off in the rendering process. Refer to the RenderingHints class for definitions of some common keys and values.

**Parameters:**hintKey - the key of the hint to be set.hintValue - the value indicating preferences for the specified hint category.**See Also:**[getRenderingHint(RenderingHints.Key)](http://docs.google.com/java/awt/Graphics2D.html#getRenderingHint(java.awt.RenderingHints.Key)), [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html)

### getRenderingHint

public abstract [Object](http://docs.google.com/java/lang/Object.html) **getRenderingHint**([RenderingHints.Key](http://docs.google.com/java/awt/RenderingHints.Key.html) hintKey)

Returns the value of a single preference for the rendering algorithms. Hint categories include controls for rendering quality and overall time/quality trade-off in the rendering process. Refer to the RenderingHints class for definitions of some common keys and values.

**Parameters:**hintKey - the key corresponding to the hint to get. **Returns:**an object representing the value for the specified hint key. Some of the keys and their associated values are defined in the RenderingHints class.**See Also:**[RenderingHints](http://docs.google.com/java/awt/RenderingHints.html), [setRenderingHint(RenderingHints.Key, Object)](http://docs.google.com/java/awt/Graphics2D.html#setRenderingHint(java.awt.RenderingHints.Key,%20java.lang.Object))

### setRenderingHints

public abstract void **setRenderingHints**([Map](http://docs.google.com/java/util/Map.html)<?,?> hints)

Replaces the values of all preferences for the rendering algorithms with the specified hints. The existing values for all rendering hints are discarded and the new set of known hints and values are initialized from the specified [Map](http://docs.google.com/java/util/Map.html) object. Hint categories include controls for rendering quality and overall time/quality trade-off in the rendering process. Refer to the RenderingHints class for definitions of some common keys and values.

**Parameters:**hints - the rendering hints to be set**See Also:**[getRenderingHints()](http://docs.google.com/java/awt/Graphics2D.html#getRenderingHints()), [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html)

### addRenderingHints

public abstract void **addRenderingHints**([Map](http://docs.google.com/java/util/Map.html)<?,?> hints)

Sets the values of an arbitrary number of preferences for the rendering algorithms. Only values for the rendering hints that are present in the specified Map object are modified. All other preferences not present in the specified object are left unmodified. Hint categories include controls for rendering quality and overall time/quality trade-off in the rendering process. Refer to the RenderingHints class for definitions of some common keys and values.

**Parameters:**hints - the rendering hints to be set**See Also:**[RenderingHints](http://docs.google.com/java/awt/RenderingHints.html)

### getRenderingHints

public abstract [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html) **getRenderingHints**()

Gets the preferences for the rendering algorithms. Hint categories include controls for rendering quality and overall time/quality trade-off in the rendering process. Returns all of the hint key/value pairs that were ever specified in one operation. Refer to the RenderingHints class for definitions of some common keys and values.

**Returns:**a reference to an instance of RenderingHints that contains the current preferences.**See Also:**[RenderingHints](http://docs.google.com/java/awt/RenderingHints.html), [setRenderingHints(Map)](http://docs.google.com/java/awt/Graphics2D.html#setRenderingHints(java.util.Map))

### translate

public abstract void **translate**(int x,  
 int y)

Translates the origin of the Graphics2D context to the point (*x*, *y*) in the current coordinate system. Modifies the Graphics2D context so that its new origin corresponds to the point (*x*, *y*) in the Graphics2D context's former coordinate system. All coordinates used in subsequent rendering operations on this graphics context are relative to this new origin.

**Specified by:**[translate](http://docs.google.com/java/awt/Graphics.html#translate(int,%20int)) in class [Graphics](http://docs.google.com/java/awt/Graphics.html) **Parameters:**x - the specified x coordinatey - the specified y coordinate**Since:** JDK1.0

### translate

public abstract void **translate**(double tx,  
 double ty)

Concatenates the current Graphics2D Transform with a translation transform. Subsequent rendering is translated by the specified distance relative to the previous position. This is equivalent to calling transform(T), where T is an AffineTransform represented by the following matrix:

[ 1 0 tx ]  
 [ 0 1 ty ]  
 [ 0 0 1 ]

**Parameters:**tx - the distance to translate along the x-axisty - the distance to translate along the y-axis

### rotate

public abstract void **rotate**(double theta)

Concatenates the current Graphics2D Transform with a rotation transform. Subsequent rendering is rotated by the specified radians relative to the previous origin. This is equivalent to calling transform(R), where R is an AffineTransform represented by the following matrix:

[ cos(theta) -sin(theta) 0 ]  
 [ sin(theta) cos(theta) 0 ]  
 [ 0 0 1 ]

Rotating with a positive angle theta rotates points on the positive x axis toward the positive y axis.

**Parameters:**theta - the angle of rotation in radians

### rotate

public abstract void **rotate**(double theta,  
 double x,  
 double y)

Concatenates the current Graphics2D Transform with a translated rotation transform. Subsequent rendering is transformed by a transform which is constructed by translating to the specified location, rotating by the specified radians, and translating back by the same amount as the original translation. This is equivalent to the following sequence of calls:

translate(x, y);  
 rotate(theta);  
 translate(-x, -y);

Rotating with a positive angle theta rotates points on the positive x axis toward the positive y axis.

**Parameters:**theta - the angle of rotation in radiansx - the x coordinate of the origin of the rotationy - the y coordinate of the origin of the rotation

### scale

public abstract void **scale**(double sx,  
 double sy)

Concatenates the current Graphics2D Transform with a scaling transformation Subsequent rendering is resized according to the specified scaling factors relative to the previous scaling. This is equivalent to calling transform(S), where S is an AffineTransform represented by the following matrix:

[ sx 0 0 ]  
 [ 0 sy 0 ]  
 [ 0 0 1 ]

**Parameters:**sx - the amount by which X coordinates in subsequent rendering operations are multiplied relative to previous rendering operations.sy - the amount by which Y coordinates in subsequent rendering operations are multiplied relative to previous rendering operations.

### shear

public abstract void **shear**(double shx,  
 double shy)

Concatenates the current Graphics2D Transform with a shearing transform. Subsequent renderings are sheared by the specified multiplier relative to the previous position. This is equivalent to calling transform(SH), where SH is an AffineTransform represented by the following matrix:

[ 1 shx 0 ]  
 [ shy 1 0 ]  
 [ 0 0 1 ]

**Parameters:**shx - the multiplier by which coordinates are shifted in the positive X axis direction as a function of their Y coordinateshy - the multiplier by which coordinates are shifted in the positive Y axis direction as a function of their X coordinate

### transform

public abstract void **transform**([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)

Composes an AffineTransform object with the Transform in this Graphics2D according to the rule last-specified-first-applied. If the current Transform is Cx, the result of composition with Tx is a new Transform Cx'. Cx' becomes the current Transform for this Graphics2D. Transforming a point p by the updated Transform Cx' is equivalent to first transforming p by Tx and then transforming the result by the original Transform Cx. In other words, Cx'(p) = Cx(Tx(p)). A copy of the Tx is made, if necessary, so further modifications to Tx do not affect rendering.

**Parameters:**Tx - the AffineTransform object to be composed with the current Transform**See Also:**[setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform)), [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html)

### setTransform

public abstract void **setTransform**([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)

Overwrites the Transform in the Graphics2D context. WARNING: This method should **never** be used to apply a new coordinate transform on top of an existing transform because the Graphics2D might already have a transform that is needed for other purposes, such as rendering Swing components or applying a scaling transformation to adjust for the resolution of a printer.

To add a coordinate transform, use the transform, rotate, scale, or shear methods. The setTransform method is intended only for restoring the original Graphics2D transform after rendering, as shown in this example:

// Get the current transform  
 AffineTransform saveAT = g2.getTransform();  
 // Perform transformation  
 g2d.transform(...);  
 // Render  
 g2d.draw(...);  
 // Restore original transform  
 g2d.setTransform(saveAT);

**Parameters:**Tx - the AffineTransform that was retrieved from the getTransform method**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [getTransform()](http://docs.google.com/java/awt/Graphics2D.html#getTransform()), [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html)

### getTransform

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

Returns a copy of the current Transform in the Graphics2D context.

**Returns:**the current AffineTransform in the Graphics2D context.**See Also:**[transform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#transform(java.awt.geom.AffineTransform)), [setTransform(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/Graphics2D.html#setTransform(java.awt.geom.AffineTransform))

### getPaint

public abstract [Paint](http://docs.google.com/java/awt/Paint.html) **getPaint**()

Returns the current Paint of the Graphics2D context.

**Returns:**the current Graphics2D Paint, which defines a color or pattern.**See Also:**[setPaint(java.awt.Paint)](http://docs.google.com/java/awt/Graphics2D.html#setPaint(java.awt.Paint)), [Graphics.setColor(java.awt.Color)](http://docs.google.com/java/awt/Graphics.html#setColor(java.awt.Color))

### getComposite

public abstract [Composite](http://docs.google.com/java/awt/Composite.html) **getComposite**()

Returns the current Composite in the Graphics2D context.

**Returns:**the current Graphics2D Composite, which defines a compositing style.**See Also:**[setComposite(java.awt.Composite)](http://docs.google.com/java/awt/Graphics2D.html#setComposite(java.awt.Composite))

### setBackground

public abstract void **setBackground**([Color](http://docs.google.com/java/awt/Color.html) color)

Sets the background color for the Graphics2D context. The background color is used for clearing a region. When a Graphics2D is constructed for a Component, the background color is inherited from the Component. Setting the background color in the Graphics2D context only affects the subsequent clearRect calls and not the background color of the Component. To change the background of the Component, use appropriate methods of the Component.

**Parameters:**color - the background color that isused in subsequent calls to clearRect**See Also:**[getBackground()](http://docs.google.com/java/awt/Graphics2D.html#getBackground()), [Graphics.clearRect(int, int, int, int)](http://docs.google.com/java/awt/Graphics.html#clearRect(int,%20int,%20int,%20int))

### getBackground

public abstract [Color](http://docs.google.com/java/awt/Color.html) **getBackground**()

Returns the background color used for clearing a region.

**Returns:**the current Graphics2D Color, which defines the background color.**See Also:**[setBackground(java.awt.Color)](http://docs.google.com/java/awt/Graphics2D.html#setBackground(java.awt.Color))

### getStroke

public abstract [Stroke](http://docs.google.com/java/awt/Stroke.html) **getStroke**()

Returns the current Stroke in the Graphics2D context.

**Returns:**the current Graphics2D Stroke, which defines the line style.**See Also:**[setStroke(java.awt.Stroke)](http://docs.google.com/java/awt/Graphics2D.html#setStroke(java.awt.Stroke))

### clip

public abstract void **clip**([Shape](http://docs.google.com/java/awt/Shape.html) s)

Intersects the current Clip with the interior of the specified Shape and sets the Clip to the resulting intersection. The specified Shape is transformed with the current Graphics2D Transform before being intersected with the current Clip. This method is used to make the current Clip smaller. To make the Clip larger, use setClip. The *user clip* modified by this method is independent of the clipping associated with device bounds and visibility. If no clip has previously been set, or if the clip has been cleared using [setClip](http://docs.google.com/java/awt/Graphics.html#setClip(java.awt.Shape)) with a null argument, the specified Shape becomes the new user clip.

**Parameters:**s - the Shape to be intersected with the current Clip. If s is null, this method clears the current Clip.

### getFontRenderContext

public abstract [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) **getFontRenderContext**()

Get the rendering context of the Font within this Graphics2D context. The [FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html) encapsulates application hints such as anti-aliasing and fractional metrics, as well as target device specific information such as dots-per-inch. This information should be provided by the application when using objects that perform typographical formatting, such as Font and TextLayout. This information should also be provided by applications that perform their own layout and need accurate measurements of various characteristics of glyphs such as advance and line height when various rendering hints have been applied to the text rendering.

**Returns:**a reference to an instance of FontRenderContext.**Since:** 1.2 **See Also:**[FontRenderContext](http://docs.google.com/java/awt/font/FontRenderContext.html), [Font.createGlyphVector(java.awt.font.FontRenderContext, java.lang.String)](http://docs.google.com/java/awt/Font.html#createGlyphVector(java.awt.font.FontRenderContext,%20java.lang.String)), [TextLayout](http://docs.google.com/java/awt/font/TextLayout.html)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Graphics2D.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/Graphics.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/GraphicsConfigTemplate.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/Graphics2D.html)    [**NO FRAMES**](http://docs.google.com/Graphics2D.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#1t3h5sf) | [METHOD](#4d34og8) | DETAIL: FIELD | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

[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).