java.awt

Class Graphics

java.lang.Object
  └ java.awt.Graphics

Direct Known Subclasses:

DebugGraphics, Graphics2D

```
public abstract class Graphics
extends Object
```

The `Graphics` class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

A `Graphics` object encapsulates state information needed for the basic rendering operations that Java supports. This state information includes the following properties:

- The `Component` object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function (XOR or Paint).
- The current XOR alternation color (see `setXORMode(java.awt.Color)`).

Coordinates are infinitely thin and lie between the pixels of the output device. Operations that draw the outline of a figure operate by traversing an infinitely thin path between pixels with a pixel-sized pen that hangs down and to the right of the anchor point on the path. Operations that fill a figure operate by filling the interior of that infinitely thin path. Operations that render horizontal text render the ascending portion of character glyphs entirely above the baseline coordinate.

The graphics pen hangs down and to the right from the path it traverses. This has the following implications:

- If you draw a figure that covers a given rectangle, that figure occupies one extra row of pixels on the right and bottom edges as compared to filling a figure that is bounded by that same rectangle.
- If you draw a horizontal line along the same *y* coordinate as the baseline of a line of text, that line is drawn entirely below the text, except for any descenders.

All coordinates that appear as arguments to the methods of this `Graphics` object are considered relative to the translation origin of this `Graphics` object prior to the invocation of the method.

All rendering operations modify only pixels which lie within the area bounded by the current clip, which is specified by a `Shape` in user space and is controlled by the program using the `Graphics` object. This *user clip* is transformed into device space 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 cannot be modified by the rendering system to reflect the resulting composite clip. The user clip can only be changed through the `setClip` or `clipRect` methods. All drawing or writing is done in the current color, using the current paint mode, and in the current font.

Since:

JDK1.0

See Also:

`Component`, `clipRect(int, int, int, int)`, `setColor(java.awt.Color)`, `setPaintMode()`, `setXORMode(java.awt.Color)`, `setFont(java.awt.Font)`

Constructor Summary

protected	`Graphics()` Constructs a new `Graphics` object.

Method Summary

abstract void	`clearRect(int x, int y, int width, int height)` Clears the specified rectangle by filling it with the background color of the current drawing surface.
abstract void	`clipRect(int x, int y, int width, int height)` Intersects the current clip with the specified rectangle.
abstract void	`copyArea(int x, int y, int width, int height, int dx, int dy)` Copies an area of the component by a distance specified by `dx` and `dy`.
abstract Graphics	`create()` Creates a new `Graphics` object that is a copy of this `Graphics` object.
Graphics	`create(int x, int y, int width, int height)` Creates a new `Graphics` object based on this `Graphics` object, but with a new translation and clip area.
abstract void	`dispose()` Disposes of this graphics context and releases any system resources that it is using.
void	`draw3DRect(int x, int y, int width, int height, boolean raised)` Draws a 3-D highlighted outline of the specified rectangle.
abstract void	`drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` Draws the outline of a circular or elliptical arc covering the specified

	rectangle.
void	drawBytes(byte[] data, int offset, int length, int x, int y) Draws the text given by the specified byte array, using this graphics context's current font and color.
void	drawChars(char[] data, int offset, int length, int x, int y) Draws the text given by the specified character array, using this graphics context's current font and color.
abstract boolean	drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer) Draws as much of the specified image as is currently available.
abstract boolean	drawImage(Image img, int x, int y, ImageObserver observer) Draws as much of the specified image as is currently available.
abstract boolean	drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer) Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
abstract boolean	drawImage(Image img, int x, int y, int width, int height, ImageObserver observer) Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
abstract boolean	drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer) Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
abstract boolean	drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer) Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
abstract void	drawLine(int x1, int y1, int x2, int y2) Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
abstract void	drawOval(int x, int y, int width, int height) Draws the outline of an oval.
abstract void	drawPolygon(int[] xPoints, int[] yPoints, int nPoints) Draws a closed polygon defined by arrays of *x* and *y* coordinates.
void	drawPolygon(Polygon p) Draws the outline of a polygon defined by the specified Polygon object.
abstract void	drawPolyline(int[] xPoints, int[] yPoints, int nPoints) Draws a sequence of connected lines defined by arrays of *x* and *y* coordinates.
void	drawRect(int x, int y, int width, int height) Draws the outline of the specified rectangle.
abstract void	drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Draws an outlined round-cornered rectangle using this graphics context's

	current color.
abstract void	drawString(AttributedCharacterIterator iterator, int x, int y) Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
abstract void	drawString(String str, int x, int y) Draws the text given by the specified string, using this graphics context's current font and color.
void	fill3DRect(int x, int y, int width, int height, boolean raised) Paints a 3-D highlighted rectangle filled with the current color.
abstract void	fillArc(int x, int y, int width, int height, int startAngle, int arcAngle) Fills a circular or elliptical arc covering the specified rectangle.
abstract void	fillOval(int x, int y, int width, int height) Fills an oval bounded by the specified rectangle with the current color.
abstract void	fillPolygon(int[] xPoints, int[] yPoints, int nPoints) Fills a closed polygon defined by arrays of *x* and *y* coordinates.
void	fillPolygon(Polygon p) Fills the polygon defined by the specified Polygon object with the graphics context's current color.
abstract void	fillRect(int x, int y, int width, int height) Fills the specified rectangle.
abstract void	fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight) Fills the specified rounded corner rectangle with the current color.
void	finalize() Disposes of this graphics context once it is no longer referenced.
abstract Shape	getClip() Gets the current clipping area.
abstract Rectangle	getClipBounds() Returns the bounding rectangle of the current clipping area.
Rectangle	getClipBounds(Rectangle r) Returns the bounding rectangle of the current clipping area.
Rectangle	getClipRect() **Deprecated.** *As of JDK version 1.1, replaced by* getClipBounds().
abstract Color	getColor() Gets this graphics context's current color.
abstract Font	getFont() Gets the current font.
FontMetrics	getFontMetrics() Gets the font metrics of the current font.
abstract FontMetrics	getFontMetrics(Font f) Gets the font metrics for the specified font.
boolean	hitClip(int x, int y, int width, int height) Returns true if the specified rectangular area might intersect the current clipping area.

abstract void	setClip(int x, int y, int width, int height) Sets the current clip to the rectangle specified by the given coordinates.
abstract void	setClip(Shape clip) Sets the current clipping area to an arbitrary clip shape.
abstract void	setColor(Color c) Sets this graphics context's current color to the specified color.
abstract void	setFont(Font font) Sets this graphics context's font to the specified font.
abstract void	setPaintMode() Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
abstract void	setXORMode(Color c1) Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.
String	toString() Returns a String object representing this Graphics object's value.
abstract void	translate(int x, int y) Translates the origin of the graphics context to the point (*x*, *y*) in the current coordinate system.

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Graphics

```
protected Graphics()
```

Constructs a new Graphics object. This constructor is the default contructor for a graphics context.

Since Graphics is an abstract class, applications cannot call this constructor directly. Graphics contexts are obtained from other graphics contexts or are created by calling getGraphics on a component.

See Also:

create(), Component.getGraphics()

Method Detail

create

```
public abstract Graphics create()
```

Creates a new `Graphics` object that is a copy of this `Graphics` object.

Returns:

a new graphics context that is a copy of this graphics context.

---

create

```
public Graphics create(int  x,
                       int  y,
                       int  width,
                       int  height)
```

Creates a new `Graphics` object based on this `Graphics` object, but with a new translation and clip area. The new `Graphics` object has its origin translated to the specified point (*x*, *y*). Its clip area is determined by the intersection of the original clip area with the specified rectangle. The arguments are all interpreted in the coordinate system of the original `Graphics` object. The new graphics context is identical to the original, except in two respects:

- The new graphics context is translated by (*x*, *y*). That is to say, the point (`0`, `0`) in the new graphics context is the same as (*x*, *y*) in the original graphics context.
- The new graphics context has an additional clipping rectangle, in addition to whatever (translated) clipping rectangle it inherited from the original graphics context. The origin of the new clipping rectangle is at (`0`, `0`), and its size is specified by the `width` and `height` arguments.

Parameters:

`x` - the *x* coordinate.
`y` - the *y* coordinate.
`width` - the width of the clipping rectangle.
`height` - the height of the clipping rectangle.

Returns:

a new graphics context.

See Also:

`translate(int, int)`, `clipRect(int, int, int, int)`

---

translate

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

Translates the origin of the graphics context to the point (*x*, *y*) in the current coordinate system. Modifies this graphics context so that its new origin corresponds to the point (*x*, *y*) in this graphics context's original coordinate system. All coordinates used in

subsequent rendering operations on this graphics context will be relative to this new origin.

Parameters:

x - the *x* coordinate.

y - the *y* coordinate.

---

getColor

```
public abstract Color getColor()
```

Gets this graphics context's current color.

Returns:

this graphics context's current color.

See Also:

Color, setColor(Color)

---

setColor

```
public abstract void setColor(Color c)
```

Sets this graphics context's current color to the specified color. All subsequent graphics operations using this graphics context use this specified color.

Parameters:

c - the new rendering color.

See Also:

Color, getColor()

---

setPaintMode

```
public abstract void setPaintMode()
```

Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color. This sets the logical pixel operation function to the paint or overwrite mode. All subsequent rendering operations will overwrite the destination with the current color.

---

setXORMode

```
public abstract void setXORMode(Color c1)
```

Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color. This specifies that logical pixel operations are

performed in the XOR mode, which alternates pixels between the current color and a specified XOR color.

When drawing operations are performed, pixels which are the current color are changed to the specified color, and vice versa.

Pixels that are of colors other than those two colors are changed in an unpredictable but reversible manner; if the same figure is drawn twice, then all pixels are restored to their original values.

Parameters:

`c1` - the XOR alternation color

---

getFont

```
public abstract Font getFont()
```

Gets the current font.

Returns:

this graphics context's current font.

See Also:

Font, setFont(Font)

---

setFont

```
public abstract void setFont(Font font)
```

Sets this graphics context's font to the specified font. All subsequent text operations using this graphics context use this font. A null argument is silently ignored.

Parameters:

`font` - the font.

See Also:

getFont(), drawString(java.lang.String, int, int), drawBytes(byte[], int, int, int, int), drawChars(char[], int, int, int, int)

---

getFontMetrics

```
public FontMetrics getFontMetrics()
```

Gets the font metrics of the current font.

Returns:

the font metrics of this graphics context's current font.

See Also:

getFont(), FontMetrics, getFontMetrics(Font)

getFontMetrics

```
public abstract FontMetrics getFontMetrics(Font  f)
```

Gets the font metrics for the specified font.

Parameters:

`f` - the specified font

Returns:

the font metrics for the specified font.

See Also:

getFont(), FontMetrics, getFontMetrics()

---

getClipBounds

```
public abstract Rectangle getClipBounds()
```

Returns the bounding rectangle of the current clipping area. This method refers to the user clip, which is independent of the clipping associated with device bounds and window visibility. If no clip has previously been set, or if the clip has been cleared using `setClip(null)`, this method returns `null`. The coordinates in the rectangle are relative to the coordinate system origin of this graphics context.

Returns:

the bounding rectangle of the current clipping area, or `null` if no clip is set.

Since:

JDK1.1

See Also:

getClip(), clipRect(int, int, int, int), setClip(int, int, int, int), setClip(Shape)

---

clipRect

```
public abstract void clipRect(int  x,
                              int  y,
                              int  width,
                              int  height)
```

Intersects the current clip with the specified rectangle. The resulting clipping area is the intersection of the current clipping area and the specified rectangle. If there is no current clipping area, either because the clip has never been set, or the clip has been cleared using `setClip(null)`, the specified rectangle becomes the new clip. This method sets the user clip, which is independent of the clipping associated with device bounds and window visibility. This method can only be used to make the current clip smaller. To set the current clip larger, use any of the setClip methods. Rendering operations have no effect outside of the clipping area.

Parameters:

`x` - the x coordinate of the rectangle to intersect the clip with

`y` - the y coordinate of the rectangle to intersect the clip with

`width` - the width of the rectangle to intersect the clip with

`height` - the height of the rectangle to intersect the clip with

See Also:

`setClip(int, int, int, int)`, `setClip(Shape)`

---

setClip

```
public abstract void setClip(int  x,
                             int  y,
                             int  width,
                             int  height)
```

Sets the current clip to the rectangle specified by the given coordinates. This method sets the user clip, which is independent of the clipping associated with device bounds and window visibility. Rendering operations have no effect outside of the clipping area.

Parameters:

`x` - the *x* coordinate of the new clip rectangle.

`y` - the *y* coordinate of the new clip rectangle.

`width` - the width of the new clip rectangle.

`height` - the height of the new clip rectangle.

Since:

JDK1.1

See Also:

`clipRect(int, int, int, int)`, `setClip(Shape)`, `getClip()`

---

getClip

```
public abstract Shape getClip()
```

Gets the current clipping area. This method returns the user clip, which is independent of the clipping associated with device bounds and window visibility. If no clip has previously been set, or if the clip has been cleared using `setClip(null)`, this method returns `null`.

Returns:

a `Shape` object representing the current clipping area, or `null` if no clip is set.

Since:

JDK1.1

See Also:

`getClipBounds()`, `clipRect(int, int, int, int)`, `setClip(int, int, int, int)`, `setClip(Shape)`

---

setClip

```
public abstract void setClip(Shape  clip)
```

Sets the current clipping area to an arbitrary clip shape. Not all objects that implement the `Shape` interface can be used to set the clip. The only `Shape` objects that are guaranteed to be supported are `Shape` objects that are obtained via the `getClip` method and via `Rectangle` objects. This method sets the user clip, which is independent of the clipping associated with device bounds and window visibility.

Parameters:

`clip` - the `Shape` to use to set the clip

Since:

JDK1.1

See Also:

`getClip()`, `clipRect(int, int, int, int)`, `setClip(int, int, int, int)`

---

copyArea

```
public abstract void copyArea(int  x,
                              int  y,
                              int  width,
                              int  height,
                              int  dx,
                              int  dy)
```

Copies an area of the component by a distance specified by `dx` and `dy`. From the point specified by `x` and `y`, this method copies downwards and to the right. To copy an area of the component to the left or upwards, specify a negative value for `dx` or `dy`. If a portion of the source rectangle lies outside the bounds of the component, or is obscured by another window or component, `copyArea` will be unable to copy the associated pixels. The area that is omitted can be refreshed by calling the component's `paint` method.

Parameters:

`x` - the *x* coordinate of the source rectangle.
`y` - the *y* coordinate of the source rectangle.
`width` - the width of the source rectangle.
`height` - the height of the source rectangle.
`dx` - the horizontal distance to copy the pixels.
`dy` - the vertical distance to copy the pixels.

---

drawLine

```
public abstract void drawLine(int  x1,
                              int  y1,
                              int  x2,
                              int  y2)
```

Draws a line, using the current color, between the points `(x1, y1)` and `(x2, y2)` in this graphics context's coordinate system.

Parameters:

`x1` - the first point's *x* coordinate.
`y1` - the first point's *y* coordinate.
`x2` - the second point's *x* coordinate.
`y2` - the second point's *y* coordinate.

---

fillRect

```
public abstract void fillRect(int  x,
                              int  y,
                              int  width,
                              int  height)
```

Fills the specified rectangle. The left and right edges of the rectangle are at `x` and `x + width - 1`. The top and bottom edges are at `y` and `y + height - 1`. The resulting rectangle covers an area `width` pixels wide by `height` pixels tall. The rectangle is filled using the graphics context's current color.

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.

See Also:

`clearRect(int, int, int, int)`, `drawRect(int, int, int, int)`

---

drawRect

```
public void drawRect(int  x,
                     int  y,
                     int  width,
                     int  height)
```

Draws the outline of the specified rectangle. The left and right edges of the rectangle are at `x` and `x + width`. The top and bottom edges are at `y` and `y + height`. The rectangle is drawn using the graphics context's current color.

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.

See Also:

`fillRect(int, int, int, int)`, `clearRect(int, int, int, int)`

---

clearRect

```
public abstract void clearRect(int  x,
                               int  y,
                               int  width,
                               int  height)
```

Clears the specified rectangle by filling it with the background color of the current drawing surface. This operation does not use the current paint mode.

Beginning with Java 1.1, the background color of offscreen images may be system dependent. Applications should use `setColor` followed by `fillRect` to ensure that an offscreen image is cleared to a specific color.

Parameters:

`x` - the *x* coordinate of the rectangle to clear.
`y` - the *y* coordinate of the rectangle to clear.
`width` - the width of the rectangle to clear.
`height` - the height of the rectangle to clear.

See Also:

`fillRect(int, int, int, int)`, `drawRect(int, int, int, int)`, `setColor(java.awt.Color)`, `setPaintMode()`, `setXORMode(java.awt.Color)`

---

drawRoundRect

```
public abstract void drawRoundRect(int  x,
                                   int  y,
                                   int  width,
                                   int  height,
                                   int  arcWidth,
                                   int  arcHeight)
```

Draws an outlined round-cornered rectangle using this graphics context's current color. The left and right edges of the rectangle are at `x` and `x + width`, respectively. The top and bottom edges of the rectangle are at `y` and `y + height`.

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.
`arcWidth` - the horizontal diameter of the arc at the four corners.
`arcHeight` - the vertical diameter of the arc at the four corners.

See Also:

`fillRoundRect(int, int, int, int, int, int)`

---

fillRoundRect

```
public abstract void fillRoundRect(int  x,
                                   int  y,
                                   int  width,
                                   int  height,
```

```
                      int  arcWidth,
                      int  arcHeight)
```

Fills the specified rounded corner rectangle with the current color. The left and right edges of the rectangle are at `x` and `x + width - 1`, respectively. The top and bottom edges of the rectangle are at `y` and `y + height - 1`.

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.

`arcWidth` - the horizontal diameter of the arc at the four corners.

`arcHeight` - the vertical diameter of the arc at the four corners.

See Also:

drawRoundRect(int, int, int, int, int, int)

---

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.

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:

fill3DRect(int, int, int, int, boolean)

---

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 will be 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 will be determined from the current color.

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:

draw3DRect(int, int, int, int, boolean)

---

drawOval

```
public abstract void drawOval(int  x,
                              int  y,
                              int  width,
                              int  height)
```

Draws the outline of an oval. The result is a circle or ellipse that fits within the rectangle specified by the `x`, `y`, `width`, and `height` arguments.

The oval covers an area that is `width + 1` pixels wide and `height + 1` pixels tall.

Parameters:

`x` - the *x* coordinate of the upper left corner of the oval to be drawn.
`y` - the *y* coordinate of the upper left corner of the oval to be drawn.
`width` - the width of the oval to be drawn.
`height` - the height of the oval to be drawn.

See Also:

fillOval(int, int, int, int)

---

fillOval

```
public abstract void fillOval(int  x,
                              int  y,
                              int  width,
                              int  height)
```

Fills an oval bounded by the specified rectangle with the current color.

Parameters:

`x` - the *x* coordinate of the upper left corner of the oval to be filled.
`y` - the *y* coordinate of the upper left corner of the oval to be filled.
`width` - the width of the oval to be filled.

`height` - the height of the oval to be filled.

See Also:

drawOval(int, int, int, int)

---

drawArc

```
public abstract void drawArc(int  x,
                             int  y,
                             int  width,
                             int  height,
                             int  startAngle,
                             int  arcAngle)
```

Draws the outline of a circular or elliptical arc covering the specified rectangle.

The resulting arc begins at `startAngle` and extends for `arcAngle` degrees, using the current color. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is (*x*, *y*) and whose size is specified by the `width` and `height` arguments.

The resulting arc covers an area `width + 1` pixels wide by `height + 1` pixels tall.

The angles are specified relative to the non-square extents of the bounding rectangle such that 45 degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

Parameters:

`x` - the *x* coordinate of the upper-left corner of the arc to be drawn.

`y` - the *y* coordinate of the upper-left corner of the arc to be drawn.

`width` - the width of the arc to be drawn.

`height` - the height of the arc to be drawn.

`startAngle` - the beginning angle.

`arcAngle` - the angular extent of the arc, relative to the start angle.

See Also:

fillArc(int, int, int, int, int, int)

---

fillArc

```
public abstract void fillArc(int  x,
                             int  y,
                             int  width,
                             int  height,
                             int  startAngle,
                             int  arcAngle)
```

Fills a circular or elliptical arc covering the specified rectangle.

The resulting arc begins at `startAngle` and extends for `arcAngle` degrees. Angles are interpreted such that 0 degrees is at the 3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin is (*x*, *y*) and whose size is specified by the `width` and `height` arguments.

The resulting arc covers an area `width + 1` pixels wide by `height + 1` pixels tall.

The angles are specified relative to the non-square extents of the bounding rectangle such that 45 degrees always falls on the line from the center of the ellipse to the upper right corner of the bounding rectangle. As a result, if the bounding rectangle is noticeably longer in one axis than the other, the angles to the start and end of the arc segment will be skewed farther along the longer axis of the bounds.

Parameters:

`x` - the *x* coordinate of the upper-left corner of the arc to be filled.
`y` - the *y* coordinate of the upper-left corner of the arc to be filled.
`width` - the width of the arc to be filled.
`height` - the height of the arc to be filled.
`startAngle` - the beginning angle.
`arcAngle` - the angular extent of the arc, relative to the start angle.

See Also:

`drawArc(int, int, int, int, int, int)`

---

drawPolyline

```
public abstract void drawPolyline(int[]  xPoints,
                                  int[]  yPoints,
                                  int  nPoints)
```

Draws a sequence of connected lines defined by arrays of *x* and *y* coordinates. Each pair of (*x*, *y*) coordinates defines a point. The figure is not closed if the first point differs from the last point.

Parameters:

`xPoints` - an array of *x* points
`yPoints` - an array of *y* points
`nPoints` - the total number of points

Since:

JDK1.1

See Also:

`drawPolygon(int[], int[], int)`

---

drawPolygon

```
public abstract void drawPolygon(int[]  xPoints,
                                 int[]  yPoints,
                                 int  nPoints)
```

Draws a closed polygon defined by arrays of *x* and *y* coordinates. Each pair of (*x*, *y*) coordinates defines a point.

This method draws the polygon defined by `nPoint` line segments, where the first `nPoint - 1` line segments are line segments from `(xPoints[i - 1], yPoints[i - 1])` to `(xPoints[i], yPoints[i])`, for 1 ≤ *i* ≤ `nPoints`. The figure is automatically closed by drawing a line connecting the final point to the first point, if those points are different.

Parameters:

`xPoints` - a an array of `x` coordinates.

`yPoints` - a an array of `y` coordinates.

`nPoints` - a the total number of points.

See Also:

`fillPolygon(int[], int[], int)`, `drawPolyline(int[], int[], int)`

---

drawPolygon

```
public void drawPolygon(Polygon  p)
```

Draws the outline of a polygon defined by the specified `Polygon` object.

Parameters:

`p` - the polygon to draw.

See Also:

`fillPolygon(int[], int[], int)`, `drawPolyline(int[], int[], int)`

---

fillPolygon

```
public abstract void fillPolygon(int[]  xPoints,
                                 int[]  yPoints,
                                 int  nPoints)
```

Fills a closed polygon defined by arrays of *x* and *y* coordinates.

This method draws the polygon defined by `nPoint` line segments, where the first `nPoint - 1` line segments are line segments from `(xPoints[i - 1], yPoints[i - 1])` to `(xPoints[i], yPoints[i])`, for 1 ≤ *i* ≤ `nPoints`. The figure is automatically closed by drawing a line connecting the final point to the first point, if those points are different.

The area inside the polygon is defined using an even-odd fill rule, also known as the alternating rule.

Parameters:

`xPoints` - a an array of `x` coordinates.

`yPoints` - a an array of `y` coordinates.

`nPoints` - a the total number of points.

See Also:

drawPolygon(int[], int[], int)

---

fillPolygon

```
public void fillPolygon(Polygon  p)
```

Fills the polygon defined by the specified Polygon object with the graphics context's current color.

The area inside the polygon is defined using an even-odd fill rule, also known as the alternating rule.

Parameters:

`p` - the polygon to fill.

See Also:

drawPolygon(int[], int[], int)

---

drawString

```
public abstract void drawString(String  str,
                                int  x,
                                int  y)
```

Draws the text given by the specified string, using this graphics context's current font and color. The baseline of the leftmost character is at position (*x*, *y*) in this graphics context's coordinate system.

Parameters:

`str` - the string to be drawn.

`x` - the *x* coordinate.

`y` - the *y* coordinate.

Throws:

NullPointerException - if `str` is `null`.

See Also:

drawBytes(byte[], int, int, int, int), drawChars(char[], int, int, int, int)

---

drawString

```
public abstract void drawString(AttributedCharacterIterator  iterator,
                                int  x,
                                int  y)
```

Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.

The baseline of the leftmost character is at position (*x*, *y*) in this graphics context's coordinate system.

Parameters:

iterator - the iterator whose text is to be drawn

x - the *x* coordinate.

y - the *y* coordinate.

Throws:

NullPointerException - if iterator is null.

See Also:

drawBytes(byte[], int, int, int, int), drawChars(char[], int, int, int, int)

---

drawChars

```
public void drawChars(char[]  data,
                      int  offset,
                      int  length,
                      int  x,
                      int  y)
```

Draws the text given by the specified character array, using this graphics context's current font and color. The baseline of the first character is at position (*x*, *y*) in this graphics context's coordinate system.

Parameters:

data - the array of characters to be drawn

offset - the start offset in the data

length - the number of characters to be drawn

x - the *x* coordinate of the baseline of the text

y - the *y* coordinate of the baseline of the text

Throws:

NullPointerException - if data is null.

IndexOutOfBoundsException - if offset or lengthis less than zero, or offset+length is greater than the length of the data array.

See Also:

drawBytes(byte[], int, int, int, int), drawString(java.lang.String, int, int)

---

drawBytes

```
public void drawBytes(byte[]  data,
                      int  offset,
                      int  length,
                      int  x,
                      int  y)
```

Draws the text given by the specified byte array, using this graphics context's current font and color. The baseline of the first character is at position (*x*, *y*) in this graphics context's coordinate system.

Use of this method is not recommended as each byte is interpreted as a Unicode code point in the range 0 to 255, and so can only be used to draw Latin characters in that range.

Parameters:

`data` - the data to be drawn
`offset` - the start offset in the data
`length` - the number of bytes that are drawn
`x` - the *x* coordinate of the baseline of the text
`y` - the *y* coordinate of the baseline of the text

Throws:

`NullPointerException` - if `data` is `null`.
`IndexOutOfBoundsException` - if `offset` or `length`is less than zero, or `offset+length` is greater than the length of the `data` array.

See Also:

`drawChars(char[], int, int, int, int)`, `drawString(java.lang.String, int, int)`

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                  int  x,
                                  int  y,
                                  ImageObserver  observer)
```

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (*x*, *y*) in this graphics context's coordinate space. Transparent pixels in the image do not affect whatever pixels are already there.

This method returns immediately in all cases, even if the complete image has not yet been loaded, and it has not been dithered and converted for the current output device.

If the image has completely loaded and its pixels are no longer being changed, then `drawImage` returns `true`. Otherwise, `drawImage` returns `false` and as more of the image becomes available or it is time to draw another frame of animation, the process that loads the image notifies the specified image observer.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`x` - the *x* coordinate.
`y` - the *y* coordinate.
`observer` - object to be notified as more of the image is converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                  int  x,
                                  int  y,
                                  int  width,
                                  int  height,
                                  ImageObserver  observer)
```

Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

The image is drawn inside the specified rectangle of this graphics context's coordinate space, and is scaled if necessary. Transparent pixels do not affect whatever pixels are already there.

This method returns immediately in all cases, even if the entire image has not yet been scaled, dithered, and converted for the current output device. If the current output representation is not yet complete, then `drawImage` returns `false`. As more of the image becomes available, the process that loads the image notifies the image observer by calling its `imageUpdate` method.

A scaled version of an image will not necessarily be available immediately just because an unscaled version of the image has been constructed for this output device. Each size of the image may be cached separately and generated from the original data in a separate image production sequence.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`x` - the *x* coordinate.
`y` - the *y* coordinate.
`width` - the width of the rectangle.
`height` - the height of the rectangle.
`observer` - object to be notified as more of the image is converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                  int  x,
```

```
                                int  y,
                                Color  bgcolor,
                                ImageObserver  observer)
```

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (*x*, *y*) in this graphics context's coordinate space. Transparent pixels are drawn in the specified background color.

This operation is equivalent to filling a rectangle of the width and height of the specified image with the given color and then drawing the image on top of it, but possibly more efficient.

This method returns immediately in all cases, even if the complete image has not yet been loaded, and it has not been dithered and converted for the current output device.

If the image has completely loaded and its pixels are no longer being changed, then `drawImage` returns `true`. Otherwise, `drawImage` returns `false` and as more of the image becomes available or it is time to draw another frame of animation, the process that loads the image notifies the specified image observer.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`x` - the *x* coordinate.
`y` - the *y* coordinate.
`bgcolor` - the background color to paint under the non-opaque portions of the image.
`observer` - object to be notified as more of the image is converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                int  x,
                                int  y,
                                int  width,
                                int  height,
                                Color  bgcolor,
                                ImageObserver  observer)
```

Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.

The image is drawn inside the specified rectangle of this graphics context's coordinate space, and is scaled if necessary. Transparent pixels are drawn in the specified background color. This operation is equivalent to filling a rectangle of the width and height of the specified image with the given color and then drawing the image on top of it, but possibly more efficient.

This method returns immediately in all cases, even if the entire image has not yet been scaled, dithered, and converted for the current output device. If the current output representation is not yet complete then `drawImage` returns `false`. As more of the image becomes available, the process that loads the image notifies the specified image observer.

A scaled version of an image will not necessarily be available immediately just because an unscaled version of the image has been constructed for this output device. Each size of the image may be cached separately and generated from the original data in a separate image production sequence.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`x` - the *x* coordinate.
`y` - the *y* coordinate.
`width` - the width of the rectangle.
`height` - the height of the rectangle.
`bgcolor` - the background color to paint under the non-opaque portions of the image.
`observer` - object to be notified as more of the image is converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                  int  dx1,
                                  int  dy1,
                                  int  dx2,
                                  int  dy2,
                                  int  sx1,
                                  int  sy1,
                                  int  sx2,
                                  int  sy2,
                                  ImageObserver  observer)
```

Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface. Transparent pixels do not affect whatever pixels are already there.

This method returns immediately in all cases, even if the image area to be drawn has not yet been scaled, dithered, and converted for the current output device. If the current output representation is not yet complete then `drawImage` returns `false`. As more of the image becomes available, the process that loads the image notifies the specified image observer.

This method always uses the unscaled version of the image to render the scaled rectangle and performs the required scaling on the fly. It does not use a cached, scaled

version of the image for this operation. Scaling of the image from source to destination is performed such that the first coordinate of the source rectangle is mapped to the first coordinate of the destination rectangle, and the second source coordinate is mapped to the second destination coordinate. The subimage is scaled and flipped as needed to preserve those mappings.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`dx1` - the *x* coordinate of the first corner of the destination rectangle.
`dy1` - the *y* coordinate of the first corner of the destination rectangle.
`dx2` - the *x* coordinate of the second corner of the destination rectangle.
`dy2` - the *y* coordinate of the second corner of the destination rectangle.
`sx1` - the *x* coordinate of the first corner of the source rectangle.
`sy1` - the *y* coordinate of the first corner of the source rectangle.
`sx2` - the *x* coordinate of the second corner of the source rectangle.
`sy2` - the *y* coordinate of the second corner of the source rectangle.
`observer` - object to be notified as more of the image is scaled and converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

Since:

JDK1.1

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

drawImage

```
public abstract boolean drawImage(Image  img,
                                  int  dx1,
                                  int  dy1,
                                  int  dx2,
                                  int  dy2,
                                  int  sx1,
                                  int  sy1,
                                  int  sx2,
                                  int  sy2,
                                  Color  bgcolor,
                                  ImageObserver  observer)
```

Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.

Transparent pixels are drawn in the specified background color. This operation is equivalent to filling a rectangle of the width and height of the specified image with the given color and then drawing the image on top of it, but possibly more efficient.

This method returns immediately in all cases, even if the image area to be drawn has not yet been scaled, dithered, and converted for the current output device. If the current output representation is not yet complete then `drawImage` returns `false`. As more of the

image becomes available, the process that loads the image notifies the specified image observer.

This method always uses the unscaled version of the image to render the scaled rectangle and performs the required scaling on the fly. It does not use a cached, scaled version of the image for this operation. Scaling of the image from source to destination is performed such that the first coordinate of the source rectangle is mapped to the first coordinate of the destination rectangle, and the second source coordinate is mapped to the second destination coordinate. The subimage is scaled and flipped as needed to preserve those mappings.

Parameters:

`img` - the specified image to be drawn. This method does nothing if `img` is null.
`dx1` - the *x* coordinate of the first corner of the destination rectangle.
`dy1` - the *y* coordinate of the first corner of the destination rectangle.
`dx2` - the *x* coordinate of the second corner of the destination rectangle.
`dy2` - the *y* coordinate of the second corner of the destination rectangle.
`sx1` - the *x* coordinate of the first corner of the source rectangle.
`sy1` - the *y* coordinate of the first corner of the source rectangle.
`sx2` - the *x* coordinate of the second corner of the source rectangle.
`sy2` - the *y* coordinate of the second corner of the source rectangle.
`bgcolor` - the background color to paint under the non-opaque portions of the image.
`observer` - object to be notified as more of the image is scaled and converted.

Returns:

`false` if the image pixels are still changing; `true` otherwise.

Since:

JDK1.1

See Also:

Image, ImageObserver, ImageObserver.imageUpdate(java.awt.Image, int, int, int, int, int)

---

dispose

```
public abstract void dispose()
```

Disposes of this graphics context and releases any system resources that it is using. A `Graphics` object cannot be used after `dispose`has been called.

When a Java program runs, a large number of `Graphics` objects can be created within a short time frame. Although the finalization process of the garbage collector also disposes of the same system resources, it is preferable to manually free the associated resources by calling this method rather than to rely on a finalization process which may not run to completion for a long period of time.

Graphics objects which are provided as arguments to the `paint` and `update` methods of components are automatically released by the system when those methods return. For

efficiency, programmers should call `dispose` when finished using a `Graphics` object only if it was created directly from a component or another `Graphics` object.

See Also:

`finalize()`, `Component.paint(java.awt.Graphics)`, `Component.update(java.awt.Graphics)`, `Component.getGraphics()`, `create()`

---

finalize

```
public void finalize()
```

Disposes of this graphics context once it is no longer referenced.

Overrides:

`finalize` in class `Object`

See Also:

`dispose()`

---

toString

```
public String toString()
```

Returns a `String` object representing this `Graphics` object's value.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this graphics context.

---

getClipRect

```
@Deprecated
public Rectangle getClipRect()
```

Deprecated. *As of JDK version 1.1, replaced by* `getClipBounds()`.

Returns the bounding rectangle of the current clipping area.

Returns:

the bounding rectangle of the current clipping area or `null` if no clip is set.

---

hitClip

```
public boolean hitClip(int  x,
                       int  y,
                       int  width,
                       int  height)
```

Returns true if the specified rectangular area might intersect the current clipping area. The coordinates of the specified rectangular area are in the user coordinate space and are relative to the coordinate system origin of this graphics context. This method may use an algorithm that calculates a result quickly but which sometimes might return true even if the specified rectangular area does not intersect the clipping area. The specific algorithm employed may thus trade off accuracy for speed, but it will never return false unless it can guarantee that the specified rectangular area does not intersect the current clipping area. The clipping area used by this method can represent the intersection of the user clip as specified through the clip methods of this graphics context as well as the clipping associated with the device or image bounds and window visibility.

Parameters:

`x` - the x coordinate of the rectangle to test against the clip
`y` - the y coordinate of the rectangle to test against the clip
`width` - the width of the rectangle to test against the clip
`height` - the height of the rectangle to test against the clip

Returns:

`true` if the specified rectangle intersects the bounds of the current clip; `false` otherwise.

---

getClipBounds

`public Rectangle getClipBounds(Rectangle r)`

Returns the bounding rectangle of the current clipping area. The coordinates in the rectangle are relative to the coordinate system origin of this graphics context. This method differs from `getClipBounds` in that an existing rectangle is used instead of allocating a new one. This method refers to the user clip, which is independent of the clipping associated with device bounds and window visibility. If no clip has previously been set, or if the clip has been cleared using `setClip(null)`, this method returns the specified `Rectangle`.

Parameters:

`r` - the rectangle where the current clipping area is copied to. Any current values in this rectangle are overwritten.

Returns:

the bounding rectangle of the current clipping area.
