| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AlphaComposite.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/Adjustable.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/AWTError.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/AlphaComposite.html)    [**NO FRAMES**](http://docs.google.com/AlphaComposite.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#2et92p0) | CONSTR | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | CONSTR | [METHOD](#3o7alnk) |

## **java.awt**

Class AlphaComposite

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

**All Implemented Interfaces:** [Composite](http://docs.google.com/java/awt/Composite.html)

public final class **AlphaComposite**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Composite](http://docs.google.com/java/awt/Composite.html)

The AlphaComposite class implements basic alpha compositing rules for combining source and destination colors to achieve blending and transparency effects with graphics and images. The specific rules implemented by this class are the basic set of 12 rules described in T. Porter and T. Duff, "Compositing Digital Images", SIGGRAPH 84, 253-259. The rest of this documentation assumes some familiarity with the definitions and concepts outlined in that paper.

This class extends the standard equations defined by Porter and Duff to include one additional factor. An instance of the AlphaComposite class can contain an alpha value that is used to modify the opacity or coverage of every source pixel before it is used in the blending equations.

It is important to note that the equations defined by the Porter and Duff paper are all defined to operate on color components that are premultiplied by their corresponding alpha components. Since the ColorModel and Raster classes allow the storage of pixel data in either premultiplied or non-premultiplied form, all input data must be normalized into premultiplied form before applying the equations and all results might need to be adjusted back to the form required by the destination before the pixel values are stored.

Also note that this class defines only the equations for combining color and alpha values in a purely mathematical sense. The accurate application of its equations depends on the way the data is retrieved from its sources and stored in its destinations. See [Implementation Caveats](#_3znysh7) for further information.

The following factors are used in the description of the blending equation in the Porter and Duff paper:

| Factor | Definition |
| --- | --- |
| *As* | the alpha component of the source pixel |
| *Cs* | a color component of the source pixel in premultiplied form |
| *Ad* | the alpha component of the destination pixel |
| *Cd* | a color component of the destination pixel in premultiplied form |
| *Fs* | the fraction of the source pixel that contributes to the output |
| *Fd* | the fraction of the destination pixel that contributes to the output |
| *Ar* | the alpha component of the result |
| *Cr* | a color component of the result in premultiplied form |

Using these factors, Porter and Duff define 12 ways of choosing the blending factors *Fs* and *Fd* to produce each of 12 desirable visual effects. The equations for determining *Fs* and *Fd* are given in the descriptions of the 12 static fields that specify visual effects. For example, the description for [SRC\_OVER](#3rdcrjn) specifies that *Fs* = 1 and *Fd* = (1-*As*). Once a set of equations for determining the blending factors is known they can then be applied to each pixel to produce a result using the following set of equations:

*Fs* = *f*(*Ad*)  
 *Fd* = *f*(*As*)  
 *Ar* = *As*\**Fs* + *Ad*\**Fd*  
 *Cr* = *Cs*\**Fs* + *Cd*\**Fd*

The following factors will be used to discuss our extensions to the blending equation in the Porter and Duff paper:

| Factor | Definition |
| --- | --- |
| *Csr* | one of the raw color components of the source pixel |
| *Cdr* | one of the raw color components of the destination pixel |
| *Aac* | the "extra" alpha component from the AlphaComposite instance |
| *Asr* | the raw alpha component of the source pixel |
| *Adr* | the raw alpha component of the destination pixel |
| *Adf* | the final alpha component stored in the destination |
| *Cdf* | the final raw color component stored in the destination |

### Preparing Inputs

The AlphaComposite class defines an additional alpha value that is applied to the source alpha. This value is applied as if an implicit SRC\_IN rule were first applied to the source pixel against a pixel with the indicated alpha by multiplying both the raw source alpha and the raw source colors by the alpha in the AlphaComposite. This leads to the following equation for producing the alpha used in the Porter and Duff blending equation:

*As* = *Asr* \* *Aac*

All of the raw source color components need to be multiplied by the alpha in the AlphaComposite instance. Additionally, if the source was not in premultiplied form then the color components also need to be multiplied by the source alpha. Thus, the equation for producing the source color components for the Porter and Duff equation depends on whether the source pixels are premultiplied or not:

*Cs* = *Csr* \* *Asr* \* *Aac* (if source is not premultiplied)  
 *Cs* = *Csr* \* *Aac* (if source is premultiplied)

No adjustment needs to be made to the destination alpha:

*Ad* = *Adr*

The destination color components need to be adjusted only if they are not in premultiplied form:

*Cd* = *Cdr* \* *Ad* (if destination is not premultiplied)   
 *Cd* = *Cdr* (if destination is premultiplied)

### Applying the Blending Equation

The adjusted *As*, *Ad*, *Cs*, and *Cd* are used in the standard Porter and Duff equations to calculate the blending factors *Fs* and *Fd* and then the resulting premultiplied components *Ar* and *Cr*.

### Preparing Results

The results only need to be adjusted if they are to be stored back into a destination buffer that holds data that is not premultiplied, using the following equations:

*Adf* = *Ar*  
 *Cdf* = *Cr* (if dest is premultiplied)  
 *Cdf* = *Cr* / *Ar* (if dest is not premultiplied)

Note that since the division is undefined if the resulting alpha is zero, the division in that case is omitted to avoid the "divide by zero" and the color components are left as all zeros.

### Performance Considerations

For performance reasons, it is preferrable that Raster objects passed to the compose method of a [CompositeContext](http://docs.google.com/java/awt/CompositeContext.html) object created by the AlphaComposite class have premultiplied data. If either the source Raster or the destination Raster is not premultiplied, however, appropriate conversions are performed before and after the compositing operation.

### Implementation Caveats

* Many sources, such as some of the opaque image types listed in the BufferedImage class, do not store alpha values for their pixels. Such sources supply an alpha of 1.0 for all of their pixels.
* Many destinations also have no place to store the alpha values that result from the blending calculations performed by this class. Such destinations thus implicitly discard the resulting alpha values that this class produces. It is recommended that such destinations should treat their stored color values as non-premultiplied and divide the resulting color values by the resulting alpha value before storing the color values and discarding the alpha value.
* The accuracy of the results depends on the manner in which pixels are stored in the destination. An image format that provides at least 8 bits of storage per color and alpha component is at least adequate for use as a destination for a sequence of a few to a dozen compositing operations. An image format with fewer than 8 bits of storage per component is of limited use for just one or two compositing operations before the rounding errors dominate the results. An image format that does not separately store color components is not a good candidate for any type of translucent blending. For example, BufferedImage.TYPE\_BYTE\_INDEXED should not be used as a destination for a blending operation because every operation can introduce large errors, due to the need to choose a pixel from a limited palette to match the results of the blending equations.
* Nearly all formats store pixels as discrete integers rather than the floating point values used in the reference equations above. The implementation can either scale the integer pixel values into floating point values in the range 0.0 to 1.0 or use slightly modified versions of the equations that operate entirely in the integer domain and yet produce analogous results to the reference equations.  
  Typically the integer values are related to the floating point values in such a way that the integer 0 is equated to the floating point value 0.0 and the integer 2^*n*-1 (where *n* is the number of bits in the representation) is equated to 1.0. For 8-bit representations, this means that 0x00 represents 0.0 and 0xff represents 1.0.
* The internal implementation can approximate some of the equations and it can also eliminate some steps to avoid unnecessary operations. For example, consider a discrete integer image with non-premultiplied alpha values that uses 8 bits per component for storage. The stored values for a nearly transparent darkened red might be:  
   (A, R, G, B) = (0x01, 0xb0, 0x00, 0x00)  
  If integer math were being used and this value were being composited in [SRC](#2s8eyo1) mode with no extra alpha, then the math would indicate that the results were (in integer format):  
   (A, R, G, B) = (0x01, 0x01, 0x00, 0x00)  
  Note that the intermediate values, which are always in premultiplied form, would only allow the integer red component to be either 0x00 or 0x01. When we try to store this result back into a destination that is not premultiplied, dividing out the alpha will give us very few choices for the non-premultiplied red value. In this case an implementation that performs the math in integer space without shortcuts is likely to end up with the final pixel values of:  
   (A, R, G, B) = (0x01, 0xff, 0x00, 0x00)  
  (Note that 0x01 divided by 0x01 gives you 1.0, which is equivalent to the value 0xff in an 8-bit storage format.)  
  Alternately, an implementation that uses floating point math might produce more accurate results and end up returning to the original pixel value with little, if any, roundoff error. Or, an implementation using integer math might decide that since the equations boil down to a virtual NOP on the color values if performed in a floating point space, it can transfer the pixel untouched to the destination and avoid all the math entirely.  
  These implementations all attempt to honor the same equations, but use different tradeoffs of integer and floating point math and reduced or full equations. To account for such differences, it is probably best to expect only that the premultiplied form of the results to match between implementations and image formats. In this case both answers, expressed in premultiplied form would equate to:  
   (A, R, G, B) = (0x01, 0x01, 0x00, 0x00)  
  and thus they would all match.
* Because of the technique of simplifying the equations for calculation efficiency, some implementations might perform differently when encountering result alpha values of 0.0 on a non-premultiplied destination. Note that the simplification of removing the divide by alpha in the case of the SRC rule is technically not valid if the denominator (alpha) is 0. But, since the results should only be expected to be accurate when viewed in premultiplied form, a resulting alpha of 0 essentially renders the resulting color components irrelevant and so exact behavior in this case should not be expected.

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

| **Field Summary** | |
| --- | --- |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**Clear**](http://docs.google.com/java/awt/AlphaComposite.html#Clear)            AlphaComposite object that implements the opaque CLEAR rule with an alpha of 1.0f. |
| static int | [**CLEAR**](http://docs.google.com/java/awt/AlphaComposite.html#CLEAR)            Both the color and the alpha of the destination are cleared (Porter-Duff Clear rule). |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**Dst**](http://docs.google.com/java/awt/AlphaComposite.html#Dst)            AlphaComposite object that implements the opaque DST rule with an alpha of 1.0f. |
| static int | [**DST**](http://docs.google.com/java/awt/AlphaComposite.html#DST)            The destination is left untouched (Porter-Duff Destination rule). |
| static int | [**DST\_ATOP**](http://docs.google.com/java/awt/AlphaComposite.html#DST_ATOP)            The part of the destination lying inside of the source is composited over the source and replaces the destination (Porter-Duff Destination Atop Source rule). |
| static int | [**DST\_IN**](http://docs.google.com/java/awt/AlphaComposite.html#DST_IN)            The part of the destination lying inside of the source replaces the destination (Porter-Duff Destination In Source rule). |
| static int | [**DST\_OUT**](http://docs.google.com/java/awt/AlphaComposite.html#DST_OUT)            The part of the destination lying outside of the source replaces the destination (Porter-Duff Destination Held Out By Source rule). |
| static int | [**DST\_OVER**](http://docs.google.com/java/awt/AlphaComposite.html#DST_OVER)            The destination is composited over the source and the result replaces the destination (Porter-Duff Destination Over Source rule). |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**DstAtop**](http://docs.google.com/java/awt/AlphaComposite.html#DstAtop)            AlphaComposite object that implements the opaque DST\_ATOP rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**DstIn**](http://docs.google.com/java/awt/AlphaComposite.html#DstIn)            AlphaComposite object that implements the opaque DST\_IN rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**DstOut**](http://docs.google.com/java/awt/AlphaComposite.html#DstOut)            AlphaComposite object that implements the opaque DST\_OUT rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**DstOver**](http://docs.google.com/java/awt/AlphaComposite.html#DstOver)            AlphaComposite object that implements the opaque DST\_OVER rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**Src**](http://docs.google.com/java/awt/AlphaComposite.html#Src)            AlphaComposite object that implements the opaque SRC rule with an alpha of 1.0f. |
| static int | [**SRC**](http://docs.google.com/java/awt/AlphaComposite.html#SRC)            The source is copied to the destination (Porter-Duff Source rule). |
| static int | [**SRC\_ATOP**](http://docs.google.com/java/awt/AlphaComposite.html#SRC_ATOP)            The part of the source lying inside of the destination is composited onto the destination (Porter-Duff Source Atop Destination rule). |
| static int | [**SRC\_IN**](http://docs.google.com/java/awt/AlphaComposite.html#SRC_IN)            The part of the source lying inside of the destination replaces the destination (Porter-Duff Source In Destination rule). |
| static int | [**SRC\_OUT**](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OUT)            The part of the source lying outside of the destination replaces the destination (Porter-Duff Source Held Out By Destination rule). |
| static int | [**SRC\_OVER**](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER)            The source is composited over the destination (Porter-Duff Source Over Destination rule). |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**SrcAtop**](http://docs.google.com/java/awt/AlphaComposite.html#SrcAtop)            AlphaComposite object that implements the opaque SRC\_ATOP rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**SrcIn**](http://docs.google.com/java/awt/AlphaComposite.html#SrcIn)            AlphaComposite object that implements the opaque SRC\_IN rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**SrcOut**](http://docs.google.com/java/awt/AlphaComposite.html#SrcOut)            AlphaComposite object that implements the opaque SRC\_OUT rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**SrcOver**](http://docs.google.com/java/awt/AlphaComposite.html#SrcOver)            AlphaComposite object that implements the opaque SRC\_OVER rule with an alpha of 1.0f. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**Xor**](http://docs.google.com/java/awt/AlphaComposite.html#Xor)            AlphaComposite object that implements the opaque XOR rule with an alpha of 1.0f. |
| static int | [**XOR**](http://docs.google.com/java/awt/AlphaComposite.html#XOR)            The part of the source that lies outside of the destination is combined with the part of the destination that lies outside of the source (Porter-Duff Source Xor Destination rule). |

| **Method Summary** | |
| --- | --- |
| [CompositeContext](http://docs.google.com/java/awt/CompositeContext.html) | [**createContext**](http://docs.google.com/java/awt/AlphaComposite.html#createContext(java.awt.image.ColorModel,%20java.awt.image.ColorModel,%20java.awt.RenderingHints))([ColorModel](http://docs.google.com/java/awt/image/ColorModel.html) srcColorModel, [ColorModel](http://docs.google.com/java/awt/image/ColorModel.html) dstColorModel, [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html) hints)            Creates a context for the compositing operation. |
| [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**derive**](http://docs.google.com/java/awt/AlphaComposite.html#derive(float))(float alpha)            Returns a similar AlphaComposite object that uses the specified alpha value. |
| [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**derive**](http://docs.google.com/java/awt/AlphaComposite.html#derive(int))(int rule)            Returns a similar AlphaComposite object that uses the specified compositing rule. |
| boolean | [**equals**](http://docs.google.com/java/awt/AlphaComposite.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Determines whether the specified object is equal to this AlphaComposite. |
| float | [**getAlpha**](http://docs.google.com/java/awt/AlphaComposite.html#getAlpha())()            Returns the alpha value of this AlphaComposite. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**getInstance**](http://docs.google.com/java/awt/AlphaComposite.html#getInstance(int))(int rule)            Creates an AlphaComposite object with the specified rule. |
| static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) | [**getInstance**](http://docs.google.com/java/awt/AlphaComposite.html#getInstance(int,%20float))(int rule, float alpha)            Creates an AlphaComposite object with the specified rule and the constant alpha to multiply with the alpha of the source. |
| int | [**getRule**](http://docs.google.com/java/awt/AlphaComposite.html#getRule())()            Returns the compositing rule of this AlphaComposite. |
| int | [**hashCode**](http://docs.google.com/java/awt/AlphaComposite.html#hashCode())()            Returns the hashcode for this composite. |

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

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

### CLEAR

public static final int **CLEAR**

Both the color and the alpha of the destination are cleared (Porter-Duff Clear rule). Neither the source nor the destination is used as input.

*Fs* = 0 and *Fd* = 0, thus:

*Ar* = 0  
 *Cr* = 0

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

### SRC

public static final int **SRC**

The source is copied to the destination (Porter-Duff Source rule). The destination is not used as input.

*Fs* = 1 and *Fd* = 0, thus:

*Ar* = *As*  
 *Cr* = *Cs*

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

### DST

public static final int **DST**

The destination is left untouched (Porter-Duff Destination rule).

*Fs* = 0 and *Fd* = 1, thus:

*Ar* = *Ad*  
 *Cr* = *Cd*

**Since:** 1.4 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.AlphaComposite.DST)

### SRC\_OVER

public static final int **SRC\_OVER**

The source is composited over the destination (Porter-Duff Source Over Destination rule).

*Fs* = 1 and *Fd* = (1-*As*), thus:

*Ar* = *As* + *Ad*\*(1-*As*)  
 *Cr* = *Cs* + *Cd*\*(1-*As*)

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

### DST\_OVER

public static final int **DST\_OVER**

The destination is composited over the source and the result replaces the destination (Porter-Duff Destination Over Source rule).

*Fs* = (1-*Ad*) and *Fd* = 1, thus:

*Ar* = *As*\*(1-*Ad*) + *Ad*  
 *Cr* = *Cs*\*(1-*Ad*) + *Cd*

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

### SRC\_IN

public static final int **SRC\_IN**

The part of the source lying inside of the destination replaces the destination (Porter-Duff Source In Destination rule).

*Fs* = *Ad* and *Fd* = 0, thus:

*Ar* = *As*\**Ad*  
 *Cr* = *Cs*\**Ad*

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

### DST\_IN

public static final int **DST\_IN**

The part of the destination lying inside of the source replaces the destination (Porter-Duff Destination In Source rule).

*Fs* = 0 and *Fd* = *As*, thus:

*Ar* = *Ad*\**As*  
 *Cr* = *Cd*\**As*

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

### SRC\_OUT

public static final int **SRC\_OUT**

The part of the source lying outside of the destination replaces the destination (Porter-Duff Source Held Out By Destination rule).

*Fs* = (1-*Ad*) and *Fd* = 0, thus:

*Ar* = *As*\*(1-*Ad*)  
 *Cr* = *Cs*\*(1-*Ad*)

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

### DST\_OUT

public static final int **DST\_OUT**

The part of the destination lying outside of the source replaces the destination (Porter-Duff Destination Held Out By Source rule).

*Fs* = 0 and *Fd* = (1-*As*), thus:

*Ar* = *Ad*\*(1-*As*)  
 *Cr* = *Cd*\*(1-*As*)

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

### SRC\_ATOP

public static final int **SRC\_ATOP**

The part of the source lying inside of the destination is composited onto the destination (Porter-Duff Source Atop Destination rule).

*Fs* = *Ad* and *Fd* = (1-*As*), thus:

*Ar* = *As*\**Ad* + *Ad*\*(1-*As*) = *Ad*  
 *Cr* = *Cs*\**Ad* + *Cd*\*(1-*As*)

**Since:** 1.4 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.AlphaComposite.SRC_ATOP)

### DST\_ATOP

public static final int **DST\_ATOP**

The part of the destination lying inside of the source is composited over the source and replaces the destination (Porter-Duff Destination Atop Source rule).

*Fs* = (1-*Ad*) and *Fd* = *As*, thus:

*Ar* = *As*\*(1-*Ad*) + *Ad*\**As* = *As*  
 *Cr* = *Cs*\*(1-*Ad*) + *Cd*\**As*

**Since:** 1.4 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.AlphaComposite.DST_ATOP)

### XOR

public static final int **XOR**

The part of the source that lies outside of the destination is combined with the part of the destination that lies outside of the source (Porter-Duff Source Xor Destination rule).

*Fs* = (1-*Ad*) and *Fd* = (1-*As*), thus:

*Ar* = *As*\*(1-*Ad*) + *Ad*\*(1-*As*)  
 *Cr* = *Cs*\*(1-*Ad*) + *Cd*\*(1-*As*)

**Since:** 1.4 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.AlphaComposite.XOR)

### Clear

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **Clear**

AlphaComposite object that implements the opaque CLEAR rule with an alpha of 1.0f.

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

### Src

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **Src**

AlphaComposite object that implements the opaque SRC rule with an alpha of 1.0f.

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

### Dst

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **Dst**

AlphaComposite object that implements the opaque DST rule with an alpha of 1.0f.

**Since:** 1.4 **See Also:**[DST](http://docs.google.com/java/awt/AlphaComposite.html#DST)

### SrcOver

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **SrcOver**

AlphaComposite object that implements the opaque SRC\_OVER rule with an alpha of 1.0f.

**See Also:**[SRC\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER)

### DstOver

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **DstOver**

AlphaComposite object that implements the opaque DST\_OVER rule with an alpha of 1.0f.

**See Also:**[DST\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#DST_OVER)

### SrcIn

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **SrcIn**

AlphaComposite object that implements the opaque SRC\_IN rule with an alpha of 1.0f.

**See Also:**[SRC\_IN](http://docs.google.com/java/awt/AlphaComposite.html#SRC_IN)

### DstIn

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **DstIn**

AlphaComposite object that implements the opaque DST\_IN rule with an alpha of 1.0f.

**See Also:**[DST\_IN](http://docs.google.com/java/awt/AlphaComposite.html#DST_IN)

### SrcOut

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **SrcOut**

AlphaComposite object that implements the opaque SRC\_OUT rule with an alpha of 1.0f.

**See Also:**[SRC\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OUT)

### DstOut

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **DstOut**

AlphaComposite object that implements the opaque DST\_OUT rule with an alpha of 1.0f.

**See Also:**[DST\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#DST_OUT)

### SrcAtop

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **SrcAtop**

AlphaComposite object that implements the opaque SRC\_ATOP rule with an alpha of 1.0f.

**Since:** 1.4 **See Also:**[SRC\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#SRC_ATOP)

### DstAtop

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **DstAtop**

AlphaComposite object that implements the opaque DST\_ATOP rule with an alpha of 1.0f.

**Since:** 1.4 **See Also:**[DST\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#DST_ATOP)

### Xor

public static final [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **Xor**

AlphaComposite object that implements the opaque XOR rule with an alpha of 1.0f.

**Since:** 1.4 **See Also:**[XOR](http://docs.google.com/java/awt/AlphaComposite.html#XOR)

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

### getInstance

public static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **getInstance**(int rule)

Creates an AlphaComposite object with the specified rule.

**Parameters:**rule - the compositing rule **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if rule is not one of the following: [CLEAR](http://docs.google.com/java/awt/AlphaComposite.html#CLEAR), [SRC](http://docs.google.com/java/awt/AlphaComposite.html#SRC), [DST](http://docs.google.com/java/awt/AlphaComposite.html#DST), [SRC\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER), [DST\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#DST_OVER), [SRC\_IN](http://docs.google.com/java/awt/AlphaComposite.html#SRC_IN), [DST\_IN](http://docs.google.com/java/awt/AlphaComposite.html#DST_IN), [SRC\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OUT), [DST\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#DST_OUT), [SRC\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#SRC_ATOP), [DST\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#DST_ATOP), or [XOR](http://docs.google.com/java/awt/AlphaComposite.html#XOR)

### getInstance

public static [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **getInstance**(int rule,  
 float alpha)

Creates an AlphaComposite object with the specified rule and the constant alpha to multiply with the alpha of the source. The source is multiplied with the specified alpha before being composited with the destination.

**Parameters:**rule - the compositing rulealpha - the constant alpha to be multiplied with the alpha of the source. alpha must be a floating point number in the inclusive range [0.0, 1.0]. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if alpha is less than 0.0 or greater than 1.0, or if rule is not one of the following: [CLEAR](http://docs.google.com/java/awt/AlphaComposite.html#CLEAR), [SRC](http://docs.google.com/java/awt/AlphaComposite.html#SRC), [DST](http://docs.google.com/java/awt/AlphaComposite.html#DST), [SRC\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER), [DST\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#DST_OVER), [SRC\_IN](http://docs.google.com/java/awt/AlphaComposite.html#SRC_IN), [DST\_IN](http://docs.google.com/java/awt/AlphaComposite.html#DST_IN), [SRC\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OUT), [DST\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#DST_OUT), [SRC\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#SRC_ATOP), [DST\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#DST_ATOP), or [XOR](http://docs.google.com/java/awt/AlphaComposite.html#XOR)

### createContext

public [CompositeContext](http://docs.google.com/java/awt/CompositeContext.html) **createContext**([ColorModel](http://docs.google.com/java/awt/image/ColorModel.html) srcColorModel,  
 [ColorModel](http://docs.google.com/java/awt/image/ColorModel.html) dstColorModel,  
 [RenderingHints](http://docs.google.com/java/awt/RenderingHints.html) hints)

Creates a context for the compositing operation. The context contains state that is used in performing the compositing operation.

**Specified by:**[createContext](http://docs.google.com/java/awt/Composite.html#createContext(java.awt.image.ColorModel,%20java.awt.image.ColorModel,%20java.awt.RenderingHints)) in interface [Composite](http://docs.google.com/java/awt/Composite.html) **Parameters:**srcColorModel - the [ColorModel](http://docs.google.com/java/awt/image/ColorModel.html) of the sourcedstColorModel - the ColorModel of the destinationhints - the hint that the context object uses to choose between rendering alternatives **Returns:**the CompositeContext object to be used to perform compositing operations.

### getAlpha

public float **getAlpha**()

Returns the alpha value of this AlphaComposite. If this AlphaComposite does not have an alpha value, 1.0 is returned.

**Returns:**the alpha value of this AlphaComposite.

### getRule

public int **getRule**()

Returns the compositing rule of this AlphaComposite.

**Returns:**the compositing rule of this AlphaComposite.

### derive

public [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **derive**(int rule)

Returns a similar AlphaComposite object that uses the specified compositing rule. If this object already uses the specified compositing rule, this object is returned.

**Parameters:**rule - the compositing rule **Returns:**an AlphaComposite object derived from this object that uses the specified compositing rule. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if rule is not one of the following: [CLEAR](http://docs.google.com/java/awt/AlphaComposite.html#CLEAR), [SRC](http://docs.google.com/java/awt/AlphaComposite.html#SRC), [DST](http://docs.google.com/java/awt/AlphaComposite.html#DST), [SRC\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OVER), [DST\_OVER](http://docs.google.com/java/awt/AlphaComposite.html#DST_OVER), [SRC\_IN](http://docs.google.com/java/awt/AlphaComposite.html#SRC_IN), [DST\_IN](http://docs.google.com/java/awt/AlphaComposite.html#DST_IN), [SRC\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#SRC_OUT), [DST\_OUT](http://docs.google.com/java/awt/AlphaComposite.html#DST_OUT), [SRC\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#SRC_ATOP), [DST\_ATOP](http://docs.google.com/java/awt/AlphaComposite.html#DST_ATOP), or [XOR](http://docs.google.com/java/awt/AlphaComposite.html#XOR)**Since:** 1.6

### derive

public [AlphaComposite](http://docs.google.com/java/awt/AlphaComposite.html) **derive**(float alpha)

Returns a similar AlphaComposite object that uses the specified alpha value. If this object already has the specified alpha value, this object is returned.

**Parameters:**alpha - the constant alpha to be multiplied with the alpha of the source. alpha must be a floating point number in the inclusive range [0.0, 1.0]. **Returns:**an AlphaComposite object derived from this object that uses the specified alpha value. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if alpha is less than 0.0 or greater than 1.0**Since:** 1.6

### hashCode

public int **hashCode**()

Returns the hashcode for this composite.

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

### equals

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

Determines whether the specified object is equal to this AlphaComposite.

The result is true if and only if the argument is not null and is an AlphaComposite object that has the same compositing rule and alpha value as this object.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**obj - the Object to test for equality **Returns:**true if obj equals this AlphaComposite; false otherwise.**See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.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/AlphaComposite.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/Adjustable.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/AWTError.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/AlphaComposite.html)    [**NO FRAMES**](http://docs.google.com/AlphaComposite.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#2et92p0) | CONSTR | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | CONSTR | [METHOD](#3o7alnk) |

[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).