| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AffineTransform.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   [**NEXT CLASS**](http://docs.google.com/java/awt/geom/Arc2D.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/geom/AffineTransform.html)    [**NO FRAMES**](http://docs.google.com/AffineTransform.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#2et92p0) | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: [FIELD](#4d34og8) | [CONSTR](#3j2qqm3) | [METHOD](#qsh70q) |

## **java.awt.geom**

Class AffineTransform

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

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html)

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

The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates that preserves the "straightness" and "parallelness" of lines. Affine transformations can be constructed using sequences of translations, scales, flips, rotations, and shears.

Such a coordinate transformation can be represented by a 3 row by 3 column matrix with an implied last row of [ 0 0 1 ]. This matrix transforms source coordinates (x,y) into destination coordinates (x',y') by considering them to be a column vector and multiplying the coordinate vector by the matrix according to the following process:

[ x'] [ m00 m01 m02 ] [ x ] [ m00x + m01y + m02 ]  
 [ y'] = [ m10 m11 m12 ] [ y ] = [ m10x + m11y + m12 ]  
 [ 1 ] [ 0 0 1 ] [ 1 ] [ 1 ]

#### Handling 90-Degree Rotations

In some variations of the rotate methods in the AffineTransform class, a double-precision argument specifies the angle of rotation in radians. These methods have special handling for rotations of approximately 90 degrees (including multiples such as 180, 270, and 360 degrees), so that the common case of quadrant rotation is handled more efficiently. This special handling can cause angles very close to multiples of 90 degrees to be treated as if they were exact multiples of 90 degrees. For small multiples of 90 degrees the range of angles treated as a quadrant rotation is approximately 0.00000121 degrees wide. This section explains why such special care is needed and how it is implemented.

Since 90 degrees is represented as PI/2 in radians, and since PI is a transcendental (and therefore irrational) number, it is not possible to exactly represent a multiple of 90 degrees as an exact double precision value measured in radians. As a result it is theoretically impossible to describe quadrant rotations (90, 180, 270 or 360 degrees) using these values. Double precision floating point values can get very close to non-zero multiples of PI/2 but never close enough for the sine or cosine to be exactly 0.0, 1.0 or -1.0. The implementations of Math.sin() and Math.cos() correspondingly never return 0.0 for any case other than Math.sin(0.0). These same implementations do, however, return exactly 1.0 and -1.0 for some range of numbers around each multiple of 90 degrees since the correct answer is so close to 1.0 or -1.0 that the double precision significand cannot represent the difference as accurately as it can for numbers that are near 0.0.

The net result of these issues is that if the Math.sin() and Math.cos() methods are used to directly generate the values for the matrix modifications during these radian-based rotation operations then the resulting transform is never strictly classifiable as a quadrant rotation even for a simple case like rotate(Math.PI/2.0), due to minor variations in the matrix caused by the non-0.0 values obtained for the sine and cosine. If these transforms are not classified as quadrant rotations then subsequent code which attempts to optimize further operations based upon the type of the transform will be relegated to its most general implementation.

Because quadrant rotations are fairly common, this class should handle these cases reasonably quickly, both in applying the rotations to the transform and in applying the resulting transform to the coordinates. To facilitate this optimal handling, the methods which take an angle of rotation measured in radians attempt to detect angles that are intended to be quadrant rotations and treat them as such. These methods therefore treat an angle *theta* as a quadrant rotation if either Math.sin(*theta*) or Math.cos(*theta*) returns exactly 1.0 or -1.0. As a rule of thumb, this property holds true for a range of approximately 0.0000000211 radians (or 0.00000121 degrees) around small multiples of Math.PI/2.0.

**Since:** 1.2 **See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.awt.geom.AffineTransform)

| **Field Summary** | |
| --- | --- |
| static int | [**TYPE\_FLIP**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP)            This flag bit indicates that the transform defined by this object performs a mirror image flip about some axis which changes the normally right handed coordinate system into a left handed system in addition to the conversions indicated by other flag bits. |
| static int | [**TYPE\_GENERAL\_ROTATION**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION)            This flag bit indicates that the transform defined by this object performs a rotation by an arbitrary angle in addition to the conversions indicated by other flag bits. |
| static int | [**TYPE\_GENERAL\_SCALE**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE)            This flag bit indicates that the transform defined by this object performs a general scale in addition to the conversions indicated by other flag bits. |
| static int | [**TYPE\_GENERAL\_TRANSFORM**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM)            This constant indicates that the transform defined by this object performs an arbitrary conversion of the input coordinates. |
| static int | [**TYPE\_IDENTITY**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY)            This constant indicates that the transform defined by this object is an identity transform. |
| static int | [**TYPE\_MASK\_ROTATION**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_MASK_ROTATION)            This constant is a bit mask for any of the rotation flag bits. |
| static int | [**TYPE\_MASK\_SCALE**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_MASK_SCALE)            This constant is a bit mask for any of the scale flag bits. |
| static int | [**TYPE\_QUADRANT\_ROTATION**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION)            This flag bit indicates that the transform defined by this object performs a quadrant rotation by some multiple of 90 degrees in addition to the conversions indicated by other flag bits. |
| static int | [**TYPE\_TRANSLATION**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION)            This flag bit indicates that the transform defined by this object performs a translation in addition to the conversions indicated by other flag bits. |
| static int | [**TYPE\_UNIFORM\_SCALE**](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE)            This flag bit indicates that the transform defined by this object performs a uniform scale in addition to the conversions indicated by other flag bits. |

| **Constructor Summary** | |
| --- | --- |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform())()            Constructs a new AffineTransform representing the Identity transformation. |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Constructs a new AffineTransform that is a copy of the specified AffineTransform object. |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform(double%5B%5D))(double[] flatmatrix)            Constructs a new AffineTransform from an array of double precision values representing either the 4 non-translation entries or the 6 specifiable entries of the 3x3 transformation matrix. |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform(double,%20double,%20double,%20double,%20double,%20double))(double m00, double m10, double m01, double m11, double m02, double m12)            Constructs a new AffineTransform from 6 double precision values representing the 6 specifiable entries of the 3x3 transformation matrix. |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform(float%5B%5D))(float[] flatmatrix)            Constructs a new AffineTransform from an array of floating point values representing either the 4 non-translation enries or the 6 specifiable entries of the 3x3 transformation matrix. |
| [**AffineTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#AffineTransform(float,%20float,%20float,%20float,%20float,%20float))(float m00, float m10, float m01, float m11, float m02, float m12)            Constructs a new AffineTransform from 6 floating point values representing the 6 specifiable entries of the 3x3 transformation matrix. |

| **Method Summary** | |
| --- | --- |
| [Object](http://docs.google.com/java/lang/Object.html) | [**clone**](http://docs.google.com/java/awt/geom/AffineTransform.html#clone())()            Returns a copy of this AffineTransform object. |
| void | [**concatenate**](http://docs.google.com/java/awt/geom/AffineTransform.html#concatenate(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Concatenates an AffineTransform Tx to this AffineTransform Cx in the most commonly useful way to provide a new user space that is mapped to the former user space by Tx. |
| [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**createInverse**](http://docs.google.com/java/awt/geom/AffineTransform.html#createInverse())()            Returns an AffineTransform object representing the inverse transformation. |
| [Shape](http://docs.google.com/java/awt/Shape.html) | [**createTransformedShape**](http://docs.google.com/java/awt/geom/AffineTransform.html#createTransformedShape(java.awt.Shape))([Shape](http://docs.google.com/java/awt/Shape.html) pSrc)            Returns a new [Shape](http://docs.google.com/java/awt/Shape.html) object defined by the geometry of the specified Shape after it has been transformed by this transform. |
| void | [**deltaTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#deltaTransform(double%5B%5D,%20int,%20double%5B%5D,%20int,%20int))(double[] srcPts, int srcOff, double[] dstPts, int dstOff, int numPts)            Transforms an array of relative distance vectors by this transform. |
| [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) | [**deltaTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#deltaTransform(java.awt.geom.Point2D,%20java.awt.geom.Point2D))([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc, [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)            Transforms the relative distance vector specified by ptSrc and stores the result in ptDst. |
| boolean | [**equals**](http://docs.google.com/java/awt/geom/AffineTransform.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Returns true if this AffineTransform represents the same affine coordinate transform as the specified argument. |
| double | [**getDeterminant**](http://docs.google.com/java/awt/geom/AffineTransform.html#getDeterminant())()            Returns the determinant of the matrix representation of the transform. |
| void | [**getMatrix**](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))(double[] flatmatrix)            Retrieves the 6 specifiable values in the 3x3 affine transformation matrix and places them into an array of double precisions values. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getQuadrantRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getQuadrantRotateInstance(int))(int numquadrants)            Returns a transform that rotates coordinates by the specified number of quadrants. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getQuadrantRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getQuadrantRotateInstance(int,%20double,%20double))(int numquadrants, double anchorx, double anchory)            Returns a transform that rotates coordinates by the specified number of quadrants around the specified anchor point. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getRotateInstance(double))(double theta)            Returns a transform representing a rotation transformation. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getRotateInstance(double,%20double))(double vecx, double vecy)            Returns a transform that rotates coordinates according to a rotation vector. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getRotateInstance(double,%20double,%20double))(double theta, double anchorx, double anchory)            Returns a transform that rotates coordinates around an anchor point. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getRotateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getRotateInstance(double,%20double,%20double,%20double))(double vecx, double vecy, double anchorx, double anchory)            Returns a transform that rotates coordinates around an anchor point accordinate to a rotation vector. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getScaleInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getScaleInstance(double,%20double))(double sx, double sy)            Returns a transform representing a scaling transformation. |
| double | [**getScaleX**](http://docs.google.com/java/awt/geom/AffineTransform.html#getScaleX())()            Returns the X coordinate scaling element (m00) of the 3x3 affine transformation matrix. |
| double | [**getScaleY**](http://docs.google.com/java/awt/geom/AffineTransform.html#getScaleY())()            Returns the Y coordinate scaling element (m11) of the 3x3 affine transformation matrix. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getShearInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getShearInstance(double,%20double))(double shx, double shy)            Returns a transform representing a shearing transformation. |
| double | [**getShearX**](http://docs.google.com/java/awt/geom/AffineTransform.html#getShearX())()            Returns the X coordinate shearing element (m01) of the 3x3 affine transformation matrix. |
| double | [**getShearY**](http://docs.google.com/java/awt/geom/AffineTransform.html#getShearY())()            Returns the Y coordinate shearing element (m10) of the 3x3 affine transformation matrix. |
| static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) | [**getTranslateInstance**](http://docs.google.com/java/awt/geom/AffineTransform.html#getTranslateInstance(double,%20double))(double tx, double ty)            Returns a transform representing a translation transformation. |
| double | [**getTranslateX**](http://docs.google.com/java/awt/geom/AffineTransform.html#getTranslateX())()            Returns the X coordinate of the translation element (m02) of the 3x3 affine transformation matrix. |
| double | [**getTranslateY**](http://docs.google.com/java/awt/geom/AffineTransform.html#getTranslateY())()            Returns the Y coordinate of the translation element (m12) of the 3x3 affine transformation matrix. |
| int | [**getType**](http://docs.google.com/java/awt/geom/AffineTransform.html#getType())()            Retrieves the flag bits describing the conversion properties of this transform. |
| int | [**hashCode**](http://docs.google.com/java/awt/geom/AffineTransform.html#hashCode())()            Returns the hashcode for this transform. |
| void | [**inverseTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#inverseTransform(double%5B%5D,%20int,%20double%5B%5D,%20int,%20int))(double[] srcPts, int srcOff, double[] dstPts, int dstOff, int numPts)            Inverse transforms an array of double precision coordinates by this transform. |
| [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) | [**inverseTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#inverseTransform(java.awt.geom.Point2D,%20java.awt.geom.Point2D))([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc, [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)            Inverse transforms the specified ptSrc and stores the result in ptDst. |
| void | [**invert**](http://docs.google.com/java/awt/geom/AffineTransform.html#invert())()            Sets this transform to the inverse of itself. |
| boolean | [**isIdentity**](http://docs.google.com/java/awt/geom/AffineTransform.html#isIdentity())()            Returns true if this AffineTransform is an identity transform. |
| void | [**preConcatenate**](http://docs.google.com/java/awt/geom/AffineTransform.html#preConcatenate(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Concatenates an AffineTransform Tx to this AffineTransform Cx in a less commonly used way such that Tx modifies the coordinate transformation relative to the absolute pixel space rather than relative to the existing user space. |
| void | [**quadrantRotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#quadrantRotate(int))(int numquadrants)            Concatenates this transform with a transform that rotates coordinates by the specified number of quadrants. |
| void | [**quadrantRotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#quadrantRotate(int,%20double,%20double))(int numquadrants, double anchorx, double anchory)            Concatenates this transform with a transform that rotates coordinates by the specified number of quadrants around the specified anchor point. |
| void | [**rotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#rotate(double))(double theta)            Concatenates this transform with a rotation transformation. |
| void | [**rotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#rotate(double,%20double))(double vecx, double vecy)            Concatenates this transform with a transform that rotates coordinates according to a rotation vector. |
| void | [**rotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#rotate(double,%20double,%20double))(double theta, double anchorx, double anchory)            Concatenates this transform with a transform that rotates coordinates around an anchor point. |
| void | [**rotate**](http://docs.google.com/java/awt/geom/AffineTransform.html#rotate(double,%20double,%20double,%20double))(double vecx, double vecy, double anchorx, double anchory)            Concatenates this transform with a transform that rotates coordinates around an anchor point according to a rotation vector. |
| void | [**scale**](http://docs.google.com/java/awt/geom/AffineTransform.html#scale(double,%20double))(double sx, double sy)            Concatenates this transform with a scaling transformation. |
| void | [**setToIdentity**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToIdentity())()            Resets this transform to the Identity transform. |
| void | [**setToQuadrantRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToQuadrantRotation(int))(int numquadrants)            Sets this transform to a rotation transformation that rotates coordinates by the specified number of quadrants. |
| void | [**setToQuadrantRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToQuadrantRotation(int,%20double,%20double))(int numquadrants, double anchorx, double anchory)            Sets this transform to a translated rotation transformation that rotates coordinates by the specified number of quadrants around the specified anchor point. |
| void | [**setToRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToRotation(double))(double theta)            Sets this transform to a rotation transformation. |
| void | [**setToRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToRotation(double,%20double))(double vecx, double vecy)            Sets this transform to a rotation transformation that rotates coordinates according to a rotation vector. |
| void | [**setToRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToRotation(double,%20double,%20double))(double theta, double anchorx, double anchory)            Sets this transform to a translated rotation transformation. |
| void | [**setToRotation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToRotation(double,%20double,%20double,%20double))(double vecx, double vecy, double anchorx, double anchory)            Sets this transform to a rotation transformation that rotates coordinates around an anchor point according to a rotation vector. |
| void | [**setToScale**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToScale(double,%20double))(double sx, double sy)            Sets this transform to a scaling transformation. |
| void | [**setToShear**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToShear(double,%20double))(double shx, double shy)            Sets this transform to a shearing transformation. |
| void | [**setToTranslation**](http://docs.google.com/java/awt/geom/AffineTransform.html#setToTranslation(double,%20double))(double tx, double ty)            Sets this transform to a translation transformation. |
| void | [**setTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#setTransform(java.awt.geom.AffineTransform))([AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) Tx)            Sets this transform to a copy of the transform in the specified AffineTransform object. |
| void | [**setTransform**](http://docs.google.com/java/awt/geom/AffineTransform.html#setTransform(double,%20double,%20double,%20double,%20double,%20double))(double m00, double m10, double m01, double m11, double m02, double m12)            Sets this transform to the matrix specified by the 6 double precision values. |
| void | [**shear**](http://docs.google.com/java/awt/geom/AffineTransform.html#shear(double,%20double))(double shx, double shy)            Concatenates this transform with a shearing transformation. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/awt/geom/AffineTransform.html#toString())()            Returns a String that represents the value of this [Object](http://docs.google.com/java/lang/Object.html). |
| void | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(double%5B%5D,%20int,%20double%5B%5D,%20int,%20int))(double[] srcPts, int srcOff, double[] dstPts, int dstOff, int numPts)            Transforms an array of double precision coordinates by this transform. |
| void | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(double%5B%5D,%20int,%20float%5B%5D,%20int,%20int))(double[] srcPts, int srcOff, float[] dstPts, int dstOff, int numPts)            Transforms an array of double precision coordinates by this transform and stores the results into an array of floats. |
| void | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(float%5B%5D,%20int,%20double%5B%5D,%20int,%20int))(float[] srcPts, int srcOff, double[] dstPts, int dstOff, int numPts)            Transforms an array of floating point coordinates by this transform and stores the results into an array of doubles. |
| void | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(float%5B%5D,%20int,%20float%5B%5D,%20int,%20int))(float[] srcPts, int srcOff, float[] dstPts, int dstOff, int numPts)            Transforms an array of floating point coordinates by this transform. |
| void | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(java.awt.geom.Point2D%5B%5D,%20int,%20java.awt.geom.Point2D%5B%5D,%20int,%20int))([Point2D](http://docs.google.com/java/awt/geom/Point2D.html)[] ptSrc, int srcOff, [Point2D](http://docs.google.com/java/awt/geom/Point2D.html)[] ptDst, int dstOff, int numPts)            Transforms an array of point objects by this transform. |
| [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) | [**transform**](http://docs.google.com/java/awt/geom/AffineTransform.html#transform(java.awt.geom.Point2D,%20java.awt.geom.Point2D))([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc, [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)            Transforms the specified ptSrc and stores the result in ptDst. |
| void | [**translate**](http://docs.google.com/java/awt/geom/AffineTransform.html#translate(double,%20double))(double tx, double ty)            Concatenates this transform with a translation transformation. |

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

### TYPE\_IDENTITY

public static final int **TYPE\_IDENTITY**

This constant indicates that the transform defined by this object is an identity transform. An identity transform is one in which the output coordinates are always the same as the input coordinates. If this transform is anything other than the identity transform, the type will either be the constant GENERAL\_TRANSFORM or a combination of the appropriate flag bits for the various coordinate conversions that this transform performs.

**Since:** 1.2 **See Also:**[TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_IDENTITY)

### TYPE\_TRANSLATION

public static final int **TYPE\_TRANSLATION**

This flag bit indicates that the transform defined by this object performs a translation in addition to the conversions indicated by other flag bits. A translation moves the coordinates by a constant amount in x and y without changing the length or angle of vectors.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_TRANSLATION)

### TYPE\_UNIFORM\_SCALE

public static final int **TYPE\_UNIFORM\_SCALE**

This flag bit indicates that the transform defined by this object performs a uniform scale in addition to the conversions indicated by other flag bits. A uniform scale multiplies the length of vectors by the same amount in both the x and y directions without changing the angle between vectors. This flag bit is mutually exclusive with the TYPE\_GENERAL\_SCALE flag.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_UNIFORM_SCALE)

### TYPE\_GENERAL\_SCALE

public static final int **TYPE\_GENERAL\_SCALE**

This flag bit indicates that the transform defined by this object performs a general scale in addition to the conversions indicated by other flag bits. A general scale multiplies the length of vectors by different amounts in the x and y directions without changing the angle between perpendicular vectors. This flag bit is mutually exclusive with the TYPE\_UNIFORM\_SCALE flag.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_GENERAL_SCALE)

### TYPE\_MASK\_SCALE

public static final int **TYPE\_MASK\_SCALE**

This constant is a bit mask for any of the scale flag bits.

**Since:** 1.2 **See Also:**[TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_MASK_SCALE)

### TYPE\_FLIP

public static final int **TYPE\_FLIP**

This flag bit indicates that the transform defined by this object performs a mirror image flip about some axis which changes the normally right handed coordinate system into a left handed system in addition to the conversions indicated by other flag bits. A right handed coordinate system is one where the positive X axis rotates counterclockwise to overlay the positive Y axis similar to the direction that the fingers on your right hand curl when you stare end on at your thumb. A left handed coordinate system is one where the positive X axis rotates clockwise to overlay the positive Y axis similar to the direction that the fingers on your left hand curl. There is no mathematical way to determine the angle of the original flipping or mirroring transformation since all angles of flip are identical given an appropriate adjusting rotation.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_FLIP)

### TYPE\_QUADRANT\_ROTATION

public static final int **TYPE\_QUADRANT\_ROTATION**

This flag bit indicates that the transform defined by this object performs a quadrant rotation by some multiple of 90 degrees in addition to the conversions indicated by other flag bits. A rotation changes the angles of vectors by the same amount regardless of the original direction of the vector and without changing the length of the vector. This flag bit is mutually exclusive with the TYPE\_GENERAL\_ROTATION flag.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_QUADRANT_ROTATION)

### TYPE\_GENERAL\_ROTATION

public static final int **TYPE\_GENERAL\_ROTATION**

This flag bit indicates that the transform defined by this object performs a rotation by an arbitrary angle in addition to the conversions indicated by other flag bits. A rotation changes the angles of vectors by the same amount regardless of the original direction of the vector and without changing the length of the vector. This flag bit is mutually exclusive with the TYPE\_QUADRANT\_ROTATION flag.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_GENERAL_ROTATION)

### TYPE\_MASK\_ROTATION

public static final int **TYPE\_MASK\_ROTATION**

This constant is a bit mask for any of the rotation flag bits.

**Since:** 1.2 **See Also:**[TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_MASK_ROTATION)

### TYPE\_GENERAL\_TRANSFORM

public static final int **TYPE\_GENERAL\_TRANSFORM**

This constant indicates that the transform defined by this object performs an arbitrary conversion of the input coordinates. If this transform can be classified by any of the above constants, the type will either be the constant TYPE\_IDENTITY or a combination of the appropriate flag bits for the various coordinate conversions that this transform performs.

**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_FLIP](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_FLIP), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [Constant Field Values](http://docs.google.com/constant-values.html#java.awt.geom.AffineTransform.TYPE_GENERAL_TRANSFORM)

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

### AffineTransform

public **AffineTransform**()

Constructs a new AffineTransform representing the Identity transformation.

**Since:** 1.2

### AffineTransform

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

Constructs a new AffineTransform that is a copy of the specified AffineTransform object.

**Parameters:**Tx - the AffineTransform object to copy**Since:** 1.2

### AffineTransform

public **AffineTransform**(float m00,  
 float m10,  
 float m01,  
 float m11,  
 float m02,  
 float m12)

Constructs a new AffineTransform from 6 floating point values representing the 6 specifiable entries of the 3x3 transformation matrix.

**Parameters:**m00 - the X coordinate scaling element of the 3x3 matrixm10 - the Y coordinate shearing element of the 3x3 matrixm01 - the X coordinate shearing element of the 3x3 matrixm11 - the Y coordinate scaling element of the 3x3 matrixm02 - the X coordinate translation element of the 3x3 matrixm12 - the Y coordinate translation element of the 3x3 matrix**Since:** 1.2

### AffineTransform

public **AffineTransform**(float[] flatmatrix)

Constructs a new AffineTransform from an array of floating point values representing either the 4 non-translation enries or the 6 specifiable entries of the 3x3 transformation matrix. The values are retrieved from the array as { m00 m10 m01 m11 [m02 m12]}.

**Parameters:**flatmatrix - the float array containing the values to be set in the new AffineTransform object. The length of the array is assumed to be at least 4. If the length of the array is less than 6, only the first 4 values are taken. If the length of the array is greater than 6, the first 6 values are taken.**Since:** 1.2

### AffineTransform

public **AffineTransform**(double m00,  
 double m10,  
 double m01,  
 double m11,  
 double m02,  
 double m12)

Constructs a new AffineTransform from 6 double precision values representing the 6 specifiable entries of the 3x3 transformation matrix.

**Parameters:**m00 - the X coordinate scaling element of the 3x3 matrixm10 - the Y coordinate shearing element of the 3x3 matrixm01 - the X coordinate shearing element of the 3x3 matrixm11 - the Y coordinate scaling element of the 3x3 matrixm02 - the X coordinate translation element of the 3x3 matrixm12 - the Y coordinate translation element of the 3x3 matrix**Since:** 1.2

### AffineTransform

public **AffineTransform**(double[] flatmatrix)

Constructs a new AffineTransform from an array of double precision values representing either the 4 non-translation entries or the 6 specifiable entries of the 3x3 transformation matrix. The values are retrieved from the array as { m00 m10 m01 m11 [m02 m12]}.

**Parameters:**flatmatrix - the double array containing the values to be set in the new AffineTransform object. The length of the array is assumed to be at least 4. If the length of the array is less than 6, only the first 4 values are taken. If the length of the array is greater than 6, the first 6 values are taken.**Since:** 1.2

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

### getTranslateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getTranslateInstance**(double tx,  
 double ty)

Returns a transform representing a translation transformation. The matrix representing the returned transform is:

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

**Parameters:**tx - the distance by which coordinates are translated in the X axis directionty - the distance by which coordinates are translated in the Y axis direction **Returns:**an AffineTransform object that represents a translation transformation, created with the specified vector.**Since:** 1.2

### getRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getRotateInstance**(double theta)

Returns a transform representing a rotation transformation. The matrix representing the returned transform is:

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

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radians **Returns:**an AffineTransform object that is a rotation transformation, created with the specified angle of rotation.**Since:** 1.2

### getRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getRotateInstance**(double theta,  
 double anchorx,  
 double anchory)

Returns a transform that rotates coordinates around an anchor point. This operation is equivalent to translating the coordinates so that the anchor point is at the origin (S1), then rotating them about the new origin (S2), and finally translating so that the intermediate origin is restored to the coordinates of the original anchor point (S3).

This operation is equivalent to the following sequence of calls:

AffineTransform Tx = new AffineTransform();  
 Tx.translate(anchorx, anchory); // S3: final translation  
 Tx.rotate(theta); // S2: rotate around anchor  
 Tx.translate(-anchorx, -anchory); // S1: translate anchor to origin

The matrix representing the returned transform is:

[ cos(theta) -sin(theta) x-x\*cos+y\*sin ]  
 [ sin(theta) cos(theta) y-x\*sin-y\*cos ]  
 [ 0 0 1 ]

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radiansanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point **Returns:**an AffineTransform object that rotates coordinates around the specified point by the specified angle of rotation.**Since:** 1.2

### getRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getRotateInstance**(double vecx,  
 double vecy)

Returns a transform that rotates coordinates according to a rotation vector. All coordinates rotate about the origin by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, an identity transform is returned. This operation is equivalent to calling:

AffineTransform.getRotateInstance(Math.atan2(vecy, vecx));

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vector **Returns:**an AffineTransform object that rotates coordinates according to the specified rotation vector.**Since:** 1.6

### getRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getRotateInstance**(double vecx,  
 double vecy,  
 double anchorx,  
 double anchory)

Returns a transform that rotates coordinates around an anchor point accordinate to a rotation vector. All coordinates rotate about the specified anchor coordinates by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, an identity transform is returned. This operation is equivalent to calling:

AffineTransform.getRotateInstance(Math.atan2(vecy, vecx),  
 anchorx, anchory);

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vectoranchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point **Returns:**an AffineTransform object that rotates coordinates around the specified point according to the specified rotation vector.**Since:** 1.6

### getQuadrantRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getQuadrantRotateInstance**(int numquadrants)

Returns a transform that rotates coordinates by the specified number of quadrants. This operation is equivalent to calling:

AffineTransform.getRotateInstance(numquadrants \* Math.PI / 2.0);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate by **Returns:**an AffineTransform object that rotates coordinates by the specified number of quadrants.**Since:** 1.6

### getQuadrantRotateInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getQuadrantRotateInstance**(int numquadrants,  
 double anchorx,  
 double anchory)

Returns a transform that rotates coordinates by the specified number of quadrants around the specified anchor point. This operation is equivalent to calling:

AffineTransform.getRotateInstance(numquadrants \* Math.PI / 2.0,  
 anchorx, anchory);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate byanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point **Returns:**an AffineTransform object that rotates coordinates by the specified number of quadrants around the specified anchor point.**Since:** 1.6

### getScaleInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getScaleInstance**(double sx,  
 double sy)

Returns a transform representing a scaling transformation. The matrix representing the returned transform is:

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

**Parameters:**sx - the factor by which coordinates are scaled along the X axis directionsy - the factor by which coordinates are scaled along the Y axis direction **Returns:**an AffineTransform object that scales coordinates by the specified factors.**Since:** 1.2

### getShearInstance

public static [AffineTransform](http://docs.google.com/java/awt/geom/AffineTransform.html) **getShearInstance**(double shx,  
 double shy)

Returns a transform representing a shearing transformation. The matrix representing the returned transform is:

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

**Parameters:**shx - the multiplier by which coordinates are shifted in the direction of the positive X axis as a factor of their Y coordinateshy - the multiplier by which coordinates are shifted in the direction of the positive Y axis as a factor of their X coordinate **Returns:**an AffineTransform object that shears coordinates by the specified multipliers.**Since:** 1.2

### getType

public int **getType**()

Retrieves the flag bits describing the conversion properties of this transform. The return value is either one of the constants TYPE\_IDENTITY or TYPE\_GENERAL\_TRANSFORM, or a combination of the appriopriate flag bits. A valid combination of flag bits is an exclusive OR operation that can combine the TYPE\_TRANSLATION flag bit in addition to either of the TYPE\_UNIFORM\_SCALE or TYPE\_GENERAL\_SCALE flag bits as well as either of the TYPE\_QUADRANT\_ROTATION or TYPE\_GENERAL\_ROTATION flag bits.

**Returns:**the OR combination of any of the indicated flags that apply to this transform**Since:** 1.2 **See Also:**[TYPE\_IDENTITY](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_IDENTITY), [TYPE\_TRANSLATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_TRANSLATION), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE), [TYPE\_GENERAL\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_SCALE), [TYPE\_QUADRANT\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_QUADRANT_ROTATION), [TYPE\_GENERAL\_ROTATION](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_ROTATION), [TYPE\_GENERAL\_TRANSFORM](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_GENERAL_TRANSFORM)

### getDeterminant

public double **getDeterminant**()

Returns the determinant of the matrix representation of the transform. The determinant is useful both to determine if the transform can be inverted and to get a single value representing the combined X and Y scaling of the transform.

If the determinant is non-zero, then this transform is invertible and the various methods that depend on the inverse transform do not need to throw a [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html). If the determinant is zero then this transform can not be inverted since the transform maps all input coordinates onto a line or a point. If the determinant is near enough to zero then inverse transform operations might not carry enough precision to produce meaningful results.

If this transform represents a uniform scale, as indicated by the getType method then the determinant also represents the square of the uniform scale factor by which all of the points are expanded from or contracted towards the origin. If this transform represents a non-uniform scale or more general transform then the determinant is not likely to represent a value useful for any purpose other than determining if inverse transforms are possible.

Mathematically, the determinant is calculated using the formula:

| m00 m01 m02 |  
 | m10 m11 m12 | = m00 \* m11 - m01 \* m10  
 | 0 0 1 |

**Returns:**the determinant of the matrix used to transform the coordinates.**Since:** 1.2 **See Also:**[getType()](http://docs.google.com/java/awt/geom/AffineTransform.html#getType()), [createInverse()](http://docs.google.com/java/awt/geom/AffineTransform.html#createInverse()), [inverseTransform(java.awt.geom.Point2D, java.awt.geom.Point2D)](http://docs.google.com/java/awt/geom/AffineTransform.html#inverseTransform(java.awt.geom.Point2D,%20java.awt.geom.Point2D)), [TYPE\_UNIFORM\_SCALE](http://docs.google.com/java/awt/geom/AffineTransform.html#TYPE_UNIFORM_SCALE)

### getMatrix

public void **getMatrix**(double[] flatmatrix)

Retrieves the 6 specifiable values in the 3x3 affine transformation matrix and places them into an array of double precisions values. The values are stored in the array as { m00 m10 m01 m11 m02 m12 }. An array of 4 doubles can also be specified, in which case only the first four elements representing the non-transform parts of the array are retrieved and the values are stored into the array as { m00 m10 m01 m11 }

**Parameters:**flatmatrix - the double array used to store the returned values.**Since:** 1.2 **See Also:**[getScaleX()](http://docs.google.com/java/awt/geom/AffineTransform.html#getScaleX()), [getScaleY()](http://docs.google.com/java/awt/geom/AffineTransform.html#getScaleY()), [getShearX()](http://docs.google.com/java/awt/geom/AffineTransform.html#getShearX()), [getShearY()](http://docs.google.com/java/awt/geom/AffineTransform.html#getShearY()), [getTranslateX()](http://docs.google.com/java/awt/geom/AffineTransform.html#getTranslateX()), [getTranslateY()](http://docs.google.com/java/awt/geom/AffineTransform.html#getTranslateY())

### getScaleX

public double **getScaleX**()

Returns the X coordinate scaling element (m00) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the X coordinate of the scaling element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### getScaleY

public double **getScaleY**()

Returns the Y coordinate scaling element (m11) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the Y coordinate of the scaling element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### getShearX

public double **getShearX**()

Returns the X coordinate shearing element (m01) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the X coordinate of the shearing element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### getShearY

public double **getShearY**()

Returns the Y coordinate shearing element (m10) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the Y coordinate of the shearing element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### getTranslateX

public double **getTranslateX**()

Returns the X coordinate of the translation element (m02) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the X coordinate of the translation element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### getTranslateY

public double **getTranslateY**()

Returns the Y coordinate of the translation element (m12) of the 3x3 affine transformation matrix.

**Returns:**a double value that is the Y coordinate of the translation element of the affine transformation matrix.**Since:** 1.2 **See Also:**[getMatrix(double[])](http://docs.google.com/java/awt/geom/AffineTransform.html#getMatrix(double%5B%5D))

### translate

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

Concatenates this transform with a translation transformation. This is equivalent to calling concatenate(T), where T is an AffineTransform represented by the following matrix:

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

**Parameters:**tx - the distance by which coordinates are translated in the X axis directionty - the distance by which coordinates are translated in the Y axis direction**Since:** 1.2

### rotate

public void **rotate**(double theta)

Concatenates this transform with a rotation transformation. This is equivalent to calling concatenate(R), where R is an AffineTransform represented by the following matrix:

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

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radians**Since:** 1.2

### rotate

public void **rotate**(double theta,  
 double anchorx,  
 double anchory)

Concatenates this transform with a transform that rotates coordinates around an anchor point. This operation is equivalent to translating the coordinates so that the anchor point is at the origin (S1), then rotating them about the new origin (S2), and finally translating so that the intermediate origin is restored to the coordinates of the original anchor point (S3).

This operation is equivalent to the following sequence of calls:

translate(anchorx, anchory); // S3: final translation  
 rotate(theta); // S2: rotate around anchor  
 translate(-anchorx, -anchory); // S1: translate anchor to origin

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radiansanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.2

### rotate

public void **rotate**(double vecx,  
 double vecy)

Concatenates this transform with a transform that rotates coordinates according to a rotation vector. All coordinates rotate about the origin by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, no additional rotation is added to this transform. This operation is equivalent to calling:

rotate(Math.atan2(vecy, vecx));

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vector**Since:** 1.6

### rotate

public void **rotate**(double vecx,  
 double vecy,  
 double anchorx,  
 double anchory)

Concatenates this transform with a transform that rotates coordinates around an anchor point according to a rotation vector. All coordinates rotate about the specified anchor coordinates by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, the transform is not modified in any way. This method is equivalent to calling:

rotate(Math.atan2(vecy, vecx), anchorx, anchory);

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vectoranchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.6

### quadrantRotate

public void **quadrantRotate**(int numquadrants)

Concatenates this transform with a transform that rotates coordinates by the specified number of quadrants. This is equivalent to calling:

rotate(numquadrants \* Math.PI / 2.0);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate by**Since:** 1.6

### quadrantRotate

public void **quadrantRotate**(int numquadrants,  
 double anchorx,  
 double anchory)

Concatenates this transform with a transform that rotates coordinates by the specified number of quadrants around the specified anchor point. This method is equivalent to calling:

rotate(numquadrants \* Math.PI / 2.0, anchorx, anchory);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate byanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.6

### scale

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

Concatenates this transform with a scaling transformation. This is equivalent to calling concatenate(S), where S is an AffineTransform represented by the following matrix:

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

**Parameters:**sx - the factor by which coordinates are scaled along the X axis directionsy - the factor by which coordinates are scaled along the Y axis direction**Since:** 1.2

### shear

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

Concatenates this transform with a shearing transformation. This is equivalent to calling concatenate(SH), where SH is an AffineTransform represented by the following matrix:

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

**Parameters:**shx - the multiplier by which coordinates are shifted in the direction of the positive X axis as a factor of their Y coordinateshy - the multiplier by which coordinates are shifted in the direction of the positive Y axis as a factor of their X coordinate**Since:** 1.2

### setToIdentity

public void **setToIdentity**()

Resets this transform to the Identity transform.

**Since:** 1.2

### setToTranslation

public void **setToTranslation**(double tx,  
 double ty)

Sets this transform to a translation transformation. The matrix representing this transform becomes:

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

**Parameters:**tx - the distance by which coordinates are translated in the X axis directionty - the distance by which coordinates are translated in the Y axis direction**Since:** 1.2

### setToRotation

public void **setToRotation**(double theta)

Sets this transform to a rotation transformation. The matrix representing this transform becomes:

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

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radians**Since:** 1.2

### setToRotation

public void **setToRotation**(double theta,  
 double anchorx,  
 double anchory)

Sets this transform to a translated rotation transformation. This operation is equivalent to translating the coordinates so that the anchor point is at the origin (S1), then rotating them about the new origin (S2), and finally translating so that the intermediate origin is restored to the coordinates of the original anchor point (S3).

This operation is equivalent to the following sequence of calls:

setToTranslation(anchorx, anchory); // S3: final translation  
 rotate(theta); // S2: rotate around anchor  
 translate(-anchorx, -anchory); // S1: translate anchor to origin

The matrix representing this transform becomes:

[ cos(theta) -sin(theta) x-x\*cos+y\*sin ]  
 [ sin(theta) cos(theta) y-x\*sin-y\*cos ]  
 [ 0 0 1 ]

Rotating by a positive angle theta rotates points on the positive X axis toward the positive Y axis. Note also the discussion of [Handling 90-Degree Rotations](#3znysh7) above.

**Parameters:**theta - the angle of rotation measured in radiansanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.2

### setToRotation

public void **setToRotation**(double vecx,  
 double vecy)

Sets this transform to a rotation transformation that rotates coordinates according to a rotation vector. All coordinates rotate about the origin by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, the transform is set to an identity transform. This operation is equivalent to calling:

setToRotation(Math.atan2(vecy, vecx));

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vector**Since:** 1.6

### setToRotation

public void **setToRotation**(double vecx,  
 double vecy,  
 double anchorx,  
 double anchory)

Sets this transform to a rotation transformation that rotates coordinates around an anchor point according to a rotation vector. All coordinates rotate about the specified anchor coordinates by the same amount. The amount of rotation is such that coordinates along the former positive X axis will subsequently align with the vector pointing from the origin to the specified vector coordinates. If both vecx and vecy are 0.0, the transform is set to an identity transform. This operation is equivalent to calling:

setToTranslation(Math.atan2(vecy, vecx), anchorx, anchory);

**Parameters:**vecx - the X coordinate of the rotation vectorvecy - the Y coordinate of the rotation vectoranchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.6

### setToQuadrantRotation

public void **setToQuadrantRotation**(int numquadrants)

Sets this transform to a rotation transformation that rotates coordinates by the specified number of quadrants. This operation is equivalent to calling:

setToRotation(numquadrants \* Math.PI / 2.0);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate by**Since:** 1.6

### setToQuadrantRotation

public void **setToQuadrantRotation**(int numquadrants,  
 double anchorx,  
 double anchory)

Sets this transform to a translated rotation transformation that rotates coordinates by the specified number of quadrants around the specified anchor point. This operation is equivalent to calling:

setToRotation(numquadrants \* Math.PI / 2.0, anchorx, anchory);

Rotating by a positive number of quadrants rotates points on the positive X axis toward the positive Y axis.

**Parameters:**numquadrants - the number of 90 degree arcs to rotate byanchorx - the X coordinate of the rotation anchor pointanchory - the Y coordinate of the rotation anchor point**Since:** 1.6

### setToScale

public void **setToScale**(double sx,  
 double sy)

Sets this transform to a scaling transformation. The matrix representing this transform becomes:

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

**Parameters:**sx - the factor by which coordinates are scaled along the X axis directionsy - the factor by which coordinates are scaled along the Y axis direction**Since:** 1.2

### setToShear

public void **setToShear**(double shx,  
 double shy)

Sets this transform to a shearing transformation. The matrix representing this transform becomes:

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

**Parameters:**shx - the multiplier by which coordinates are shifted in the direction of the positive X axis as a factor of their Y coordinateshy - the multiplier by which coordinates are shifted in the direction of the positive Y axis as a factor of their X coordinate**Since:** 1.2

### setTransform

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

Sets this transform to a copy of the transform in the specified AffineTransform object.

**Parameters:**Tx - the AffineTransform object from which to copy the transform**Since:** 1.2

### setTransform

public void **setTransform**(double m00,  
 double m10,  
 double m01,  
 double m11,  
 double m02,  
 double m12)

Sets this transform to the matrix specified by the 6 double precision values.

**Parameters:**m00 - the X coordinate scaling element of the 3x3 matrixm10 - the Y coordinate shearing element of the 3x3 matrixm01 - the X coordinate shearing element of the 3x3 matrixm11 - the Y coordinate scaling element of the 3x3 matrixm02 - the X coordinate translation element of the 3x3 matrixm12 - the Y coordinate translation element of the 3x3 matrix**Since:** 1.2

### concatenate

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

Concatenates an AffineTransform Tx to this AffineTransform Cx in the most commonly useful way to provide a new user space that is mapped to the former user space by Tx. Cx is updated to perform the combined transformation. Transforming a point p by the updated transform Cx' is equivalent to first transforming p by Tx and then transforming the result by the original transform Cx like this: Cx'(p) = Cx(Tx(p)) In matrix notation, if this transform Cx is represented by the matrix [this] and Tx is represented by the matrix [Tx] then this method does the following:

[this] = [this] x [Tx]

**Parameters:**Tx - the AffineTransform object to be concatenated with this AffineTransform object.**Since:** 1.2 **See Also:**[preConcatenate(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/geom/AffineTransform.html#preConcatenate(java.awt.geom.AffineTransform))

### preConcatenate

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

Concatenates an AffineTransform Tx to this AffineTransform Cx in a less commonly used way such that Tx modifies the coordinate transformation relative to the absolute pixel space rather than relative to the existing user space. Cx is updated to perform the combined transformation. Transforming a point p by the updated transform Cx' is equivalent to first transforming p by the original transform Cx and then transforming the result by Tx like this: Cx'(p) = Tx(Cx(p)) In matrix notation, if this transform Cx is represented by the matrix [this] and Tx is represented by the matrix [Tx] then this method does the following:

[this] = [Tx] x [this]

**Parameters:**Tx - the AffineTransform object to be concatenated with this AffineTransform object.**Since:** 1.2 **See Also:**[concatenate(java.awt.geom.AffineTransform)](http://docs.google.com/java/awt/geom/AffineTransform.html#concatenate(java.awt.geom.AffineTransform))

### createInverse

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

Returns an AffineTransform object representing the inverse transformation. The inverse transform Tx' of this transform Tx maps coordinates transformed by Tx back to their original coordinates. In other words, Tx'(Tx(p)) = p = Tx(Tx'(p)).

If this transform maps all coordinates onto a point or a line then it will not have an inverse, since coordinates that do not lie on the destination point or line will not have an inverse mapping. The getDeterminant method can be used to determine if this transform has no inverse, in which case an exception will be thrown if the createInverse method is called.

**Returns:**a new AffineTransform object representing the inverse transformation. **Throws:** [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html) - if the matrix cannot be inverted.**Since:** 1.2 **See Also:**[getDeterminant()](http://docs.google.com/java/awt/geom/AffineTransform.html#getDeterminant())

### invert

public void **invert**()  
 throws [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html)

Sets this transform to the inverse of itself. The inverse transform Tx' of this transform Tx maps coordinates transformed by Tx back to their original coordinates. In other words, Tx'(Tx(p)) = p = Tx(Tx'(p)).

If this transform maps all coordinates onto a point or a line then it will not have an inverse, since coordinates that do not lie on the destination point or line will not have an inverse mapping. The getDeterminant method can be used to determine if this transform has no inverse, in which case an exception will be thrown if the invert method is called.

**Throws:** [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html) - if the matrix cannot be inverted.**Since:** 1.6 **See Also:**[getDeterminant()](http://docs.google.com/java/awt/geom/AffineTransform.html#getDeterminant())

### transform

public [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) **transform**([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc,  
 [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)

Transforms the specified ptSrc and stores the result in ptDst. If ptDst is null, a new [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) object is allocated and then the result of the transformation is stored in this object. In either case, ptDst, which contains the transformed point, is returned for convenience. If ptSrc and ptDst are the same object, the input point is correctly overwritten with the transformed point.

**Parameters:**ptSrc - the specified Point2D to be transformedptDst - the specified Point2D that stores the result of transforming ptSrc **Returns:**the ptDst after transforming ptSrc and stroring the result in ptDst.**Since:** 1.2

### transform

public void **transform**([Point2D](http://docs.google.com/java/awt/geom/Point2D.html)[] ptSrc,  
 int srcOff,  
 [Point2D](http://docs.google.com/java/awt/geom/Point2D.html)[] ptDst,  
 int dstOff,  
 int numPts)

Transforms an array of point objects by this transform. If any element of the ptDst array is null, a new Point2D object is allocated and stored into that element before storing the results of the transformation.

Note that this method does not take any precautions to avoid problems caused by storing results into Point2D objects that will be used as the source for calculations further down the source array. This method does guarantee that if a specified Point2D object is both the source and destination for the same single point transform operation then the results will not be stored until the calculations are complete to avoid storing the results on top of the operands. If, however, the destination Point2D object for one operation is the same object as the source Point2D object for another operation further down the source array then the original coordinates in that point are overwritten before they can be converted.

**Parameters:**ptSrc - the array containing the source point objectsptDst - the array into which the transform point objects are returnedsrcOff - the offset to the first point object to be transformed in the source arraydstOff - the offset to the location of the first transformed point object that is stored in the destination arraynumPts - the number of point objects to be transformed**Since:** 1.2

### transform

public void **transform**(float[] srcPts,  
 int srcOff,  
 float[] dstPts,  
 int dstOff,  
 int numPts)

Transforms an array of floating point coordinates by this transform. The two coordinate array sections can be exactly the same or can be overlapping sections of the same array without affecting the validity of the results. This method ensures that no source coordinates are overwritten by a previous operation before they can be transformed. The coordinates are stored in the arrays starting at the specified offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source point coordinates. Each point is stored as a pair of x, y coordinates.dstPts - the array into which the transformed point coordinates are returned. Each point is stored as a pair of x, y coordinates.srcOff - the offset to the first point to be transformed in the source arraydstOff - the offset to the location of the first transformed point that is stored in the destination arraynumPts - the number of points to be transformed**Since:** 1.2

### transform

public void **transform**(double[] srcPts,  
 int srcOff,  
 double[] dstPts,  
 int dstOff,  
 int numPts)

Transforms an array of double precision coordinates by this transform. The two coordinate array sections can be exactly the same or can be overlapping sections of the same array without affecting the validity of the results. This method ensures that no source coordinates are overwritten by a previous operation before they can be transformed. The coordinates are stored in the arrays starting at the indicated offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source point coordinates. Each point is stored as a pair of x, y coordinates.dstPts - the array into which the transformed point coordinates are returned. Each point is stored as a pair of x, y coordinates.srcOff - the offset to the first point to be transformed in the source arraydstOff - the offset to the location of the first transformed point that is stored in the destination arraynumPts - the number of point objects to be transformed**Since:** 1.2

### transform

public void **transform**(float[] srcPts,  
 int srcOff,  
 double[] dstPts,  
 int dstOff,  
 int numPts)

Transforms an array of floating point coordinates by this transform and stores the results into an array of doubles. The coordinates are stored in the arrays starting at the specified offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source point coordinates. Each point is stored as a pair of x, y coordinates.dstPts - the array into which the transformed point coordinates are returned. Each point is stored as a pair of x, y coordinates.srcOff - the offset to the first point to be transformed in the source arraydstOff - the offset to the location of the first transformed point that is stored in the destination arraynumPts - the number of points to be transformed**Since:** 1.2

### transform

public void **transform**(double[] srcPts,  
 int srcOff,  
 float[] dstPts,  
 int dstOff,  
 int numPts)

Transforms an array of double precision coordinates by this transform and stores the results into an array of floats. The coordinates are stored in the arrays starting at the specified offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source point coordinates. Each point is stored as a pair of x, y coordinates.dstPts - the array into which the transformed point coordinates are returned. Each point is stored as a pair of x, y coordinates.srcOff - the offset to the first point to be transformed in the source arraydstOff - the offset to the location of the first transformed point that is stored in the destination arraynumPts - the number of point objects to be transformed**Since:** 1.2

### inverseTransform

public [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) **inverseTransform**([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc,  
 [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)  
 throws [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html)

Inverse transforms the specified ptSrc and stores the result in ptDst. If ptDst is null, a new Point2D object is allocated and then the result of the transform is stored in this object. In either case, ptDst, which contains the transformed point, is returned for convenience. If ptSrc and ptDst are the same object, the input point is correctly overwritten with the transformed point.

**Parameters:**ptSrc - the point to be inverse transformedptDst - the resulting transformed point **Returns:**ptDst, which contains the result of the inverse transform. **Throws:** [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html) - if the matrix cannot be inverted.**Since:** 1.2

### inverseTransform

public void **inverseTransform**(double[] srcPts,  
 int srcOff,  
 double[] dstPts,  
 int dstOff,  
 int numPts)  
 throws [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html)

Inverse transforms an array of double precision coordinates by this transform. The two coordinate array sections can be exactly the same or can be overlapping sections of the same array without affecting the validity of the results. This method ensures that no source coordinates are overwritten by a previous operation before they can be transformed. The coordinates are stored in the arrays starting at the specified offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source point coordinates. Each point is stored as a pair of x, y coordinates.dstPts - the array into which the transformed point coordinates are returned. Each point is stored as a pair of x, y coordinates.srcOff - the offset to the first point to be transformed in the source arraydstOff - the offset to the location of the first transformed point that is stored in the destination arraynumPts - the number of point objects to be transformed **Throws:** [NoninvertibleTransformException](http://docs.google.com/java/awt/geom/NoninvertibleTransformException.html) - if the matrix cannot be inverted.**Since:** 1.2

### deltaTransform

public [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) **deltaTransform**([Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptSrc,  
 [Point2D](http://docs.google.com/java/awt/geom/Point2D.html) ptDst)

Transforms the relative distance vector specified by ptSrc and stores the result in ptDst. A relative distance vector is transformed without applying the translation components of the affine transformation matrix using the following equations:

[ x' ] [ m00 m01 (m02) ] [ x ] [ m00x + m01y ]  
 [ y' ] = [ m10 m11 (m12) ] [ y ] = [ m10x + m11y ]  
 [ (1) ] [ (0) (0) ( 1 ) ] [ (1) ] [ (1) ]

If ptDst is null, a new Point2D object is allocated and then the result of the transform is stored in this object. In either case, ptDst, which contains the transformed point, is returned for convenience. If ptSrc and ptDst are the same object, the input point is correctly overwritten with the transformed point.

**Parameters:**ptSrc - the distance vector to be delta transformedptDst - the resulting transformed distance vector **Returns:**ptDst, which contains the result of the transformation.**Since:** 1.2

### deltaTransform

public void **deltaTransform**(double[] srcPts,  
 int srcOff,  
 double[] dstPts,  
 int dstOff,  
 int numPts)

Transforms an array of relative distance vectors by this transform. A relative distance vector is transformed without applying the translation components of the affine transformation matrix using the following equations:

[ x' ] [ m00 m01 (m02) ] [ x ] [ m00x + m01y ]  
 [ y' ] = [ m10 m11 (m12) ] [ y ] = [ m10x + m11y ]  
 [ (1) ] [ (0) (0) ( 1 ) ] [ (1) ] [ (1) ]

The two coordinate array sections can be exactly the same or can be overlapping sections of the same array without affecting the validity of the results. This method ensures that no source coordinates are overwritten by a previous operation before they can be transformed. The coordinates are stored in the arrays starting at the indicated offset in the order [x0, y0, x1, y1, ..., xn, yn].

**Parameters:**srcPts - the array containing the source distance vectors. Each vector is stored as a pair of relative x, y coordinates.dstPts - the array into which the transformed distance vectors are returned. Each vector is stored as a pair of relative x, y coordinates.srcOff - the offset to the first vector to be transformed in the source arraydstOff - the offset to the location of the first transformed vector that is stored in the destination arraynumPts - the number of vector coordinate pairs to be transformed**Since:** 1.2

### createTransformedShape

public [Shape](http://docs.google.com/java/awt/Shape.html) **createTransformedShape**([Shape](http://docs.google.com/java/awt/Shape.html) pSrc)

Returns a new [Shape](http://docs.google.com/java/awt/Shape.html) object defined by the geometry of the specified Shape after it has been transformed by this transform.

**Parameters:**pSrc - the specified Shape object to be transformed by this transform. **Returns:**a new Shape object that defines the geometry of the transformed Shape, or null if pSrc is null.**Since:** 1.2

### toString

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

Returns a String that represents the value of this [Object](http://docs.google.com/java/lang/Object.html).

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a String representing the value of this Object.**Since:** 1.2

### isIdentity

public boolean **isIdentity**()

Returns true if this AffineTransform is an identity transform.

**Returns:**true if this AffineTransform is an identity transform; false otherwise.**Since:** 1.2

### clone

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

Returns a copy of this AffineTransform object.

**Overrides:**[clone](http://docs.google.com/java/lang/Object.html#clone()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**an Object that is a copy of this AffineTransform object.**Since:** 1.2 **See Also:**[Cloneable](http://docs.google.com/java/lang/Cloneable.html)

### hashCode

public int **hashCode**()

Returns the hashcode for this transform.

**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 transform.**Since:** 1.2 **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)

Returns true if this AffineTransform represents the same affine coordinate transform as the specified argument.

**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 with this AffineTransform **Returns:**true if obj equals this AffineTransform object; false otherwise.**Since:** 1.2 **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/AffineTransform.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   [**NEXT CLASS**](http://docs.google.com/java/awt/geom/Arc2D.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/geom/AffineTransform.html)    [**NO FRAMES**](http://docs.google.com/AffineTransform.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#2et92p0) | [CONSTR](#tyjcwt) | [METHOD](#3dy6vkm) | DETAIL: [FIELD](#4d34og8) | [CONSTR](#3j2qqm3) | [METHOD](#qsh70q) |

[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).