We will use boldface for row vectors such as **0** and **d** and calligraphic symbols to represent matrices. The translation vector **d** is transposed to convert it to a column vector. The upper left-hand submatrix $\mathcal{A}$ produces rotation and scaling while the vector $\mathbf{d} = [d_x, d_y]$ produces translation.

Homogeneous coordinates have another advantage because they can be used to distinguish between points and vectors. Unlike points, vectors should remain invariant under translation. To transform vectors using the same transformation matrices that we use to transform points, we set w to zero, thereby removing the effect of the last column. Note that the difference between two homogeneous points produces a w equal to zero. The elimination of the effect of translation makes sense because the difference between two points is a displacement vector, and vectors are defined in terms of their components, not their location.

The `AffineTransform` class in the `java.awt.geom` package defines two-dimensional affine transformations. Instances of this class are constructed as

```
AffineTransform at = new AffineTransform(double m00, double m10,
     double m01, double m11, double m02, double m12);
```

The methods in this class encapsulate most of the matrix arithmetic that is required for two-dimensional visualization. For example, there are methods to calculate a transformation's inverse and to combine transformations using the rules of matrix multiplication. There are also static methods for constructing pure rotations, scalings, and translations that require only one or two parameters.

```
double theta = Math.PI/6;
AffineTransform at = AffineTransform.getRotateInstance(theta);
```

A method such as `getRotateInstance` is known as a *convenience method* because it simplifies a complicated API.

The `AffineTransform` class can transform geometric objects, images, and even text. The following code fragment shows how this class is used to rotate a point and a rectangle.

```
Point2D.Double pt = Point2D.Double(2.0,3.0);
pt = AffineTransform.getRotateInstance(Math.PI/3).transform(pt,null);
Shape shape = new Rectangle2D.Double(50,50,100,150);
shape = AffineTransform.getRotateInstance(Math.PI/3).
        createTransformedShape(shape);
```

The `Affine2DApp` class in Listing 17.1 demonstrates affine transformations by applying them to a rectangle.

Listing 17.1 The `Affine2DApp` class.

```
package org.opensourcephysics.sip.ch17;
import java.awt.*;
import java.awt.geom.*;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

public class Affine2DApp extends AbstractCalculation {
   DisplayFrame frame = new DisplayFrame("2D Affine transformation");
   RectShape rect = new RectShape();

   public void calculate() {
```



```
      // allocate 3 rows but not the row elements
      double[][] matrix = new double[3][];
      // set the first row of the matrix
      matrix[0] = (double[]) control.getObject("row 0");
      // set the second row
      matrix[1] = (double[]) control.getObject("row 1");
      // set the third row
      matrix[2] = (double[]) control.getObject("row 2");
      rect.transform(matrix);
   }

   public void reset() {
      control.clearMessages();
      control.setValue("row 0", new double[] {1, 0, 0});
      control.setValue("row 1", new double[] {0, 1, 0});
      control.setValue("row 2", new double[] {0, 0, 1});
      rect = new RectShape();
      frame.clearDrawables();
      frame.addDrawable(rect);
      calculate();
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new Affine2DApp());
   }

   class RectShape implements Drawable { // inner class
      Shape shape = new Rectangle2D.Double(50, 50, 100, 100);

      public void draw(DrawingPanel panel, Graphics g) {
         Graphics2D g2 = ((Graphics2D) g);
         g2.setPaint(Color.BLUE);
         g2.fill(shape);
         g2.setPaint(Color.RED);
         g2.draw(shape);
      }

      public void transform(double[][] mat) {
         shape = (new AffineTransform(mat[0][0], mat[1][0], mat[0][1],
                  mat[1][1], mat[0][2], mat[1][2])).
                       createTransformedShape(shape);
      }
   }
}
```

Exercise 17.2 Two-dimensional affine transformations

(a) Enter an affine transformation for a 30° clockwise rotation into `Affine2DApp`. About what point does the rectangle rotate? Why?

(b) Add and test a convenience method named `translate` to the `RectShape` class that takes two parameters (d_x, d_y). Add a custom button to invoke this method.

(c) Add and test a convenience method named `rotate` to the `RectShape` class that takes a θ parameter.