```
      int top = control.getInt("ytop");
      // gets rectangle dimensions
      int width = control.getInt("width");
      int height = control.getInt("height");
      Drawable rectangle = new PixelRectangle(left, top, width,
           height);
      frame.addDrawable(rectangle);
      // frame is automatically rendered after Calculate button is
      // clicked
   }

   public void reset() {
      // removes drawables added by the user
      frame.clearDrawables();
      // sets default input values
      control.setValue("xleft", 60);
      control.setValue("ytop", 70);
      control.setValue("width", 100);
      control.setValue("height", 150);
   }

   // creates a calculation control structure using this class
   public static void main(String[] args) {
      CalculationControl.createApp(new DrawingApp());
   }
}
```

Note that multiple rectangles are drawn in the order that they are added to the drawing panel. Rectangles or portions of rectangles may be hidden because they are outside the drawing panel.

Although it is possible to use pixel-based drawing methods to produce visualizations, creating even a simple graph in such an environment would require much tedious programming. The `DrawingPanel` object passed to the `draw` method simplifies this task by defining a system of *world coordinates* that enable us to specify the location and size of various objects in physical units rather than pixels. In the `WorldRectangle` class in Listing 3.3, methods from the `DrawingPanel` class are used to convert pixel coordinates to world coordinates. The range of the world coordinates in the horizontal and vertical directions is defined in the `frame.setPreferredMinMax` method in `DrawingApp`. (This method is not needed if pixel coordinates are used.)

Listing 3.3 `WorldRectangle` illustrates the use of world coordinates.

```
package org.opensourcephysics.sip.ch03;
import java.awt.*;
import org.opensourcephysics.display.*;

public class WorldRectangle implements Drawable {
   double left, top;     // position of rectangle in world coordinates
   double width, height; // size of rectangle in world units

   public WorldRectangle(double left, double top, double width,
                         double height) {
      this.left = left; // location of left edge
      this.top = top;   // location of top edge
      this.width = width;
```



```
      this.height = height;
   }

   public void draw(DrawingPanel panel, Graphics g) {
      // This method implements the Drawable interface
      g.setColor(Color.RED); // sets drawing color to red
      // converts from world to pixel coordinates
      int leftPixels = panel.xToPix(left);
      int topPixels = panel.yToPix(top);
      int widthPixels = (int) (panel.getXPixPerUnit()*width);
      int heightPixels = (int) (panel.getYPixPerUnit()*height);
      // draws rectangle
      g.fillRect(leftPixels, topPixels, widthPixels, heightPixels);
   }
}
```

Exercise 3.4 Simple graphics

(a) Run `DrawingApp` and test how the different inputs change the size and location of the rectangle. Note that the pixel coordinates that are obtained from the control window are not the same as the world coordinates that are displayed.

(b) Read the documentation at `<java.sun.com/reference/api/>` for the `Graphics` class, and modify the `WorldRectangle` class to draw lines, filled ovals, and strings of characters. Also play with different colors.

(c) Modify `DrawingApp` to use the `WorldRectangle` class and repeat part (a). Note that the coordinates that are displayed and the inputs are now consistent.

(d) Define and test a `TextMessage` class to display text messages in a drawing panel using world coordinates to position the text. In the `draw` method, use the syntax `g.drawString("string to draw",x,y)`, where `(x,y)` are the pixel coordinates. ■

Although simple geometric shapes such as circles and rectangles are often all that are needed to visualize many physical models, Java provides a drawing environment based on the Java 2D Application Programming Interface (API) which can render arbitrary geometric shapes, images, and text using composition and matrix-based transformations. We will use a subset of these features to define the `DrawableShape` and `InteractiveShape` classes in the display package of Open Source Physics, which we will introduce in Chapter 9. (See also the *Open Source Physics: A User's Guide with Examples*.)

So far we have created rectangles using two different classes. Each implementation of a `Drawable` rectangle defined a different `draw` method. Notice that in the display frame's definition of `addDrawable` in `DrawingApp`, the argument is specified to be the interface `Drawable` rather than a specific class. Any class that implements `Drawable` can be an argument of `addDrawable`. Without the interface construct, we would need to write an `addDrawable` method for each type of class.

3.4 ■ SPECIFYING THE STATE OF A SYSTEM USING ARRAYS

Imagine writing the code for the numerical solution of the motion of three particles in three dimensions using the Euler–Richardson algorithm. The resulting code would be tedious