```
    System.out.println("x = " + x + "  f(x)" + f.evaluate(x));
}
```

By using the `Function` interface, we can write methods that use this mathematical abstraction. For example, we can program a simple plot as follows:

```
public void plotFunction(Function f, double xmin, double xmax) {
   PlotFrame frame = new PlotFrame("x","y", "Function");
   double n = 100;         // number of points in plot
   double x = xmin, dx = (xmax - xmin)/(n-1);
   for (int i = 0; i < 100; i++) {
      frame.append(0, x, f.evaluate());
      x += dx;
   }
   frame.show();   // display frame on screen
}
```

We can also compute a numerical derivative based on the definition of the derivative found in calculus textbooks.

```
public double derivative(Function f, double x, double dx) {
   return (f.evaluate(x+dx) - f.evaluate(x))/dx;
}
```

This way of approximating a derivative is not optimum, but that is not the point here. (A better approximation is given in Problem 3.8.) The important point is that the interface enables us to define the abstract concept $y = f(x)$ and to write code that uses this abstraction.

Exercise 3.3 Function interface

(a) Define a class that encapsulates the function $f(u) = ae^{-bu^2}$.
(b) Write a test program that plots $f(u)$ with $b = 1$ and $b = 4$. Choose $a = 1$ for simplicity.
(c) Write a test program that plots the derivatives of the functions used in part (b) without using the analytic expression for the derivative. ■

Although interfaces are very useful for developing large scale software projects, you will not need to define interfaces to do the problems in this book. However, you will use several interfaces, including the `Function` interface, that are defined in the Open Source Physics library. We describe two of the more important interfaces in the following sections.

3.3 ■ DRAWING

An interface that we will use often is the `Drawable` interface:

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

public interface Drawable {
   public void draw (DrawingPanel panel, Graphics g);
}
```

Notice that this interface contains only one method, `draw`. Objects that implement this interface are rendered in a `DrawingPanel` after they have been added to a `DisplayFrame`. As we saw in Chapter 2, a `DisplayFrame` consists of components including a title bar, menu, and buttons for minimizing and closing the frame. The `DisplayFrame` contains a `DrawingPanel` on which graphical output will be displayed. The `Graphics` class contains methods for drawing simple geometrical objects such as lines, rectangles, and ovals on the panel. In Listing 3.1 we define a class that draws a rectangle using pixel-based coordinates.

Listing 3.1 `PixelRectangle`.

```
package org.opensourcephysics.sip.ch03;
import java.awt.*; // uses Abstract Window Toolkit
import org.opensourcephysics.display.*;

public class PixelRectangle implements Drawable {
   int left, top;      // position of rectangle in pixels
   int width, height; // size of rectangle in pixels

   PixelRectangle(int left, int top, int width, int 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);                  // set drawing color to red
      g.fillRect(left, top, width, height); // draws rectangle
   }
}
```

In method `draw` we used `fillRect`, a primitive method in the `Graphics` class. This method draws a filled rectangle using pixel coordinates with the origin at the top left corner of the panel.

To use `PixelRectangle`, we instantiate an object and add it to a `DisplayFrame` as shown in Listing 3.2.

Listing 3.2 Listing of `DrawingApp`.

```
package org.opensourcephysics.sip.ch03;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

public class DrawingApp extends AbstractCalculation {
   DisplayFrame frame = new DisplayFrame("x", "y", "Graphics");

   public DrawingApp() {
      frame.setPreferredMinMax(0, 10, 0, 10);
   }

   public void calculate() {
      // gets rectangle location
      int left = control.getInt("xleft");
```