can be written as

$$y(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2. \tag{11.5}$$

What is the value of $y(x)$ at $x = x_1$? The area under the parabola $y(x)$ between x_0 and x_2 can be found by integration and is given by

$$F_0 = \frac{1}{3}(y_0 + 4y_1 + y_2)\Delta x, \tag{11.6}$$

where $\Delta x = x_1 - x_0 = x_2 - x_1$. The total area under all the parabolic segments yields the parabolic approximation for the total area:

$$F_n = \frac{1}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]\Delta x \quad \text{(Simpson's rule)}. \tag{11.7}$$

This approximation is known as *Simpson's rule*, although a more descriptive name would be the *parabolic approximation*. Note that Simpson's rule requires that n be even.

To write a program that implements the rectangular approximation, we must define the function we wish to integrate. Although we could define a new integration class for each function (or change the function in the class and recompile each time), it is convenient to input the name of the function as a string and then *parse* the string so that the function can be evaluated. The `ParsedFunction` class in the numerics package is designed for this task.

```
String str = "cos(x)"; // default string; this string could be an input
Function f;
try {
   f = new ParsedFunction(str);
   } catch (ParserException ex) {
   // recover if str does not represent a valid function
   }
```

Because the `ParsedFunction` is often used with keyboard input and it is common for users to mistype the name of a function, the `ParsedFunction` constructor throws an exception that must be caught.

One way to display a function in a drawing panel is to evaluate the function $f(x)$ at various x values and plot the $(x, f(x))$ data points. Although we could do so using a loop to add a predetermined number of points to a data set, a better way is to use the `FunctionDrawer` class in the display package. The `FunctionDrawer` evaluates a given function at every pixel location within a drawing panel thereby producing a plot with optimum resolution.

```
// drawingPanel created previously
drawingPanel.addDrawable(new FunctionDrawer(f));
```

We next define the class `RectangularApproximation` which computes the area under the curve using the rectangular approximation. This class also displays the rectangles used to compute the area. Note how we have extended the `Dataset` class to produce the visualization.

Listing 11.1 The class `RectangularApproximation` illustrates the nature of the rectangular approximation.

```
package org.opensourcephysics.sip.ch11;
import org.opensourcephysics.display.Dataset;
import org.opensourcephysics.numerics.Function;

public class RectangularApproximation extends Dataset {
   double sum = 0;

   public RectangularApproximation(Function f, double a, double b,
        int n) {
     // transparent red
     setMarkerColor(new java.awt.Color(255, 0, 0, 128));
     setMarkerShape(Dataset.AREA);
     sum = 0;
     double x = a; // lower limit
     double y = f.evaluate(a);
     double dx = (b-a)/n;
     // use methods in Dataset superclass
     append(x, 0); // start on the x axis
     append(x, y); // the top left hand corner of the first rectangle
     while(x<b) {           // b is the upper limit
        x += dx;
        // top right hand corner of current rectangle
        append(x, y);
        sum += y;
        y = f.evaluate(x); // calculate a new y at the new x
        // the top left hand corner of the next rectangle
        append(x, y);
     }
     append(x, 0); // finish on the x axis
     sum *= dx;
   }
}
```

Listing 11.2 `NumericalIntegrationApp` target class.

```
package org.opensourcephysics.sip.ch11;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.numerics.*;

public class NumericalIntegrationApp extends AbstractCalculation {
   PlotFrame plotFrame = new PlotFrame("x", "f(x)",
                              "Numerical integration visualization");

   public void reset() {
      control.setValue("f(x)", "cos(x)");
      control.setValue("lower limit a", 0);
      control.setValue("upper limit b", Math.PI/2);
      control.setValue("number of intervals n", 4);
   }

   public void calculate() {
      String fstring = control.getString("f(x)");
```