To use an ODE solver, we express the wave function rate in terms of the independent variable x:

$$\frac{d\phi}{dx} = \phi' \tag{16.12a}$$

$$\frac{d\phi'}{dx} = -\frac{2m}{\hbar^2}[E - V(x)]\phi \tag{16.12b}$$

$$\frac{dx}{dx} = 1. \tag{16.12c}$$

Because the time-independent Schrödinger equation is a second-order differential equation, two initial conditions must be specified to obtain a solution. For simplicity, we first assume that the wave function is zero at the starting point `xmin`, and the derivative is nonzero. We also assume that the range of values of x is finite and divide this range into intervals of width Δx. We initially consider potential energy functions $V(x)$ such that $V(x) = 0$ for $x < 0$; $V(x)$ changes abruptly at $x = 0$ to V_0, the value of the `stepHeight` parameter. An implementation of the numerical solution of (16.12) is shown in Listing 16.1.

Listing 16.1 The `Schroedinger` class models the one-dimensional time-independent Schrödinger equation.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.numerics.*;

public class Schroedinger implements ODE {
   double energy = 0;
   double[] phi;
   double[] x;
   double xmin, xmax;              // range of values of x
   double[] state = new double[3]; // state = phi, dphi/dx, x
   ODESolver solver = new RK45MultiStep(this);
   double stepHeight = 0;
   int numberOfPoints;

   public void initialize() {
      phi = new double[numberOfPoints];
      x = new double[numberOfPoints];
      double dx = (xmax-xmin)/(numberOfPoints-1);
      solver.setStepSize(dx);
   }

   void solve() {
      // zeros wave function
      for(int i = 0;i<numberOfPoints;i++) {
         phi[i] = 0;
      }
      state[0] = 0;     // initial phi
      state[1] = 1.0;   // nonzero initial dphi/dx
      state[2] = xmin;  // initial value of x
      for(int i = 0;i<numberOfPoints;i++) {
         phi[i] = state[0];           // stores wave function value
         x[i] = state[2];
         solver.step();               // steps Schroedinger equation
         // checks for diverging solution
         if(Math.abs(state[0])>1.0e9) {
            break;                    // leave the loop
         }
      }
   }

   public double[] getState() {
      return state;
   }

   public void getRate(double[] state, double[] rate) {
      rate[0] = state[1];
      rate[1] = 2.0*(-energy+evaluatePotential(state[2]))*state[0];
      rate[2] = 1.0;
   }

   // potential is nonzero for x > 0
   public double evaluatePotential(double x) {
      if(x<0) {
         return 0;
      } else {
         return stepHeight;
      }
   }
}
```


The `solve` method initializes the wave function and position arrays and sets the initial value of $d\phi/dx$ to an arbitrary nonzero value of unity. A loop is then used to compute values of ϕ until the solution diverges or until $x \geq$ `xmax`.

`SchroedingerApp` in Listing 16.2 produces a graphical view of $\phi(x)$. We will use this program in Problem 16.1 to study the behavior of the solution as we vary the height of the potential step.

Listing 16.2 `SchroedingerApp` solves the one-dimensional time-independent Schrödinger equation for a given energy.

```
package org.opensourcephysics.sip.ch16;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;

public class SchroedingerApp extends AbstractCalculation {
   PlotFrame frame = new PlotFrame("x", "phi", "Wave function");
   Schroedinger schroedinger = new Schroedinger();

   public SchroedingerApp() {
      frame.setConnected(0, true);
      frame.setMarkerShape(0, Dataset.NO_MARKER);
   }

   public void calculate() {
      schroedinger.xmin = control.getDouble("xmin");
      schroedinger.xmax = control.getDouble("xmax");
      schroedinger.stepHeight =
           control.getDouble("step height at x = 0");
      schroedinger.numberOfPoints = control.getInt("number of points");
```