The program also estimates the ground state wave function by accumulating the spatial distribution of the walkers at discrete intervals of position. The input parameters are the desired number of walkers N_0, the number of position intervals to accumulate data for the ground state wave function `numberOfBins`, and the step size `ds`. We also use `ds` for the interval size in the wave function computation. The program computes the current number of walkers, the estimate of the ground state energy, and the value of V_{ref}. The unnormalized ground state wave function is also plotted.

Listing 16.10 The `QMWalk` class calculates the ground state of the simple harmonic oscillator using the random walk Monte Carlo algorithm.

```
package org.opensourcephysics.sip.ch16;
public class QMWalk {
   int numberOfBins = 1000; // for wave function
   double[] x;              // positions of walkers
   double[] phi0;           // estimate of ground state wave function
   double[] xv;             // x values for computing phi0
   int N0;                  // desired number of walkers
   int N;                   // actual number of walkers
   double ds;               // step size
   double dt;               // time interval
   double vave = 0;         // mean potential
   double vref = 0;         // reference potential
   double eAccum = 0;       // accumulation of energy values
   double xmin;             // minimum x
   int mcs;

   public void initialize() {
      N0 = N;
      x = new double[2*numberOfBins];
      phi0 = new double[numberOfBins];
      xv = new double[numberOfBins];
      // minimum location for computing phi0
      xmin = -ds*numberOfBins/2.0;
      double binEdge = xmin;
      for(int i = 0;i<numberOfBins;i++) {
         xv[i] = binEdge;
         binEdge += ds;
      }
      double initialWidth = 1; // initial width for location of walkers
      for(int i = 0;i<N;i++) {
         // initial location of walkers
         x[i] = (2*Math.random()-1)*initialWidth;
         vref += potential(x[i]);
      }
      vave = 0;
      vref = 0;
      eAccum = 0;
      mcs = 0;
      dt = ds*ds;
   }

   void walk() {
      double vsum = 0;
      for(int i = N-1;i>=0;i--) {
         if(Math.random()<0.5) {
```



```
            x[i] += ds;
         } else {
            x[i] -= ds;
         }
         double pot = potential(x[i]);
         double dv = pot-vref;
         vsum += pot;
         if(dv<0) {                  // decide to add or delete walker
            if(N==0||(Math.random()<-dv*dt)&&(N<x.length)) {
               x[N] = x[i];          // new walker at the current location
               vsum += pot;          // add energy of new walker
               N++;
            }
         } else {
            if((Math.random()<dv*dt)&&(N>0)) {
               N--;
               // relabel last walker to deleted walker index
               x[i] = x[N];
               vsum -= pot;          // subtract energy of deleted walker
            }
         }
      }
      vave = (N==0) ? 0: vsum/N; // if no walkers poential = 0
      vref = vave-(N-N0)/(N0*dt);
      mcs++;
   }

   void doMCS() {
      walk();
      eAccum += vave;
      for(int i = 0;i<N;i++) {
         int bin = (int) Math.floor((x[i]-xmin)/ds); // bin index
         if(bin>=0&&bin<numberOfBins) {
            phi0[bin]++;
         }
      }
   }

   void resetData() {
      for(int i = 0;i<numberOfBins;i++) {
         phi0[i] = 0;
      }
      eAccum = 0;
      mcs = 0;
   }

   public double potential(double x) {
      return 0.5*x*x;
   }
}
```

Listing 16.11 The `QMWalkApp` class computes and displays the result of a random walk Monte Carlo calculation.

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