```
      frame.setInteractiveMouseHandler(this);
   }

   public void calculate() { \\ creates new charge
      double x = control.getDouble("x");
      double y = control.getDouble("y");
      double q = control.getDouble("q");
      Charge charge = new Charge(x, y, q);
      frame.addDrawable(charge);
      calculateField();
   }

   public void reset() {
      control.println("charges can be dragged.");
      frame.clearDrawables(); // removes all charges
      control.setValue("x", 0);
      control.setValue("y", 0);
      control.setValue("q", 1);
      calculateField();
   }

   void calculateField() {
      for(int ix = 0;ix<n;ix++) {
         for(int iy = 0;iy<n;iy++) {
            eField[0][ix][iy] = eField[1][ix][iy] = 0; // zeros field
         }
      }
      // the charges in the frame
      List chargeList = frame.getDrawables(Charge.class);
      Iterator it = chargeList.iterator();
      while(it.hasNext()) {
         Charge charge = (Charge) it.next();
         double xs = charge.getX(), ys = charge.getY();
         for(int ix = 0;ix<n;ix++) {
            double x = frame.indexToX(ix);
            // distance of charge to gridpoint
            double dx = (x-xs);
            for(int iy = 0;iy<n;iy++) {
               double y = frame.indexToY(iy);
               double dy = (y-ys);          // charge to gridpoint
               double r2 = dx*dx+dy*dy;     // distance squared
               double r3 = Math.sqrt(r2)*r2; // distance cubed
               if(r3>0) {
                  eField[0][ix][iy] += charge.q*dx/r3;
                  eField[1][ix][iy] += charge.q*dy/r3;
               }
            }
         }
      }
      frame.setAll(eField);
   }

   public void handleMouseAction(InteractivePanel panel,
        MouseEvent evt) {
      panel.handleMouseAction(panel, evt); // panel moves the charge
      if(panel.getMouseAction()==InteractivePanel.MOUSE_DRAGGED) {
```

```
         // remove this line if user interface is sluggish
         calculateField();
         panel.repaint();
      }
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new ElectricFieldApp());
   }
}
```

To make the program interactive, the `ElectricFieldApp` class implements the `InteractiveMouseHandler` to process mouse events when a charge is dragged. (See Section 5.7 for a discussion of interactive panels and interactive mouse handlers.) The class registers its interest in handling these events using the `setInteractiveMouseHandler` method. The handler passes the event to the panel to move the charge and then recalculates the field. Note that the `Charge` class in Listing 10.2 inherits from the `InteractiveCircle` class.[2]

Listing 10.2 The `Charge` class extends the `InteracticeCircle` class and adds the charge property.

```
package org.opensourcephysics.sip.ch10;
import java.awt.Color;
import org.opensourcephysics.display.InteractiveCircle;

public class Charge extends InteractiveCircle {
   double q = 0;

   public double getQ() {
      return q;
   }

   public Charge(double x, double y, double q) {
      super(x, y);
      this.q = q;
      if(q>0) {
         color = Color.red;
      } else {
         color = Color.blue;
      }
   }
}
```

Problem 10.1 Motion of a charged particle in an electric field

(a) Test `ElectricFieldApp` by adding one charge at a time at various locations. Do the electric field patterns look reasonable? For example, does the electric field point away from positive charges and toward negative charges? How well is the magnitude of the electric field represented?

(b) Modify `ElectricFieldApp` so that it uses an `AbstractAnimation` to compute the motion of a test particle of mass m and charge q in the presence of the electric field

[2]Dragging may become sluggish if too many computations are performed within the mouse action method.