created by a fixed distribution of point charges. That is, create a drawable test charge that implements the `ODE` interface and add it to the vector field frame. Use the same approach that was used for the trajectory problems in Chapter 5. The acceleration of the charge is given by $q\mathbf{E}/m$, where $\mathbf{E}$ is the electric field due to the fixed point charges. Use a higher-order algorithm to advance the position and velocity of the particle. (Ignore the effects of radiation due to accelerating charges.)

(c) Assume that $\mathbf{E}$ is due to a charge `q(1)` $= 1.5$ fixed at the origin. Simulate the motion of a charged particle of mass $m = 0.1$ and charge $q = 1$ initially at $x = 1$, $y = 0$. Consider the following initial conditions for its velocity: $v_x = 0$, $v_y = 0$; $v_x = 1$, $v_y = 0$; $v_x = 0$, $v_y = 1$; and $v_x = -1$, $v_y = 0$. Is the trajectory of the particle tangent to the field vectors? Explain.

(d) Assume that the electric field is due to two fixed point charges: `q(1)` $= 1$ at `x(1)` $= 2$, `y(1)` $= 0$ and `q(2)` $= -1$ at `x(2)` $= -2$, `y(2)` $= 0$. Place a charged particle of unit mass and unit positive charge at $x = 0.05$, $y = 0$. What do you expect the motion of this charge to be? Do the simulation and determine the qualitative nature of the motion.

*(e) Consider the motion of a charged particle in the vicinity of the electric dipole defined in part (d). Choose the initial position to be five times the separation of the charges in the dipole. Do you find any bound orbits? Do you find any closed orbits or do all orbits show some precession? ■

10.3 ■ ELECTRIC FIELD LINES

Another way of visualizing the electric field is to draw *electric field lines*. The properties of these lines are as follows:

1. An electric field line is a directed line whose tangent at every position is parallel to the electric field at that position.
2. The lines are smooth and continuous except at singularities such as point charges. (It makes no sense to talk about the electric field *at* a point charge.)
3. The density of lines at any point in space is proportional to the magnitude of the field at that point. This property implies that the total number of electric field lines from a point charge is proportional to the magnitude of that charge. The value of the proportionality constant is chosen to provide the clearest pictorial representation of the field. The drawing of field lines is art plus science.

The `FieldLineApp` program draws electric field lines in two dimensions. The program makes extensive use of the `FieldLine` class which implements the following algorithm:

1. Begin at a point (x, y) and compute the components E_x and E_y of the electric field vector $\mathbf{E}$ using (10.1).

2. Draw a small line segment of size $\Delta s = |\Delta \mathbf{s}|$ tangent to $\mathbf{E}$ at that point. The components of the line segment are given by

$$\Delta x = \Delta s \frac{E_x}{|\mathbf{E}|} \quad \text{and} \quad \Delta y = \Delta s \frac{E_y}{|\mathbf{E}|}. \tag{10.4}$$

3. Iterate the process beginning at the new point $(x + \Delta x, y + \Delta y)$. Continue until the field line approaches a point charge singularity or escapes toward infinity.

This field line algorithm is equivalent to solving the following differential equations:

$$\frac{dx}{ds} = \frac{E_x}{|\mathbf{E}|} \tag{10.5a}$$

$$\frac{dy}{ds} = \frac{E_y}{|\mathbf{E}|}. \tag{10.5b}$$

Because a field line extends in both directions from the algorithm's starting point, the computation must be repeated in the $(-E_x/|\mathbf{E}|, -E_y/|\mathbf{E}|)$ direction to obtain a complete visualization of the field line. Note that this algorithm draws a correct field line but does not draw a collection of field lines with a density proportional to the field intensity.

To draw the field lines, a computation starts when a user double clicks in the panel and ends when the field line approaches a point charge or when the magnitude of the field becomes too small. Although we can easily describe these stopping conditions, we do not know how long the computation will take, and we might want to compute multiple field lines simultaneously. An elegant way to do this computation is to use threads.

As we discussed in Section 2.6, Java programs can have multiple threads to separate and organize related tasks. A thread is an *independent* task within a single program that shares the program's data with other threads.[3] In the following example, we create a thread to compute the solution of the differential equation for an electric field line. It is natural to use threads in this context because the drawing of a field line involves starting the field line, drawing each piece of the field line, and then stopping the calculation when some stopping condition is met. The computation begins when the `FieldLine` object is created and ends when the stopping condition is satisfied.

A thread executes statements within an object, such as `FieldLine`, that implements the `Runnable` interface. This interface consists of a single method, the `run` method, and the thread executes the code within this method. The run method is not invoked directly, but is invoked automatically by the thread after the thread is started. When the run method exits, the thread that invoked the run method stops executing and is said to die. After a thread dies, it cannot be restarted. Another thread must be created if we wish to invoke the run method a second time.

We build a `FieldLine` class by subclassing `Thread` and adding the necessary drawing and differential equation capabilities using the `Drawable` and `ODE` interfaces, respectively. This class is shown in Listing 10.3.

[3] *Open Source Physics: A User's Guide with Examples* describes simulation threads in more detail.