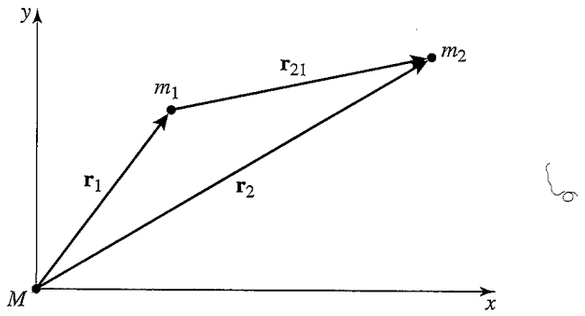

Figure 5.6 The coordinate system used in (5.25). Planets of mass m_1 and m_2 orbit a sun of mass M.

by m_2 and to write the equations of motion as

$$\frac{d^2\mathbf{r}_1}{dt^2} = -\frac{GM}{{r_1}^3}\mathbf{r}_1 + \frac{Gm_2}{{r_{21}}^3}\mathbf{r}_{21} \tag{5.26a}$$

$$\frac{d^2\mathbf{r}_2}{dt^2} = -\frac{GM}{{r_2}^3}\mathbf{r}_2 - \frac{Gm_1}{{r_{21}}^3}\mathbf{r}_{21}. \tag{5.26b}$$

A numerical solution of (5.26) can be obtained by the straightforward extension of the `Planet` class as shown in Listing 5.5. To simplify the drawing of the particle trajectories, the `Planet2` class defines an *inner class* `Mass`, which extends `Circle` and contains a `Trail`. Whenever a planet moves, a point is added to the trail so that its location and path are shown on the plot. Inner classes are an organizational convenience that save us the trouble of having to create another file, which in this case would be named `Mass.java`. When we compile the `Planet2` class, we will produce a bytecode file named `Planet2$Mass.class` in addition to the file `Planet2.class`. Inner classes are most effective as short helper classes which work in conjuction with the containing class, because they have access to all the data (including private variables) in the containing class.

Listing 5.5 A class that implements the rate equation for two interacting planets acted on by an inverse-square law force.

```
package org.opensourcephysics.sip.ch05;
import java.awt.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;

public class Planet2 implements Drawable, ODE {
   // GM in units of (AU)^3/(yr)^2
   final static double GM = 4*Math.PI*Math.PI;
   final static double GM1 = 0.04*GM;
   final static double GM2 = 0.001*GM;
   double[] state = new double[9];
   ODESolver odeSolver = new RK45MultiStep(this);
   Mass mass1 = new Mass(), mass2 = new Mass();

   public void doStep() {
      odeSolver.step();
      mass1.setXY(state[0], state[2]);
      mass2.setXY(state[4], state[6]);
   }
```



```
   public void draw(DrawingPanel panel, Graphics g) {
      mass1.draw(panel, g);
      mass2.draw(panel, g);
   }

   void initialize(double[] initState) {
      System.arraycopy(initState, 0, state, 0, initState.length);
      mass1.clear(); // clears data from the old trail
      mass2.clear();
      mass1.setXY(state[0], state[2]);
      mass2.setXY(state[4], state[6]);
   }

   public void getRate(double[] state, double[] rate) {
      // state[]: x1, vx1, y1, vy1, x2, vx2, y2, vy2, t
      double r1Squared = state[0]*state[0]+state[2]*state[2];
      double r1Cubed = r1Squared*Math.sqrt(r1Squared);
      double r2Squared = state[4]*state[4]+state[6]*state[6];
      double r2Cubed = r2Squared*Math.sqrt(r2Squared);
      double dx = state[4]-state[0];      // x12 separation
      double dy = state[6]-state[2];      // y12 separation
      double dr2 = dx*dx+dy*dy;           // r12 squared
      double dr3 = Math.sqrt(dr2)*dr2;    // r12 cubed
      rate[0] = state[1];                 // x1 rate
      rate[2] = state[3];                 // y1 rate
      rate[4] = state[5];                 // x2 rate
      rate[6] = state[7];                 // y2 rate
      rate[1] = ((-GM*state[0])/r1Cubed)+((GM1*dx)/dr3); // vx1 rate
      rate[3] = ((-GM*state[2])/r1Cubed)+((GM1*dy)/dr3); // vy1 rate
      rate[5] = ((-GM*state[4])/r2Cubed)-((GM2*dx)/dr3); // vx2 rate
      rate[7] = ((-GM*state[6])/r2Cubed)-((GM2*dy)/dr3); // vy2 rate
      rate[8] = 1;                        // time rate
   }

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

   class Mass extends Circle {
      Trail trail = new Trail();

      public void draw(DrawingPanel panel, Graphics g) {
         trail.draw(panel, g);
         super.draw(panel, g);
      }

      void clear() {
         trail.clear();
      }

      public void setXY(double x, double y) {
         super.setXY(x, y);
         trail.addPoint(x, y);
      }
   }
}
```