```
/*************************************************************************
 *  COS226 - Algorithms 4 booksite, Section 6.1 applications
 *
 *  Compilation:  javac CollisionSystem.java
 *  Execution:    java CollisionSystem N               (N random particles)
 *                java CollisionSystem < input.txt     (from a file)
 *
 *  Creates N random particles and simulates their motion according
 *  to the laws of elastic collisions.
 *
   Week 4 - handout questions

   1 - Where is "pq" declared?

   2 - Where is "pq" created?

   3 - Look at the method predict(). It inserts into "pq". How many
       different events are created in one call? What different kinds are they?

   4 - Now look at the method simulate(). How many events are initially
       put on "pq" (before the while loop begins)?

   5 - When will simulate() end?

   6 - One iteration of the while loop in simulate() processes how many events?

   7 - When will an event taken off "pq" cause no collision to happen?

 *************************************************************************/

import java.awt.Color;

public class CollisionSystem {

    private MinPQ<Event> pq;        // the priority queue
    private double t  = 0.0;        // simulation clock time
    private double Hz = 0.5;        // number of redraw events per clock tick
    private Particle[] particles;   // the array of particles

    // create a new collision system with the given set of particles
    public CollisionSystem(Particle[] particles) {
        this.particles = particles;
    }

    // updates priority queue with all new events for particle a
    private void predict(Particle a, double limit) {
        if (a == null) return;

        // particle-particle collisions
        for(int i = 0; i < particles.length; i++) {
            double dt = a.timeToHit(particles[i]);
            if(t + dt <= limit)
                pq.insert(new Event(t + dt, a, particles[i]));
        }

        // particle-wall collisions
        double dtX = a.timeToHitVerticalWall();
        double dtY = a.timeToHitHorizontalWall();
        if (t + dtX <= limit) pq.insert(new Event(t + dtX, a, null));
        if (t + dtY <= limit) pq.insert(new Event(t + dtY, null, a));
    }
```



```
    // redraw all particles
    private void redraw(double limit) {
        StdDraw.clear();
        for(int i = 0; i < particles.length; i++) {
            particles[i].draw();
        }
        StdDraw.show(20);
        if (t < limit) {
            pq.insert(new Event(t + 1.0 / Hz, null, null));
        }
    }


   /*************************************************************************
    *  Event based simulation for limit seconds
    *************************************************************************/
    public void simulate(double limit) {

        // initialize PQ with collision events and redraw event
        pq = new MinPQ<Event>();
        for(int i = 0; i < particles.length; i++) {
            predict(particles[i], limit);
        }
        pq.insert(new Event(0, null, null));        // redraw event


        // the main event-driven simulation loop
        while(!pq.isEmpty()) {

            // get impending event, discard if invalidated
            Event e = pq.delMin();
            if(!e.isValid()) continue;
            Particle a = e.a;
            Particle b = e.b;

            // physical collision, update positions, and then simulation clock
            for(int i = 0; i < particles.length; i++)
                particles[i].move(e.time - t);
            t = e.time;

            // process event
            if      (a != null && b != null)
                a.bounceOff(b);              // particle-particle collision
            else if (a != null && b == null)
                a.bounceOffVerticalWall();   // particle-wall collision
            else if (a == null && b != null)
                b.bounceOffHorizontalWall(); // particle-wall collision
            else if (a == null && b == null)
                redraw(limit);               // redraw event

            // update the priority queue with new collisions involving a or b
            predict(a, limit);
            predict(b, limit);
        }
    }
```

```
    /*************************************************************************
     *  An event during a particle collision simulation. Each event contains
     *  the time at which it will occur (assuming no supervening actions)
     *  and the particles a and b involved.
     *
     *    -  a and b both null:      redraw event
     *    -  a null, b not null:     collision with vertical wall
     *    -  a not null, b null:     collision with horizontal wall
     *    -  a and b both not null:  binary collision between a and b
     *
     *************************************************************************/
    private class Event implements Comparable<Event> {
        private final double time;         // when event is scheduled to occur
        private final Particle a, b;       // particles involved in event,
                                           // possibly null
        private final int countA, countB;  // collision counts at event creation


    // create a new event to occur at time t involving a and b
    public Event(double t, Particle a, Particle b) {
        this.time = t;
        this.a    = a;
        this.b    = b;
        if (a != null) countA = a.count();
        else           countA = -1;
        if (b != null) countB = b.count();
        else           countB = -1;
    }

    // compare times when two events will occur
    public int compareTo(Event that) {
        if      (this.time < that.time) return -1;
        else if (this.time > that.time) return +1;
        else                            return  0;
    }

    // has any collision occurred between when event was created and now?
    public boolean isValid() {
        if (a != null && a.count() != countA) return false;
        if (b != null && b.count() != countB) return false;
        return true;
    }

}
```



```
   /*************************************************************************
    *  Sample client
    *************************************************************************/
    public static void main(String[] args) {

        // remove the border
        StdDraw.setXscale(1.0/22.0, 21.0/22.0);
        StdDraw.setYscale(1.0/22.0, 21.0/22.0);

        // turn on animation mode
        StdDraw.show(0);

        // the array of particles
        Particle[] particles;

        // create N random particles
        if (args.length == 1) {
            int N = Integer.parseInt(args[0]);
            particles = new Particle[N];
            for(int i = 0; i < N; i++) particles[i] = new Particle();
        }

        // or read from standard input
        else {
            int N = StdIn.readInt();
            particles = new Particle[N];
            for(int i = 0; i < N; i++) {
                double rx     = StdIn.readDouble();
                double ry     = StdIn.readDouble();
                double vx     = StdIn.readDouble();
                double vy     = StdIn.readDouble();
                double radius = StdIn.readDouble();
                double mass   = StdIn.readDouble();
                int r         = StdIn.readInt();
                int g         = StdIn.readInt();
                int b         = StdIn.readInt();
                Color color   = new Color(r, g, b);
                particles[i] = new Particle(rx, ry, vx, vy, radius,
                                            mass, color);
            }
        }

        // create collision system and simulate
        CollisionSystem system = new CollisionSystem(particles);
        system.simulate(10000);
    }

}
```