```
/**
 * A basic physics simulation using a class to define a
 * Ball object that is under the influence of gravity, and
 * that can collide with other objects.
 *
 * This code is also used as a demonstration of proper coding
 * style.  It shows how to comment the program, variables,
 * classes, and functions.  Additionally, it showcases good
 * variable names, indentation, and whitespace.
 *
 * @author Jason Healy
 */

/** A collection of all the balls in the simulation */
Ball[] balls = new Ball[10];

/** Processing setup function: runs only when the program begins */
void setup() {
  size(800, 600);
  // populate each space in the array with a randomly-generated
  // ball
  for (int i = 0; i < balls.length; i=i+1) {
    balls[i] = new Ball(random(width), random(height));
  }
}

/** Processing draw function: run automatically for each frame */
void draw() {
  background(0);
  // have each ball render itself to the screen
  for (int i = 0; i < balls.length; i=i+1) {
    balls[i].render();
  }
}

/** Processing mouse event handler
 * When the user presses the mouse, impart some additional
 * velocity to each of the balls so they jump up on the screen
 */
void mousePressed() {
  background(0);
  for (int i = 0; i < balls.length; i=i+1) {
    balls[i].pop();
  }
}


/**
 * A Ball is represented by a circle on the screen.  Every Ball
 * moves with its own velocity, and is capable of being influenced
 * by gravity and by colliding with the sides of the screen and
 * other Balls.
*/
class Ball {
  /** Radius of each ball */
  int radius = 25;

  /** Array of possible colors (alternated when it collides) */
  color[] c = {
    color(255, 0, 0, 255), color(0, 255, 0, 255), color(0, 0, 255, 255)
  };

  /** X location of the Ball on the screen */
  private float x;
  /** Y location of the Ball on the screen */
  private float y;

  /** X velocity */
  private float vx;
  /** Y velocity */
  private float vy;

  /** Timestamp (millis) since the last update; used to
   *  calculate accelerations and velocities */
  private int last;

  /** Number to track which color to use for the Ball;
   *  it is an index into the colors array "c" */
  private int colour = 0;

  /** Tracks whether this Ball has been pressed by the mouse */
  private boolean pressed = false;


  /** Constructor.  Initializes Ball at the given coordinates
   *  and assigns a random initial velocity. */
  Ball(float x, float y) {
    this.x = x;
    this.y = y;
    vx = random(-width/10, width/10)*10;
    vy = random(-height/10, height/10)*10;
    last = millis();
  }

  /** Updates the Ball based on physics engine, and then
   *  renders the Ball to the screen. */
  void render() {
    // determine how long it’s been since the last update
    int now = millis();
    float elapsed = (now - last) / 1000.0;
    last = now;

    // change position based on velocity and elapsed time
    x += elapsed * vx;
    y += elapsed * vy;

    // change y velocity based on gravity
    vy += elapsed * 1000;

    // bounce off the left and right sides of the screen
    if ( (x < radius && vx < 0) || (x > width-radius && vx > 0) ) {
      vx *= -0.9;
    }
    // bounce off the top and bottom sides of the screen
    if ( (y < radius && vy < 0) || (y > height-radius && vy > 0) ) {
      vy *= -0.9;
    }

    // try to collide with every single other Ball in the program

    // start with the first Ball in the array
    int i = 0;
    // loop through all the Balls "before" this one in the array
    // and ignore them
    while (balls[i] != this) {
      i=i+1;
    }
    // now skip "this" Ball (so we don’t collide with ourself)
    i=i+1;

    // finally, actually try to collide with all the remaining
    // Balls in the array.  Because we skipped the ones "before"
    // us in the array (and all the others will do the same),
    // only one collision between any two Ball objects is ever
    // attempted.
    while (i < balls.length) {
      collide(balls[i]);
      i=i+1;
    }

    // now that all the physics updates are done, render the
    // ball on screen
    noStroke();
    fill(c[colour]);
    ellipse(x, y, radius*2, radius*2);
  }

  /** Rotate to the next color in the array. */
  void bump() {
    colour = (colour+1)%c.length;
  }

  /** Give a random "kick" in y velocity to this Ball. */
  void pop() {
    vy = random(-height, 0)*2;
  }

  /** Attempt to collide with another Ball object (passed
   *  as a parameter). */
  void collide(Ball o) {
    // see how far apart the centers of both Ball objects are
    float overlap = 2*radius - dist(x, y, o.x, o.y);

    if (overlap < 0) {
      // no collision if they aren't touching...
      // just return without doing anything else
      return;
    }

    // If we didn't return above, there must be some overlap

    // get the angle that this ball collides with the other
    float theta = atan2(o.y-y, o.x-x);

    // generate a force (velocity change) along the axis
    // that connects the two balls, pushing away from the
    // point of contact
    // 7-10 is a fudge factor to make the bounces look good
    vx -= cos(theta) * radius * 10;
    vy -= sin(theta) * radius * 10;
    o.vx += cos(theta) * radius * 10;
    o.vy += sin(theta) * radius * 10;

    // move the balls apart so they're no longer touching
    x -= cos(theta) * overlap / 2;
    y -= sin(theta) * overlap / 2;
    o.x += cos(theta) * overlap / 2;
    o.y += sin(theta) * overlap / 2;

    // tell each Ball to "bump" so its color changes
    bump();
    o.bump();
  }
}
```