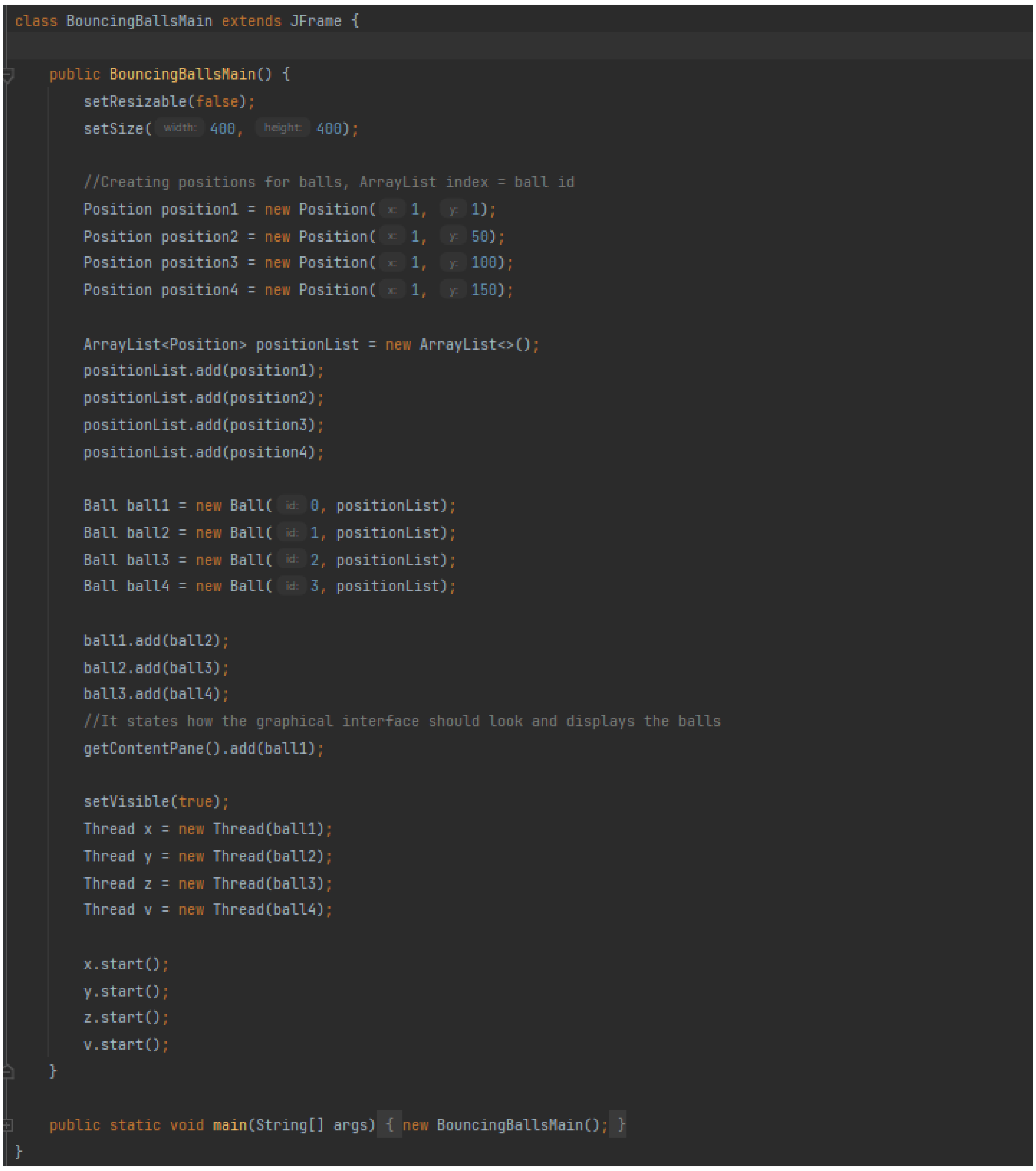
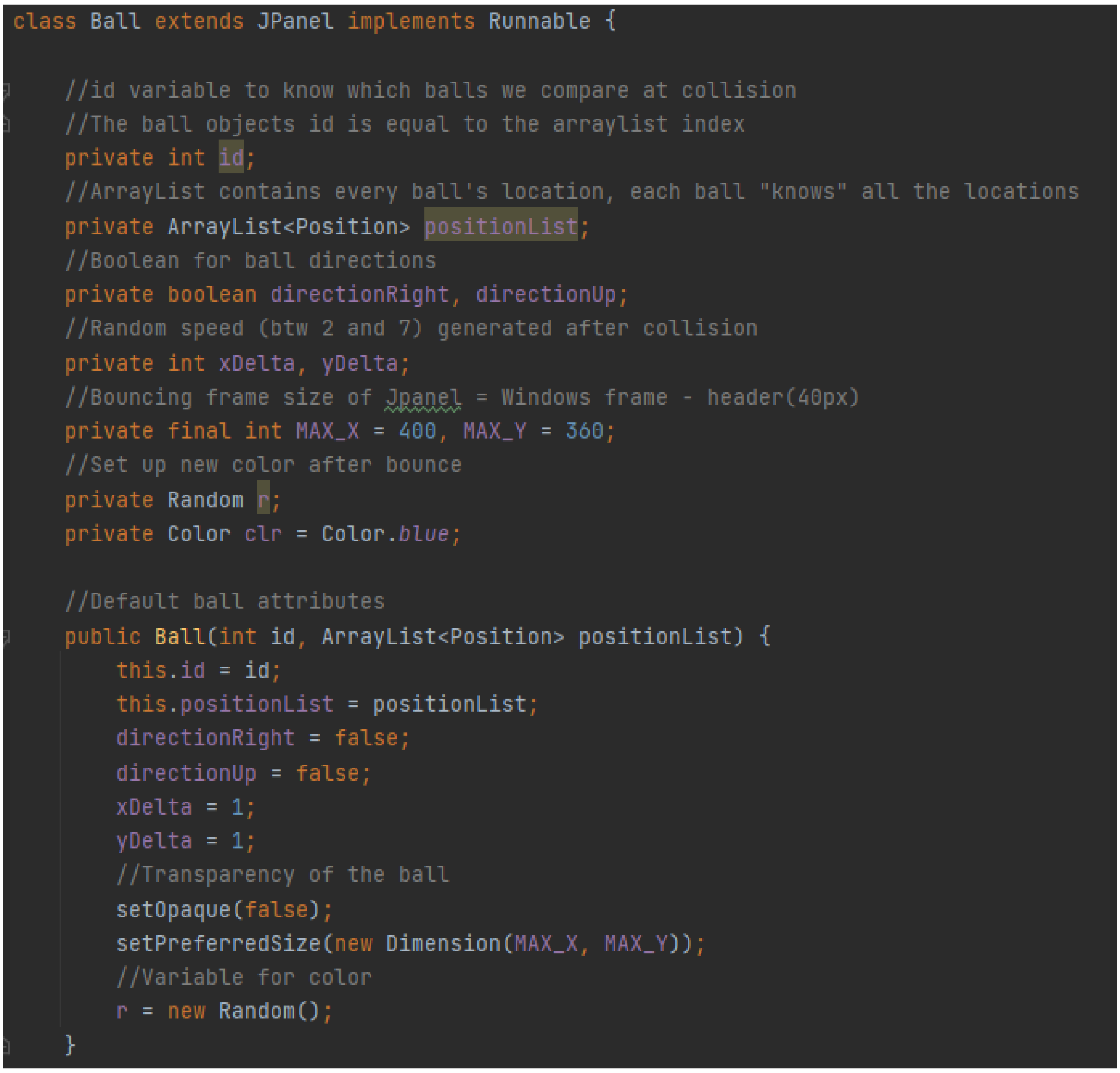
Our bouncing ball project uses Java Swing as its GUI. Extends JFrame and JPanel. Each ball is run as its own thread.



```java
class BouncingBallsMain extends JFrame {

    public BouncingBallsMain() {
        setResizable(false);
        setSize( width: 400,  height: 400);

        //Creating positions for balls, ArrayList index = ball id
        Position position1 = new Position( x: 1,  y: 1);
        Position position2 = new Position( x: 1,  y: 50);
        Position position3 = new Position( x: 1,  y: 100);
        Position position4 = new Position( x: 1,  y: 150);

        ArrayList<Position> positionList = new ArrayList<>();
        positionList.add(position1);
        positionList.add(position2);
        positionList.add(position3);
        positionList.add(position4);

        Ball ball1 = new Ball( id: 0, positionList);
        Ball ball2 = new Ball( id: 1, positionList);
        Ball ball3 = new Ball( id: 2, positionList);
        Ball ball4 = new Ball( id: 3, positionList);

        ball1.add(ball2);
        ball2.add(ball3);
        ball3.add(ball4);
        //It states how the graphical interface should look and displays the balls
        getContentPane().add(ball1);

        setVisible(true);
        Thread x = new Thread(ball1);
        Thread y = new Thread(ball2);
        Thread z = new Thread(ball3);
        Thread v = new Thread(ball4);

        x.start();
        y.start();
        z.start();
        v.start();
    }

    public static void main(String[] args) { new BouncingBallsMain(); }
}
```

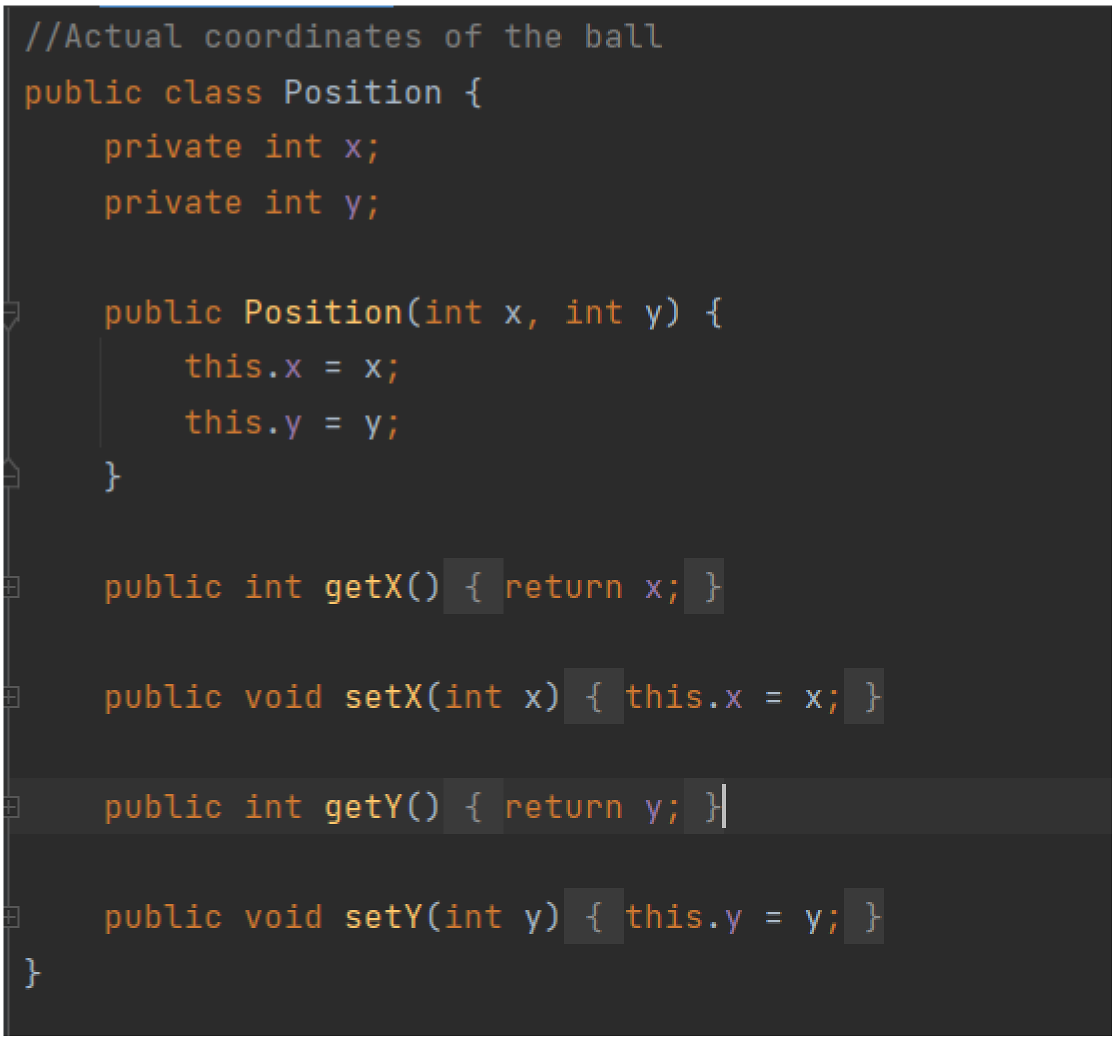
Our BouncingBallMain class is responsible for setting up the JFrame, initializing the objects needed for the application (ball, location), creating its own thread for each ball in use, and running them.

The attributes of the ball:



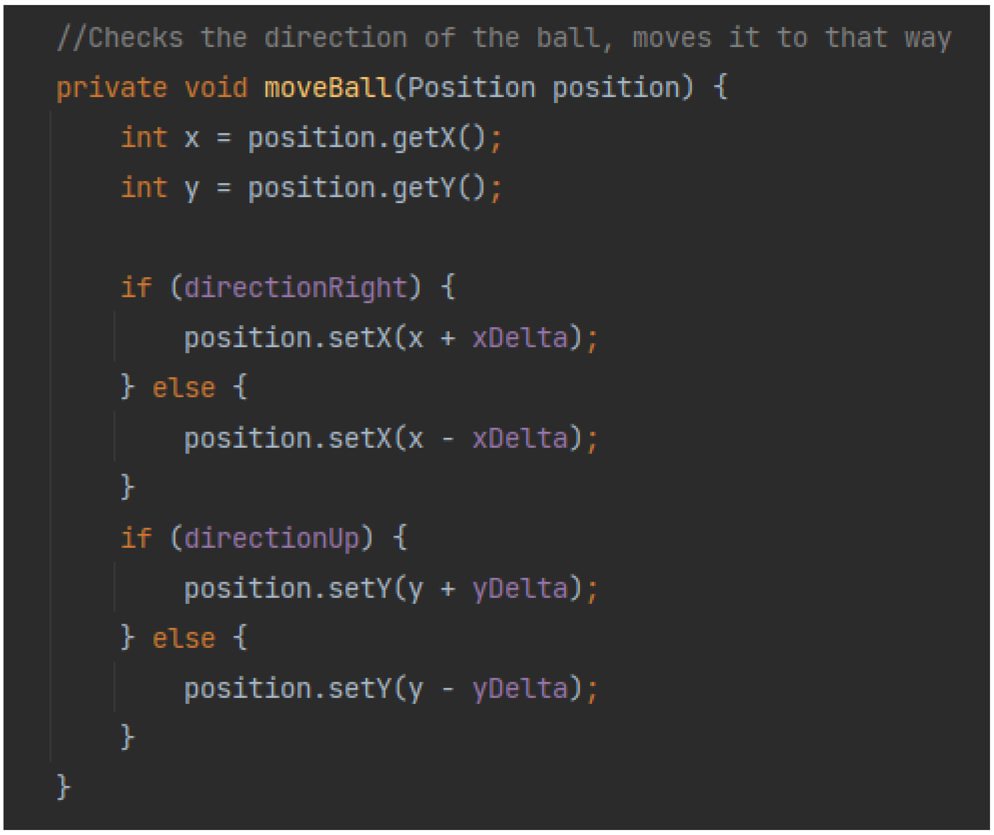
```java
class Ball extends JPanel implements Runnable {

    //id variable to know which balls we compare at collision
    //The ball objects id is equal to the arraylist index
    private int id;
    //ArrayList contains every ball's location, each ball "knows" all the locations
    private ArrayList<Position> positionList;
    //Boolean for ball directions
    private boolean directionRight, directionUp;
    //Random speed (btw 2 and 7) generated after collision
    private int xDelta, yDelta;
    //Bouncing frame size of Jpanel = Windows frame - header(40px)
    private final int MAX_X = 400, MAX_Y = 360;
    //Set up new color after bounce
    private Random r;
    private Color clr = Color.blue;

    //Default ball attributes
    public Ball(int id, ArrayList<Position> positionList) {
        this.id = id;
        this.positionList = positionList;
        directionRight = false;
        directionUp = false;
        xDelta = 1;
        yDelta = 1;
        //Transparency of the ball
        setOpaque(false);
        setPreferredSize(new Dimension(MAX_X, MAX_Y));
        //Variable for color
        r = new Random();
    }
```

Each ball object contains an **id** and a **positionList** that stores every position. This way, each ball “knows” the locations of all the other balls. Since the ball’s id is equal to the position’s index in the list, each ball also knows which position belongs to them, so the ball doesn’t check a collision with itself. The other variables are used to determine the direction, speed, and color of the ball, as well as the size of the bouncing area.



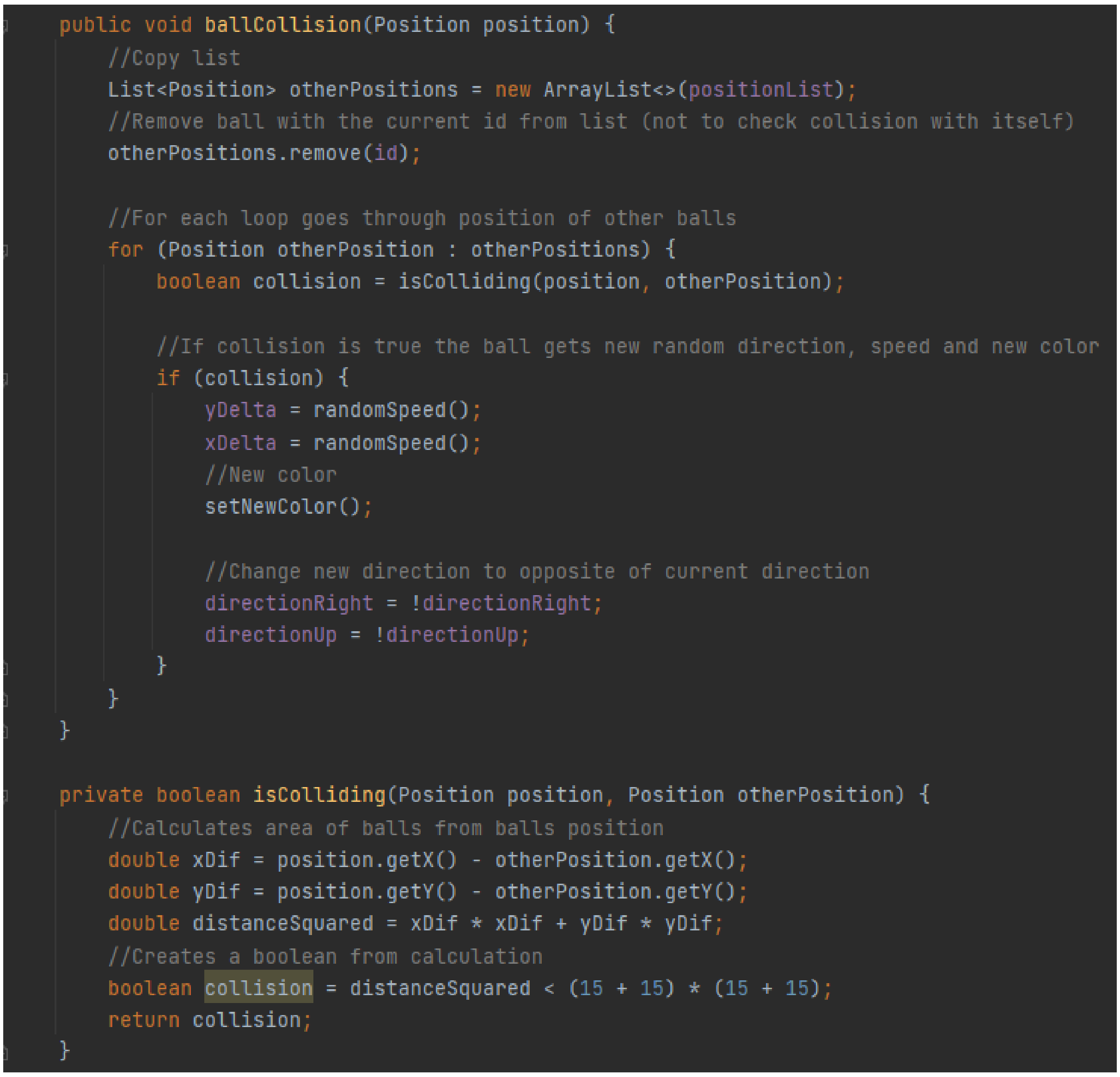
```java
//Actual coordinates of the ball
public class Position {
    private int x;
    private int y;

    public Position(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }

    public void setX(int x) { this.x = x; }

    public int getY() { return y; }

    public void setY(int y) { this.y = y; }
}
```

The ball’s position (int x, y) is stored in its own class called Position. Each ball contains a list with all the positions, used to later check the collisions between them.



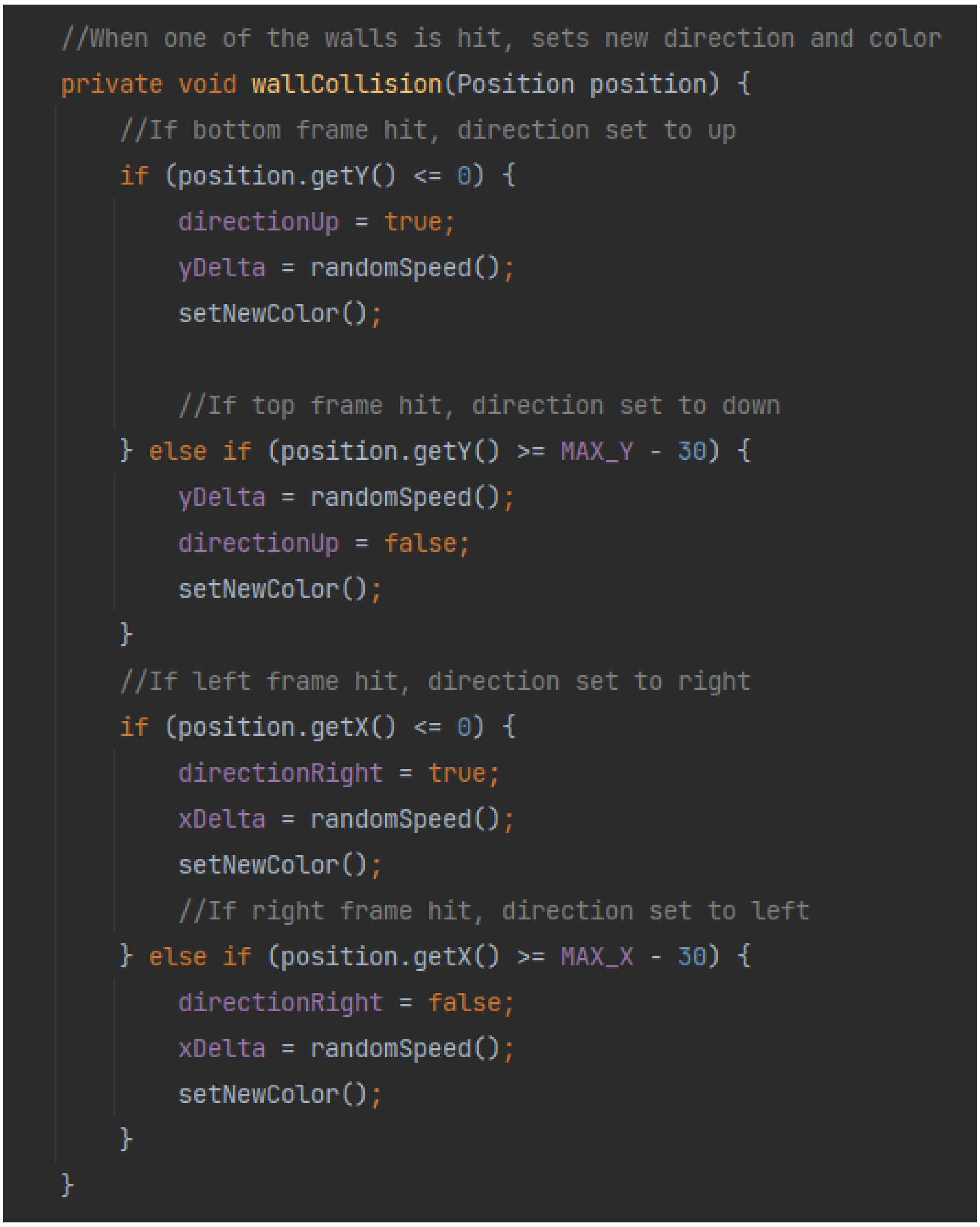
```java
//Checks the direction of the ball, moves it to that way
private void moveBall(Position position) {
    int x = position.getX();
    int y = position.getY();

    if (directionRight) {
        position.setX(x + xDelta);
    } else {
        position.setX(x - xDelta);
    }
    if (directionUp) {
        position.setY(y + yDelta);
    } else {
        position.setY(y - yDelta);
    }
}
```

The moveBall method changes the location of the ball object, by checking its direction in the coordination system, and adding or reducing a new value to/from it. For example, If the ball should move to the left, its int x value will be increased. The Swing's (JPanel) inner method, "repaint()", is used for graphically displaying the ball in its new location.



```java
public void ballCollision(Position position) {
    //Copy list
    List<Position> otherPositions = new ArrayList<>(positionList);
    //Remove ball with the current id from list (not to check collision with itself)
    otherPositions.remove(id);

    //For each loop goes through position of other balls
    for (Position otherPosition : otherPositions) {
        boolean collision = isColliding(position, otherPosition);

        //If collision is true the ball gets new random direction, speed and new color
        if (collision) {
            yDelta = randomSpeed();
            xDelta = randomSpeed();
            //New color
            setNewColor();

            //Change new direction to opposite of current direction
            directionRight = !directionRight;
            directionUp = !directionUp;
        }
    }
}

private boolean isColliding(Position position, Position otherPosition) {
    //Calculates area of balls from balls position
    double xDif = position.getX() - otherPosition.getX();
    double yDif = position.getY() - otherPosition.getY();
    double distanceSquared = xDif * xDif + yDif * yDif;
    //Creates a boolean from calculation
    boolean collision = distanceSquared < (15 + 15) * (15 + 15);
    return collision;
}
```

The **ballCollision** method first removes the checked ball's position from the list of all positions. Then calls the **isColliding** method which calculates the two checked balls hitboxes with an equation, and checks if they overlap. If the balls do collide, their speed and color are changed and their direction is reversed.



```java
//When one of the walls is hit, sets new direction and color
private void wallCollision(Position position) {
    //If bottom frame hit, direction set to up
    if (position.getY() <= 0) {
        directionUp = true;
        yDelta = randomSpeed();
        setNewColor();

        //If top frame hit, direction set to down
    } else if (position.getY() >= MAX_Y - 30) {
        yDelta = randomSpeed();
        directionUp = false;
        setNewColor();
    }
    //If left frame hit, direction set to right
    if (position.getX() <= 0) {
        directionRight = true;
        xDelta = randomSpeed();
        setNewColor();
        //If right frame hit, direction set to left
    } else if (position.getX() >= MAX_X - 30) {
        directionRight = false;
        xDelta = randomSpeed();
        setNewColor();
    }
}
```

The wallCollision method checks if a ball’s position has reached one of the sides of the bouncing area. If it does, its direction is changed, it receives a random speed, and it also changes to a random color.