# Objective

You will create a class to represent a bouncing ball. The bouncing ball will encapsulate fields representing the ball’s position and velocity, and the class will provide methods to update the balls location and to render the ball on the screen. We will use PVector to implement the location and velocity. See <http://processing.org/reference/PVector.html> for more details.

You will create a collection of ball objects, and invoke their update and render methods in the **draw()** loop of your sketch. Finally, you will add a new ball every time the mouse is clicked on the sketch.

## Topics: abstraction, fields and methods, vectors, collections

# Instructions

Begin by switching to the Ball tab in the sketch. You will need to add the following instance fields to the class:

* PVector pos – this vector represents the (x, y) position of the ball
* PVector vel – this vector represents the ball’s velocity
* float size – the diameter of the ball

Add two constructors to your Ball class, a default constructor and an initialization constructor. Use the following method signature for the initialization constructor:

**Ball(**floatx, float y, float velX, float velY, float aSize**)**

You may decide on the default values for the default constructor. You might want them to be randomly generated.

Now add the following two methods to the Ball class. Use *stubs,* which are methods with no code in theirbody*,* for now, we will add the implementation in a later step.

* void **update()** – This method will update the ball’s position based on its velocity
* void **render()** – This method will cause the ball to paint itself on the screen at its current location

Now we will add code to the main tab of the sketch. Define a global variable of type ArrayList<Ball> with an appropriate name. In the **setup()** method, you will instantiate the ArrayList. After instantiating the ArrayList, add a new default Ball object to the list.

In the **draw()** method, you will add a loop that traverses your list of ball objects. In the loop body, invoke the update method, followed by the **render()** method, for each ball object in the list.

# Instructions (continued)

You will now implement the **render()** method of the Ball class, so we can see the Ball that we created. Here is high level algorithm

1. Draw a circle at the ball’s location with a diameter equal to size

Well, that was easy! Refinement may suggest some additional steps, perhaps involving translation and pushing/popping the transformation matrix. You may write the implementation however you see fit.

The **update()** method presents a more interesting challenge. Here is the high level algorithm:

1. Add the velocity vector to the location vector
2. Check to see if the ball went out of bounds
3. If the ball is out of bounds:
   1. Move the ball back in bounds
   2. Reverse the sign of the velocity vector’s relevant component (i.e. x or y)

Now check your work, and you should have a Ball that bounces forever inside the sketch window. To create more balls, you will override the following method:

void **mouseClicked**() – this method is called whenever the mouse is clicked on the sketch

Whenever a click is received, you will instantiate a new ball object at the current mouse position and add it to the list. It will then update and render along with the rest of the balls.

# Hints

Try reading this tutorial on the Processing website if you get stuck: <http://processing.org/learning/pvector/>

# Challenge

Challenge #1: Add gravity! Gravity is acceleration in a constant direction, which would be added to the velocity before the velocity is added to the locaton.

Challenge #2: Give the balls more properties, like customizable stroke and fill, or mass.

Challenge #3: Make the balls bounce off each other. This is a bit harder.