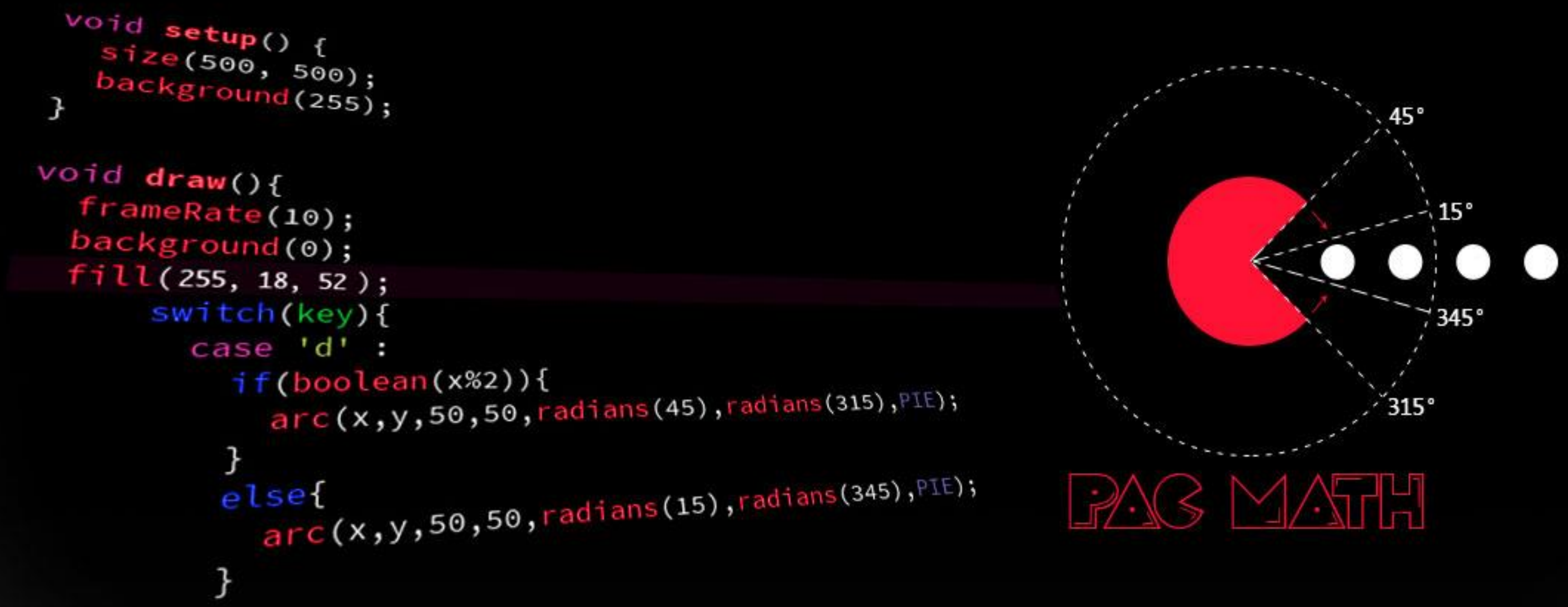


```
void setup() {
  size(500, 500);
  background(255);
}

void draw(){
  frameRate(10);
  background(0);
  fill(255, 18, 52 );
    switch(key){
      case 'd' :
        if(boolean(x%2)){
          arc(x,y,50,50,radians(45),radians(315),PIE);
        }
        else{
          arc(x,y,50,50,radians(15),radians(345),PIE);
        }
```

PROGRAMMING FOR ARTISTS II

ARRAY LISTS

OBJECT-ARRAYS: SUMMARY

Array declaration

```
Dot[] dotsArr = new Dot[100];
```

Array initialization

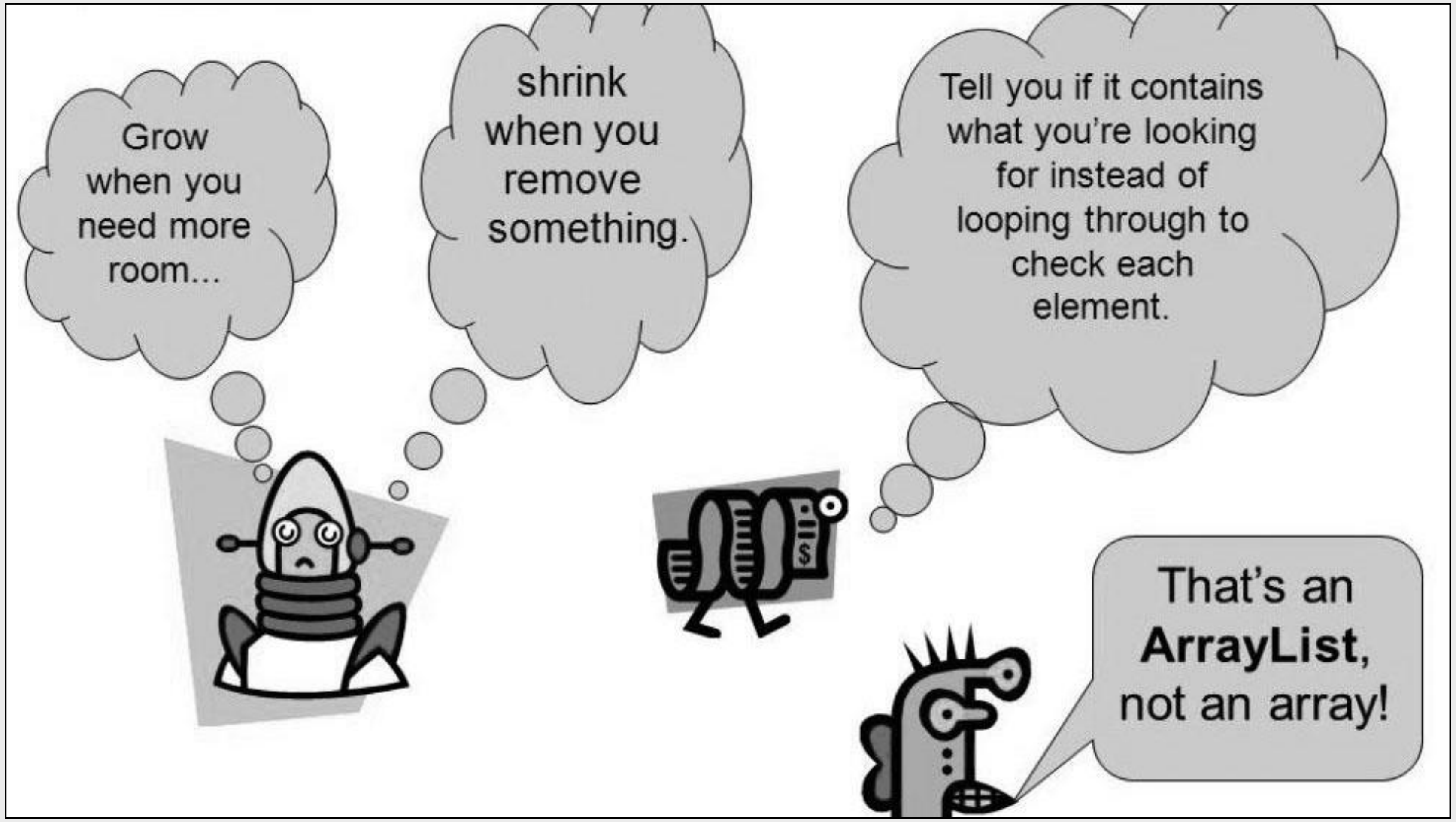
```
for(int i = 0 ; i < dotsArr.length; i++){
    dotsArr[i] = new Dot(new Pvector(200,200), … );
}
```

Use the objects

```
for(int i = 0 ; i < dotsArr.length; i++){
        dotsArr[i].display();
}
```

ARRAY LISTS

...if only an array could...


Grow when you need more room...
shrink when you remove something.
Tell you if it contains what you're looking for instead of looping through to check each element.
That's an **ArrayList**, not an array!

ARRAY LISTS: WHAT?

Arraylists are **dynamic** arrays in Java:

→ **stores objects, no primitive data types**

- to use other primitive types (int, float, ..), you must specify an equivalent wrapper class (IntList, FloatList, ..)

→ **no definition of size required before use**

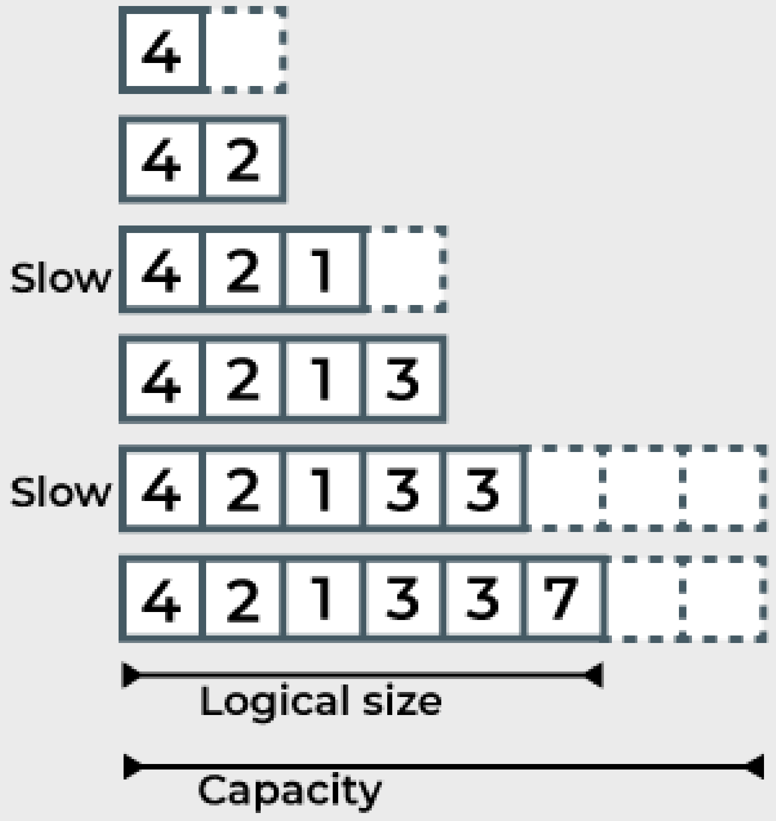
- the **size** is the **number of elements** in the list;
- the **capacity** is **how many elements** the list can **potentially accommodate** without reallocating its internal structures (default 10)
  f.e. when you call ***new ArrayList<Dot>(20)*,** you are setting the list's initial capacity, not its size. In other words, when constructed in this manner, the array list starts its life empty.

ARRAY LISTS: WHAT?

→ **is resized dynamically!**

- easily add/remove objects

➢ **Note: it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed!**



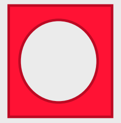
ARRAY LISTS: HOW?

- **Create a list:**

```
ArrayList<Dot> dotsArr = new ArrayList<Dot>();
```

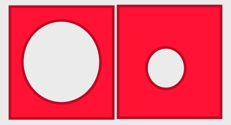
- **Put something in it:**

```
Dot bigDot = new Dot();
dotsArr.add(bigDot);
```



- **Put something else in it:**

```
Dot smallDot = new Dot();
dotsArr.add(smallDot);
```



ARRAY LISTS: WHAT CAN YOU DO WITH IT?

- **Add** new objects to the list
- **Remove** old objects from the list
- Ask the list if it **contains** a certain object
- Ask if the list is **empty**
- **Find** the **index** of a certain object
- **Get** the **size** of the list
- **Get** an object from a **certain position** in the list
- …

ARRAY LISTS: METHODS

- **size() : used when we need to know the size**

  → size NOT EQUAL TO capacity!

You can use the size() function in a loop:

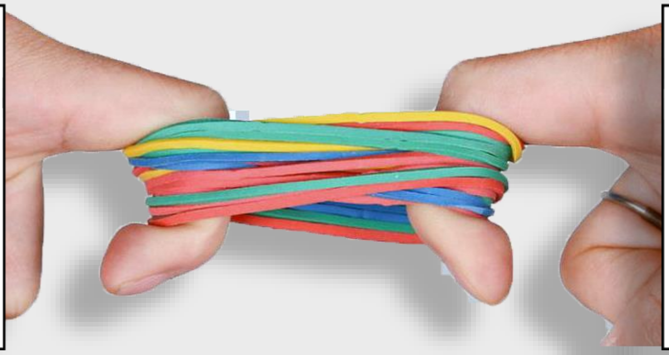
```
for(int i = 0; i < dotsArr.size(); i++){
        //doSomething…
}
```

ARRAY LISTS: METHODS

- **add() : used when we need to add objects**
  sets the object into the first available slot and so on, **until** it **needs** to **resize** the array, which it does by creating a new array and **copying** the **contents** of the old array.

→ The Processing **append()** method is **allocating** a **new array** and **copying** contents **each time** it is called.

```
//add object at end of list
Dot dotObject = new Dot();
dotsArr.add(dotObject);
```



```
//add object at index 4
Dot dotObject = new Dot();
dotsArr.add(4, dotObject);
```

ARRAY LISTS: METHODS

- **get() : used when we need the information of a specific object of a given index parameter**

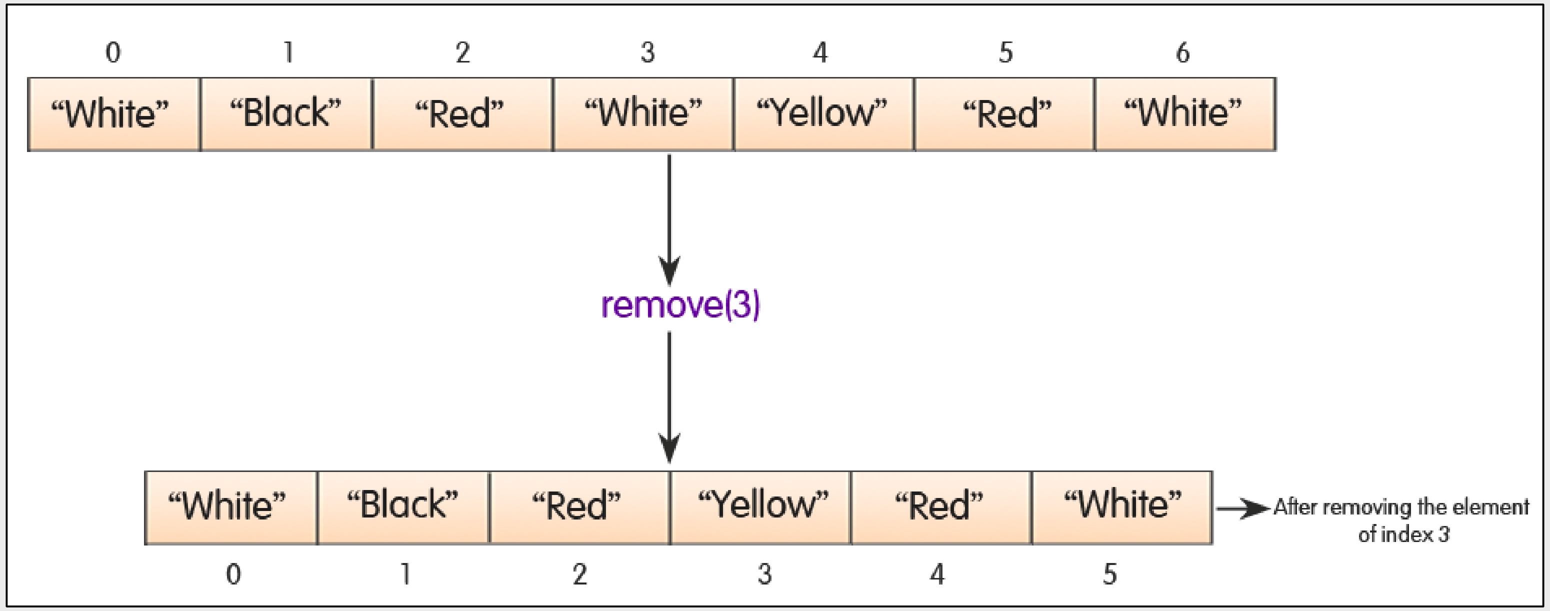
  → An ArrayList holds values at different indexes, as an array, but **acces index with the .get(*index*) method, not [*index*] !!**

```
  //create a temporary variable
  Dot myDot = dotsArr.get(0);
  println(myDot.getXPosition();
OR
  //call a method immediately
  println(dotsArr.get(0).getXPosition();
```

ARRAY LISTS: METHODS

- **remove() : used when we need to remove objects** with the given parameter, it removes the object at that index

```
//remove object at index position 4
dotsArr.remove(4);
```



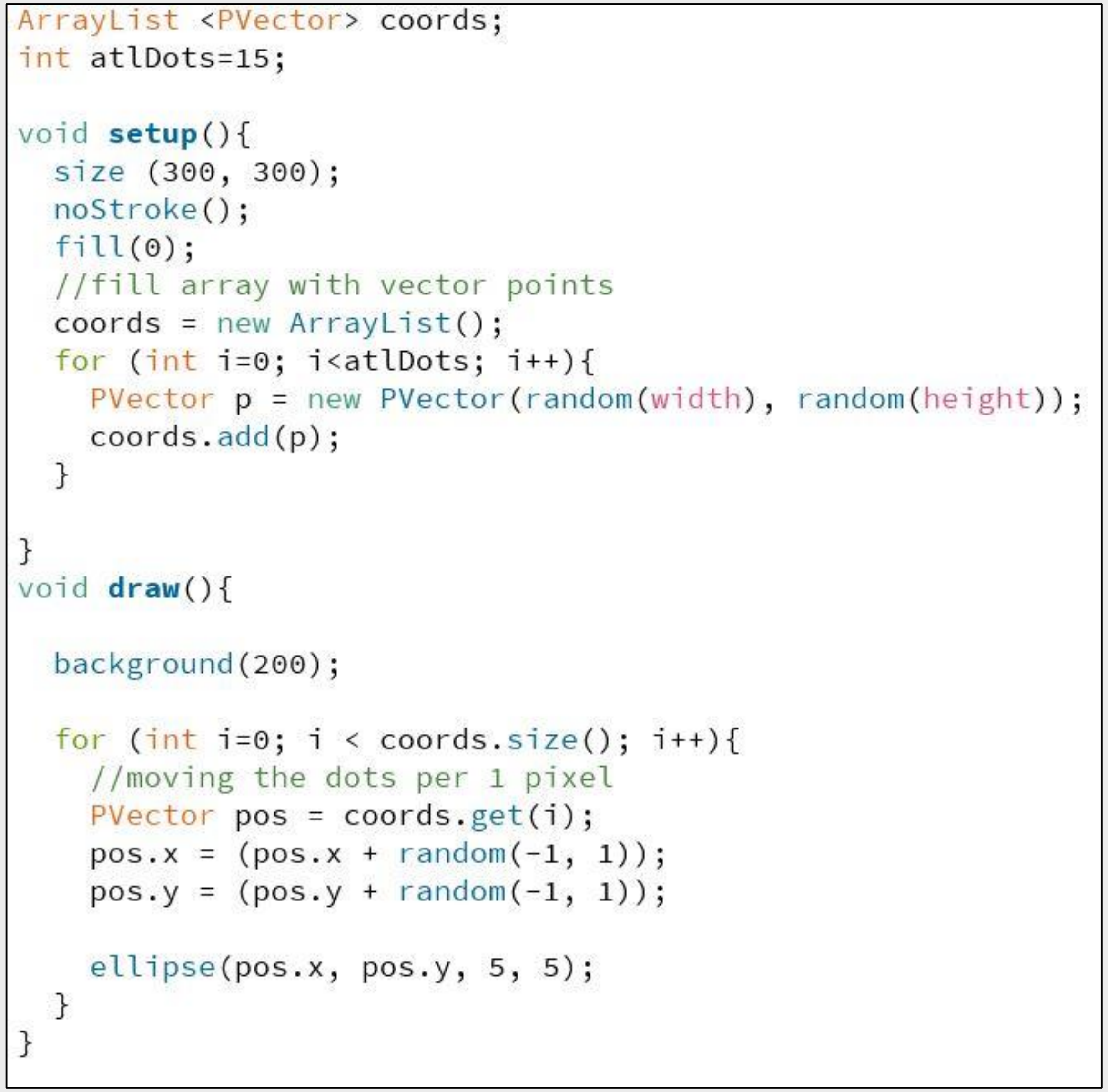
ARRAY LISTS: METHODS

- when you **remove** a random object from an ArrayList **inside a loop**, starting at index 0, everything after that index is **shifted down**, so during the **next iteration**, original object at **index 0** is **skipped**.

→ **loop backwards, starting at size -1 !!**

```
for(int i = dotsArr.size()-1; i >= 0; i--){
    if(dotsArr.get(i).isDead()){
        dotsArr.remove(i);
    }
}
```

ARRAY LIST: EXAMPLE PVector



```
ArrayList <PVector> coords;
int atlDots=15;

void setup(){
  size (300, 300);
  noStroke();
  fill(0);
  //fill array with vector points
  coords = new ArrayList();
  for (int i=0; i<atlDots; i++){
    PVector p = new PVector(random(width), random(height));
    coords.add(p);
  }

}
void draw(){

  background(200);

  for (int i=0; i < coords.size(); i++){
    //moving the dots per 1 pixel
    PVector pos = coords.get(i);
    pos.x = (pos.x + random(-1, 1));
    pos.y = (pos.y + random(-1, 1));

    ellipse(pos.x, pos.y, 5, 5);
  }
}
```

ARRAY LIST: EXAMPLE Dots

Main application

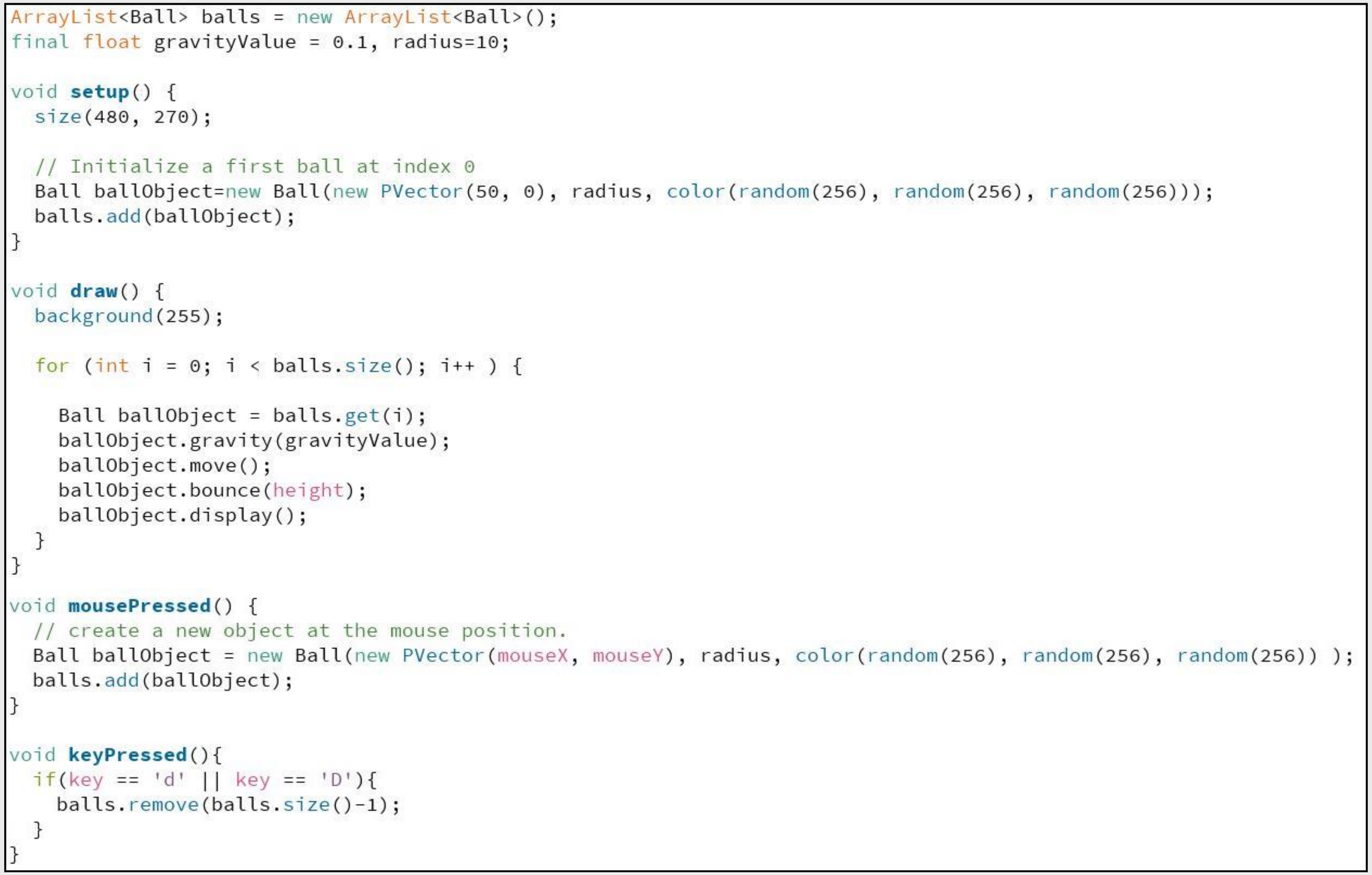
```
// create an empty array for 100 Dot objects
int numDots=100;
ArrayList<Dot> dotsArr = new ArrayList<Dot>();

void setup() {
  size(400, 400);
  smooth();

  // and actually create the objects and populate the
  // array with them
  for (int i=0; i<numDots; i++) {
    dotsArr.add(new Dot(new PVector(200,200),
                new PVector(random(-10,10),
                random(-10,10)),
                5));
  }
}

void draw() {

  background(0);

  // iterate through every moving dot
  for (int i=0; i<dotsArr.size(); i++) {

    Dot dotObject = dotsArr.get(i);
    dotObject.update();
    dotObject.checkCollisions(width, height);
    dotObject.display();
  }
}
```

Class Dot

```
class Dot {
  private PVector position;
  private PVector speed;
  private float radius;

  Dot(PVector position, PVector speed, float radius) {
    this.position = position;
    this.speed = speed;
    this.radius = radius;
  }

  void update() {
    //ADD VECTORS WITH add method!
    position.add(speed);
  }

  void checkCollisions(float maxWidth, float maxHeight) {

    if ( (position.x < radius) || (position.x > maxWidth-radius)) {
      speed.x = -speed.x;
    }

    if ( (position.y < radius) || (position.y > maxHeight-radius)) {
      speed.y = -speed.y;
    }
  }

  void display() {
    fill(255);
    ellipse(position.x, position.y, radius*2, radius*2);
  }
}
```

ARRAY LIST: EXAMPLE Balls

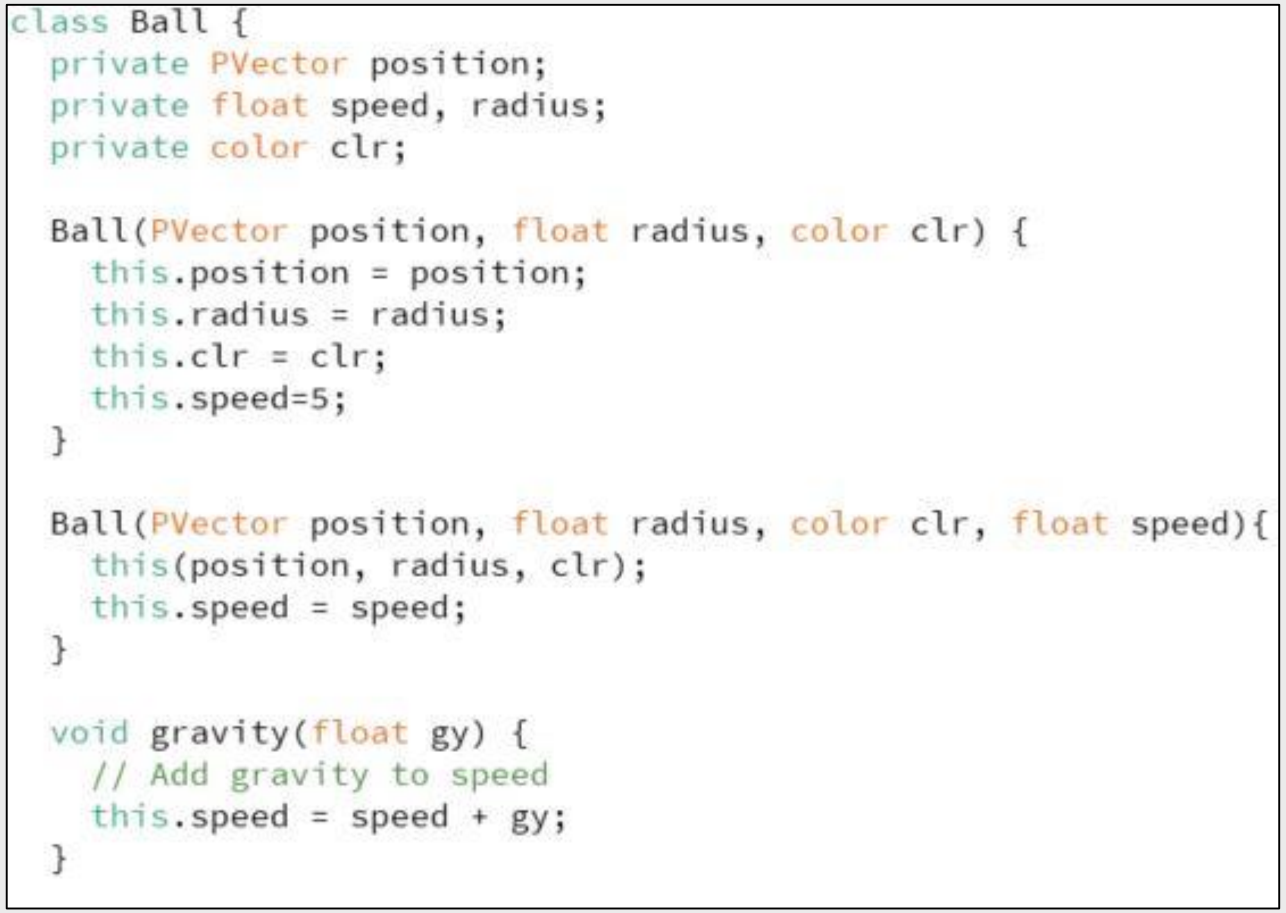
Main application



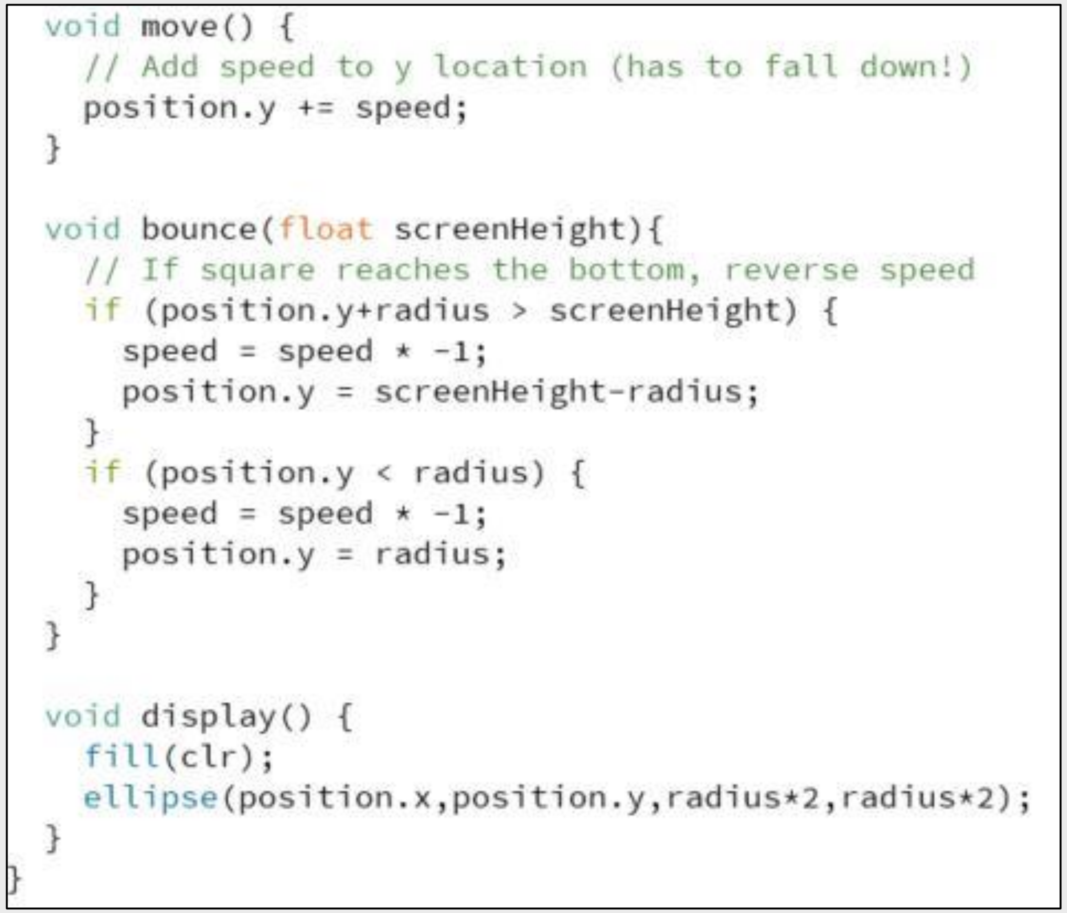
```
ArrayList<Ball> balls = new ArrayList<Ball>();
final float gravityValue = 0.1, radius=10;

void setup() {
  size(480, 270);

  // Initialize a first ball at index 0
  Ball ballObject=new Ball(new PVector(50, 0), radius, color(random(256), random(256), random(256)));
  balls.add(ballObject);
}

void draw() {
  background(255);

  for (int i = 0; i < balls.size(); i++ ) {

    Ball ballObject = balls.get(i);
    ballObject.gravity(gravityValue);
    ballObject.move();
    ballObject.bounce(height);
    ballObject.display();
  }
}

void mousePressed() {
  // create a new object at the mouse position.
  Ball ballObject = new Ball(new PVector(mouseX, mouseY), radius, color(random(256), random(256), random(256)) );
  balls.add(ballObject);
}

void keyPressed(){
  if(key == 'd' || key == 'D'){
    balls.remove(balls.size()-1);
  }
}
```

ARRAY LIST: EXAMPLE Balls

Class Ball



```
class Ball {
  private PVector position;
  private float speed, radius;
  private color clr;

  Ball(PVector position, float radius, color clr) {
    this.position = position;
    this.radius = radius;
    this.clr = clr;
    this.speed=5;
  }

  Ball(PVector position, float radius, color clr, float speed){
    this(position, radius, clr);
    this.speed = speed;
  }

  void gravity(float gy) {
    // Add gravity to speed
    this.speed = speed + gy;
  }
```



```
  void move() {
    // Add speed to y location (has to fall down!)
    position.y += speed;
  }

  void bounce(float screenHeight){
    // If square reaches the bottom, reverse speed
    if (position.y+radius > screenHeight) {
      speed = speed * -1;
      position.y = screenHeight-radius;
    }
    if (position.y < radius) {
      speed = speed * -1;
      position.y = radius;
    }
  }

  void display() {
    fill(clr);
    ellipse(position.x,position.y,radius*2,radius*2);
  }
}
```