

PAT 204/504 (Fall 2026)

Creative Coding

Lecture 7: Lists & Dictionaries

Instructor: Hao-Wen (Herman) Dong

Lists

| Lists

- Similar to arrays, lists hold several items of the same data type
- However, lists are built to have a **dynamic size**
- The simplest ones are **IntList** and **FloatList**

Example: Three Circles

```
IntList pos = new IntList();
```

Declaration

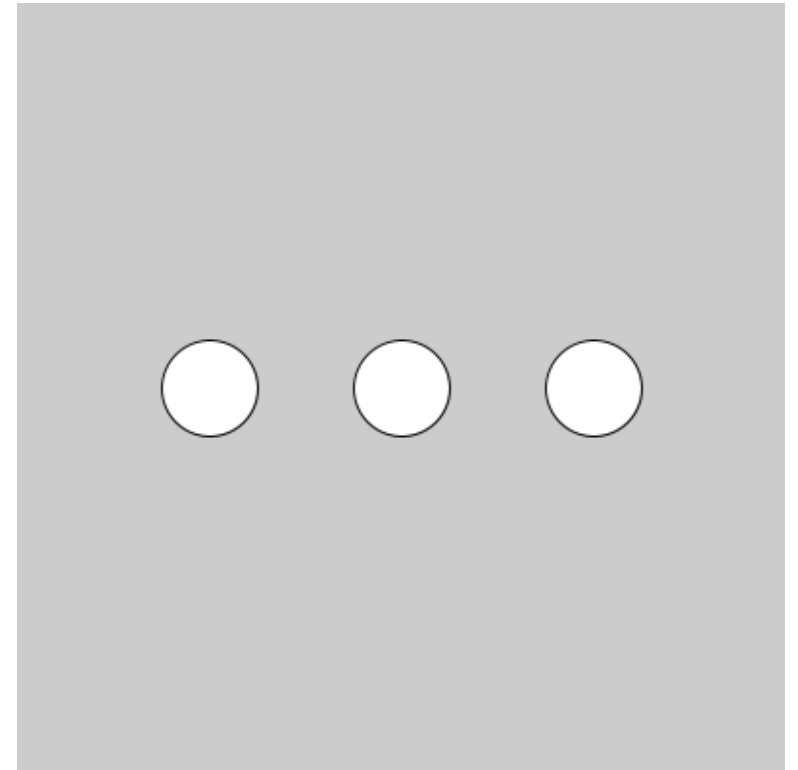
```
void setup() {  
    size(400, 400);
```

```
    pos.append(100);  
    pos.append(200);  
    pos.append(300);
```

Initialization

```
}
```

```
void draw() {  
    for (int i = 0; i < pos.size(); i++) {  
        circle(pos.get(i), 200, 50);  
    }  
}
```

Length of the list

Example: Three Circles

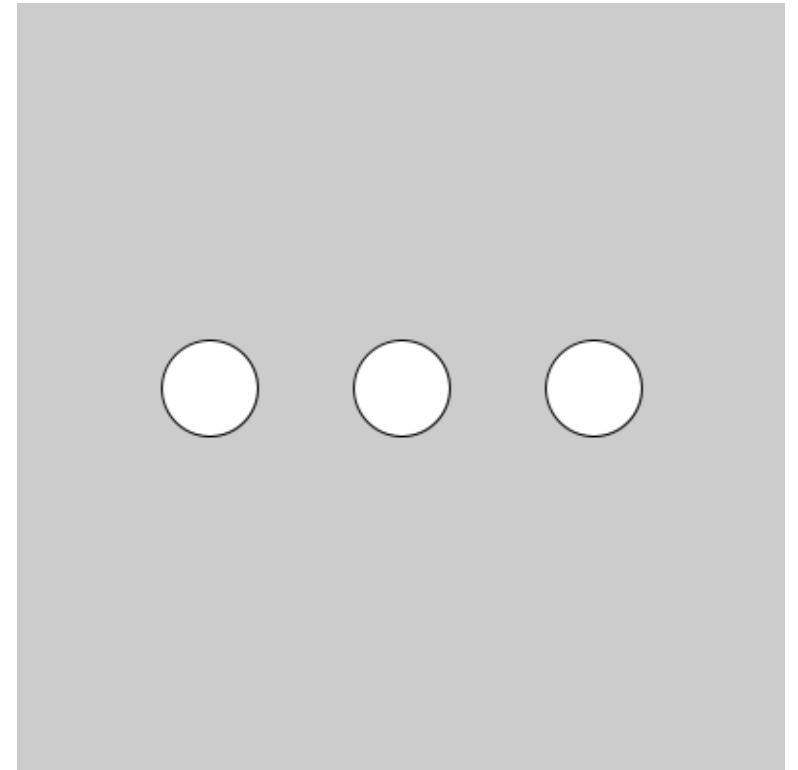
```
IntList pos = new IntList();
```

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

```
    pos.append(100);  
    pos.append(200);  
    pos.append(300);  
}
```

```
void draw() {  
    for (int x: pos) {  
        circle(x, 200, 50);  
    }  
}
```

For-each loop



Arrays vs. Lists

Array

```
float[] pos = new float[3]; Declaration
```

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

```
  pos[0] = 100;  
  pos[1] = 200;  
  pos[2] = 300;
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.length; i++) {  
    circle(pos[i], 200, 50);  
  }  
}
```

Length of the array

```
FloatList pos = new IntList(); Declaration
```

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

```
  pos.append(100);  
  pos.append(200);  
  pos.append(300);
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.size(); i++) {  
    circle(pos.get(i), 200, 50);  
  }  
}
```

Length of the list

List Methods

- `size()` Return the size of the list
- `get()` Return the value at the specified index
- `append()` Append an item to the end of the list
- `insert()` Insert an item to the specific index
- `set()` Set the value at the specified index
- `remove()` Remove an item at the specified index
- `clear()` Clear everything in the list
- `hasValue()` Whether the value is in the list or not

List Methods

- Unlike arrays, most list methods are **in-place**

- `add()`

- `sub()`

- `mult()`

- `div()`

- `sort()`

- `sortReverse()`

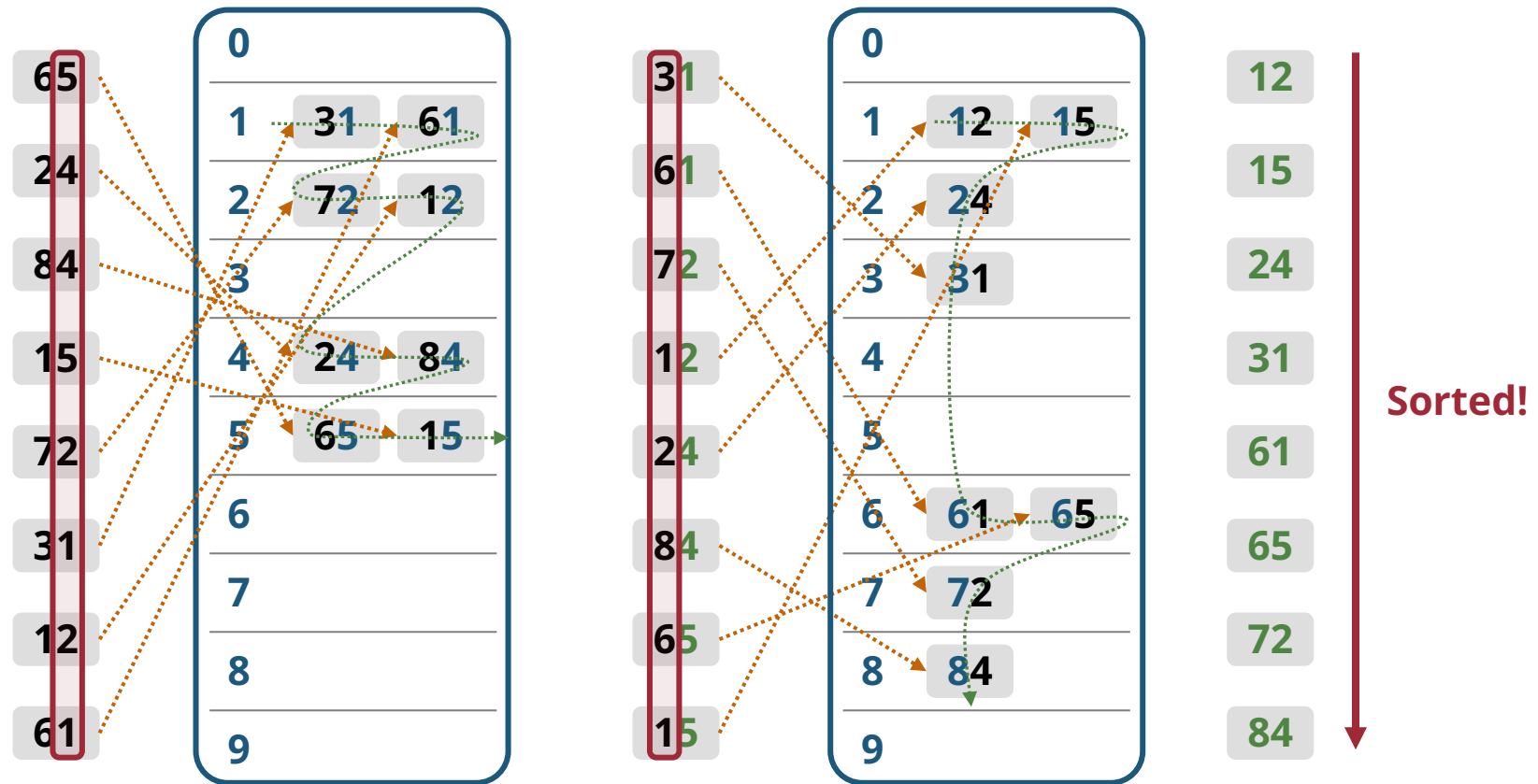
- `shuffle()`

Does not create
a new copy

| In-place vs. Not-in-place Algorithm

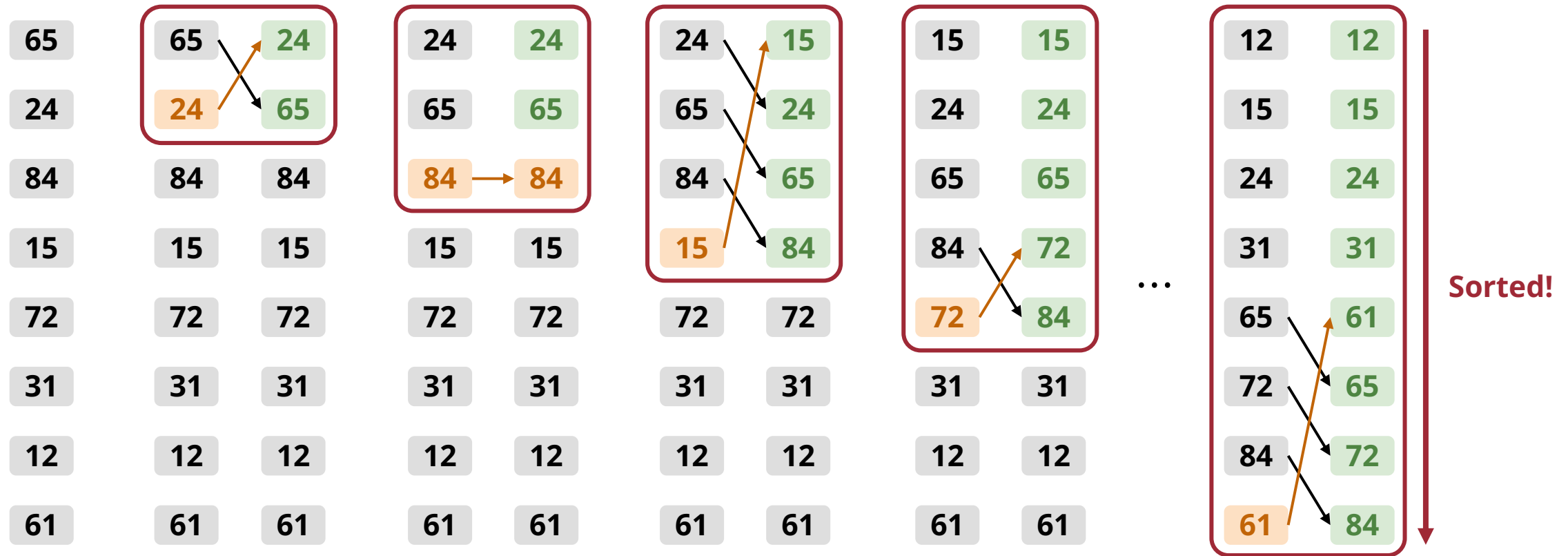
- In-place algorithm does not require additional memory
- Not-in-place algorithm needs extra memory to store intermediate results

Example of Not-in-place Algorithms: Radix Sort



Need extra memory to store intermediate results!

Example of In-place Algorithms: Insertion Sort



Don't need extra memory!

Sorting an Array vs. Sorting a List

```
int[] arr = {3, 2, 1};  
sort(arr);  
println(arr);
```

[0] 3
[1] 2
[2] 1

**sort(arr) returns a
new sorted array**

```
int[] arr = {3, 2, 1};  
arr = sort(arr);  
println(arr);
```

[0] 1
[1] 2
[2] 3

```
IntList li = new IntList();  
li.append(3);  
li.append(2);  
li.append(1);  
li.sort();  
println(li);
```

IntList size=3 [1, 2, 3]

**List.sort() returns the
original list, sorted**

| List-to-Array Conversion

- **List.toArray()** converts a list into an array
- No easy way the other way around

| Arrays vs. Lists

	Arrays	Lists
Size		
Item data type		
Access speed		
Memory requirement		
Multi-dimensional		

| Arrays vs. Lists

	Arrays	Lists
Size	Fixed	Dynamic
Item data type	Same	Can be different
Access speed	Faster	Slower
Memory requirement	Low	High
Multi-dimensional	Possible	Not supported

List of Objects: **ArrayList**

- **ArrayList** is a list of objects
- What's the difference?

Array of objects

```
Ball[] balls = new Ball[20];
```

```
String[] strs = new String[20];
```

ArrayList

```
ArrayList<Ball> balls = new ArrayList<Ball>()
```

```
StringList strs = new StringList()
```

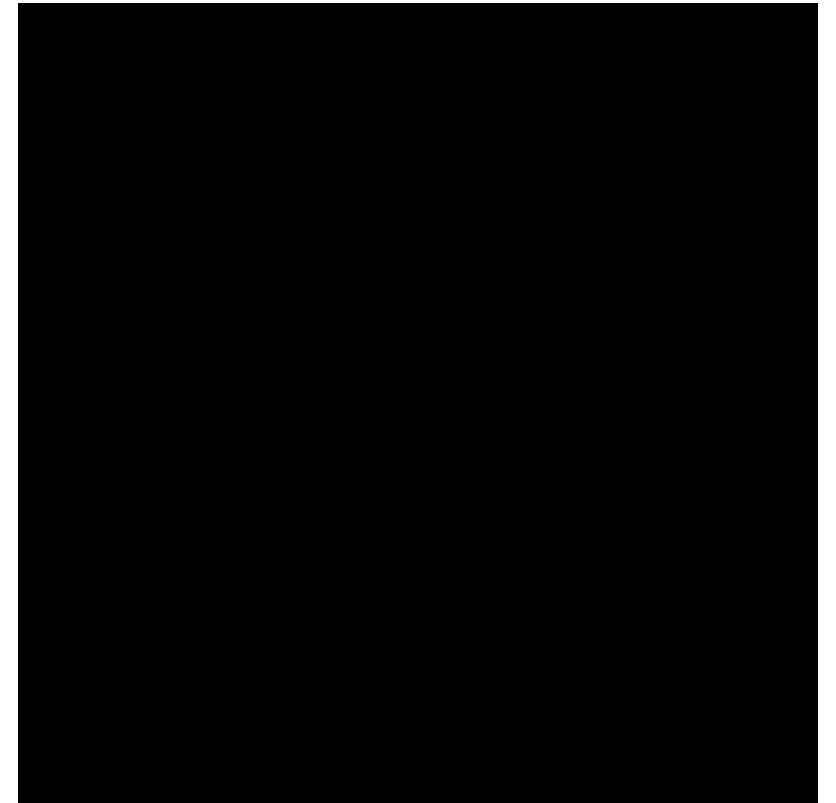
🤔 Exercise: Bouncing Balls

- Add a ball when the mouse is clicked
- The new ball starts from where the mouse is
- Two approaches
 - Use an **array of objects**

```
Ball[] balls = new Ball[20];
```

- Use an **ArrayList**

```
ArrayList<Ball> balls = new ArrayList<Ball>()
```



Example: Bouncing Ball

```
class Ball {
```

```
    float size = 10;  
    float speed = 5;  
    float x, y, speedX, speedY;
```

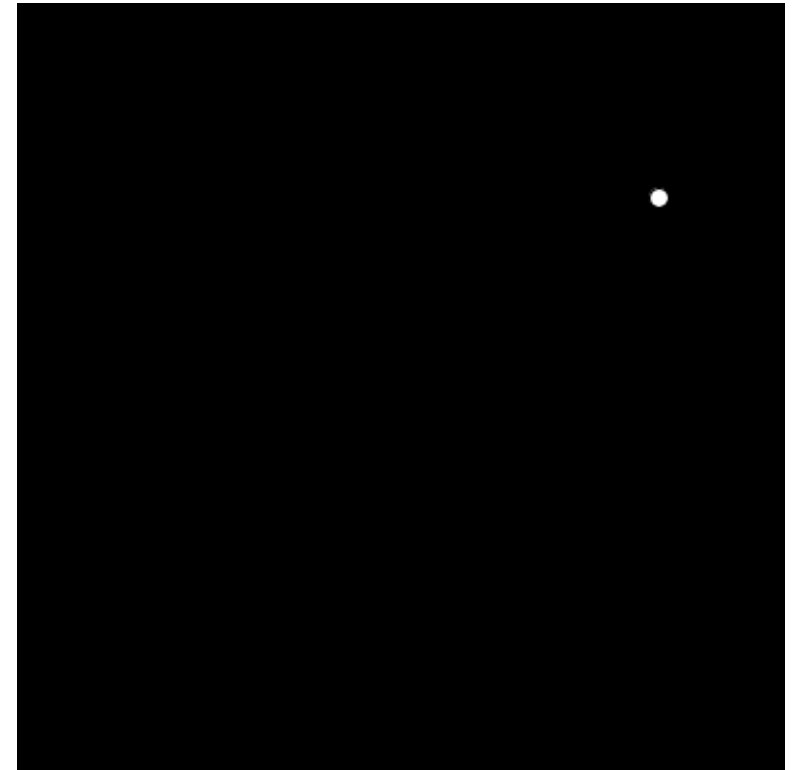
Fields

```
    Ball() {  
        // Constructor  
    }
```

Constructor

```
    void show() {  
        // Show the ball  
    }  
  
    void move() {  
        // Move the ball  
    }  
  
    void checkWalls() {  
        // Check if the ball hit the walls  
    }  
}
```

Methods



Example: Bouncing Balls

`Ball[] balls = new Ball[20];` An array of objects

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

```
  for (int i = 0; i < balls.length; i++) {  
    balls[i] = new Ball();  
  }
```

Initialization

```
void draw() {  
  background(0);
```

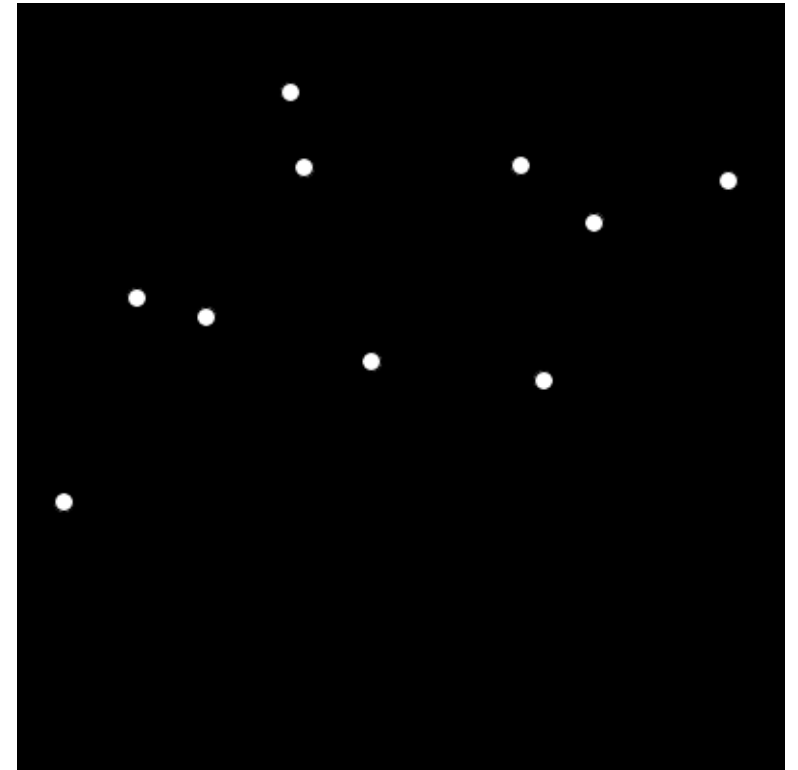
```
  for (int i = 0; i < balls.length; i++) {
```

```
    balls[i].move();
```

```
    balls[i].checkWalls();
```

```
    balls[i].show();
```

Call the methods!



Arrays vs. Lists

Array

```
float[] pos = new float[3]; Declaration
```

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

```
  pos[0] = 100;  
  pos[1] = 200;  
  pos[2] = 300;
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.length; i++) {  
    circle(pos[i], 200, 50);  
  }  
}
```

Length of the array

```
FloatList pos = new IntList(); Declaration
```

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

```
  pos.append(100);  
  pos.append(200);  
  pos.append(300);
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.size(); i++) {  
    circle(pos.get(i), 200, 50);  
  }  
}
```

Length of the list

| Arrays vs. Lists

	Arrays	Lists
Size	Fixed	Dynamic
Item data type	Same	Can be different
Access speed	Faster	Slower
Memory requirement	Low	High
Multi-dimensional	Possible	Not supported

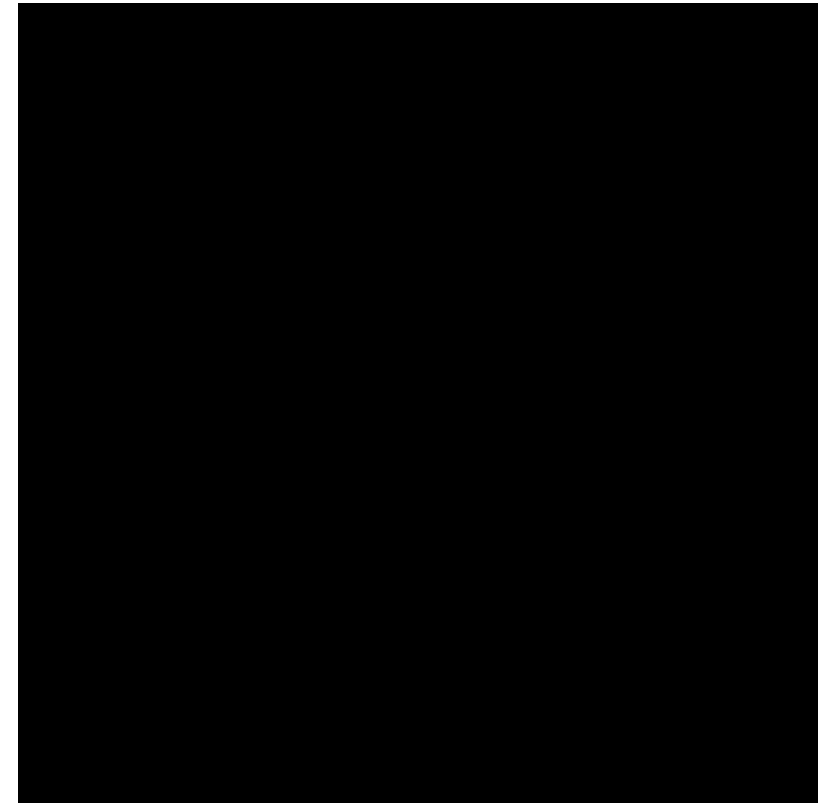
🤔 Exercise: Bouncing Balls

- Add a ball when the mouse is clicked
- The new ball starts from where the mouse is
- Two approaches
 - Use an **array of objects**

```
Ball[] balls = new Ball[20];
```

- Use an **ArrayList**

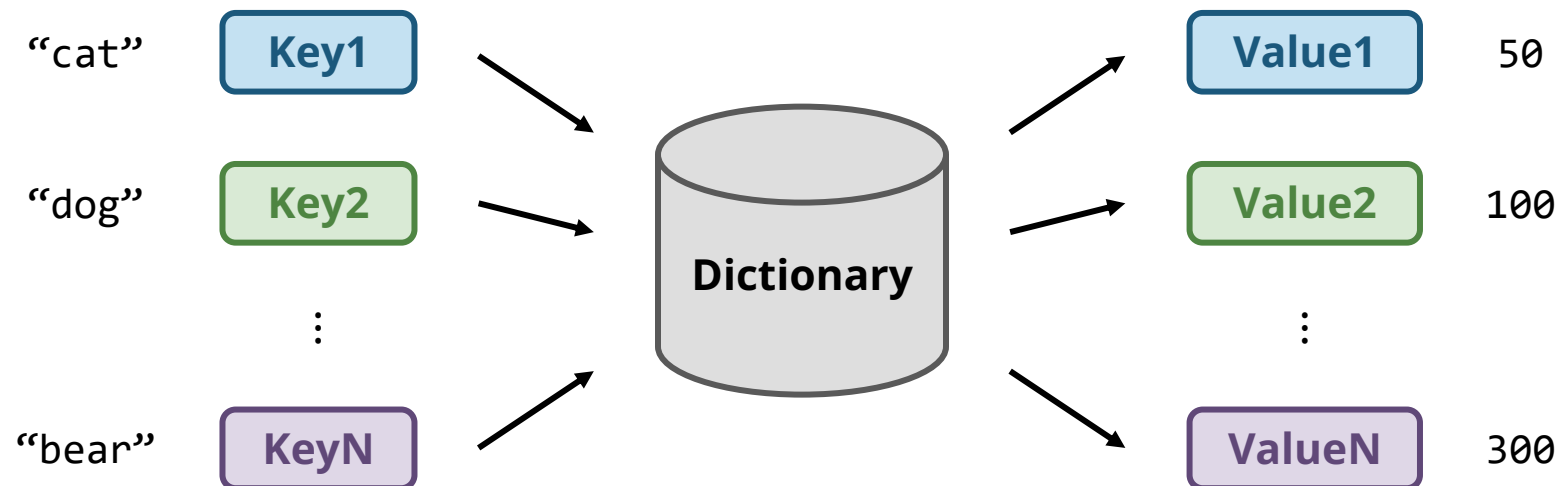
```
ArrayList<Ball> balls = new ArrayList<Ball>()
```



Dictionaries

Dictionary

- A dictionary supports **fast lookup** of the corresponding “**value**” for a “**key**”
- **IntDict** String-to-integer mapping
- **FloatDict** String-to-float mapping
- **StringDict** String-to-string mapping



IntDict: String-to-Integer Mapping

`IntDict weightMap = new IntDict();` Declare the shape

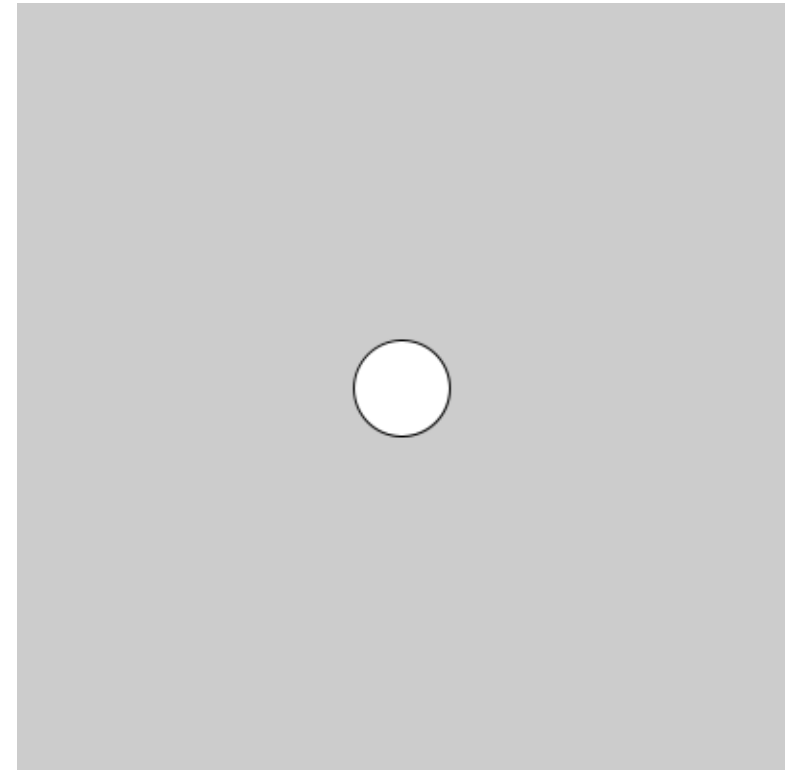
```
void setup() {  
  size(400, 400);  
  noLoop();  
}
```

```
weightMap.set("cat", 50);  
weightMap.set("dog", 100);  
weightMap.set("bear", 300);
```

Set up the dictionary

```
void draw() {  
  int weight = weightMap.get("cat");  
  circle(200, 200, weight);  
}
```

Query the dictionary



IntDict vs. "string[] & int[]"

- Can we use **a string array** and **an integer array** to do something similar?

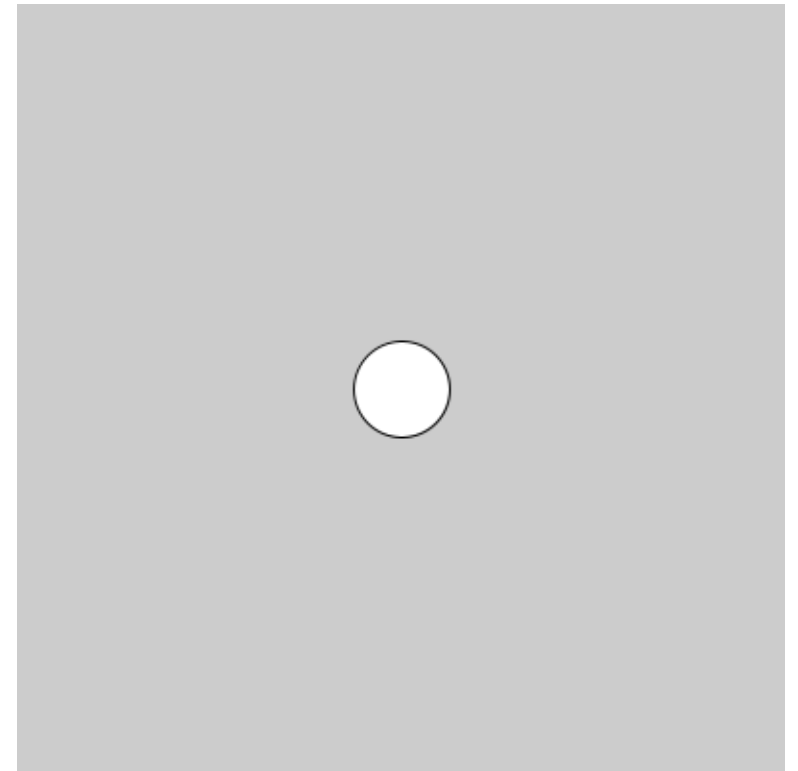
```
String[] animals = {"cat", "dog", "bear"};  
int[] weights = {50, 100, 300};
```

```
void setup() {  
  size(400, 400);  
  noLoop();  
}
```

```
void draw() {  
  for (int i = 0; i < animals.length; i++) {  
    if (animals[i] == "cat") {  
      circle(200, 200, weights[i]);  
    }  
  }  
}
```

This is slow because we may need to go through the whole array to find the match!

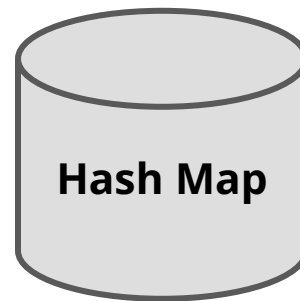
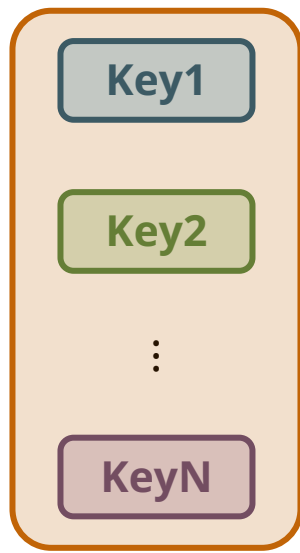
Use a for loop to look up the "key"



Hash Maps

- A **hash map** is the magic behind dictionaries
- **HashMap** Key-to-value mapping with desired key and value data types

Can be any data type!



Can be any data type!



A String-to-Array Hash Map

```
HashMap<String, int[]> m;
```

Declare a hash map that maps strings to integer arrays

```
void setup() {  
  size(400, 400);  
  noLoop();  
}
```

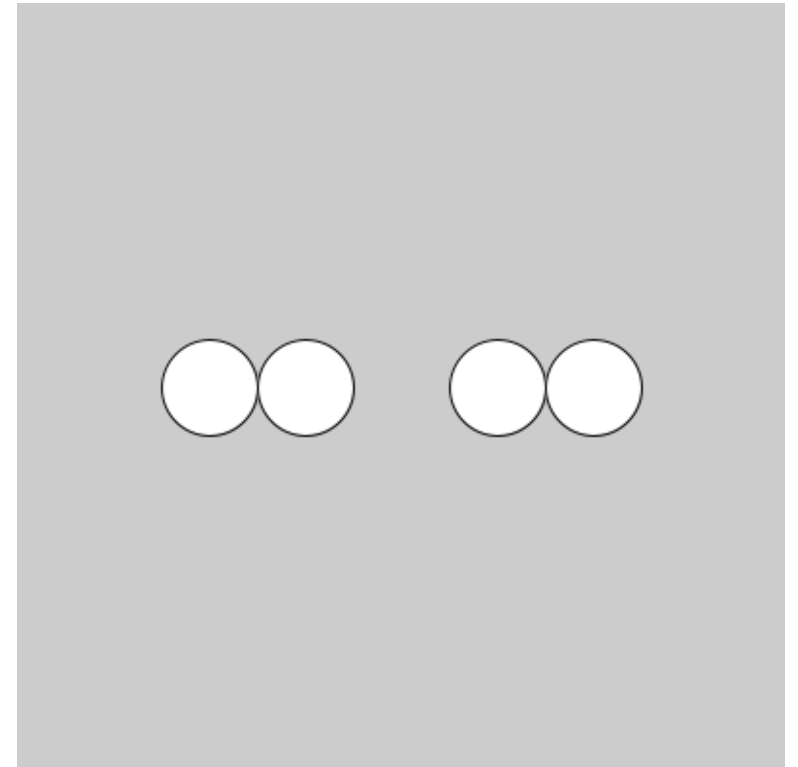
```
m = new HashMap<String, int[]>();  
m.put("cat", new int[]{100, 150, 200});  
m.put("dog", new int[]{100, 300});  
m.put("bear", new int[]{100, 150, 250, 300});
```

Set up the dictionary

```
void draw() {  
  int[] val = m.get("bear");  
  for (int x: val) {  
    circle(x, 200, 50);  
  }  
  println(m);  
}
```

Query the hash map

```
{cat=[I@76cc6ab0, bear=[I@1d88e68, dog=[I@4e0d571f}
```



Recap

Arrays vs. Lists

Array

```
float[] pos = new float[3];
```

 Declaration

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

```
  pos[0] = 100;  
  pos[1] = 200;  
  pos[2] = 300;
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.length; i++) {  
    circle(pos[i], 200, 50);  
  }  
}
```

Length of the array

```
FloatList pos = new IntList();
```

 Declaration

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

```
  pos.append(100);  
  pos.append(200);  
  pos.append(300);
```

Initialization

```
}
```

```
void draw() {  
  for (int i = 0; i < pos.size(); i++) {  
    circle(pos.get(i), 200, 50);  
  }  
}
```

Length of the list

Sorting an Array vs. Sorting a List

```
int[] arr = {3, 2, 1};  
sort(arr);  
println(arr);
```

[0] 3
[1] 2
[2] 1

**sort(arr) returns a
new sorted array**

```
int[] arr = {3, 2, 1};  
arr = sort(arr);  
println(arr);
```

[0] 1
[1] 2
[2] 3

```
IntList li = new IntList();  
li.append(3);  
li.append(2);  
li.append(1);  
li.sort();  
println(li);
```

IntList size=3 [1, 2, 3]

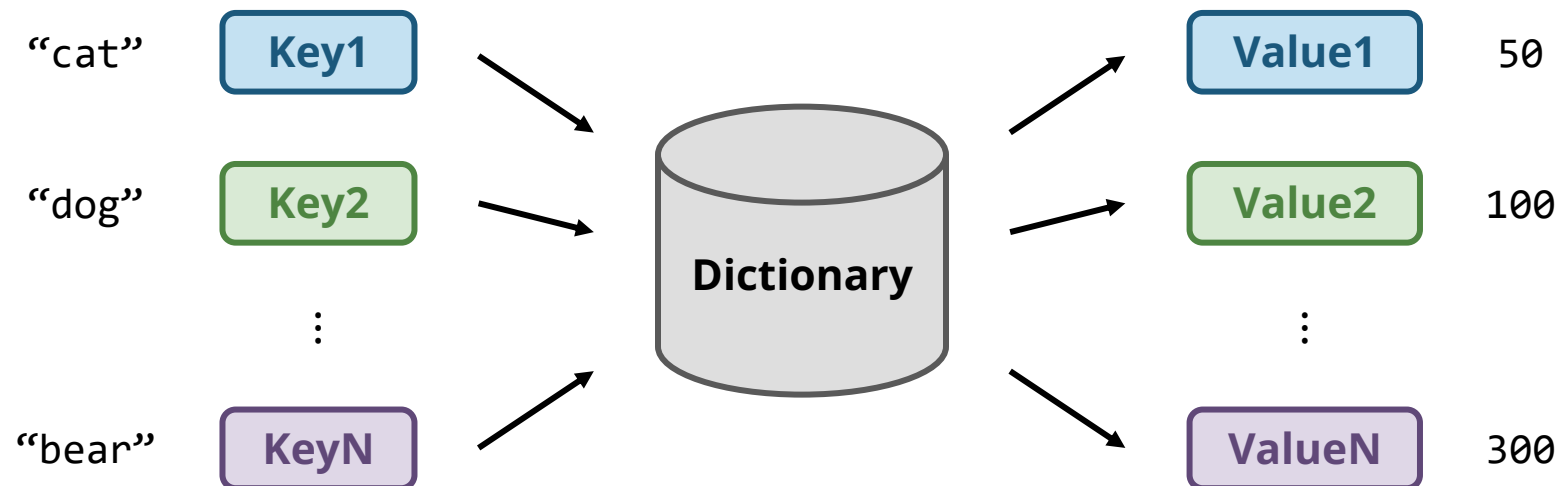
**List.sort() returns the
original list, sorted**

| Arrays vs. Lists

	Arrays	Lists
Size	Fixed	Dynamic
Item data type	Same	Can be different
Access speed	Faster	Slower
Memory requirement	Low	High
Multi-dimensional	Possible	Not supported

Dictionaries

- A dictionary supports **fast lookup** of the corresponding “**value**” for a “**key**”
- **IntDict** String-to-integer mapping
- **FloatDict** String-to-float mapping
- **StringDict** String-to-string mapping



IntDict: String-to-Integer Mapping

`IntDict weightMap = new IntDict();` Declare the shape

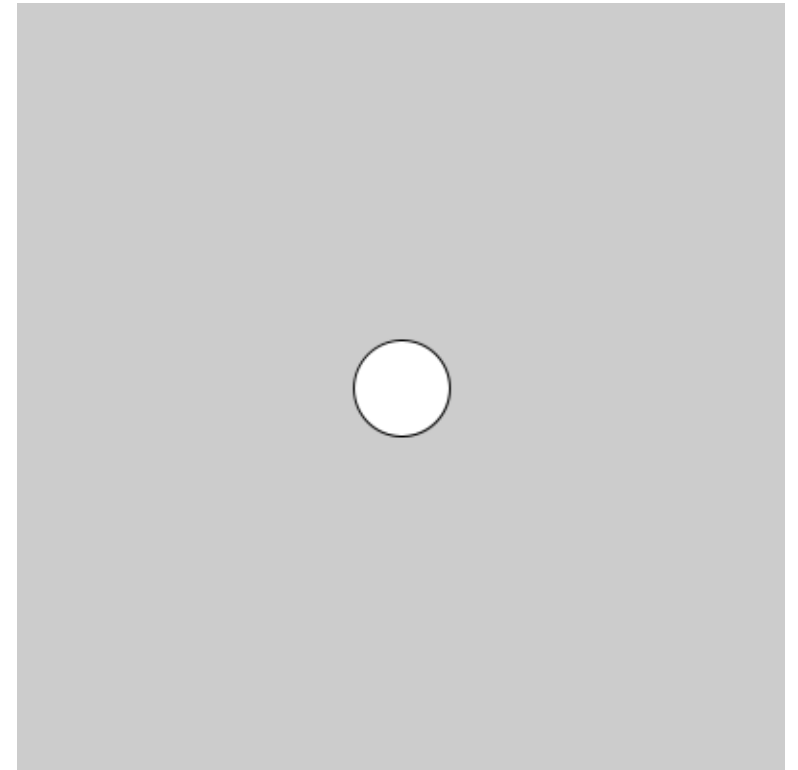
```
void setup() {  
  size(400, 400);  
  noLoop();  
}
```

```
weightMap.set("cat", 50);  
weightMap.set("dog", 100);  
weightMap.set("bear", 300);
```

Set up the dictionary

```
void draw() {  
  int weight = weightMap.get("cat");  
  circle(200, 200, weight);  
}
```

Query the dictionary



A String-to-Array Hash Map

```
HashMap<String, int[]> m;
```

Declare a hash map that maps strings to integer arrays

```
void setup() {  
  size(400, 400);  
  noLoop();  
}
```

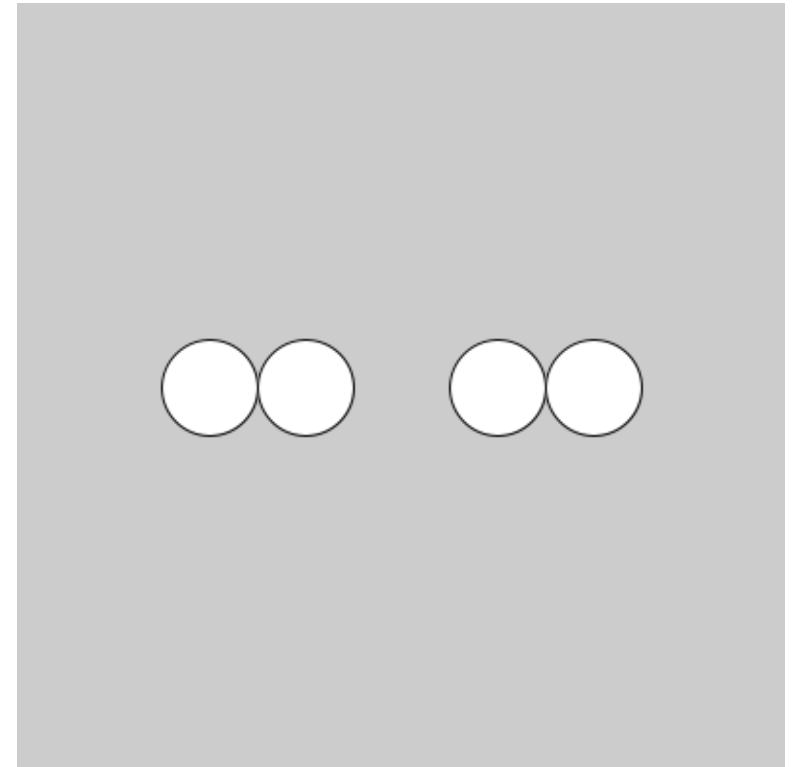
```
m = new HashMap<String, int[]>();  
m.put("cat", new int[]{100, 150, 200});  
m.put("dog", new int[]{100, 300});  
m.put("bear", new int[]{100, 150, 250, 300});
```

Set up the dictionary

```
void draw() {  
  int[] val = m.get("bear");  
  for (int x: val) {  
    circle(x, 200, 50);  
  }  
  println(m);  
}
```

Query the hash map

```
{cat=[I@76cc6ab0, bear=[I@1d88e68, dog=[I@4e0d571f}
```



Next Lecture

Images & Videos

