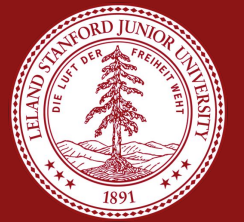
ArrayLists

Lecture 18

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech and others.



Announcements

- Assignment 4 due Monday July 29th at 10AM
- Blank lecture code on website schedule

Learning Goal for Today

Know how to store data in and retrieve data from an

`ArrayList`

Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

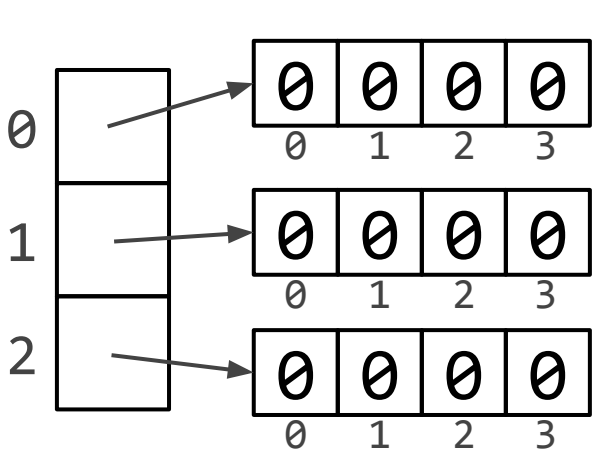
Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

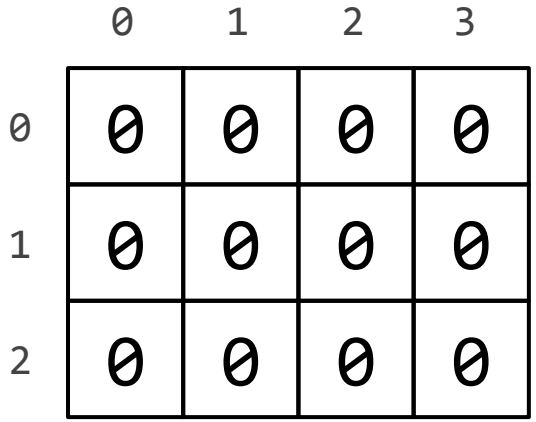
Review 2D: Arrays

Review 2D: Arrays

```
int[][] matrix = new int[3][4];
```



An array of arrays



A unit (ie, a grid/matrix)

Review 2D: Arrays

```
int[][] matrix = new int[3][4];

matrix[row][col];              // get element
matrix[row][col] = value;      // set element
```

Review: 2D Array Dimensions

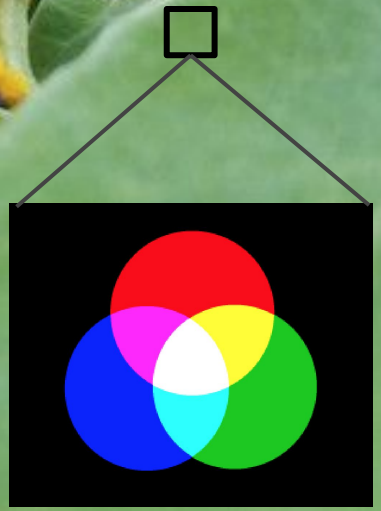
```
private int numRows(int[][] matrix) {
    return matrix.length;
}

private int numCols(int[][] matrix) {
    return matrix[0].length;
}
```

Review: 2D Arrays ❤️ For Loops

- The canonical way to loop over a 2D array is with a double `for` loop

```
type[][] arr = ...
for (int row = 0; row < numRows(arr); row++) {
    for (int col = 0; col < numCols(arr); col++) {
        // do something with arr[row][col] ...
    }
}
```

Review: images are 2D arrays of *pixels*.



GImage Pixel Methods

Method name	Description
`img.getPixelArray()`	returns pixels as 2D array of ints, where each int in the array contains all 3 of Red, Green, and Blue merged into a single integer
`img.setPixelArray(array);`	updates pixels using the given 2D array of ints
`GImage.createRGBPixel(r, g, b)`	returns an int that merges the given amounts of red, green and blue (each 0-255)

`GImage.getRed(px)`
`GImage.getGreen(px)`
`GImage.getBlue(px)`
	returns the redness, greenness, or blueness of the given pixel as an integer from 0-255

Plan for Today

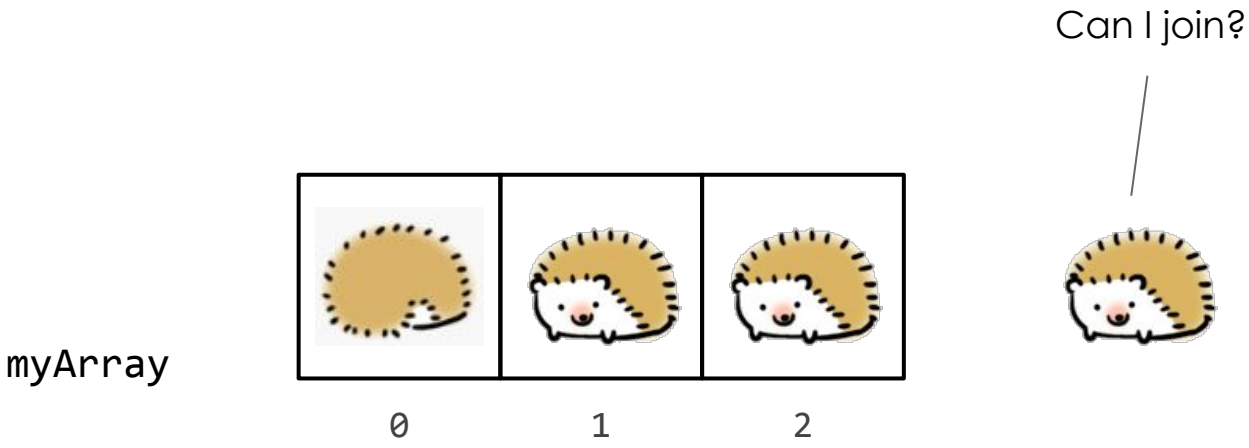
- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

Limitations of Arrays

- Size must be specified upon creation
- Can't add/remove/insert elements later (because size is fixed)

Limitations of Arrays

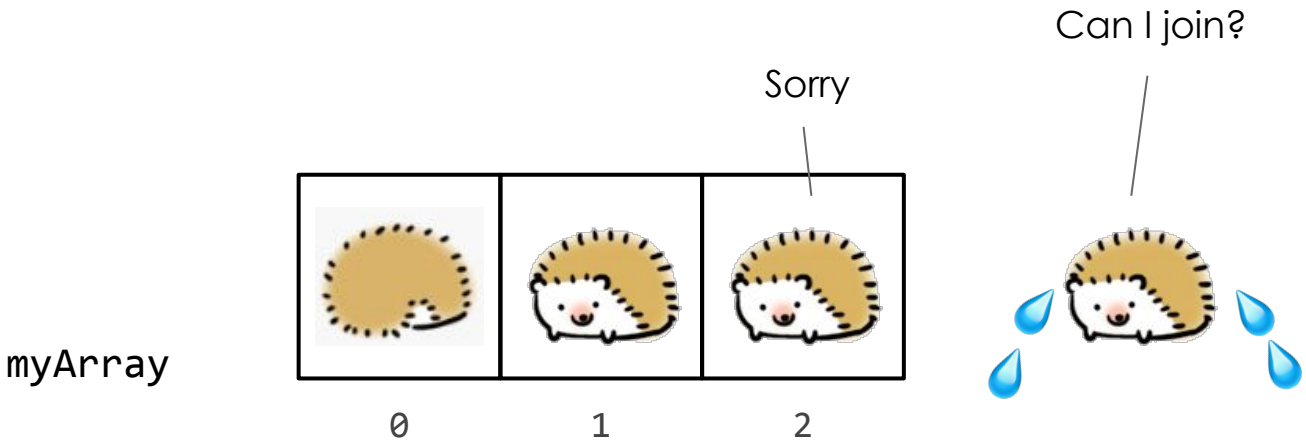
- Size must be specified upon creation
- Can't add/remove/insert elements later (because size is fixed)



hedgehog icon from Slack

Limitations of Arrays

- Size must be specified upon creation
- Can't add/remove/insert elements later (because size is fixed)



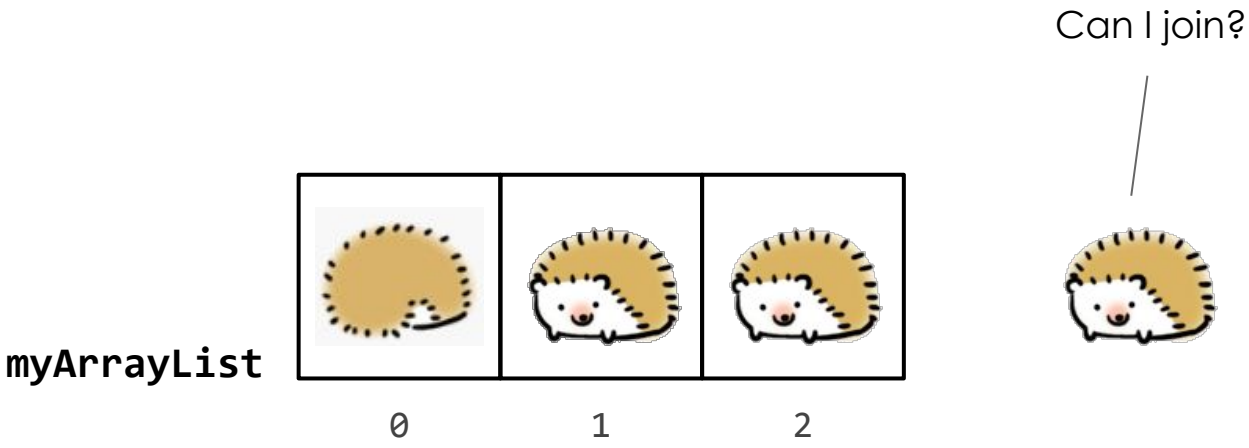
How can we help  ?

Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)

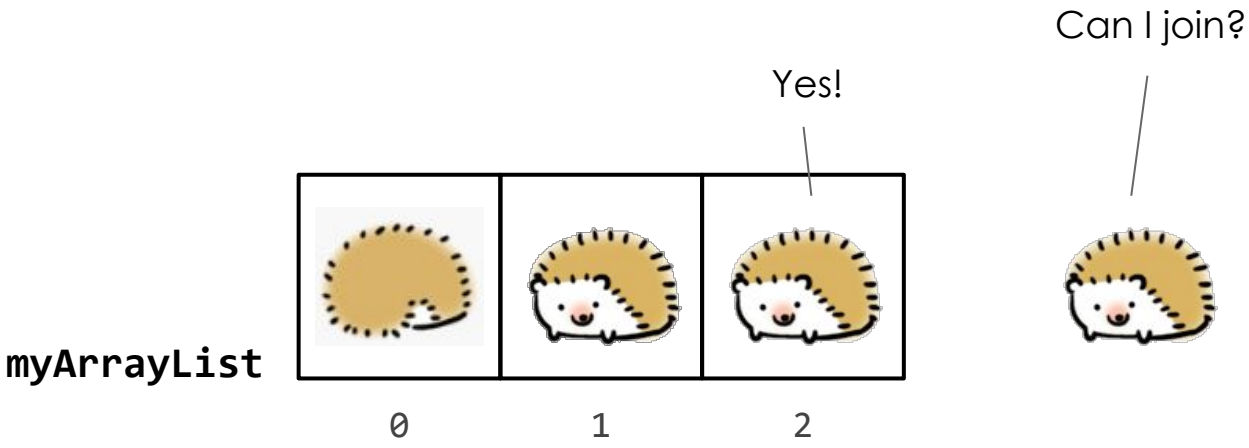
Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



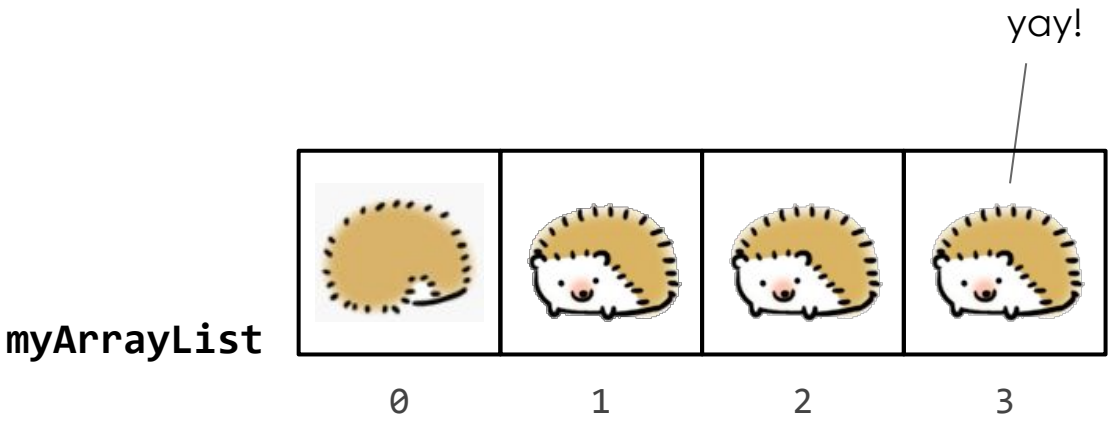
Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



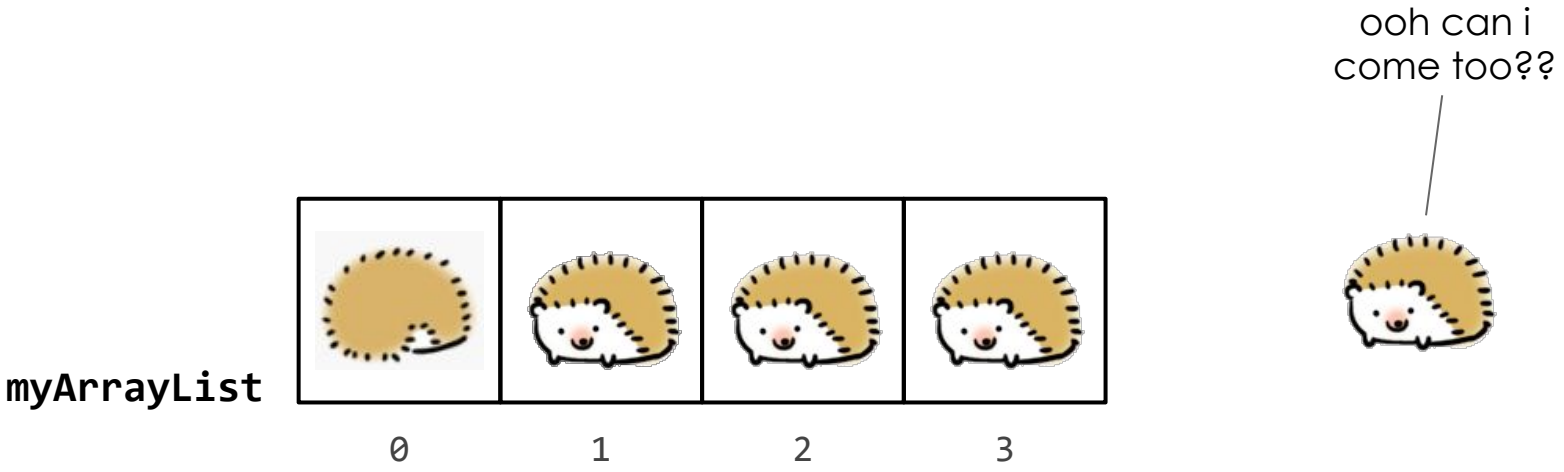
Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



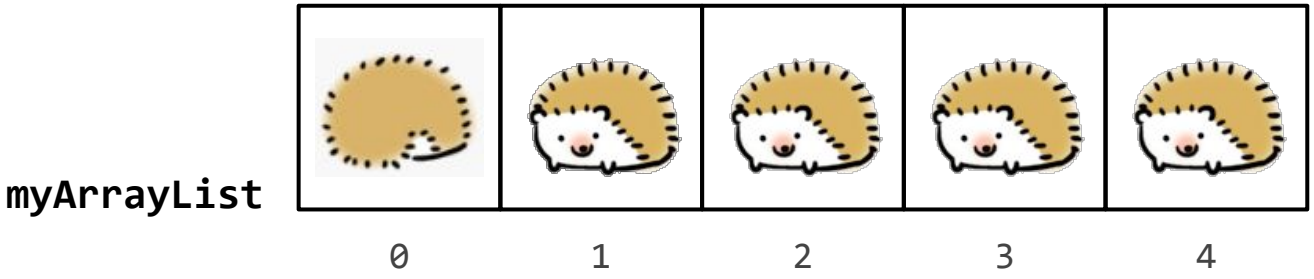
Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



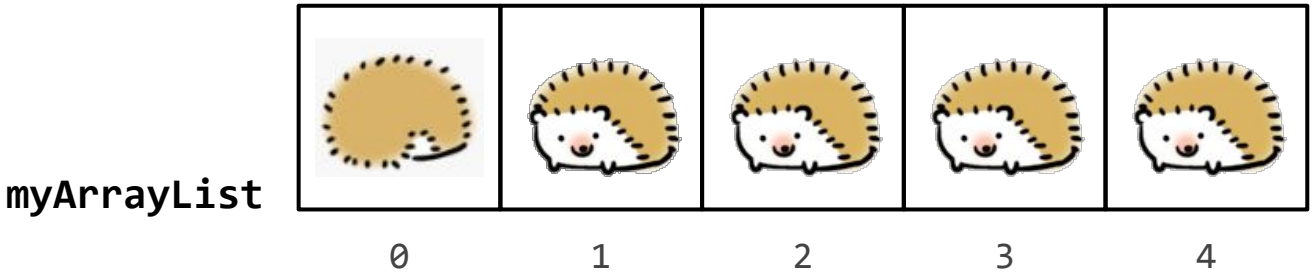
Introducing ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)



ArrayLists

- An ordered, *resizable* list of information
- Can add and remove elements (among other cool functionality)
- Homogenous
- Can store any **Object** type
- Access individual items by *index*



Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

```
import java.util.*;
```

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

Type of thing your ArrayList will store

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

Same type here.

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

Our First ArrayList

Can optionally leave empty because of type inference

```
ArrayList<String> myArrayList = new ArrayList<>();
```

Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();
```

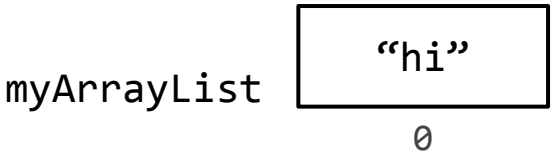
Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty
```

myArrayList

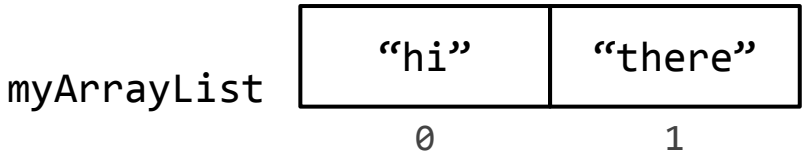
Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
```



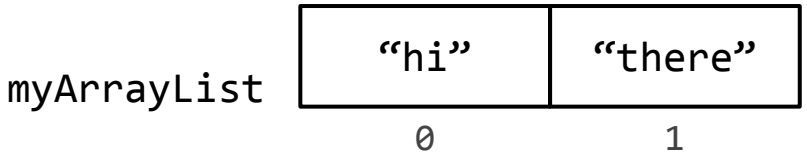
Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");
```



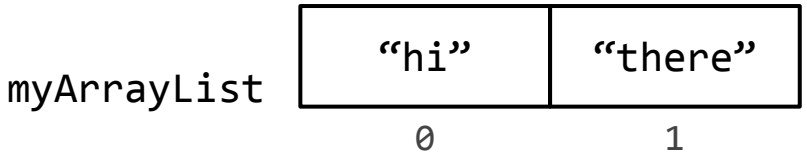
Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0));   // prints "hi"
println(myArrayList.get(1));   // prints "there"
```



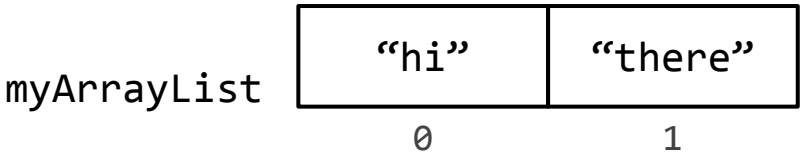
Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0));   // prints "hi"
println(myArrayList.get(1));   // prints "there"

// Wrong type - bad times! Won't compile
GLabel label = new GLabel("hi there");
myArrayList.add(label);
```



Our First ArrayList

```
ArrayList<String> myArrayList = new ArrayList<String>();  // initially empty

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
println(myArrayList.get(0));   // prints "hi"
println(myArrayList.get(1));   // prints "there"

// Wrong type - bad times! Won't compile
GLabel label = new GLabel("hi there");
myArrayList.add(label);

// Invalid index – crashes! IndexOutOfBounds Exception
println(myArrayList.get(2));
```



Looping over ArrayLists

```
ArrayList<String> myArrayList = new ArrayList<String>();

// Adds elements to the back
myArrayList.add(“hi”);
myArrayList.add(“there”);

// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
     String str = myArrayList.get(i);
     println(str);
}

// hi
// there
```

Looping over ArrayLists

```
ArrayList<String> myArrayList = new ArrayList<String>();

// Adds elements to the back
myArrayList.add("hi");
myArrayList.add("there");

// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
     String str = myArrayList.get(i);
     println(str);
}

// A beautiful way to access each element
for (String str : myArrayList) {
     println(str);
}
```

Looping over ArrayLists

```
// Access elements by index (starting at 0!)
for (int i = 0; i < myArrayList.size(); i++) {
     String str = myArrayList.get(i);
     println(str);
}

// A beautiful way to access each element
for (String str : myArrayList) {
     println(str);
}
```

ArrayList Methods

`boolean add(<T> element)` Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.
`void add(int index, <T> element)` Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.
`<T> remove(int index)` Removes the element at the specified position and returns that value.
`boolean remove(<T> element)` Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.
`void clear()` Removes all elements from the **`ArrayList`**.
`int size()` Returns the number of elements in the **`ArrayList`**.
`<T> get(int index)` Returns the object at the specified index.
`<T> set(int index, <T> value)` Sets the element at the specified index to the new value and returns the old value.
`int indexOf(<T> value)` Returns the index of the first occurrence of the specified value, or −1 if it does not appear.
`boolean contains(<T> value)` Returns **`true`** if the **`ArrayList`** contains the specified value.
`boolean isEmpty()` Returns **`true`** if the **`ArrayList`** contains no elements.

`<T>` ?
This means, whatever **T**ype your ArrayList stores

/

Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

Example: Reversible Writing

Let's write a program that reverses a text file.

I am not a person who contributes
And I refuse to believe that
I will be useful

Example: Reversible Writing

Let's write a program that reverses a text file.

I am not a person who contributes
And I refuse to believe that
I will be useful

I will be useful
And I refuse to believe that
I am not a person who contributes

"I Have a Dream" by Antonia Lee, Sara Fung, Christy Fung, Rachel Lam
http://poets.spice.org.hk/index.php?option=com_content&view=article&id=45:my-family&catid=6:reversepoem&Itemid=7

Example: Reversible Writing

Let's write a program that reverses a text file.

"I am not a person who contributes"

Example: Reversible Writing

Let's write a program that reverses a text file.

"I am not a person who contributes"
"And I refuse to believe that"

Example: Reversible Writing

Let's write a program that reverses a text file.

"I am not a person who contributes"
"And I refuse to believe that"
"I will be useful"

Example: Reversible Writing

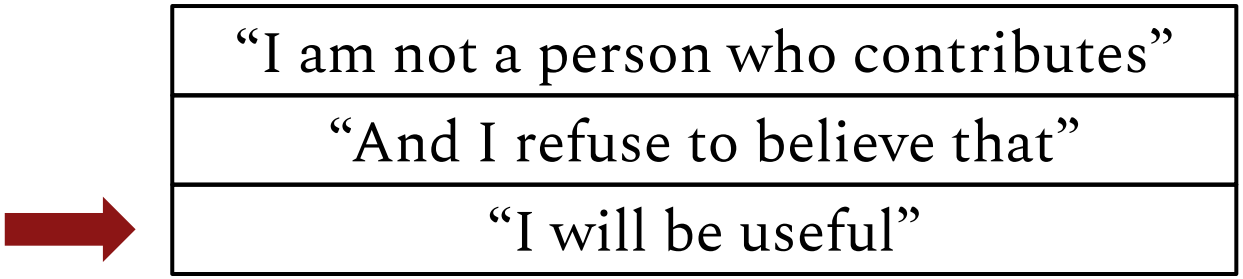
Let's write a program that reverses a text file.

"I am not a person who contributes"
"And I refuse to believe that"
"I will be useful"

Key Idea # 1: fill an `ArrayList` with each line in the file.

Example: Reversible Writing

Let's write a program that reverses a text file.



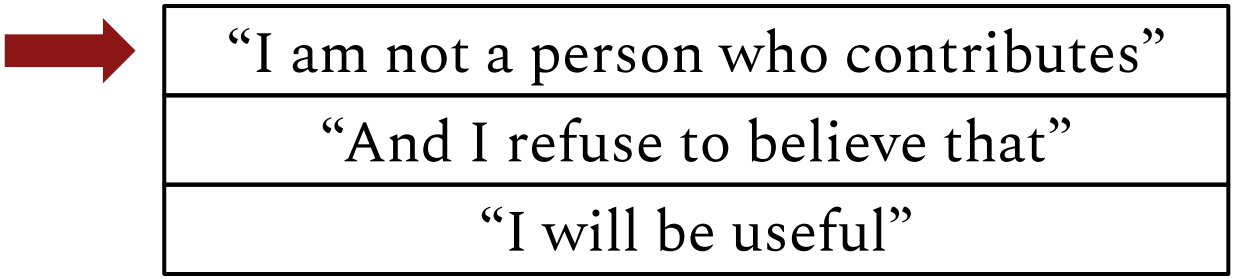
Example: Reversible Writing

Let's write a program that reverses a text file.

"I am not a person who contributes"
"And I refuse to believe that"
"I will be useful"

Example: Reversible Writing

Let's write a program that reverses a text file.



Example: Reversible Writing

Let's write a program that reverses a text file.

"I am not a person who contributes"
"And I refuse to believe that"
"I will be useful"

Key Idea # 2: print the `ArrayList` items in reverse order.

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Example: Reversible Writing

```
try {
    Scanner scanner = new Scanner(new File(FILENAME));
    ArrayList<String> lines = new ArrayList<String>();

    // Read all lines and store in our ArrayList
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }

    // Output the lines from back to front
    for (int i = lines.size() - 1; i >= 0; i--) {
        println(lines.get(i));
    }
    scanner.close();
} catch (IOException ex) {
    println("Could not read file.");
}
```

Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

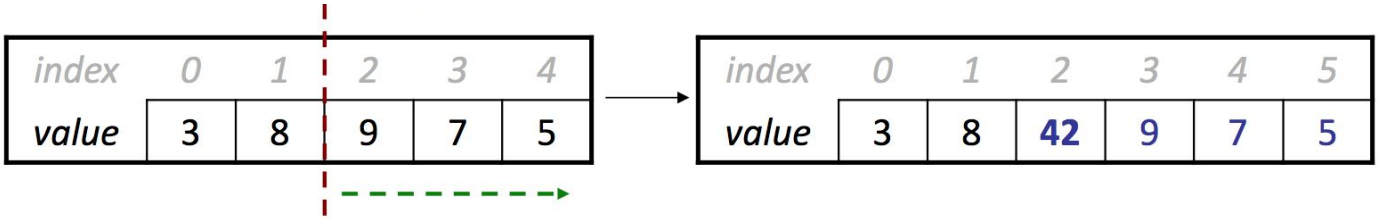
A Note on Insert/Remove

- If you insert or remove an element from a list, any elements to the right of it shift to fit

A Note on Insert/Remove

- If you insert or remove an element from a list, any elements to the right of it shift to fit

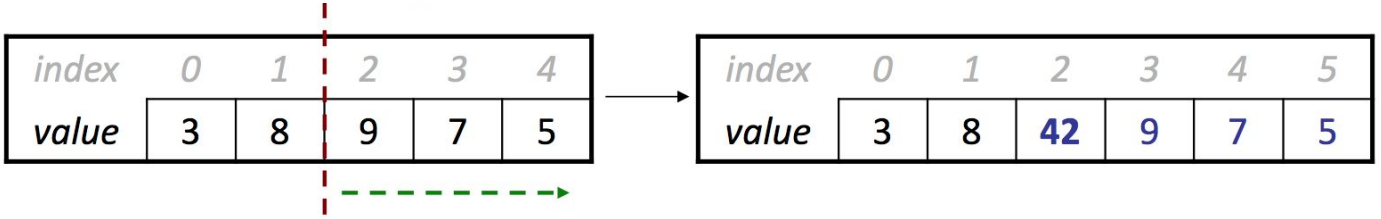
```
list.add(2, 42); // add the value 42 before index 2
```



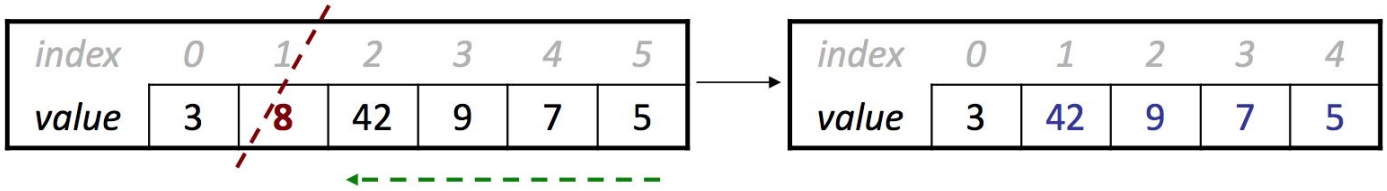
A Note on Insert/Remove

- If you insert or remove an element from a list, any elements to the right of it shift to fit

```
list.add(2, 42); // add the value 42 before index 2
```



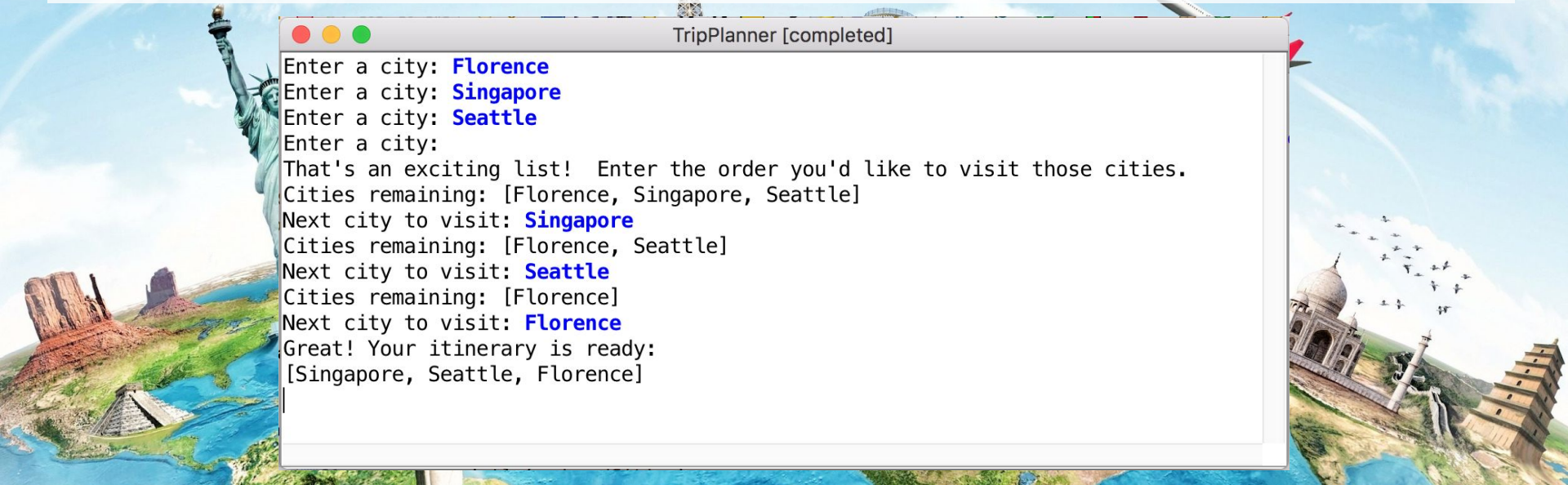

```
list.remove(1); // remove the element at index 1
```



Example: Trip Planner

It's summer, and you want to travel! Let's write a program to plan our itinerary.

- Program first prompts the user for all the cities they want to visit
- Then, it asks user to re-enter them in the order they'd like to visit them
- Finally, outputs the itinerary: the order in which to visit the cities



Trip Planner: Approach

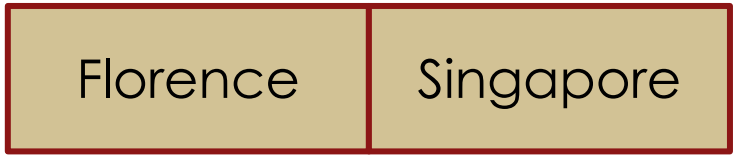
Cities:

Florence

Order:

Trip Planner: Approach

Cities:



Order:

Trip Planner: Approach

Cities:

Order:

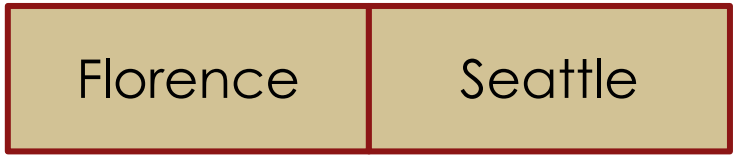
Trip Planner: Approach

Cities:

Order:

Trip Planner: Approach

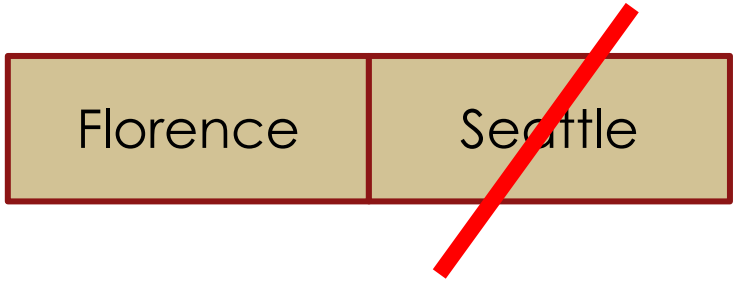
Cities:



Order:

Trip Planner: Approach

Cities:

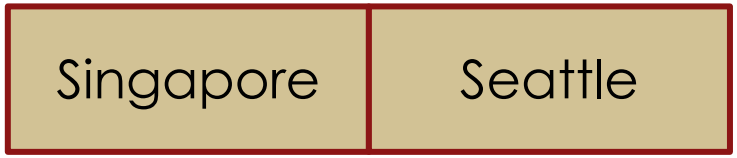


Order:

Trip Planner: Approach

Cities:

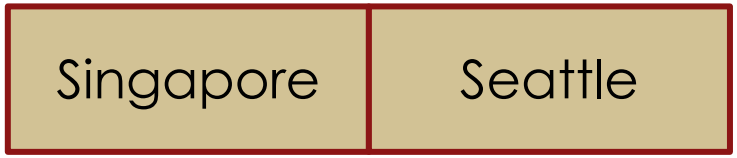
Order:



Trip Planner: Approach

Cities:

Order:



Trip Planner: Approach

Cities: Done!

Order:

Let’s Code It!

Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

ArrayLists + Primitives

```
// Doesn't compile ~ sad times :(
ArrayList<int> myArrayList = new ArrayList<int>();
```

Unlike Arrays, ArrayLists can only store **Objects**.

Wrapper Classes

Primitive	"Wrapper" Class
`int`	`Integer`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`

ArrayLists + Primitives

```
// Use wrapper classes when making an ArrayList
ArrayList<Integer> numList = new ArrayList<Integer>();

// Java converts Integer <-> int automatically!
numList.add(22);
numList.add(44);

int firstNum = numList.get(0);  // 22
int secondNum = numList.get(1); // 44
```

Conversion happens automatically!

An Example

Let's check out
an example

Arrays vs. ArrayLists

Operation	Arrays	ArrayLists
Make a new one	`int arr = new int[5];`	`ArrayList<String> list = new ArrayList<String>();`
Length?	`arr.length`	`list.size()`
Get element?	`arr[i]`	`list.get(i)`
Set element?	`arr[i] = value`	`list.set(i, value)`
Loop?	`for(int i = 0; i < arr.length; i++)`	`for(String value : list)`

Array vs. ArrayLists

Why do both of these exist in the language?

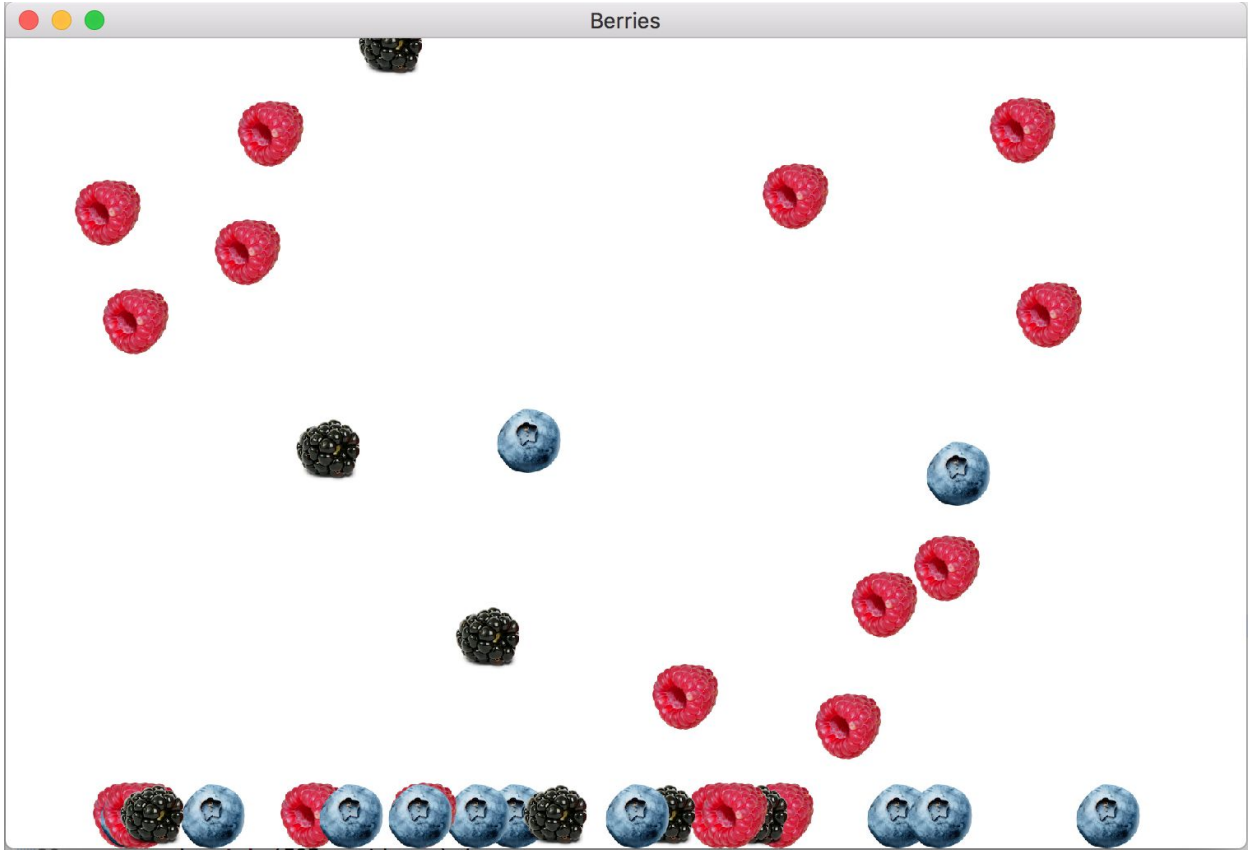
- Arrays are Java's fundamental data storage
- ArrayList is a library built on top of an array

When would you choose an array over an ArrayList?

- When you need a fixed size that you know ahead of time
  - Simpler syntax for getting/setting, more efficient
- Multi-dimensional arrays (e.g. images)
- Histograms/tallying

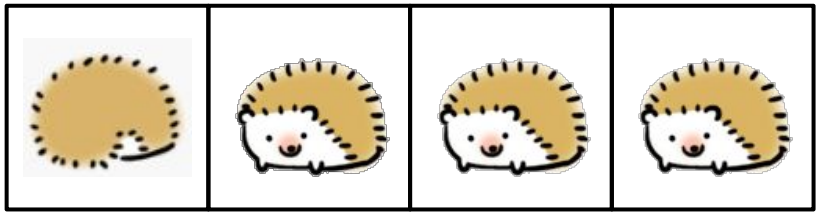
[Extra Practice] Picking Berries

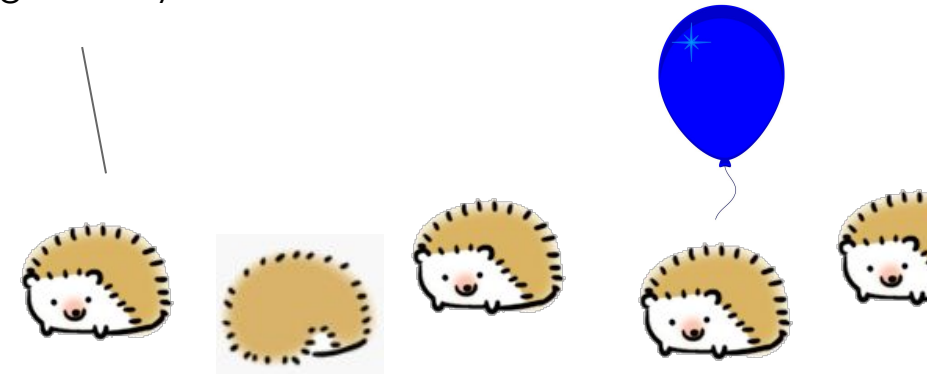
[Extra Practice] Picking Berries



When you don't know how many  are coming to the party

I love ArrayLists!!
I brought all my friends





hedgehogPartyList

Plan for Today

- Review: 2D Arrays
- ArrayLists
- Example: Reversible Writing
- Example: Trip Planner
- ArrayLists vs. Arrays

Next Time: HashMaps!