Lecture 5 (Lists 3)

DLLists and Arrays

CS61B, Spring 2024 @ UC Berkeley

Slides credit: Josh Hug

Summary of SLLists So Far

Lecture 5, CS61B, Spring 2024

SLLists:

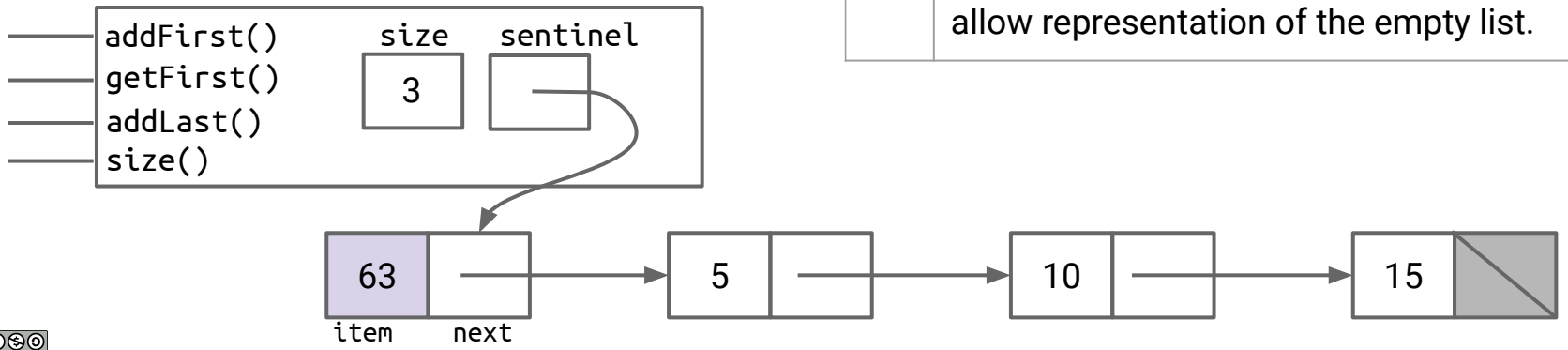
- **Summary of SLLists So Far**
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- Generic Lists

Arrays:

- Array Overview
- Basic Array Syntax
- 2D Arrays
- Arrays vs. Classes

Summary of Last Time (From IntList to SLList)

Methods	Non-Obvious Improvements	
`addFirst(int x)`	#1	Rebranding: `IntList` → `IntNode`
`getFirst()`	#2	Bureaucracy: `SLList`
`addLast(int x)`	#3	Access Control: `public` → `private`
`size()`	#4	Nested Class: Bringing `IntNode` into `SLList`
	#5	Caching: Saving `size` as an `int`.
	#6	Generalizing: Adding a `sentinel` node to allow representation of the empty list.



One Downside of SLLists

Inserting at the back of an `SLList` is much slower than the front.

```
public void addFirst(int x) {
   sentinel.next = new IntNode(x, sentinel.next);
}
```

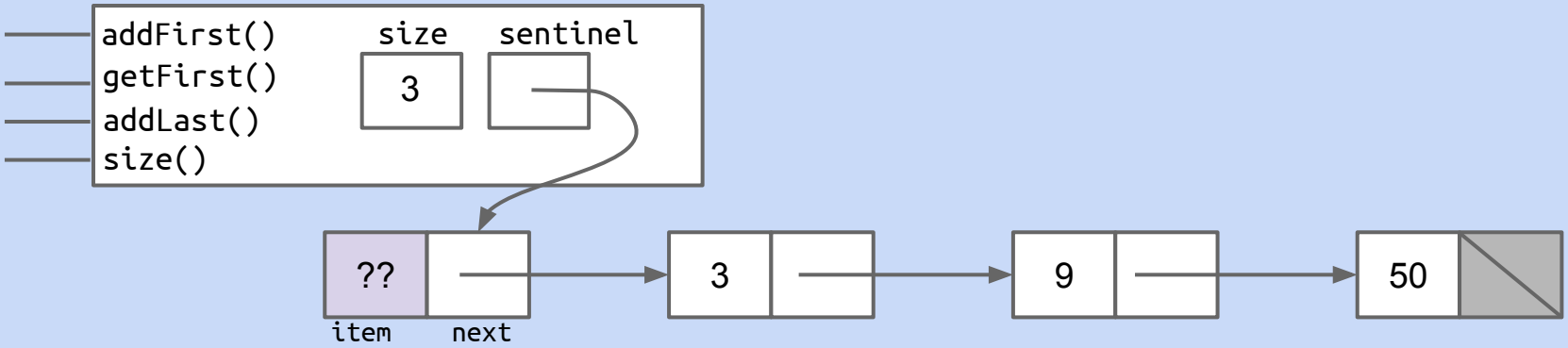
```
public void addLast(int x) {
   size += 1;

   IntNode p = sentinel;
   while (p.next != null) {
      p = p.next;
   }

   p.next = new IntNode(x, null);
}
```

Improvement #7: (???) Goal: Fast `addLast`

How could we modify our list data structure so that `addLast` is also fast?



Why a Last Pointer Isn’t Enough

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- **Why a Last Pointer Isn’t Enough**
- Doubly Linked Lists
- Generic Lists

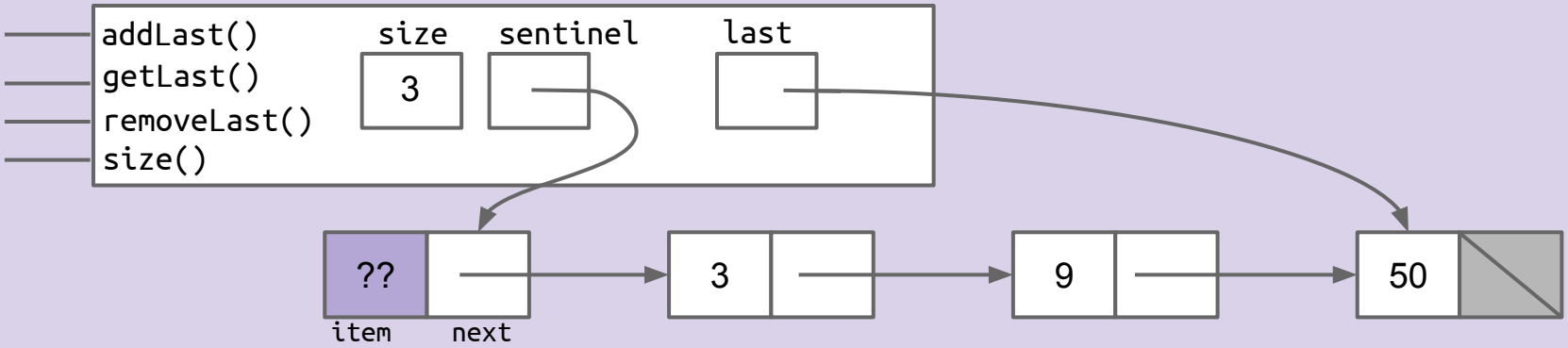
Arrays:

- Array Overview
- Basic Array Syntax
- 2D Arrays
- Arrays vs. Classes

Is .last enough? http://yellkey.com/camera

Suppose we want to support **add**, **get**, and **remove** operations for both ends, will having a `last` pointer result for fast operations on long lists?

A. Yes
B. No, add would be slow.
C. No, get would be slow.
D. No, remove would be slow.



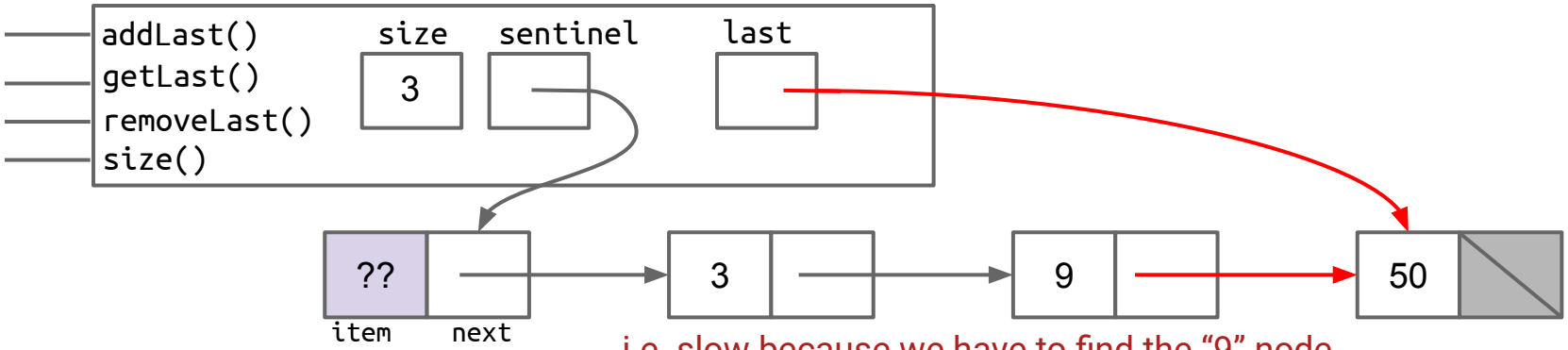
`.last` Is Not Enough

Suppose we want to support add, get, and remove operations, will having a `last` pointer result for fast operations on long lists?

- **No, remove would be slow.**

RemoveLast requires setting 9's next pointer to null, and point last at the 9 node.

- Have to search through list to find the 9 node (second to last).



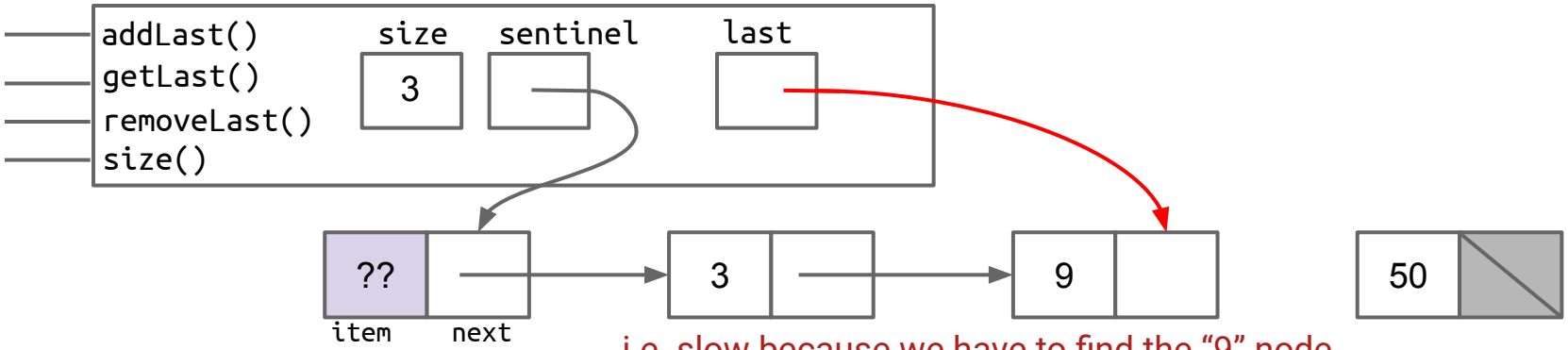
`.last` Is Not Enough

Suppose we want to support add, get, and remove operations, will having a `last` pointer result for fast operations on long lists?

- **No, remove would be slow.**

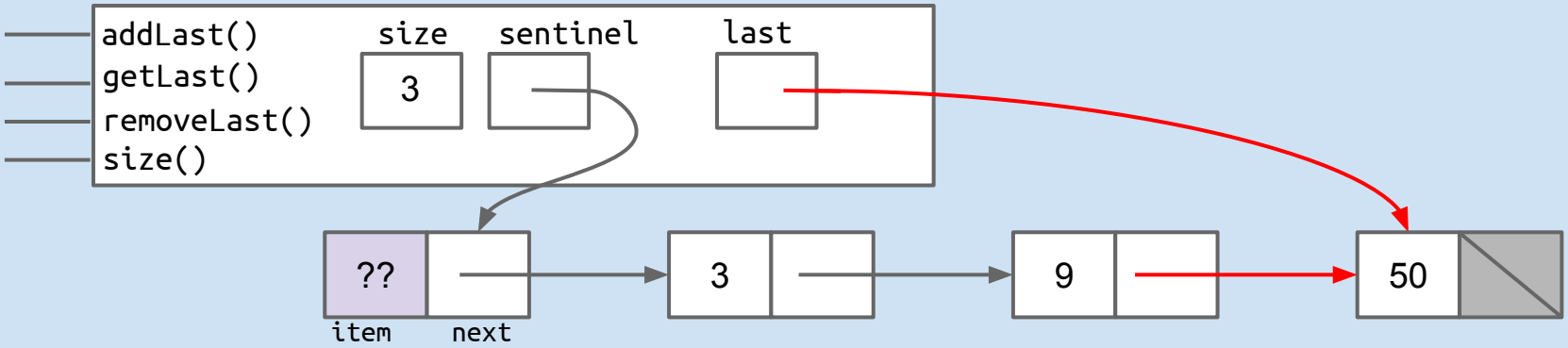
RemoveLast requires setting 9's next pointer to null, and point last at the 9 node.

- Have to search through list to find the 9 node (second to last).



Improvement #7: .last and ??? Goal: Fast operations on last.

We added .last. What other changes might we make so that remove is also fast?



Doubly Linked Lists

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- **Doubly Linked Lists**
- Generic Lists

Arrays:

- Array Overview
- Basic Array Syntax
- 2D Arrays
- Arrays vs. Classes

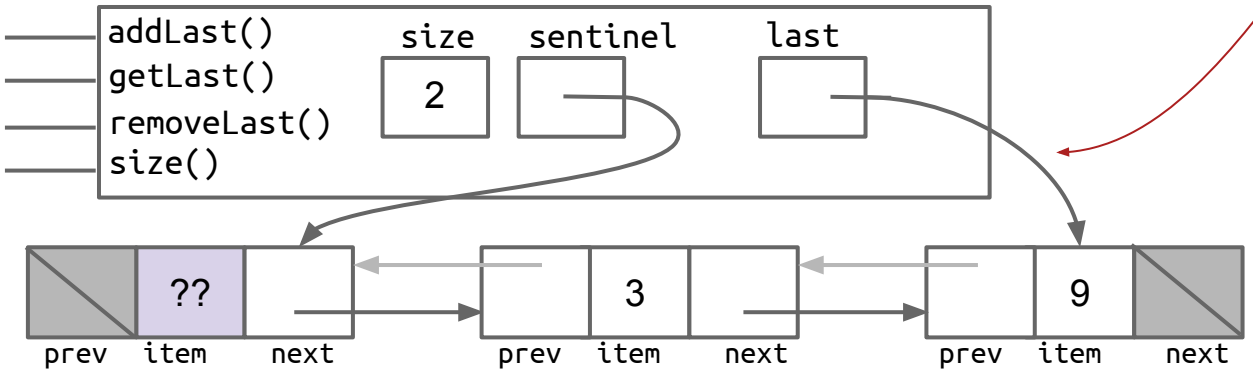
Improvement #7: .last and .prev

We added .last. What other changes might we make so that remove is also fast?

- Add backwards links from every node.
- This yields a "**doubly linked list**" or `DLList`, as opposed to our earlier "**singly linked list**" or `SLList`.

Note: Arrows point at entire nodes, not fields!

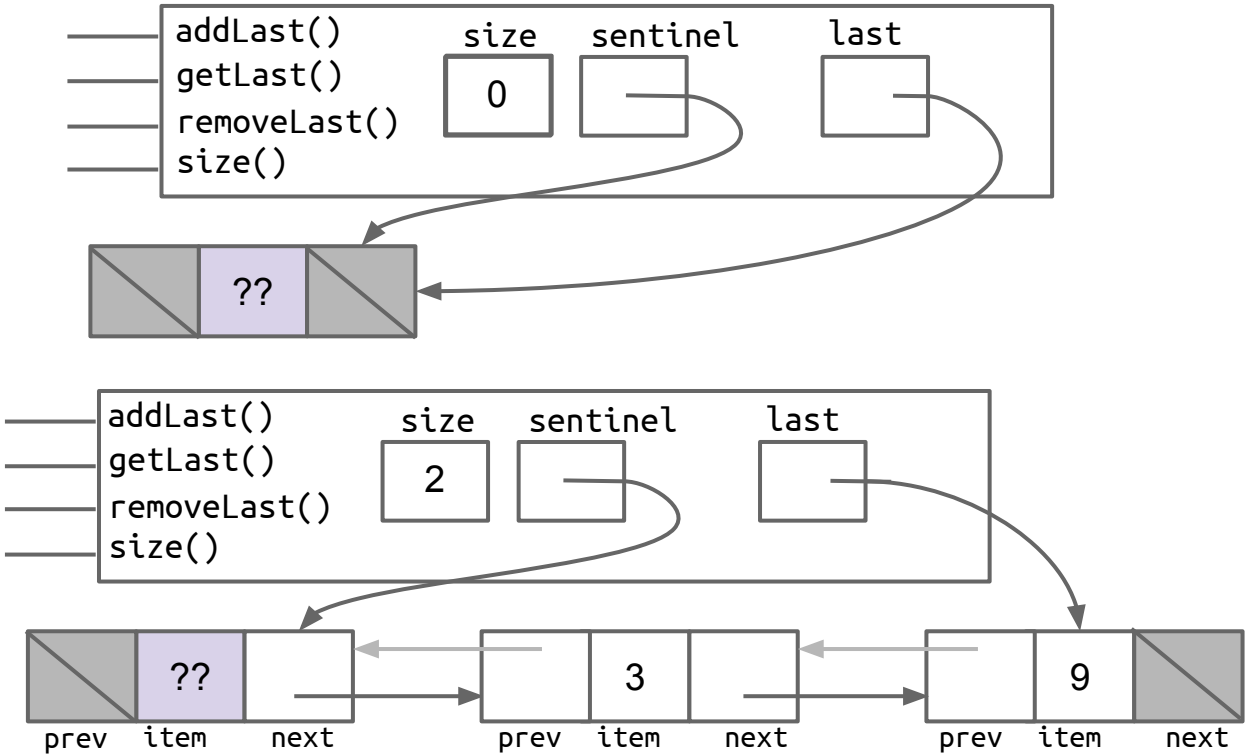
Example: last holds the address of the last node, not the item field of the sentinel node.



Doubly Linked Lists (Naive)

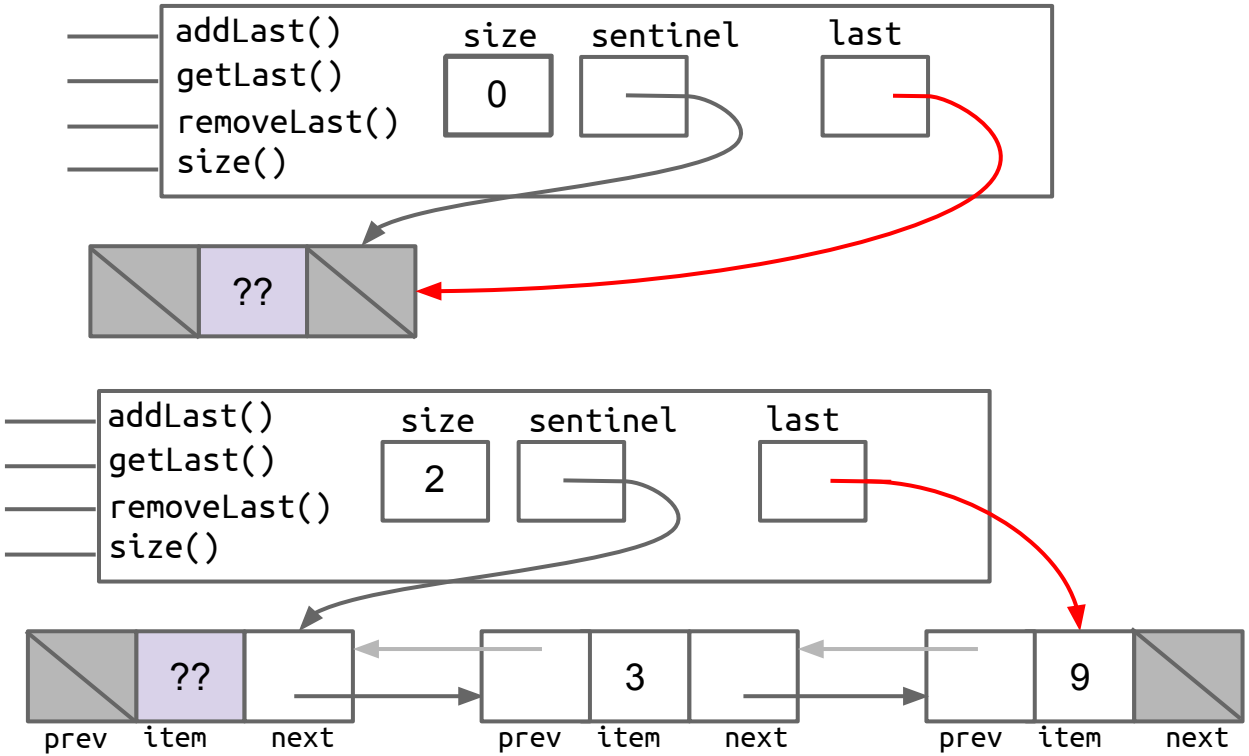
Reverse pointers allow all operations (add, get, remove) to be fast.

- We call such a list a “doubly linked list” or `DLList`.



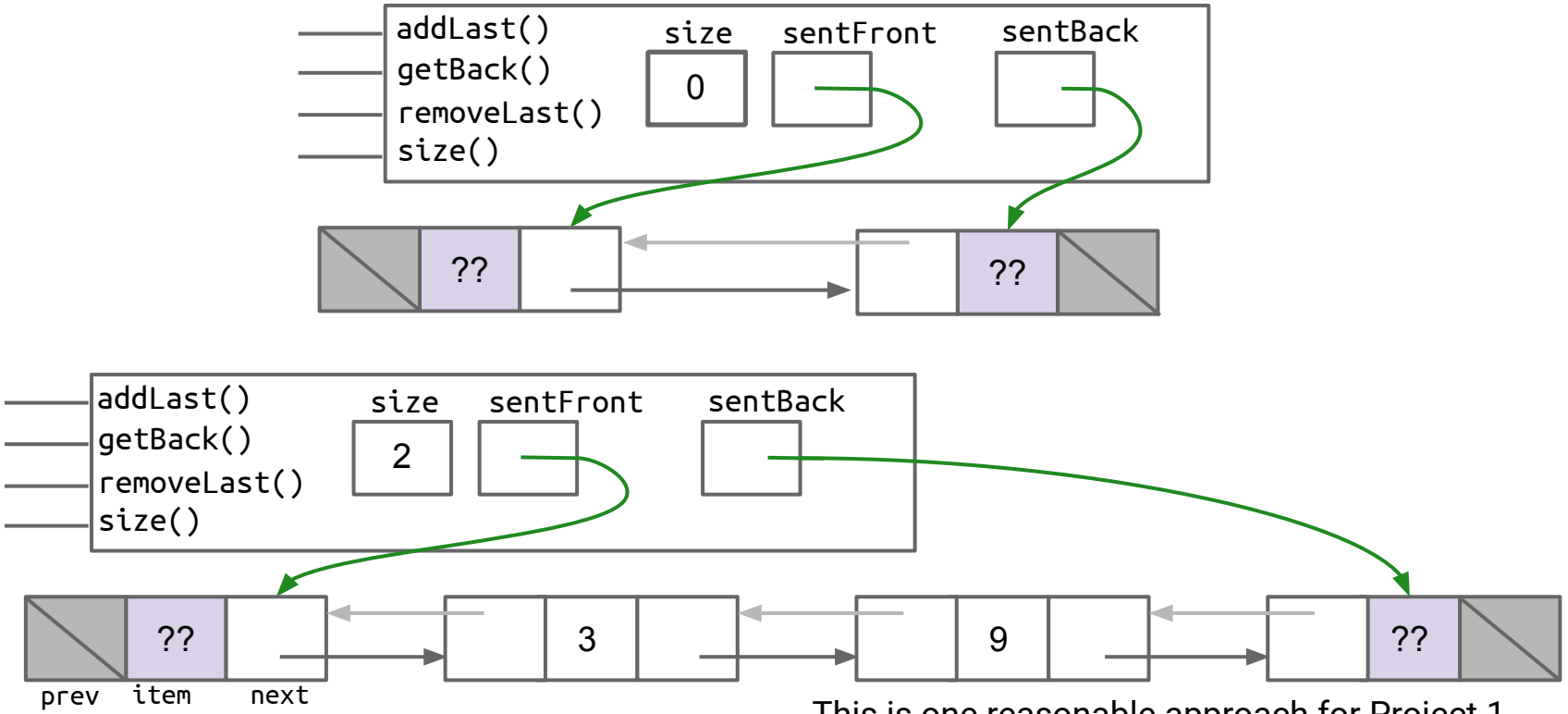
Doubly Linked Lists (Naive)

Non-obvious fact: This approach has an annoying special case: `last` sometimes points at the sentinel, and sometimes points at a 'real' node.



Doubly Linked Lists (Double Sentinel)

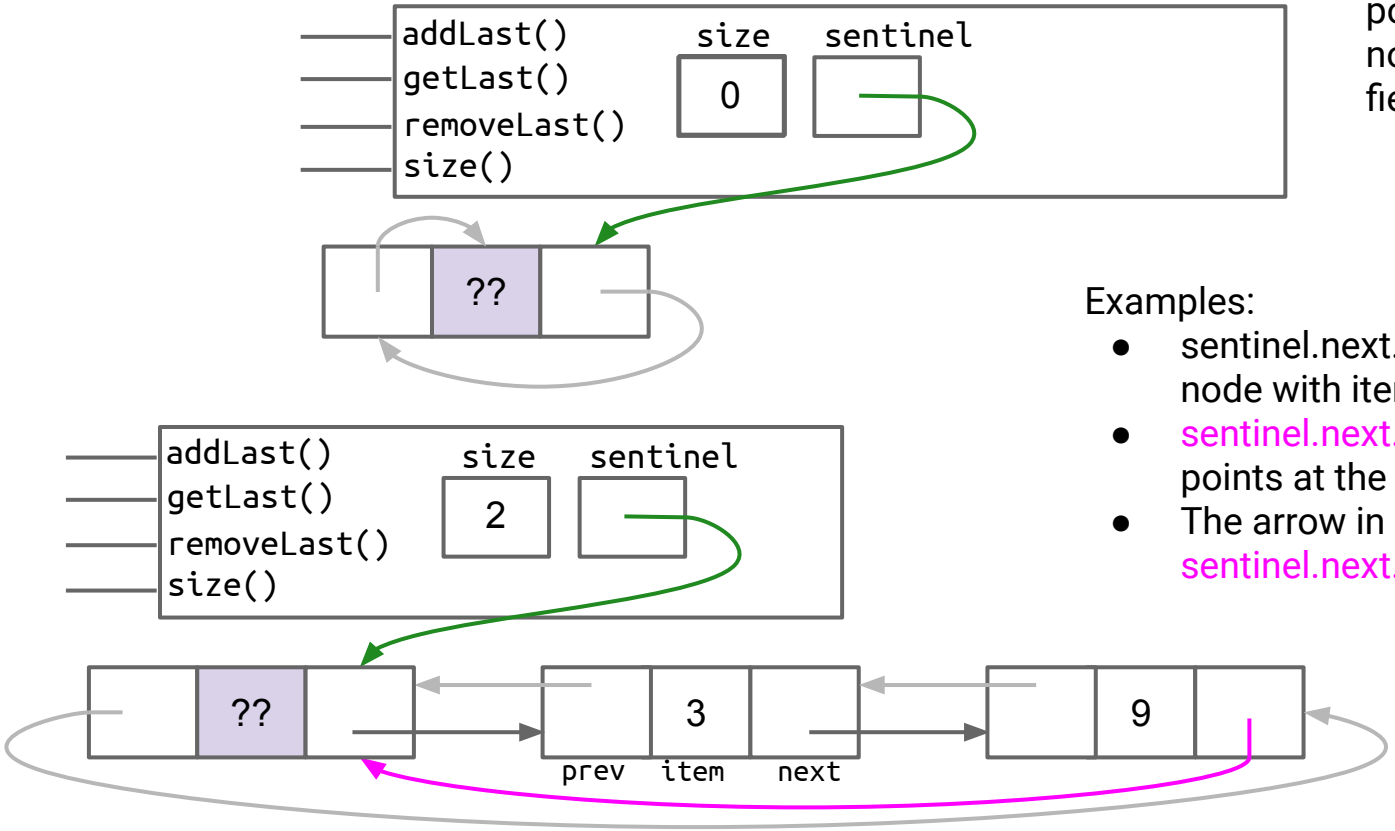
One solution: Have two sentinels.



This is one reasonable approach for Project 1.

Doubly Linked Lists (Circular Sentinel)

Even better topology (IMO):



Note: arrows are pointing at entire nodes, not specific fields of nodes.

Examples:

- sentinel.next.next is the node with item=9.
- sentinel.next.next.next points at the sentinel node.
- The arrow in magenta is sentinel.next.next.next

This is my preferred approach for Project 1.

Improvement #8: Fancier Sentinel Node(s)

While fast, adding `.last` and `.prev` introduces lots of special cases.

To avoid these, either:

- Add an additional `sentBack` sentinel at the end of the list.
- Make your linked list circular (highly recommended for project 1), with a single sentinel in the middle.

DLList Summary

Methods	Non-Obvious Improvements	
`addFirst(int x)`	#1	Rebranding: `IntList` → `IntNode`
`getFirst()`	#2	Bureaucracy: `SLList`
`size()`	#3	Access Control: `public` → `private`
`addLast(int x)`	#4	Nested Class: Bringing `IntNode` into `SLList`
`removeLast()`	#5	Caching: Saving `size` as an `int`.
	#6	Generalizing: Adding a `sentinel` node to allow representation of the empty list.
	#7	Looking back: `.last` and `.prev` allow fast `removeLast`
	#8	Sentinel upgrade: Avoiding special cases with `sentBack` or circular list.

Still many steps before we have an industrial strength data structure. Will discuss over coming weeks.

Generic Lists

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- **Generic Lists**

Arrays:

- Array Overview
- Basic Array Syntax
- 2D Arrays
- Arrays vs. Classes

Integer Only Lists

One issue with our list classes: They only support integers.

```
public class SLList {
   private IntNode sentinel;
   private int size;


   public class IntNode {
      public int item;
      public IntNode next;
      ...
   }
   ...
}
```

```
SLList s1 = new SLList(5);
s1.addFirst(10);
```

Works fine!

```
SLList s2 = new SLList("hi");
s2.addFirst("apple");
```

```
SLListLauncher.java:6: error:
incompatible types: String cannot
be converted to int

SLList s2 = new SLList("hi");
```

Coding Demo: Generic Lists

SLList.java

```
public class SLList {
   private IntNode sentinel;
   private int size;

   private class IntNode {
      public int item;
      public IntNode next;

      public IntNode(int i, IntNode n) {
         item = i;
         next = n;
      }
   }



}
```

In this demo, we'll modify our SLList to support lists of any data type, not just lists of integers.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private IntNode sentinel;
   private int size;

   private class IntNode {
      public int item;
      public IntNode next;

      public IntNode(int i, IntNode n) {
         item = i;
         next = n;
      }
   }



}
```

A placeholder name, which will get replaced by the true data type each time a new SLList is created.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private IntNode sentinel;
   private int size;

   private class IntNode {
      public LochNess item;
      public IntNode next;

      public IntNode(LochNess i, IntNode n) {
         item = i;
         next = n;
      }
   }



}
```

Items are no longer integers, but the LochNess placeholder data type.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private StuffNode sentinel;
   private int size;

   private class StuffNode {
      public LochNess item;
      public StuffNode next;

      public StuffNode(LochNess i, StuffNode n) {
         item = i;
         next = n;
      }
   }



}
```

Renaming IntNode to StuffNode to be more descriptive.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private StuffNode sentinel;
   private int size;

   public SLList(LochNess x) {
      sentinel = new StuffNode(null, null);
      sentinel.next = new StuffNode(x, null);
      size = 1;
   }

   public SLList() {
      sentinel = new StuffNode(null, null);
      size = 0;
   }


}
```

Replaced int x with LochNess x, the placeholder data type.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private StuffNode sentinel;
   private int size;

   public void addFirst(LochNess x) {
      sentinel.next = new StuffNode(x, sentinel.next);
      size += 1;
   }

   public LochNess getFirst() {
      return sentinel.next.item;
   }




}
```

Replaced int x with LochNess x, the placeholder data type.

Return type is LochNess, not int.

Coding Demo: Generic Lists

SLList.java

```
public class SLList<LochNess> {
   private StuffNode sentinel;
   private int size;

   public void addLast(LochNess x) {
      size += 1;
      StuffNode p = sentinel;

      /** Move p until it reaches the end of the list. */
      while (p.next != null) {
         p = p.next;
      }
      p.next = new StuffNode(x, null);
   }


}
```

Replaced int x with LochNess x, the placeholder data type.

SLists

Java allows us to defer type selection until declaration.

```
public class SLList<BleepBlorp> {
   private IntNode sentinel;
   private int size;

   public class IntNode {
      public BleepBlorp item;
      public IntNode next;
      ...
   }

   ...
}
```

```
SLList<Integer> s1 = new SLList<>(5);
s1.addFirst(10);

SLList<String> s2 = new SLList<>("hi");
s2.addFirst("apple");
```

Generics

We'll spend a lot more time with generics later, but here are the rules of thumb you'll need for project 1:

- In the .java file **implementing** your data structure, specify your "generic type" **only once** at the very top of the file.
- In .java files that **use** your data structure, specify desired type **once**:
  - Write out desired type during **declaration**.
  - Use the empty diamond operator <> during **instantiation**.
- When declaring or instantiating your data structure, use the reference type.
  - int: Integer
  - double: Double
  - char: Character
  - boolean: Boolean
  - long: Long
  - etc.

```
DLList<Double> s1 = new DLList<>(5.3);

double x = 9.3 + 15.2;
s1.addFirst(x);
```

Array Overview

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- Generic Lists

Arrays:

- **Array Overview**
- Basic Array Syntax
- 2D Arrays
- Arrays vs. Classes

Our Long Term Goal (next two lectures): The AList

In the last few lectures, we've seen how we can harness a recursive class definition to build an expandable list, ie. the `IntList`, the `SLList`, and the `DLList`.

In the next two, we'll see how we can harness arrays to build such a list.

Getting Memory Boxes

To store information, we need memory boxes, which we can get in Java by declaring variables or instantiating objects. Examples:

- `int x;` ← Gives us a memory box of 32 bits that stores ints.
- `Walrus w1;` ← Gives us a memory box of 64 bits that stores Walrus references.
- `Walrus w2 = new Walrus(30, 5.6);` ← Gives us a memory box of 64 bits that stores Walrus references, and also gives us 96 bits for storing the int size (32 bits) and double tuskSize (64 bits) of our Walrus.

Arrays are a special kind of object which consists of a **numbered** sequence of memory boxes.

- To get ith item of array A, use A[i].
- Unlike **class** instances which have have **named** memory boxes.

Arrays

Arrays consist of:

- A fixed integer **length** (cannot change!)
- A sequence of N memory boxes where **N=length**, such that:
  - All of the boxes hold the same type of value (and have same # of bits).
  - The boxes are numbered 0 through length-1.

Like instances of classes:

- You get one reference when its created.
- If you reassign all variables containing that reference, you can never get the array back.

Unlike classes, arrays do not have methods.

Basic Array Syntax

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- Generic Lists

Arrays:

- Array Overview
- **Basic Array Syntax**
- 2D Arrays
- Arrays vs. Classes

Arrays

Like classes, arrays are (almost always) instantiated with new.

Three valid notations:

Creates array containing 3 int boxes (32 x 3 = 96 bits total). Each container gets a default value.

```
x = new int[3];
y = new int[]{1, 2, 3, 4, 5};
int[] z = {9, 10, 11, 12, 13};
```

Can omit the new if you are also declaring a variable.

All three notations create an array, which we saw on the last slide comprises:

- A **length** field.
- A sequence of **N boxes**, where **N** = **length**.

As an aside: In Oracle's implementation of Java, all Java objects also have some overhead. Total size of an array=192 + KN bits, where K is the number of bits per item (Sedgewick/Wayne pg. 201 for more)

Array Basics: http://goo.gl/tFyMEJ

```
int[] z = null;
int[] x, y;

x = new int[]{1, 2, 3, 4, 5};
y = x;
x = new int[]{-1, 2, 5, 4, 99};
y = new int[3];
z = new int[0];
int xL = x.length;

String[] s = new String[6];
s[4] = "ketchup";
s[x[3] - x[1]] = "muffins";

int[] b = {9, 10, 11};
System.arraycopy(b, 0, x, 3, 2);
```

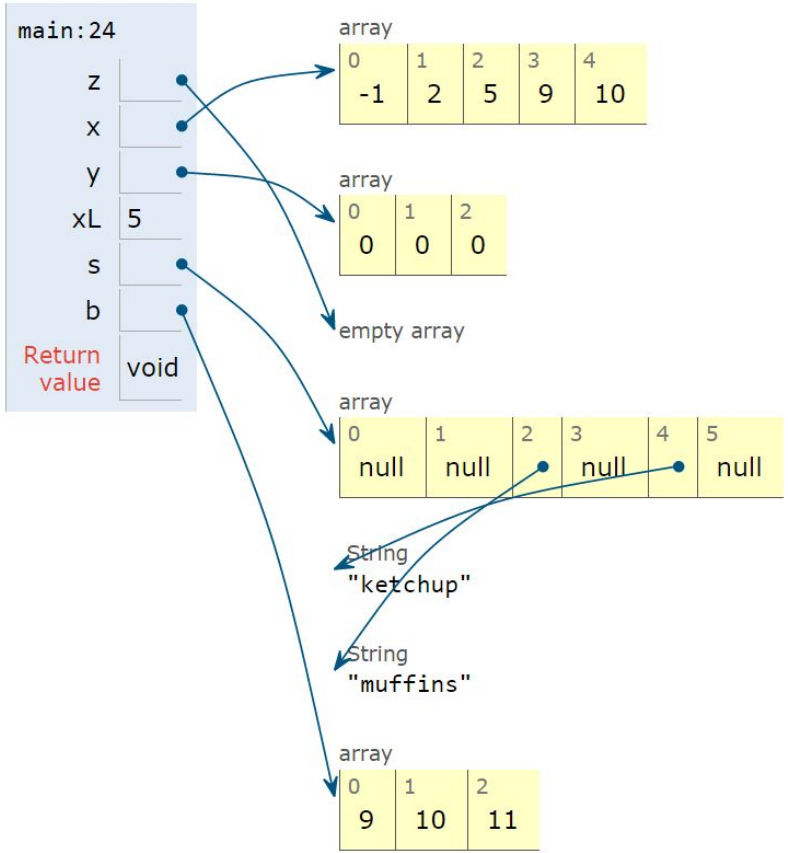
Array Basics: https://goo.gl/gzAuBa

```
int[] z = null;
int[] x, y;

x = new int[]{1, 2, 3, 4, 5};
y = x;
x = new int[]{-1, 2, 5, 4, 99};
y = new int[3];
z = new int[0];
int xL = x.length;

String[] s = new String[6];
s[4] = "ketchup";
s[x[3] - x[1]] = "muffins";

int[] b = {9, 10, 11};
System.arraycopy(b, 0, x, 3, 2);
```



Arraycopy

Two ways to copy arrays:

- Item by item using a loop.
- Using arraycopy. Takes 5 parameters:
  - Source array
  - Start position in source
  - Target array
  - Start position in target
  - Number to copy

```
System.arraycopy(b, 0, x, 3, 2);
```

```
(In Python): x[3:5] = b[0:2]
```

arraycopy is (likely to be) faster, particularly for large arrays. More compact code.

- Code is (arguably) harder to read.

2D Arrays

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- Generic Lists

Arrays:

- Array Overview
- Basic Array Syntax
- **2D Arrays**
- Arrays vs. Classes

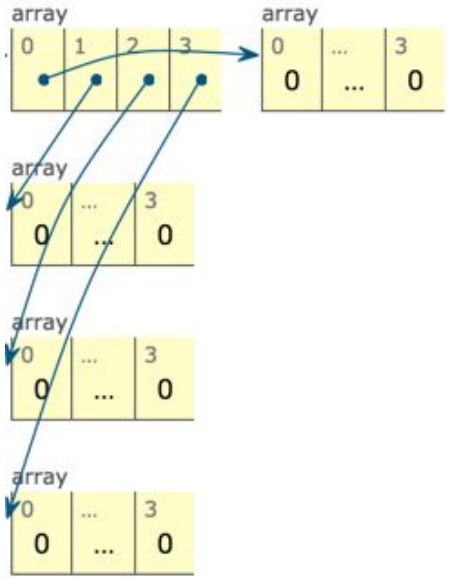
Arrays of Array Addresses (http://goo.gl/VS4cOK)

```
int[][] pascalsTriangle;
pascalsTriangle = new int[4][];
int[] rowZero = pascalsTriangle[0];

pascalsTriangle[0] = new int[]{1};
pascalsTriangle[1] = new int[]{1, 1};
pascalsTriangle[2] = new int[]{1, 2, 1};
pascalsTriangle[3] = new int[]{1, 3, 3, 1};
int[] rowTwo = pascalsTriangle[2];
rowTwo[1] = -5;

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

int[][] pascalAgain = new int[][]{{1}, {1, 1},
                                  {1, 2, 1}, {1, 3, 3, 1}};
```



- Syntax for arrays of arrays can be a bit confounding. You'll learn through practice (much later).

Array Boxes Can Contain References to Arrays!

```
int[][] pascalsTriangle;
pascalsTriangle = new int[4][];
int[] rowZero = pascalsTriangle[0];

pascalsTriangle[0] = new int[]{1};
pascalsTriangle[1] = new int[]{1, 1};
pascalsTriangle[2] = new int[]{1, 2, 1};
pascalsTriangle[3] = new int[]{1, 3, 3, 1};
int[] rowTwo = pascalsTriangle[2];
rowTwo[1] = -5;

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

int[][] pascalAgain = new int[][]{{1}, {1, 1},
                                  {1, 2, 1}, {1, 3, 3, 1}};
```

Array of int array references.

Create four boxes, each can store an int array reference

Create a new array with three boxes, storing integers 1, 2, 1, respectively. Store a reference to this array in pascalsTriangle box #2.

Creates 1 total array.

Creates 5 total arrays.

- Syntax for arrays of arrays can be a bit confounding. You'll learn through practice (much later).

What Does This Code Do? (Bonus Slides Only Exercise)

What will be the value of x[0][0] and w[0][0] when the code shown completes?

A. x: 1, w: 1
B. x: 1, w: -1
C. x: -1, w: 1
D. x: -1, w: -1
E. Other

```
int[][] x = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int[][] z = new int[3][];
z[0] = x[0];
z[0][0] = -z[0][0];

int[][] w = new int[3][3];
System.arraycopy(x[0], 0, w[0], 0, 3);
w[0][0] = -w[0][0];
```

arraycopy parameters are:

1. Source array
2. Start position in source
3. Target array
4. Start position in target
5. Number to copy

Answer: https://goo.gl/CqrZ7Y

Arrays vs. Classes

Lecture 5, CS61B, Spring 2024

SLLists:

- Summary of SLLists So Far
- Why a Last Pointer Isn't Enough
- Doubly Linked Lists
- Generic Lists

Arrays:

- Array Overview
- Basic Array Syntax
- 2D Arrays
- **Arrays vs. Classes**

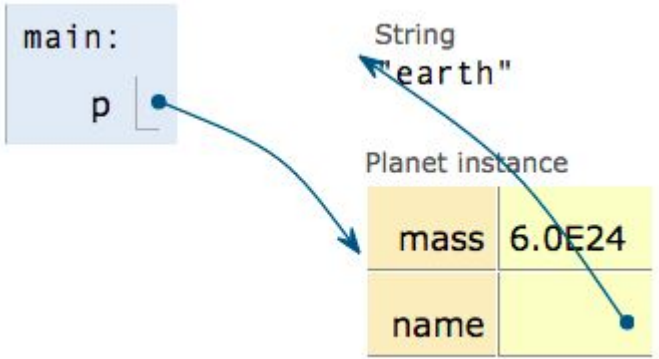
Arrays vs. Classes

Arrays and Classes can both be used to organize a bunch of memory boxes.

- Array boxes are accessed using [] notation.
- Class boxes are accessed using dot notation.
- Array boxes must all be of the same type.
- Class boxes may be of different types.
- Both have a fixed number of boxes.

```
public class Planet {
   public double mass;
   public String name;
   ...
}
```

```
int[] x = new int[]{100, 101, 102, 103};
Planet p = new Planet(6e24, "earth");
```

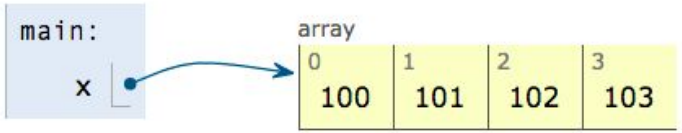


Arrays vs. Classes

Array indices can be computed at runtime.

```
int[] x = new int[]{100, 101, 102, 103};
int indexOfInterest = askUser();
int k = x[indexOfInterest];
System.out.println(k);
```

```
jug ~/Dropbox/61b/lec/lists3
$ javac ArrayDemo.java
$ java ArrayDemo
What index do you want? 2
102
```



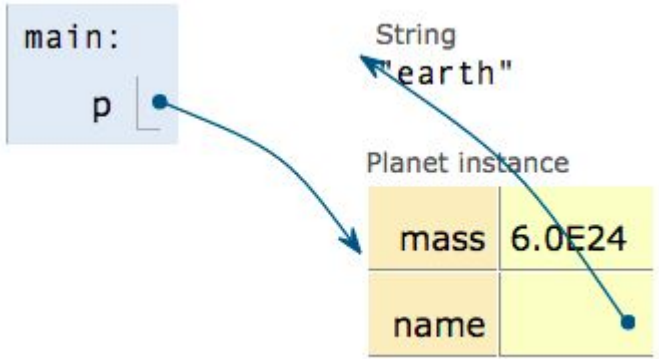
Arrays vs. Classes

Class member variable names CANNOT be computed and used at runtime.

```
String fieldOfInterest = "mass";
Planet earth = new Planet(6e24, "earth");
double mass = earth[fieldOfInterest];
System.out.println(mass);
```

```
jug ~/Dropbox/61b/lec/lists3
$ javac ClassDemo.java
ClassDemo.java:5: error: array required,
   but Planet found.

   double mass = earth[fieldOfInterest];
                       ^
```



… if you reallllly want to do this, you can: https://goo.gl/JxpyLq

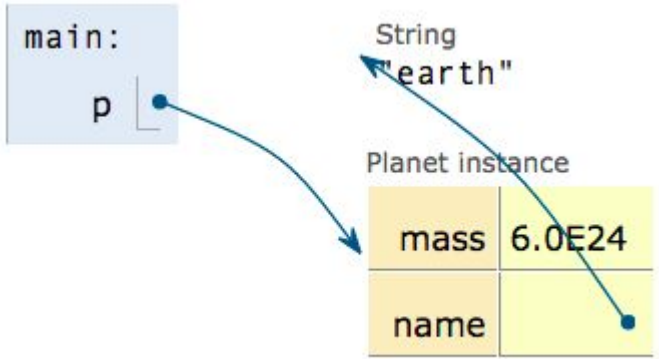
Arrays vs. Classes

Class member variable names CANNOT be computed and used at runtime.

- Dot notation doesn't work either.

```
String fieldOfInterest = "mass";
Planet earth = new Planet(6e24, "earth");
double mass = earth.fieldOfInterest;
System.out.println(mass);
```

```
jug ~/Dropbox/61b/lec/lists3
$ javac ClassDemo.java
ClassDemo.java:5: error: cannot find Symbol
   double mass = earth.fieldOfInterest;
                       ^
   symbol:   variable fieldOfInterest
   location: variable earth of type Planet
```



… if you reallllly want to do this, you can: https://goo.gl/JxpyLq

Another view

The only (easy) way to access a member of a class is with hard-coded dot notation.

```
int k = x[indexOfInterest];     /* no problem */

double m = p.fieldOfInterest;   /* won't work */
double z = p[fieldOfInterest];  /* won't work */
/* No (sane) way to use field of interest */

double w = p.mass;              /* works fine */
```

The Java compiler does not treat text on either side of a dot as an expression, and thus it is not evaluated.

- See a compilers or programming languages class for more!