CS159

Java Collections

- Naming convention for Java Collection types: ArrayList
  - Array – Coded using Arrays "under the hood"
  - List – Implements the List Interface

    An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list

List Interface (abridged)

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the end of this list (optional operation).
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list (optional operation).
E	**get**(int index) Returns the element at the specified position in this list.
boolean	**isEmpty**() Returns `true` if this list contains no elements.
E	**remove**(int index) Removes the element at the specified position in this list (optional operation).
E	**set**(int index, **E** element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	**size**() Returns the number of elements in this list.

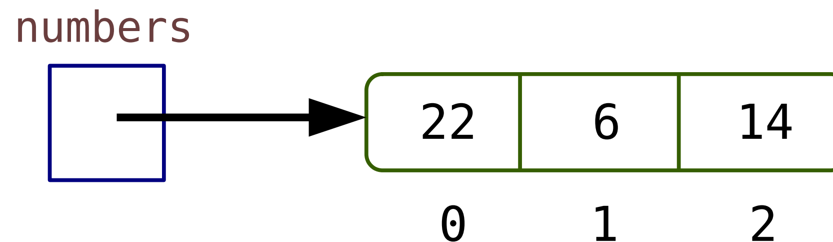
Question for Today

- Arrays are great. Why would we want any other implementation?
- In general: why would Java provide multiple implementation for their collection interfaces? Why not just pick the best one?

Why Arrays Are Great (And Not Great)

- Arrays store elements in one contiguous block of memory:

```
int[] numbers = {22, 6, 14};
```



- Advantages of arrays:
  - Very fast access to elements by index.

Socrative Quiz…

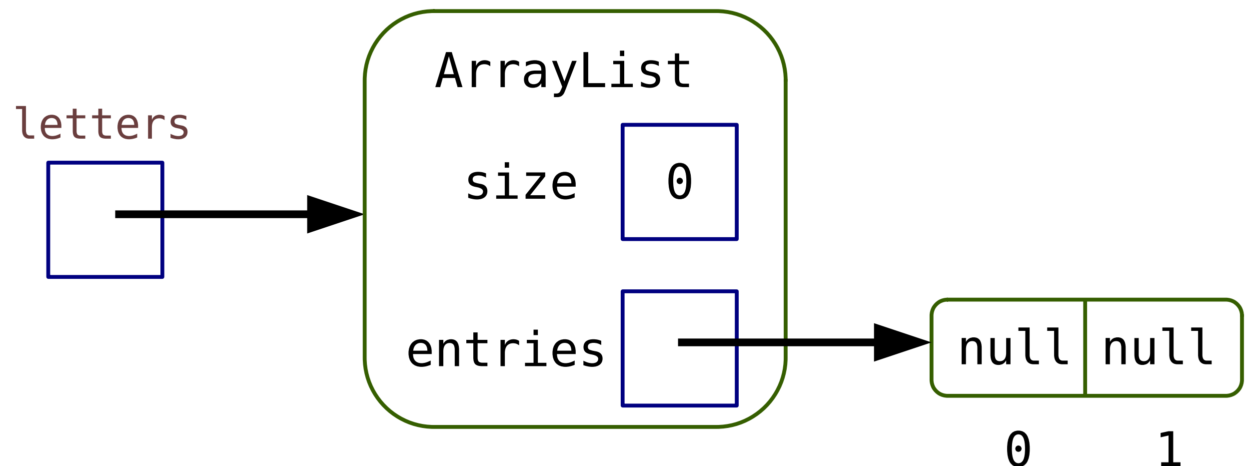
- Let's look at some code…
  - TimingDemo1.java

Exercise...

- Trace the execution of the following code snippet. Draw the contents of memory.
- Assume that array size is doubled when needed.

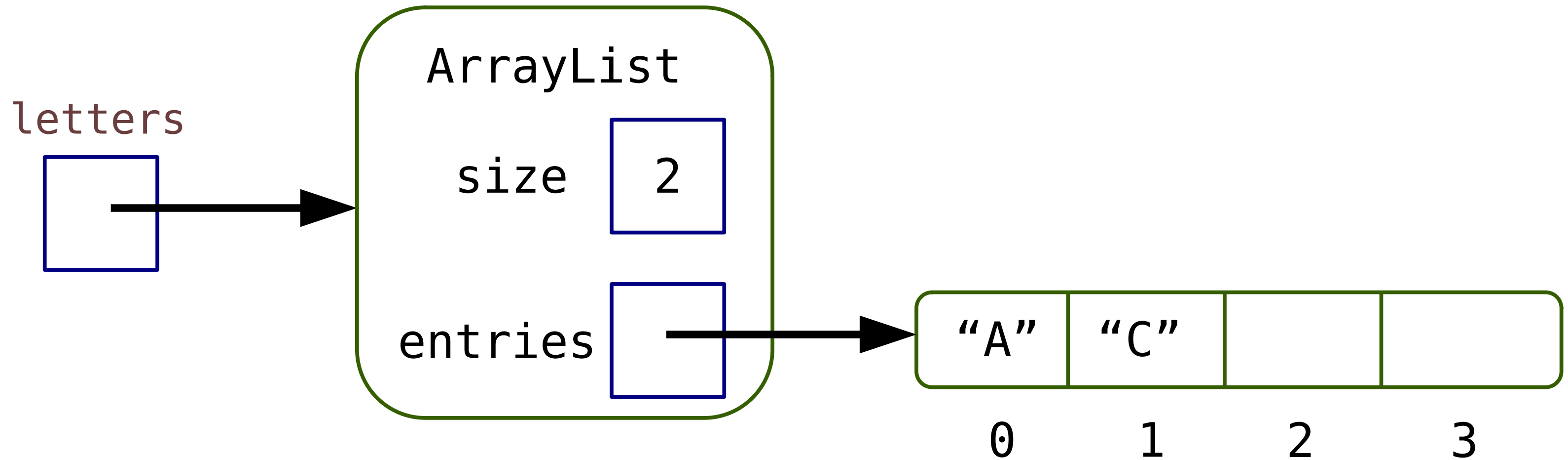
```
ArrayList<String> letters = new ArrayList<>();
letters.add("A");
letters.add(0, "B");
letters.add(1, "C");
letters.remove(0);
```

First line finished for you:



Exercise…

```
ArrayList<String> letters = new ArrayList<>();
letters.add("A");
letters.add(0, "B");
letters.add(1, "C");
letters.remove(0);
```

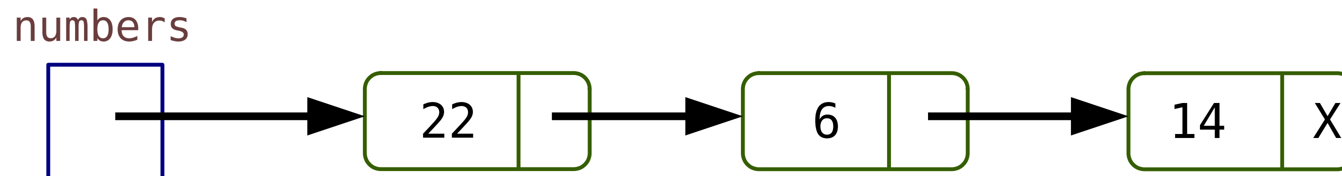


ArrayList Weaknesses

- Insertion and deletion near the beginning is slooow
  - Insertion:
    - Every element to the right needs to be shifted right to make space
  - Deletion:
    - Every element to the right needs to be shifted left to fill the gap
- Maybe a LinkedList (??) will do better. Let's try...

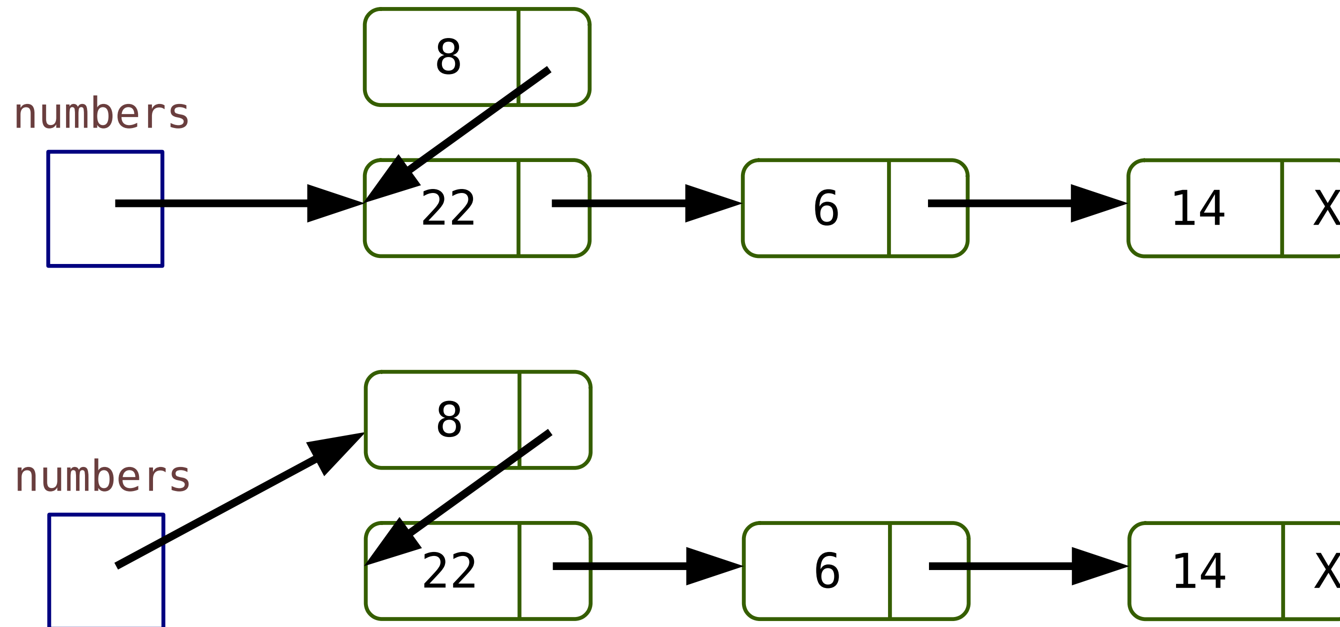
Linked Structures

- Linked structures “chain” elements using references:



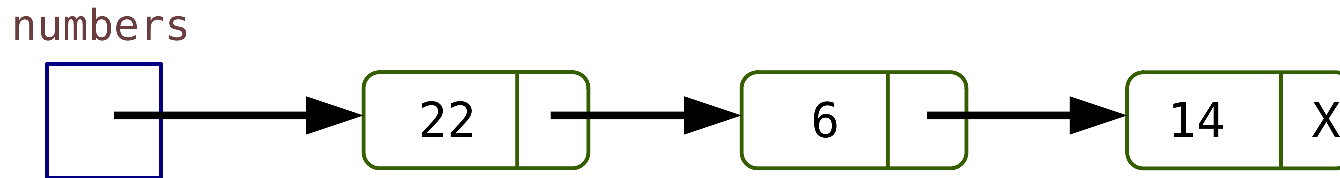
Linked Insertion

- This organization allows fast insertions/deletions near the beginning.
- Adding 8:



Linked List Implementation

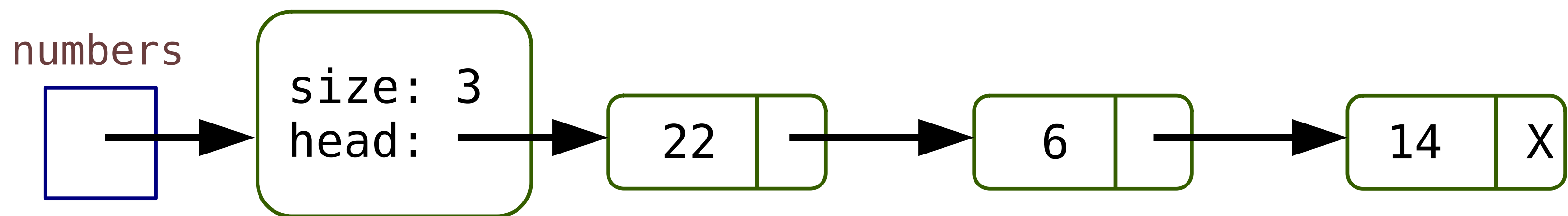
- Linked objects are referred to as “nodes”



- Node.java
- NodeDriver.java

LinkedList Implementation

- Inconvenient to work with "naked" nodes.
- Create a wrapper class to handle list logic:



- SimpleLinkedList.java
- ListDriver.java

Socrative Quiz

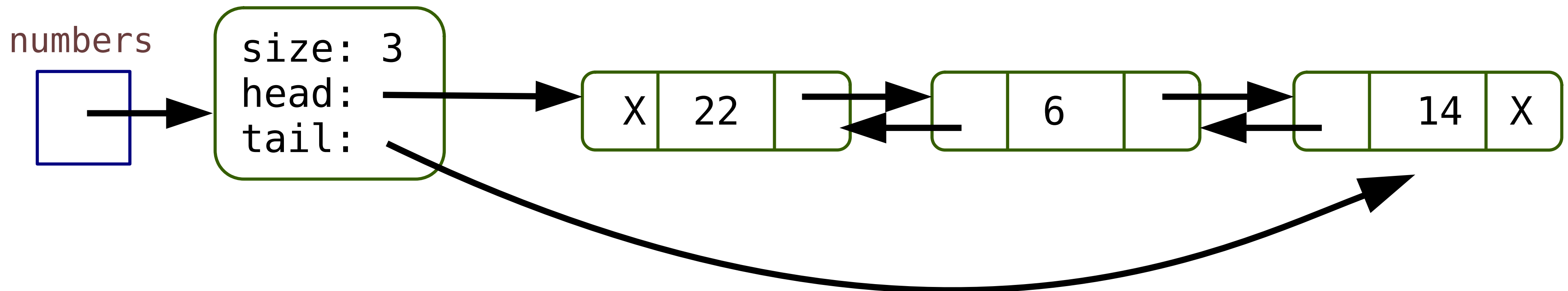
- TimingDemo2.java

LinkedList vs ArrayList

- In general:
  - LinkedLists slow for random access
  - ArrayList slow for insertion/deletion near the beginning

Doubly-Linked Lists

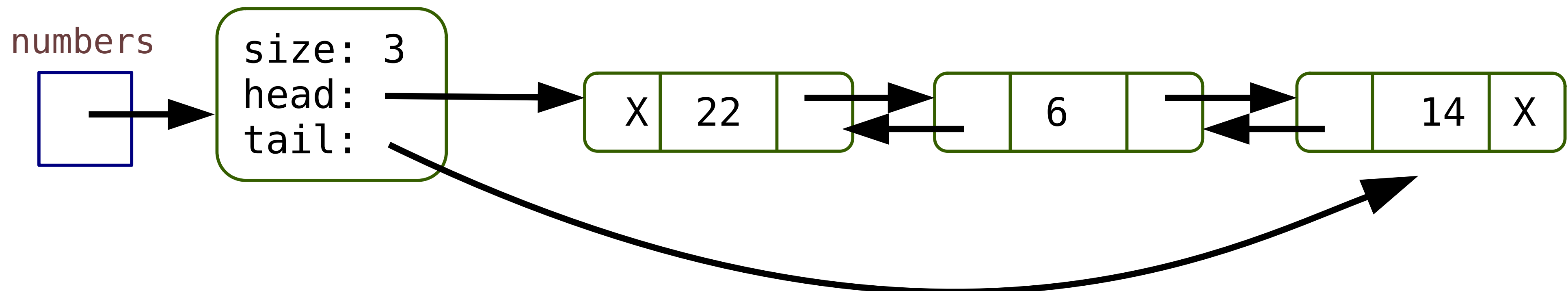
- Java's LinkedList class is doubly-linked:



- What are the advantages to this approach?
- Are there any disadvantages?

Doubly-Linked Lists

- Java's LinkedList class is doubly-linked:



- What are the advantages to this approach?
  - Efficiently add/remove near either end
  - Efficient backward iteration
- Are there any disadvantages?
  - The backward references require extra space