Last Time

Multidimensional arrays

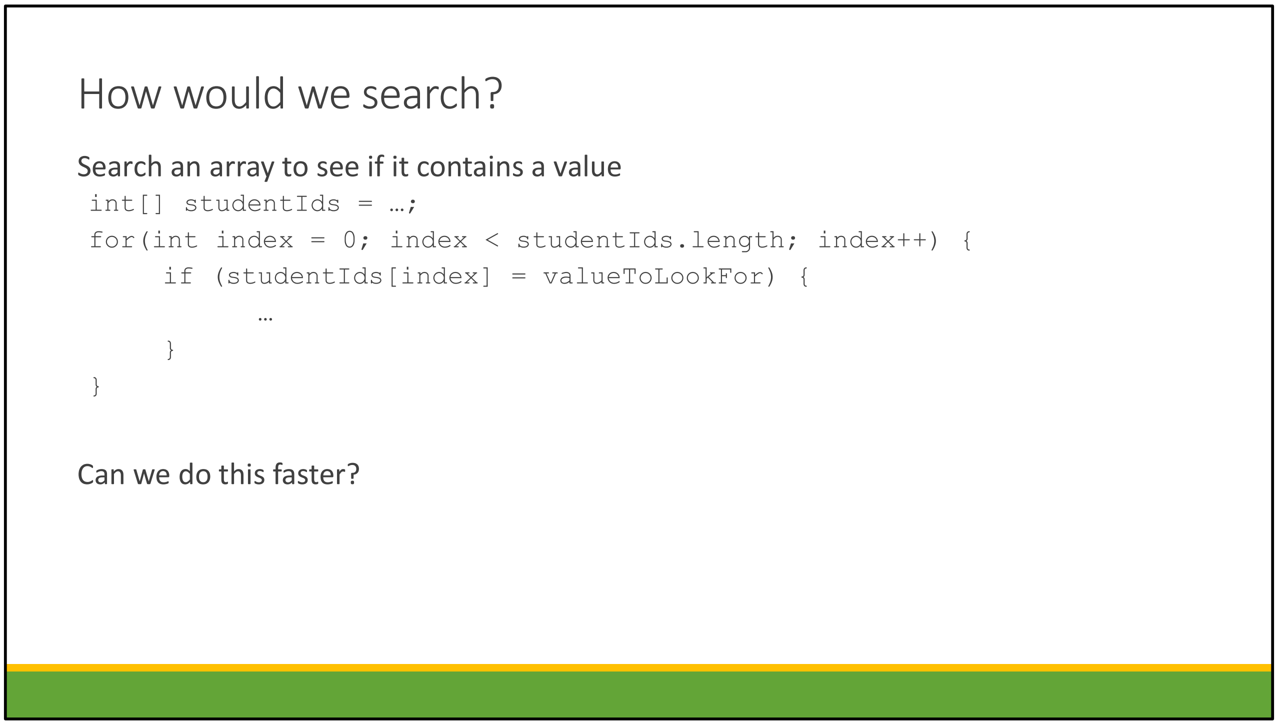
class ArrayList

class HashMap

Iterators

Searching and Binary Search

DO WE HAVE TIME FOR THIS?



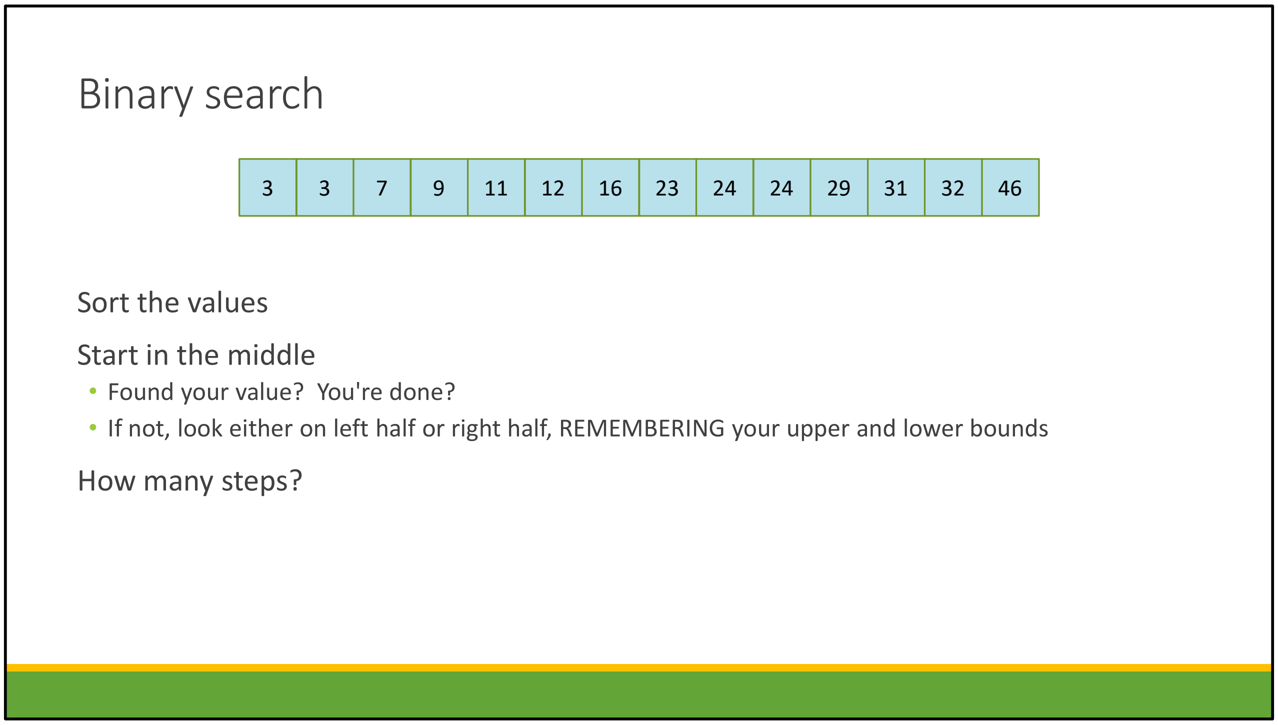
How would we search?

Search an array to see if it contains a value

```
int[] studentIds = …;
for(int index = 0; index < studentIds.length; index++) {
     if (studentIds[index] = valueToLookFor) {
            …
     }
}
```

Can we do this faster?

If list length is L, how many compares do we do on average? Worst case?



Binary search

3	3	7	9	11	12	16	23	24	24	29	31	32	46

Sort the values

Start in the middle

- Found your value? You're done?
- If not, look either on left half or right half, REMEMBERING your upper and lower bounds

How many steps?

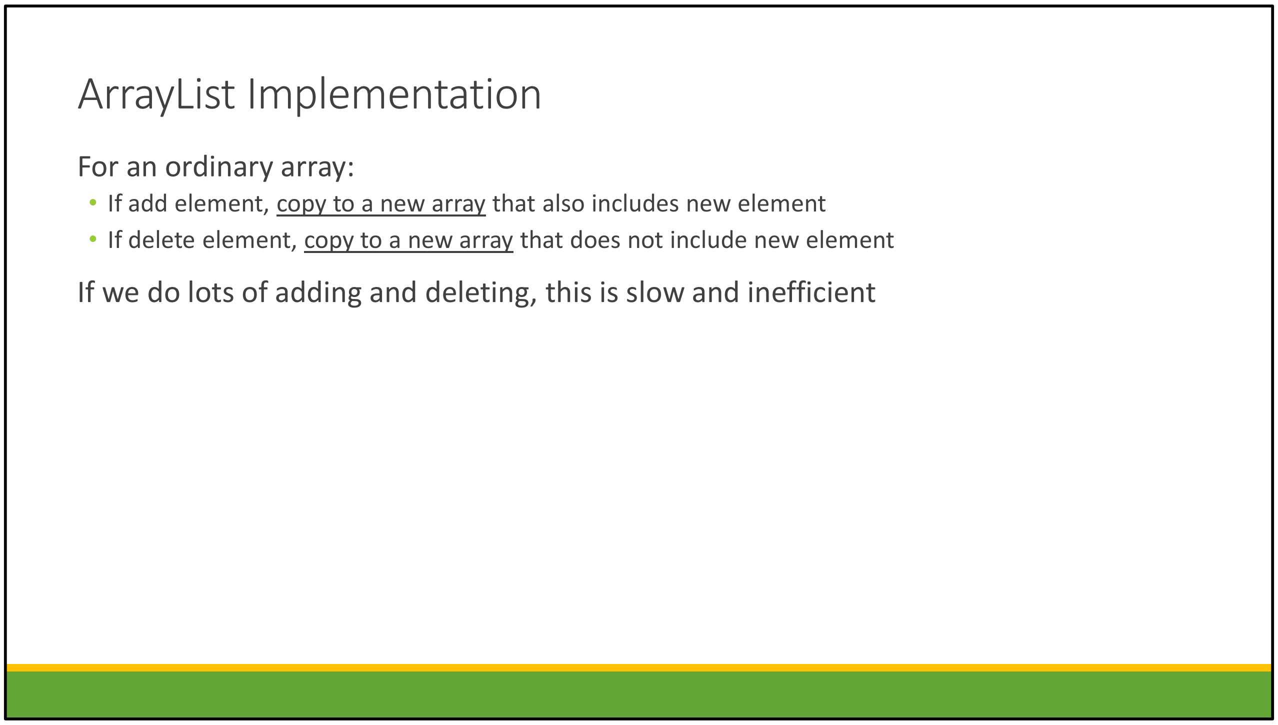
Show # steps is log2(length)
Log2(length) << length, if length is reasonably big. What is log2(1000000) ?

This is great if we sort once and search repeatedly

Analysis of Data Structures

Performance trade-offs

What's "performance"?



ArrayList Implementation

For an ordinary array:

- If add element, copy to a new array that also includes new element
- If delete element, copy to a new array that does not include new element

If we do lots of adding and deleting, this is slow and inefficient

DO THIS AS A CODE-ALONG.
If we spend much more time looking up values than adding/deleting, ArrayList is great.
…but much worse for random ACCESS

Advanced Data Structures

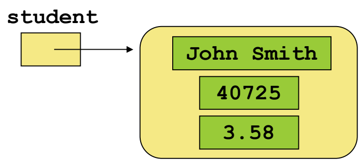
What's a data structure? A class and objects for storing, retrieving, and manipulating data, with desired properties.
Array, dictionary, list, queue, etc

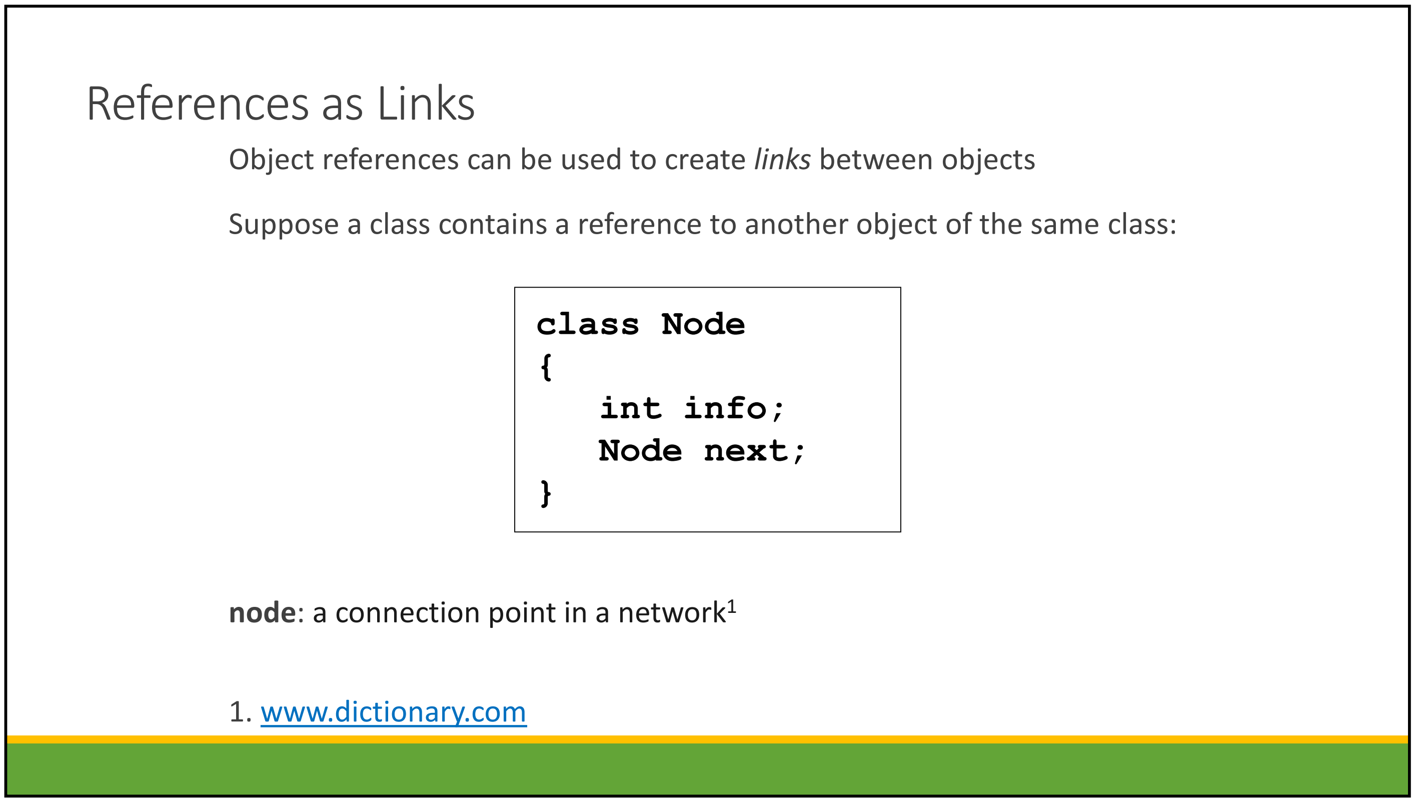
Object References

Recall that an *object reference* is a variable that stores the address of an object

A reference also can be called a *pointer*

References often are depicted graphically:





References as Links

Object references can be used to create *links* between objects

Suppose a class contains a reference to another object of the same class:

```
class Node
{
   int info;
   Node next;
}
```

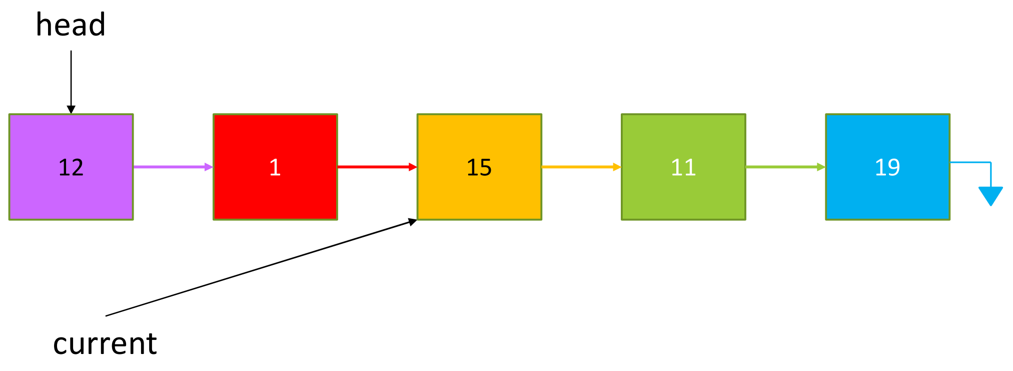
node: a connection point in a network[1]

1. www.dictionary.com

This lets us connect objects together into networks

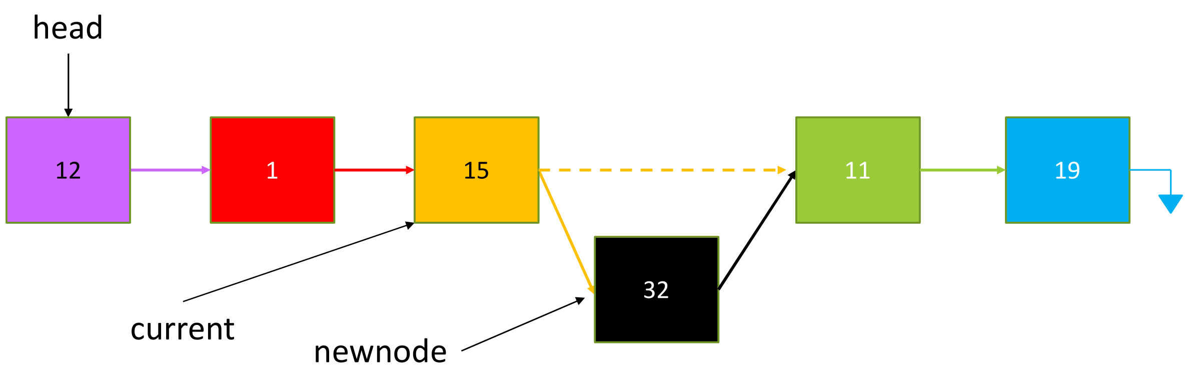
References as Links

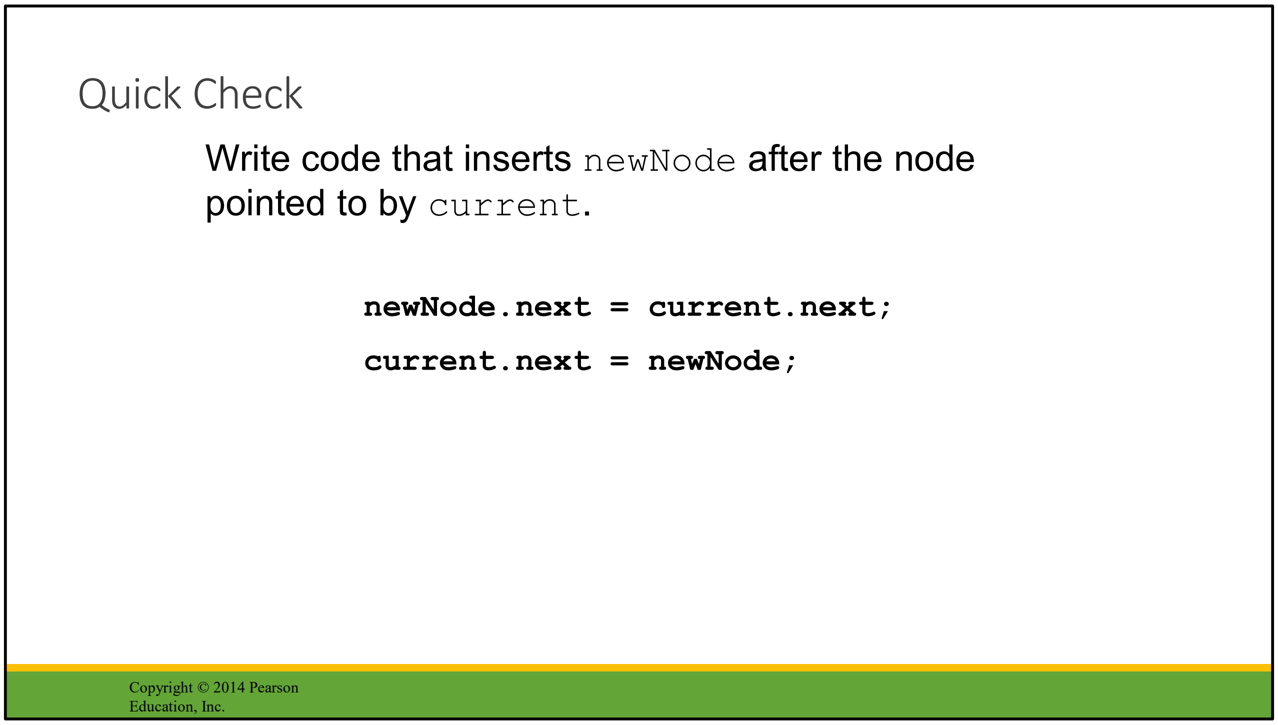
References can be used to create a variety of linked structures, such as a *linked list*:



Inserting a Node

A node can be inserted into the middle of a linked list with a few reference changes:





Quick Check

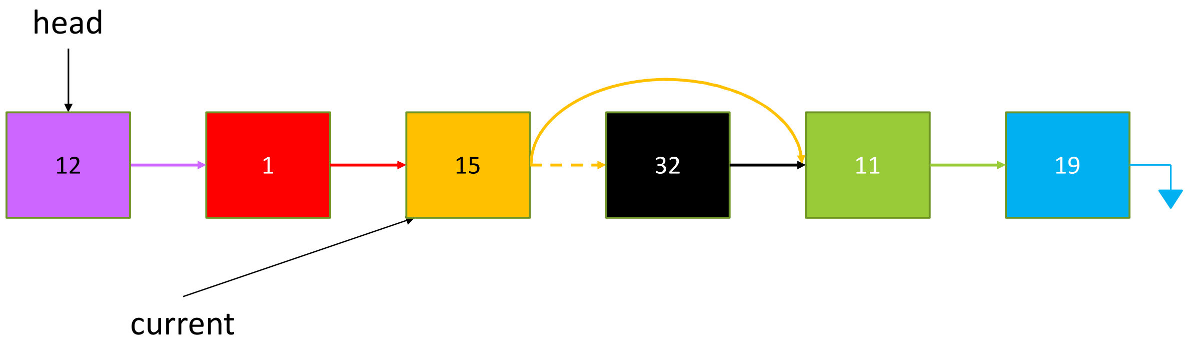
Write code that inserts `newNode` after the node pointed to by `current`.

```
newNode.next = current.next;
current.next = newNode;
```

L-value vs R-value discussion again! On the left? A reference variable. On the right? The object!

Deleting a Node

Likewise, a node can be removed from a linked list by changing the `next` pointer of the preceding node:



What's the code for that?

```
current.next = current.next.next;
```

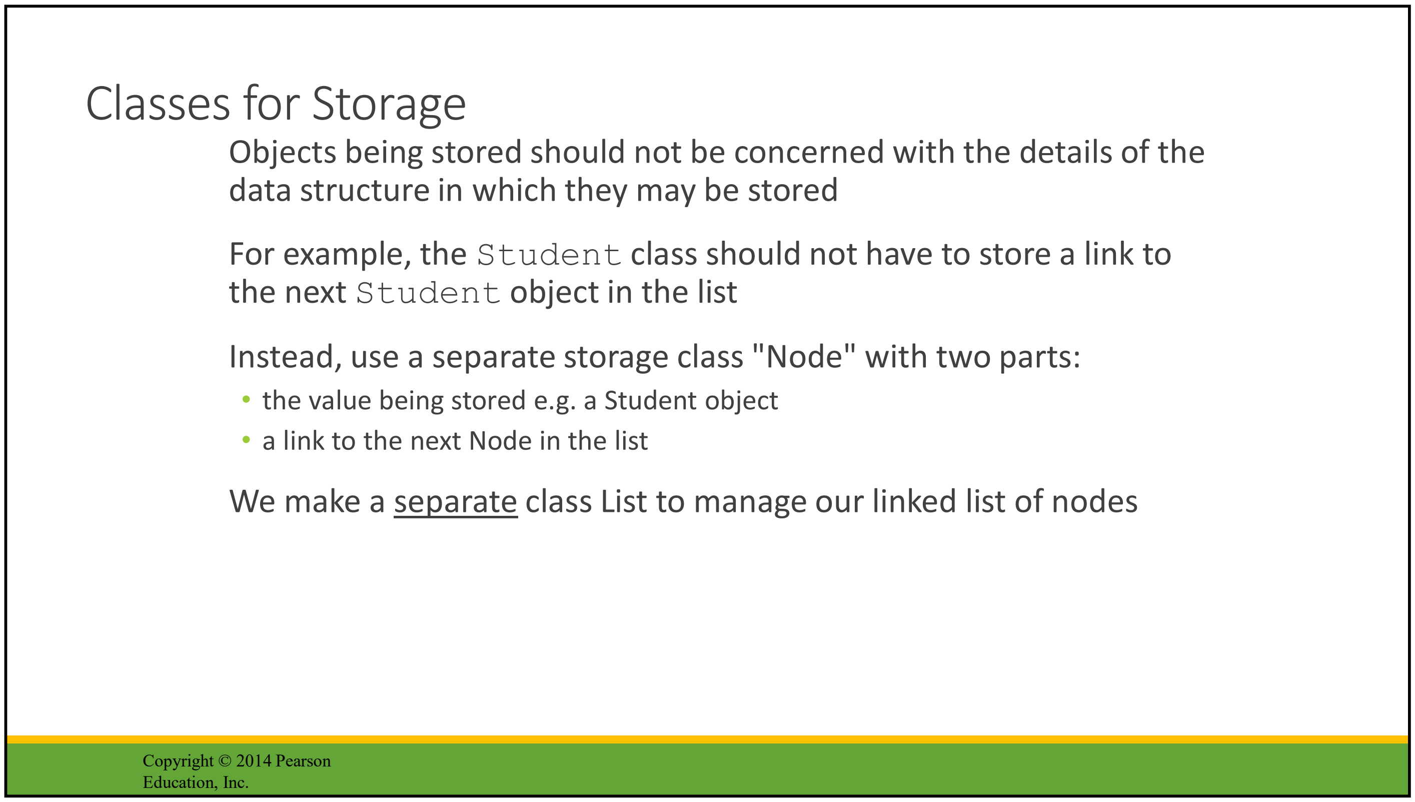
L-value vs R-value discussion again! On the left? A reference variable. On the right? The object!

Accessing the list

Suppose we want to print out every element in the list – how do we do it?

DO THIS IN ECLIPSE!

Designing Collections



Classes for Storage

Objects being stored should not be concerned with the details of the data structure in which they may be stored

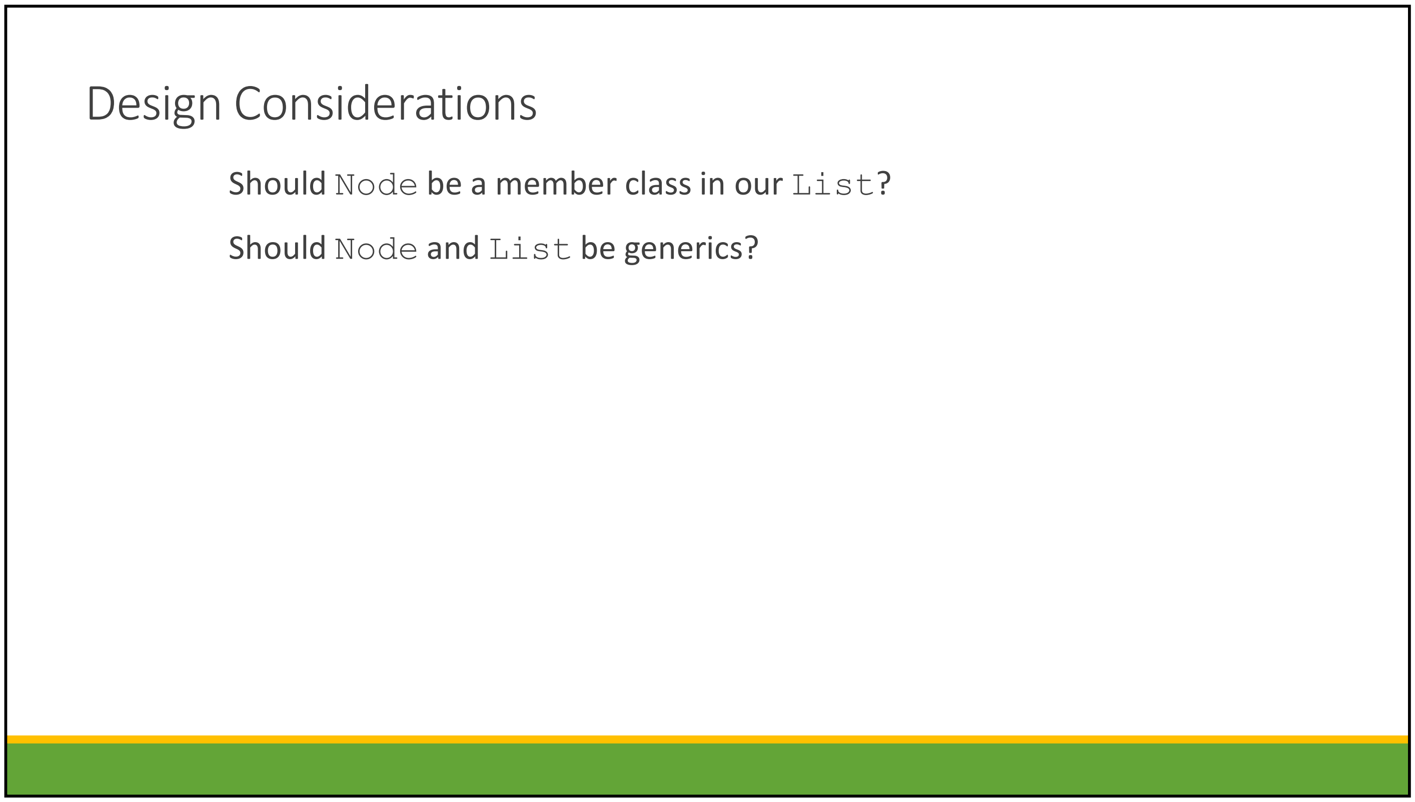
For example, the `Student` class should not have to store a link to the next `Student` object in the list

Instead, use a separate storage class "Node" with two parts:

- the value being stored e.g. a Student object
- a link to the next Node in the list

We make a <u>separate</u> class List to manage our linked list of nodes

Becomes a list because we connect everything in a straight line.
3 classes: the thing we store, the Node, the List.



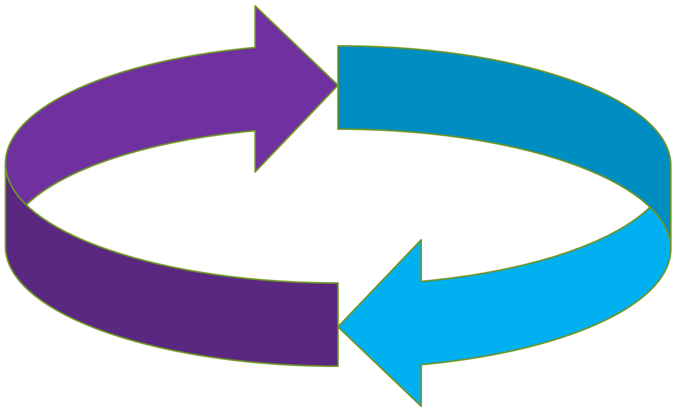
Design Considerations

Should `Node` be a member class in our `List`?

Should `Node` and `List` be generics?

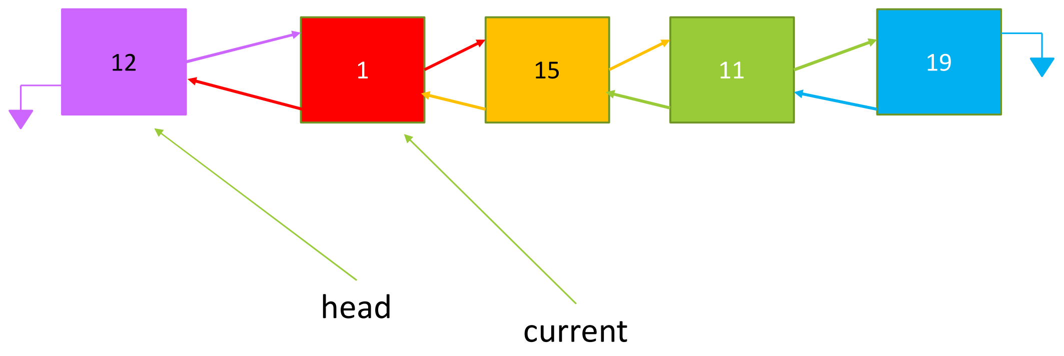
Not member class—doesn't need access to List members. But yes private, and Yes generic
Maybe Not generic if we only want to store a particular built-in type…

Another Collection

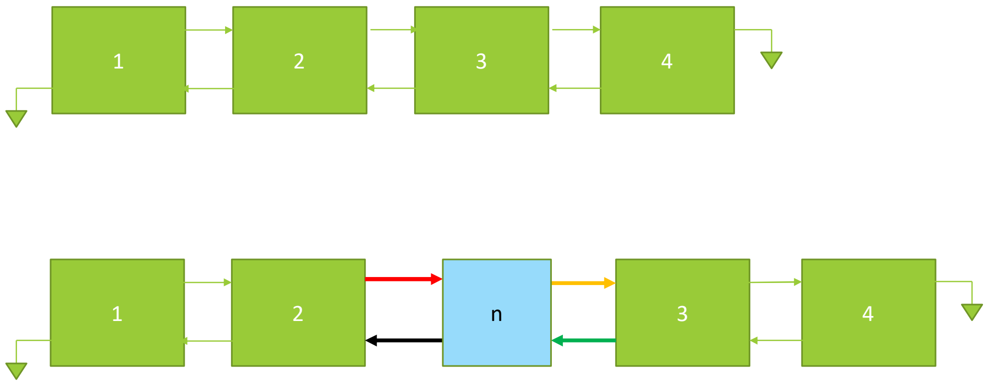


Other Dynamic Representations

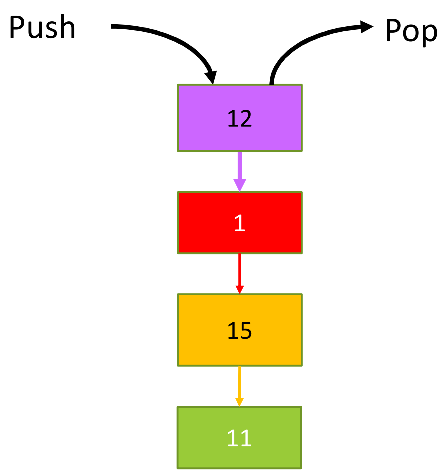
It may be convenient to implement a list as a *doubly linked list*, with `next` and `previous` references:



Doubly Linked List operations

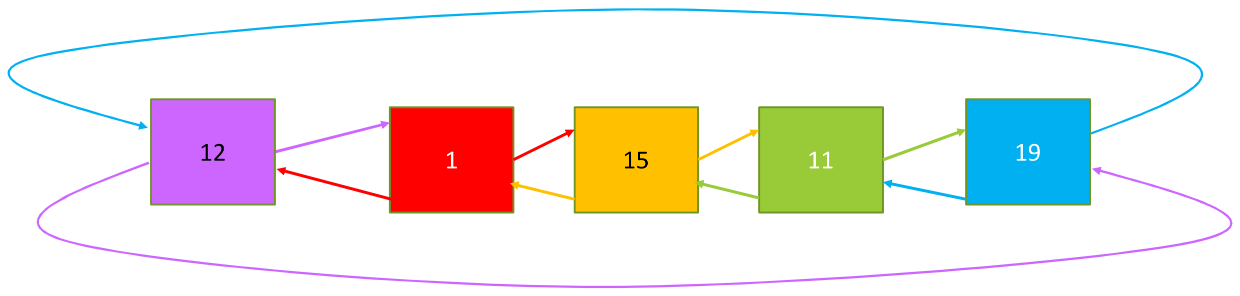


Use a List to build a Stack



Like a stack of plates in the cafeteria. Use to store functions in the "function call stack".
Only can access the top entry

Another Collection

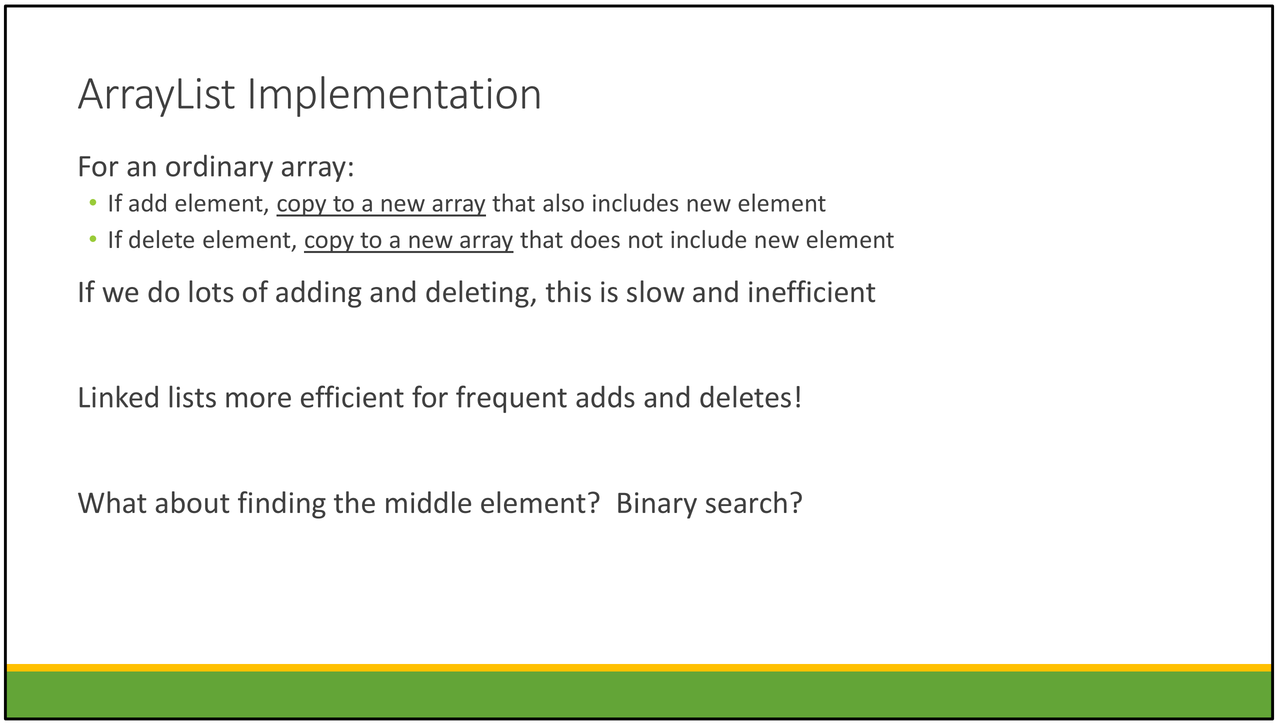


A Ring. Why used? Say we want to store tasks for a CPU to process in turn
Often used as a "buffer": data arrives in bursts, and we process it steadily, one-at-a-time, everyone gets a turn.

Analysis of Data Structures

Performance trade-offs

What's "performance"?



ArrayList Implementation

For an ordinary array:

- If add element, copy to a new array that also includes new element
- If delete element, copy to a new array that does not include new element

If we do lots of adding and deleting, this is slow and inefficient

Linked lists more efficient for frequent adds and deletes!

What about finding the middle element? Binary search?

If we spend much more time looking up random values than adding/deleting,
ArrayList is great.
…but much worse for insert and delete