CSC 1052 – Algorithms & Data Structures II: Lists

Professor Henry Carter
Spring 2017

Recap

- Collections hold and access elements based on content
  - Order and index no longer considered
- Comparable elements implement methods to compare objects more meaningfully than comparing pointers
- Collections may be sorted or unsorted, with tradeoffs in the time to search and the time to add/remove elements
- Collections may be extended to emulate other data structures
  - Bags
  - Mathematical sets

New ADT: Lists

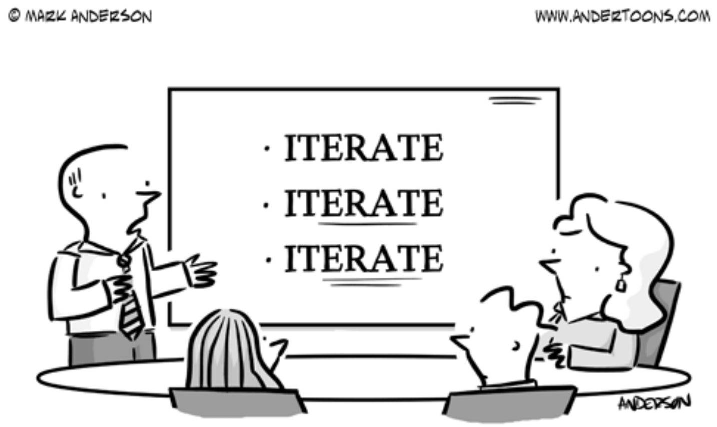
- Common and powerful data structure in many languages
    - E.g., Lisp, Python
- Maintains a linear relation between elements
- Can be used to implement all of the observed ADTs so far
    - E.g., ArrayList for implementing stacks and queues
- Many applications: allow similar operations to primitive arrays with fewer limitations

Assumptions

- Unbounded
- Duplicates allowed
- No null elements
- Indices are contiguous
  - Only significant limitation when compared to primitive arrays
- Two operations are optional: add() and set()
  - Must be implemented, even if just for exceptions

Iterators

- Iterating over the contents of a list is a common operation
- In some cases, multiple indices into the list need to be maintained
- Rather than require the developer to hard code indices, Iterator objects simplify iterating over data structures
- Two interfaces are required:
  Iterable and Iterator

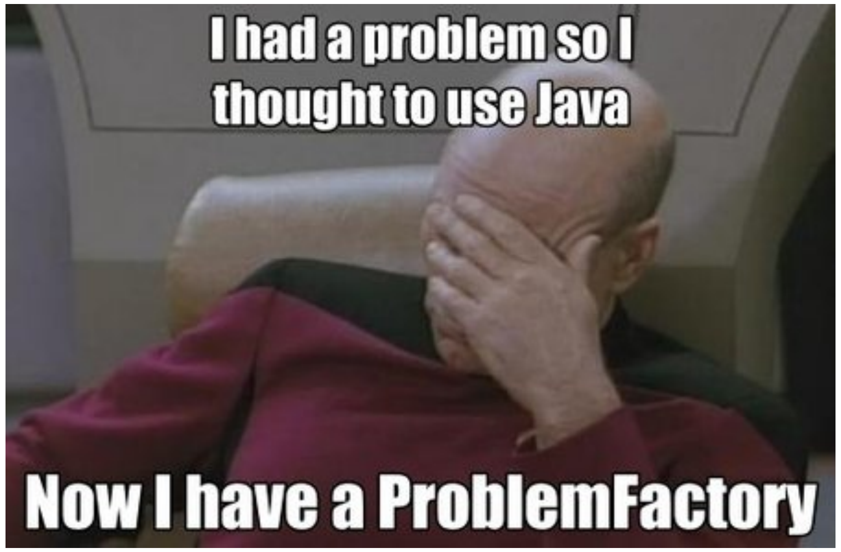


"Let me reiterate..."

Iterable interface

- Implemented by ADTs that allow iteration
- Contains a single method: iterator()
  - A factory method returning an iterator
- Allows new for loop syntax

  -
    ```
    for(String temp: strings)
      System.out.println(temp);
    ```


I had a problem so I thought to use Java
Now I have a ProblemFactory

Iterator interface

- Implemented by classes that specify how to iterate through a data structure
- Contains three methods:
  - next()
  - hasNext()
  - remove()
- Must be implemented dependent on the data structure!

ListInterface

- Extends Collections and Iterable
- Essentially adds index-related operations to collections
- Contains five new methods:
  - add()
  - set()
  - get()
  - indexOf()
  - remove()

ListInterface

```
package ch06.lists;

import java.util.*;
import ch05.collections.CollectionInterface;

public interface ListInterface<T> extends CollectionInterface<T>,
                                          Iterable<T>
{
  void add(int index, T element);

  T set(int index, T newElement);

  T get(int index);

  int indexOf(T target);

  T remove(int index);
}
```

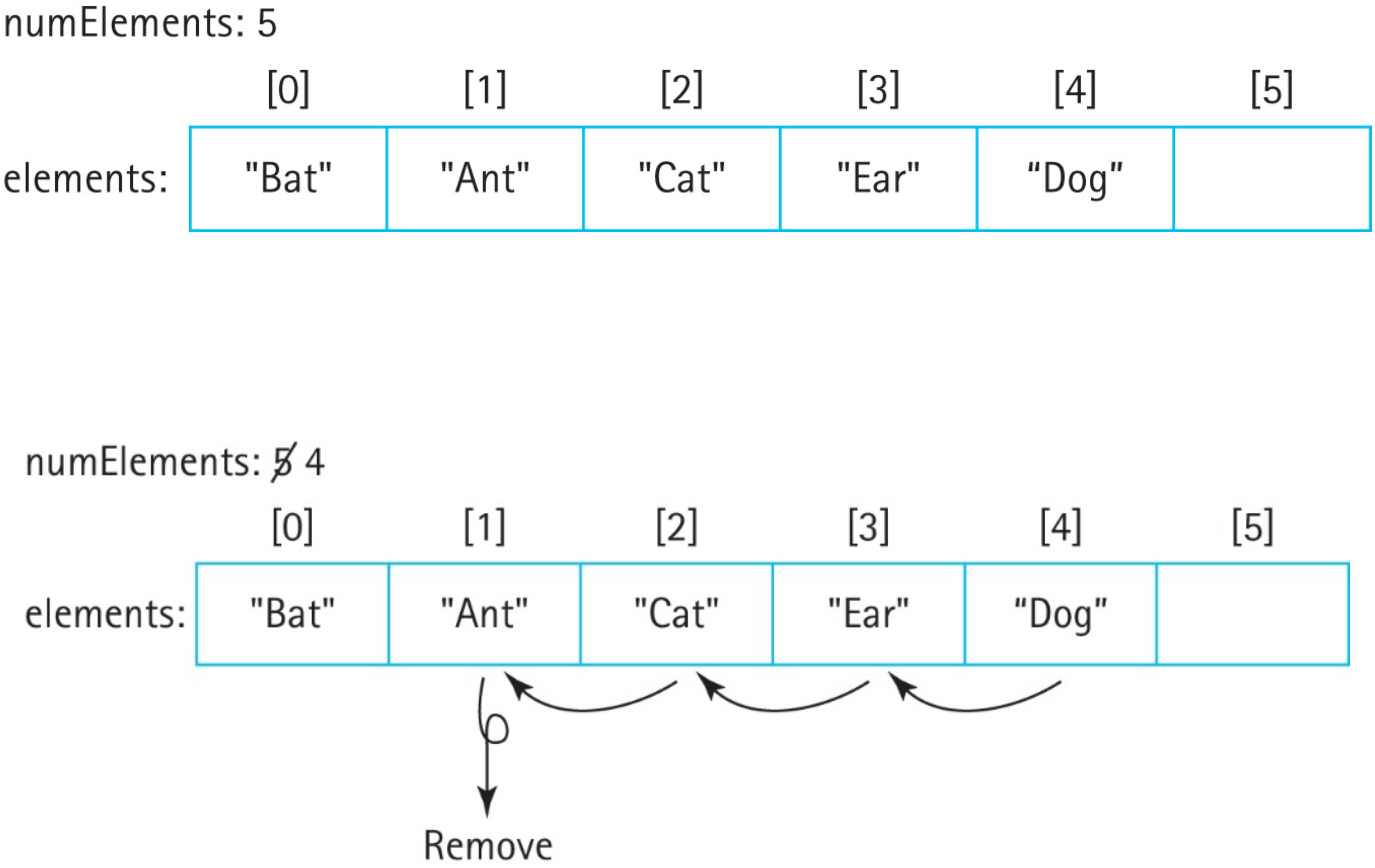
Implementation: Array

- We implement the list first with an array
- Separate the method implementation into three groups:
  - Collection methods
  - New list methods
  - Iterator()
- The internal array will match the external list ordering
  - Unsorted

Collection methods

- Recall: what five methods are required for collections?
- What two hidden methods were helpful in implementing these?
- Will any of these methods change?

Index-Preserving Remove



numElements: 5

	[0]	[1]	[2]	[3]	[4]	[5]
elements:	"Bat"	"Ant"	"Cat"	"Ear"	"Dog"	

numElements: ~~5~~ 4

	[0]	[1]	[2]	[3]	[4]	[5]
elements:	"Bat"	"Ant"	"Cat"	"Ear"	"Dog"	

New Methods

- Each of the new methods requires an index as either input or output
- Indexing exactly matches the indexing of the internal array
  - The range must match number of elements, not the size of the primitive array
- While the collection methods add and remove return boolean, indexed methods throw exceptions
  - Why?

Example: set()

```
public T set(int index, T newElement)
// Throws IndexOutOfBoundsException if passed an index argument
// such that index < 0 or index >= size().
// Otherwise, replaces element on this list at position index with
// newElement and returns the replaced element.
{
  if ((index < 0) || (index >= size()))
    throw new IndexOutOfBoundsException("Illegal index of " + index +
                                   " passed to ABList set method.\n");

  T hold = elements[index];
  elements[index] = newElement;
  return hold;
}
```

ListInterface

```
package ch06.lists;

import java.util.*;
import ch05.collections.CollectionInterface;

public interface ListInterface<T> extends CollectionInterface<T>,
                                          Iterable<T>
{
  void add(int index, T element);

  T set(int index, T newElement);

  T get(int index);

  int indexOf(T target);

  T remove(int index);
}
```

Iteration

- To make the list iterable, we need to define the iterator() method
  - Simply return an Iterator object
- What is an Iterator object?
- Where is it implemented?

Implementing the List Iterator

- Public class
  - Requires an external class to modify internal state in the List
- Inner class
  - Allows hidden definition used only inside the List
- Anonymous class
  - Allows in-line definition used only inside the iterator() method

iterator() code

```
public Iterator<T> iterator()
{
   return new Iterator<T>()
   {
      private int previousPos = -1;

      public boolean hasNext()
      {
         return (previousPos < (size() - 1)) ;
      }
```

iterator() code

```
      public T next()
      {
         if (!hasNext())
            throw new IndexOutOfBoundsException("Illegal invocation of next " +
                                                " in LBList iterator.\n");
         previousPos++;
         return elements[previousPos];
      }

      public void remove()
      {
         for (int i = previousPos; i <= numElements - 2; i++)
            elements [i] = elements[i+1];
         elements [numElements - 1] = null;
         numElements--;
         previousPos--;
      }
   };
}
```

Efficiency

- Collection methods match array implementation
  - With the exception of add at an index
- Indexed observers run in constant time O(1)
- Indexed modifiers run in linear O(N) time
- Iterator methods run in what time?

Implementation: Linked List

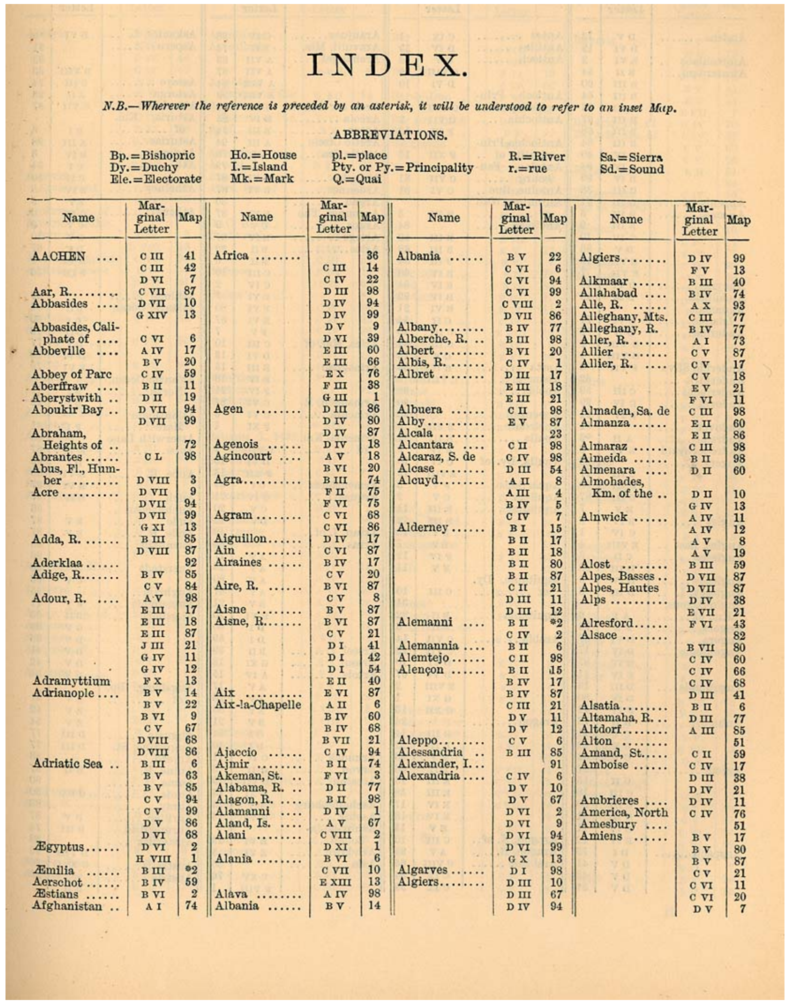
- Much of the linked list implementation mirrors previous data structures
- Allows for simplified modifiers at the cost of less efficient observers
- A more complex version of the LinkedList from project 1

Modifications to Collection Methods

- Add() works with the new rear pointer instead of the head
- TargetIndex variable set by the find() method
- Remove() sometimes has to update the rear pointer

Index Operations

- Requires iterating to the index given
  - Incurring O(N) link operations
- Eliminates the need to shift
  - Requires special case considerations
- Compare the textbook code to your own code!



INDEX.

N.B.—Wherever the reference is preceded by an asterisk, it will be understood to refer to an inset Map.

ABBREVIATIONS.

Bp.=Bishopric	Ho.=House	pl.=place	R.=River	Sa.=Sierra
Dy.=Duchy	I.=Island	Pty. or Py.=Principality	r.=rue	Sd.=Sound
Ele.=Electorate	Mk.=Mark	Q.=Quai		

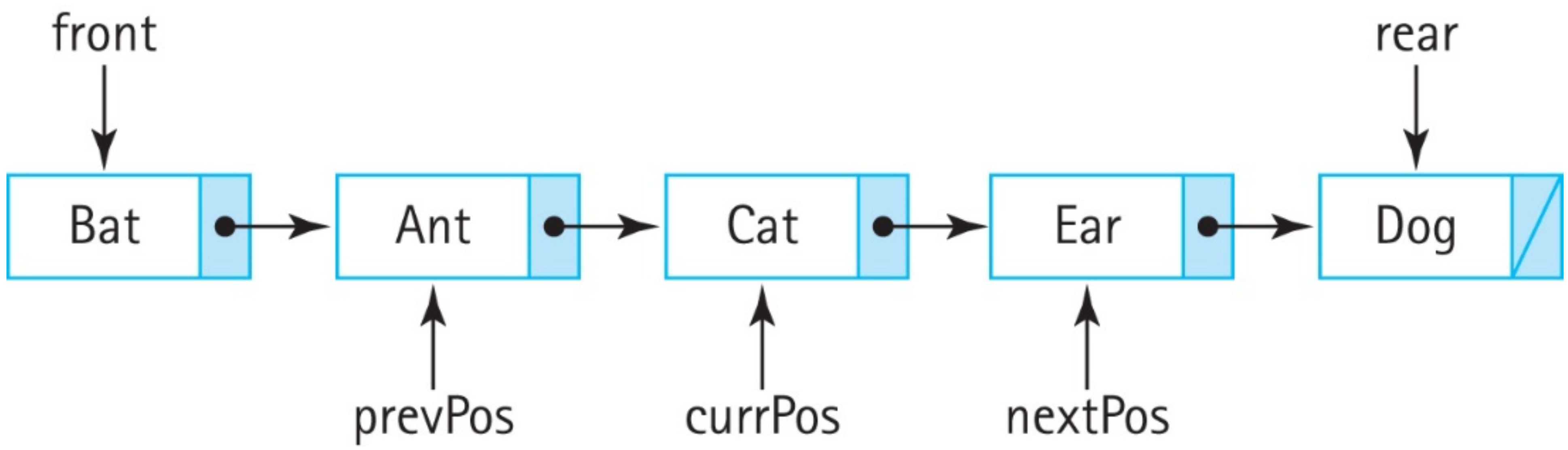
Name	Mar-ginal Letter	Map	Name	Mar-ginal Letter	Map	Name	Mar-ginal Letter	Map	Name	Mar-ginal Letter	Map
AACHEN	C III	41	Africa		36	Albania	B V	22	Algiers	D IV	99
	C III	42		C III	14		C VI	6		F V	13
	D VI	7		C IV	22		C VI	94	Alkmaar	B III	40
Aar, R.	C VII	87		D III	98		C VI	99	Allahabad	B IV	74
Abbasides	D VII	10		D IV	94		C VIII	2	Alle, R.	A X	93
	G XIV	13		D IV	99		D VII	86	Alleghany, Mts.	C III	77
Abbasides, Cali-				D V	9	Albany	B IV	77	Alleghany, R.	B IV	77
phate of	C VI	6		D VI	39	Alberche, R.	B III	98	Aller, R.	A I	73
Abbeville	A IV	17		E III	60	Albert	B VI	20	Allier	C V	87
	B V	20		E III	66	Albis, R.	C IV	1	Allier, R.	C V	17
Abbey of Parc	C IV	59		E X	76	Albret	D III	17		C V	18
Aberffraw	B II	11		F III	38		E III	18		E V	21
Aberystwith	D II	19		G III	1		E III	21		F VI	11
Aboukir Bay	D VII	94	Agen	D III	86	Albuera	C II	98	Almaden, Sa. de	C III	98
	D VII	99		D IV	80	Alby	E V	87	Almanza	E II	60
Abraham,				D IV	87	Alcala		23		E II	86
Heights of		72	Agenois	D IV	18	Alcantara	C II	98	Almaraz	C III	98
Abrantes	C L	98	Agincourt	A V	18	Alcaraz, S. de	C IV	98	Almeida	B II	98
Abus, Fl., Hum-				B VI	20	Alcase	D III	54	Almenara	D II	60
ber	D VIII	3	Agra	B III	74	Alcuyd	A II	8	Almohades,		
Acre	D VII	9		F II	75		A III	4	Km. of the	D II	10
	D VII	94		F VI	75		B IV	5		G IV	13
	D VII	99	Agram	C VI	68		C IV	7	Alnwick	A IV	11
	G XI	13		C VI	86	Alderney	B I	15		A IV	12
Adda, R.	B III	85	Aiguillon	D IV	17		B II	17		A V	8
	D VIII	87	Ain	C VI	87		B II	18		A V	19
Aderklaa		92	Airaines	B IV	17		B II	80	Alost	B III	59
Adige, R.	B IV	85		C V	20		B II	87	Alpes, Basses	D VII	87
	C V	84	Aire, R.	B VI	87		C II	21	Alpes, Hautes	D VII	87
Adour, R.	A V	98		C V	8		D III	11	Alps	D IV	38
	E III	17	Aisne	B V	87		D III	12		E VII	21
	E III	18	Aisne, R.	B VI	87	Alemanni	B II	*2	Alresford	F VI	43
	E III	87		C V	21		C IV	2	Alsace		82
	J III	21		D I	41	Alemannia	B II	6		B VII	80
	G IV	11		D I	42	Alemtejo	C II	98		C IV	60
	G IV	12		D I	54	Alençon	B II	15		C IV	66
Adramyttium	F X	13		E II	40		B IV	17		C IV	68
Adrianople	B V	14	Aix	E VI	87		B IV	87		D III	41
	B V	22	Aix-la-Chapelle	A II	6		C III	21	Alsatia	B II	6
	B VI	9		B IV	60		D V	11	Altamaha, R.	D III	77
	C V	67		B IV	68		D V	12	Altdorf	A III	85
	D VIII	68		B VII	21	Aleppo	C V	6	Alton		51
	D VIII	86	Ajaccio	C IV	94	Alessandria	B III	85	Amand, St.	C II	59
Adriatic Sea	B III	6	Ajmir	B II	74	Alexander, I.		91	Amboise	C IV	17
	B V	63	Akeman, St.	F VI	3	Alexandria	C IV	6		D III	38
	B V	85	Alabama, R.	D II	77		D V	10		D IV	21
	C V	94	Alagon, R.	B II	98		D V	67	Ambrieres	D IV	11
	C V	99	Alamanni	D IV	1		D VI	2	America, North	C IV	76
	D V	86	Aland, Is.	A V	67		D VI	9	Amesbury		51
	D VI	68	Alani	C VIII	2		D VI	94	Amiens	B V	17
Ægyptus	D VI	2		D XI	1		D VI	99		B V	80
	H VIII	1	Alania	B VI	6		G X	13		B V	87
Æmilia	B III	*2		C VII	10	Algarves	D I	98		C V	21
Aerschot	B IV	59		E XIII	13	Algiers	D III	10		C VI	11
Æstians	B VI	2	Alava	A IV	98		D III	67		C VI	20
Afghanistan	A I	74	Albania	B V	14		D IV	94		D V	7

iterator() changes

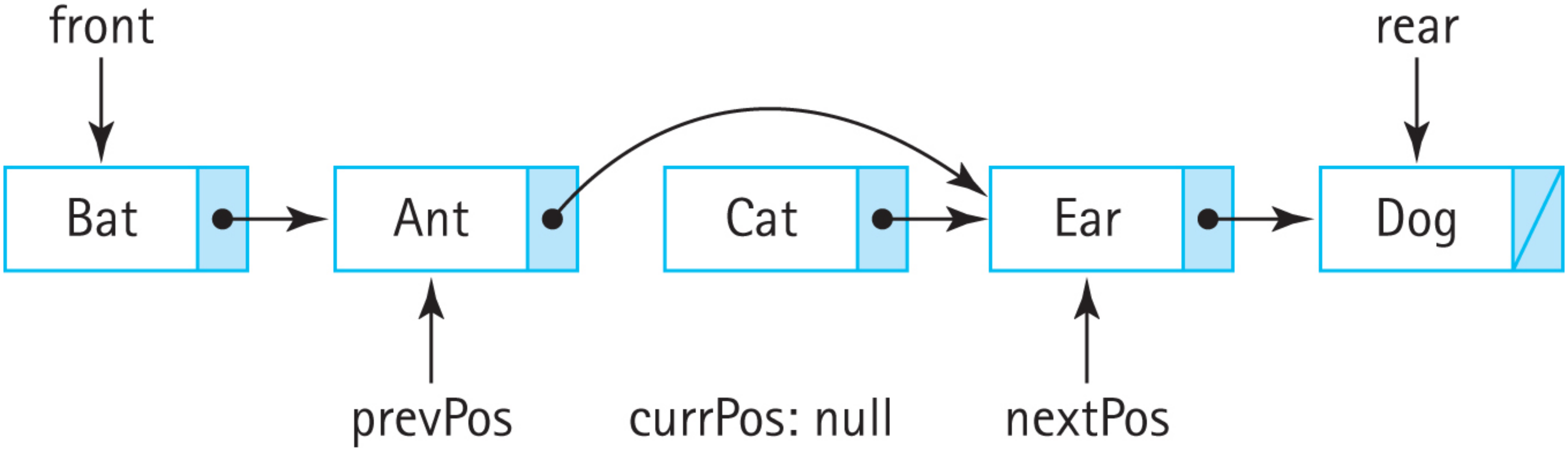
- We again implement the iterator as an anonymous class
- Note the iterator object only moves forward
    - No doubly linked list required
- What special consideration is needed for each method?

remove()

- Recall: removing a node requires modifying the *previous* node
- Our iterator will maintain a previous and current pointer
- For simplicity, we also maintain a next pointer



remove()



Application: Cards

- A deck of cards can be simulated using a list
  - Linear order of cards matches the order of the deck
- CardDeck implemented as a list of Card objects
  - Note the enum types used to specify constants
- Both GUI and CLI interfaces implemented

Examples

- Arrange a hand
- Higher or Lower game
- Probability simulation
- Try them out on your own!

CigarInspector.com
KING
"I'm not sure who it is, but I heard one of these guys tends to smoke a lot when he has a bad hand."

List Variations

- Sorted Lists
  - Does not allow the optional set() and add() methods
  - Allows comparison to be re-defined using Java Comparators
- Linked lists *inside* arrays
  - Store an array of nodes
  - Instead of a pointer to a node, `.next` holds the index of the next node
- Java Lists
  - Contains 28 abstract methods in the List interface!

Recap

- Lists maintain a linear ordering of objects
- Lists can be used like previous data structures but more flexibly
- Iterator objects allow for simple iteration through list elements
- Lists can be used for a wide range of applications
  - Example: card deck
  - Consider: anywhere an array could be used without null gaps in entries

Next Time...

- Dale, Joyce, Weems Chapter 7.1-3
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/