CSSE 220 Day 26

Linked List Implementation
Data-structure-palooza

Checkout *LinkedLists* project from SVN

Questions

Data Structures

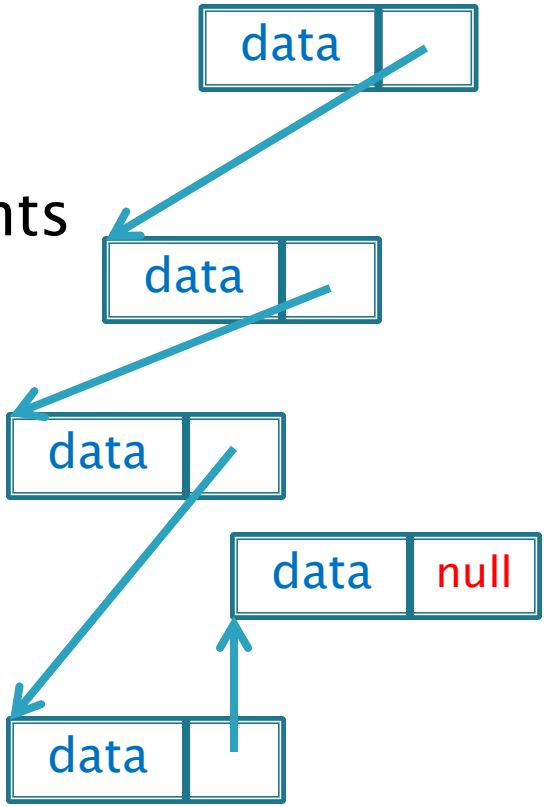
- Understanding the engineering trade-offs when storing data

Data Structures Recap

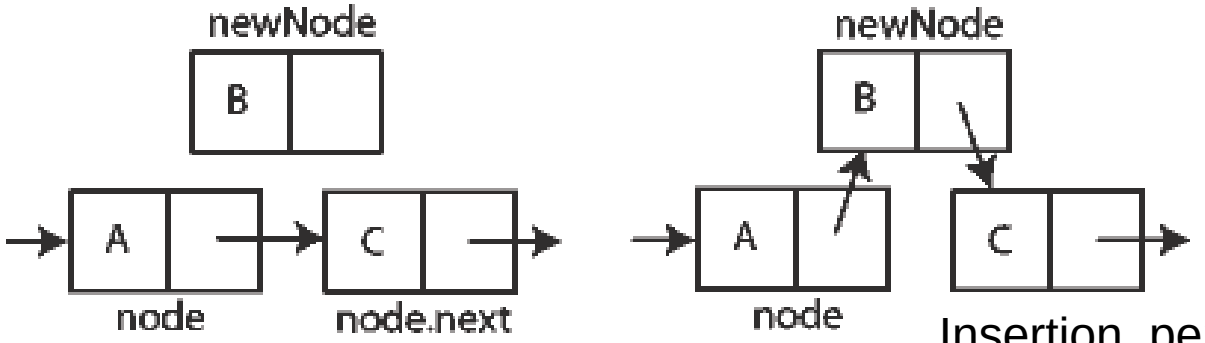
- Efficient ways to store data based on how we'll use it
- The main theme for the last 1/6 of the course
- So far we've seen `ArrayLists`
  - Fast addition **to end of list**
  - Fast access to any existing position
  - Slow inserts to and deletes from middle of list

Another List Data Structure

- What if we have to add/remove data from a list frequently?
- A `LinkedList` supports this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements



"random access"



Insertion, per Wikipedia

LinkedList<E> methods

- `void addFirst(E element)`
  `E getFirst()`
  `E removeFirst()`

What would you expect the run-time of these operations to be?

Answer: O(1) [do you see why?]

- `E get(int k)`

What would you expect the run-time of this operation to be, in terms of k? For a worst-case value of k?

Answer: O(k) to get the kth element, worst-case is O(n) where n is the length of the list [do you see why?]

- What if you want to access the rest of the list?

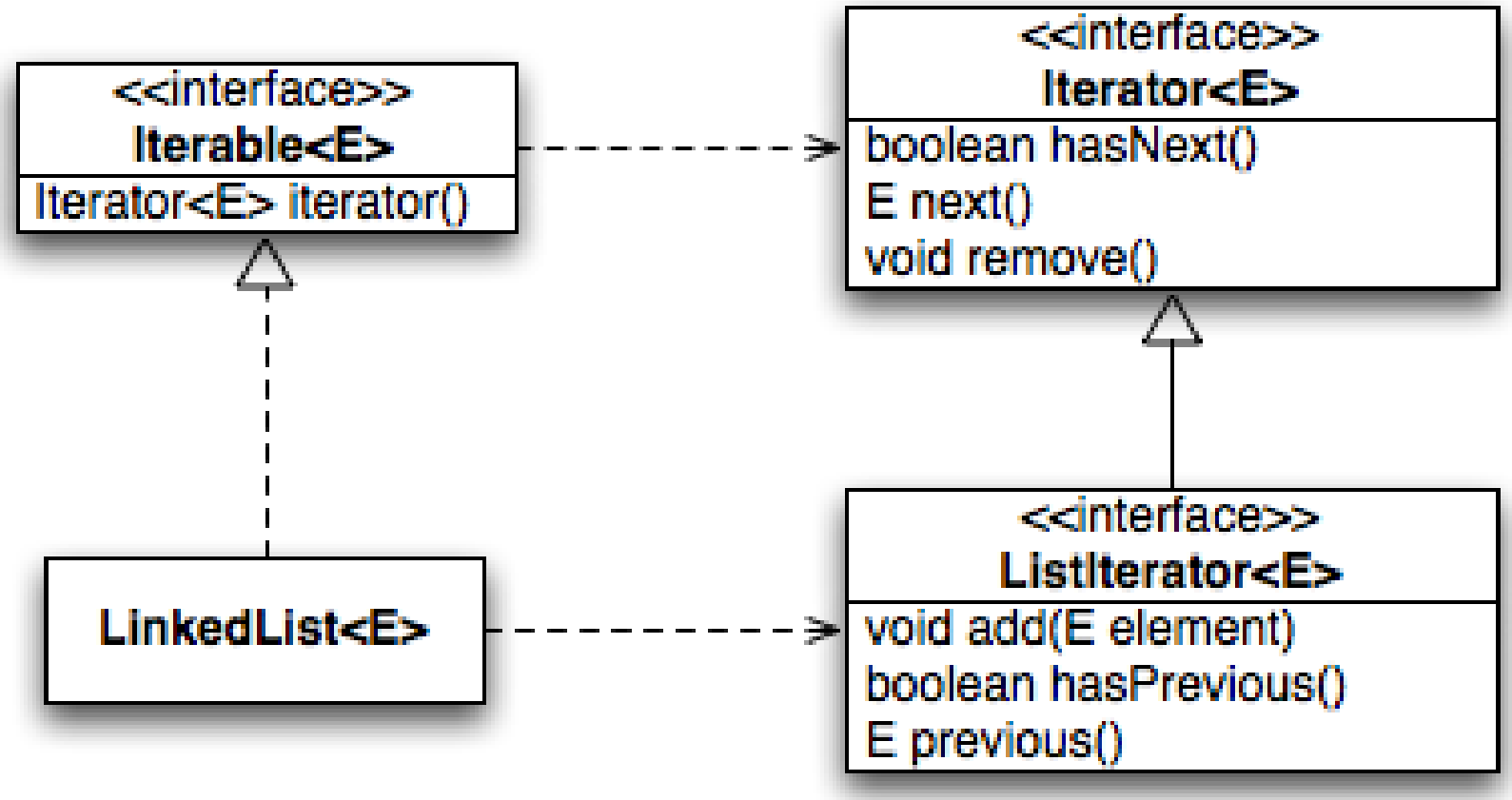
  `Iterator<E> iterator()`
  - `An iterator<E> has methods:`
    - `boolean hasNext()`
    - `E next()`
    - `E remove()`

What do you think these methods do? In particular, what element should *remove* remove?

What would you expect the run-times of these operation to be?

Answer: O(1) [do you see why?]

Accessing the Middle of a LinkedList



An Insider's View

```
for (String s : list) {
  // do something
}
```

```
Iterator<String> iter =
    list.iterator();

while (iter.hasNext()) {
  String s = iter.next();
  // do something
}
```

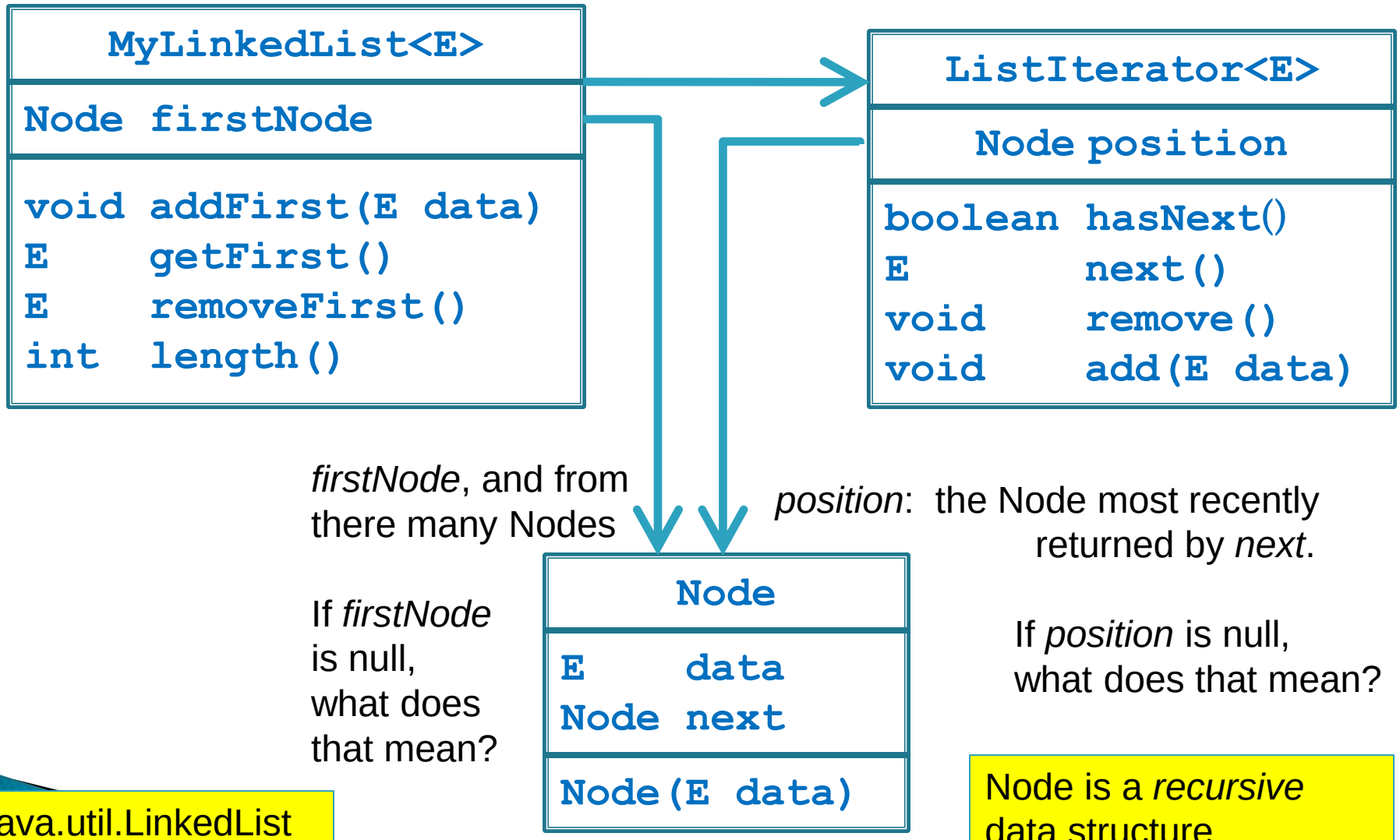
Enhanced For Loop

What Compiler Generates

Implementing LinkedList

- A simplified version, with just the essentials
- Won't implement the java.util.List interface
- Will have the usual linked list behavior
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow random access

Your implementation of LinkedList



firstNode, and from there many Nodes

If *firstNode* is null, what does that mean?

position: the Node most recently returned by *next*.

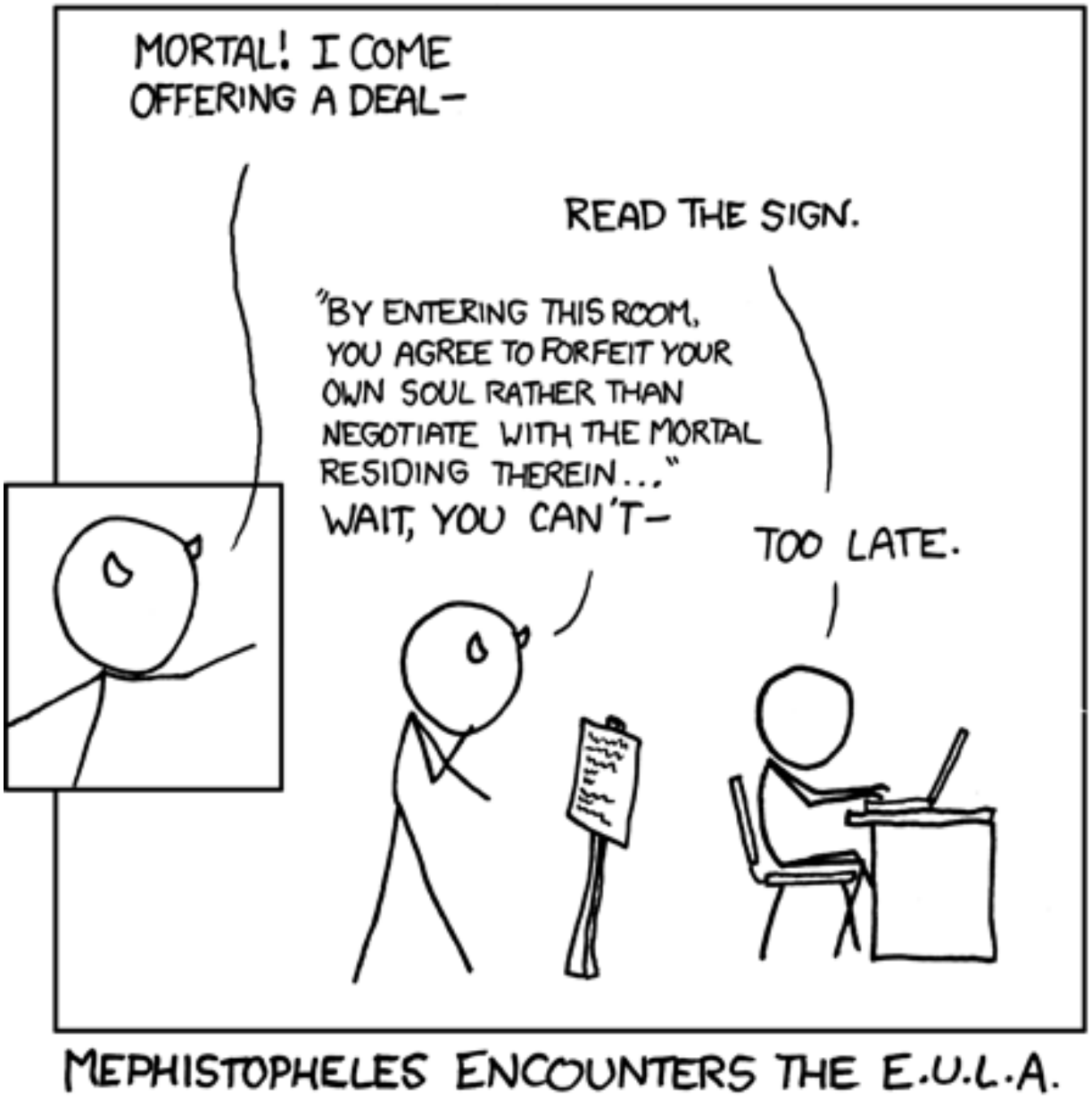
If *position* is null, what does that mean?

java.util.LinkedList has many more methods

Node is a *recursive* data structure

Faust 2.0

The only blood these contracts are signed in is from me cutting my hand trying to open the d@^mned CD case.



Abstract Data Types (ADTs)

- Boil down data types (e.g., lists) to their essential operations
- Choosing a data structure for a project then becomes:
  - Identify the operations needed
  - Identify the abstract data type that most efficient supports those operations
- Goal: that you understand several basic abstract data types and when to use them

Common ADTs

- Array List
- Linked List
- Stack
- Queue
- Set
- Map

Implementations for all of these are provided by the Java Collections Framework in the **`java.util`** package.

Array Lists and Linked Lists

Operations Provided	Array List Efficiency	Linked List Efficiency
Random access	O(1)	O(n)
Add/remove item	O(n)	O(1)

Stacks

- A last-in, first-out (LIFO) data structure
- Real-world stacks
  - Plate dispensers in the cafeteria
  - Pancakes!
- Some uses:
  - Tracking paths through a maze
  - Providing "unlimited undo" in an application

Operations Provided	Efficiency
Push item	O(1)
Pop item	O(1)

Implemented by `Stack`, `LinkedList`, and `ArrayDeque` in Java

Q1

Queues

- A first-in, first-out (FIFO) data structure
- Real-world queues
  - Waiting line at the BMV
  - Character on Star Trek TNG
- Some uses:
  - Scheduling access to shared resource (e.g., printer)

Operations Provided	Efficiency
Enqueue item	O(1)
Dequeue item	O(1)

Implemented by **`LinkedList`** and **`ArrayDeque`** in Java

Q1

Sets

- **Unordered** collections **without duplicates**
- Real-world sets
  - Students
  - Collectibles
- Some uses:
  - Quickly checking if an item is in a collection

Operations	HashSet	TreeSet
Add/remove item	O(1)	O(log n)
Contains?	O(1)	O(log n)

Can hog space

Sorts items!

Q1

Maps

- Associate keys with values
- Real-world "maps"
  - Dictionary
  - Phone book
- Some uses:
  - Associating student ID with transcript
  - Associating name with high scores

Operations	HashMap	TreeMap
Insert key-value pair	O(1)	O(log n)
Look up value for key	O(1)	O(log n)

Can hog space

Sorts items by key!

Q1