CSSE 220 Day 26

Linked List Implementation

Checkout *LinkedListSimple* project from SVN

Understanding the engineering trade-offs when storing data

DATA STRUCTURES

Data Structures

- Efficient ways to store data based on how we'll use it
- The main theme for the rest 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

Big-O Notation

- Describes the limiting behavior
  - How slow it can possibly run?
  - Describes the worst case
- Used for Classifying Algorithm Efficiency
- “O” for “Order”
  - O(n) → said as “Order n”
  - O(n^2) → said as “Order n-squared”

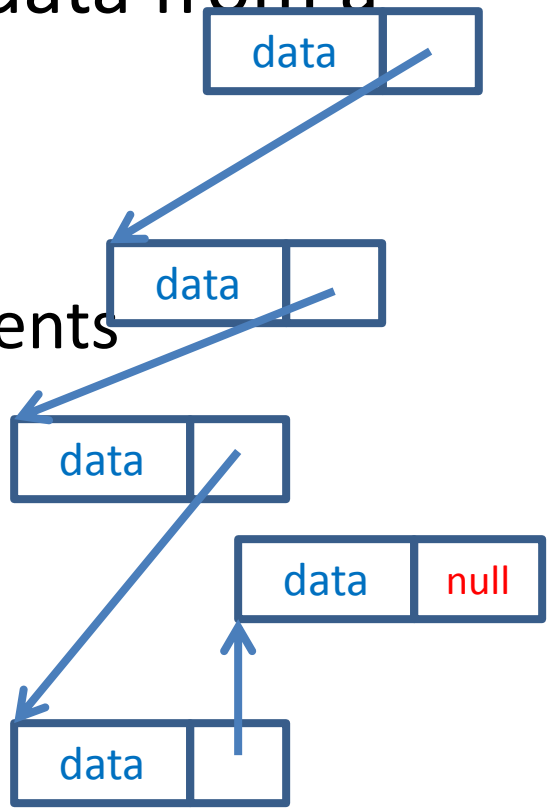
Big-O Notation (continued)

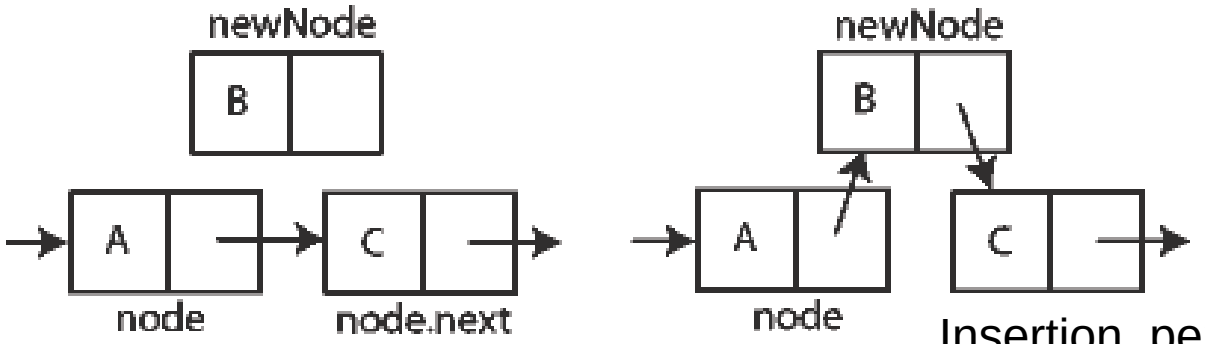
- Don’t Care About Constants
  - O(2n + 7) → O(n)
- Don’t Care About Smaller Powers
  - O(6n^2 + 7n) →O(n^2)
  - Algorithm grows asymptotically no faster than n^2
- If constant value, we say O(1) → “Order 1”
  - O(48) → O(1)

- Get into pairs
- Look at/run the code in LinkedList.java main
- Draw a boxes and pointer diagram of what's happening in the main code (your diagram should show the final state). To figure it out you'll have to look at the LinkedList constructor and addAtBeginning.
- If you've forgotten how to do a boxes and pointer diagram, checkout the handout on Day 4 of the schedule

Another List Data Structure

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





Insertion, per Wikipedia

LinkedList<E> Methods

- **void addFirst(E element)**
- **void addLast(E element)**
- **E getFirst()**
- **E getLast()**
- **E removeFirst()**
- **E removeLast()**

- What about accessing the middle of the list?
  - **LinkedList<E> implements Iterable<E>**

Linked Lists Day 2

Linked List Implementation

Checkout *SinglyLinkedList* project from SVN

3 Topics

- Generics
- Iterators
- Debugging

Let’s modify our simple linked list to take arbitrary objects!

- Two ways:
  - Object
  - Generics

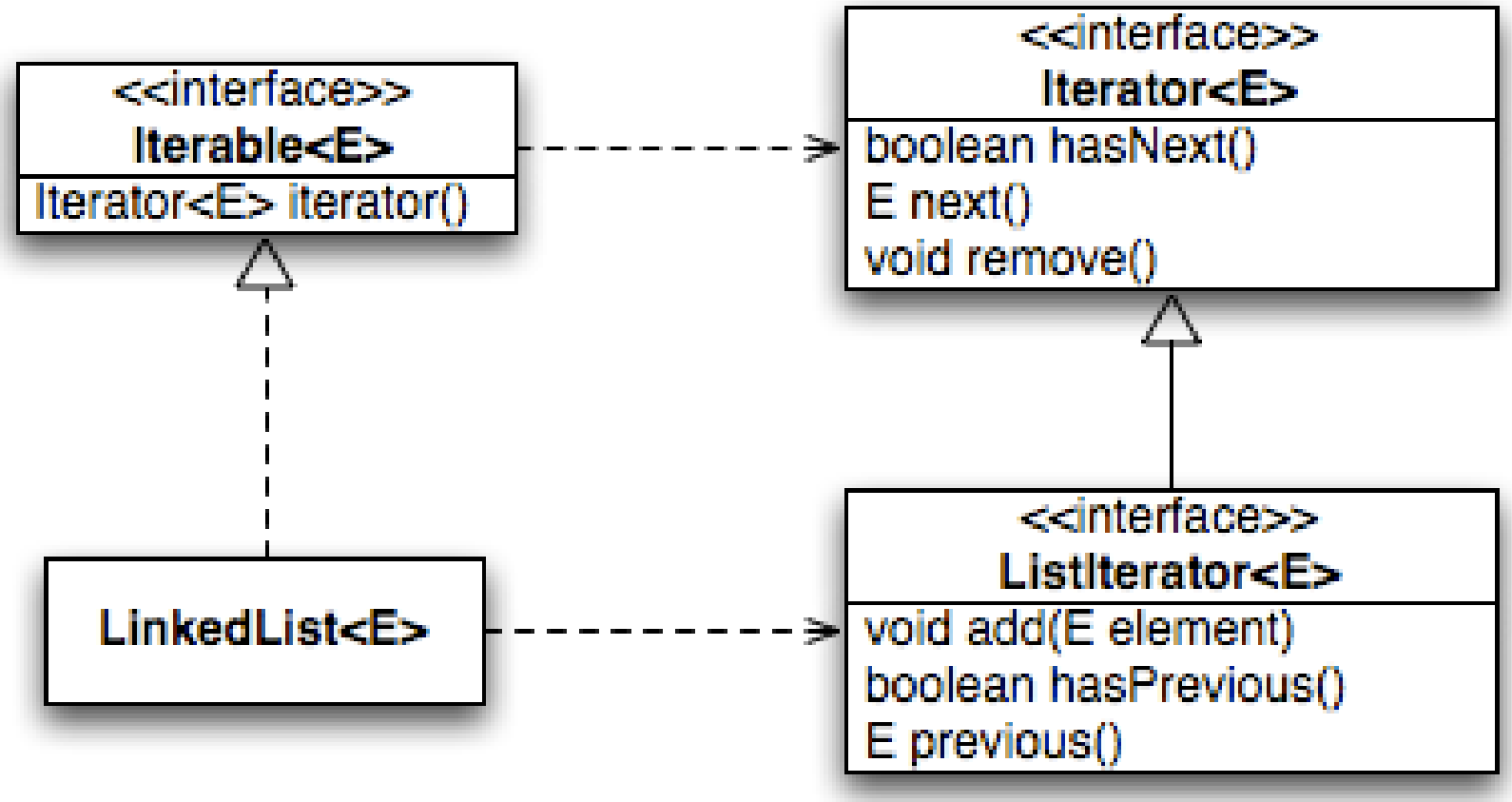
Generics Advanced

- Type parameters:
  - `class DLList<E>`
- Bounds:
  - `class DLList<E extends Comparable>`
  - `class DLList<E extends Comparable<E>>`
  - `class DLList<E extends Comparable<? super E>>`
- Generic methods:
  - `public static <T> void shuffle(T[ ] array)`
- http://docs.oracle.com/javase/tutorial/java/generics/index.html

What are iterators and why do they exist?

- Iterators are objects designed to encapsulate a position in a data structure – in the case, a pointer to a current (and previous) node in a list
- Your textbook has a detailed discussion of the operation of linked list iterators, including lots of sample code

Accessing the Middle of a LinkedList



An Insider's View

Enhanced For Loop

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

What Compiler Generates

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

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

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

Using the debugger

LodeRunner next cycle due next class

TEAM PROJECT WORK TINE