

# 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

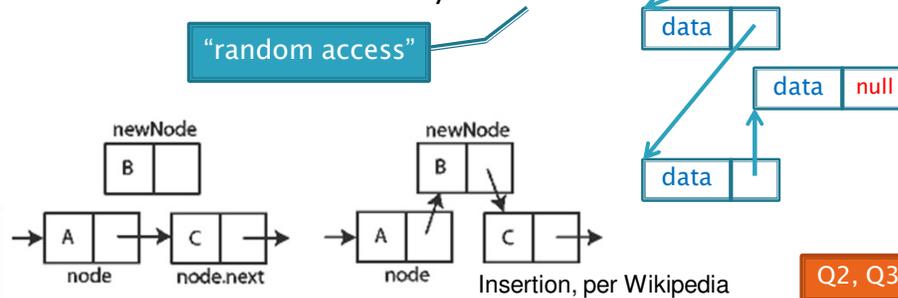
## Data Structures

- ▶ 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

Q1

## Another List Data Structure

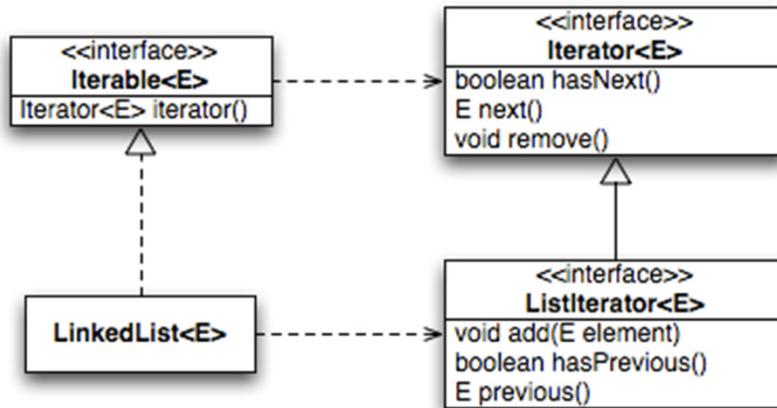
- ▶ What if we have to add/remove data from a list frequently?
- ▶ **LinkedLists** support this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements



## 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>**

## 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

## 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 efficiently 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

Q4

## 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

Q5

## 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(lg n)
Contains?	O(1)	O(lg n)

Can hog space

Sorts items!

Q6

## 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(lg n)
Look up value for key	O(1)	O(lg n)

Can hog space

Sorts items by key!

Q7-9

# Team Project Work Time

»» If by some miracle there is  
anytime left