# Lists and Iterators

CSSE 221

Fundamentals of Software Development Honors

Rose-Hulman Institute of Technology

# Announcements

Understanding the engineering trade-offs when storing data

# Data Structures

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Data Structures

- Efficient ways to store data based on how we'll use it

- 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

# Data Structures and the Java Collections Framework

- An approach to storing several items of the same type.
- Three aspects:
  - Specification (interface)
  - Implementation (sometimes several alternate implementations)
  - Applications (how can it be used?)

# 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
  - Sketch one now

"random access"

# LinkedList<E> methods

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

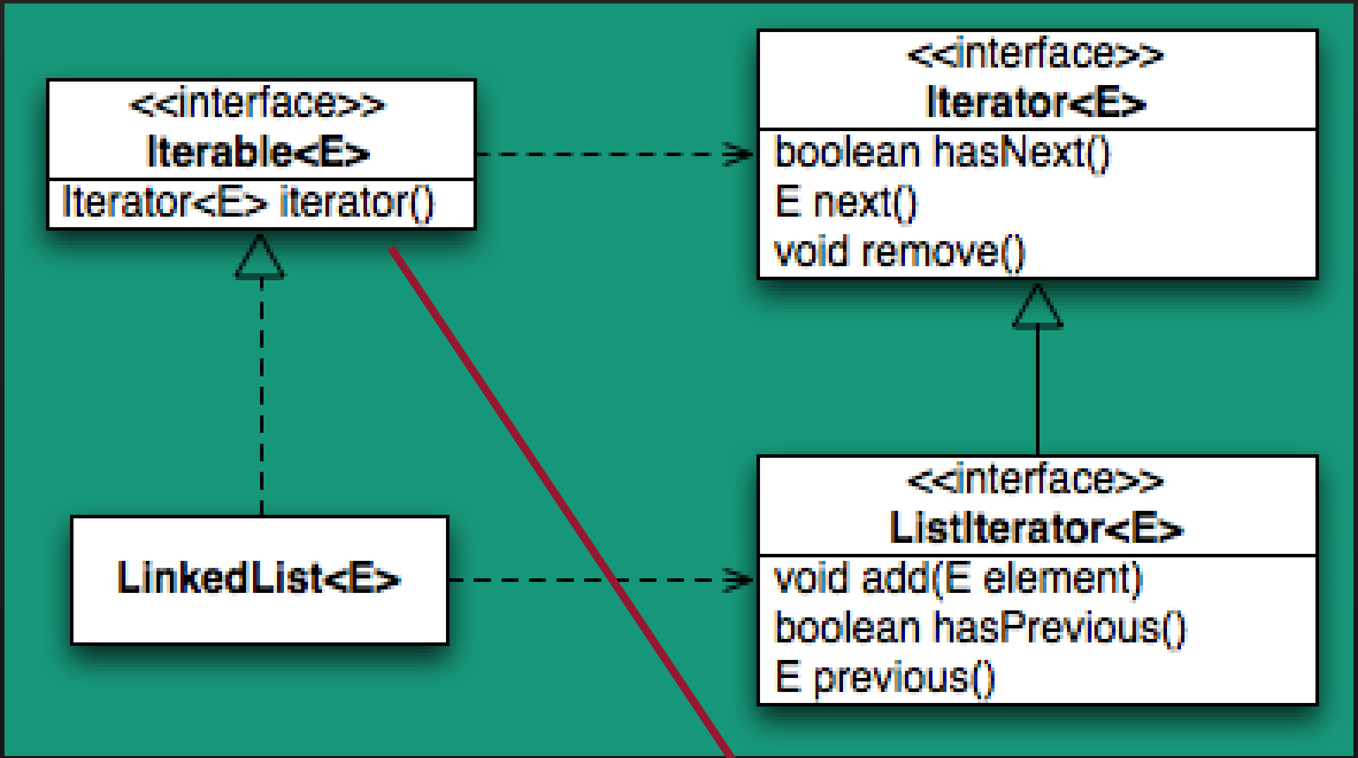
- E get(int k)

# LinkedList<E> iterator

- 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 should remove() remove?

# Accessing the Middle of a LinkedList




- iterator() is what is called a *factory method*: it returns a new concrete iterator, but using an interface type.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# 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
}
```

How to use linked lists and iterators

# Demo

More with big-Oh

# Abstract Data Types

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# 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

# Arrays

- Size must be declared when the array is constructed.
- We access items by *index*

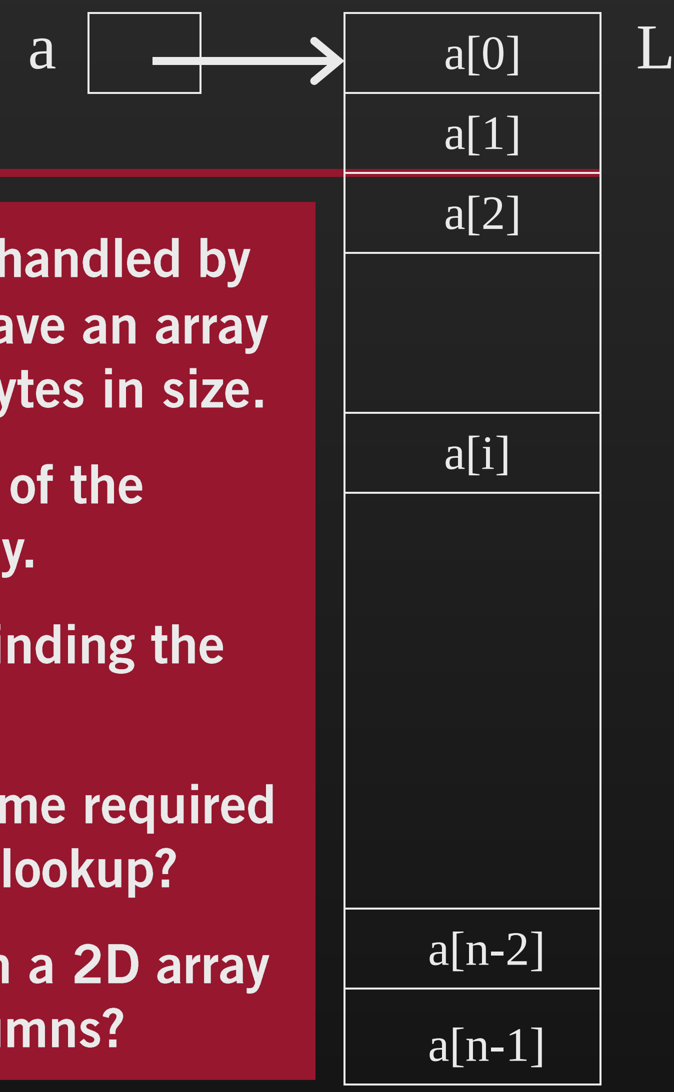
**Implementation** **(handled by the compiler): We have an array of N items, each b bytes in size.**

**Let L be the address of the beginning of the array.**

**What is involved in finding the address of a[i]?**

**What is the Big-oh time required for an array-element lookup?**

**What about lookup in a 2D array of n rows and m columns?**



# What about Array Lists?

- We said **Array Lists** have
  - Fast addition to end of list
  - Fast access to any existing position
  - Slow inserts to and deletes from middle of list

- Big-Oh runtimes of each?

# Runtimes of LinkedList methods

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

- E get(int k)

- To access the rest of the list: Iterator<E> iterator()
  - boolean hasNext()
  - E next()
  - E remove()

# Summary

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

# Common ADTs

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

- Look at the *Collections* interface now.

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

# A longer list

- Array (1D, 2D, …)
- List
  - ArrayList
  - LinkedList
- Stack
- Queue
- Set
- MultiSet
- Map (a.k.a. table, dictionary)
  - HashMap
  - TreeMap
- PriorityQueue
- Tree
- Graph
- Network

What is "special" about each data type?

What is each used for?

What can you say about time required for:
adding an element?
removing an element?
finding an element?

You will know these, inside and out, by the end of CSSE230.