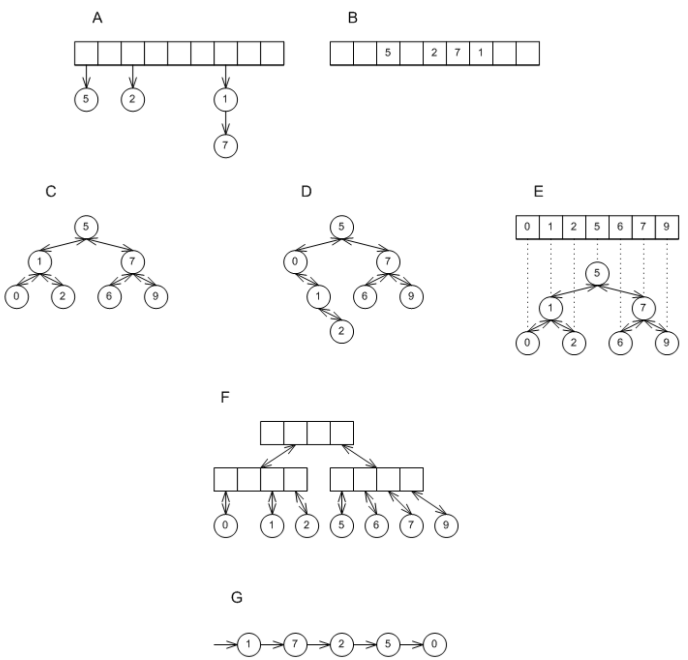
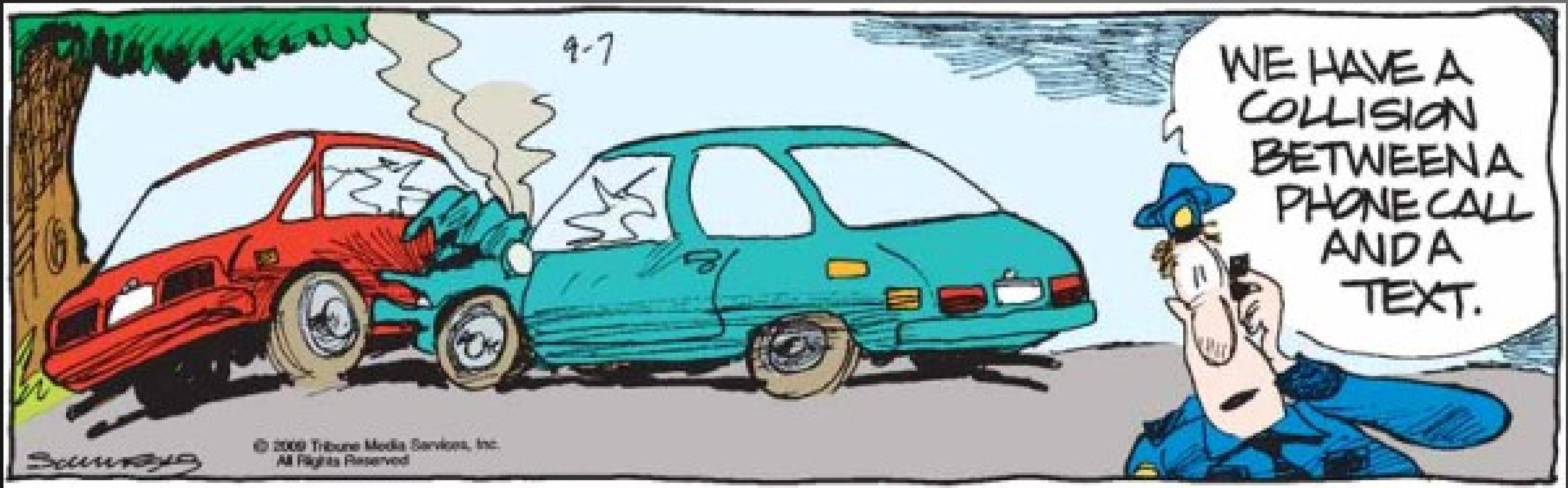
CSSE 230 Day 5

Abstract Data Types
Data Structure "Grand Tour"
Java Collections



http://gcc.gnu.org/onlinedocs/libstdc++/images/pbds_different_underlying_dss_1.png

Questions?


9-7
WE HAVE A COLLISION BETWEEN A PHONE CALL AND A TEXT.
© 2009 Tribune Media Services, Inc. All Rights Reserved

ADTs and Data Structures

What is data?

What do we mean by structure?

A *data type* is an interpretation of bits

- A set of operations
- May be provided by the hardware (`int` and `double`)
- By software (`java.math.BigInteger`)
- By software + hardware (`int[]`)

What is an Abstract Data Type (ADT)?

- A mathematical model of a data type
- Specifies:
  - The type of data stored
  - The operations supported
  - Argument types and return types of these operations

  - What each operation does, but not how

An Example ADT: Non-negative integers

- One special value: ***zero***
- Three basic operations:
  - *succ*
  - *pred*
  - *isZero*
- Derived operations include ***plus***
- Sample rules:
  - *isZero(succ(n))* ➔ *false*
  - *pred(succ(n))* ➔ *n*
  - *plus(n, zero)* ➔ *n*
  - *plus(n, succ(m))* ➔ *succ(plus(n, m))*

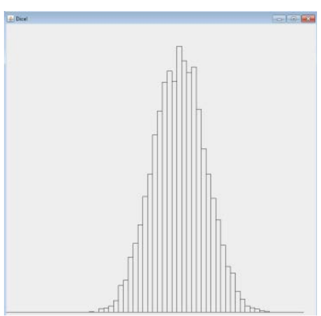
Q1

Data Structures are ADTs for collections of items

Application:
"how can you use that?"

```java
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ArrayList<SingleDie> dice = new ArrayList<SingleDie>();
    while (true) {
        System.out.printf("How many sides (Q to quit):");
        String response = scanner.next();
        if (Character.toUpperCase(response.charAt(0)) == 'Q') {
            break;
        }
        int nSides = Integer.parseInt(response);
        nSides = (nSides < 4) ? 4: nSides;
        dice.add(new SingleDie(nSides));
    }

    scanner.close();
    int minSum = dice.size();
    int maxSum = 0;
    for (SingleDie die : dice) {
        maxSum += die.getNSides();
    }
```



Specification
"what is it?"

Constructor Summary

ArrayList()
Constructs an empty list with

ArrayList(Collection<? extend
Constructs a list containing th

ArrayList(int initialCapacity
Constructs an empty list with

Method Summary

boolean	**add**(E e) Appends the speci
void	**add**(int index, E ele Inserts the specifie
boolean	**addAll**(Collection<? Appends all of the
boolean	**addAll**(int index, Co Inserts all of the el
void	**clear**() Removes all of the

Implementation:
"How do you do that?"

```java
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneabl
{
    private static final long serialVersionUID = 8

    /**
    private transient Object[] elementData;

    /**
    private int size;

    /**
    public ArrayList(int initialCapacity) {
        super();
        if (initialCapacity < 0)
            throw new IllegalArgumentException("Il
                                               ini
        this.elementData = new Object[initialCapac
    }

    /**
    public ArrayList() {
        this(10);
    }
```

CSSE220

CSSE230

Q2

Data Structures Grand Tour

- Some review
- Some new
- All will appear again

Common ADTs

- Array
- List
  - Array List
  - Linked List
- Stack
- Queue
- Set
  - Tree Set
  - Hash Set
  - Linked Hash Set
- Map
  - Tree Map
  - Hash Map
- Priority Queue
- Tree
- Graph
- Network

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

*Exceptions: Tree, Graph, Network

An exploration of Java's Collection

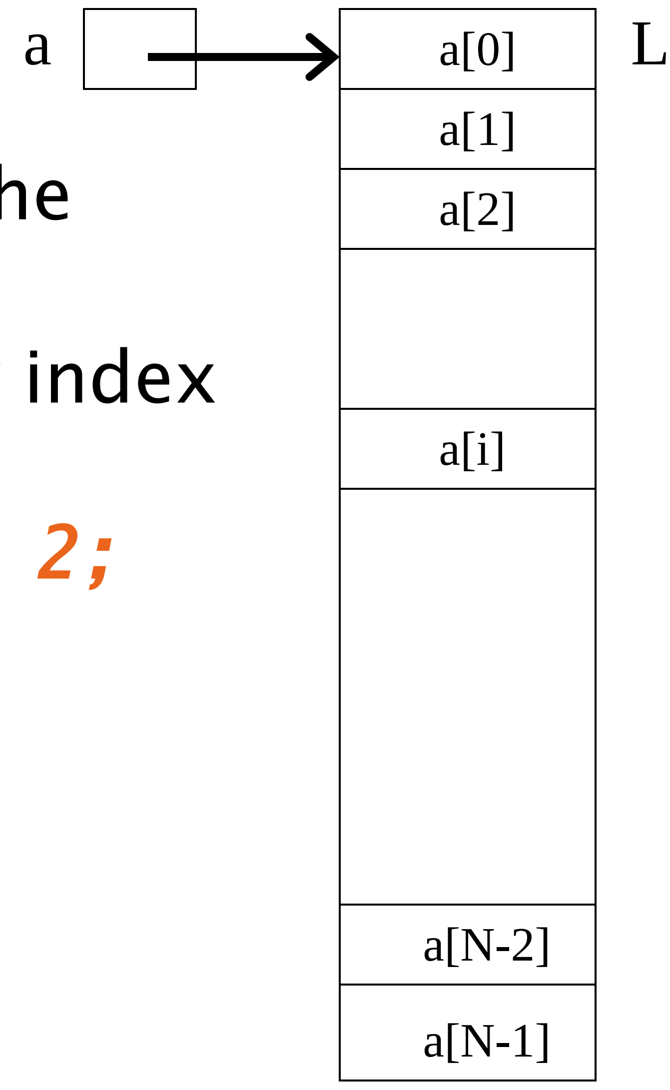
- Search for *Java 7 Collection*
- With a partner, read the javadocs to answer the quiz questions. You only need to submit one quiz per pair.

- I have used the slides that follow when teaching CSSE230 before.
  - They may be a good reference

- When you finish, you may work on your current CSSE230 assignments

Q3-11

Array



- Size must be declared when the array is constructed
- Can look up or store items by index
  Example:

```
nums[i+1] = nums[i] + 2;
```

- How is this done?

List

- A list is an ordered collection where elements may be added anywhere, and any elements may be deleted or replaced.
- **Array List:** Like an array, but growable and shrinkable.
- **Linked List**:

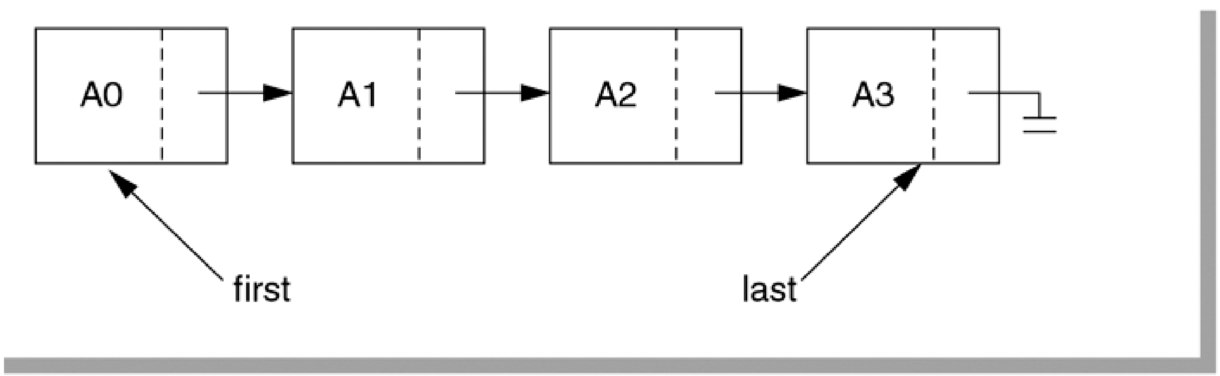


figure 6.19

A simple linked list

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)

Stack

- 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

```
public static void printInReverse(List<String> words) {
    // TODO: implement
    Stack<String> stack = new Stack<String>();
    for (String w : words) {
        stack.push(w);
    }
    while (!stack.isEmpty()) {
        System.out.println(stack.pop());
    }
}
```

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

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

Queue

```java
/**
 * Uses a queue to print pairs of words consisting of
 * a word in the input and the word that appeared five
 * words before it.
 *
 * @param words
 */
public static void printCurrentAndPreceding(List<String> words) {
    // TODO: implement
    ArrayDeque<String> queue = new ArrayDeque<String>();
    // Preloads the queue:
    for (int i = 0; i < 5; i++) {
        queue.add("NotAWord");
    }
    for (String w : words) {
        queue.add(w);
        String fiveAgo = queue.remove();
        System.out.println(w + ", " + fiveAgo);
    }
}
```

- 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

Set

- A collection of items **without duplicates** (in general, order does not matter)
  - If *a* and *b* are both in set, then *!a.equals(b)*
- Real-world sets:
  - Students
  - Collectibles
- One possible use:
  - Quickly checking if an item is in a collection

```java
public static void printSortedWords(List<String> words) {
    TreeSet<String> ts = new TreeSet<String>();
    for (String w : words) {
        ts.add(w);
    }
    for (String s : ts) {
        System.out.println(s);
    }
}
```

Example from 220

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

Can hog space

Sorts items!

Map

How is a TreeMap like a TreeSet?
How is it different?

- 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 the value associated with a given key	O(1)	O(log n)

Can hog space

Sorts items by key!

HashMap/HashSet Example (220)

```java
public static void printWordCountsByLength(List<String> words) {
    HashMap<Integer, HashSet<String>> map =
        new HashMap<Integer, HashSet<String>>();

    for (String w : words) {
        int len = w.length();
        HashSet<String> set;
        if (map.containsKey(len)) {
            set = map.get(len);
        } else {
            set = new HashSet<String>();
            map.put(len, set);
        }
        set.add(w);
    }
    System.out.printf("%d unique words of length 3.%n", getCount(map, 3));
    System.out.printf("%d unique words of length 7.%n", getCount(map, 7));
    System.out.printf("%d unique words of length 9.%n", getCount(map, 9));
    System.out.printf("%d unique words of length 15.%n", getCount(map, 15));
}
```

```java
public static int getCount(HashMap<Integer, HashSet<String>> map, int key) {
    if (map.containsKey(key)) {
        return map.get(key).size();
    } else {
        return 0;
    }
}
```

Priority Queue

Not like regular queues!

- Each **item** stored **has an** associated **priority**
  - Only item with "minimum" priority is accessible
  - Operations: ***insert***, ***findMin***, ***deleteMin***
- Real-world "priority queue":
  - Airport ticketing counter
- Some uses
  - Simulations
  - Scheduling in an OS
  - Huffman coding

```
PriorityQueue<String> stringQueue =
     new PriorityQueue<String>();

stringQueue.add("ab");
stringQueue.add("abcd");
stringQueue.add("abc");
stringQueue.add("a");

while(stringQueue.size() > 0)
       System.out.println(stringQueue.remove());
```

Operations Provided	Efficiency
Insert	O(log n)
Find Min	O(log n)
Delete Min	O(log n)

The version in Warm Up and Stretching isn't this efficient.

Trees, Not Just For Sorting

- Collection of nodes
  - One specialized node is the root.
  - A node has one parent (unless it is the root)
  - A node has zero or more children.
- Real-world "trees":
  - Organizational hierarchies
  - Some family trees
- Some uses:
  - Directory structure on a hard drive
  - Sorted collections

Only if tree is "balanced"

Operations Provided	Efficiency
Find	O(log n)
Add/remove	O(log n)

Graphs

- A collection of nodes and edges
  - Each edge joins two nodes
  - Edges can be directed or undirected
- Real-world "graph":
  - Road map
- Some uses:
  - Tracking links between web pages
  - Facebook

Operations Provided	Efficiency
Find	O(n)
Add/remove	O(1) or O(n) or O(n^2)

Depends on implementation (time/space trade off)

Networks

- Graph whose edges have numeric labels
- Examples (labels):
  - Road map (mileage)
  - Airline's flight map (flying time)
  - Plumbing system (gallons per minute)
  - Computer network (bits/second)
- Famous problems:
  - Shortest path
  - Maximum flow
  - Minimal spanning tree
  - Traveling salesman
  - Four-coloring problem for planar graphs

Common ADTs

- Array
- List
  - Array List
  - Linked List
- Stack
- Queue
- Set
  - Tree Set
  - Hash Set
- Map
  - Tree Map
  - Hash Map
- Priority Queue
- Tree
- Graph
- Network

We'll implement and use nearly all of these, some multiple ways. And a few other data structures.

Data Structure Summary

Structure	find	insert/remove	Comments
Array	O(n)	can't do it	Constant-time access by position
Stack	top only O(1)	top only O(1)	Easy to implement as an array.
Queue	front only O(1)	O(1)	insert rear, remove front.
ArrayList	O(log N)	O(N)	Constant-time access by position
Linked List	O(n)	O(1)	O(N) to find insertion position.
HashSet/Map	O(1)	O(1)	If table not very full
TreeSet/Map	O(log N)	O(log N)	Kept in sorted order
PriorityQueue	O(log N)	O(log N)	Can only find/remove smallest
Tree	O(log N)	O(log N)	If tree is balanced
Graph	O(N*M) ?	O(M)?	N nodes, M edges
Network			shortest path, maxFlow

Work Time

If we have time left