

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

Announcements

- ▶ **Stacks and Queues**
 - Ideally, you have met with your partner to start
 - Try your best to work well together, even if you have different amounts of programming experience.

- ▶ **Finish day 4 + quiz with instructor if needed.**

- ▶ **Exam 1: next Wednesday, 7–9pm.**

How is Homework 2 coming?

- ▶ From question 2:

Suppose $T_1(N)$ is $O(f(N))$ and $T_2(N)$ is $O(f(N))$. **Prove** that $T_1(N) + T_2(N)$ is $O(f(N))$ or give a counter-example.

- Hint: Supposing $T_1(N)$ and $T_2(N)$ are $O(f(N))$, that means there exist constants c_1, c_2, n_1, n_2 , such that.....
 - How can you use them?
- ▶ What about the similar question for $T_1(N) - T_2(N)$?
 - Remember, O isn't a tight bound.

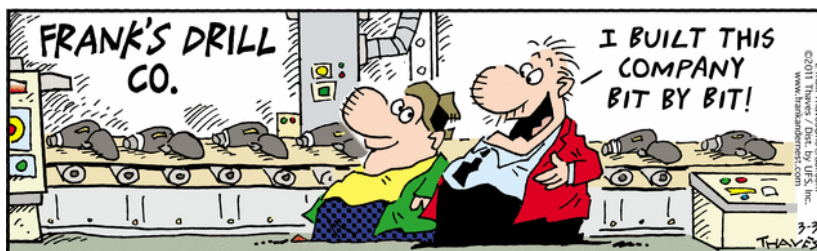
After today, you should be able to...

- ▶ explain what an Abstract Data Type (ADT) is
- ▶ List examples of ADTs in the Collections framework (from HW2 #1)
- ▶ List examples of data structures that implement the ADTs in the Collections framework
- ▶ Choose an ADT and data structure to solve a problem

ADTs and Data Structures

A *data type* is an interpretation of data (bits)

- “What is this data, and how does it work?”
- Primitive types (**int**, **double**): hardware-based
- Objects (such as **java.math.BigInteger**): require software interpretation
- Composite types (**int[]**): software + hardware



What is an Abstract Data Type (ADT)?

- ▶ A mathematical model of a data type
- ▶ Specifies:
 - The type of data stored (but not *how* it's stored)
 - The operations supported
 - Argument types and return types of these operations (but not *how* they are implemented)

An Example ADT: Stack

- ▶ Three basic operations:
 - **isEmpty**
 - **push**
 - **pop**
- ▶ Derived operations include **peek** (a.k.a. **top**)
 - How could we write it in terms of the basic operations?
 - We could have **peek** be a basic operation instead.
 - Advantages of each approach?
- ▶ Possible implementations:
 - Use a linked list.
 - Use a growable array.
 - Last time, we talked about implementation details for each.


ADTs for collections of items

Application:
"how can you use it?"

Specification
"what can it do?"

Implementation:
"How is it built?"

```
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.getSides();
    }
}
```



CSSE220

| Modifier and Type | Method and De |
|-------------------|---------------------------------|
| boolean | add(E e) |
| void | add(int index, E element) |
| boolean | addAll(Collection c) |
| boolean | addAll(int index, Collection c) |
| void | clear() |
| boolean | contains(Object o) |
| boolean | containsAll(Collection c) |
| boolean | equals(Object o) |
| E | get(int index) |

```
public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable {
    private static final long serialVersionUID = 0L;
    private transient Object[] elementData;
    private int size;

    public ArrayList(int initialCapacity) {
        super();
        if (initialCapacity < 0)
            throw new IllegalArgumentException("Illegal initial capacity");
        this.elementData = new Object[initialCapacity];
    }

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

CSSE230

Common collection ADTs and implementations (data structures!)

- ▶ List
 - Array List
 - Linked List
- ▶ Stack
- ▶ Queue
- ▶ Set
 - Tree Set
 - Hash Set
 - Linked Hash Set

- ▶ Map
 - Tree Map
 - Hash Map
- ▶ Priority Queue

Underlying data structures for many
Array
Tree

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

Java Collections Framework

Reminder: Available, efficient, bug-free implementations of many key data structures

Most classes are in `java.util`

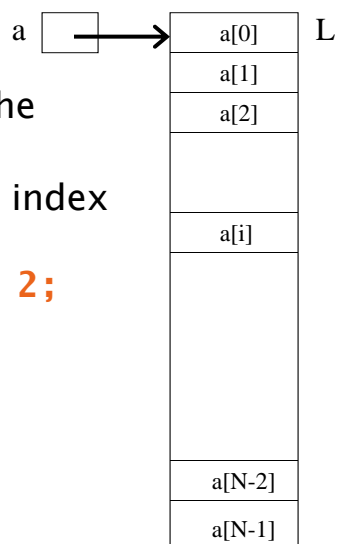
You started this in HW2 #1; Weiss Chapter 6 has more details

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:**

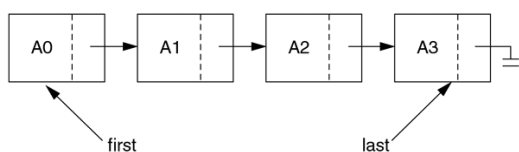


figure 6.19
A simple linked list

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

- ▶ 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)

```
/**
 * 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);
    }
}
```

| 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

```
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)

```
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));
}

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 | $O(1)$ | $O(\log N)$ | Can only find/remove smallest |
| Tree | $O(\log N)$ | $O(\log N)$ | If tree is balanced, $O(N)$ otherwise |
| Graph | $O(N*M)$? | $O(M)$? | N nodes, M edges |
| Network | | | shortest path, maxFlow |

Often, one particular ADT and implementation is best for the problem

- ▶ Which ADT to use?
 - It depends. How do you access your data? By position? By key? Do you need to iterate through it? Do you need the min/max?
- ▶ Which implementation to use?
 - It also depends. How important is fast access vs fast add/remove? Does the data need to be ordered in any way? How much space do you have?
- ▶ But real life is often messier...

Q1-9

How to figure this out?

- ▶ Use Java's Collections Framework.
 - Search for *Java 8 Collection*
 - With a partner, read the javadocs to answer the quiz questions. You only need to submit one quiz per pair. (Put both names at top)

- ▶ If you finish, you may work on your current CSSE230 assignments