What's in Compsci 100?

- Understanding tradeoffs: reasoning, analyzing, describing...
  - Algorithms
  - Data Structures
  - Programming
  - Design

- Object oriented programming using Java
  - IDE
  - Language
  - Problem-solving
  - From design to code

Toward understanding data structures

- What can be put in a TreeSet?
- What can be sorted?
  - Where do we find this information?
  - How do we understand the information?

- What can be put in an ArrayList? Why is this different?
  - What operations exist on an ArrayList?
  - What about an array, or operations done on an ArrayList as opposed to what an ArrayList does to itself?

What can an Object do (to itself)?

  - Look at java.lang.Object
  - What is this class? What is its purpose?

- toString()
  - Used to print (System.out.println) an object
  - overriding toString() useful in new classes
  - String concatenation: String s = “value “+x;
  - Default is basically a pointer-value

What else can you do to an Object?

- equals(Object o)
  - Determines if guts of two objects are the same, must override, e.g., for using a.indexOf(o) in ArrayList a
  - Default is ==, pointer equality

- hashCode()
  - Hashes object (guts) to value for efficient lookup

- If you’re implementing a new class, to play nice with others you must
  - Override equals and hashCode
  - Ensure that equal objects return same hashCode value
Objects and values

- **Primitive variables are boxes**
  - think memory location with value
- **Object variables are labels that are put on boxes**
  - String s = new String("genome");
  - String t = new String("genome");
  - if (s == t) {they label the same box}
  - if (s.equals(t)) {contents of boxes the same}

```
What's in the boxes? "genome" is in the boxes
```

Objects, values, classes

- **For primitive types: int, char, double, boolean**
  - Variables have names and are themselves boxes (metaphorically)
  - Two int variables assigned 17 are equal with ==
- **For object types: String, ArrayList, others**
  - Variables have names and are labels for boxes
  - If no box assigned, created, then label applied to null
  - Can assign label to existing box (via another label)
  - Can create new box using built-in new

Object types are references or pointers or labels to storage

David Parnas

"For much of my life, I have been a software voyeur, peeking furtively at other people's dirty code. Occasionally, I find a real jewel, a well-structured program written in a consistent style, free of kludges, developed so that each component is simple and organized, and designed so that the product is easy to change."

Parnas on re-invention

"We must not forget that the wheel is reinvented so often because it is a very good idea; I've learned to worry more about the soundness of ideas that were invented only once."
Module Design: Parnas wrote about the criteria for designing modules, in other words, the criteria for grouping functions together. This was a key predecessor to designing objects, and today's object-oriented design.

Social Responsibility: Parnas also took a key stand against the Strategic Defense Initiative (SDI) in the mid 1980s, arguing that it would be impossible to write an application that was free enough from errors to be safely deployed.

Professionalism: Parnas became one of the first software engineers to earn a professional engineering license in Canada. He believes that software engineering is a branch of traditional engineering.

What about a 'struct' (plain old data)

- We use classes, data/state is private, methods are public
  - Why do we have rules? When can they be broken?
  - Why are there both structs and classes in C++?

- What about helping class, e.g., word and frequency together?
  - We can have one class nested in another, then we don't have to worry so much about encapsulation

- We'll see recitation example for a new class that can be compared using equality and can be sorted
  - Comparable interface must be symmetric with equals
  - What happens if this isn't the case?

Anagrams/Jumbles

- How do humans solve puzzles like that at www.jumble.com
  - Is it important to get computers to solve similar puzzles? Reasons?
  - Should computers mimic humans in puzzle-solving, game playing, etc.? Lessons from chess?

- nelir, nelri, nelir, neril, neri, nler, nlere, nlire, nlrei, nlrie, nler, nleri, nlir, nlire, ...
  - lenir, lenr i, leinr, leirn, ...
  - What's the problem here?

Brute force? SillyAnagrams.java

```java
public String[] allAnagrams(String s) {
    int anaCount = permutationCount(s);
    Set<String> anagrams = new TreeSet<String>();
    ArrayList<String> list = new ArrayList<String>();
    for(int k=0; k < s.length(); k++) {
        list.add(s.substring(k,k+1));
    }
    while (anagrams.size() != anaCount) {
        Collections.shuffle(list);
        anagrams.add(listToString(list));
    }
    return (String[]) anagrams.toArray(new String[0]);
}
```

Quantifying brute force for anagrams

- All anagrams of "compute" takes average of 1 second over 20 trials. How long will "computer" take? Why? "computee"?
  - What is worst case time?
  - What is best case time?

- We're willing to do some pre-processing to make the time to find anagrams quicker
  - Often find that some initialization/up-front time or cost saves in the long run
  - What properties do words share that are anagrams?

Towards a faster anagram finder

- Words that are anagrams have the same letters; use a letter fingerprint or signature/histogram to help find anagrams
  - Count how many times each letter occurs:
    - "teacher" 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
    - "cheater" 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
  - Store words, but use fingerprint for comparison when searching for an anagram
    - How to compare fingerprints using .equals()
    - How to compare fingerprints using .compareTo()

- How do we make client programmers unaware of fingerprints? Should we do this?

Another anagram method

- Instead of fingerprint/histogram idea, use sorted form of word
  - "gable" and "bagel" both yield “abegl”
  - Anagrams share same sorted form

- Similarities/differences to histogram/fingerprint idea?
  - Both use canonical or normal/normalized form
  - Normalized form used for comparison, not for printing
  - When should this normal form be created?

- When is one method preferred over the other?
  - Big words, little words? Different alphabets? DNA vs English?
**OO and Java**

- We'll use an adapter or wrapper class called `Anaword` instead of `String`.
  - Clients can treat `Anaword` objects like strings, but the objects are better suited for finding anagrams than strings.
  - The `Anaword` for "bear" prints as "bear" but compares to other `Anaword` objects as `11001000000000000100000000`.

- In Java change behavior with `.toString()` and `.equals()`.
  - No overloaded operators as in C++.
    - Exception is +, this works for strings, but can't change it.
  - When string needed, automatically call `.toString()`.

**Understandable, extensible?**

- The code does things simply, but isn't very OO. Why is simple sometimes better? Why is it worse?

```java
void printAll(Anaword[] list, Anaword target) {
    System.out.print("anagrams of "+target+": ");
    for(Anaword word : list){
        if (target.equals(word)) {
            System.out.print(word+" ");
        }
    }
    System.out.println();
}
```

**Find all anagrams in dictionary**

- If we sort the dictionary what will happen to the anagrams?
  - capitol optical topical
danger gander garden ranged
  - lameness maleness nameless salesmen

- How can we overload `.equals()`?
  - Look at "danger" or `1001101000001000010...`

**Anaword objects with options**

- Can we use different canonical forms in different contexts?
  - Could have `Anaword`, `FingerprintAnaword`, `SortAnaword`.
  - What possible issues arise? What behavior is different in subclasses?
    - If there's no difference in behavior, don't have subclasses.

- Alternative, make canonical/normalize method a class
  - Turn a function/idea into a class, then let the class vary to encapsulate different methods.
  - Normalization done at construction time or later.
  - Where is normalizer object created? When?
Anagram: Using Normalizers

- How can we normalize an Anaword object differently?
  - Call normalize explicitly on all Anaword objects
  - Have Anaword objects normalize themselves
  - Advantages? Disadvantages?

- If Anaword objects normalize themselves, how can we experiment with different normalization techniques?
  - What about using save-as and several .java files?
  - What about deciding at runtime on normalization?

- We need inheritance! We need interfaces!

Normalizer hierarchy

- Anaword objects normalize themselves
  - Where does the normalizer come from?
    - Passed in at construction time
    - Obtained from normalizer factory
    - Other approaches?

  - How is Normalizer used?

- Normalizer is conceptually an interface
  - Different implementations of the interface have different behavior (guts) but same skin (sort of)