PFTD and toward the week

- Be able to articulate why 201 is the right course for you, in terms of being able to complete it with understanding
- Be able to explain what work is expected, collaboration policies, exams, recitations, assignments, APTs
- Be able to read some Java programs and to analyze them by applying your knowledge of programming to Java programs
- Know what work you should complete before recitation on Monday and before end of next week

Finding Maximally Occurring Word

- Tradeoffs with three different methods
  - Not expected to understand these
  - Goal: all familiar at end of next week
  - git.cs, classwork project

We'll discuss the methods at a high-level today

Finding maximally occurring word

- In all three methods, must count how many times every word occurs
  - Why? If you missed a word, guarantee results?
- methodA: Read each word, search and update a corresponding counter or add new info. This is the parallel arrays method
- methodB: read all, sort, iterate over unique words and count each occurrence - looking up efficiently. The list and set and binary search method
- methodC: as each word read, find word/counter pair efficient and update. The hashing method

Thinking about methodA

- methodA: what happens when "moose" read again, what happens when "bear" read the first time?
  - When there are 1000 words and a word never seen before is read, then ...
  - When all words are different then ...

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>apple</td>
<td>2</td>
</tr>
<tr>
<td>moose</td>
<td>5</td>
</tr>
<tr>
<td>drama</td>
<td>1</td>
</tr>
</tbody>
</table>
Thinking about methodB

- **methodB**: Store unique words and all words, sort all words to search efficiently, loop over unique words
  - If all 1024 words different, then, ...
  - If all 1024 words the same then, ...
- Search for "giant", find one of them, then find range of "giant"

```
"apple" "apple" "drama" "giant" "giant" "giant"
```

Who takes Compsci 201 Now?

Duke Connection: Fred Brooks '53

- **What Would FB Say?**
  "The most important single decision I ever made was to change the IBM 360 series from a 6-bit byte to an 8-bit byte, thereby enabling the use of lowercase letters. That change propagated everywhere."

- "Fred Brooks" by Copyright owned by SD&M (www.sdm.de) - Request for picture sent by email to Fred Brooks by uploader (Mark Polkakas).
- Irresponsible: Send most this photo back, along with contact information for Carolus Lund at SD&M, who gave copyright permission. Licensed under CC BY-SA 3.0 via Wikimedia Commons.

[File: Fred_Brooks.jpg](https://commons.wikimedia.org/wiki/File:Fred_Brooks.jpg)
Why is programming fun?  Fred Brooks

- First is the sheer joy of making things
- Second is the pleasure of making things that are useful
- Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts
- Fourth is the joy of always learning
- Finally, there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff.

Language concepts, Java Concepts

- Variables have names, types, and values
  - Type is integer, double, String
  - Value is 3, 2.718, "hello"
- In Java the type is explicit, part of declaring the variable
  - Errors can be caught at compile time rather than runtime, e.g., with Python
  - More verbose to program, possibly more optimizations possible by compiler
- Source code compiled to low-level object code
  - In Java this is byte code, interpreted by JVM

Source Code, Byte Code

- High level languages compiled to low level languages
  - In C/C++ low-level specific to platform
  - In Java byte code
- Execute by machine: real or virtual
- JVMs must be ported to platform
- Android doesn't use JVM
  - Dalvik and now ARM
- Be grateful!!!!!
  - High level source code

Language Concepts, Java Concepts

- Types have different names and values (and maybe size/memory): int, double, String
- Programmers can create new types: Class in Java
- Arrays (or lists) are indexable, random-access structures. Arrays often homogenous/same-type
  - Aggregate or collection. Traversable. Indexable
- User-defined types are important!
  - Combine data with algorithms that work on data, captured in a class in OO languages
Objects and values

- **Primitive variables label their storage**
  - think memory location with value
  - With int \( x = 5 \); we get storage with 5, label: \( x \)
- **Object variables are labels, reference storage (box)**
  - With String \( s = "genome" \); We create storage (a box) and a variable/label \( s \) that references or points to the box/storage.

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

Objects and values

- **Object variables are labels that reference/point to storage. (labels that are put on boxes)**
  - String \( s = \) new String("genome");
  - String \( t = \) new String("genome");
  - // only one if statement is true!!!
  - 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

www.javarepl.com

- **Type Java, what what happens (no need for class or main or ...)**
  - Let's use int, double, String, and array variables
  - Understand assignment to variables and what a variable references
- **Variables reference storage: primitives aren't pointers/references, Objects are**
  - Understand the distinction
  - Be able to explain the difference
- **Understand differences: == and .equals()**

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 can be applied to null
  - Can assign label to existing box (via another label)
  - Can create new box using built-in new
- **Object types are references/pointers/labels to storage**
Arrays: random access/one type

- Data stored in memory, one object per "slot"
  - Contiguous in memory, why?
  - Addressable by index
  - Zero is first index

- In Java: homogeneous
  - All int, or String or ...
  - Python lists are different

- Array v. ArrayList
  - primitive v object
  - int v Integer

Arrays and ArrayLists

- An array is an abstraction, supports operations
  - Initialization
  - Assignment
  - Iterate/Traverse

- Random access (constant time) indexed element
  - Accessing 1<sup>st</sup> entry same time as 100<sup>th</sup> as 1000<sup>th</sup> ...

- Syntax: see example programs
  - Set size, call new, use brackets
  - Size is fixed, accessed with .length
  - Can't increase in size once allocated

Why ArrayLists?

- An array is fast, stores any kind of element
  - Primitive: int, double, char
  - Object: String, Planet, URL ...

- ArrayList supports more operations
  - Iterate, remove, add, grow, ...
  - Fits into a well-designed library of structures
  - CANNOT store primitives directly

- Look at examples, can add int to ArrayList, it's converted, or "autoboxed" to Integer
  - Typically don't even think about this

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."
Anatomy of a class

```java
public class Foo {
    private int mySize;
    private String myName;
    public Foo(){
        // what's needed?
    }
    public int getSize(){
        return mySize;
    }
    public double getArea(){
        double x;
        x = Math.sqrt(mySize);
        return x;
    }
}
```

- What values for vars (variables) and ivars (instance variables)?

Some Java Vocabulary and Concepts

- Java is a (statically) typed language
  - Every variable/object has a type that's defined at compile time (compare Python)
  - More typing, but more errors caught before runtime
- Java is an object-oriented language
  - All code is in a class and all code inside function/method
  - Classes are also types, so variables/objects extendible
  - We will see interfaces and inheritance later
  - Classes have instance variables, methods local variables
- Java has different 'types' primitive and Object/class.
  This is for efficiency, but it's sometimes confusing

Solving problems, writing code

- APT Problems
  - Understand the problem, know how to solve an instance
  - Ideas? Caveats?
- Writing code to implement proposed solution
  - Will it run? In time? Constraints? Look before you code
  - How will we test the solution? When to start testing?
- What's the green dance and when do we do it?
  - Satisfaction of finishing something
  - Knowing when to stop when you're not making progress
  - Leveraging community wisdom

Let's play Jotto (for Recitation)

- Word game similar to Mastermind
- Not too hard to have computer play well in guessing word: keep a universe of words that could be secret word
  - After every guess, reduce the size of the universe
- First WOTO program, recitation and later
  - From one class to object-oriented design
Review some Java

- Use your group and understand as the javarepl
  - Don't write code, use your understanding and discussion with your group


What's coming next week

- Recitation. Read the code and the questions for recitation, come prepared. Install Eclipse and the plugins, Use Piazza as needed

- Test at least one APT problem, ideally completing one before recitation, we'll discuss CirclesCountry in recitation

- Read the NBody assignment, you don't need to start the assignment, but read carefully
  - We'll create resources to explain downloading starter code for assignment

Goals for the Course

- Given a problem statement & a real data source, design, develop, debug, and test a Java program that uses appropriate standard libraries to efficiently solve the problem.

- Write programs that effectively implement and use data structures such as: arrays, maps, linked lists, stacks, queues, trees, and graphs.

- Evaluate the time and space complexity of algorithms, especially algorithms that scale, using empirical or mathematical analysis.

- Apply basic object-oriented design and programming principles in developing software