I will throw cookies this week.
Announcements

• Assignment 1 – RSG is due Thursday Sept 9
• APTs (see APT page) are due Tues, Sept 14
  – Recitation on Sept 9-10 on APTs CirclesCountry and SoccerLeagues

Today
• Finish lecture notes from Day 1 finally!
• Discussion of Java maps, with code examples
  – Kinds of maps, what goes in maps, why arrays are maplike
  – Reference to current assignment as well as other code
• Try again - Demo of APT, plan, execute, attack, dance
  – When is enough planning too much? Coding at 2:00 am?
How and why do we organize data? Differences between data and information? What about knowledge?
Organizing Data: ideas and issues

• Often there is a *time/space* tradeoff
  – If we use more space (memory) we can solve a data/information problem in less time: *time efficient*
  – If we use more more time, we can solve a data/information problem with less space: *space efficient*

• Search v Store: repeating the same thing again ...
  – We’re not “smart” enough to avoid the repetition
    • Learn new data structures or algorithms!
  – The problem is small enough or done infrequently enough that being efficient doesn’t matter
  – Markov illustrates this (next assignment)
Map: store pairs of (key, value)

- Search engine: web pages for “clandestine”
  - Key: word or phrase, value: list of web pages
  - This is a map: search query->web pages

- DNS: domain name duke.edu → IP: 152.3.25.24
  - Key: domain name, value: IP address
  - This is a map: domain name->IP address

- Map (aka table, hash) associates keys with values
  - Insert (key,value) into map, iterate over keys or pairs
  - Retrieve value associated with a key, remove pair
What does RSG map code look like?

private Map<String,ArrayList<String[]>> myMap;
private Random myRandom;

public void generateFrom(String nont){
    ArrayList<String[]> rules = myMap.get(nont);
    String[] list =
        rules.get(myRandom.nextInt(rules.size()));
    for(String s : list){
        if (s.startsWith("<")) { }
        else { }
    }
}

– Where are ivars (instance variable) initialized? To what (and why)?
– What code goes with if? With else? Why?
– Reasons to use local vars in second line: debugging!
Notes on RSGII

• Interface
  – A group of related methods with empty bodies
  – To implement the interface, your class would implement the methods for those named in the interface.
  – Example

```java
public interface IModel {
    public void initialize(Scanner sc);
    public void generate();
}
```
Notes on RSGII (cont)

• Example – don’t use the name SomeClass! Pick another name

```java
public class SomeClass implements IMModel {
    // Start with a copy of RSGSimpleParse

    // update constructor to have an RSGViewer param

    // already has initialize method
    public void initialize(Scanner sc){  // ...

    // must implement generate method
    public void generate(){              // ...

    // other methods including a main
}
```
Notes on RSGII (generate method)

```java
class NotesOnRSGII {
    public void generate() {
        if (nameOfRSGviewer != null) {
            nameOfRSGviewer.clear();
        }
        // call to start deriving
        // random sentence
    }
}
```
Notes on RSGII (putting together)

• Doesn’t look like much happens
• Create object of SomeClass and set to be the model

```java
public class RSGMain {
    public static void main(String[] args){
        RSGViewer viewer = new RSGViewer("CompSci 100, RSG Fall 2010");
        IModel model = new SomeClass(viewer);
        viewer.setModel(model);
    }
}
```
What happens in RSGViewer

• Don’t need to modify this file
• Methods
  – clear() – clears the screen
  – setModel(IModel model)
    { myModel = model; }
  – How do things happen? Events, listeners
  – In makeInput() method (associated with Create button)
    • ActionPerformed is myModel.generate();
Example: Frequencies.java

- This map: key is a string, value is # occurrences
  - See code handout and code below

- What clues for prototype of map.get and map.put?
  - What if a key is not in map, what value returned?
  - What kind of objects can be put in a map?
  - Kinds of maps?

```java
for(String s : words) {
    s = s.toLowerCase();
    Integer count = map.get(s);
    if (count == null){
        map.put(s,1);
    }
    else{
        map.put(s,count+1);
    }
}
```
Maps, another point of view

• An array is a map, consider array \texttt{arr}
  – The key is an index, say \texttt{i}, the value is \texttt{arr[i]}
  – Values stored sequentially/consecutively, not so good if the keys/indexes are 1, 100, and 1000, great if 0,1,2,3,4,5

• Time/space trade-offs in map implementations, we’ll see more of this later
  – TreeMap: most operations take time $\log(N)$ for N-elements
  – HashMap: most operations are constant time on average
    • Time for insert, get, ... doesn’t depend on N (wow!)
  – But! Elements in TreeMap are in order and TreeMap uses less memory than HashMap
Map (foreshadowing or preview)

• Any kind of Object can be inserted as a key in a HashMap
  – But, performance might be terrible if hashValue isn’t calculated well
  – Every object has a different number associated with it, we don’t want every object to be associated with 37, we want things spread out

• Only Comparable object can be key in TreeMap
  – Basically compare for less than, equal, or greater
  – Some objects are naturally comparable: String, Integer
  – Sometimes we want to change how objects are compared
  – Sometimes we want to invent Comparable things
“Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.”

“There's no sense in being precise when you don't even know what you're talking about. “

“There are two kinds of people in the world: Johnny von Neumann and the rest of us.”

Eugene Wigner, Noble Physicist
Coding Interlude: FrequenciesSorted

• Nested classes in FrequenciesSorted
  – WordPair: combine word and count together, why?
  – WPFreq: allows WordPair objects to be compared by freq

• How are WordPair objects created?
  – In doFreqsA is the comparable-ness leveraged?
  – What about in sorting?

• Alternative in doFreqsB
  – Use TreeMap, then ArrayList then sort, why?
  – Is comparable-ness leveraged? Sorting?
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/pointers/labels to storage
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)?
"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."

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE’S COMPILING."

Hey! Get back to work!

Compiling!

Oh. Carry on.
David Parnas (entry in Wikipedia)

• *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:* He believes that software engineering is a branch of traditional engineering.
Tomato and Tomato, how to code

• `java.util.Collection` and `java.util.Collections`
  – one is an interface
    • `add()`, `addAll()`, `remove()`, `removeAll()`, `clear()`
    • `toArray()`, `size()`, `iterator()`
  – one is a collection of static methods
    • `sort()`, `shuffle()`, `reverse()`, `max()`
    • `frequency()`, `indexOfSubList()`

• `java.util.Arrays`
  – Also a collection of static methods
    • `sort()`, `fill()`, `binarySearch()`, `asList()`
Preview of APTs and Assignments

• Rules in writing code to solve APTs
  – Have a plan for how to solve the problem
  – Demonstrate plan by using algorithm on one example
  – Think about how to implement plan
    • Instance variables? Helper methods?
    • Write and rewrite. Don’t worry about lengthy methods.
    • Do worry about length methods, rewrite!

• From RSG to Markov
  – Maps, but Markov has an analytical component
  – Three braids of grading: engineering, algorithmic, analysis
  – Some assignments have extra braid: neural wonder!