

Data and Information

Google Announces Plan To Destroy All Information It Can't Index

AUGUST 31, 2005 | ISSUE 41-35

MOUNTAIN VIEW, CA—Executives at Google, the rapidly growing online-search company that promises to "organize the world's information," announced Monday the latest step in their expansion effort: a far-reaching plan to destroy all the information it is unable to index.



"Our users want the world to be as simple, clean, and accessible as the Google home page itself," said Google CEO Eric Schmidt at a press conference held in their corporate

How and why do we organize data? Differences between data and information? What about knowledge?

RELATED ARTICLES

Web-Browser History A Chronicle Of Couple's Unconcerned Dismissal



Compsci 100e

5.1

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)

Compsci 100e

5.2

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

Compsci 100e

5.3

Maps, another point of view

- An array is a map, consider array `arr`
 - The key is an index, say `i`, the value is `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

Compsci 100e

5.4

Map (foreshadowing or preview)

- Any kind of Object can be inserted as a key in a HashMap
 - But, performance might be terrible if hashCode 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

Traceroute: where's the map here?

```
traceroute www.cs.dartmouth.edu
traceroute to katahdin.cs.dartmouth.edu (129.170.213.101), 64 hops max,
 1 lou (152.3.136.61)  2.566 ms
 2 152.3.219.69 (152.3.219.69)  0.258 ms
 3 tellsp-roti.netcom.duke.edu (152.3.219.54)  0.336 ms
 4 rlgh7600-gw-to-duke7600-gw.ncren.net (128.109.70.17)  184.752 ms
 5 rlgh1-gw-to-rlgh7600-gw.ncren.net (128.109.70.37)  1.379 ms
 6 rtp11-gw-to-rpop-oc48.ncren.net (128.109.52.1)  1.840 ms
 7 rtp7600-gw-to-rtp11-gw-sec.ncren.net (128.109.70.122)  1.647 ms
 8 dep7600-gw2-to-rtp7600-gw.ncren.net (128.109.70.138)  2.273 ms
 9 internet2-to-dep7600-gw2.ncren.net (198.86.17.66)  10.494 ms
10 ge-0-1-0.10.nycmng.abilene.ucaid.edu (64.57.28.7)  24.058 ms
11 so-0-0-0.0.rtr.newy.net.internet2.edu (64.57.28.10)  45.609 ms
12 nox300gw1-v1-110-nox-internet2.nox.org (192.5.89.221)  33.839 ms
13 ...
14 ...
15 border.rop ferry1-crt.dartmouth.edu (129.170.2.193)  50.991 ms
16 katahdin.cs.dartmouth.edu (129.170.213.101)  50.480 ms
```

John von Neumann

"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



Interface at work: Frequencies.java

- From recitation: key is a string, value is # occurrences
 - Code below is slightly modified version of recitation code
- 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?

```
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);
    }
}
```

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)?

- <http://www.cs.duke.edu/csed/java/jdk1.6/api/index.html>
 - 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`
 - 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

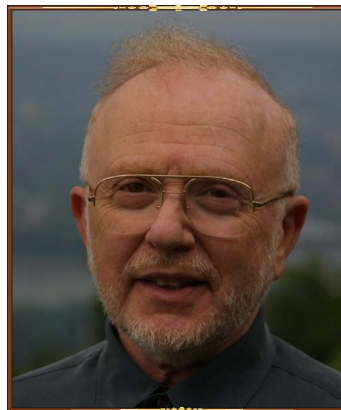
Anatomy of a class

```
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)?

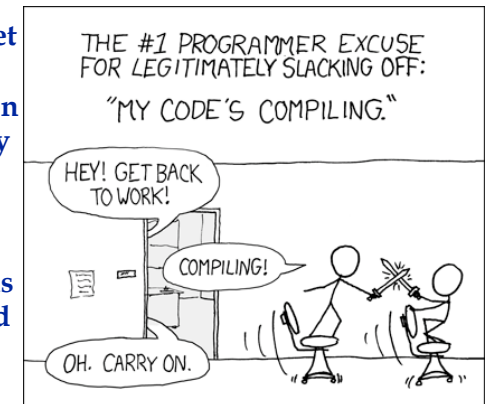
David Parnas

"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."



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()`