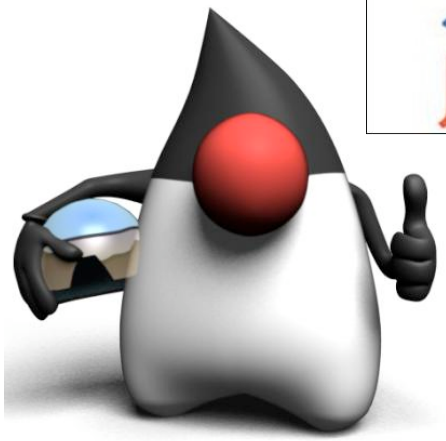
Java as a Second Language



Week 7: Text

CS3773
Simson Garfinkel

This week:

- Text
- Unicode
- Lucene

Week 8 (Feb 25) - Threading
Week 9 (March 3) - Networking
Week 10 (March 10) -
Style & Performance
Week 11 (March 17) -
Special Projects
March 27* - Final Project Due

You have learned a *lot* of Java.

Navigating the Documentation; NetBeans; JAR files

47 Keywords

Classes

- creating; fields; methods; static vs. instance
- Abstract Classes & Interfaces
- Enumes
- Containers & Generics (e.g. HashSet<String>)

I/O: Reading & Writing Files

AWT & Swing

You are responsible for these bold, underlined Java Reserved Words.

abstract	**do**	**import**	**short**	volatile
assert	**double**	**instanceof**	**static**	**while**
boolean	**else**	**int**	**strictfp**	
break	**enum**	**interface**	**super**	
byte	**extends**	**long**	**switch**	
case	**final**	**native**	synchronized	
catch	**finally**	**new**	**this**	
char	**float**	**package**	**throw**	
class	**for**	**private**	**throws**	
[const]	**[goto]**	**protected**	transient	
continue	**if**	**public**	**try**	
default	**implements**	**return**	**void**	

http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.html

Start thinking about your final projects

Your final project is:

- A chance to explore something about Java
- A chance to do something that works
- A chance to fix something you wanted to get right, but didn't have a chance.

Requirements for final project:

- You must do your own work (no team projects)
- You must write *something that works.*
- You must demonstrate at least 5 of the Learning Outcomes.

Due Friday: A 1-paragraph final project proposal

Quiz #6 Diagnostic:
Most students seem unsure about details.

Most students had problems with:

- equals()
- hashCode()
- Exceptions

Most students understood:

- enum
- JButton() / add() / setVisible()
- final classes cannot be subclassed
- JPanel() is a "canvas where you can draw" (and not JFrame)

equals()

Signature:

```
public boolean equals(Object obj){
}
```

The equals() contract:

Compares the value of two objects for equality.

Syntax: a.equals(b) must be true or false.

- Note one object "equals" to another object.
- equals() takes a *single* argument
- argument may be *any object! or null!*

equals()

Signature:

```
public boolean equals(Object obj){
}
```

Why this signature?

.equals() needs to take Object so that anything can be compared equals() with anything else.

```
Object a;
Integer i;
a.equals(i) must not throw an exception!!!
```

equals(obj) — obj may be null!

.equals() contract: .equals(null) must return false.

Nothing is equals() to null!

```
Location:

public boolean equals(Object obj){
    if(obj==null) return false;
    …
}
```

equals(obj) — obj must be the correct class!

```
Location:

public boolean equals(Object obj){
    if(obj==null) return false;
    if(this.getClass() != obj.getClass()) return false;
    …
}
```

equals(obj) —
objects are equals() if their values are equal!

```
Location:

public boolean equals(Object obj){
    if(obj==null) return false;
    if(this.getClass() != obj.getClass()) return false;

    Location l = (Location)obj;
    return this.x==l.x && this.y==l.y;
}
```

equals(obj) —
What's wrong with this code?

```
Location:

public boolean equals(Object obj){
    if(obj==null) return false;
    if(this.getClass() != obj.getClass()) return false;

    Location l = (Location)obj;
    return this.x==l.x || this.y==l.y;
}
```

equals(obj) —
What's wrong with this code?

```
Location:

public boolean equals(Location lobj){
    if(lobj==null) return false;
    return this.x==lobj.x || this.y==lobj.y;
}
```

equals(obj) — What's wrong with this code?

```
Location:

public boolean equals(Location l1,Location l2){
    if(l1==null) return false;
    if(l2==null) return false;
    if(l1.getClass() != l1.getClass()) return false;

    return l1.x==l2.x && l1.y==l2.y;
}
```

Will this compile?

What will happen when you use it?

equals(obj):
Question: Will this print "true" or "false"

```
public class x {
    public static void main(String[] args){
    Object a = new Object();
    Object b = new Object();
    System.out.println("equals: "+a.equals(b));
    }
};
```

equals

`public boolean` **`equals`**`(Object obj)`

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

Parameters:

`obj` - the reference object with which to compare.

Returns:

`true` if this object is the same as the obj argument; `false` otherwise.

See Also:

`hashCode()`, `Hashtable`

Answer: False

"The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` **if and only if `x` and `y` refer to the same object** (`x == y` has the value `true`)."

```
10:28 PM imac2:~$ javac x.java
10:29 PM imac2:~$ java x
equals: false
10:29 PM imac2:~$
```

equals(obj) — Building Wall equals() with Location equals

```
public class Wall extends FlatlandObject {
    Location p1, p2;

    public boolean equals(Object obj){
    if(obj==null) return false;
    if(this.getClass() != obj.getClass()) return false;
    Wall w = (Wall)obj;

    return(this.p1.equals(w.p1) && this.p2.equals(w.p2));
}
```

What about color, heading, line width, etc?

HashCode

hashCode

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by `java.util.Hashtable`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

HashCode: Key Ideas

If a.HashCode() != b.hashCode(), then the two objects ARE NOT EQUAL

if a.HashCode()==b.hashCode() then the two objects MAY BE EQUAL

a.HashCode() DOES NOT CHANGE while the program is running.

What's wrong with this code?

Location:

```
public int hashCode(Object o){
    if(o==null) return 0;
    if(o.hashCode() != this.hashCode()) return -1;
    return o.x + o.y;
}
```

What's wrong with this code?

Location:

```
static int objectCounter=0;
int myObjectNumber = -1;
public int hashCode(){
    if(myObjectNumber==-1){
        myObjectNumber = objectCounter++;
    }
    return myObjectNumber;
}
```

Here was problem #9 from the Quiz; What three compilation errors will the compiler catch?

```
public class x {
    public static void main(String[] args){
    try {
        System.out.println("one");
        throw new Exception();
        System.out.println("two");
    } catch (Exception e){
        System.out.println("three");
    } finally {
        System.out.println("four");
        throw new Exception();
        System.out.println("five");
    }
    }
};
```

Here was problem #9 from the Quiz; What three compilation errors will the compiler catch?

```
public class x {
    public static void main(String[] args){
    try {
        System.out.println("one");
        throw new Exception();
        System.out.println("two");
    } catch (Exception e){
        System.out.println("three");
    } finally {
        System.out.println("four");
        throw new Exception();
        System.out.println("five");
    }
    }
};
```

unreachable statement

Unreported Exception

unreachable statement

Text

Text Vocabulary: Glyph
A "glyph" is an abstract shape of a character

"a" and "b" are different glyphs.

"a," "**a,**" and "a" are the same glyph.

some glyphs:

a b c d

Text Vocabulary: typeface

A typeface is a style for displaying glyphs

Helvetica Neue

Lucida Fax

Monaco

Playbill

SimSun

Comic Sans MS

Calibri

Text Vocabulary: Weight
Modifications to a font

Regular

Light

UltraLight

Bold

Condensed Bold

Condensed Black

Thin

Wide

Text Vocabulary: Style
How the font is rendered

Regular

Italic

Getting and drawing fonts.

J2SE supports TrueType and PostScript Type 1 fonts. Built in fonts:

```
Lucida Sans
Lucida Sans Typewriter
Lucidia Bright
```

Other fonts (may) cost money.

java.awt.font - Creates a font

```
Font ft = new Font("Lucida Sans",Font.PLAIN,24);
g.setFont(f);
g.drawString("This is a test",x,y);
```

How big will the font be?

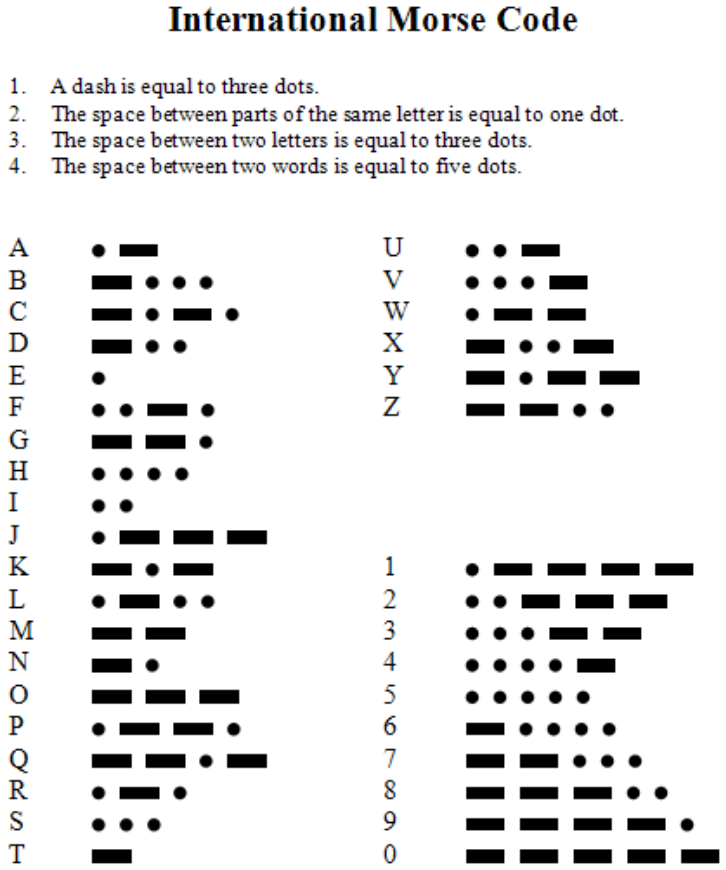
```
FontMetrics fm = g.getFontMetrics();
int width = fm.stringWidth("This is a test");
```

Codes

Morse Code

1836 - Morse & Vail invent electric telegraph

"Code" provides a 1-to-1 mapping between the *symbols* that are sent over the wire and the glyphs that we read.



http://en.wikipedia.org/wiki/Image:International_Morse_Code.PNG

Teleprinters used 5-bit Baudot Code

5 bits = 00 .. 1F = 32 characters

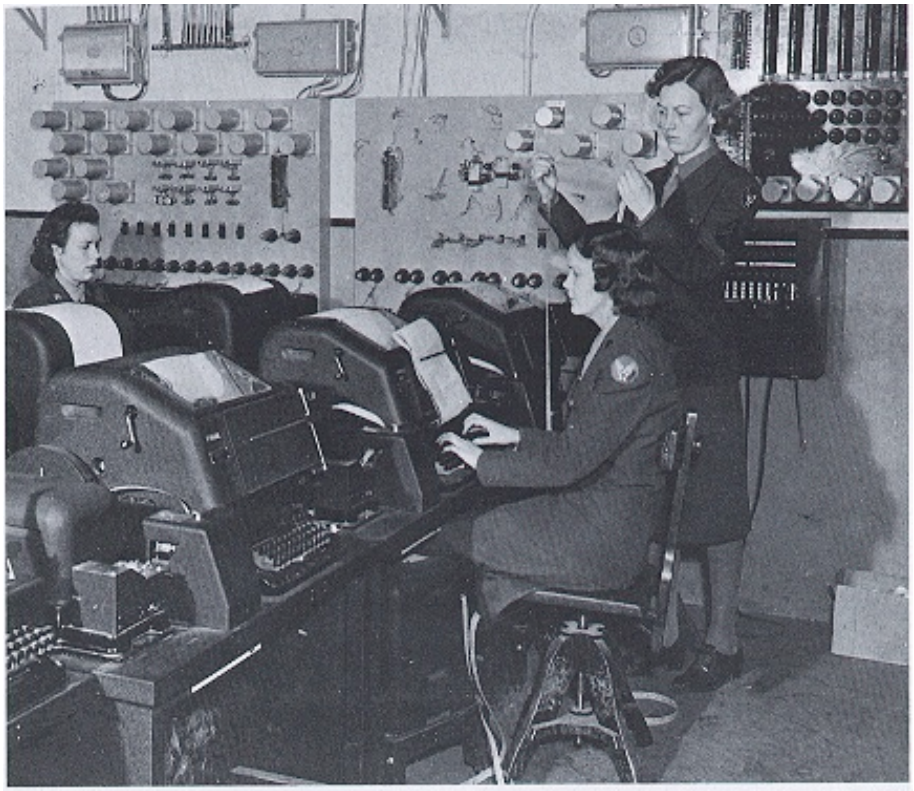
Special "Shift" codes:

1B = Figures follow

1F = Letters follow

Uppercase Only.

00	01	02	03	04	05	06	07
NUL	E 3	LF	A -	SP	S '	I 8	U 7
08	09	0A	0B	0C	0D	0E	0F
CR	D ENQ	R 4	J BEL	N ,	F !	C :	K (
10	11	12	13	14	15	16	17
T 5	Z +	L)	W 2	H £	Y 6	P 0	Q 1
18	19	1A	1B	1C	1D	1E	1F
O 9	B ?	G &	FIGS	M .	X /	V ;	LTRS
Letters			Figures			Control Chars.	



WACs assigned to the Eighth Air Force in England operate teletype machines. (DOD photograph)

ASCII
American Standard Code for Information Exchange

Big Innovations: 7 bits; Upper case, lower case, & symbols

Problems: US only (no accented characters)

Code space: 0 .. 127

- 0 = NULL
- 1 .. 31 = Control Characters
- 32 = Space
- 48..57 = Characters "0" through "9" (in order)
- 65..90 = Characters "A" through "Z"
- 97..122 = Characters "a" through "z"
- 123 = "{" ; 124 = "|" ; 125 = "}" 126 = "~"; 127 = DEL

http://en.wikipedia.org/wiki/ASCII

"Code Pages"

Developed by IBM to represent "foreign" (ie: non-English) characters

Usually maps:

0..127 as ASCII

128..255 as "special characters"

Code Pages:

437 - IBM PC Code Page http://en.wikipedia.org/wiki/Code_page_437

737 - Greek http://en.wikipedia.org/wiki/Code_page_737

869 - Greek2 http://en.wikipedia.org/wiki/Code_page_869

Don't use code pages!

Unicode: 1991—

100,000+ characters to represent every *glyph* in use

- Originally just modern glyphs
- Now extended to ancient glyphs, music, etc.

Every Unicode glyph has a specific integer.

- "A" is still 65
- U+0041
- \u0041
- &#0041

"Normal" Unicode characters are 16 bits; shift characters for codes > 65,536

Unicode: Complexity Rules

Unicode has a lot of confusing elements:

- The same printing glyph can be represented with many different code points
- Accented characters can be represent by a composed character and accent (è) or by a separate character (e) and accent (̀)
- Sorting rules are language-specific (not really a Unicode problem)

Unicode Codings

"Codings" are how a unicode character is encoded into a byte stream.

UTF-16:

- Codes all 16-bit Unicode as two bytes
- Two versions: big-endian and little-endian (0041) and (4100)
- Special "escapes" for coding 32-bit characters
- Extensively used by Microsoft; not much else

UTF-32:

- Codes all characters as 4-bytes
- Two versions: big-endian and little-endian (00000041) and (41000000)
- Rarely used anywhere

UTF-8: Most popular Unicode encoding

Codes ASCII as ASCII

2 bytes for most European & Arabic Characters

3 bytes for most other characters in the Basic Multilingual Plane

4 bytes for everything else

Advantages of UTF-8:

- ASCII is ASCII (UTF-8 is an ASCII superset)
- Bytes in position 1, 2, 3 & 4 are all distinct (allows resynchronization)
- NULLs do not appear in the text (compatible with C/C++)

Disadvantage of UTF-8:

- Hard to figure out how long a string is without parsing
- Encoding rules are complex

Unicode URLs of Interest

Glossary: http://unicode.org/glossary/

What's new in Unicode 5.0.0: http://www.unicode.org/versions/Unicode5.0.0/

Unicode Technical Reports: http://www.unicode.org/reports/

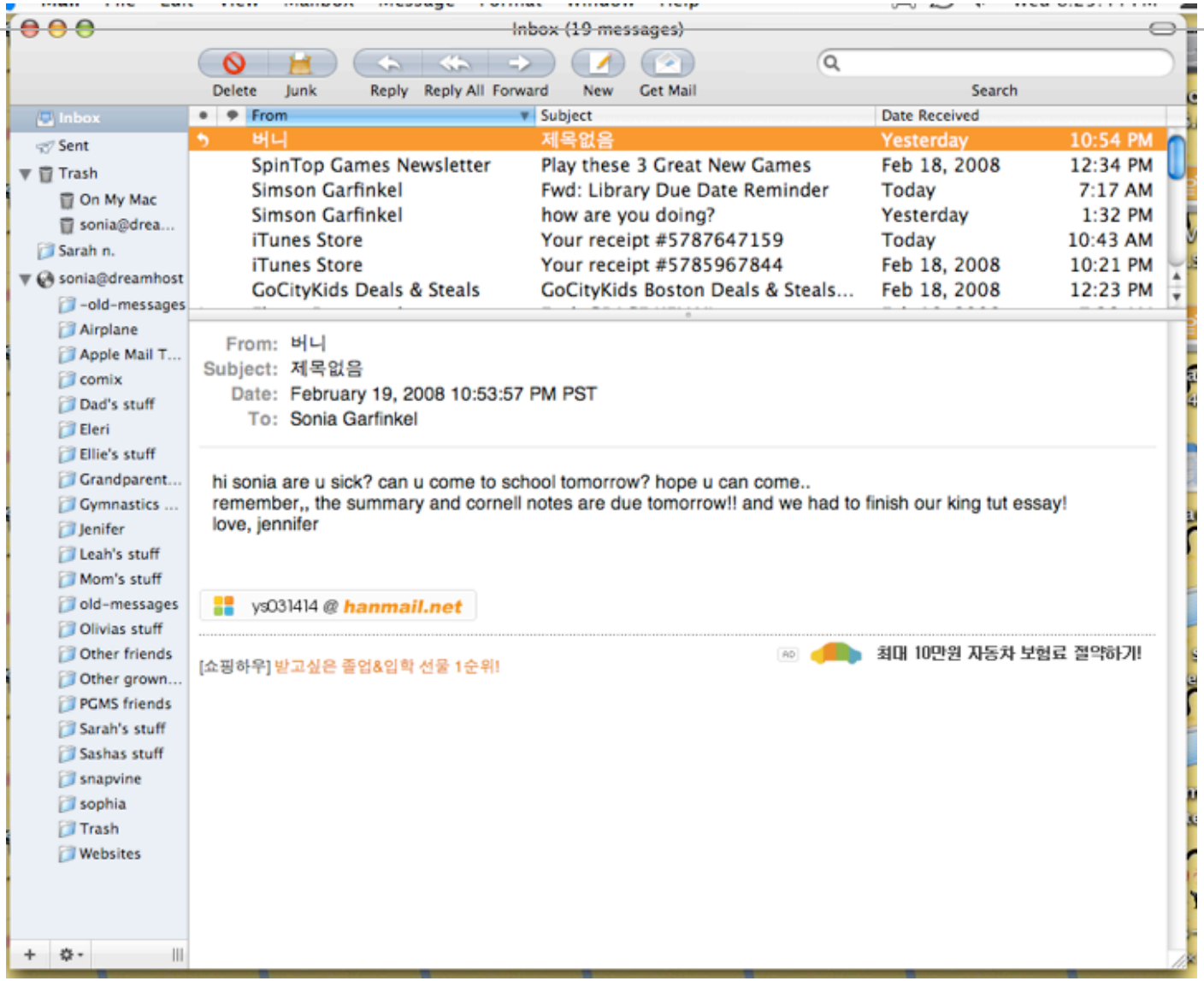
Microsoft Ask Dr. International:

- http://www.microsoft.com/globaldev/DrIntl/default.mspx
- http://www.microsoft.com/globaldev/DrIntl/columns/default.mspx

Thursday

Unicode & Lucene

Korean in Apple Mail



Lucene

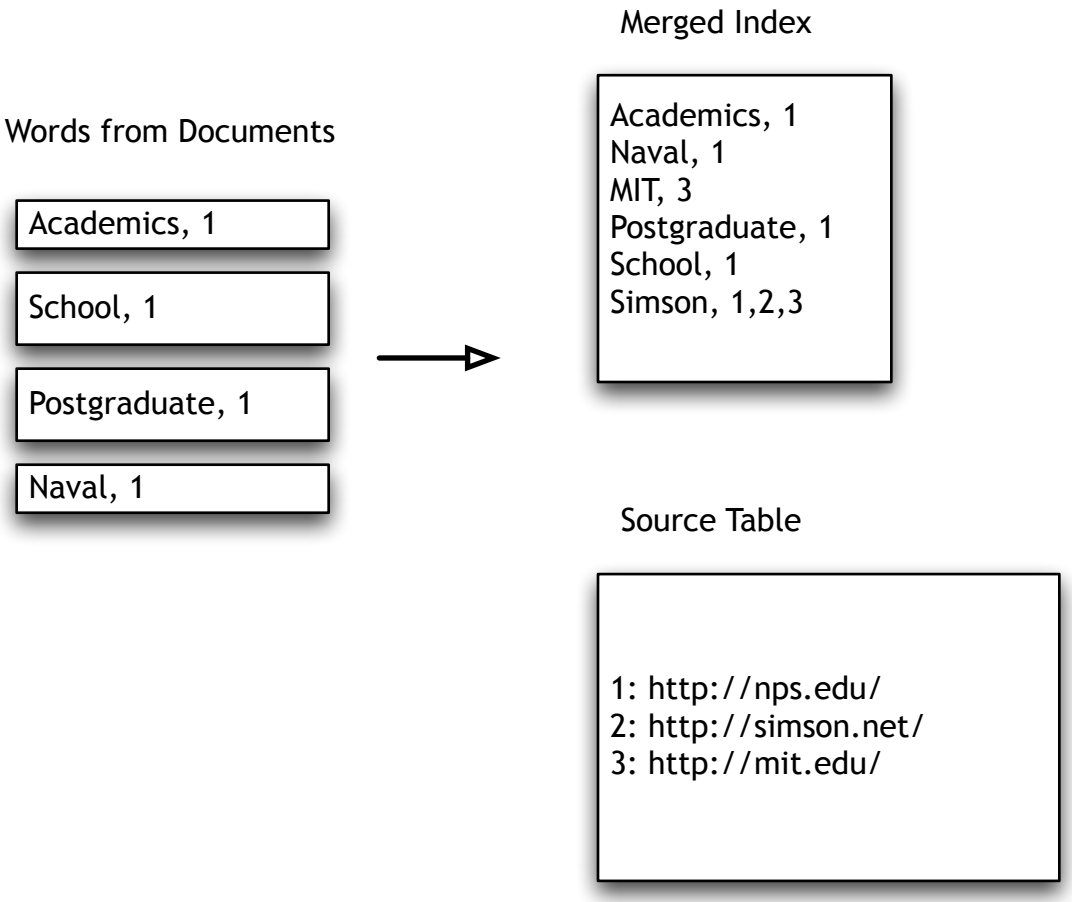
Lucene is the Apache Foundation's search engine (in Java).

- Widely used on major websites

Two parts:

- Index maker - reads files and adds them to the index
- Index searcher - takes a search term and says where it is found

This is all done with sorted tables.



Download and run Lucene!

http://lucene.apache.org/java/docs/

Demo:

```
java -classpath lucene-core-2.3.0.jar:lucene-
demos-2.3.0.jar org.apache.lucene.demo.IndexFiles
c:\slgarfin

adding /Users/simsong/current/vacation/Summer Vacation/
UAL-Sonia.pdf
adding /Users/simsong/current/visualization-thoughts.txt
adding /Users/simsong/current/whole_earth_journal_ssns.txt
Optimizing…
load: 1.45  cmd: java 56155 waiting 47.56u 5.93s
182216 total milliseconds
```

The indexer creates an "index/" directory:

```
$ ls -l index/
total 63064
-rw-r--r--  1 simsong   64568842 Feb 20 21:57 _m.cfs
-rw-r--r--  1 simsong         20 Feb 20 21:57 segments.gen
-rw-r--r--  1 simsong         45 Feb 20 21:57 segments_1b
$
```

You search with the "org.apache.lucene.demo.SearchFiles" class:

```
$ java -classpath lucene-core-2.3.0.jar:lucene-demos-2.3.0.jar
org.apache.lucene.demo.SearchFiles
Enter query:
Sonia
Searching for: sonia
2 total matching documents
1. /Users/simsong/current/vacation/Summer Vacation/iten.rtf
2. /Users/simsong/current/vacation/flights.rtf
Enter query:
```

indexDocs indexes all of the files starting at a directory using recursive descent…

```
static void indexDocs(IndexWriter writer, File file)throws IOException {
   // do not try to index files that cannot be read
   if (file.canRead()) {
     if (file.isDirectory()) {
       String[] files = file.list();
       if (files != null) {
         for (int i = 0; i < files.length; i++) {
           indexDocs(writer, new File(file, files[i]));
         }
       }
     } else {
       System.out.println("adding " + file);
       try {
         writer.addDocument(FileDocument.Document(file));
       }
       catch (FileNotFoundException fnfe) {
       }
     }
   }
 }
```

SearchFiles searches the index. It's not hard!

```
IndexReader reader = IndexReader.open(index);
Searcher searcher = new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
QueryParser parser = new QueryParser("contents", analyzer);
Query query = parser.parse("some search term");
Hits hits = searcher.search(query);

for(int i=0;start<hits.length();i++){
   Document doc = hits.doc(i);
   System.out.println("path: "+doc.get("path"));
}
```

You can improve the search with a bit of linguistics

Word Stemming:

- "Academics" is in the document
- "Academic" and "Academics" gets put into the index.

Synonyms:

- "Navy" is in the document
- "Navy", "dark blue", "United States Navy", "USN" put into the index.

- See: http://wordnet.princeton.edu/perl/webwn?s=navy

To use Lucene from NetBeans

1. Download Lucene
2. Unzip the zip file
3. Install the jar files in a known location
   - I used ~/lucene-2.3.0
4. Tell NetBeans about the location (right-click on Libraries)