CS 251 - Lab 004

TA: Kage Weiss
Office Hours: R 2-2:50 FEC 2000, or by appointment.
Contact: mmweiss@unm.edu
Website: **http://cs.unm.edu/~kageweiss/TA/cs251.html -- SLIDES POSTED**

- Sign in sheet located on desk by TA
- Today we are working on Markov (Lab 7)

How I Grade Your Code:

- Chenoweth provides the rubric and occasionally tester code.
- I write tester code that thoroughly tests your code.
- If your code runs as expected with my tester, I'm happy.
- I then review your code (.java) to make sure it follows CS251 Code Standards.
  - That means variable and method names, comments, privacy, **NO TABS**, etc.
- I **can** tell how much work you put into your code.
- I can tell who didn't cite StackOverflow.
- If you obviously put work in, I'm happy to give points and comments!
- If you obviously didn't put work in, I have to really look at your code and end up finding (and counting off for) minor infractions.
- Every point I take off is listed in the comments for the grade. **Read them**.

Markov

- WORK ON UNDERSTADING FILE IO, if you can't import the files properly, it will never work.
- Read her Markov slides
- Read her entire lab pdf before starting
- Markov is one of the most difficult labs understand properly
- It is all based on your understanding of collections (map)
- I would definitely recommend spending at least 10 minutes sketching on a piece of paper how a Markov generator works

Markov/Trie Algorithm

- The Markov/Trie Algorithm is a mapping algorithm utilizing a migratory relationship between keys and a collection of their data. The key is almost always of equal or greater order than their data.
    - For example: String -> Char, Library -> Book, String -> String, and Collection<String> -> String, would all be useful sets.
- The starting key in a Trie is generally an anchor that all branches will start from, and is mapped to all the top level keys. In a Markov, the order determines how many instances of the data level object constitute a key, in a Trie, that ordering can be set or dynamic depending on the implementation.
- M/T generally serves one of two purposes, either as a qualifying data set or as a disqualifying dataset. The predictive text on your phone is an example of a qualifier, where the M/T is providing what it thinks could come next. A searching algorithm may use a M/T to deterministically eliminate branches of trees that do not benefit the program.

Markov/Trie Algorithm

The Trie we will be using in an effective WordSearch is a disqualifier, eliminating branches of our input dictionary so we can search through both the input and the puzzle only once each.

In theory a Trie looks something like this:

- Note the anchor, and the fact that the following data is an *extension* of its key
- This is a Trie<String, Char>
- The word list here is:
  to, tea, ted, ten, inn, A, I, in, inn

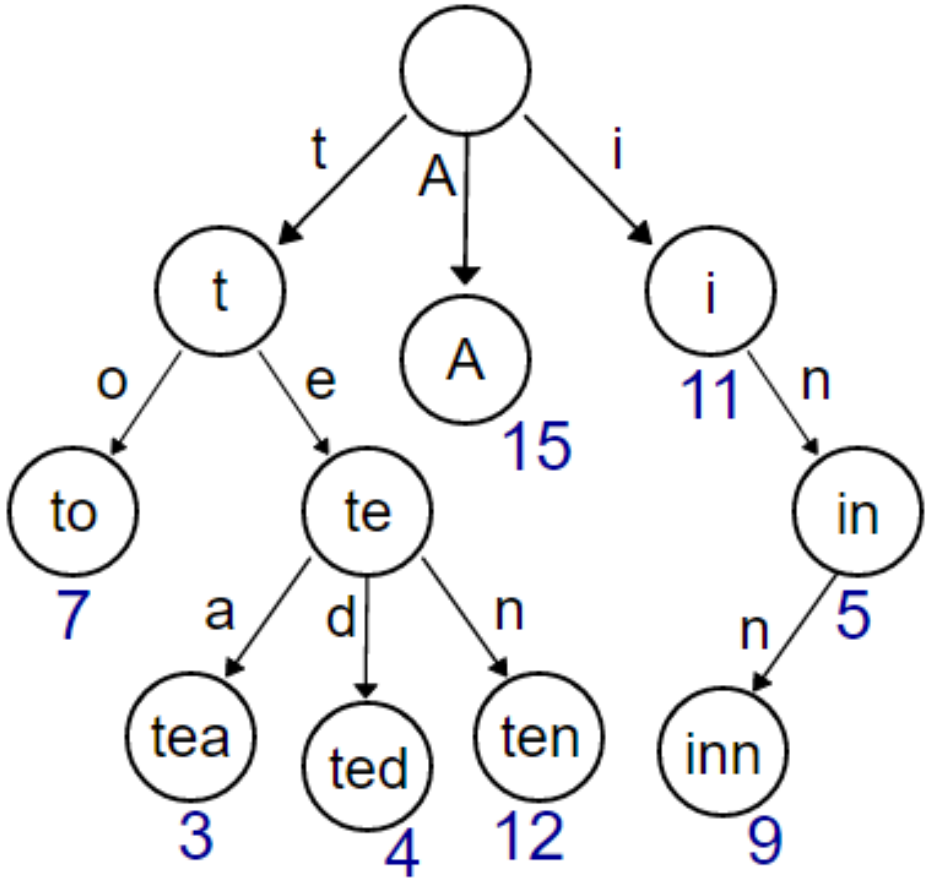


Image Source: Wikipedia: Trie

Markov/Trie Algorithm

In practice, a Trie (String -> Char) will look more like this:

null	A	B	At	An	Ba	Be
• A • B	• t • n • “”	• a • e	• “”	• t • d • “”	• a • d	• e • “”

Ant	And	Baa	Bad	Bee
• “”	• “”	• “”	• “”	• “”

Word List: A, At, An, And, Ant, Baa, Bad, Be, Bee

Notice how it’s technically the tree, but because Java maps don’t internally branch, we fake it by creating successive keys. The end of each branch is also denoted, in this case by our empty string. This means wherever we have “” as one of our possible branches, the key itself is a completed dataset.

Markov/Trie Algorithm

In practice, a Markov<String, String>, order 2, will look more like this:

null null	null A	A look	look into	into the	past. A	vision of
• A • The	• look	• into	• the	• past.	• vision	• the

of the (1)	the future. (2)	future. null	The future (3)	future of	the present.	present. null
• future. • present.	• null	• null	• of	• the	• null	• null

Why is (1) not “repeated” but (2/3) is?

Input Text: A look into the past. A vision of the future.
The future of the present.

Still not a tree as we think of it, and this is null/null rooted/terminated.

Markov/Trie Algorithm

For our purposes, if we map out all of our inputs, we can use the Trie as a qualifier to generate proper-sounding text. Because we are examining words, trying to make more words, we will set our Trie to be the exactly that: a collection of Strings that will tell us what Strings might follow.

Our steps then would resemble the following:

1. Create our trie of (order) String keys to a collection of Strings
    1. Remember to start and end with (order) number of anchors
2. Input each set of strings from our dictionary, mapping it piece by piece to the immediately following word
    1. Remember to check if our key already exists instead of always just creating a new one
3. Starting from your main anchor, randomly choose a suffix and output it, shift your key, and keep doing so until you've generated enough words.

Markov

Once you understand the theory of markov tries, a simple implementation is very straightforward, but start by considering a couple things:

- You must read in a file. How (or how not) do you want to store the information?
- if(map.contains(key)){put(key, new val());}
  is tedious, there's ***probably a method that does that***
- Test small datasets first, with small chunks of code. Don't wait until you have your parser and printer all written to try either of them!
  - At least try printing out the map of a small dataset

```java
/**
 * @version date ( in_CS_XXX_00X format : YYYY - MM - DD )
 * @author FirstName LastName
 **/

/** My class ClassName does ... */
public class ClassName {

    private memVarType memberVariable;

    /**
     * Getter for specified member variable.
     * @return this.memberVariable The memberVariable of this instance
     */
    public memVarType methodName() {
        return this.memberVariable;
    }
    /**
     * What does this method do?
     * @param param1 What is this parameter?
     * @return What are we returning?
     * @throws ExceptionName Why are we throwing this/what triggers this?
     */
    public varType methodName2 (memVarType param1) throws ExceptionName {
        if(param1 == this.memberVariable) {
            for(int i = 0; i <= param1; i++) {
                System.out,println(i + ". Our var = " + this.memberVariable)
            }
            throw new ExceptionName(ExceptionParameters);
        }
        return (this.memberVariable + param1);
    }
}
```

Info Block, tells who/when/what

Method is public, so it gets a JavaDoc comment

Method is public, so it gets a JavaDoc comment, this one's a bit longer because it has three fields

Indentations must be spaces, NOT Tabs

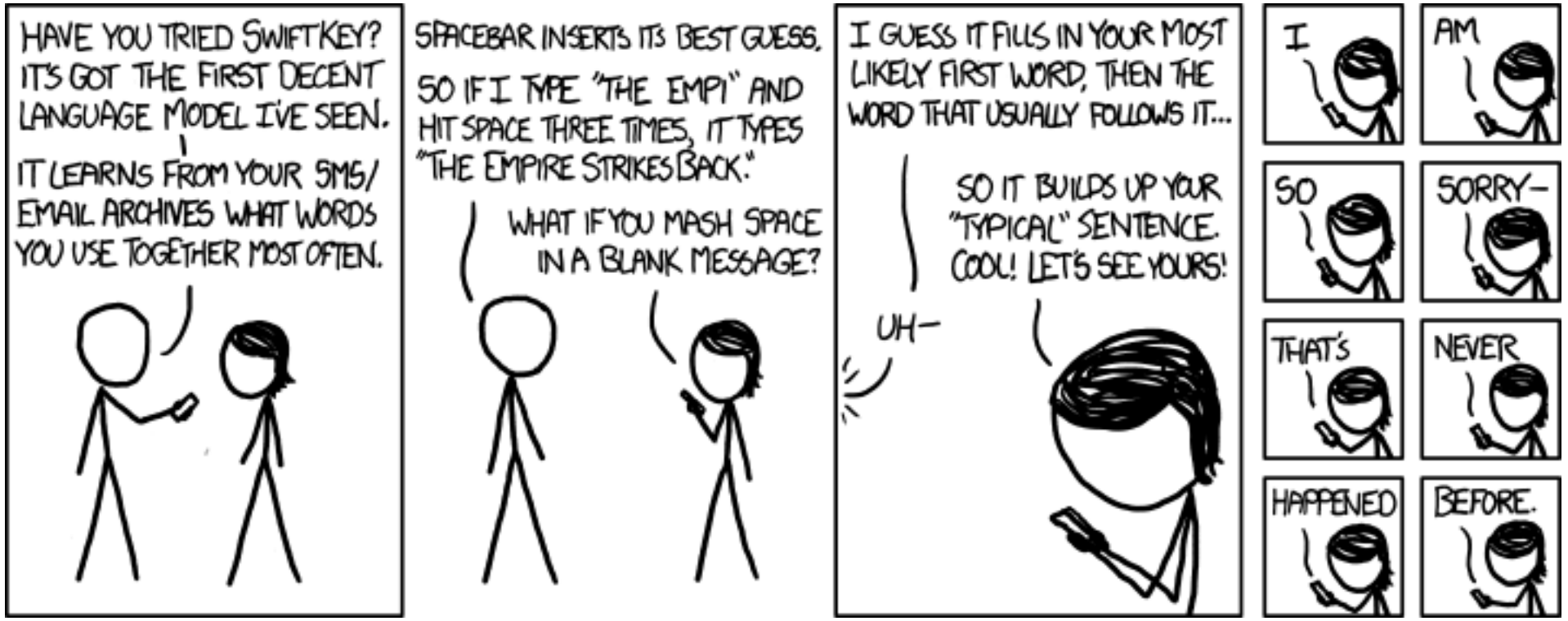
```java
import java.all.my.imports.*; // What imports from java do I need?
import cs251.interface; //(this isn't the import, but you need one)
/**
 * @version date ( in_CS_XXX_00X format : YYYY - MM - DD )
 * @author FirstName LastName
 **/
public class Demo {
    private NestedClass variable;
    private OtherCollection otherVariable;
    /** What my constructor does to prepare a new instance */
    public Demo() {
        this.variable = new NestedClass();
        this.otherVariable = prepareNewCollection();
    }
    /** This is that preparation method so we contain everything.
     * @return OtherCollection our own custom collection */
    private OtherCollection prepareNewCollection() {
        OtherCollection output = new OtherCollection();
        output.
        return output;
    }
    /* This is a nested class, it's private so no javadoc comment needed,
     * but we can say what it does here to make life easier */
    class NestedClass extends Collection implements ThatInterfaceWeAreUsing {
        @Override
        public int ThatOneMethodFromInterface() {
            return 9001;
        }
        @Override
        public double OtherMethodFromCollection(Collection args) {
            double[] math = /* oh hey we did something with args here*/;
            return math;
        }
```

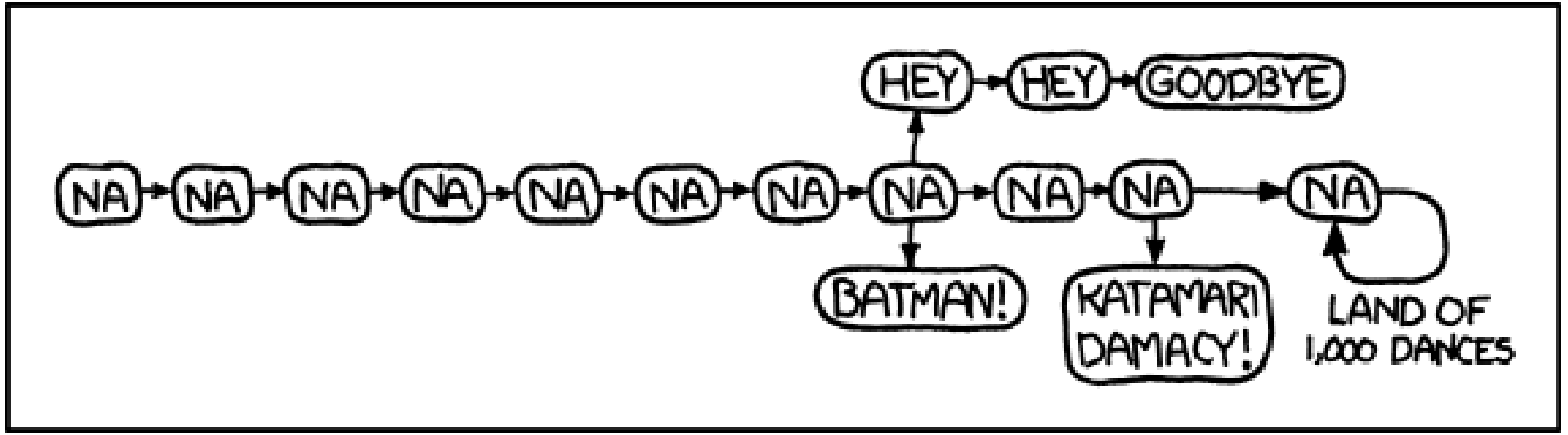
Info Block, tells who/when/what
No room on the slide, but you need a class comment too.

Constructors you write need comments too

Method is public, so it gets a JavaDoc comment.
It takes no arguments, but what does it do?

Nested class is private, so no need for JavaDoc, but comments are still a part of documenting your code, what is this class?


HAVE YOU TRIED SWIFTKEY? IT'S GOT THE FIRST DECENT LANGUAGE MODEL I'VE SEEN.
IT LEARNS FROM YOUR SMS/ EMAIL ARCHIVES WHAT WORDS YOU USE TOGETHER MOST OFTEN.
SPACEBAR INSERTS ITS BEST GUESS.
SO IF I TYPE "THE EMPI" AND HIT SPACE THREE TIMES, IT TYPES "THE EMPIRE STRIKES BACK."
WHAT IF YOU MASH SPACE IN A BLANK MESSAGE?
I GUESS IT FILLS IN YOUR MOST LIKELY FIRST WORD, THEN THE WORD THAT USUALLY FOLLOWS IT...
SO IT BUILDS UP YOUR "TYPICAL" SENTENCE. COOL! LET'S SEE YOURS!
UH-
I
AM
SO
SORRY-
THAT'S
NEVER
HAPPENED
BEFORE.


HEY
HEY
GOODBYE
NA
NA
NA
NA
NA
NA
NA
NA
NA
NA
NA
BATMAN!
KATAMARI DAMACY!
LAND OF 1,000 DANCES