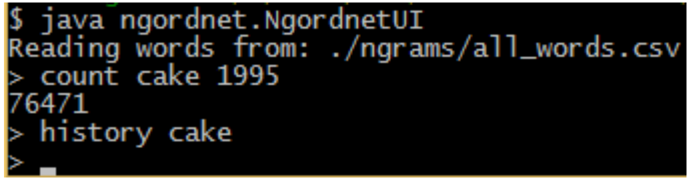
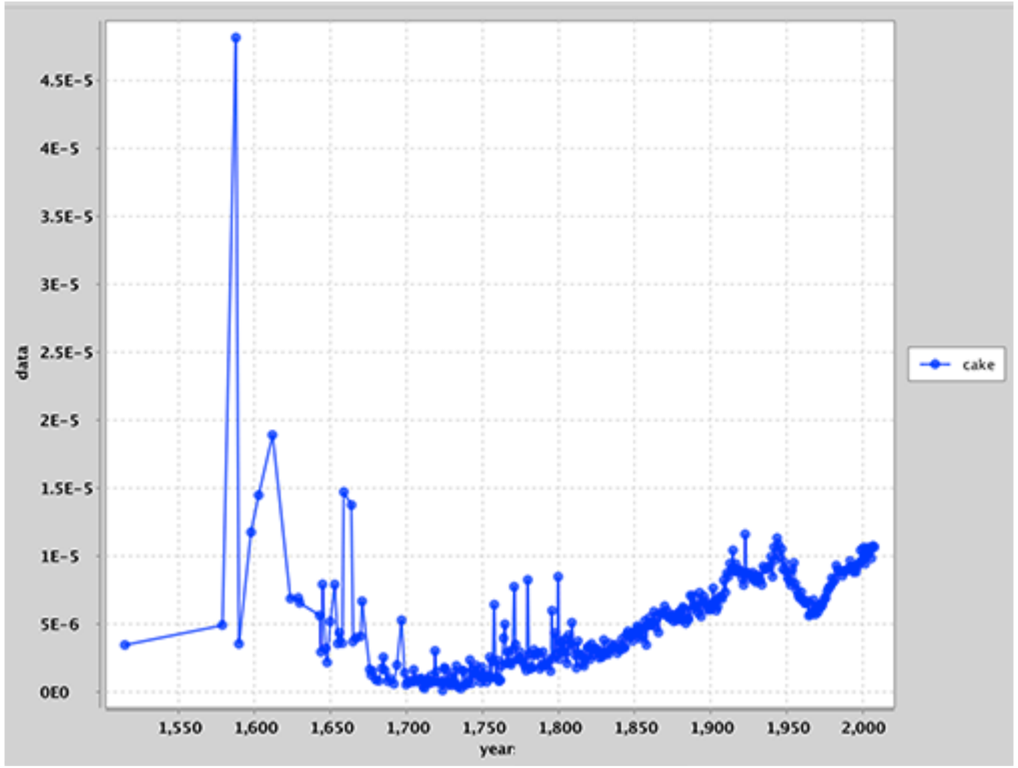
Goal

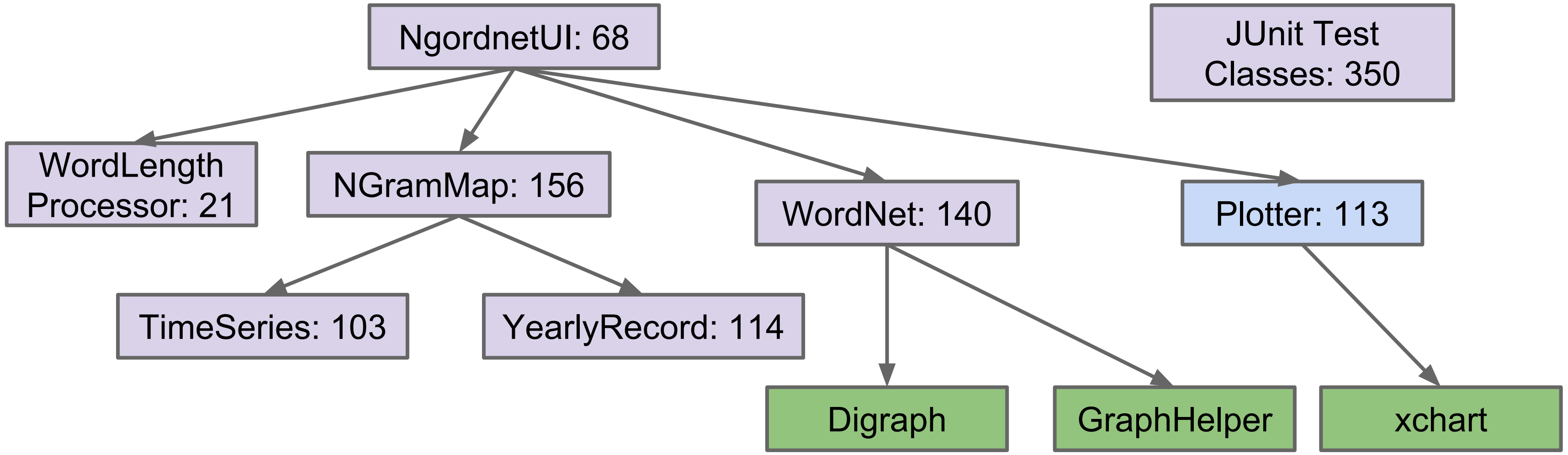
Our goal, create a text based user interface that allows queries about the histories of word frequencies and changes in English, e.g.



```
$ java ngordnet.NgordnetUI
Reading words from: ./ngrams/all_words.csv
> count cake 1995
76471
> history cake
> 
```



Overall System Architecture (Uses Relationships)



Notes:

- You’ll need to use other built-in Java datatypes (e.g. lists, maps, sets, etc).
- Classes in lavender are written by you. Number gives length of our solution.
- Green classes are provided.
- Blue class will have some major giveaways soon.

Recommended Order

1. WordNet
2. Make sure you understand the overall architecture (these slides).
3. (Do HW5 and Lecture 14 hardMode exercise first): TimeSeries
4. YearlyRecord
5. NGramMap (except processedHistory)
6. Plotter (except processedHistory and Zipf's Law)
7. NgordnetUI
8. WordLengthProcessor, also add processedHistory to #4 and #5
9. Zipf's Law
10. Post cool things that you discover on Piazza. Feel free to add new commands and public classes. For example, you might make a new YearlyRecordProcessor that uncovers something cool.

These slides will occur in narrative order (top down) to make a better story.

FAQ (on Piazza)

Your first place to look for common problems is here: https://piazza.com/class/hx9h4t96ea8qv?cid=3116!

Feel free to edit this post with things you felt particularly important.

NGordnetUI

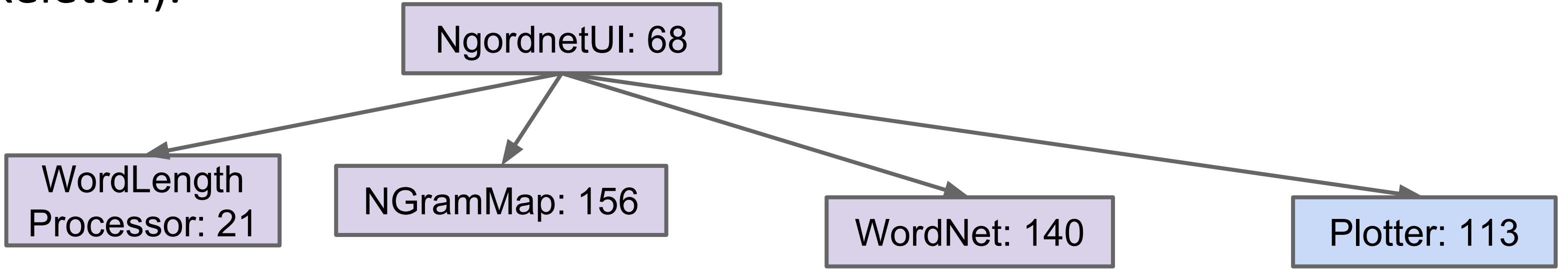
NgordnetUI

When NgordnetUI starts up, it will read ./ngordnet/ngordnetui.config to find the input files for NGramMap (storing histories of all words) and WordNet (is-a relationships between nouns).

- Uses first two files to build an NGramMap.
- Uses next two files to build a WordNet.

User then inputs commands to display interesting data from these structures.

- Plotter does the actual plotting using the xchart library (provided in skeleton).

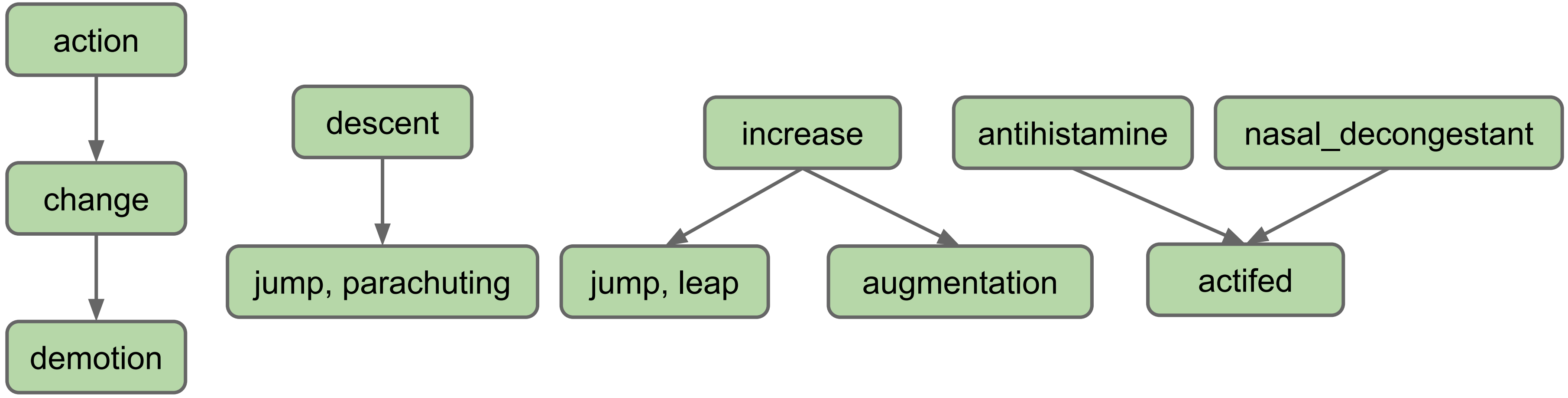


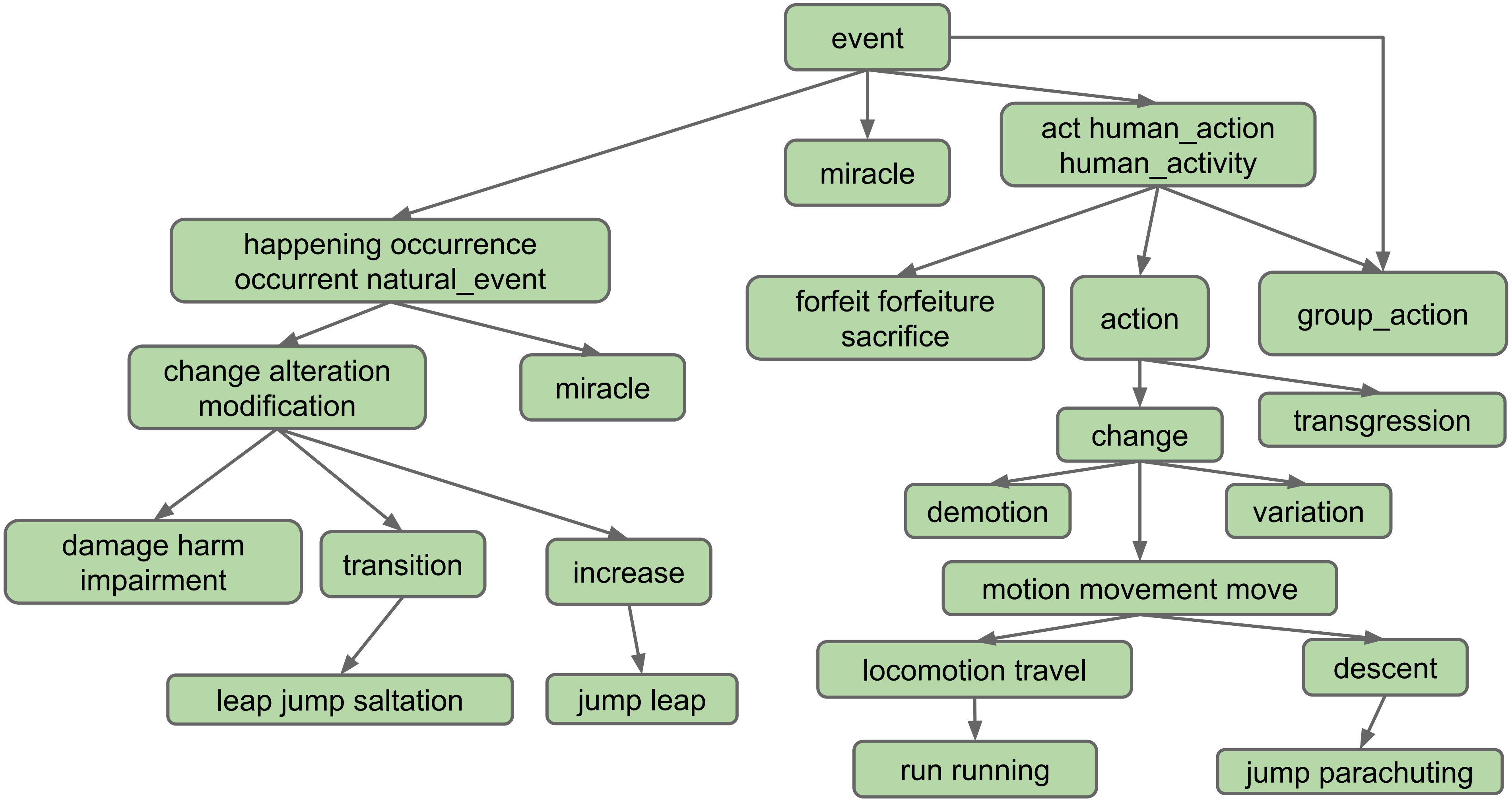
WordNet

Example: WordNet Hypernyms

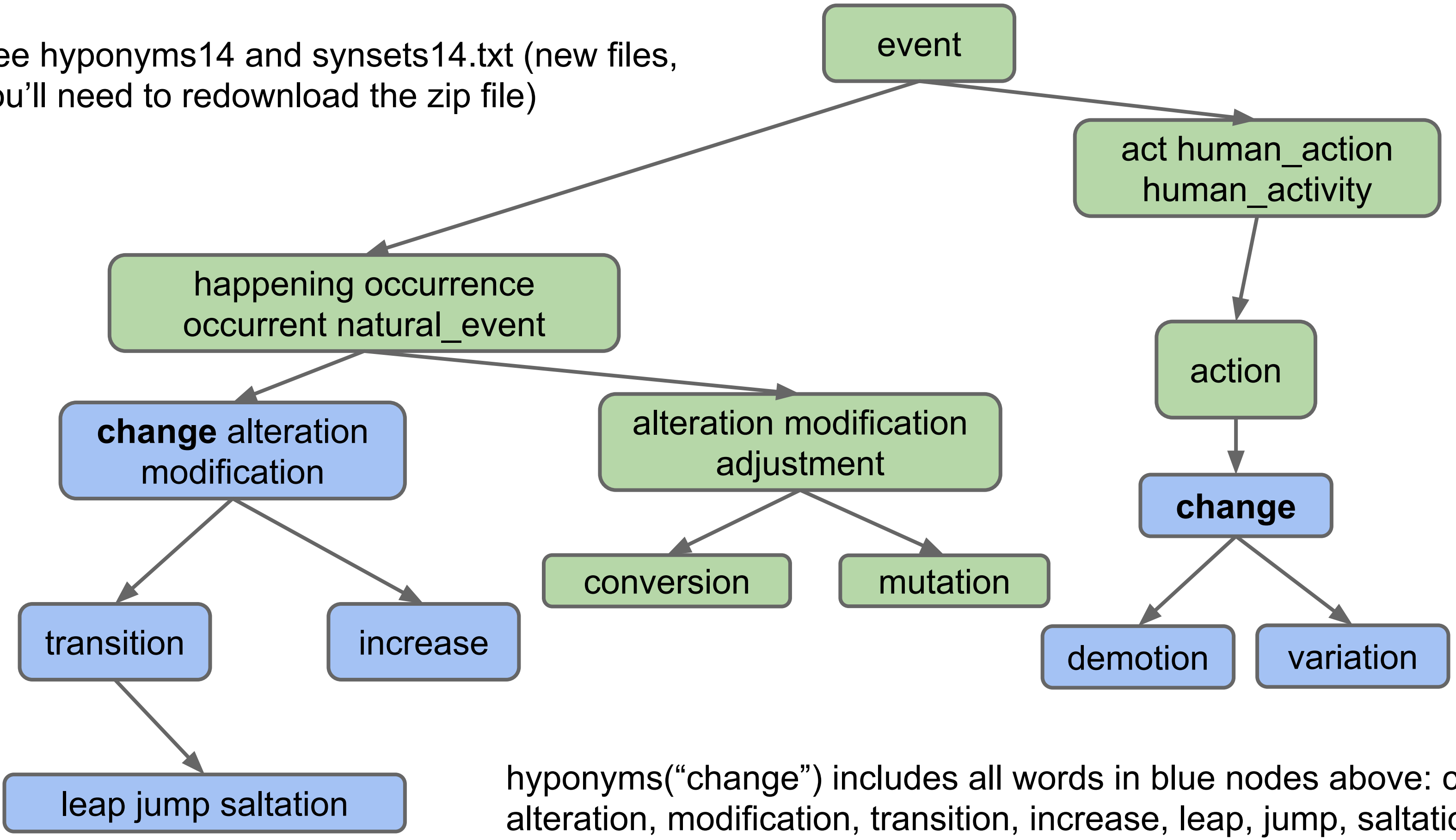
Each box is a synset: A set of one or more words representing the same idea.

- An arrow from one synset to another indicates an is-a relationship.
  - A demotion is-a change.
  - We say demotion is a hyponym of change (and change is a hypernym of demotion).
- Primary goal of this class is to find the hyponyms of a word.

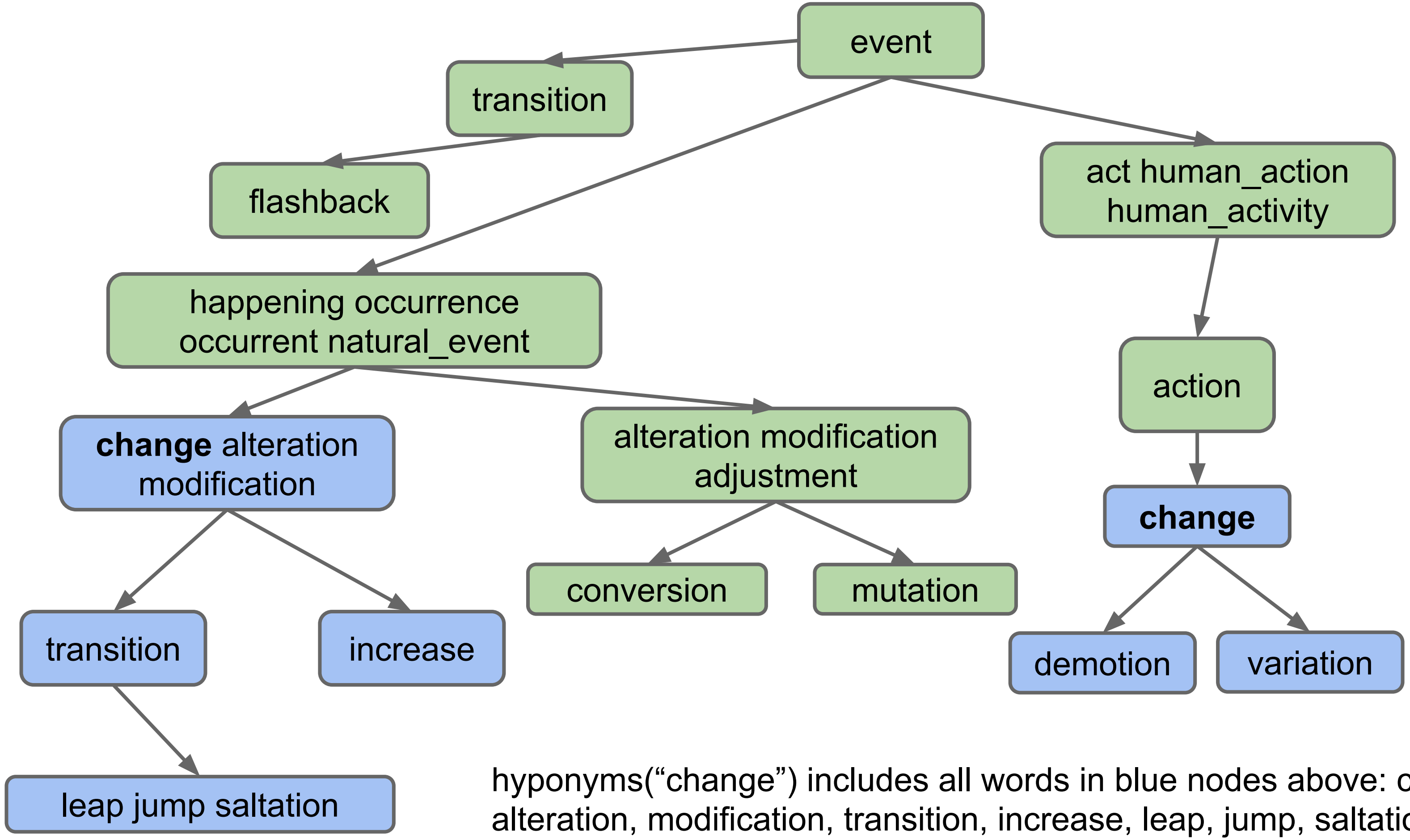



event
miracle
act human_action human_activity
happening occurrence occurrent natural_event
forfeit forfeiture sacrifice
action
group_action
change alteration modification
miracle
change
transgression
demotion
variation
damage harm impairment
transition
increase
motion movement move
leap jump saltation
jump leap
locomotion travel
descent
run running
jump parachuting

See hyponyms14 and synsets14.txt (new files, you'll need to redownload the zip file)


event
act human_action human_activity
happening occurrence occurrent natural_event
action
change alteration modification
alteration modification adjustment
change
transition
increase
conversion
mutation
demotion
variation
leap jump saltation

hyponyms("change") includes all words in blue nodes above: change, alteration, modification, transition, increase, leap, jump, saltation, demotion, variation.
Does not include hyponyms of synonyms (e.g. mutation and conversion)


event
transition
flashback
act human_action human_activity
happening occurrence occurrent natural_event
action
change alteration modification
alteration modification adjustment
change
conversion
mutation
transition
increase
demotion
variation
leap jump saltation

hyponyms(“change”) includes all words in blue nodes above: change, alteration, modification, transition, increase, leap, jump, saltation, demotion, variation.
Does not include hyponyms of synonyms (e.g. mutation and conversion) or hyponyms of other senses of hyponyms (e.g. transition)

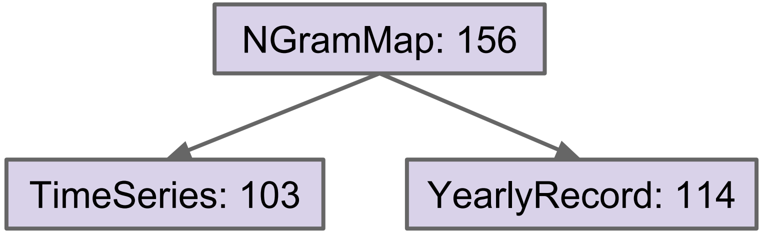
WordNet Tips

- The hyponyms method should perhaps have been named hyponymsAndSynonyms.
- You do not need to use Digraph or GraphHelper if you don't want to.
- It doesn't matter where you put the input files, as long as your tests know how to find these files.
- Write tests that you think might be useful. Don't waste tons of time writing ultra comprehensive tests. Finding the right balance is an art. For 61B, if the autograder complains, this is your signal to write more tests. The autograders will get more vague as the semester goes on.
- You should be able to read hyponyms.txt and synsets.txt (the big ones) in less than ~5 seconds (maybe a bit more on older computers).
- Make sure to check out WordNetDemo for examples (will not compile since you haven't written WordNet yet). Feel free to use as a basis for testing.

NGramMap, TimeSeries, YearlyRecord

NGramMap

Stores data files (described in spec), using TimeSeries and YearlyRecords as special purpose collections..



Goal is to allow high speed queries, with the most interesting queries below:

- Get entire history of a word or array of words, returned as a TimeSeries.
    - Absolute counts.
    - Relative frequencies.
- Get processed history of entire dataset, returns as a TimeSeries.
    - Example: Average word length in published volumes over time.

public class TimeSeries<T extends Number> extends TreeMap<Integer, T>

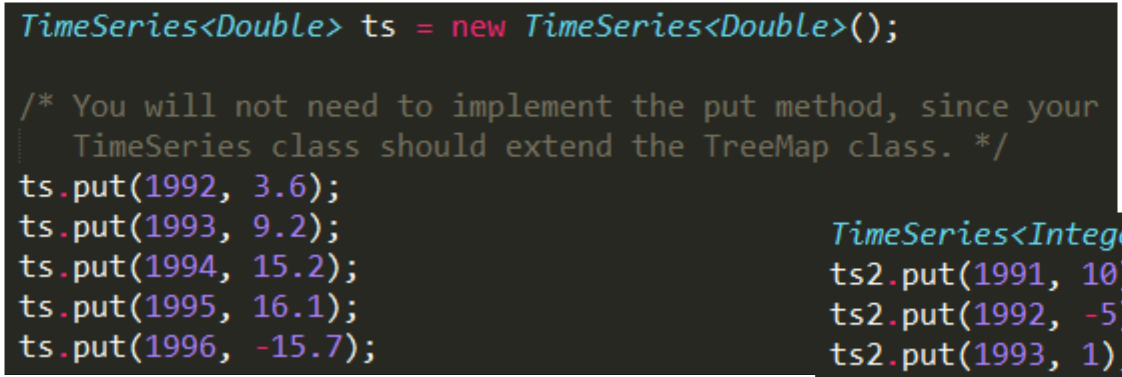
TimeSeries maps from an integer year number to some type of numerical data. Will be used for various purposes (including word counts).

Has all the TreeMap methods, as well as:

- `.plus(TimeSeries ts)`: Return ts plus this. Should be non-destructive.
  - Treat missing years as having value of 0.
- `.dividedBy(TimeSeries ts)`: Returns this divided by ts. Should be non-destructive.
  - If ts is missing any of my years, throw exception.
- `Collection<Number> years()`: Returns all years for this time series (in any order).
- `Collection<Number> data()`: Returns all data for this time series (in same order as years).

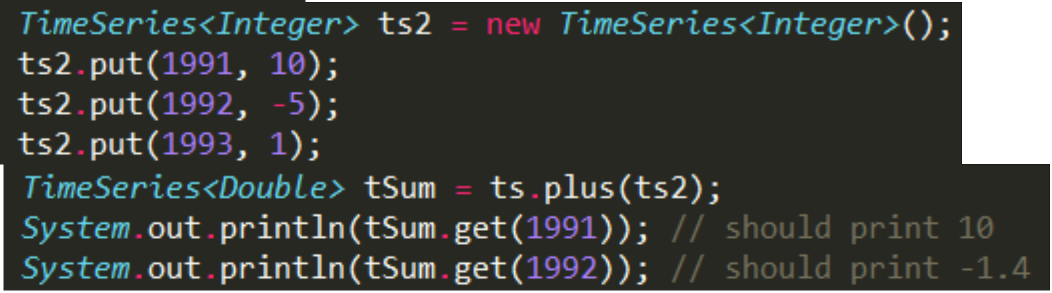
Plus and dividedBy

From TimeSeriesDemo.java:

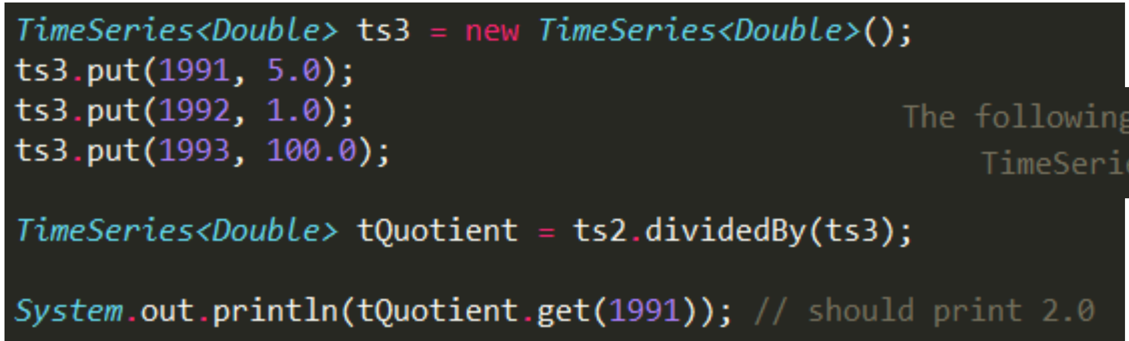


```java
TimeSeries<Double> ts = new TimeSeries<Double>();

/* You will not need to implement the put method, since your
   TimeSeries class should extend the TreeMap class. */
ts.put(1992, 3.6);
ts.put(1993, 9.2);
ts.put(1994, 15.2);
ts.put(1995, 16.1);
ts.put(1996, -15.7);
```



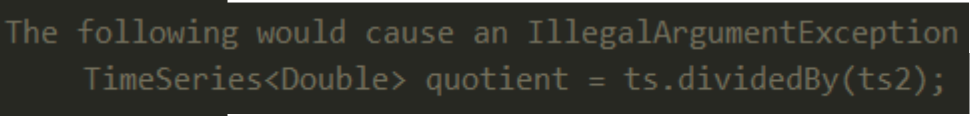
```java
TimeSeries<Integer> ts2 = new TimeSeries<Integer>();
ts2.put(1991, 10);
ts2.put(1992, -5);
ts2.put(1993, 1);
TimeSeries<Double> tSum = ts.plus(ts2);
System.out.println(tSum.get(1991)); // should print 10
System.out.println(tSum.get(1992)); // should print -1.4
```



```java
TimeSeries<Double> ts3 = new TimeSeries<Double>();
ts3.put(1991, 5.0);
ts3.put(1992, 1.0);
ts3.put(1993, 100.0);

TimeSeries<Double> tQuotient = ts2.dividedBy(ts3);

System.out.println(tQuotient.get(1991)); // should print 2.0
```



```java
The following would cause an IllegalArgumentException
    TimeSeries<Double> quotient = ts.dividedBy(ts2);
```

Collection<Number>

Why are we returning Collection<Number> instead of something easier?

- The Plotter class requires it, for example: http://xeiam.com/javadocs/xchart/com/xeiam/xchart/QuickChart.html#getChart-java.lang.String-java.lang.String-java.lang.String-java.lang.String-java.util.Collection-java.util.Collection-

Important notes about Generics in Java: They are not covariant.

- Collection<Double> is not a Collection<Number>
- However, ArrayList<Number> is-a Collection<Number>

This means you'll need to build a new Collection<Number> for these methods. Since Integer is-a Number and Double is-a Number this shouldn't be too bad. It might be a little annoying, but it'll be great practice with generics!

YearlyRecord

Used to store the history of all word counts for a single year. Provides methods:

- put(String word, int count): Stores count of a given word. May be overwritten by later put counts.
- int count(String word): Returns count of a given word.
- int size(): Number of words.
- Collection<String> words(): Returns word in ascending order of count.
- Collection<Number> counts(): Returns counts in ascending order.
- int rank(String word): Returns rank of a word. Most common is 1, next most common is 2, and so forth. Break ties arbitrarily.

Multiple words may have the same count, but this should not be a hassle.

Performance Requirements for YearlyRecord

Assuming there are no more put() calls, rank(), size(), and count() must be so fast that they are effectively independent of the size of the dataset.

- We call a YearlyRecord for which there are no more put() calls to be “frozen”. This is not an actual property of YearlyRecord, and just a convenient fiction for making performance requirements less ambiguous.
- Will formalize these notions over the coming lectures.

For 0.1 bonus points, put() should also be close to independent of dataset size. This isn’t hard, it’s just a mess-up in the spec that I didn’t want to make into a requirement. See next slide for more details.

- Warning: You won’t be able to generate really cool plots in later parts of this assignment unless you do this 0.1 bonus point.

In other words...

Sorry for the lack of precision here. We haven't learned asymptotics yet, but it still seemed worthwhile to shoot for good performance.

Suppose we create a YearlyRecord x1, and call x1.put() 1,000 times, then call x1.rank(), x1.size(), and x1.count() 1,000,000 times each.

Suppose we then create a new YearlyRecord x2 and call x2.put 1,000,000 times, then call x2.rank(), x2.size(), and x2.count() 1,000,000 times each.

- Every call to x1.put and x2.put should take around the same amount of time (for those who want a precise bound and know this from 61A: let's say O(log N))
- ***<u>On average</u>***, all calls to x1.rank(), x1.size(), x1.count(), x2.rank(), x2.size(), and x2.count() should take around the same time (though any individual call might be slow).

You may assume that all map operations are fast. You may not assume that list operations are fast.

TimeSeries, YearlyRecord, and NGramMapDemo Tips

- Make sure to see the TimeSeriesDemo, YearlyRecodDemo and NGramMapDemo files.
- Do not let yourself be broken on the wheel of syntax. Seek help if you spend too much time trying to resolve a tiny issue (particularly involving generics). If you haven’t done lec14 hardMode yet, maybe go back to that if you’re stuck.

Plotter

Visualization

The Plotter library is built to allow us to visualize interesting NGramMap queries. Uses xchart library.

- Learning to use libraries on your own is a hugely important skill, and you'll do this a lot at internships.
- If you decide to write Plotter yourself, you should lean heavily on the provided examples, as well as the xchart documentation.
- However, your time is limited, so if you'd like reference code to use / work from, see: http://joshh.ug/plotter.html

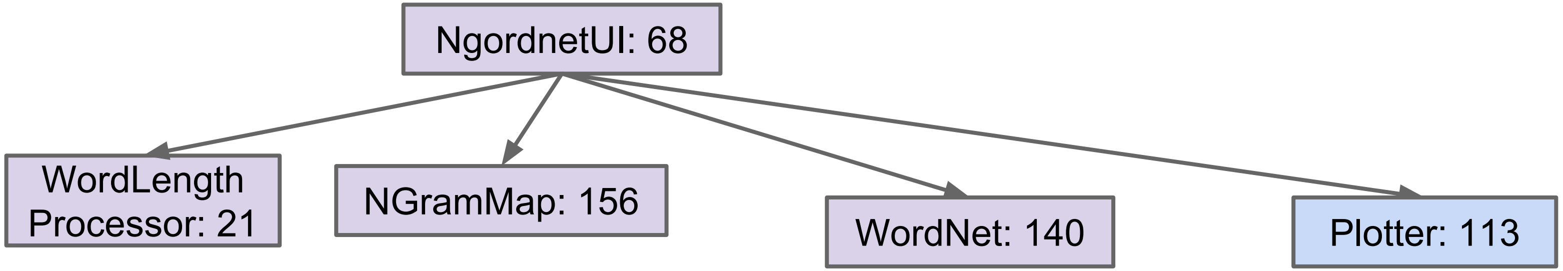
Don't worry about matching our plots exactly. We won't grade your Plotting class.

NgordnetUI

Implementing NgordnetUI

The first step: Make sure the given NgordnetUI.java compiles and runs, and that the ngordnetui.config is set up with the right paths for your computer.

Make sure to see ExampleUI.java for examples of how to parse user input.



WordLengthProcessor

WordLengthProcessor

The YearlyRecordProcessor interface defines a very simple ADT:

- Provide a YearlyRecord, get back a double.

WordLengthProcessor should be straightforward. You give it a yearly record, and it provides the average length of all words printed that year.

For example, suppose in the year 15000 BC, there were two printed books that said:

- it rained
- it rained again

Then the average word length would be (2+2+6+6+5)/5 since there were 5 words.

WordLengthProcessor

Once you’ve written WordLengthProcessor, add the appropriate commands to NgordnetUI and a plotting method to Plotter.

Use it to see the history of word lengths since books have printed.

- Notice anything interesting?

Zipfs Law and Beyond

Zipfs Law… and Beyond

In this final part of the assignment, you’ll plot counts of words vs. their ranks (finally rank becomes useful!). See the spec for more details. If you use our Plotter.java, this part should be relatively painless.

Feel free to add additional features. Post anything cool you learn on Piazza:

- Thread coming later (don’t want to spoil the discoveries I expect you to make)