Assignment 2: Linked Lists

In this assignment, you will be reading in words from a pronunciation dictionary to figure out which words rhyme with each other.

To do this, you will be implementing MyLinkedList and MySortedLinkedList. Besides the lectures in class, you can read much more about the code details of how linked lists work in the book on pages 249-268.

You'll be implementing the following:

- A linked list in `MyLinkedList` that implements interface `ListInterface`. You'll be using `MyLinkedList` in two different ways, to store a list of `RhymeGroupWords` and to store a list of `Strings` within each `RhymeGroupWords` object. More on this below.
- Code in the main method of `RhymingDict` that takes lines that have been read from a rhyming dictionary and stores them in the lists.
- A subclass of `MyLinkedList` called `MySortedLinkedList`, that has a method for adding items into the list in sorted order.
- Code in the main method of `RhymingDict` that takes pairs of words specified on the command line and outputs whether they rhyme or not.

A LinkedList is useful for storing information when you don't know how many items you need to store. In this lab, we will be iterating through a rhyming dictionary and storing words together according to their rhyming group, which is the most heavily emphasized part of their pronunciation.

In the directory of starter code for the assignment, you'll see a subdirectory called `cmudict`. Inside is a file called `cmudict-short.dict`. Opening this, you'll see lines like:

```
dignitary D IH1 G N AH0 T EH2 R IY0
necessitate N AH0 S EH1 S AH0 T EY2 T
undue AH0 N D UW1
moderated M AA1 D ER0 EY2 T IH0 D
```

The letters after the words represent the phonemes (units of sound) you speak when you pronounce the word. The phonemes that have numbers after them (0, 1 or 2) are the emphasized phonemes, with phonemes marked with 2 having the most emphasis (say the words out loud to yourself to try this out). Two words rhyme if the share the same subset of phonemes after the most emphasized phoneme. For example:

```
moderated M AA1 D ER0 EY2 T IH0 D
anticipated AE0 N T IH1 S AH0 P EY2 T IH0 D
```

both rhyme, and they both share `EY2 T IH0 D` starting with the most emphasized phoneme. The `EY2 T IH0 D` is called the rhyming group for these two words. You'll be storing rhyming groups in a linked list of `RhymeGroupWords`, a class you've been provided in the starter code. Each `RhymeGroupWords` object stores a string with the rhyme group `EY2 T IH0 D` and a list of words that share the rhyme group (a variable of type `ListInterface`). So you'll be using linked lists in two different ways: to store a list of `RhymeGroupWords`, and, within each `RhymeGroupWords` object, to store a list of words.

RhymingDict

The file RhymingDict.java is provided for you. It already has the following:

- A method `getRhymeGroup(String line)`, which returns the rhyme group for a line from the dictionary. For example `getRhymeGroup("moderated M AA1 D ER0 EY2 T IH0 D")` returns `"EY2 T IH0 D"`.
- A method `getWord(String line)`, which returns the word for a line from the dictionary. For example, `getWord("moderated M AA1 D ER0 EY2 T IH0 D")` returns `"moderated"`.
- A method `loadDictionary()`, which returns a `String[]` of all the lines from the pronunciation dictionary.

The tasks you have to do is marked with the comment `TODO` in the main method of `RhymingDict`. These tasks are:

1. Use your implementation of `MyLinkedList` to create a linked list of `RhymeGroupWords`, one for each unique rhyme group you encounter (there are 8 unique rhyme groups in `cmudict-short.dict`). Within each RhymeGroupWords you'll store a MySortedLinkedList of words that share that rhyme group.

2. Iterate through pairs of arguments to `RhymingDict` and use your linked list to determine if the pair of words rhyme. For example, with the command:

```
java RhymingDict crumbling mumbling collections abbreviated vegetate mutate
```

the output would be:

```
crumbling and mumbling rhyme
collections and abbreviated don't rhyme
mutate is not in the dictionary
```

In the case of the last pair of words, `"vegetate"` and `"mutate"`, since the word `"mutate"` doesn't appear in `cmudict-short.dict`, it reports that and ignores `"vegetate"`. If an odd number of words is passed on the command line, it ignores the last word (since it's not part of a pair).

Do **not** use built-in Java data structures (like ArrayList or LinkedList) in this file

Grading

The points for this assignment are allocated as follows:

- 35 points for implementing `MyLinkedList`. Even if you don't get the rest of the assignment working, you can still earn points for a correct implementation of `MyLinkedList` (we will be running test code to test it).
- 30 points for correctly storing the rhyme groups and words in linked lists using `MyLinkedList` for both the `RhymeGroupWords` list and the word lists in each `RhymeGroupWords`. In this case the words won't appear in sorted order in the list.
- 20 points for implementing and using `MySortedLinkedList` to store the word lists. Now the words within each rhyme group will appear in sorted order.
- 15 points for writing the code that processes arguments to determine if pairs of arguments rhyme.

Our recommendation is that you do the parts of the assignment in this order. There is some commented out test code in main for testing `MyLinkedList` without having to store the pronunciation dictionary in it. Feel free to edit this code to do more tests, but make sure to comment it out before you turn in your assignment. Once you have `MyLinkedList` working with your tests, then use it to store the data from the dictionary. Test if you're getting 8 rhyme groups. The output below shows what the rhyme groups will look like with the words in sorted order, though at this point your words won't be sorted yet. Then implement `MySortedLinkedList` and use it for storing the word lists. Now your rhyme groups should print out with the words in sorted order. Finally, write the code that searches through the lists to see if a pair of words rhyme.

Files

The following files are provide for you:

- `RhymingDict.java` - described above
- `ListInterface.java` - specifies the methods you need to implement in `MyLinkedList`
- `MyLinkedList.java` - an empty class file that implements `ListInterface`
- `MySortedLinkedList.java` - an empty class file that extends `MyLinkedList`. Comments in the file specify what the one method and one method override should be
- `RhymeGroupWords.java` - a class for storing a rhyme group and the list of words that share that rhyme group (described above)
- `ListIndexOutOfBoundsException.java` - an exception to throw when a list index is out of bounds in `MyLinkedList`.

Turning the code in

- Create a directory with the following name: `<student ID>_assignment2` where you replace `<student ID>` with your actual student ID. For example, if your student ID is 1234567, then the directory name is `1234567_assignment1`.

- Put a copy of your edited `RhymingDict.java`, `MyLinkedList.java` and `MySortedLinkedList.java` files in the directory.
- Compress the folder using zip. Zip is a compression utility available on mac, linux and windows that can compress a directory into a single file. This should result in a file named `<student ID>_assignment1.zip` (with <student ID> replaced with your real ID of course).
- Upload the zip file through the page for Assignment 2 in canvas (https://canvas.ucsc.edu/courses/12730/assignments/39157).

Example Arguments and Output

Below I show the output for one example command. You should of course test your code with different words as arguments as well.
For the example command:

```
java RhymingDict crumbling mumbling collections abbreviated vegetate mutate
```

the output will be as follows. First it prints out the 8 rhyme groups and all their words, then which pairs of words on the command line rhyme. The words in this case are in alphabetical order because they were stored using `MySortedLinkedList`.

```
EH2 R IY0: (adversary, apothecary, arbitrary, banbury, blackberry,
canterbury, cemetery, commentary, confectionery, constabulary, culinary,
customary, depositary, dictionary, dietary, dignitary, disciplinary,
discretionary, dromedary, dubarry, dysentery, emissary, estuary, feb,
february, functionary, gooseberry, hereditary, honorary, itinerary,
january, judiciary, legendary, literary, luminary, mercenary, military,
missionary, momentary, monastery, monetary, mortuary, mulberry, necessary,
obituary, ordinary, pecuniary, primary, proprietary, reactionary, rosemary,
salutary, sanitary, savagery, secondary, secretary, sedentary, seminary,
solitary, stationary, stationery, statuary, strawberry, temporary,
tributary, unnecessary, unsanitary, veterinary, visionary, vocabulary,
westbury)
EY2 T: (abdicate, abrogate, accelerate, accommodate, accumulate, aggravate,
alienate, alleviate, amalgamate, ameliorate, amputate, annihilate,
anticipate, arbitrate, arrogate, aspirate, assassinate, assimilate,
captivate, carbonate, checkmate, circulate, commemorate, commiserate,
communicate, compensate, complicate, concentrate, congregate, consecrate,
consolidate, contaminate, contemplate, cooperate, corroborate, culminate,
cultivate, decapitate, decorate, dedicate, delegate, delineate,
demonstrate, denominate, depopulate, depreciate, designate, deteriorate,
determinate, devastate, deviate, discriminate, disintegrate, disseminate,
dissipate, educate, elevate, elucidate, emanate, emancipate, emigrate,
enumerate, evacuate, exasperate, excavate, expiate, extenuate, exterminate,
```

extirpate, extricate, facilitate, fascinate, felicitate, flagellate, fluctuate, generate, gravitate, hesitate, humiliate, hundredweight, illustrate, imitate, immolate, implicate, inculcate, indicate, inmate, instigate, isolate, legislate, locate, lubricate, magistrate, mandate, mediate, meditate, migrate, militate, mitigate, mutilate, narrate, navigate, necessitate, negotiate, nitrate, obliterate, obviate, officiate, operate, originate, overstate, paperweight, participate, penetrate, perforate, permeate, perpetuate, playmate, pontificate, precipitate, predominate, premeditate, primate, probate, procrastinate, profligate, propagate, punctuate, recreate, recuperate, regenerate, regulate, reiterate, repudiate, retaliate, rotate, ruminate, saturate, schoolmate, segregate, separate, shipmate, situate, stagnate, stimulate, stipulate, subjugate, sulphate, terminate, underestimate, undulate, vacillate, vegetate, venerate, vertebrate, vitiate)
UW1: (accrue, adieu, ado, anew, askew, bamboo, bleu, blew, blue, boo, brew, canoe, chew, cou, coup, crew, crewe, cue, debut, deux, dew, doo, drew, du, due, ensue, ewe, few, flew, flue, glue, goo, grew, gu, gue, hew, hoo, hu, hue, jew, jus, kew, knew, ku, misconstrue, moo, new, nu, ooh, ou, peru, pew, phew, pooh, pursue, q, qu, que, queue, revue, rue, shampoo, shew, shoe, shoo, sioux, slew, stew, strew, su, subdue, sue, taboo, threw, through, to, too, true, two, u, undo, undue, untrue, view, vous, vue, whew, who, withdrew, woo, you, zoo)
EY2 T IH0 D: (abbreviated, abdicated, abrogated, accelerated, accentuated, accommodated, accumulated, adulterated, advocated, aggravated, aggregated, agitated, alienated, alleviated, amalgamated, ameliorated, amputated, animated, annihilated, anticipated, antiquated, appreciated, appropriated, articulated, asphyxiated, assassinated, assimilated, associated, attenuated, bifurcated, calculated, capitulated, captivated, castrated, celebrated, circulated, collaborated, commemorated, communicated, concentrated, congratulated, congregated, consecrated, contaminated, contemplated, corroborated, corrugated, cultivated, debilitated, decimated, degenerated, deliberated, delineated, demonstrated, denominated, depreciated, designated, deteriorated, devastated, deviated, dilapidated, discriminated, dissipated, duplicated, educated, elevated, eliminated, emanated, emancipated, emigrated, emulated, enumerated, enunciated, eradicated, exasperated, excavated, excoriated, exonerated, exterminated, facilitated, gravitated, hesitated, humiliated, imitated, implicated, inaugurated, infiltrated, intimated, inundated, irrigated, manipulated, migrated, miscalculated, moderated, narrated, navigated, obligated, obliterated, opinionated, outdated, oxygenated, participated, penetrated, perforated, permeated, perpetrated, perpetuated, populated, precipitated, predominated, premeditated, procrastinated, promulgated, propagated,

```
radiated, reciprocated, recuperated, regenerated, regulated, reiterated,
rejuvenated, relegated, renovated, repudiated, resuscitated, retaliated,
reverberated, ruminated, saturated, segregated, separated, serrated,
simulated, situated, sophisticated, speculated, stimulated, stipulated,
subjugated, subordinated, substantiated, suffocated, syncopated, tabulated,
tolerated, truncated, uneducated, unmitigated, venerated, ventilated)
EH1 N D IH0 NG: (amending, ascending, bending, blending, commending,
defending, depending, descending, ending, extending, lending, mending,
offending, pending, portending, pretending, rending, sending, spending,
suspending, tending, transcending, trending, unending, wending)
AH1 M B AH0 L IH0 NG: (crumbling, grumbling, humbling, mumbling, rumbling,
stumbling)
IY1 Z AH0 N AH0 B L IY0: (reasonably, unreasonably)
EH1 K SH AH0 N Z: (affections, collections, complexions, connexions,
corrections, elections, erections, objections, projections, protections,
reflections, rejections, sections)
crumbling and mumbling rhyme
collections and abbreviated don't rhyme
mutate is not in the dictionary
```