Assignment 5: The Java String Quartet

The last three assignments have focused on graphical programming and on software that manipulates numbers. This assignment is a quartet of string-processing assignments designed to give you a feel for how to manipulate text and sequences. By the time you've completed it, you'll have a much better sense for how computers store, manipulate, and analyze textual information.

Due Wednesday, February 18 at 3:15PM

Part One: Syllable Counting

Your job in this is to write a method

```
private int syllablesInWord(String word)
```

that takes in a single word and returns the number of syllables in that word.

It is difficult to determine the exact number of syllables in a word just from its spelling. For example, the word `are` has just one syllable, while the similarly-spelled `area` has three. Therefore, we aren't going to ask you to count syllables exactly. Instead, you should approximate the number of syllables using the following heuristic: count up the number of vowels in the word (including 'y'), *except* for

- Vowels that have vowels directly before them, and
- The letter *e*, if it appears by itself at the end of a word.

Notice that under this definition, the word `me` would have zero syllables in it, since the 'e' at the end of the word doesn't contribute to the total. To address this, *your method should always report that there is at least one syllable in any input word*, even if the heuristic would otherwise report zero. This resulting method will correctly estimate that there are two syllables in `quokka` and two syllables in `springbok`, though it does get the number of syllables in `syllable` wrong (guessing two instead of three). With the zero-syllable correction in place, the heuristic correctly counts syllables in the words `the`, `be`, and `she`.

Think about how the heuristic works and choose test words that ensure your code correctly implements the heuristic. When testing, don't worry if the answer your program gives isn't exactly the correct number of syllables in the word (you're using a heuristic, after all), but do make sure that your program produces values that agree with what we've described above. Here are some test cases to try out, though these are by no means exhaustive:

- `Unity`: 3 syllables
- `Unite`: 2 syllables
- `Growth`: 1 syllable
- `Possibilities`: 5 syllables
- `Nimble`: 1 syllable (technically there are two, but our heuristic reports one)
- `Me`: 1 syllable
- `Beautiful`: 3 syllables
- `Manatee`: 3 syllables

Part Two: Algorism Algorithms

As mentioned in Chapter 3 of *The Art and Science of Java*, Java's primitive data types (`int`, `double`, etc.) have ranges on what values they can store. `int`, for example, can't hold values greater than 2,147,483,647. Since we may want the computer to work with values larger than this (say, for example, the US national debt or the number of atoms in one gram of carbon), programmers sometimes use other types (such as `String`) for large integers. For example:

```
String worldPopulationin2010    = "6993309969";
String possibleRubiksCubeStates = "43252003274489856000";
```

In order to make use of numbers encoded this way, we have to have some way to add them. In grade school, you probably learned how to add two large numbers one digit at a time. You write the two numbers out, one on top of the other, then work from the right to the left adding the individual digits of the numbers. When the digits sum up to a value greater than ten, you would write out just the ones digit of their sum, then carry a 1 into the next column. For example, here's 137 + 864:

$$\begin{array}{cccc} 1 & 1 & 1 & \\ & \mathbf{1} & \mathbf{3} & \mathbf{7} \\ + & \mathbf{8} & \mathbf{6} & \mathbf{4} \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array}$$

In this part of the assignment, we'd like you to write a method

```
private String addNumericStrings(String n1, String n2)
```

This method will accept as input two numbers represented as `String`s. It should then return a `String` representing the sum of the two input numbers. To compute this sum, you'll program the computer to do arithmetic the same way that you learned in grade school: work from right to left, summing the digits of each string, and keeping track of the carry across columns.

When implementing this method, you can assume the following:

- The two input strings will be nonnegative (that is, you'll never get an input like "-137").
- The two input strings will consist purely of digits and will be valid representations of numbers. Therefore, you will never get inputs like "+3", "1,000,000,000", or "".
- The two input strings will not necessarily be the same length as one another.

We recommend implementing this method in stages. First, write a method that adds two input numbers, assuming they're the same length and that no carrying will be needed involved. Then, implement carrying. Finally, generalize your method so that it works on numbers of differing lengths.

There is one important detail about the `char` type that you're likely to run into when working on this part of the assignment. Internally, each `char` is represented by a numeric value. For example, the character `'A'` has value 65, and the character `'*'` has value 42. Somewhat counterintuitively, the characters `'0'`, `'1'`, `'2'`, etc. have numeric values 48, 49, 50, etc. If you want to get the numeric value that's represented by a particular character, you can do so by writing

```
int value = ch - '0';
```

Similarly, if you have the numeric value of a digit, you can obtain its `char` representation by writing

```
char ch = (char)(value + '0');
```

The art of performing arithmetic with a place-value system is called *algorism*. Once you've finished part of the assignment, you'll have programmed the algorism algorithm and have taught the computer to do arithmetic the way that you did back in grade school. Pretty neat!

Part Three: The License Plate Game

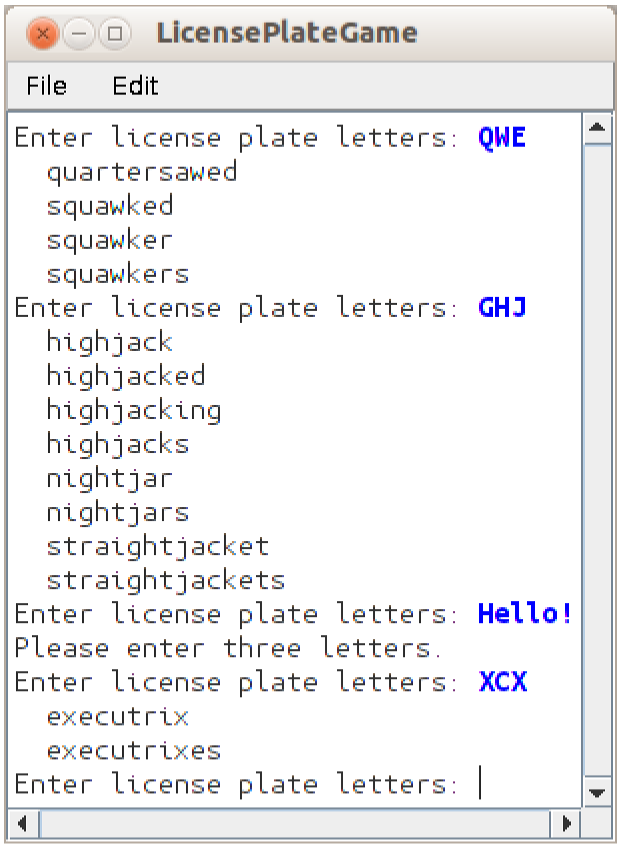
If you've ever driven on California streets, you've probably noticed that California license plates always consist of a number, three letters, then another three numbers. For example:

5NPT839 4KDD232 7FTW142

On a long highway drive in California, one fun way to pass the time is to play the *license plate game*, in which you look at the license plate of the nearest car, then try to find a word that uses the letters in that license plate in the order in which they appear. For example, for the license plate 5NPT839, you might use the word I**N**U**P**U**T**. For 4KDD232, you could use **K**I**DD**ING. For 7FTW142, you could use A**FT**ERGLO**W**.

For a word to match a license plate, it has to contain all the letters in the license plate in the order in which they occur. This means, for instance, that FTW doesn't match **W**A**T**ER**F**ALL because the letters are in the wrong order. Similarly, NNN doesn't match E**N**TIRE, since only one of the N's was matched.

Certain letter combinations are a lot harder to find words for than others. The letter combination YRW only matches a handful of words (like E**Y**EB**R**O**W**). The combination XCX only matches two English words (E**X**E**C**UTRI**X**, a woman who executes a will, and its plural E**X**E**C**UTRI**X**ES), and some combinations don't match anything at all. There are no English words that match QQQ, for example.


LicensePlateGame

File Edit

```
Enter license plate letters: QWE
  quartersawed
  squawked
  squawker
  squawkers
Enter license plate letters: GHJ
  highjack
  highjacked
  highjacking
  highjacks
  nightjar
  nightjars
  straightjacket
  straightjackets
Enter license plate letters: Hello!
Please enter three letters.
Enter license plate letters: XCX
  executrix
  executrixes
Enter license plate letters:
```

Your job in this part of the assignment is to write a program that prompts the user for a three-letter string, then prints all English words that match that pattern. Your program should sit in a loop prompting the user to enter a three-letter string (which can be upper-case or lower-case), reprompting as necessary until the user enters a string consisting of three letters. Then, your program should list all English words that match that string. There's a screenshot of the program in action on the left.

We've provided you a file (`dictionary.txt`) containing all words in the dictionary, which you'll need for this assignment.

There are many ways you can implement this program. You may find the `String`'s `indexOf` method useful here, though it's not strictly necessary.

Although we haven't talked about efficiency this quarter, file reading is slower than most other program operations. For full credit, your program should only read in the dictionary file once. Other than that, don't worry about efficiency.

Part Four: CSV Parsing

In class, we saw how to read a file one line at a time. This is a great way to read data if the file is structured so that there is one piece of data per line. Most of the time, though, data isn't stored this way. This question explores a common data format called the *comma-separated value format* (or CSV).

The CSV format is a common data format for storing tables. Each row of the table is represented by one line in the CSV file, where the columns are delimited by commas (hence "comma-separated values.") As an example, here is a sample table and what its CSV representation might look like:

Dish	Origin
Ful Medammas	Middle East
Vindaloo	India, Portugal
Maafe	West Africa
Soondubu	Korea
Kuku	Iran, Azerbaijan
Shakshuka	North Africa

```
Dish,Origin
Ful Medammas,Middle East
Vindaloo,"India, Portugal"
Maafe,West Africa
Soondubu,Korea
Kuku,"Iran, Azerbaijan"
Shakshuka,North Africa
```

Notice that each row is formed by writing out the contents of each of the cells separated by a comma. You'll also notice that the cell contents are glued together by a comma with no intervening spaces. For example, in the line for Soondubu, notice that there is no space between the comma and `Korea`. In some cases, the contents of a cell might contain a comma (for example, the origin entries for vindaloo and kuku). In that case, the contents of the cell will be surrounded by quotation marks to indicate that the comma is a part of the field and not a separator. Other cells may optionally be quoted, even if the cell doesn't contain an internal comma.

We'd like you to write a method

```
private ArrayList<String> extractColumn(String filename, int columnIndex)
```

that opens the CSV file whose name is given by `filename`, finds the column at the given index (0 is the first column, 1 is the second column, etc.), then returns an `ArrayList<String>` holding the contents of that column. If the input file does not exist, your method should return `null` as a sentinel.

For example, if the above CSV file was stored in the file `food-origins.csv`, then calling `extractColumn("food-origins.csv", 1)` would return the following `ArrayList`:

`Origin`	`Middle East`	`India, Portugal`	`West Africa`	`Korea`	`Iran, Azerbaijan`	`North Africa`

Notice that none of the entries here are quoted – the quotes are just a structural part of the CSV file and not actually a part of the cell contents.

In implementing the `extractColumn` method, you should assume the following:

- All input files (assuming they exist) are well-formed CSV files.
- The specified column index will always be in bounds.
- Each row in any input file has the same number of columns.
- Any field may be quoted, not just fields containing a comma.

Before you jump into `extractColumn`, we *strongly* recommend that you begin by writing a method

```
private ArrayList<String> fieldsIn(String line)
```

that accepts a line from a CSV file and returns an `ArrayList<String>` containing all the fields from a given line (with any quotation marks stripped out, of course). Once you have this method working, it's not much extra work to implement `extractColumn`. The `indexOf` method might be very useful here.

(Optional) Part Five: Extensions!

Do you have any cool ideas for how to make these programs more interesting? If so, please feel free to submit your assignment with extensions! Here are a few suggestions of what you might work on, though this is by no means an exhaustive list:

- *Improve the syllable-counting heuristic.* Our heuristic gets the wrong answers in a few predicable places. Can you improve it to be more accurate?
- *Count syllables in other languages*. Do you have any ideas about how to implement a syllable counter for a different language like French, Chinese, or Arabic? If so, we'd love to see it!
- *Implement subtraction, multiplication, or division*. We asked you to implement addition in Part Two of this assignment, but there are lots of other operations you could support as well. Try teaching the computer how to subtract or multiply. For a real challenge, try teaching the computer long division. ☺
- *Find all losing license plates*. We mentioned earlier that some three-letter combinations, such as QQQ, can't be expanded into an English word. What other letter combinations have this property? Can you find all of them?
- *Summarize license plate answers succinctly*. Once you start playing around with your solution to the license plate game, you'll probably find that there are a *lot* of matches for many common letter combinations. See if you can find a way to intelligently summarize those matches.
- *Explore an interesting data set*. Now that you have the ability to parse CSV files, you can import data from a bunch of different sources. Find an interesting CSV file (the World Bank and the US Government's www.data.gov are good sources) and write a program that uses that information to tell us something interesting.
- *Build a complete table representation*. Rather than just extracting one column from a CSV file, see if you can instead store the entire table in memory.

As always, if you're submitting extended versions of the programs, please include both the original version and the extended version in your submission. For example, if you wanted to extend the syllable counting program, please submit two files – the base `SyllableCounting.java` and your own custom `ExtendedSyllableCounting.java`.

The Importance of Testing

It's really, *really* important to test your assignment before submitting. The methods you're writing need to work in a lot of different cases, and the only way to be confident that your programs and methods work correctly is to test them and make sure they behave as expected. Before you submit this assignment, be sure to test each of your programs on a variety of inputs. At the very least, we expect that you'll at least test your programs on the sample inputs in this handout. You should invent some of your own test code, though.

Good Luck!