Overview

In this assignment, you'll practice working with strings and with lists in two contexts. First, you'll write a series of `static` methods that interact with Strings, and then with objects that obey the `List` interface. Then, you'll extend the `ArrayList` class, adding new instance methods to a subclass of `ArrayList`.

We've provided a large set of unit tests to help with automated testing, though you might also want to write a driver class with a `main` method for interactive testing. The Gradescope autograder includes a few more tests, but they exist primarily to verify you're not gaming the autograder. If your code can pass the tests we've provided, it is likely correct.

Goals

- Practice working with `Strings`.
- Practice interacting with the `List` abstraction.
- Practice writing `static` methods.
- Practice writing instance methods.
- Translate written descriptions of behavior into code.
- Test code using unit tests.

Downloading and importing the starter code

As in a previous assignment, download and decompress the provided archive file containing the starter code. Then import it into Code in the same way; you should end up with a `working-with-strings-and-lists-student` project.

What to do

There are three files that you need to open and complete.

First, open `src/string/exercises/StringExercises.java`. This file contains a set of `static` methods and documentation describing them. Implement each method as described.

Be sure to read over the String API. There are methods in there you will want to use. One of the `String.indexOf`s and the two `String.substring` methods will make your life easier here – they let you "slice" strings the same way you do in Python, through a method call instead of special syntax. You shouldn't need to write *any* loops to complete this file.

Finally, note you can (and should!) use the `String.split` method as follows to get an array of the words: `String[] words = s.split("\\s+");`. (You can treat this as "magic" – it's a use of "regular expressions", something we haven't covered yet.)

Next, open `src/list/exercises/ListExercises.java`. This file contains a set of `static` methods and documentation describing them. Implement each method as described. Some hints:

- Lists are a lot like arrays, as we'll see in lecture soon. CICS 160 and/or AP CS may have familiarized you with them, but to review: you'll be using `get()`, `set()`, `add()`, `size()` and the like instead of the array index operators `[]` or attribute `.length`.
- Read the List API! There are methods in there you will want to use.
- Remember to consider the cases of empty lists and empty strings.
- For `split`, you'll want to use `String.split`.
- For `allCapitalizedWords`, you may find `Character.isUpperCase` helpful.
- For `insertInOrder`, you can use `String.compareTo` to determine which string comes first. If you have two strings `s1` and `s2`, use `s1.compareTo(s2) < 0` to check if `s1` "comes before" `s2`.
- If a list cannot be modified, then you'll have to create a new list to work on. Use `ArrayList` when you need a new `List`. For example, `List<String> list = new ArrayList<String>();` creates a new, empty `List` of `Strings`. `List<String> anotherList = new ArrayList<String>(aThirdList);` creates a new `List` of `Strings` called `anotherList` that's a copy of `aThirdList`.

Finally, open `src/list/exercises/ExtendedArrayList.java`. This file describes a class that *extends* (that is, "makes a subclass of") `ArrayList`, adding several instance methods. Implement each method as described. Remember, these methods are *instance* methods, and your code is a subclass. They are operating on the current instance (`this`) of the `ExtendedArrayList`. You can call methods like `size()` directly, and they'll operate on the current (`this`) list. Or write, for example, `this.size()` if it helps you to think about it that way. `this` is just like `self` in Python!

We will not test your code on inputs that are not described by the documentation. For example, all of the list and string parameters are described as "non-null", so you don't need to (and thus shouldn't) write code to handle the case of a null value being supplied as one of these parameters.

We've commented out the "timeout" code at the top of each test class, so if you want to use the debugger you won't have any obstacles in your way. On the other hand, this means that if you see a test get "stuck," it's probably because you have an infinite loop in the code it's exercising. Uncomment these lines to enable the timeouts if you think this is happening.

Again, what not to do

Many of the `static` methods are standalone "functions," with well-defined behavior that depends only upon their input. This makes gaming the autograder tempting. Don't do it.

What do I mean by gaming the autograder? Since we are still giving you the tests, you may be tempted to write code that is not general, but that does pass the tests. Put another way, you may be tempted to hard-code the "correct" answers to the tests (that won't work on other inputs!) into your methods like so:

```
public static int findMarc(String string) {
    if (string.equals("")) {
        return -1;
```

```
    } else if (string.equals("Marc")) {
        return 0;
    } else if (string.equals("Marc ")) {
        return 0;
    } else if (string.equals("  Marc  ")) {
        return 2;
    } else {
        return -1;
    }
}
```

Do not take this approach, or anything similar. We expect you to write *general* solutions to the assignments, not just attempt to pass the tests.

If you are having trouble understanding the assignment or need help, please ask! But don't game the autograder.

Another reminder about AI/LLMs

This is another assignment that an AI assistant will find trivial. Again, a friendly reminder that you're not going to build up your knowledge of Java, nor your brain's programming-problem-skillset, if you rely on code generators. And for the quizzes and exams (and for many more complicated real-life problems), those systems might not be there or be able to help you. You gotta put in the work to build up your own neural net!

Submitting the assignment

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, follow the same steps as from Assignment 01 to produce a `.zip` file, and upload it to Gradescope. Note that if you want things to upload faster, you can use an external program to zip only the `src/` directory by expanding the project; that's all this autograder requires.

Remember, you can resubmit the assignment as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does.