Assignment 07: Language

CS101 - Intro To Computer Science

Spring 2024

In this assignment, you will implement two classes that utilize the same interface, which is provided to you. Several additional class definitions will also be needed to solve the assignment.

The files for this assignment are in a zip file called `07_language.zip`. There are four groups of files in the folder

- `OrderedThing.java`: the provided interface.
- `Main.java`: an example main function to run your implementation against.
- `LanguageExerciseTest.java`: unit tests - make sure you pass them all before submitting your assignment.
- `.jar` files in the `lib` folder: libraries to enable the use of the unit tests.

1. Project Setup

The zip file contains a folder called `07_languages`, which in turn contains a VS Code project. Extract the folder from the zip file and open it with VS Code to begin implementing and running your assignment. Put all of your classes in the `src/edu/nyu/cs/assignment7` folder (the same as the `SequentiallyOrderedThing.java` file).

In particular, you need to implement the following classes:

- `OrderedThing`
- `Character`
- `Word`
- `Sentence`

2. Language

Sentences contain words. Words contain characters. The order of these words and characters is important for many human languages.

If you were trying to model human written language, you might decide to create a `Sentence` class to represent sentences, a `Word` class to represent words, and a `Character` class to represent characters.

- Since sentences contain words, any `Sentence` object has to encapsulate a list of `Word` objects.
- Since words contain `Character` objects, any `Word` object has to encapsulate a list of `Character` objects.
- Since both `Sentence` and `Word` objects contain sequentially ordered lists of things, they both implement the same `SequentiallyOrdered` interface to guarantee consistency of behavior.
- Since both `Word` and `Character` objects can be stored in ordered lists, they both inherit from a common `OrderedThing` class that may contain any attributes shared by all ordered things.

3. SequentiallyOrdered Interface

The interface code is given to you. The `Sentence` and `Word` classes that you will create must implement this interface.

4. OrderedThing Class

You will need to create an `OrderedThing` class that represents the base class for the `Character` and `Word` classes. All ordered things share the common attribute of position.

The `OrderedThing` class should have the following attribute:

- `position`: stores the position of the `OrderedThing` in a higher-level grouping in which it is being used. It is of type `int`.

The `OrderedThing` class should have the following methods:

- `getPosition()`: returns the `int` value of the `position` attribute.
- `setPosition(int p)`: sets the `int` value of the `position` attribute.

5. Character Class

You will need to create a `Character` class that represents a single character of text.

- `Character` extends `OrderedThing` because each `Character` object is stored in an ordered `ArrayList` of `Character` objects in a `Word` object.

Note: a class named `Character` already exists in the Java API *java.lang* package, so your class with the same name hides that one. If you want to refer to that API class (which you should not need to), you can need to reference it by its full package and class name, such as *java.lang.Character* in your code.

The `Character` class should have the following instance attribute:

- `letter`: stores a single character of text. It is of type `char`.

The `Character` constructor should take two parameters:

- a `char` parameter that represents the `letter` attribute and assigns the relevant instance attribute accordingly.

- an `int` parameter that represents the `position` attribute of the `Character` in `Word`.

The `Character` class should have the following instance methods:

- `toString()`: returns the `String` representation of the `Character` for output.
- `equals(Object other)`: compares two `Character` objects for equality. It checks if the provided object `other` is an instance of `Character`, then compares the `letter` and `position` attributes of both `Character` objects. It returns a `boolean`.

6. Word Class

You will need to create a `Word` class that represents words in a language.

- `Word` implements the `SequentiallyOrdered` interface because a word is a sequence of characters.
- `Word` extends `OrderedThing` because each `Word` object is stored in an ordered `ArrayList` of `Word` objects in a `Sequence` object.

The `Word` class should have the following instance attributes:

- `wordAL`: stores the `Word`'s character sequence as `Character` objects. It is of type `ArrayList<Character>`.
- `position`: stores the position of the `Word` in a `Sentence` in which it is being used (with the first `Word` in a `Sentence` as position 0). It is of type `int`.

The `Word` constructor should take two parameters:

- a `String` parameter and breaks it down into individual characters, creating `Character` objects for each character and adding them to the `wordAL`.
- an `int` parameter that represents the `position` attribute and assigns the relevant instance attribute accordingly.

The `Word` class should have the following instance methods:

- `getFirst()`: returns the first `Character` object of the `Word`.
- `getLast()`: returns the last `Character` object of the `Word`.
- `getSequence()`: returns an `ArrayList` containing all the `Character` objects in the `Word`.
- `getPosition()`: returns the `position` attribute of the `Word`.
- `toString()`: returns the `String` representation of the `Word` for output.
- `equals(Object other)`: compares two `Word` objects for equality. It checks if the provided object `other` is an instance of `Word`, then compares the `wordAL` and `position` attributes of both `Word` objects. It returns a `boolean`.

Hint: Based on the description of this class above, it should be clear to you how `Word` implements the `SequentiallyOrdered` interface. This interface requires that the `getFirst()` and `getLast()` methods return an `OrderedThing`. If not, here is a hint: a child class can be considered an instance of its parent class. This principle is known as polymorphism. Consequently, because a `Character` object extends `OrderedThing`, it can also be considered an instance of the `OrderedThing` class.

7. Sentence Class

You will need to create a `Sentence` class that represents sentences in a language.

- `Sentence` implements the `SequentiallyOrdered` interface because a sentence is a sequence of words.

The `Sentence` class should have the following instance attribute:

- `sentenceAL`: stores the `Sentence`'s word sequence as `Word` objects. It is of type `ArrayList<Word>`. This relationship between the `Sentence` and `Word` classes is called composition because a `Sentence` is composed of `Word` objects.

Note: there is NOT an inheritance relationship between `Sentence` and `Word`.

The `Sentence` constructor should take one parameter:

- a `String` parameter that represents the `Sentence` and splits the sentence into words and creates a `Word` object for each word, adding them to `sentenceAL`. You can split the `String` into words by using the `String` `split()` method in the following way:

```
// split by any non-alphanumeric character
String[] words = s.split("[^\\w']+");
```

Notice that the `split()` method gives you an array of `String` objects, and you need to go through that array, creating `Word` objects and adding them to `sentenceAL`.

The `Sentence` class should have the following instance methods:

- `getFirst()`: returns the first `Word` of the `Sentence`.
- `getLast()`: returns the last `Word` of the `Sentence`.
- `getSequence()`: returns an ArrayList containing all the `Word` objects in the `Sentence`.
- `toString()`: returns the `String` representation of the `Sentence` for output.
- `equals(Object other)`: compares two `Sentence` objects for equality. It checks if the provided object `other` is an instance of `Sentence`, then compares the `SentenceAL` attribute of both `Sentence` objects. It returns a `boolean`.

Hint: Based on the description of this class above, it should be clear to you how `Sentence` implements the `SequentiallyOrdered` interface. This interface requires that the `getFirst()` and `getLast()` methods return an `OrderedThing`. If not, here is a hint: a child class can be considered an instance of its parent class. This principle is known as polymorphism. Consequently, because a `Word` object extends `OrderedThing`, it can also be considered an instance of the `OrderedThing` class.

8. Submission

Submit the following files:

- `OrderedThing.java`
- `Character.java`
- `Word.java`
- `Sentence.java`