CS 349 Task 7: Flyweight Pattern

Description

The goal of this task is to investigate the creational Flyweight pattern as a way to minimize redundancy in a data representation. The premise is a simplistic word processor that can primarily add and remove words and assign basic font styles to them. The flyweight aspect eliminates the need to store more than one instance of each unique word, no matter how many times it appears in the text. In effect, this approach functions as a collection of Singletons, one per unique word, with garbage collection.

Specifications

As in Task 6, most of the specifications are presented in the Javadoc. Read them carefully. You do not need to implement class `Test` or any `toString()` methods. Java methods like `compareTo()`, `equals()`, and `hashCode()` are your choice. Unlike in previous tasks, you have access to my executable solution (but not source code) to compare results.

The top-down architectural overview is as follows. The details were covered in depth in Lecture 44 from 24 November.

- Class `Text` defines a passage of text as a collection of `LinkStyled` references to the `WordPool` of unique words.
- Class `LinkStyled` contains a `Style` and a `Link` to the corresponding `Word` entry in the `WordPool`.
- Class `Style` defines the three basic formatting options that can apply to any word: bold, italic, and underline.
- Class `Link` defines a pointer to a `Word` entry in the `WordPool`.
- Class `Word` defines a word as an unbroken character sequence consisting of lowercase alphabetic characters only.
- Class `WordPool` is a mapping of `Link`s to `Word`s such that a passage of text containing only `Link`s (in `LinkStyled` objects) can be reconstructed by looking up their corresponding words and optionally applying their styles as HTML tags.

Deliverables

Submit all your source files in a zip file. Include a plain-text `readme.txt` that indicates what does not work. If you believe everything works, indicate so.

Include error checking where appropriate, as especially where indicated in the Javadoc. You may add any code to facilitate your design, but you must not deviate from the specifications. It is not necessary to comment your code.

Code that does not compile will not be graded.