Assignment 8 Design Overview

Our main goal for this assignment was to follow through with one of the main concepts from our course, “One class, one responsibility”. We felt that there are three core, overarching responsibilities for this project. The program must process data in a csv file, use a template to produce personalized letters and emails, and manage these tasks using a command line interpreter. Each of these pieces can be further broken down into smaller parts.

CSV Processor

While we are using a csv file for this assignment, there are other file types out there, and it is a good idea to make our programs as general as possible to allow for future extensions of functionality. However, we also don’t need to make our program work with every type of file right away. We felt a good middle point for generality was tabular data in text based files. Thus the interface for this portion of the program is called `TableDataProcessor` and has methods `processRecord` and `processAll`. It is called a processor because the actual reading should be done by a different class (i.e. a `BufferedReader`). Tabular data can come in many forms, so the abstract class for the interface only establishes the basic fields that child classes should have, a `BufferedReader` and an array of strings to hold the field names. For the purposes of this assignment there is only one subclass, `CsvProcessor`. This class provides output in the form of a `Record`, which represents a single record in a table.

Template Processor

A template can be nearly anything, so the `TemplateProcessor` interface is a good level of generality, with methods `writeTemplate` and `writeMany`. A `TemplateProcessor` simply takes in a `Record`, and outputs a processed template. The abstract class in this hierarchy is what defines the specific templates that we are using, with placeholders that look like: [[field_name]]. Since our letter and email templates are only different in classification, the abstract class contains all the functionality needed for processing templates. The subclasses `EmailTemplateProcessor` and `LetterTemplateProcessor` simply provide a string that classifies the output files as an “email” template or a “letter” template.

Command Line Processor

The command line processor is the part of the program that orchestrates everything and allows the program to be run as a command line tool. At a high level, the command line processor needs to validate the given command line arguments, store the information given, and use the information with the above two tools in order to provide the expected output. The `main` method takes care of the last task. At the time of writing the `CommandLineProcessor` class takes care of the first two. The idea of “One class, one responsibility” would suggest these two tasks be split into seperate class hierarchies, however the reality of our time limit, and other responsibilities, meant we were not able to make that happen. Ultimately, we felt it was better to have a working product than an unfinished one that looked good on paper.