Aidan Duffy

MET CS 665

Boston University

Assignment 3, Task 1: Description

Flexibility

In terms of flexibility, it is my goal to ensure that this program is as flexible as possible. Therefore, the program will utilize a adapter design pattern. The adapter interface will be the new API, which will send its commands to the old API. All commands will be issued by a Company object, which will be a singleton. The Company will contain the "database" of all of the customer's and their associated data, in this case their emails. Any changes to customer data can be handled in the Customer class, and any API changes can be ported to the obfuscated old API until full functionality of the new API has been achieved by the company's developers.

Simplicity and Understandability

I wanted to ensure that the code was modular, was properly marked, well named, and commented so that it was not cluttered, easy for developers to read, and simple enough to understand. All of the classes will be generalized enough so that new objects can easily be created. All API information will be separated into their own package, the Customer and Company classes will also be partitioned off so those can easily be found and edited, if need be.

Duplication Avoidance

Given that all of the major objects will have entirely separate classes, and they will have little to no overlap, then there will be little to no opportunity for duplications.

Design Patterns

I utilized the adapter and singleton design patterns as they made the most logical sense. Any instance of a company when the program runs should only exist once, hence the singleton to avoid duplications and any confusion. As for the adapter, this will ensure the system is forced to read run everything through the new API, though the operations will still be handled by the old API. This is especially helpful because developers can make changes piece by piece, ensuring everything is fully functional while the new system is being full designed. Additionally, for the general email generation aspect, I utilized much of the same code as I used in assignment 3, so I had a factory and singleton design there, although I only needed 2 types: accepted and rejected. When a company creates a new message request,it will sendEmail(), learn the context of the message by entering the factory and calling getEmail() on all of the subclasses in the factory, and populate the contents of an Email object.

General Overview

Currently, the program operates from a simple adapter and singleton design. A single company exists as well as a database of their customers. The Company object will interface directly with the new API, the CustomerDataManagementSystem, which will redirect all method calls to the unexposed old API, CustomerDataOld. For email generation, the program operates from a simple factory and singleton design.

Running the Program

Simply run the relevant maven commands on the Main.java in the edu.bu.met.cs665 package.