DRY CLEANERS

CSCI - 2400

## Overview:

The Dry Cleaners program is a program that will simulate the workings of a dry cleaning business from both the employee and customer side of the interaction. The employee roles of the program are manager, cleaner, and sorter make up the workforce in the program and are the roles that run most of the program. The other two roles are the customer and the admin which supply all of the information that's needed for the program to operate. There are also several interfaces and an abstract class that were used to build the structures of the previously stated roles so all of them would have a solid blueprint of how to be constructed. The only file that is imported and used in the program is the standard java.util.\* package that has all of the essential tools needed to create the program in it.

## Use Case Analysis:

The use cases of this program are pretty straight forward because the names of the main five classes correlate with the user that will be operating it. The Admin class will be used by the administrator to add any needed roles to the program. Any customers will be able to use the Customer class to manage their items and interact monetarily with the business. The Manager class will be operated by the manager to hire, fire, edit pay, and other tasks that a manager of a business can do. Employes can access their respective classes of Cleaner or Sorter to do their jobs and keep the business running.

## Data Design:

The data in this project will be mainly kept in several array lists that will be accessed and used to alter data when requested. The data in these various array lists will be objects of the Sorter, Cleaner, and Customer class. The array lists will be held in the DryCleanerDriver class under main where the whole program will be operated and run from.

## Classes:

**Person**:

This class is an interface that every role class will implement in their structure. The only method currently in this class is the menu() method which every role class will need for the users to be able to interact with the program.

**Cleaning**:

This class is an interface for the cleaner employee class and it has basic methods that are needed to clean an item such as addSoap() and clean(). The Cleaner class will use these as the basic foundation of its structure and build off of it.

**Sorting**:

The sorting class is similar to the cleaning class with it also being an interface for its respective role class. The method in the interface is currently just sort(), but as the program gets more complex more methods will be added

**Employee**:

The employee class is an abstract class that will be inherited by all of the working roles in this program. The class has a boolean variable named “working” that will switch between true and false depending on if the worker is clocked in or out. This will be determined by the clockIn() and clockOut() methods which already have code in them. The viewPay() method is abstract and will return a double specific to the class it is inherited by.

**Admin**:

The admin class is one of the role classes and has the most editing power of all the classes. In this class the administrator can add any of the role classes to their respective array lists though the methods addCustomer(), addCleaner(), etc. admin will also implement the person interface so it will have a menu.

**Customer**:

The customer class is where most of the data used in the program will come from. The customers will be constructed of a name and their items with an id being assigned to them after they are created. The customer will be able to clean(), check() their items, pay() the cost, and leave a tip() if they so choose. The class will also contain various getter methods to ensure encapsulation of the data. Also since the customer class is a role it will implement the person interface and have to have a menu().

**Manager**:

In the manager class the manager will be able to listEmployees() and manage them accordingly. This will be done though the hire() and fire() classes which will either add an employee to the desired role or remove a specific employee. The manager can also editPay() of the employees to help regulate the business expenses. This class will also contain various getter methods to ensure encapsulation of the data. Since the manager is a person and technically an employee the class will inherit from the abstract employee class and implement from the person interface.

**Cleaner**:

The cleaner class is where the items that the customers brought in will be cleaned. This is done using the implemented clean() and addSoap() methods in the cleaning interface to clean the items. The boolean variable soap will be used to make sure that the machine washing the items has soap before washing them. This class will have various getter methods to ensure encapsulation of the data. Also because the cleaners are employees and people the class will inherit from the abstract employee class and implement the people interface to get the remaining methods. Every cleaner just like the customers will have an id number to make retrieving data smoother.

**Sorter**:

The sorter class will be used to sort the customers items and make sure all items are accounted for. After creation the sorter employees will receive an id for easier data retrieval. This class will implement the sorting interface where it will receive the sort() method for doing the classes job. Sorter is also an employee and person so it will inherit from the abstract employee class and implement the people interface to get the remaining methods.

**DryCleaningDriver**:

This class is where all the code for actually running the program will go. It will be in the static void main method along with all the array lists, user responses, and a while loop to keep it all going until the user decides to exit the program.

## Techniques Used:

**Object-Oriented Programming** -

**Inheritance** - all of the employee classes inherit from the employee class

**Encapsulation** - all data in the classes are only accessible through methods

**Polymorphism** - the abstract class Employee gets inherited and the abstract method in it gets implemented

**Abstraction** - the program has several interfaces and one abstract class used to structure the role classes

## UI Design:

Possible implementation in the future.

## UML Design:

