Course Details:

Title: BSc (Hons) Computer Science

Code: CMWSLCSIT

Mode of study: DISTANCE (P/T)

Module: Software Architecture

Code: 6WCM2002-0901-2024

Assignment Title: Part 2: Implementation

Student Details:

Name: Shameer Naveed Cheema

ID Number: 23049802

Table of Contents:

1.0 Reflective Report on the Design and Implementation of DMS:

The design phase of any software project is always a blueprint to the implementation, thus ensuring that the system would follow the principles of modularity, scalability, and maintainability. Part 1 of the Depot Management System project invested considerable effort in the design details, which include UML diagrams, class relationships, and process flows. The designs made in this part have been a strong influence on the implementation phase and were realized through the effective use of the MVC pattern.

1.1 Design Implementation:

The system has been based on class structures and relationships developed in Part 1. For example, the classes of `Customer` and `Parcel` are the model design with attributes such as `name`, `parcelId`, `dimensions`, and `weight`. All these relationships have been translated correctly to the code in association between the classes, like an association between a customer and parcel through `parcelId`. Identified in the Part 1 as a center component, class `Manager` was implemented into the system of the controller of the business logic and acting, therefore, like the intermediary between data (model) and user interface (view).

The queue system for controlling customers, which was conceptualized in Part 1, is implemented through the `QueueOfCustomers` class encapsulating the add, remove, and process operations on customers. Similarly, the `Log` class, as designed to ensure events in the system were recorded in a systematic way, was implemented using the Singleton pattern to preserve a single globally accessible instance of the class. This implementation, therefore, reflects the design's thrust in maintaining a centralized logging mechanism that would be helpful in tracing and debugging.

1.2 Implementation using the MVC Pattern:

The design of the Depot Management System has followed the MVC pattern that models the application in three interconnected elements: Model, View, and Controller.

1. **Model:**
   The `Customer` and `Parcel` classes represent the core data structures of the system. They encapsulate the system's data and provide methods to access and manipulate this data. The `Manager` class manages these data models, ensuring consistency and performing operations like sorting parcels, calculating fees, and processing customers.

2. **View:**
   The GUI was implemented using Swing components, including tables for the display of customers and parcels, buttons for user actions, and text areas for logs. The view is fully decoupled from the business logic, which means it is dedicated to the presentation of data and capturing user input.

3. **Controller:**
   The `DepotGUI` class essentially acts as the controller, because it mediates activities between the model and view. It reacts to the actions performed by users, such as button clicks, by calling the respective methods on the `Manager` class for performing the operations. For example, for the user's "Process Customer by Name," the controller would retrieve the customer from the queue (model), process their parcel, update the tables (view), and log the event.

1.3 Reflection on the Design Impact:

Part 1 designs made sure the implementation is modular and reusable. The separation of the `Log` class as a Singleton allows centralizing logging consistently for all the components in the system. Queueing design for customers was clearly designed by UML diagrams that allowed an easy direct translation to the `QueueOfCustomers` class. Use of MVC meant a clean separation of concerns that makes it easier to test, debug, and extend.

2.0 Conclusion:

Transitioning to implementation appreciates the need to have a good design phase. Since it guides the design to the level outlined in Part 1, the Depot Management System illustrates how careful consideration in design helps create a structured, maintainable, and scalable application. Thus, the MVC pattern helped separate the system into clear layers for data, presentation, and logic, which makes it more open to future implementation. This not only validates the effort put in Part 1 but also gives it a crucial role in the success of the project.