**REFLECTIVE REPORT**

It is quite clear from the implementation of the Parcel Management System that there has been excellent use of the Model-View-Controller (MVC) architectural design pattern and the three-tier architecture. This communicates heavily the findings from Part 1 regarding the design concepts. This architecture principles followed will definitely be close to current software engineering techniques in the aspects of modularity, maintainability, adaptability, and future scalability.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **Alignment with Design Principles**

**1. Use Case Realisation**

The implementation directly addresses the system's use cases as that represented in the use case diagram of a design. Core functions like handling parcels and processing customer queues are best realized with logical flow between components.

* User Interaction: The ParcelManagementGUI class provides the graphical user interface capturing actions like adding new parcels and processing customers matching the use cases of "Customer Queue Management" and "Parcel Processing".
* Backend Logic: Supportive classes such as Manager, Worker, Log encapsulate core logic of system through which all user interaction is converted into meaningful operations.

**2. UML Mapping**

The UML Diagram stresses clearly its separation of entities and their interactions as it has been coded in the implementation:

* Entity Design: Classes such as Customer and Parcel encapsulate all their respective attributes and behaviours. For example, Customer models customer details and relates parcels which totally relate to the UML's relationship diagram.
* Flow of Control: The Manager class serves as an intermediary, coordinating the tasks across entities. It follows closely the defined relationship by UML.

**3. 3-Tier Architecture**

It applies a 3- tier structure, ensuring separation of concerns and hence also clear distinctions between the entities.

* Application Layer: The Manager class serves as the central controller, managing business rules and interactions between the UI and the data layer.
* Data Layer: Entities like Customer and Parcel, along with data structures (Queue, Map), encapsulate system data and provide storage mechanisms.
* **Implementation of MVC Pattern**

The MVC pattern is a cornerstone of the system’s design, ensuring a robust separation of responsibilities that promotes maintainability and extensibility.

1. **Model:**

* The application classes will shield the core data structures like Customer, Parcel, and Log. They separate the state and behaviour of data and they don't require the person that uses the program to be able to see them.
* Supporting additional utility classes as ParcelMap and QueueOfCustomers conceals specification for data storage and retrieval, providing encapsulation and reusability.

1. **View:**

* The ParcelManagementGUI serves as the system’s view, presenting data and receiving user inputs through graphical components such as JList, JTextArea, and JButton.
* Panels for customer and parcel views ensure logical grouping of UI elements, while methods like updateLists synchronize the displayed data with changes in the model.

1. **Controller:**

* The Manager class acts as the controller, facilitating communication between the view and the model. It interprets user actions (e.g., button clicks) and applies the corresponding changes to the model while updating the view.
* Event listeners in the GUI delegate user actions to the Manager, maintaining the view’s independence from backend logic.
* **Strengths of the Design**

1. **Modularity**  
   The separation of concerns ensures that each class focuses on a specific responsibility. For example, Log handles event recording, while Worker encapsulates parcel processing logic. This modular design simplifies debugging, testing, and future modifications.
2. **Scalability**  
   The system’s design supports future enhancements with minimal impact on existing code. For instance, integrating a database for persistent storage or expanding the UI to support advanced features (e.g., search and filtering) would require limited changes to the application and data layers.
3. **Reusability**  
   Independent components such as Log, Worker, and utility classes can be reused in other systems without modifications, reflecting the robustness of the design.
4. **Extensibility**  
   The class structure is conducive to adding new features. For example, additional methods in Manager could handle advanced fee calculations, or a reporting module could be added for system analysis.

**Potential Enhancements**

1. **Centralized Input Validation**  
   Validation logic for parcel attributes could be centralized within the Manager class. This would reduce redundancy in the GUI and ensure consistent checks across all input sources.
2. **Automated Testing Framework**  
   Introducing a testing framework like JUnit could improve the system’s reliability. Unit tests for individual components (e.g., fee calculations in Worker) and integration tests for end-to-end workflows would ensure system correctness and robustness.
3. **Dynamic Configuration**  
   Parameters like fee rates and storage capacities could be externalized into configuration files. This would enhance adaptability to changing business requirements without requiring code changes.
4. **Enhanced User Experience**

* Tooltips, input masks, and real-time error feedback in the GUI could improve usability.
* Pagination or filtering for long customer or parcel lists would ensure scalability for large datasets.

**Conclusion**

The Parcel Management System demonstrates a meticulous alignment with the design principles outlined in part 1. Its layered architecture and adherence to the MVC pattern ensure a clean, modular, and maintainable implementation. The system is well-equipped to accommodate future requirements, making it a robust solution for parcel and customer management. By addressing potential enhancements, the system could further elevate its usability, adaptability, and reliability, solidifying its role as a scalable and modern software solution.