Where does Command decouple UI from business logic in your codebase?

The Command pattern decouples the user interface from the business logic implemented inside the OrderService and other domain classes. The PosRemote never needs to know how an order is created, paid, or updated, it only knows that each button is linked to a Command object that implements execute() and undo(). This design allows the UI layer to remain the same even if the business logic changes. For example, if new payment types are introduced, only new command or receiver classes need to be added, therefore the UI code remains untouched. This separation makes the system more flexible, maintainable, and testable.

Why is adapting the legacy printer better than changing the domain or vendor class?

Adapting the legacy printer using the Adapter pattern is better because it respects both encapsulation and software ownership boundaries:

- The legacy printer (vendor.legacy.LegacyThermalPrinter) belongs to an external vendor library, so editing it would be unsafe and could break compatibility when the library updates.
- The domain classes (Order, OrderService, etc.) should not depend on vendor-specific APIs. They should only talk to a simple Printer interface.

By introducing LegacyPrinterAdapter, the adapter acts as a bridge that converts the domain's clean print(String text) call into the legacy printer's legacyPrint(byte[] payload) method. This ensures that vendor changes or encoding details are isolated inside one class, while the rest of the system remains consistent and independent.