Part 1 – Design Pattern and Class Diagram

UML Diagram

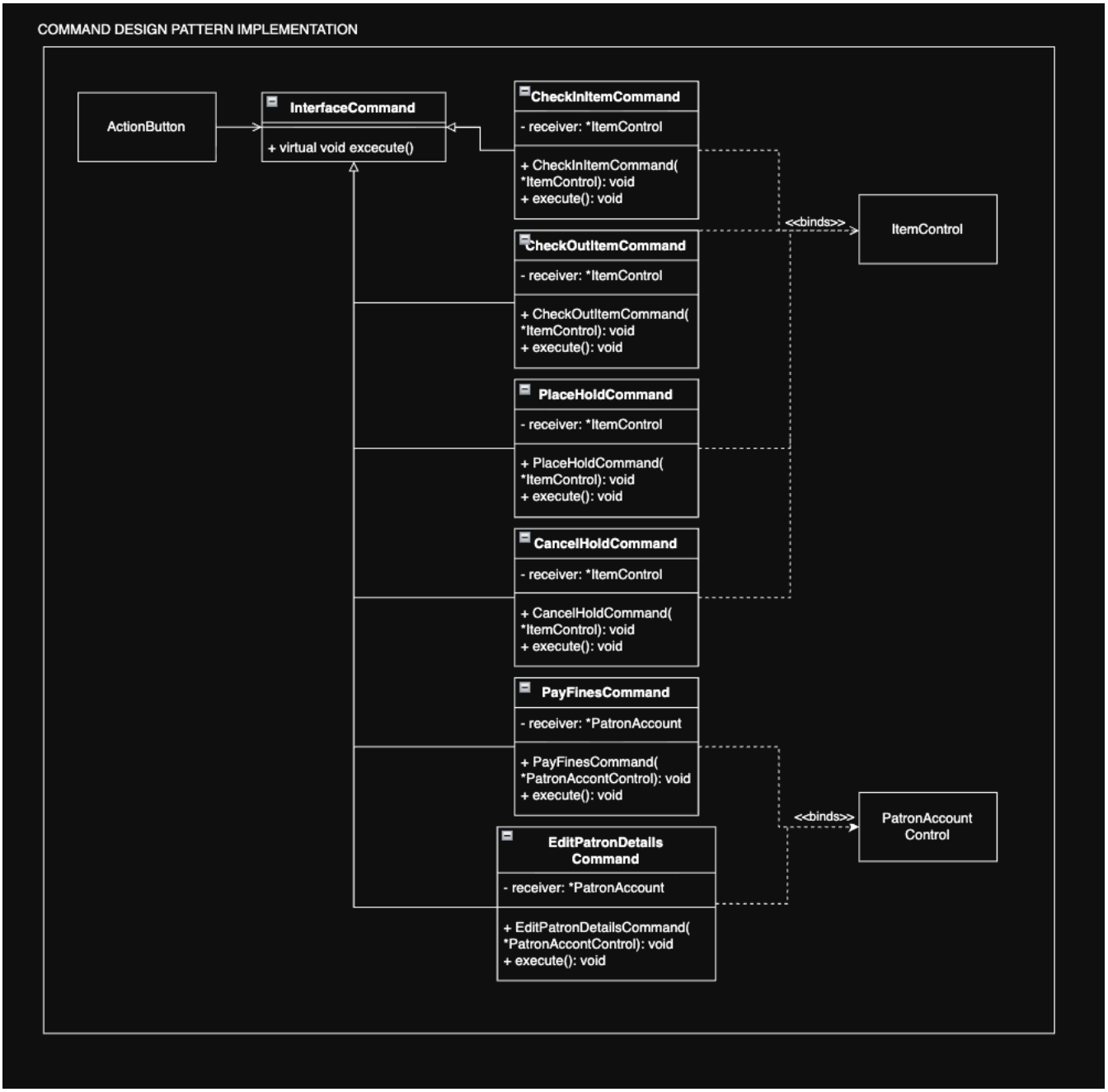

Figure 1: Design Pattern “Command” to be used in the implementation of HinLIBS

Justification

The HINLIBS library system relies heavily on user interactions with UI commands through interface elements, such as buttons and menu options, to perform core operations, including checking out items and managing patron profiles. The Command design pattern is ideally suited for this context, as it encapsulates each user request as an object, providing a clear separation between the invoker, the user interface components which trigger actions, and the receiver, the control classes which execute them.

By transforming requests into command objects, the pattern enables undoable and redoable operations, which are particularly valuable in a library management system where mistakes or accidental clicks are widespread. In a transactional system such as HINLIBS, errors like checking out the wrong book, cancelling a valid hold, or placing an incorrect one can be easily corrected through the undo functionality provided by the Command design pattern. This capability supports error recovery, enhanced user confidence, and strengthens data integrity and auditability.

Furthermore, the Command pattern establishes a uniform request-handling layer within the HINLIBS architecture. Rather than allowing the UI to invoke domain logic directly, all requests are funnelled through a consistent command interface. This approach promotes cohesion across system components, improves flexibility and maintainability, and supports advanced capabilities such as command scheduling, queuing, and logging for future scalability.

The Command design pattern addresses the need for consistent transaction management in HINLIBS by encapsulating each library operation as an independent, executable object. This structure enables features like operation queuing and scheduling, allowing commands to be stored, executed later, or retried if a transaction fails. It also supports

audit trails and logging for patron items, since each command object contains and maintains accountability for the action it represents.

Within the HINLIBS architecture, the Command design pattern fits between the user interface layer and the controller/service layer, forming the core of the request-handling process. The invoker, such as a button or menu item, stores a reference to a command and triggers it, rather than directly calling the controller. The Client creates and assigns the appropriate command objects. At the same time, the ConcreteCommand classes encapsulate request details and delegate execution to Receiver classes, such as ItemControl or PatronAccountControl, which contain the actual business logic. This structure ensures a clear separation of concerns, improves maintainability through decoupled components, and supports easy testing, as each command represents a small, self-contained unit of functionality.

The Command pattern supports maintainability by encapsulating each operation in a single, self-contained command class. This enforces the single responsibility principle, ensuring that logic for a specific action is localized rather than spread across controllers or services. Since commands are small and independent, they can be easily tested in isolation, allowing the system to remain stable even as other individual commands evolve. The loose coupling between invokers, commands, and receivers further simplifies debugging and updates.

It enhances extensibility by allowing new operations to be added without requiring modifications to existing client code. Developers can introduce new command classes or combine existing ones into macro commands to create more complete behaviours, all without altering the core framework.

Reusability is also promoted, since well-defined command objects can be reused across different parts of the system or in future projects that require similar actions or workflows.

Works Cited

“Command Design Pattern.” *GeeksforGeeks*, 21 September 2025, https://www.geeksforgeeks.org/system-design/command-pattern/. Accessed 12 November 2025.

Shvets, Alexander. “Command design pattern.” *Refactoring.Guru*, https://refactoring.guru/design-patterns/command. Accessed 12 November 2025.