Chain of Responsibility Design Pattern

Avoiding tight coupling between sender and receiver of a request and allowing multiple objects to handle a request.

Key Components:
- **Handler**: Defines an interface for handling requests and optionally passing them to the next handler.
- **Concrete Handler**: Implements the handler interface, handles requests, and may pass them to the next handler.
- **Client**: Initiates requests, unaware of the handlers' hierarchy.

Advantages:
- **Decoupling**: Separates request senders from receivers, promoting loose coupling.
- **Dynamic Handling**: Allows dynamic addition, removal, or reordering of handlers.
- **Responsibility Distribution**: Divides responsibilities among multiple handlers.

Disadvantages:
- **Unprocessed Requests**: There's a risk that requests may go unhandled if there's no suitable handler in the chain.
- **Complexity**: Managing the chain hierarchy can introduce complexity.

Examples:
- **Approval Workflows**: Handling approval requests through multiple stages.
- **Exception Handling**: Handling exceptions through a series of exception handlers.
- **Security Filters**: Authorizing and authenticating requests in a web application.

Iterator Design Pattern

Providing a uniform way to traverse collections without exposing their underlying structure or implementation.

Key Components:
- **Iterator**: Defines a common interface for iterating elements.
- **Concrete Iterator**: Implements the iterator interface for a specific collection.
- **Aggregate**: Defines an interface for creating an iterator.
- **Concrete Aggregate**: Implements the aggregate interface and provides an iterator for its elements.

Advantages:
- **Decoupling**: Separates collection traversal from its internal structure.
- **Uniform Interface**: Provides a consistent way to access elements in various collections.
- **Iteration Control**: Allows iteration control (e.g., forward, backward) without changing collection code.

Disadvantages:
- **Complexity**: Introducing iterators can make the code more complex.
- **Overhead**: May introduce overhead when creating iterator objects.
- **Not Suitable for All Collections**: May not be practical for small or simple collections.

Examples:
- **File Systems**: Iterating files and directories in a file system.
- **Menu Systems**: Iterating through menu items in user interfaces.
- **Text Processing**: Scanning words or characters in a text document.
- **Playlist Management**: Managing song playlists in music players.