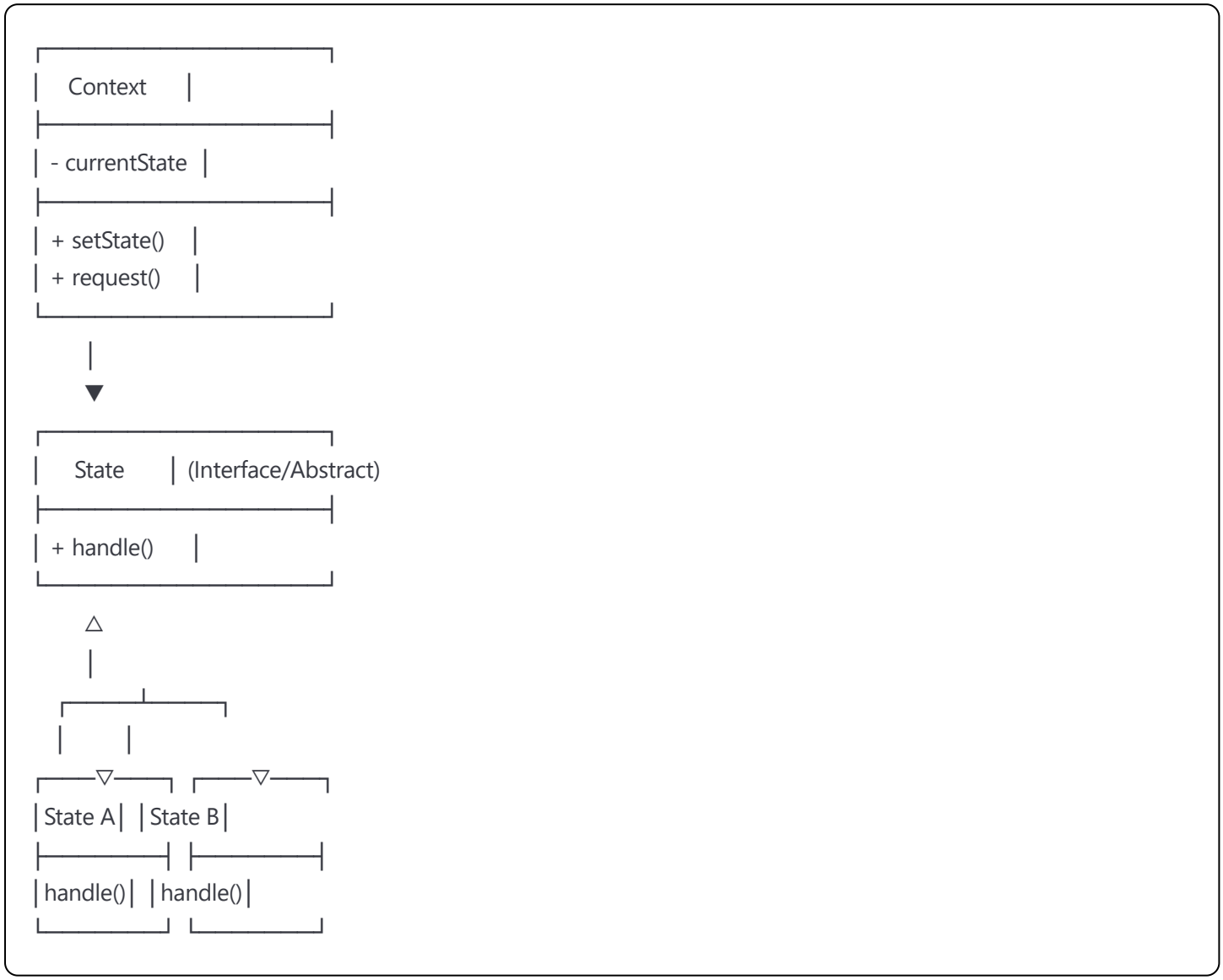
State Design Pattern

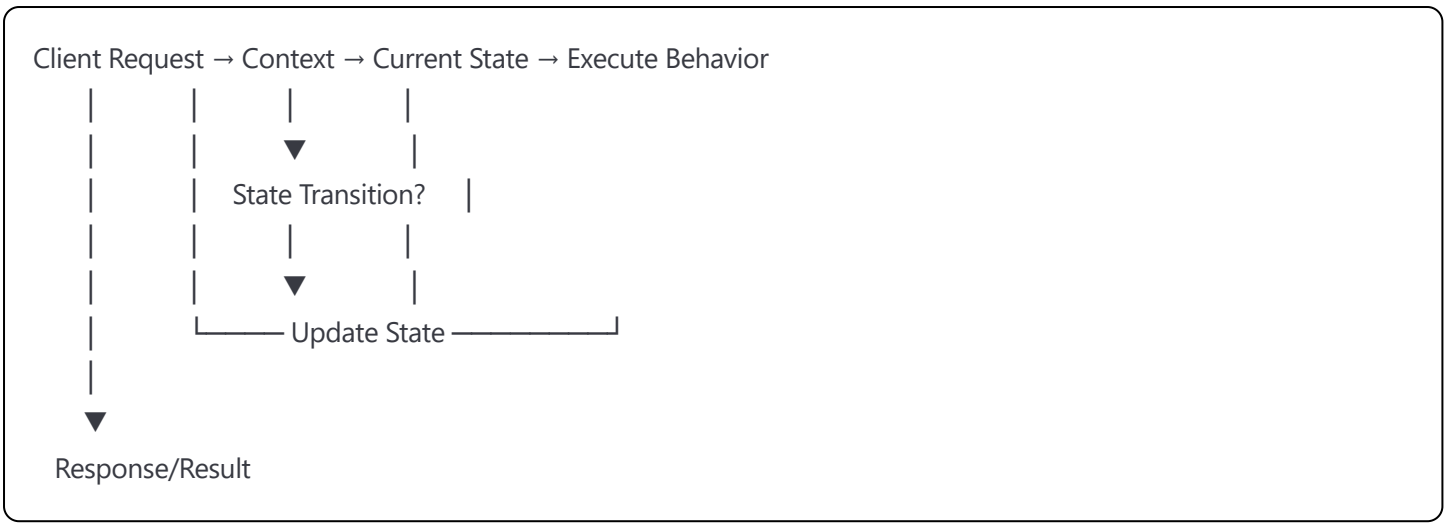
Concept

The State pattern allows an object to alter its behavior when its internal state changes. The object appears to change its class by delegating state-specific behavior to separate state objects.

UML Class Diagram



Flow Diagram



Advantages

- **Eliminates Conditional Statements**: Replaces complex if-else or switch statements
- **Single Responsibility**: Each state handles only its specific behavior
- **Open/Closed Principle**: Easy to add new states without modifying existing code
- **State Transitions**: Clear and explicit state transition logic
- **Maintainability**: State-specific code is localized and easier to maintain

Disadvantages

- **Increased Complexity**: More classes needed for simple state machines
- **Memory Overhead**: Multiple state objects may be created and maintained
- **Overkill for Simple Cases**: May be excessive for objects with few states
- **State Management**: Careful coordination needed between context and states
- **Performance Impact**: Additional method calls through state objects

Common Use Cases

- **UI Components**: Button states (enabled, disabled, pressed, hover)
- **Game Characters**: Different behaviors based on health, power-ups, modes
- **Document Editors**: Edit mode, read-only mode, review mode
- **Network Connections**: Connected, disconnected, connecting, error states
- **Vending Machines**: Idle, coin inserted, product selected, dispensing
- **Media Players**: Playing, paused, stopped, buffering states

Sequence Diagram

```
Client → Context: request()
Context → CurrentState: handle(request)
CurrentState → CurrentState: process request
alt state change needed
    CurrentState → Context: setState(newState)
    Context → Context: currentState = newState
end
CurrentState → Context: return result
Context → Client: return result
```

Key Components

1. **Context**: Maintains reference to current state and delegates requests
2. **State Interface/Abstract Class**: Defines common interface for all states
3. **Concrete States**: Implement specific behavior for each state
4. **State Transitions**: Logic for moving between different states