STATE DESIGN PATTERN

– Design Patterns –

Roxana Sultan

OVERVIEW

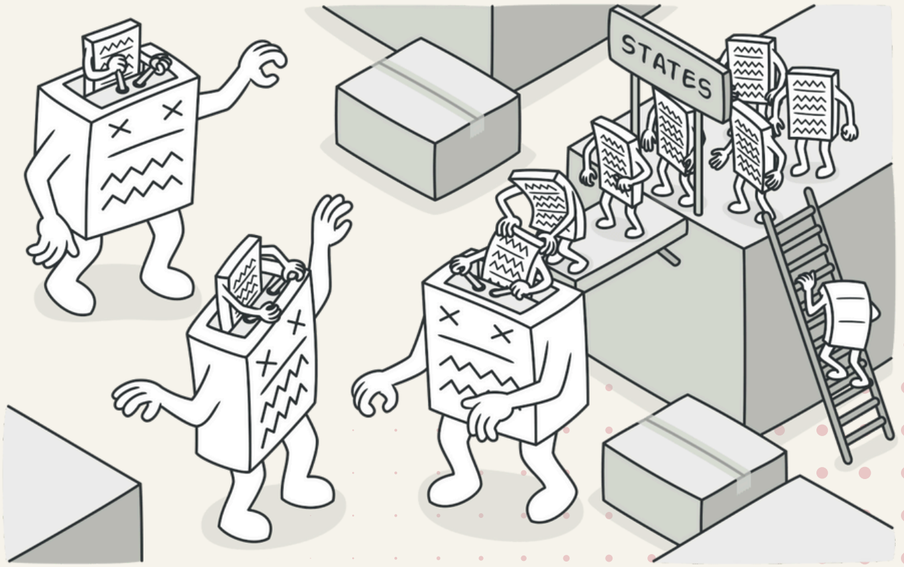
- What is a State Design Patter?
- Real-Life Example
- How to implement the State Design Pattern
- Code Example
- Advantages
- When to use State Design Pattern?

WHAT IS A STATE DESIGN PATTERN?

→ It allows an object to alter its behavior when its internal state changes.

→ It encapsulates varying behavior for the same object based on its internal state

→ Useful when an object needs to go through several states and its behavior differs for each state



REAL-LIFE EXAMPLE

States	Operations
Debit Card Not Inserted	I. You can insert the debit card 2. You cannot eject the debit card 3. You cannot enter the PIN number 4. You cannot withdraw money
Debit Card Inserted	I. You cannot insert the debit card 2. You can eject the debit card 3. You can enter the PIN number 4. You can withdraw money

HOW TO IMPLEMENT THE STATE DESIGN PATTERN

1. State interface → Defines an interface for encapsulating the behavior associated with a particular context state

2. Concrete State Classes → Implement the State interface and provide the actual behavior specific to a state

3. Context → Maintains an instance of a Concrete State subclass that defines the current state. The context delegates state-specific behavior to the current State object

4. Client Code → Modifies the state of the context

CODE EXAMPLE

ADVANTAGES

Encapsulation of State-Based Behavior : State-specific logic is encapsulated in state classes.

Easy to Add New States : Introducing new states doesn't require changing the context or other states.

Eliminates Conditional Statements : It helps to eliminate conditional statements for behavior changes based on the state.

Maintainability and Flexibility : The state pattern makes it easier to maintain and extend state-based behavior.

WHEN TO USE STATE DESIGN PATTERN?

Object Behavior Depends on Its State

When an object's behavior varies based on its internal state, the State pattern encapsulates each state's behavior in separate classes — eliminating large if/else or switch statements.

Complex State Logic

For objects with multiple states and transitions, the State pattern simplifies complexity by separating state-specific logic into distinct, manageable classes.

Maintaining Clean Code

It promotes clean, decoupled code — each state handles its own logic independently, making the system easier to maintain and modify.

Ease of Extending States and Behaviors

When new states or behaviors are needed, they can be added without changing existing code, following the Open/Closed Principle.

THANK YOU!