







**🧩 What is the State Pattern?**

The **State Pattern** is a behavioral design pattern that allows an object to alter its behavior when its **internal state** changes.

It appears as if the object has changed its class.

**Key Idea:** Move behavior for different states into separate classes and delegate state-specific behavior to the current state object.

**✅ Use Cases of the State Pattern**

| **Use Case** | **Example** |
| --- | --- |
| ✅ **Workflow engines** | Different behavior based on task status: Draft, Submitted, Approved |
| ✅ **UI Components** | Button behaves differently when Enabled vs Disabled |
| ✅ **Vending Machine / ATM** | Different behavior based on current machine state (Idle, Processing, Dispensing) |
| ✅ **Game development** | Player actions vary based on state: Idle, Jumping, Attacking |
| ✅ **TCP connection** | Open, Closed, Listening, Established, etc. |
| ✅ **Document lifecycle** | Draft → Review → Published transitions with custom behavior |

**🧰 Participants in the State Pattern**

| **Role** | **Description** |
| --- | --- |
| **Context** | The object whose behavior changes depending on its state |
| **State Interface** | Defines behavior associated with a particular state |
| **Concrete States** | Implement behavior for a specific state and can transition to other states |
| ✅ Advantages of State Pattern  | **Advantage** | **Description** | | --- | --- | | ✅ **Encapsulation of behavior** | State-specific behavior is separated in classes | | ✅ **Eliminates large if-else or switch** | Replaces complex conditionals based on state | | ✅ **Open/Closed Principle** | Add new states without modifying existing code | | ✅ **Cleaner code** | State transitions and logic are decoupled from the context |  ❌ Disadvantages of State Pattern  | **Disadvantage** | **Description** | | --- | --- | | ❌ **More classes** | Each state is a separate class, increasing code size | | ❌ **Can be overkill** | Too complex for simple state handling (better with enum) | | ❌ **Harder to debug** | Runtime state transitions might not be obvious without logging | | ❌ **Requires careful state management** | Incorrect transitions can cause hard-to-track issues |  ✅ When to Use Use State Pattern when:   * The object has **complex state-dependent behavior** * You want to **change behavior at runtime** * You want to **encapsulate transitions and behavior per state**  ✅ Summary Table  | **Feature** | **Description** | | --- | --- | | Type | Behavioral Design Pattern | | Intent | Change object behavior based on its state | | Based on | Polymorphism | | Java Analogy | Thread.State, Lifecycle.State | | Common Alternatives | switch-case, enum with strategy logic | |  |

## ✅ Java Example: Document Workflow

### 1. ****State Interface****

java

CopyEdit

public interface DocumentState {

void handleRequest(DocumentContext context);

}

### 2. ****Concrete States****

java

CopyEdit

public class DraftState implements DocumentState {

public void handleRequest(DocumentContext context) {

System.out.println("Document is in Draft state. Moving to Review...");

context.setState(new ReviewState());

}

}

public class ReviewState implements DocumentState {

public void handleRequest(DocumentContext context) {

System.out.println("Document is in Review state. Moving to Published...");

context.setState(new PublishedState());

}

}

public class PublishedState implements DocumentState {

public void handleRequest(DocumentContext context) {

System.out.println("Document is Published. No further action.");

}

}

### 3. ****Context Class****

java

CopyEdit

public class DocumentContext {

private DocumentState currentState;

public DocumentContext() {

currentState = new DraftState(); // initial state

}

public void setState(DocumentState state) {

currentState = state;

}

public void request() {

currentState.handleRequest(this);

}

}

### 4. ****Client Code****

java

CopyEdit

public class StatePatternDemo {

public static void main(String[] args) {

DocumentContext doc = new DocumentContext();

doc.request(); // Draft -> Review

doc.request(); // Review -> Published

doc.request(); // Published -> No further action

}

}