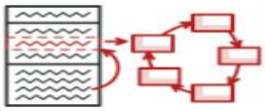


# State

Modelul **de stare** este un model de design **comportamental** . Conform definiției GoF, o stare **permite unui obiect să-și modifice comportamentul atunci când starea sa internă se schimbă** . Obiectul va părea să-și schimbe clasa.

Din definiția de mai sus se poate deduce că va exista o **clasă concretă separată pentru starea posibilă a unui obiect** . Fiecare obiect de stare concretă va avea logica să accepte sau să respingă o cerere de tranziție de stare pe baza informațiilor despre starea sa actuală și contextul transmise lui ca argumente de metodă.

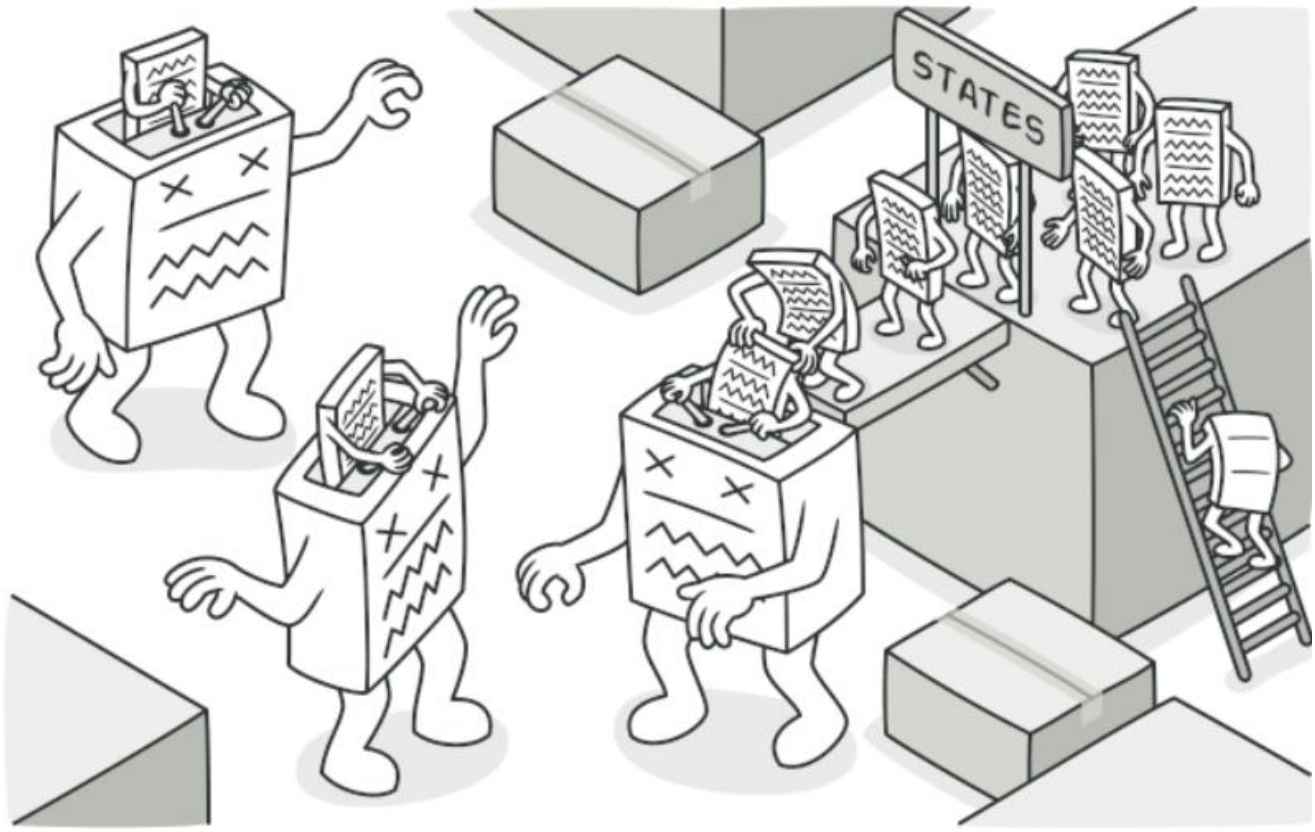
**Complexitate:** ★☆☆

**Popularitate:** ★★☆

## Intenție

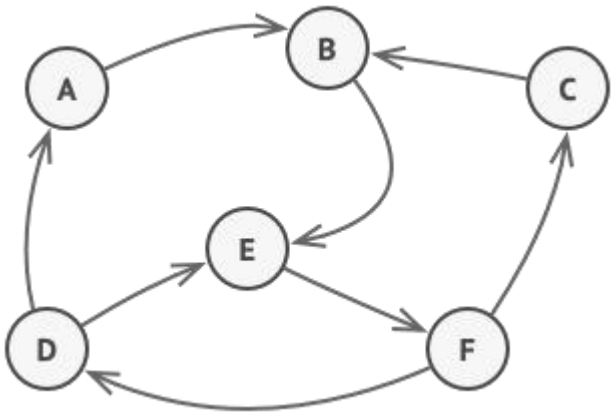
**State** este un model de design comportamental care permite unui obiect să-și schimbe comportamentul atunci când starea sa internă se schimbă.

Modelul extrage comportamentele legate de stare în clase de stare separate și forțează obiectul original să delege munca unei instanțe a acestor clase, în loc să acționeze pe cont propriu.




## ☹Problemă

*Modelul de stare este strâns legat de conceptul de mașină cu stări finite .*




Mașină cu stări finite.

*Ideea principală este că, în orice moment, există un număr finit de stări în care se poate afla un program. În orice stare unică, programul se comportă diferit, iar programul poate fi comutat de la o stare la alta instantaneu. Totuși, în funcție de starea curentă, programul*

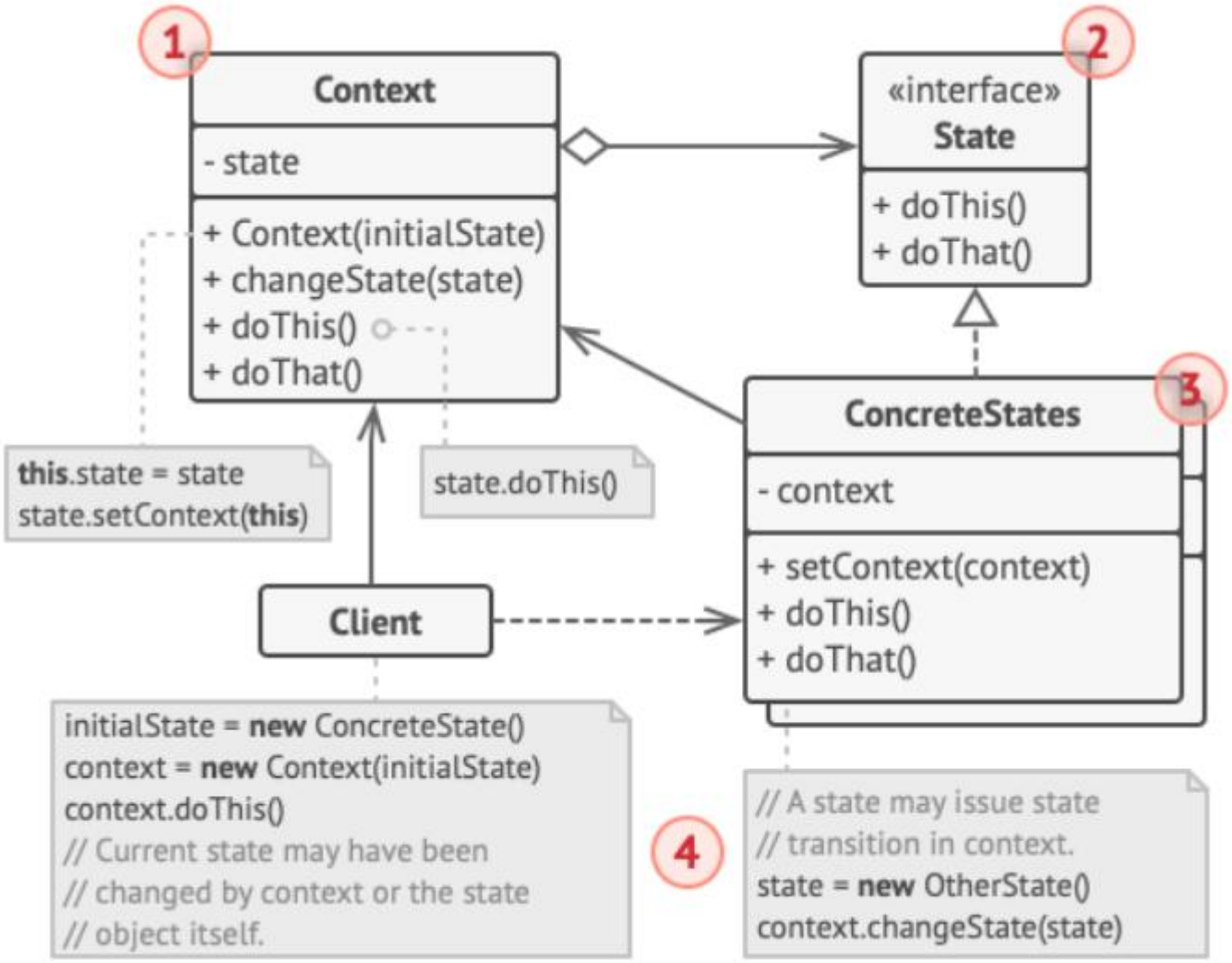
poate sau nu trece la anumite alte stări. Aceste reguli de comutare, numite *tranziții*, sunt, de asemenea, finite și predeterminate.

## Analogie din lumea reală

Butoanele și comutatoarele din smartphone se comportă diferit în funcție de starea curentă a dispozitivului:

- Când telefonul este deblocat, apăsarea butoanelor duce la executarea diferitelor funcții.
- Când telefonul este blocat, apăsarea oricărui buton duce la ecranul de deblocare.
- Când încărcarea telefonului este scăzută, apăsarea oricărui buton afișează ecranul de încărcare.

## Structura



1. **Context** stochează o referință la unul dintre obiectele de stare concrete și îi delegă toate lucrările specifice stării. Contextul comunică cu obiectul de stare prin interfața de stare. Contextul expune un setter pentru a-i transmite un nou obiect de stare.
2. Interfața **State** declară metodele specifice stării. Aceste metode ar trebui să aibă sens pentru toate stările concrete, deoarece nu vrei ca unele dintre stările tale să aibă metode inutile care nu vor fi apelate niciodată.
3. **Concrete States** oferă propriile lor implementări pentru metodele specifice state-ului. Pentru a evita duplicarea unui cod similar în mai multe state, se pot furniza clase abstracte intermediare care încapsulează un comportament comun.

   Obiectele de state pot stoca o referință la obiectul context. Prin această referință, stateul poate prelua orice informații necesare din obiectul context și poate iniția tranziții de stare.

4. Atât stările de context, cât și cele concrete pot seta următoarea stare a contextului și pot efectua tranziția reală a stării prin înlocuirea obiectului de stare legat de context.

## Aplicabilitate

**Utilizați modelul State atunci când aveți un obiect care se comportă diferit în funcție de starea sa curentă, numărul de stări este enorm și codul specific stării se modifică frecvent.**
Modelul sugerează să extrageți tot codul specific de state într-un set de clase distincte. Drept urmare, puteți adăuga stări noi sau le puteți modifica pe cele existente independent una de alta, reducând costurile de întreținere.

**Utilizați State atunci când aveți mult cod duplicat în stări și tranziții similare ale unei mașini de stări bazate pe condiții.**
Modelul State vă permite să compuneți ierarhii de clase și să reduceți duplicarea prin extragerea codului comun în clase de bază abstracte.

## Argumente pro şi contra

✓ *Principiul responsabilității unice* . Organizați codul legat de anumite stări în clase separate.

✓ *Principiul deschis/închis* . Introduceți noi state fără a modifica clasele existente sau contextul.

✓ Simplificați codul contextului eliminând condițiile voluminoase ale mașinii de stări.

✕ Aplicarea modelului poate fi exagerată dacă o mașină de stări are doar câteva stări sau se modifică rar.

## ⇄ Relații cu alte modele

- **Bridge** , **State** , **Strategy** (și într-o anumită măsură **Adapter** ) au structuri foarte asemănătoare. Într-adevăr, toate aceste modele se bazează pe compoziție, care delegă munca altor obiecte. Cu toate acestea, toate rezolvă probleme diferite.
- **State** poate fi considerat ca o extensie a **Strategy** . Ambele modele se bazează pe compoziție: ele schimbă comportamentul contextului prin delegarea unor lucrări obiectelor auxiliare. *Strategy* face aceste obiecte complet independente și inconștiente unele de altele. Cu toate acestea, *State* nu restricționează dependențele dintre stările concrete, lăsându-le să modifice starea contextului după bunul plac.

Bibliografie:


https://refactoring.guru/design-patterns/state

https://howtodoinjava.com/design-patterns/behavioral/state-design-pattern/

https://github.com/BogdanIancu/CTS2023

https://www.geeksforgeeks.org/state-design-pattern/