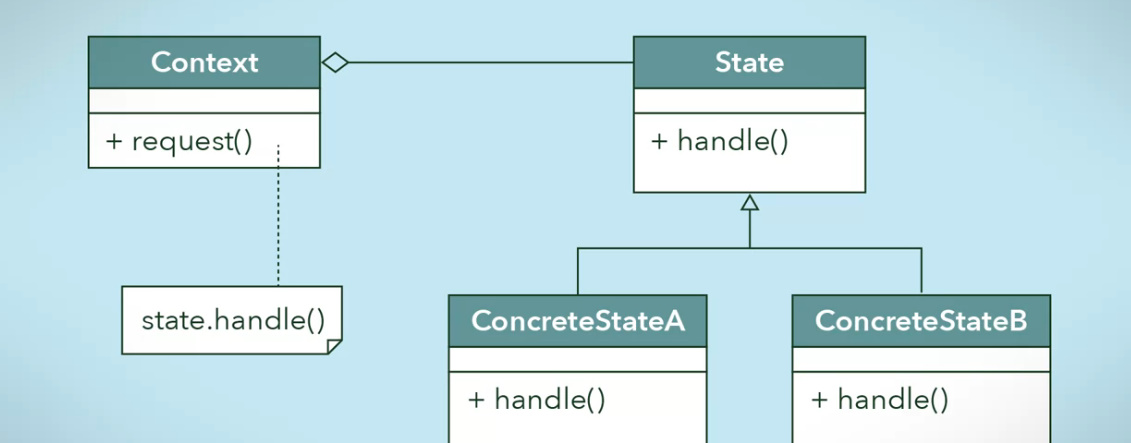
* Multe actiuni pe care le facem depind de statusul nostru curent
* De ex, daca ni se spune sa dansam, atunci o vom face diferit, in dependenta de pozitia noastra. Daca suntem in picioare, vom dansa intr-un fel, daca asezati in alt fel, daca culcati, tot in alt fel
* Unele dansuri ne vor obliga sa luam o anumita pozitie, adica status
* Exact asa in code, obiectele se pot purta intr-un mod in dependenta de statutul lor actual.
* Odata ce state e modificat si comportamentul lor poate fi diferit
* **State pattern** – se bazeaza pe faptul ca obiectele sunt constiente de state lor si in dependenta de el, se pot purta diferit
* State pattern e folosit cand vrem sa modificam behaviour la un obiect in dependenta de statusul sau la runtime
* Cam asa arata UML:

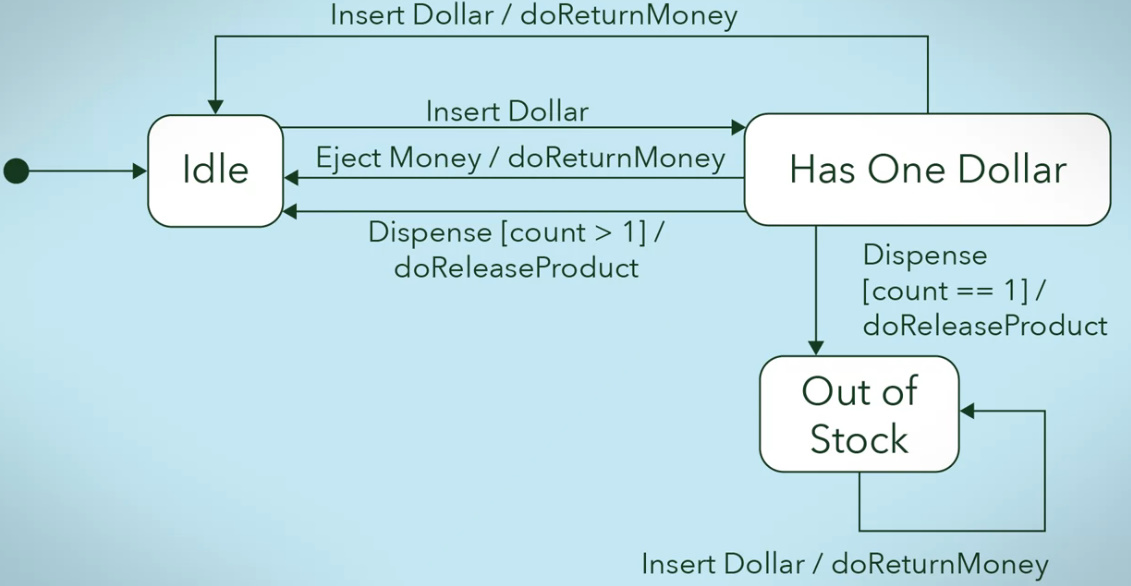


Context e clasa ce foloseste State ca field, si anume acest State field e responsabil de comportament

* **Acest pattern se bazeaza pe a defini comportamentul in state object, nu prin a folosi if si else pe baza la field si sa determine in clasa context comportamentele posibile. State pot sa se modifice si sa fie foarte multe, si clasa Context poate creste enorm,dar asa daca state objects definesc comportamentul, asa nu e o problema.**

**Exemplu**

* Sa zicem ca avem un automat ce ofera dulciuri odata ce introducem un dolar.
* In mod norma, introducem dolarul, facem alegera si primim ce am ales
* Dar mai pot fi si alte situatii, ca gen, punem dolarul, ne razgandim si apasam sa dea dolarul inapoi, sau putem cere un produs, dar el deja e out of stock.
* UML State diagram va fi asa:



Daca count==1, se returneaza produsul, dar masina intra in Out of Stock, caci count va fi 0, si ea nu va putea functiona bine, si daca se insereaza un dolar cand in out of stock state, il va intorce mereu

Despense inseamna sa dai produsul

vedem ca sunt 3 states posibile:

- Idle

- Has one dollar

- Out of Stock

Si 3 events posibile:

- Insert Dollar

- Dispense(da produsul)

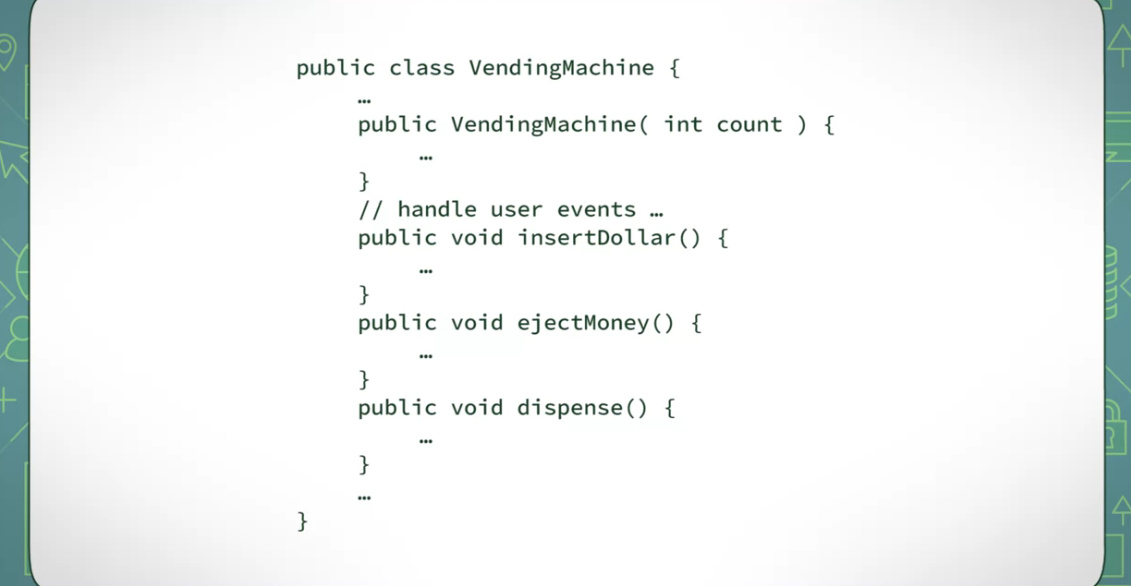
- Eject Money(da dolarul inapoi)

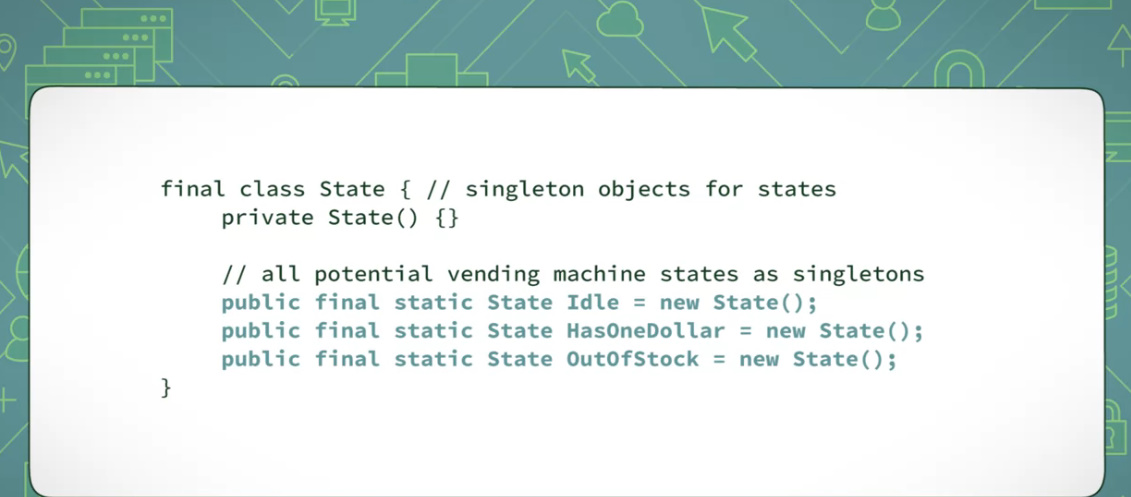
Si 2 actiuni ale masinei

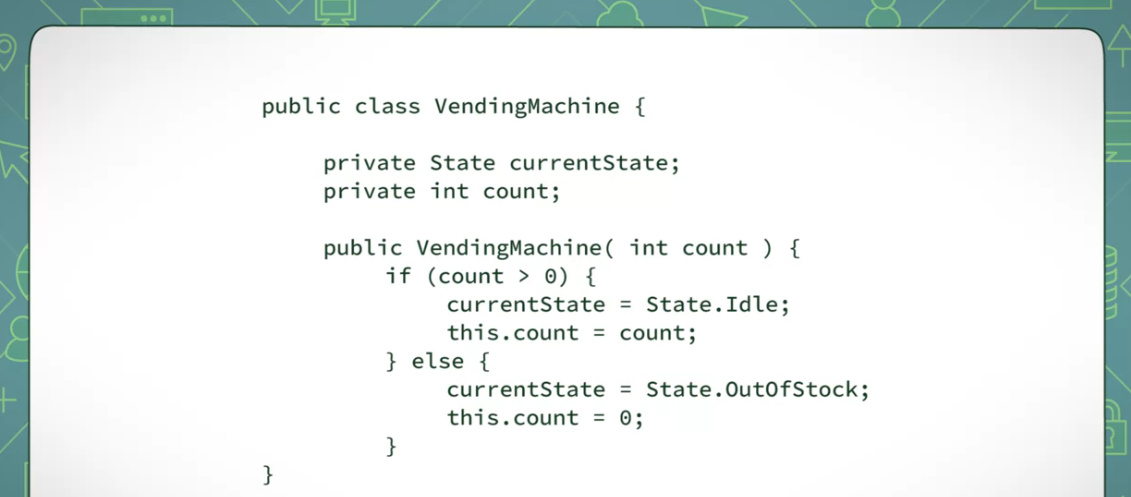
- do return money

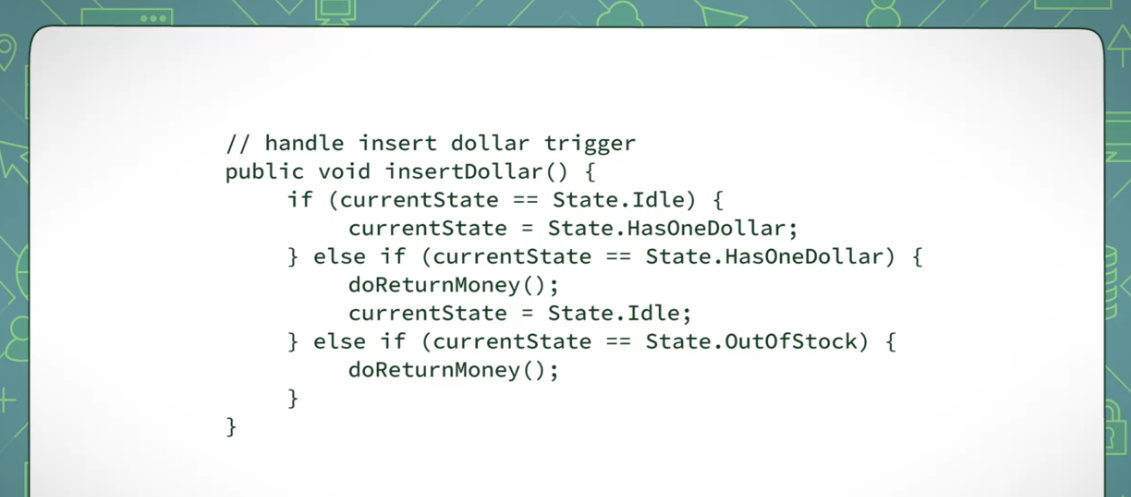
- do release product

**Cam asa am face in mod normal**



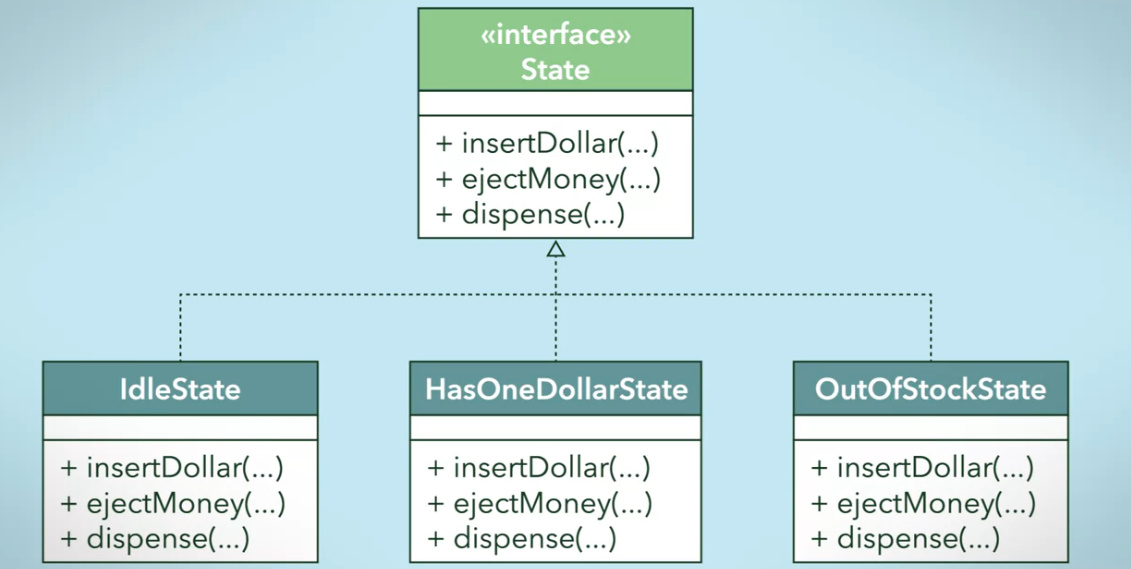


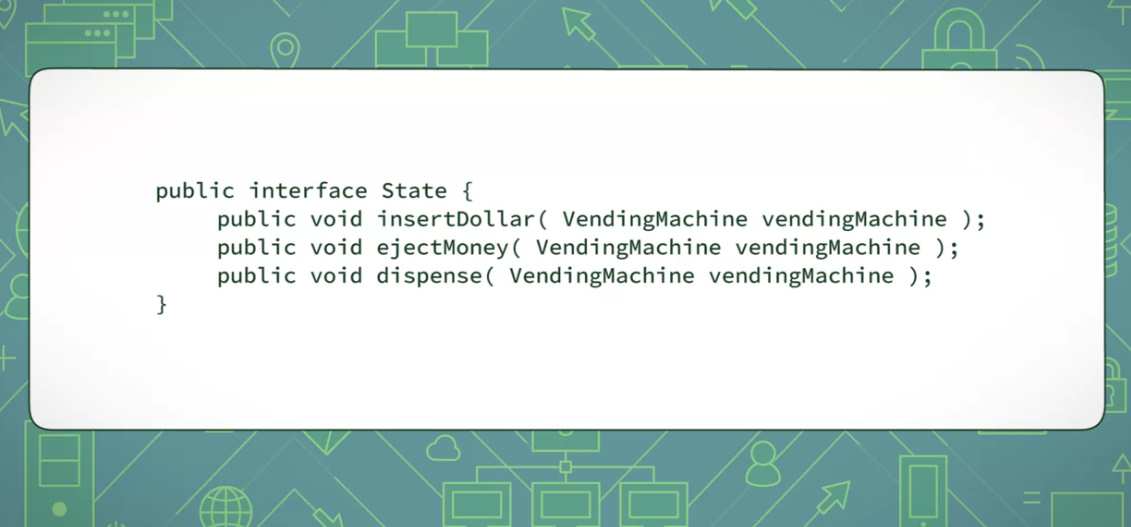


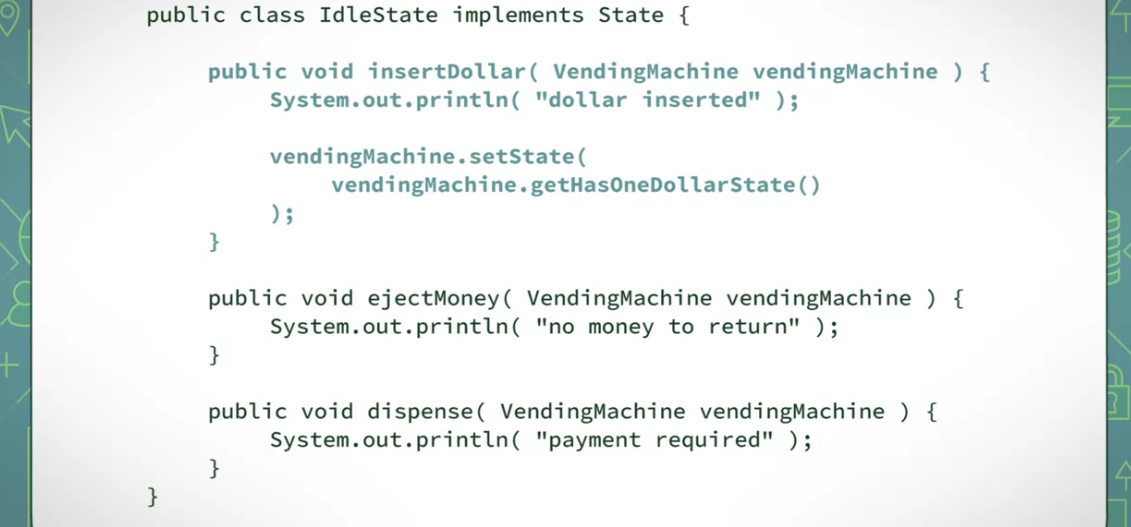


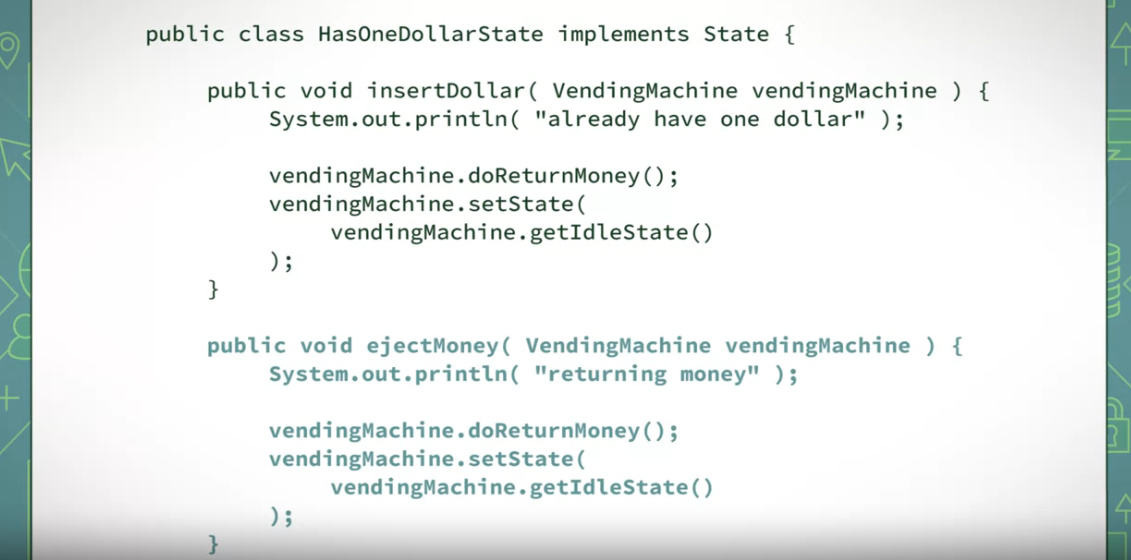
Dar aceste obiecte de State nu au responsabilitati de loc, sunt pasive si statusurile mai pot fi adaugate cu timpul si mereu va trebui sa modificam metodele, si asta poate fi extrem de greu sa o facem mereu cu if else pentru fiecare status posibil

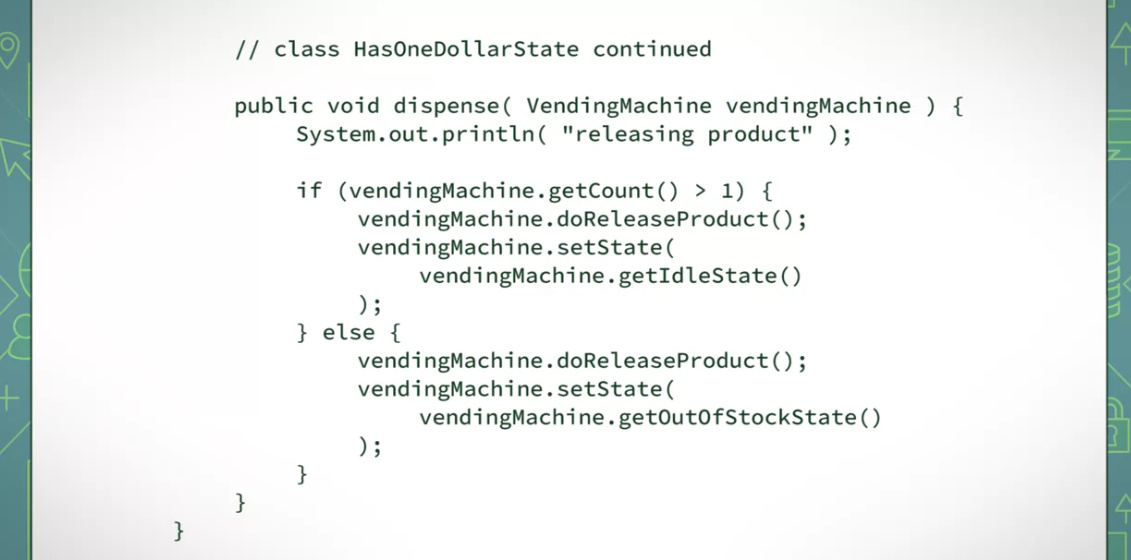
**Asa ar fi UML cu State pattern**

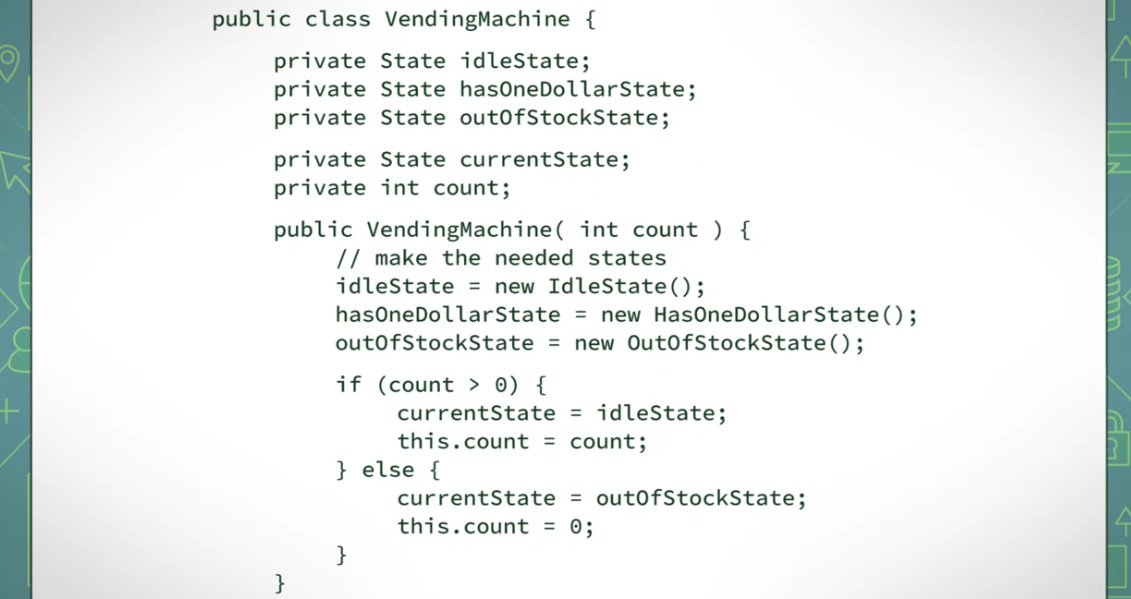


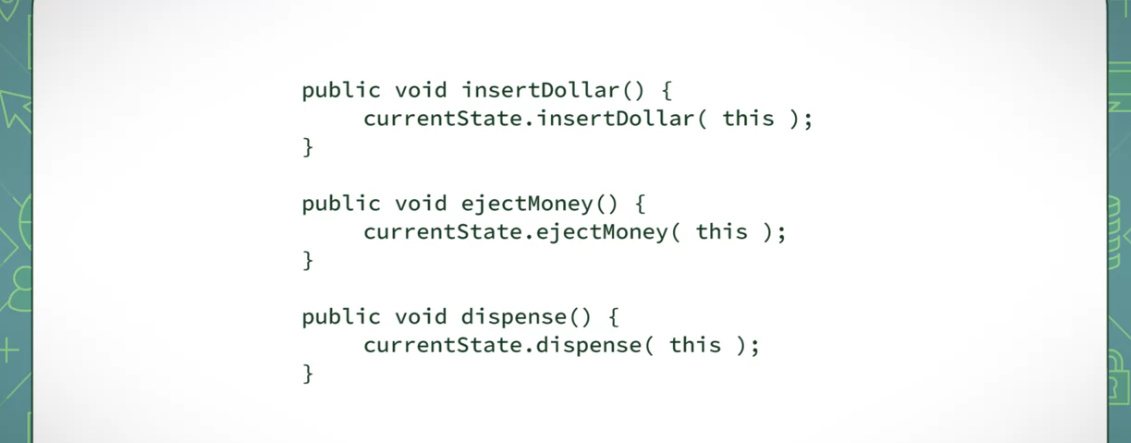












Clasa VendingMachine ar putea implementa si ea State, doar ca asta ar putea fi putin confusing

Acum, state objects vor fi cele care vor defini comportamentul si asa si lucreaza State pattern

In plus, vedem ca fiecare state are definite metodele pentru cele 3 events posibile, ceea ce cu if else mai sus era mult mai greu de facut si inteles.