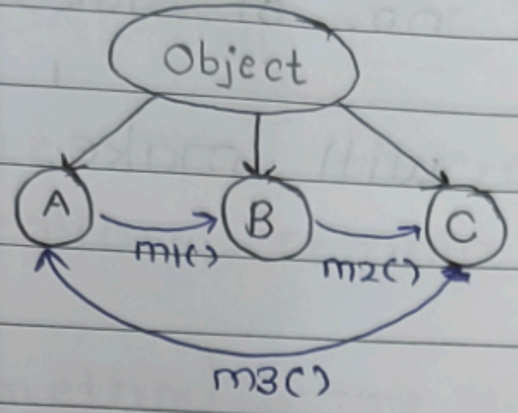
Lecture 32 : State Design Pattern

Introduction

- Suppose, ~~thre~~ there is an object which can exist in limited no. of states at a time.

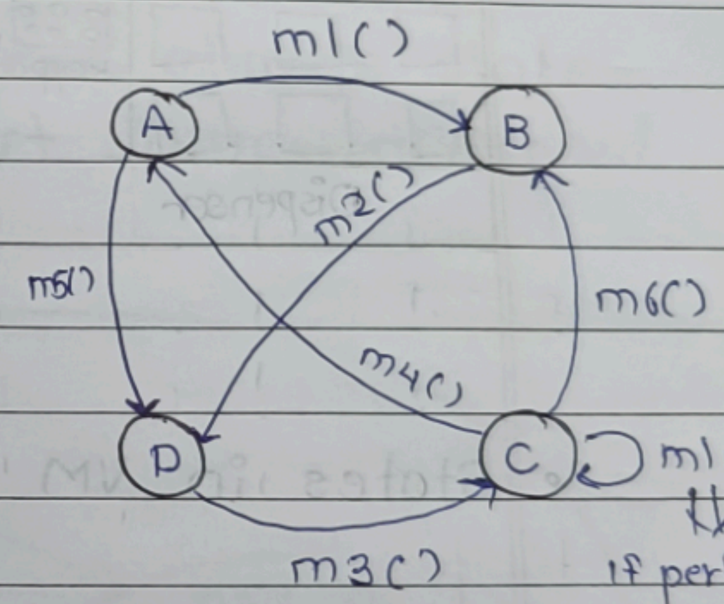
This is State machine diagram



- m1() - obj changes from state A to state B.
- m2() - state B to C
- m3() - state C to A

State Machine Diagram

- Objects
  - A
  - B
  - C
  - D
- Methods
  - m1() ⇒ A → B
  - m2() ⇒ B → D
  - m3() ⇒ D → C
  - m4() ⇒ C → A
  - m5() ⇒ A → D
  - m6() ⇒ C → B



- Inshort, when we perform any method on any object, it will change its state or stay on that state.

- Note : No. of states and methods will also be finite.

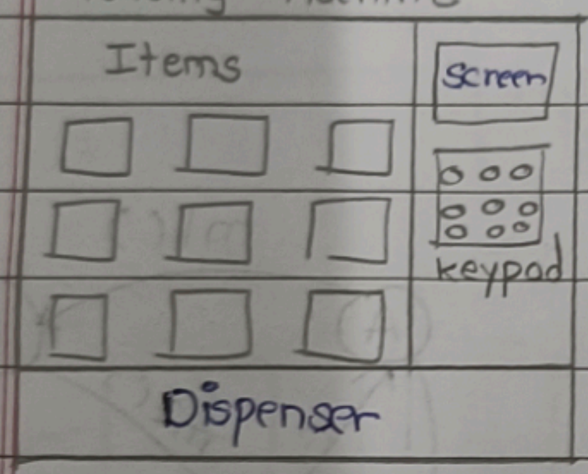
- When to use state Design Pattern
  whenever object changes its states after particular operation/method, we can use state design Pattern.

Example : Building Vending Machine

- Standard example to understand ~~a~~ SMD as it remains in limited no. of states.
- Using state pattern will make its design loosely coupled.

* Working

Vending Machine ← VM



→ we will enter item needed through keypad then VM give us that item in dispenser

→

- States in VM

→ No Coin State : When coin is not inserted

→ HasCoinState : when coin is inserted

→ Dispense state : item is being dispensed

→ soldOutstate : when item is finished

→ ReturnCoinstate : t/M returns coin if unavailability of item or extra money inserted or other reasons

- Scenarios when object doesn't change state -

1. When object does not perform that operation on that state.

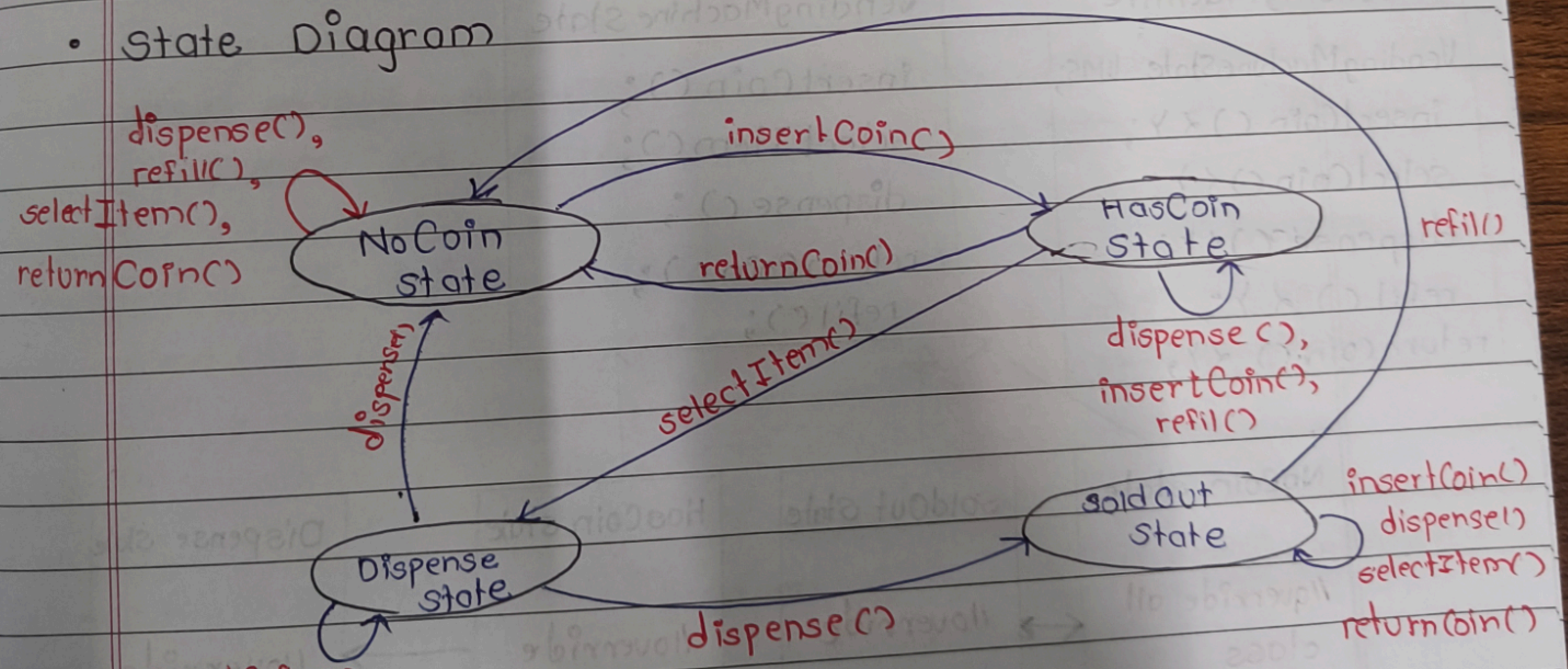
   Ex: At hasCoin state $\xrightarrow{\text{calling}}$ dispense()

2. When object can perform operation but the result of that operation remain VM in same state.

Ex: refil() method ⟶ NoCoin state.
VM is refilled but no state change.

Ex: insertCoin() ⟶ HasCoin state
It will accept coin but does not change state as item is not selected to get item dispensed

- State Diagram



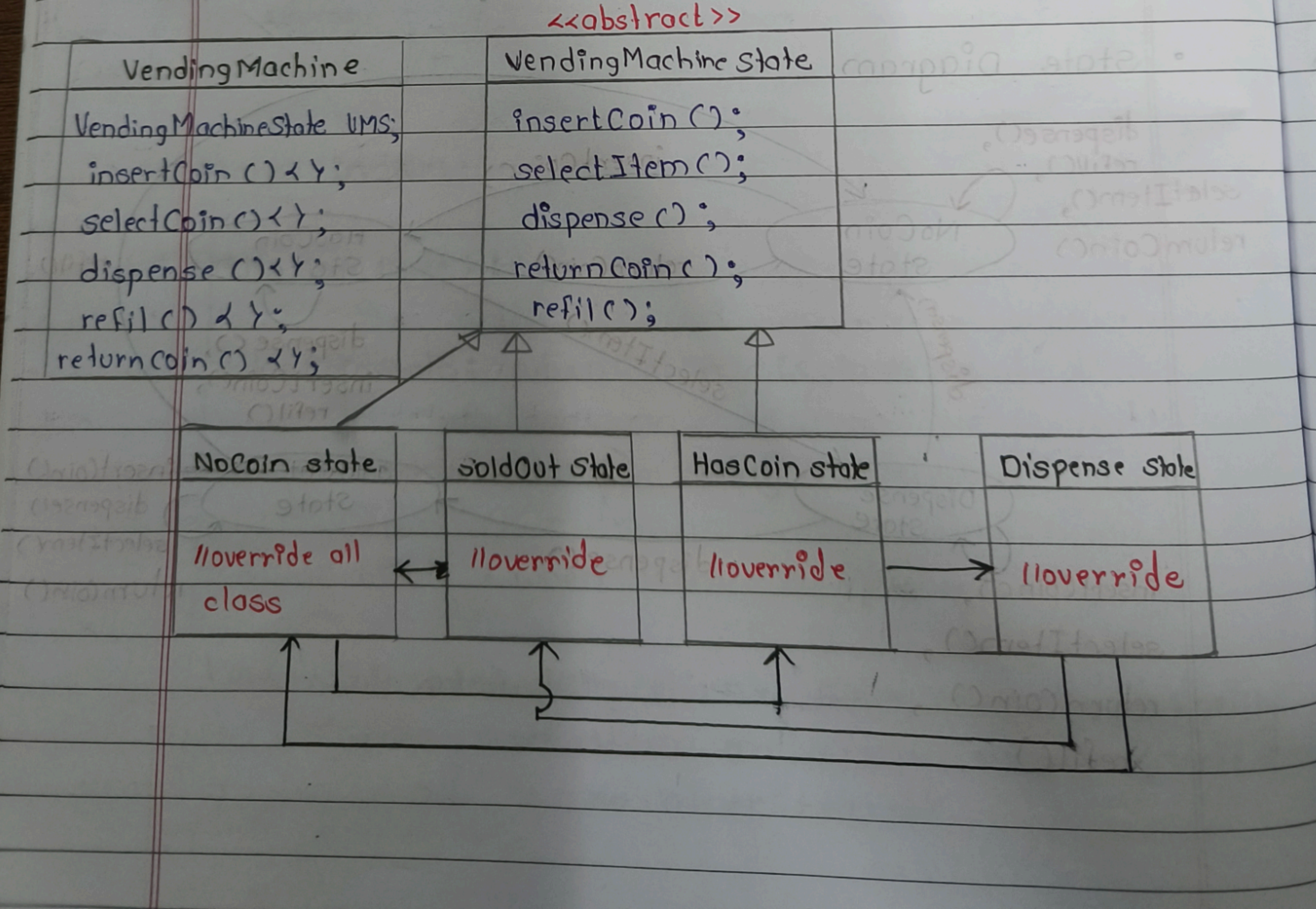
st = state

Actions / states	Insert Coin	Select Item	Dispense	Return Coin	Refil
NoCoin st	HasCoin st.	-	-	-	-
HasCoin st	-	Dispense st	-	NoCoin st.	-
Dispense st	-	-	NoCoin st. soldOut st.	-	-
soldOut st	-	-	-	-	NoCoin st

UML Design

- Hum saare states ko as classes banayenge. And ek ~~virtu~~ Vending Machine state banayenge (abstract class)

Final UML -



- Now main-class : VendingMachine jiska kaam hai to delegate task ⟹ ~~then~~ it is known as context class

How method works ?

- insertCoin() {
  state = state → insertCoin();
  }

} Assume currentState is noCoin state. toh jaise hi insertCoin() call hoga toh we will move to hasCoin state.

- Ab state ke paas reference hasCoin state.
selectCoin() {
  state = state → selectItem()
}

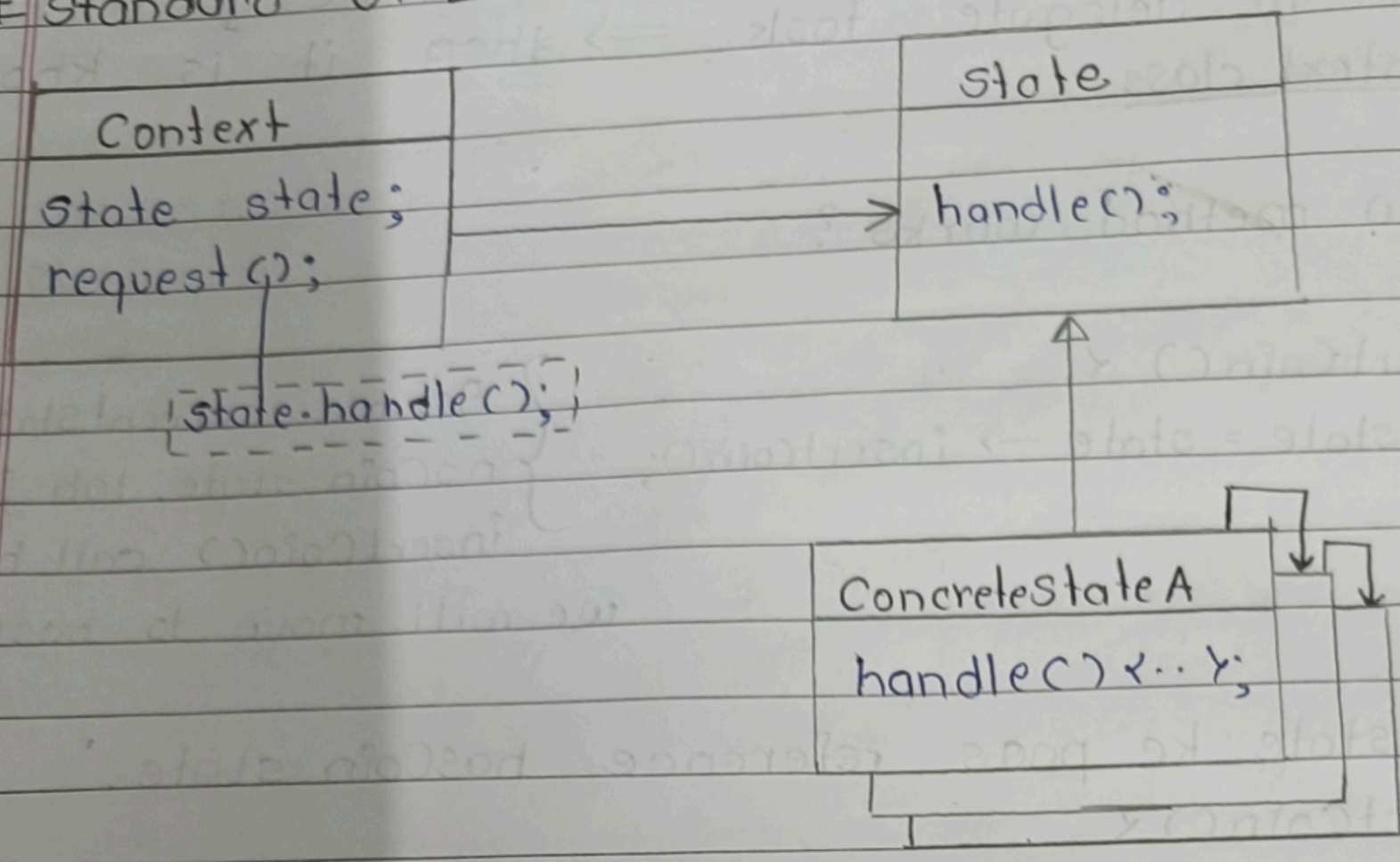
- Now state changes to dispense state aur fir reference of dispense state stores in it

- Aise hi saare state methods are performed
↳ VM khud saare method call nhi karta but uses state method which take over it to other method

- VM ka bss ek hi task hai ⟹ to update state variables / reference.

- Let's assume : Client calls selectItem() of VM → VM dumb object hai and selectItem() is called on NoCoin state toh state change nhi hoga aur exception message throw hoga screen par. aur NoCoin state return karega its own state.

Standard UML



Standard Definition

It allows an object to alter its behaviour when its internal state changes. The object will appear to change the class.

Real World Use Case

① Vending Machine (as H.W add more states)
② Document editor (Ex: GFG publish sft articles)
   ↳ Publish
   ↳ Archive
③ ATM M/C (COR to state)