Chain of Responsibility Pattern

Chain of Responsibility:

- The *Chain of Responsibility Pattern* is a ***behavioral design pattern*** that allows you to pass a ***request*** along a ***chain*** of ***potential handlers*** until one of them handles it.
- Instead of coupling a request sender to a specific receiver, this pattern **lets *multiple objects*** get a ***chance*** to ***handle*** the ***request***.

In simple terms,

Chain of Responsibility pattern creates a ***chain*** of ***receiver objects***. Each ***receiver decides*** either to ***process*** the **request** or to **pass it** to the ***next receiver*** in the ***chain***.

In this <u>example</u>, we implement the *ATM Money Dispenser*, where the ATM dispenses currency notes using different ***denominations*** (₹1000, ₹500, ₹100). Each denomination is handled by a ***separate handler*** in the ***chain***.

***<u>Problem</u> <u>Statement</u>*:**

An ATM must **dispense money** in ***minimum notes***.

- If the user requests **₹*3700***:
  - First, dispense as many **₹*1000* notes** as possible.
  - Then, dispense remaining with **₹*500* notes**.
  - Finally, use **₹*100* notes**.

We need a solution that is:

- ***Flexible*** (easy to add/remove denominations).
- ***Decoupled*** (each handler only knows about its own responsibility).

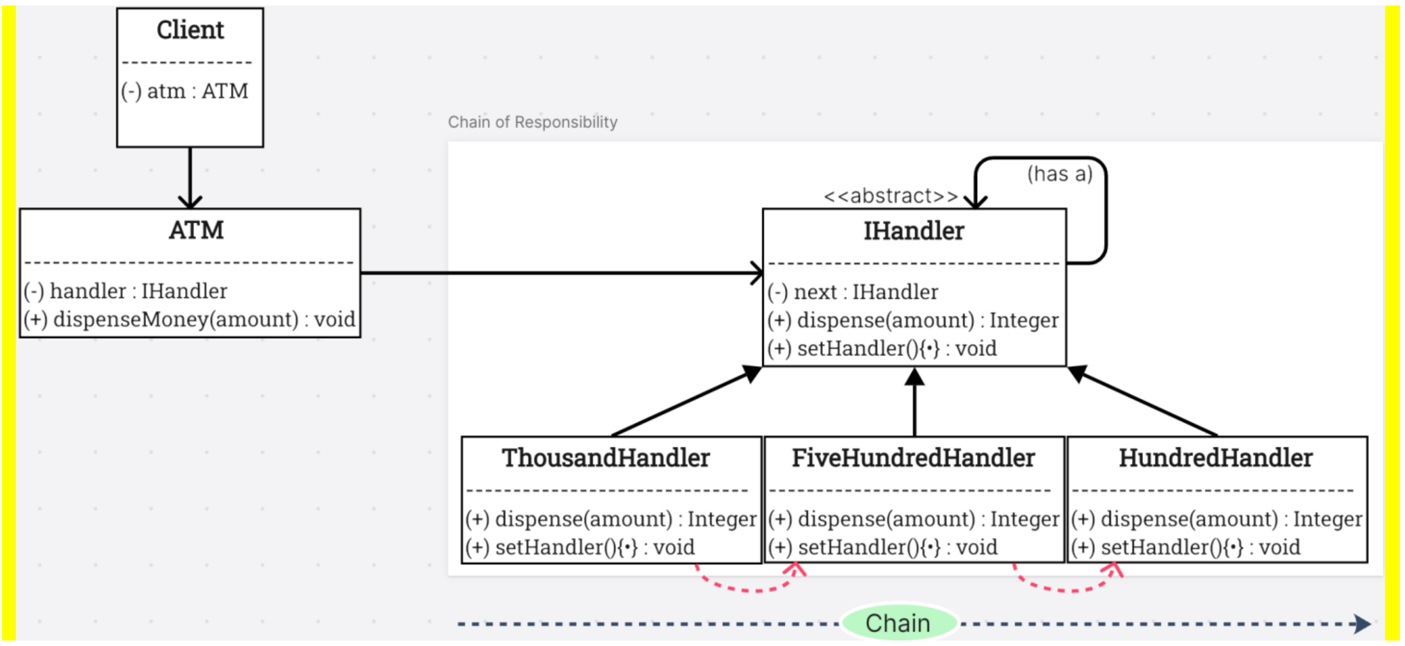
<u>Solution</u>*:** Applying ***Chain of Responsibility

We design a ***chain*** of ***handlers***:

1. **ThousandHandler** → tries to dispense ***₹1000*** notes.
2. **FiveHundredHandler** → handles the remaining amount with ***₹500*** notes.
3. **HundredHandler** → handles the remaining amount with ***₹100*** notes.

If a handler **can't fully process** the request, it **forwards** it to the **next handler** in the ***chain***.

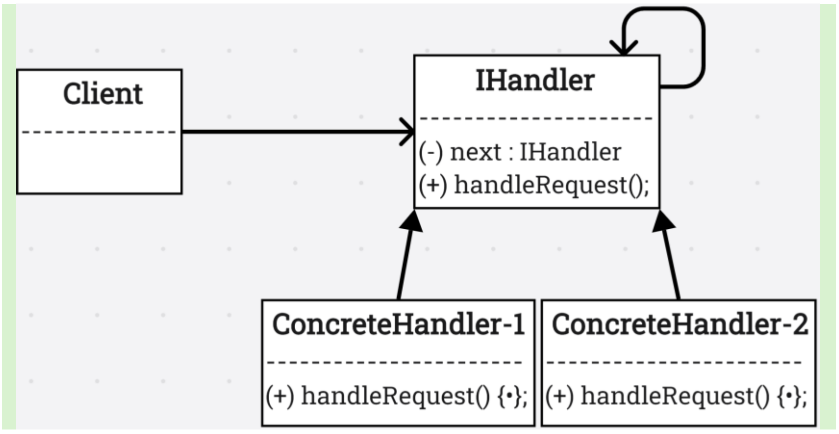
UML:


Client
(-) atm : ATM
ATM
(-) handler : IHandler
(+) dispenseMoney(amount) : void
Chain of Responsibility
(has a)
<<abstract>>
IHandler
(-) next : IHandler
(+) dispense(amount) : Integer
(+) setHandler(){•} : void
ThousandHandler
(+) dispense(amount) : Integer
(+) setHandler(){•} : void
FiveHundredHandler
(+) dispense(amount) : Integer
(+) setHandler(){•} : void
HundredHandler
(+) dispense(amount) : Integer
(+) setHandler(){•} : void
Chain

Code Link: [https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/22 Chain%20of%20Responsibility%20Pattern%20code](https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/22%20Chain%20of%20Responsibility%20Pattern%20code)

@Sibasundarj8

Standard UML:



Standard Def^n :-
Allow an object to pass request along a chain of potential Handlers. Each handler in the chain decides either to process the request or pass it to the next handler.

Use Cases of Chain of Responsibility Pattern:

1. *Logging Frameworks*
   - Different **loggers** form a chain: *ErrorLogger* → *FileLogger* → *ConsoleLogger*.
   - A log request passes through the chain.
   - **Example**: An ERROR message may be logged to ***file*** and ***console***, while an *INFO* message may only go to ***console***.
2. *Customer Support / Escalation System*
   - **Customer requests** pass through a chain of ***support levels***: *Level 1* Support → *Level 2* Support → *Manager*.
   - Each level decides whether it can ***handle*** the request or ***escalates*** it further.
3. *Access Control / Authorization*
   - Request for a resource goes through handlers like: *Authentication* → *RoleValidation* → *PermissionCheck*.
   - Each handler ensures ***its part of the security check*** before allowing access.
4. *Approval Workflows*
   - In organizations: *Team Lead* → *Project Manager* → *Director*.
   - An expense request is approved at the ***appropriate level***, or ***escalated*** further.

Chain of Responsibility (CoR) ***VS*** Linked List (LL):

- CoR → A ***design pattern*** for **request handling**. Each object (**handler**) decides: "*Can I handle this? If not, pass to next.*"
- LL → A ***data structure*** for **storing data**. Each ***node*** just ***points*** to the ***next node***, no decision-making.

In short:

- CoR = *behavior-driven* (who handles request)
- LL = *data-driven* (how elements are stored/linked)

@Sibasundarj8