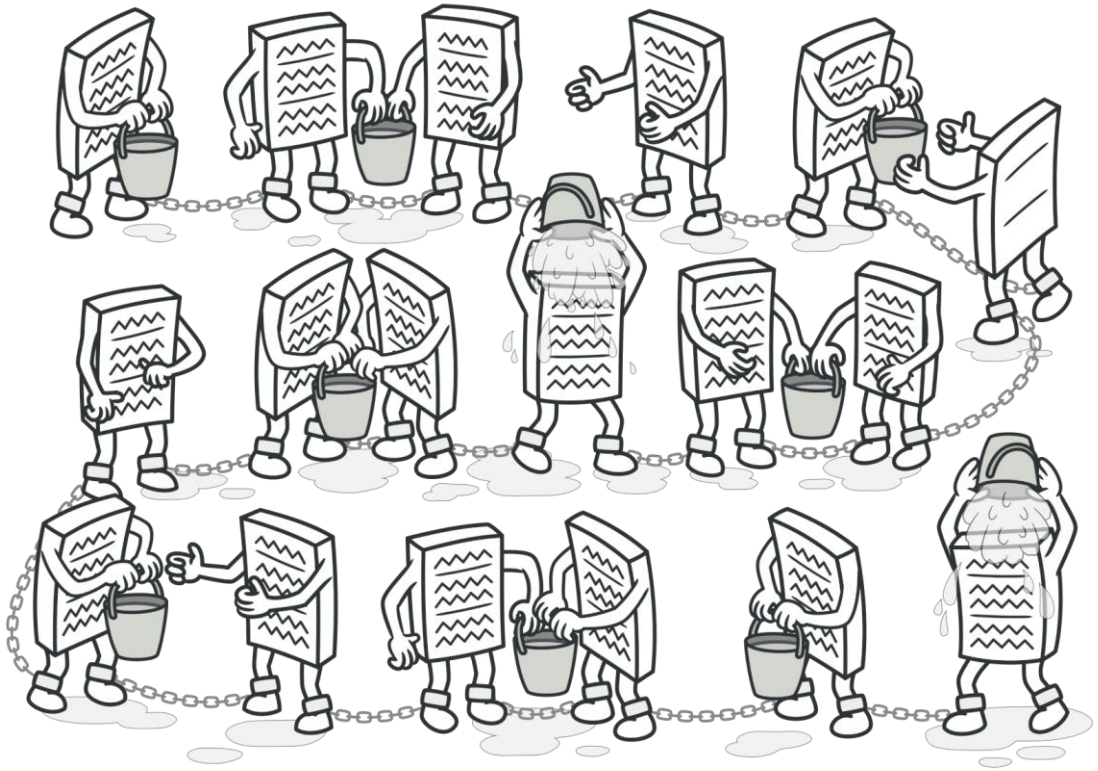
CHAIN OF RESPONSIBILTY

The chain of responsibility design pattern is defined by the following roles:

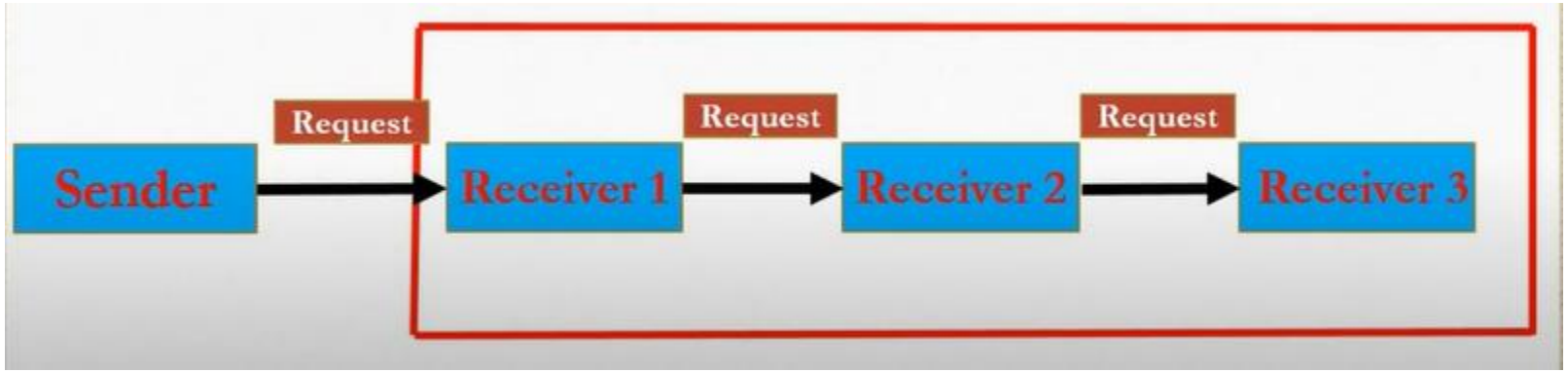
- Sender
- Handler
- Receiver



INTENT

- **Chain of Responsibility** is designed to avoid any coupling between the sender of request and the receiver.
- It allows you to pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

- It will allow and object to send a responsibility to another object without knowing whether the object will be able to handle it.



- A group of objects consists of many receiver and the Sender is the client. The client requests for a service to the receiver.
- The receiver 1 one checks if it can handle the request and forwards to receiver 2 if unhandled.
- The receiver 2 checks again if the request can be handled, else it passes the request. The chain goes on till the request is processed.
- This is termed as the **CHAIN OF RESPONSIBILITY.**

EXAMPLE

- The ATM machine have four handlers. The Two Thousand handler will give 2000 rupees. The Five Hundred Handler will give 500 hundred rupees and in the same for 200 and 100 handlers.
- Anurag wants to withdraw 4600 rupees from the ATM machine.
- The ATM machine sends the request to the first handler i.e. the Two Thousand Handler, and it checks the amount and give two 2000 rupees notes. The remaining is 600 rupees. .

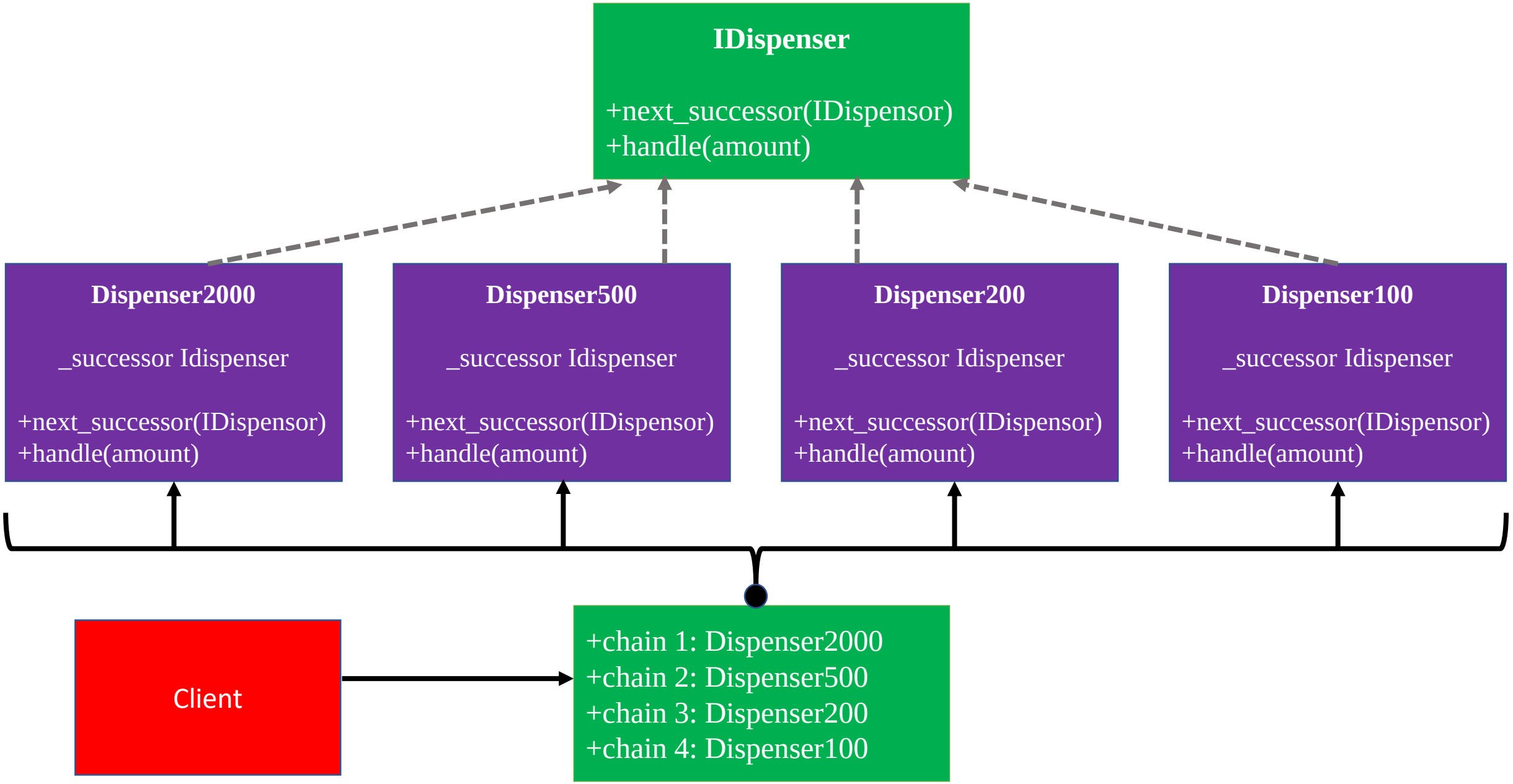
- It then sends the request to the Five Hundred Handler and this checks the remaining amount and give one 500 rupees note and forward the request to the next handler i.e. Two Hundred Handler

- Two Hundred Handler checks the remaining amount which is 100. So, it can’t handle the request and it simply forwards the request to the next handler which is Hundred Handler.

- The Hundred Handler check the remaining amount which is 100 and will give one 100 rupees note.

- In this way, it will handle the request and provide 4600 (2 * 2000, 1*500, and 1*100) to Anurag.

- This is one of the best examples of Chain of Responsibility. Similar working procedure is used in Vending Machine


IDispenser
+next_successor(IDispensor)
+handle(amount)
Dispenser2000
_successor Idispenser
+next_successor(IDispensor)
+handle(amount)
Dispenser500
_successor Idispenser
+next_successor(IDispensor)
+handle(amount)
Dispenser200
_successor Idispenser
+next_successor(IDispensor)
+handle(amount)
Dispenser100
_successor Idispenser
+next_successor(IDispensor)
+handle(amount)
Client
+chain 1: Dispenser2000
+chain 2: Dispenser500
+chain 3: Dispenser200
+chain 4: Dispenser100

```
""" ATM Machine Dispenser"""
from abc import ABCMeta,
abstractmethod


class IDispenser(metaclass=ABCMeta):

    @abstractmethod
    def next_successor(successor):
        """Set the next handler in the chain"""

    @abstractmethod
    def handle(amount):
        """Handle the event"""
```

```
class ATMDispenserChain:
    # The Chain Client

    def __init__(self):
        # initializing the successors chain
        self.chain1 = Dispenser2000()
        self.chain2 = Dispenser500()
        self.chain3 = Dispenser200()
        self.chain4 = Dispenser100()

        # The successor chain will be recalculated
dynamically at runtime.
        self.chain1.next_successor(self.chain2)
        self.chain2.next_successor(self.chain3)
        self.chain3.next_successor(self.chain4)
```

```
class Dispenser2000(IDispenser):
    # Dispenses Rs 2000 notes if applicable, otherwise
continues to next successor

    def __init__(self):
        self._successor = None

    def next_successor(self, successor):
        # Set the next successor
        self._successor = successor

    def handle(self, amount):
        # Handle the dispensing of notes
        if amount >= 2000:
            num = amount // 2000
            remainder = amount % 2000
            print(f"Dispensing {num} Rs 2000 note")
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)
```

```
class Dispenser500(IDispenser):
    # Dispenses Rs 500 notes if applicable, otherwise
continues to next successor

    def __init__(self):
        self._successor = None

    def next_successor(self, successor):
        # Set the next successor
        self._successor = successor

    def handle(self, amount):
        # Handle the dispensing of notes
        if amount >= 500:
            num = amount // 500
            remainder = amount % 500
            print(f"Dispensing {num} Rs 500 note")
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)
```

```python
class Dispenser100(IDispenser):
    # Dispenses Rs 100 notes if applicable, otherwise continues to next successor

    def __init__(self):
        self._successor = None

    def next_successor(self, successor):
        # Set the next successor
        self._successor = successor

    def handle(self, amount):
        # Handle the dispensing of notes
        if amount >= 100:
            num = amount // 100
            remainder = amount % 100
            print(f"Dispensing {num} Rs 100 note")
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)
```

```python
class Dispenser200(IDispenser):
    # Dispenses Rs 200 notes if applicable, otherwise continues to next successor

    def __init__(self):
        self._successor = None

    def next_successor(self, successor):
        # Set the next successor
        self._successor = successor

    def handle(self, amount):
        # Handle the dispensing of notes
        if amount >= 200:
            num = amount // 200
            remainder = amount % 200
            print(f"Dispensing {num} Rs 200 note")
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)
```

```python
ATM = ATMDispenserChain()
AMOUNT = int(input("Enter an amount to withdraw : "))
if AMOUNT < 100 or AMOUNT % 100 != 0:
    print("Amount should in multiple of 100s.")
    exit()
```

```python
# process the request
ATM.chain1.handle(AMOUNT)
print("Your withdrawal is successful!!!")
```

OUTPUT

```
Enter an amount to withdraw : 4600
Dispensing 2 Rs 2000 note
Dispensing 1 Rs 500 note
Dispensing 1 Rs 100 note
Your withdrawal is successful!!!


Enter an amount to withdraw : 1051
Amount should in multiple of 100s.
```