**The Scenario**:

Imagine you have multiple bank accounts, and you want to process payments using a specific sequence of banks. If one bank account doesn't have enough balance to cover the payment, you want to automatically pass the payment request to the next bank in line until the payment is successful.

**Chain of Responsibility Design Pattern Explained**:

* **Linked Accounts**: The Chain of Responsibility pattern is all about creating a linked chain of objects (bank accounts in this case). Each object can either handle a request or pass it along to the next object in the chain.
* **Abstract Account Class**: You define a common structure using the Account abstract class. It has methods like SetNext() to set the next account in the chain and Pay() to process payment requests.
* **Concrete Account Classes**: The concrete bank classes (HDFC, SBI, AXIS) inherit from the Account class. Each bank knows how to process a payment based on its balance and capabilities.
* **Setting Up the Chain**: In the Main method, you create instances of different bank accounts (HDFC, SBI, AXIS). You set up the order of the chain by using the SetNext() method, linking the accounts together.
* **Payment Processing**: When you initiate a payment using the first bank account (HDFC), it attempts to process the payment. If it can't cover the payment, it passes the request to the next bank (SBI). This process continues until a bank with sufficient balance is found or the end of the chain is reached.

**In Simple Words**:

The Chain of Responsibility pattern is like standing in a line where each person (bank account) has a specific task to perform. If one person can't complete the task, they pass it to the next person until someone can handle it. In this code, the banks form a line, and when a payment request comes in, the first bank tries to pay. If it can't, it hands off the request to the next bank, and so on, until the payment is either successful or none of the banks can handle it. This way, you ensure that payments are processed by the banks in the right order, and the payment gets through even if one bank can't cover it.

**Step 1: Define Classes**

In this step, you have defined several classes representing different banks and their accounts. These classes inherit from the Account abstract class.

internal class Program

{

**// Abstract class representing an account**

abstract class Account

{

**// ...**

}

**// Concrete classes for different banks (HDFC, SBI, AXIS)**

class HDFC : Account

{

**// ...**

}

class SBI : Account

{

**// ...**

}

class AXIS : Account

{

**// ...**

}

**// ...**

}

**Step 2: Implement Chain of Responsibility**

In this step, you set up the Chain of Responsibility pattern by linking the banks together in a chain.

static void Main(string[] args)

{

**// Creating instances of different bank accounts**

var bank1 = new HDFC(100);

var bank2 = new SBI(200);

var bank3 = new AXIS(300);

**// Setting up the chain of responsibility**

bank1.SetNext(bank2);

bank2.SetNext(bank3);

**// Initiating payment using the chain**

bank1.Pay((decimal)100);

}

**Step 3: Using the Chain**

In the Main method, you create instances of different bank accounts and link them together using the SetNext method. This sets up a chain where the payment request will pass through each bank in the order they were linked.

abstract class Account

{

**// ...**

**// Setting the next account in the chain**

public void SetNext(Account account)

{

mSuccessor = account;

}

**// ...**

}

**Step 4: Processing Payment Requests**

In the Pay method of the Account class, you process payment requests based on the balance of each account. If an account can pay the requested amount, it processes the payment. If not, the request is passed to the next account in the chain.

public void Pay(decimal amountTopay)

{

if (CanPay(amountTopay))

{

Console.WriteLine($"Paid {amountTopay} using {this.GetType().Name}.");

}

else if (this.mSuccessor != null)

{

Console.WriteLine($"Cannot pay using {this.GetType().Name}. Proceeding..");

mSuccessor.Pay(amountTopay);

}

else

{

Console.WriteLine("None of the accounts have enough balance");

}

}

**Explanation of Chain of Responsibility Pattern**:

The Chain of Responsibility pattern is used to create a chain of processing objects (in this case, bank accounts) where each object in the chain has the ability to process a request or pass it along the chain.

In this example,

* Account abstract class serves as the basis for the chain. It defines the structure of how accounts should process payment requests and pass them to the next account if needed.
* The concrete bank classes (HDFC, SBI, AXIS) extend the Account class and implement their own logic for processing payment requests.
* By linking the bank accounts using the SetNext method, you establish the sequence in which the payment request will be processed. If one account cannot pay, it delegates the request to the next account in the chain.
* The payment request travels through the chain, stopping at the first account that can process it or reaching the end of the chain if no account has enough balance.

In this code example, the Chain of Responsibility pattern enables a flexible and sequential processing of payment requests across different bank accounts in the order they are linked.