**Chain of Responsibility Design Pattern:** Chain of Responsibility Pattern or Chain of Responsibility Method is a [Behavioral Design Pattern](https://www.geeksforgeeks.org/behavioral-design-patterns/), which allows an object to send a request to other objects without knowing who is going to handle it.

* This pattern is frequently used in the chain of multiple objects, where each object either handles the request or passes it on to the next object in the chain if it is unable to handle that request.
* This pattern encourages loose coupling between sender and receiver, providing freedom in handling the request.

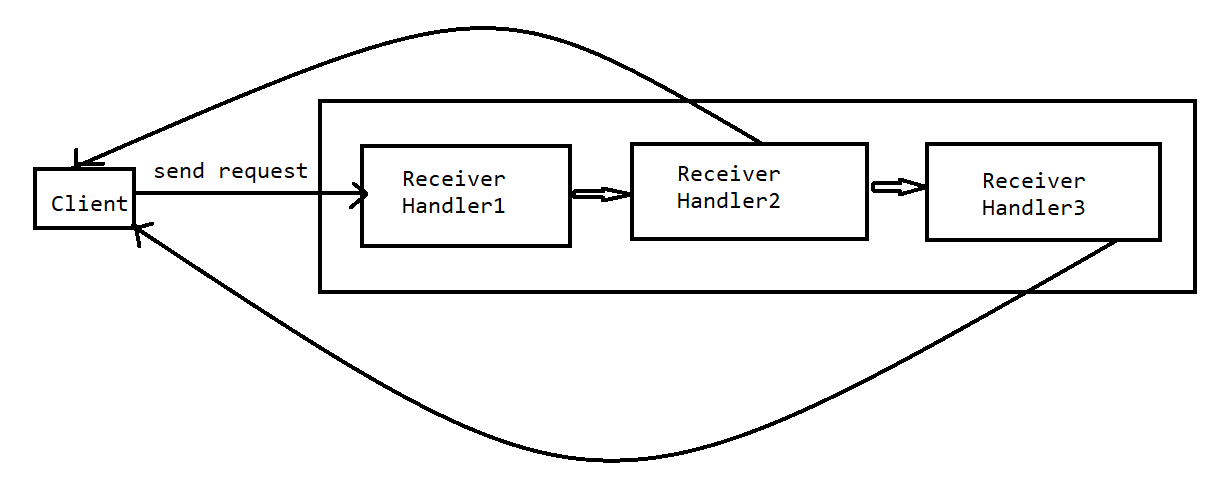
**Characteristics of the Chain of Responsibility Design Pattern**

* **Loose Coupling**: The pattern promotes loose coupling between the sender and receiver of a request, as the sender doesn’t need to know which object will handle the request and the receiver doesn’t need to know the structure of the chain.
* **Dynamic Chain**: The chain can be modified dynamically at runtime, allowing for flexibility in adding or removing handlers without affecting the client code.
* **Single Responsibility Principle**: Each handler in the chain has a single responsibility, either handling the request or passing it to the next handler, which helps in maintaining a clean and modular design.
* **Sequential Order**: Requests are processed sequentially along the chain, ensuring that each request is handled in a predefined order.
* **Fallback Mechanism**: The chain can include a mechanism to handle requests that are not handled by any handler in the chain, providing a fallback or default behavior.
* **Variants**: The pattern has variants like a linear chain, where each handler has a single successor, or a tree-like structure, where a handler can have multiple successors, allowing for more complex processing logic.
* **Enhanced Flexibility**: The pattern allows for enhanced flexibility in handling requests, as the chain can be configured or modified to suit different requirements without changing the client code.

Example: imagine a customer support system where customer requests need to be handled based on their priority. There are three levels of support: Level 1, Level 2, and Level 3. Level 1 support handles basic requests, Level 2 support handles more complex requests, and Level 3 support handles critical issues that cannot be resolved by Level 1 or Level 2.

Real time examples:

1. ATM / Vinding Machine
2. Design Logger



In the above shown diagram client only responsible to send the request, first it will go to handle in Receiver Handle1, if it not able to handle the request then forward the request to next handler, until getting handled the request it will going to forward. If none of the handler able to resolve it will going to fall under fallback mechanism.

