Communication Between Objects in Chain of Responsibility Pattern

In the Chain of Responsibility pattern, communication between objects is organized into a chain where each object in the chain has the opportunity to process a request or pass it along to the next object in the chain. This pattern decouples the sender of a request from its receiver by giving multiple objects the chance to handle the request.

Key Components and Their Communication:

1. Handler (Abstract Class or Interface):

- Role: Declares a method (`handleRequest()`) that each concrete handler will implement to either process the request or pass it along to the next handler.

- Communication: The handler interface defines the way requests will be communicated between handlers.

2. Concrete Handlers:

- Role: Implement the `handleRequest()` method. Each concrete handler can either process the request or determine that it cannot handle it and pass it to the next handler in the chain.

- Communication: Each handler checks if it can process the request. If it can, it processes the request and ends the communication. If the handler cannot process the request, it passes the request to the next handler in the chain by calling `nextHandler.handleRequest()`.

3. Client:

- Role: The client creates and configures the chain of handlers and sends the request to the first handler in the chain.

- Communication: The client initiates the communication by passing the request to the first handler in the chain.

Communication Flow in the Chain of Responsibility Example (Support Ticket System):

1. Client Setup:

- The client creates instances of different levels of support (`Level1Support`, `Level2Support`, `Level3Support`).

- The client links these handlers together to form a chain: `Level1Support` -> `Level2Support` -> `Level3Support`.

- The client sends a request to the first handler (`Level1Support`).

2. Handler Communication:

- Level1Support:

  - Receives the request with severity `2`.

  - Checks if it can handle the request (if severity is 1). Since it can't, it passes the request to `Level2Support`.

  - Code:

  ```java
  if (severity <= 1) {
      System.out.println("Level 1 Support: Handling issue with severity " + severity);
  } else if (nextHandler != null) {
      nextHandler.handleRequest(severity);
  }
  ```

- Level2Support:

  - Receives the request from `Level1Support`.

  - Checks if it can handle the request (if severity is 2). Since it can, it processes the request and communication stops.

  - Code:

  ```java
  if (severity <= 2) {
      System.out.println("Level 2 Support: Handling issue with severity " + severity);
  } else if (nextHandler != null) {
      nextHandler.handleRequest(severity);
  }
  ```

3. Passing the Request:

- If neither `Level1Support` nor `Level2Support` could handle the request, it would be passed to `Level3Support` in a similar manner.

- If `Level3Support` also couldn't handle it, the chain would end, and the request would go unhandled.

Summary of Communication in the Chain of Responsibility:

- Client: Initiates the communication by passing the request to the first handler in the chain.

- Handlers: Each handler checks if it can process the request. If it can, it processes the request and stops the communication. If it can't, it passes the request to the next handler in the chain.

- Next Handler: The request is passed down the chain until one handler processes it or the end of the chain is reached.

In this pattern, the responsibility for handling a request is shared among a chain of objects, with each object having the opportunity to either handle the request or pass it along. This results in a flexible and dynamic way of processing requests, where handlers can be added, removed, or reordered without affecting the client code.