Chain of Responsibility Pattern

1. Customer Requirement

A customer wants to develop a support ticket system where users can submit their issues. Depending on the issue's severity, the system should automatically escalate the ticket to the appropriate support level (Level 1, Level 2, Level 3). The system should be flexible enough to allow adding or removing support levels without affecting other parts of the system.

2. Design Pattern Explanation

The Chain of Responsibility pattern allows an object to pass a request along a chain of potential handlers until one of them handles the request. It decouples the sender of a request from its receiver, allowing more than one object to handle the request.

Example:

- Support Ticket System:
  - Level 1 Support: Handles simple issues.
  - Level 2 Support: Handles more complex issues.
  - Level 3 Support: Handles the most complex issues.
  - If Level 1 cannot handle the issue, it passes the request to Level 2, and so on.

3. Java Code Implementation

```java
// Abstract Handler
abstract class SupportHandler {
    protected SupportHandler nextHandler;

    public void setNextHandler(SupportHandler nextHandler) {
```

```
        this.nextHandler = nextHandler;
    }

    public abstract void handleRequest(int severity);
}

// Concrete Handler 1
class Level1Support extends SupportHandler {
    @Override
    public void handleRequest(int severity) {
        if (severity <= 1) {
            System.out.println("Level 1 Support: Handling issue with severity " + severity);
        } else if (nextHandler != null) {
            nextHandler.handleRequest(severity);
        }
    }
}

// Concrete Handler 2
class Level2Support extends SupportHandler {
    @Override
    public void handleRequest(int severity) {
        if (severity <= 2) {
            System.out.println("Level 2 Support: Handling issue with severity " + severity);
        } else if (nextHandler != null) {
            nextHandler.handleRequest(severity);
        }
```

```
    }
}

// Concrete Handler 3
class Level3Support extends SupportHandler {
    @Override
    public void handleRequest(int severity) {
        if (severity <= 3) {
            System.out.println("Level 3 Support: Handling issue with severity " + severity);
        } else {
            System.out.println("No support available for severity " + severity);
        }
    }
}

// Client
public class ChainOfResponsibilityDemo {
    public static void main(String[] args) {
        SupportHandler level1 = new Level1Support();
        SupportHandler level2 = new Level2Support();
        SupportHandler level3 = new Level3Support();

        level1.setNextHandler(level2);
        level2.setNextHandler(level3);

        int[] issues = {1, 2, 3, 4};
```

```
        for (int severity : issues) {
            level1.handleRequest(severity);
        }
    }
}
```

4. Other Important Information

- Decoupling: The sender and receiver of the request are decoupled, allowing for flexible object interactions.
- Open/Closed Principle: You can extend the chain with new handlers without modifying existing code.
- Responsibility: Each handler has a single responsibility, enhancing code readability and maintainability.