Chain of Responsibility Pattern

Resources

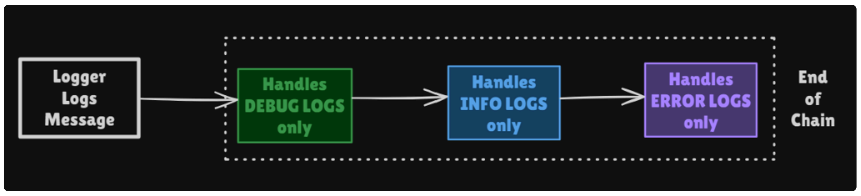
- Low Level Design from Basics to Advanced (Some Initial Videos are in Hindi, rest in English)
- 10. Design Logging System (Hindi) | Chain of Responsibility Design Pattern | System Design interview

Definition

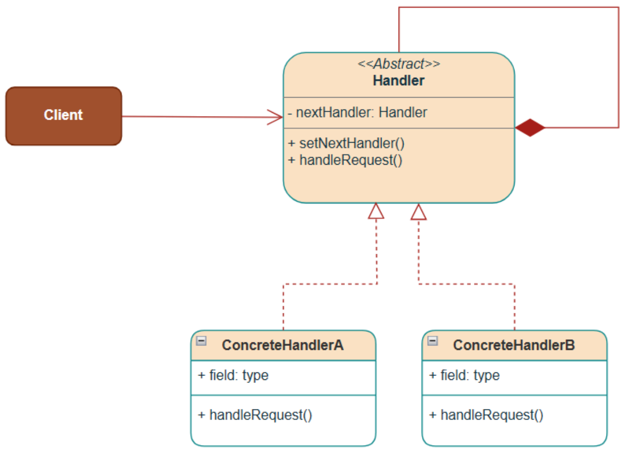
> ***The Chain of Responsibility pattern is a behavioral design pattern that passes requests along a chain of handlers. Each handler decides either to process the request or pass it to the next handler in the chain.***

Real-life Examples

- ATM Vending Machine
- Application Logging System

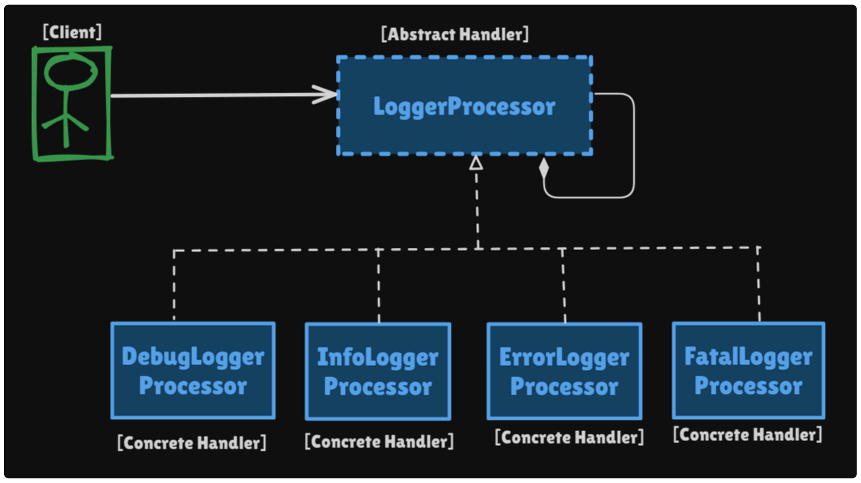


Class Diagram



Structure of CoR Pattern

Let's understand the class diagram using the Logging System example:



1. **Handler Interface (`LogProcessor`)**
   - This can be an Abstract Base Class or an Interface.
   - The abstract class defines the common interface and chain mechanism.
   - Declares the `handleRequest()` method.
   - Defines the `setNextHandler()` method.
   - Holds a reference to concrete handlers that implement this interface.
2. **ConcreteHandlers (e.g., `DebugLogProcessor` , `InfoLogProcessor` , and so on)**
   - Implement the `Handler` interface.
   - It will handle the request or forward it along to the next handler in the chain.
3. **Client**
   - Composes the chain of handlers using the `setNext()` method dynamically.
   - Initiates the request using the first handler in the chain.

Implementation (Example: Logging System)

```java
// Abstract Logger class - defines the chain structure
public abstract class LogProcessor {

    public static final int DEBUG = 1;
    public static final int INFO = 2;
    public static final int ERROR = 3;
    public static final int FATAL = 4;
    int level;
    LogProcessor nextLoggerProcessor;

    public void setNextLogger(LogProcessor nextLogger) {
        this.nextLoggerProcessor = nextLogger;
    }

    public void logMessage(int level, String message) {
        if (this.level <= level) {
            write(message);
        }

        // Pass to next handler in chain if exists
        if (this.nextLoggerProcessor != null) {
            this.nextLoggerProcessor.logMessage(level, message);
        }
    }

    abstract protected void write(String message);
}
```

```java
// Concrete handler for DEBUG level
public class DebugLogProcessor extends LogProcessor {
    public DebugLogProcessor(int level) {
        this.level = level;
    }

    @Override
    protected void write(String message) {
        System.out.println("DEBUG: " + message);
    }
}
```

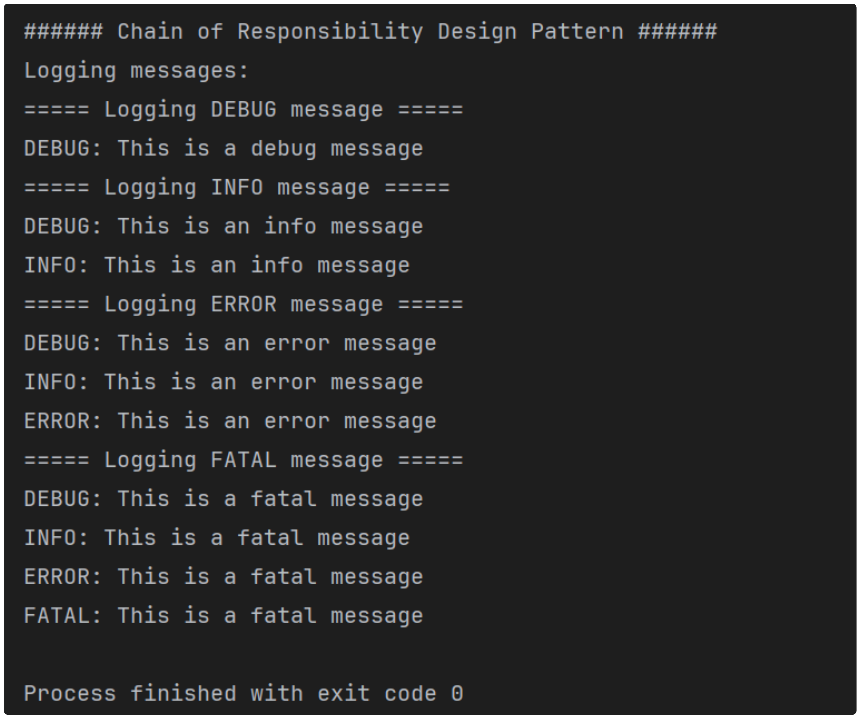
```java
// Concrete handler for INFO level
public class InfoLogProcessor extends LogProcessor {
    public InfoLogProcessor(int level) {
        this.level = level;
    }

    @Override
    protected void write(String message) {
        System.out.println("INFO: " + message);
    }
}
```

```java
// Concrete handler for ERROR level
public class ErrorLogProcessor extends LogProcessor {
    public ErrorLogProcessor(int level) {
        this.level = level;
    }

    @Override
    protected void write(String message) {
        System.out.println("ERROR: " + message);
    }
}
```

```java
// Concrete handler for FATAL level
public class FatalLogProcessor extends LogProcessor {
    public FatalLogProcessor(int level) {
        this.level = level;
    }

    @Override
    protected void write(String message) {
        System.out.println("FATAL: " + message);
    }
}
```

```java
// Client code
public class LoggerDemo {
    public static void main(String[] args) {
        System.out.println("###### Chain of Responsibility Design Pattern ######");

        // Get the chain of loggers
        LogProcessor logProcessor = getChainOfLoggers();

        System.out.println("Logging messages:");
        System.out.println("===== Logging DEBUG message =====");
        logProcessor.logMessage(LogProcessor.DEBUG, "This is a debug message");
        System.out.println("===== Logging INFO message =====");
        logProcessor.logMessage(LogProcessor.INFO, "This is an info message");
        System.out.println("===== Logging ERROR message =====");
        logProcessor.logMessage(LogProcessor.ERROR, "This is an error message");
        System.out.println("===== Logging FATAL message =====");
        logProcessor.logMessage(LogProcessor.FATAL, "This is a fatal message");
    }

    private static LogProcessor getChainOfLoggers() {
        LogProcessor fatalLogger = new FatalLogProcessor(LogProcessor.FATAL); // 4
        LogProcessor errorLogger = new ErrorLogProcessor(LogProcessor.ERROR); // 3
        LogProcessor infoLogger = new InfoLogProcessor(LogProcessor.INFO); // 2
        LogProcessor debugLogger = new DebugLogProcessor(LogProcessor.DEBUG); // 1

        // Dynamic Chaining: DEBUG -> INFO -> ERROR -> FATAL
        debugLogger.setNextLogger(infoLogger);
        infoLogger.setNextLogger(errorLogger);
        errorLogger.setNextLogger(fatalLogger);
        // fatalLogger.nextLoggerProcessor is null; // Last logger in chain

        return debugLogger;  // Return the first LogProcessor in the chain
    }
}
```

Output



```
###### Chain of Responsibility Design Pattern ######
Logging messages:
===== Logging DEBUG message =====
DEBUG: This is a debug message
===== Logging INFO message =====
DEBUG: This is an info message
INFO: This is an info message
===== Logging ERROR message =====
DEBUG: This is an error message
INFO: This is an error message
ERROR: This is an error message
===== Logging FATAL message =====
DEBUG: This is a fatal message
INFO: This is a fatal message
ERROR: This is a fatal message
FATAL: This is a fatal message

Process finished with exit code 0
```