22 Behavioral: Chain of Responsibility for Tiered Logging

**Scenario** Your micro-service must log messages at **DEBUG < INFO < ERROR**. Each logger may forward unhandled messages to the next level. The order and presence of loggers must be configurable at runtime.

#### **Tasks**

1 analysis/cor\_need.md—Explain why branching (if(level...)) inside the caller violates the Open/Closed Principle and how a chain fixes it.  
 2 Implementation (src/main/java/logging/)  
 • LogLevel.java enum  
 • Logger interface void log(String msg, LogLevel level)  
 • AbstractLogger—holds AbstractLogger next, default log(...) method with forwarding, abstract canHandle(level) and write(msg)  
 • Concrete handlers DebugLogger, InfoLogger, ErrorLogger—implement canHandle & write (print with label)  
 3 LoggingChainBuilder.java—utility to build a chain from a List<LogLevel> at runtime.  
 4 ChainDemo.java—build chain DEBUG→INFO→ERROR, send three messages and show output; build chain INFO→ERROR only, demonstrate DEBUG is dropped.  
 5 Tests (src/test/java/logging/)  
 • CorrectOrderTest—capture stdout, ensure messages appear in handler order.  
 • DynamicReorderTest—create reversed chain ERROR→DEBUG, verify DEBUG message handled.  
 • UnhandledLevelTest—chain INFO only, send DEBUG, assert nothing printed.  
 6 reflection.md—Discuss loose coupling, runtime flex, and a downside (long chains).

#### **Deliverables**

analysis/cor\_need.md

src/main/java/logging/LogLevel.java

src/main/java/logging/Logger.java

src/main/java/logging/AbstractLogger.java

src/main/java/logging/DebugLogger.java

src/main/java/logging/InfoLogger.java

src/main/java/logging/ErrorLogger.java

src/main/java/logging/LoggingChainBuilder.java

src/main/java/logging/ChainDemo.java

src/test/java/logging/CorrectOrderTest.java

src/test/java/logging/DynamicReorderTest.java

src/test/java/logging/UnhandledLevelTest.java

reflection.md

README.md

### **Solution Sketch**

// LogLevel.java

package logging;

public enum LogLevel {DEBUG,INFO,ERROR}

// Logger.java

package logging;

public interface Logger {void log(String msg, LogLevel lvl);}

// AbstractLogger.java

package logging;

public abstract class AbstractLogger implements Logger{

private AbstractLogger next;

public void setNext(AbstractLogger n){this.next=n;}

public void log(String m,LogLevel l){

if(canHandle(l)) write(m);

else if(next!=null) next.log(m,l);

}

protected abstract boolean canHandle(LogLevel l);

protected abstract void write(String m);

}

// DebugLogger.java

package logging;

public class DebugLogger extends AbstractLogger{

protected boolean canHandle(LogLevel l){return l==LogLevel.DEBUG;}

protected void write(String m){System.out.println("DEBUG: "+m);}

}

// InfoLogger.java and ErrorLogger.java analogous with their level/label

// LoggingChainBuilder.java

package logging;

import java.util.\*;

public class LoggingChainBuilder{

public static AbstractLogger build(List<LogLevel> order){

AbstractLogger head=null,prev=null;

for(LogLevel lv:order){

AbstractLogger curr=switch(lv){

case DEBUG->new DebugLogger();

case INFO ->new InfoLogger();

case ERROR->new ErrorLogger();

};

if(head==null) head=curr;

if(prev!=null) prev.setNext(curr);

prev=curr;

}

return head;

}

}

// ChainDemo.java

package logging;

import java.util.List;

public class ChainDemo{

public static void main(String[] a){

AbstractLogger chain=LoggingChainBuilder.build(List.of(LogLevel.DEBUG,LogLevel.INFO,LogLevel.ERROR));

chain.log("first debug",LogLevel.DEBUG);

chain.log("second info",LogLevel.INFO);

chain.log("third error",LogLevel.ERROR);

System.out.println("--INFO→ERROR only--");

AbstractLogger chain2=LoggingChainBuilder.build(List.of(LogLevel.INFO,LogLevel.ERROR));

chain2.log("skip debug",LogLevel.DEBUG);

chain2.log("show info",LogLevel.INFO);

}

}

**Expected console**

DEBUG: first debug

INFO: second info

ERROR: third error

--INFO→ERROR only--

INFO: show info

### **Reflection (summary)**

The chain removes log-level conditionals from callers, letting new levels or different ordering be added by configuration. Each logger has a single responsibility. Downside: deep chains add slight latency and make tracing harder; instrumentation or shortening the chain mitigates this.