Chain of responsibility pattern is used to achieve loose coupling in software design where a request from client is passed to a chain of objects to process them. Then the object in the chain will decide themselves who will be processing the request and whether the request is required to be sent to the next object in the chain or not.

This pattern comes under behavioral patterns.

In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

Use-Cases : ATM , Vending Machines , Logger Design

public abstract class LogProcessor {  
  
 public static int *INFO* = 1 ;  
 public static int *DEBUG* = 2 ;  
 public static int *ERROR* = 3 ;  
  
 LogProcessor nextLoggerProcessor ;  
  
 public LogProcessor(LogProcessor loggerProcessor ){  
 this.nextLoggerProcessor = loggerProcessor ;  
 }  
  
 public void log(int logLevel , String message ){  
  
 if(nextLoggerProcessor != null ){  
 nextLoggerProcessor.log(logLevel , message );  
 }  
  
 }  
  
  
}

public class InfoLogProcessor extends LogProcessor{  
 public InfoLogProcessor(LogProcessor loggerProcessor) {  
 super(loggerProcessor);  
 }  
  
 public void log(int logLevel , String message ) {  
 *// If it is the current request then handle it here* if(logLevel == *INFO* ){  
 System.*out*.println("INFO : " + message );  
 }else{ *// Otherwise let the parent know so that it will carry to next nested object* super.log(logLevel , message );  
 }  
 }  
  
}

public class DebugLogProcessor extends LogProcessor{  
 public DebugLogProcessor(LogProcessor loggerProcessor) {  
 super(loggerProcessor);  
 }  
  
 public void log(int logLevel , String message ) {  
 *// If it is the current request then handle it here* if(logLevel == *DEBUG* ){  
 System.*out*.println("DEBUG : " + message );  
 }else{ *// Otherwise let the parent know so that it will carry to next nested object* super.log(logLevel , message );  
 }  
 }  
  
}

public class ErrorLogProcessor extends LogProcessor{  
 public ErrorLogProcessor(LogProcessor loggerProcessor) {  
 super(loggerProcessor);  
 }  
  
 public void log(int logLevel , String message ) {  
 *// If it is the current request then handle it here* if(logLevel == *ERROR* ){  
 System.*out*.println("ERROR : " + message );  
 }else{ *// Otherwise let the parent know so that it will carry to next nested object* super.log(logLevel , message );  
 }  
 }  
}

public class ChainOfResponsibilityDesignPattern {  
 public static void main(String[] args) {  
  
 *// This is the chain of objects , at last we have given as null if it does not match with any* LogProcessor logProcessor = new InfoLogProcessor(new DebugLogProcessor( new ErrorLogProcessor(null))) ;  
  
 logProcessor.log(LogProcessor.*ERROR* , "Error Log");  
 logProcessor.log(LogProcessor.*INFO* , "Info Log");  
 logProcessor.log(LogProcessor.*DEBUG* , "Debug Log");  
 logProcessor.log(-1, "Out of request Log"); *// It does not handle any request* }  
}

Output :

ERROR : Error Log

INFO : Info Log

DEBUG : Debug Log

Process finished with exit code 0