Chain Of Responsibility Pattern

In chain of responsibility, sender sends a request to a chain of objects. The request can be handled by any object in the chain.

A Chain of Responsibility Pattern says that just **"avoid coupling the sender of a request to its receiver by giving multiple objects a chance to handle the request".** For example, an ATM uses the Chain of Responsibility design pattern in money giving process.

In other words, we can say that normally each receiver contains reference of another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

Advantage of Chain of Responsibility Pattern

* It reduces the coupling.
* It adds flexibility while assigning the responsibilities to objects.
* It allows a set of classes to act as one; events produced in one class can be sent to other handler classes with the help of composition.

Usage of Chain of Responsibility Pattern:

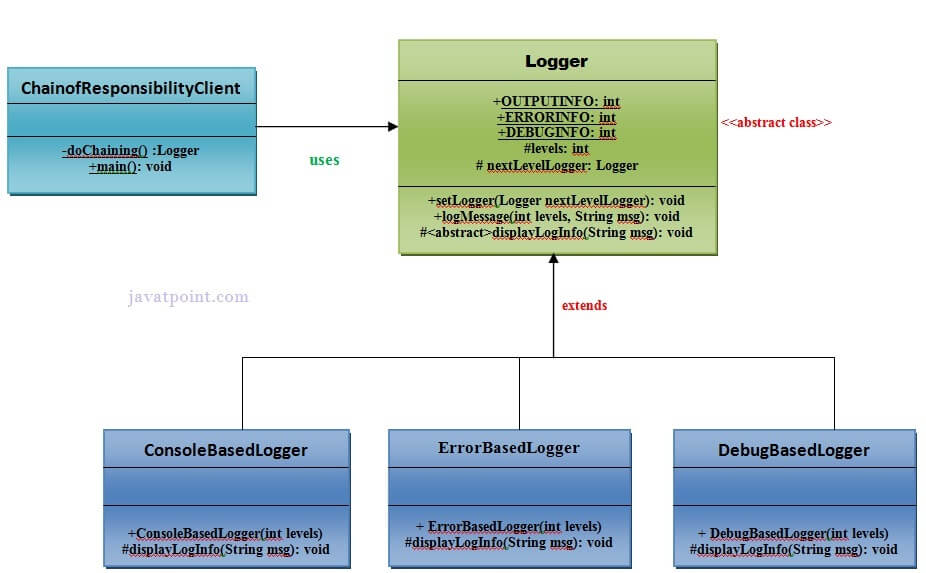
It is used:

* When more than one object can handle a request and the handler is unknown.
* When the group of objects that can handle the request must be specified in dynamic way.

Example of Chain of Responsibility Pattern

Let's understand the example of Chain of Responsibility Pattern by the above UML diagram.

UML for Chain of Responsibility Pattern:



Implementation of above UML:

Step 1

Create a **Logger** abstract class.

1. **public** **abstract** **class** Logger {
2. **public** **static** **int** OUTPUTINFO=1;
3. **public** **static** **int** ERRORINFO=2;
4. **public** **static** **int** DEBUGINFO=3;
5. **protected** **int** levels;
6. **protected** Logger nextLevelLogger;
7. **public** **void** setNextLevelLogger(Logger nextLevelLogger) {
8. **this**.nextLevelLogger = nextLevelLogger;
9. }
10. **public** **void** logMessage(**int** levels, String msg){
11. **if**(**this**.levels<=levels){
12. displayLogInfo(msg);
13. }
14. **if** (nextLevelLogger!=**null**) {
15. nextLevelLogger.logMessage(levels, msg);
16. }
17. }
18. **protected** **abstract** **void** displayLogInfo(String msg);
19. }

Step 2

Create a **ConsoleBasedLogger** class.

*File: ConsoleBasedLogger.java*

1. **public** **class** ConsoleBasedLogger **extends** Logger {
2. **public** ConsoleBasedLogger(**int** levels) {
3. **this**.levels=levels;
4. }
5. @Override
6. **protected** **void** displayLogInfo(String msg) {
7. System.out.println("CONSOLE LOGGER INFO: "+msg);
8. }
9. }

Step 3

Create a **DebugBasedLogger** class.

*File: DebugBasedLogger.java*

1. **public** **class** DebugBasedLogger **extends** Logger {
2. **public** DebugBasedLogger(**int** levels) {
3. **this**.levels=levels;
4. }
5. @Override
6. **protected** **void** displayLogInfo(String msg) {
7. System.out.println("DEBUG LOGGER INFO: "+msg);
8. }
9. }// End of the DebugBasedLogger class.

Step 4

Create a **ErrorBasedLogger** class.

*File: ErrorBasedLogger.java*

1. **public** **class** ErrorBasedLogger **extends** Logger {
2. **public** ErrorBasedLogger(**int** levels) {
3. **this**.levels=levels;
4. }
5. @Override
6. **protected** **void** displayLogInfo(String msg) {
7. System.out.println("ERROR LOGGER INFO: "+msg);
8. }
9. }// End of the ErrorBasedLogger class.

Step 5

Create a **ChainOfResponsibilityClient** class.

*File: ChainofResponsibilityClient.java*

1. **public** **class** ChainofResponsibilityClient {
2. **private** **static** Logger doChaining(){
3. Logger consoleLogger = **new** ConsoleBasedLogger(Logger.OUTPUTINFO);
5. Logger errorLogger = **new** ErrorBasedLogger(Logger.ERRORINFO);
6. consoleLogger.setNextLevelLogger(errorLogger);
8. Logger debugLogger = **new** DebugBasedLogger(Logger.DEBUGINFO);
9. errorLogger.setNextLevelLogger(debugLogger);
11. **return** consoleLogger;
12. }
13. **public** **static** **void** main(String args[]){
14. Logger chainLogger= doChaining();
16. chainLogger.logMessage(Logger.OUTPUTINFO, "Enter the sequence of values ");
17. chainLogger.logMessage(Logger.ERRORINFO, "An error is occured now");
18. chainLogger.logMessage(Logger.DEBUGINFO, "This was the error now debugging is compeled");
19. }
20. }