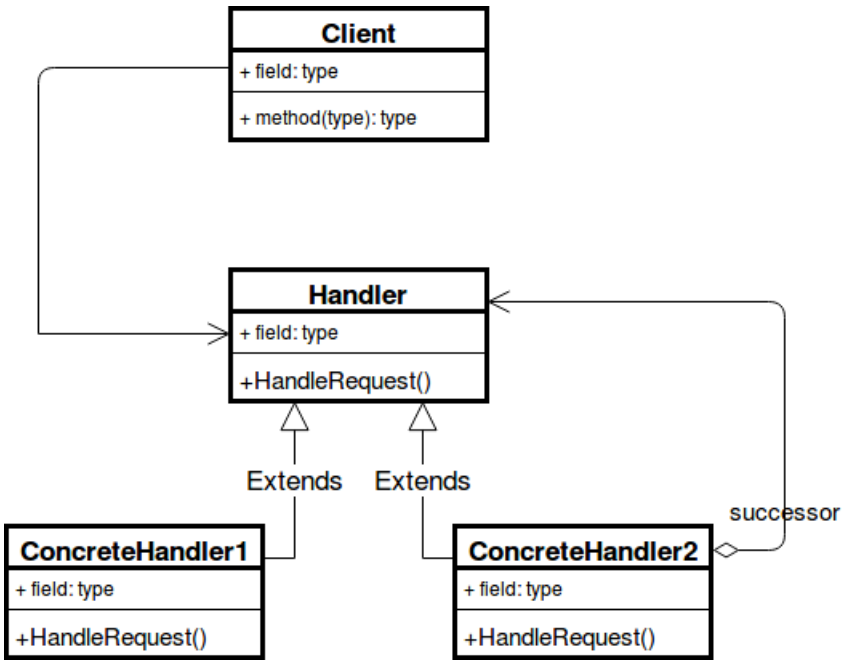
Chain of Responsibility

Famous questions: ATM/Vendor machine, Logger

This design pattern is used to achieve loose coupling between the request's sender and receiver/handler.

By this, for any request from a client, the handler of that request is determined in runtime from all the handler candidates.



Code:

```
// Sachin Mahawar
#include <bits/stdc++.h>
using namespace std;

class Logger
{
protected:
    int INFO = 1;
    int DEBUG = 2;
    int ERROR = 3;

public:
    Logger *successor;
    Logger()
    {
    }

    Logger(Logger *s)
    {
        this->successor = s;
    }

    virtual void log(int type, string msg)
    {
        if (successor != nullptr)
        {
            successor->log(type, msg);
        }
        else
        {
            cout << "Oops request error!\n";
        }
    }
};
```

```
class InfoLogger: public Logger
{
public:
    InfoLogger(Logger *s)
    {
        this->successor = s;
    }

    void log(int type, string msg)
    {
        if (type == this->INFO)
        {
            cout << "INFO: " << msg << "\n";
        }
        else
        {
            cout << "[INFO] forwarding req..\n";
            Logger::log(type, msg);
        }
    }
};

class DebugLogger: public Logger
{
public:
    DebugLogger(Logger *s)
    {
        this->successor = s;
    }

    void log(int type, string msg)
    {
        if (type == this->DEBUG)
        {
            cout << "DEBUG: " << msg << "\n";
        }
        else
        {
            cout << "[DEBUG] forwarding req..\n";
            Logger::log(type, msg);
        }
    }
};

class ErrorLogger: public Logger
{
public:
    ErrorLogger(Logger *s)
    {
        this->successor = s;
    }

    void log(int type, string msg)
    {
        if (type == this->ERROR)
        {
            cout << "ERROR: " << msg << "\n";
        }
        else
        {
            cout << "[ERROR] forwarding req..\n";
            Logger::log(type, msg);
        }
    }
};
```

```
int main()
{
    Logger *logObj = new ErrorLogger(new InfoLogger(new DebugLogger(nullptr)));

    logObj->log(1, "your info");
    cout << "------------------\n";
    logObj->log(2, "debugging");
    cout << "------------------\n";
    logObj->log(3, "exception happens");
    cout << "------------------\n";

    delete logObj;
    return 0;
}
```

Output:

```
[ERROR] forwarding req..
INFO: your info
------------------
[ERROR] forwarding req..
[INFO] forwarding req..
DEBUG: debugging
------------------
ERROR: exception happens
------------------
```

Here logObj is a chain of request handlers/loggers, which are formed (as a chain) during runtime if the type of class doesn’t match with requested.