



[Home](#) / [Design Patterns](#) / [Behavioral patterns](#) / [Chain of Responsibility](#)

Chain of Responsibility in C++



[Back to Chain of Responsibility description](#)

Chain of Responsibility design pattern

1. Put a "next" pointer in the base class
2. The "chain" method in the base class always delegates to the next object
3. If the derived classes cannot handle, they delegate to the base class

```
#include <iostream>
#include <vector>
#include <ctime>
using namespace std;

class Base
{
    Base *next; // 1. "next" pointer in the base class
public:
    Base()
    {
        next = 0;
    }
    void setNext(Base *n)
    {
        next = n;
    }
    void add(Base *n)
    {
        if (next)
            next->add(n);
        else
            next = n;
    }
    // 2. The "chain" method in the base class always delegates to the next obj
    virtual void handle(int i)
    {
        next->handle(i);
    }
};

class Handler1: public Base
{
public:
    /*virtual*/void handle(int i)
    {
        if (rand() % 3)
        {
            // 3. Don't handle requests 3 times out of 4
            cout << "H1 passed " << i << " ";
            Base::handle(i); // 3. Delegate to the base class
        }
        else
            cout << "H1 handled " << i << " ";
    }
};

class Handler2: public Base
```

```
{
    public:
        /*virtual*/void handle(int i)
        {
            if (rand() % 3)
            {
                cout << "H2 passed " << i << " ";
                Base::handle(i);
            }
            else
                cout << "H2 handled " << i << " ";
        }
};

class Handler3: public Base
{
    public:
        /*virtual*/void handle(int i)
        {
            if (rand() % 3)
            {
                cout << "H3 passed " << i << " ";
                Base::handle(i);
            }
            else
                cout << "H3 handled " << i << " ";
        }
};

int main()
{
    srand(time(0));
    Handler1 root;
    Handler2 two;
    Handler3 thr;
    root.add(&two);
    root.add(&thr);
    thr.setNext(&root);
    for (int i = 1; i < 10; i++)
    {
        root.handle(i);
        cout << '\n';
    }
}
```

Output

```

H1 passed 1  H2 passed 1  H3 passed 1  H1 passed 1  H2 handled 1
H1 handled 2
H1 handled 3
H1 passed 4  H2 passed 4  H3 handled 4
H1 passed 5  H2 handled 5
H1 passed 6  H2 passed 6  H3 passed 6  H1 handled 6
H1 passed 7  H2 passed 7  H3 passed 7  H1 passed 7  H2 handled 7
H1 handled 8
H1 passed 9  H2 passed 9  H3 handled 9

```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



♥ Learn more

Code examples

Java	Chain of Responsibility in Java: Before and after	Chain of Responsibility in Java
C++	Chain of Responsibility in C++	Chain of Responsibility in C++: Chain and Composite
PHP	Chain of Responsibility in PHP	

[Design Patterns](#)

[AntiPatterns](#)

[Refactoring](#)

[UML](#)

[My account](#)

[Forum](#)

[Contact us](#)

[About us](#)

© 2007-2018 SourceMaking.com
All rights reserved.

[Terms / Privacy policy](#)