



[Home](#) / [Design Patterns](#) / [Structural patterns](#) / [Decorator](#)

# Decorator in C++



[Back to \*\*Decorator\*\* description](#)

## Decorator design pattern

1. Create a "lowest common denominator" that makes classes interchangeable
2. Create a second level base class for optional functionality
3. "Core" class and "Decorator" class declare an "isa" relationship
4. Decorator class "has a" instance of the "lowest common denominator"
5. Decorator class delegates to the "has a" object
6. Create a Decorator derived class for each optional embellishment
7. Decorator derived classes delegate to base class AND add extra stuff
8. Client has the responsibility to compose desired configurations

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

// 1. "lowest common denominator"
class Widget
{
public:
    virtual void draw() = 0;
};

class TextField: public Widget
{
    // 3. "Core" class & "is a"
    int width, height;
public:
    TextField(int w, int h)
    {
        width = w;
        height = h;
    }

    /*virtual*/
    void draw()
    {
        cout << "TextField: " << width << ", " << height << '\n';
    }
};

// 2. 2nd level base class
class Decorator: public Widget // 4. "is a" relationship
{
    Widget *wid; // 4. "has a" relationship
public:
    Decorator(Widget *w)
    {
        wid = w;
    }

    /*virtual*/
    void draw()
    {
        wid->draw(); // 5. Delegation
    }
};

class BorderDecorator: public Decorator
{
public:
```

```
// 6. Optional embellishment
BorderDecorator(Widget *w): Decorator(w){}

/*virtual*/
void draw()
{
    // 7. Delegate to base class and add extra stuff
    Decorator::draw();
    cout << "    BorderDecorator" << '\n';
}
};

class ScrollDecorator: public Decorator
{
public:
    // 6. Optional embellishment
    ScrollDecorator(Widget *w): Decorator(w){}

    /*virtual*/
    void draw()
    {
        // 7. Delegate to base class and add extra stuff
        Decorator::draw();
        cout << "    ScrollDecorator" << '\n';
    }
};

int main()
{
    // 8. Client has the responsibility to compose desired configurations
    Widget *aWidget = new BorderDecorator(new BorderDecorator(new ScrollDecorator
        (new TextField(80, 24))));
    aWidget->draw();
}
```

## Output

```
TextField: 80, 24
    ScrollDecorator
    BorderDecorator
    BorderDecorator
```

## 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	Decorator in Java	Decorator in Java	Decorator in Java	Decorator in Java
C++	Decorator in C++: Before and after	Decorator in C++	Decorator in C++: Encoding and decoding layers of header/packet/trailer	
PHP	Decorator in PHP			
Delphi	Decorator in Delphi			
Python	Decorator in			

Design Patterns  
AntiPatterns  
Refactoring  
UML

My account  
Forum  
Contact us  
About us

