Decorator in C++: Before and after

Read full article

See the code

contents

Design Patterns

Before

01 Inheritance run amok.

02

```C++
class A {
  public:
    virtual void do_it() {
        cout << 'A';
    }
};

class AwithX: public A {
  public:
```

```
    /*virtual*/
    void do_it() {
        A::do_it();
        do_X();
    };
  private:
    void do_X() {
        cout << 'X';
    }
};

class AwithY: public A {
  public:
    /*virtual*/
    void do_it() {
        A::do_it();
        do_Y();
    }
  protected:
    void do_Y() {
        cout << 'Y';
    }
};

class AwithZ: public A {
  public:
    /*virtual*/
    void do_it() {
        A::do_it();
        do_Z();
    }
  protected:
    void do_Z() {
        cout << 'Z';
    }
};
```

```
class AwithXY: public AwithX, public AwithY
{
  public:
    /*virtual*/
    void do_it() {
        AwithX::do_it();
        AwithY::do_Y();
    }
};

class AwithXYZ: public AwithX, public AwithY, public AwithZ
{
  public:
    /*virtual*/
    void do_it() {
        AwithX::do_it();
        AwithY::do_Y();
        AwithZ::do_Z();
    }
};

int main() {
  AwithX anX;
  AwithXY anXY;
  AwithXYZ anXYZ;
  anX.do_it();
  cout << '\n';
  anXY.do_it();
  cout << '\n';
  anXYZ.do_it();
  cout << '\n';
}
```

Output

```
AX
AXY
AXYZ
```

After

Replacing inheritance with wrapping-delegation

Discussion. Use aggregation instead of inheritance to implement embellishments to a “core” object. Client can dynamically compose permutations, instead of the architect statically wielding multiple inheritance.

```C++
class I {
  public:
    virtual ~I(){}
    virtual void do_it() = 0;
};

class A: public I {
  public:
    ~A() {
        cout << "A dtor" << '\n';
    }
    /*virtual*/
    void do_it() {
        cout << 'A';
    }
};

class D: public I {
```

```
  public:
    D(I *inner) {
        m_wrappee = inner;
    }
    ~D() {
        delete m_wrappee;
    }
    /*virtual*/
    void do_it() {
        m_wrappee->do_it();
    }
  private:
    I *m_wrappee;
};

class X: public D {
  public:
    X(I *core): D(core){}
    ~X() {
        cout << "X dtor" << "   ";
    }
    /*virtual*/
    void do_it() {
        D::do_it();
        cout << 'X';
    }
};

class Y: public D {
  public:
    Y(I *core): D(core){}
    ~Y() {
        cout << "Y dtor" << "   ";
    }
    /*virtual*/
    void do_it() {
```

```
        D::do_it();
        cout << 'Y';
    }
};

class Z: public D {
  public:
    Z(I *core): D(core){}
    ~Z() {
        cout << "Z dtor" << "   ";
    }
    /*virtual*/
    void do_it() {
        D::do_it();
        cout << 'Z';
    }
};

int main() {
  I *anX = new X(new A);
  I *anXY = new Y(new X(new A));
  I *anXYZ = new Z(new Y(new X(new A)));
  anX->do_it();
  cout << '\n';
  anXY->do_it();
  cout << '\n';
  anXYZ->do_it();
  cout << '\n';
  delete anX;
  delete anXY;
  delete anXYZ;
}
```

Output

```
AX
```

```
AXY
AXYZ
X dtor   A dtor
Y dtor   X dtor   A dtor
Z dtor   Y dtor   X dtor   A dtor
```

List of Decorator examples

C# examples

- Decorator in C#

C++ examples

- Decorator in C++: Encoding and decoding layers of header/packet/trailer
- Decorator in C++
- Decorator in C++: Before and after <=[You are here]

Delphi examples

- Decorator in Delphi

Java examples

- Decorator in Java
- Decorator in Java
- Decorator in Java
- Decorator in Java

PHP examples

- Decorator in PHP