Նախագծման Ձևանմուշներ: Decorator

Հրաչյա Թանդիլյան

2020

Decorator

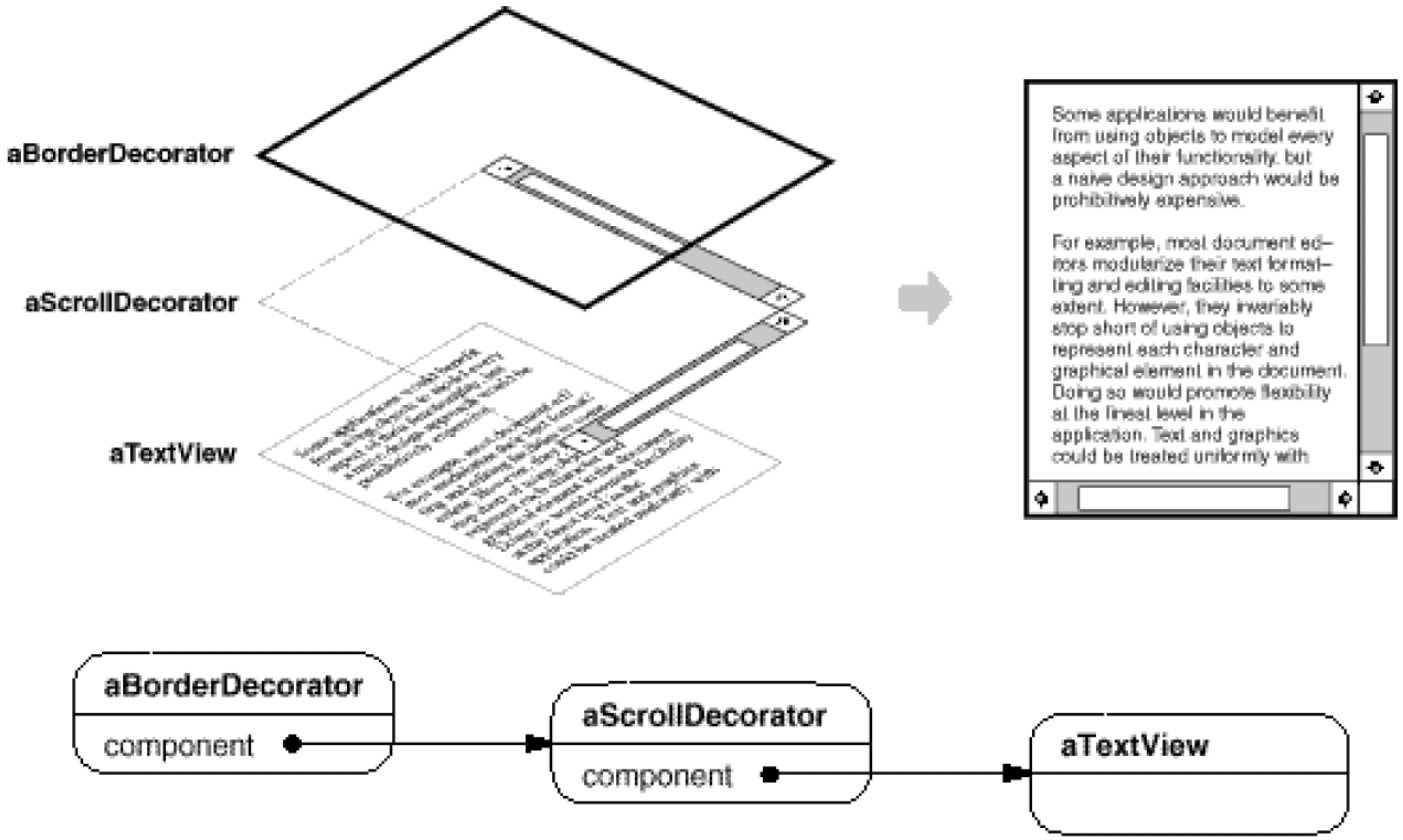
Նպատակը

Օբյեկտին դինամիկ կերպով հավելյալ պատասխանատվություն է կցում: Հանդիսանում է ընդլայնման նպատակով ժառանգման առավել ճկուն այլընտրանք:

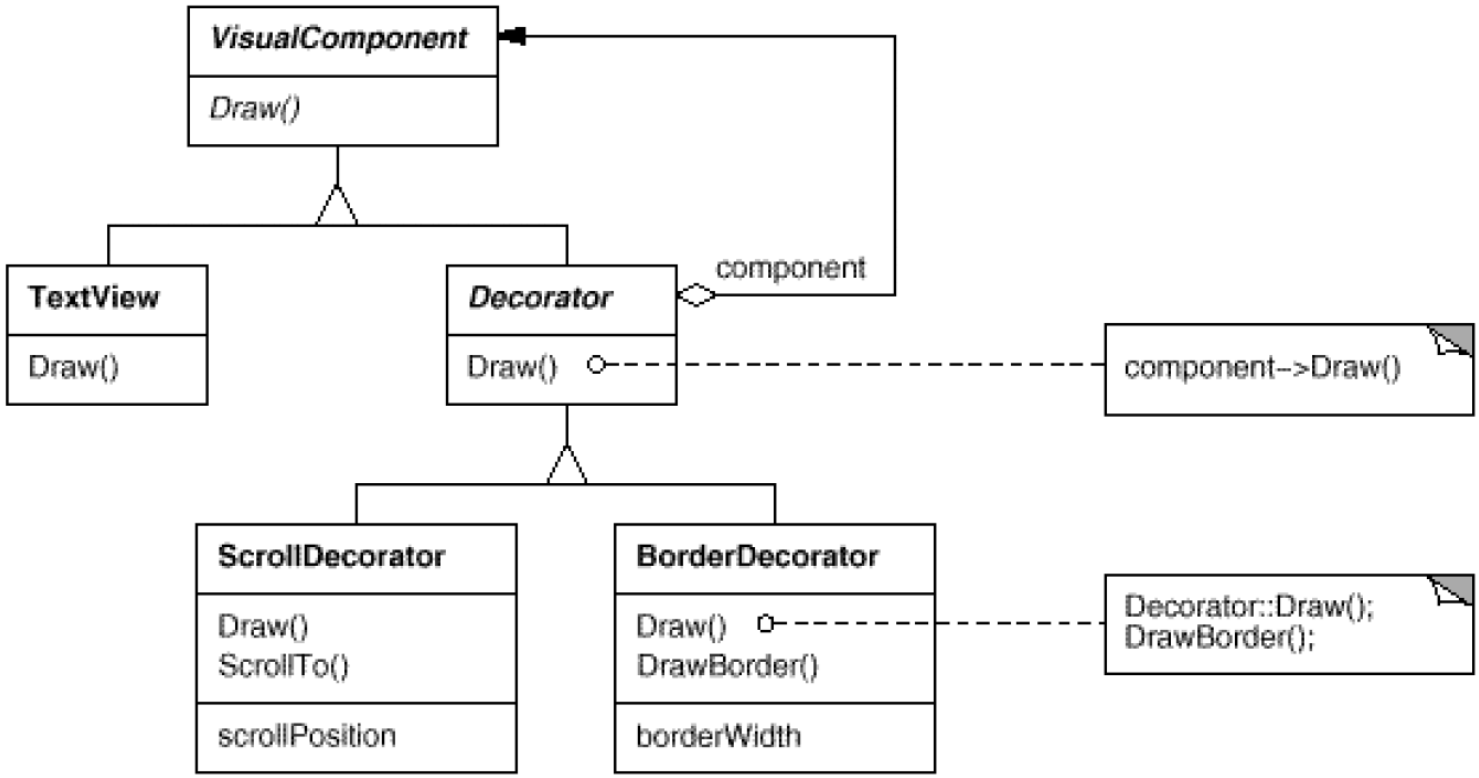
Նաև հայտնի է որպես

- Wrapper

Մոտիվացիան


aBorderDecorator
aScrollDecorator
aTextView
Some applications would benefit from using objects to model every aspect of their functionality, but a naive design approach would be prohibitively expensive.
For example, most document editors modularize their text formatting and editing facilities to some extent. However, they invariably stop short of using objects to represent each character and graphical element in the document. Doing so would promote flexibility at the finest level in the application. Text and graphics could be treated uniformly with
aBorderDecorator
component
aScrollDecorator
component
aTextView

Մոտիվացիան


VisualComponent
Draw()
TextView
Draw()
Decorator
Draw()
component
component->Draw()
ScrollDecorator
Draw()
ScrollTo()
scrollPosition
BorderDecorator
Draw()
DrawBorder()
borderWidth
Decorator::Draw();
DrawBorder();

Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

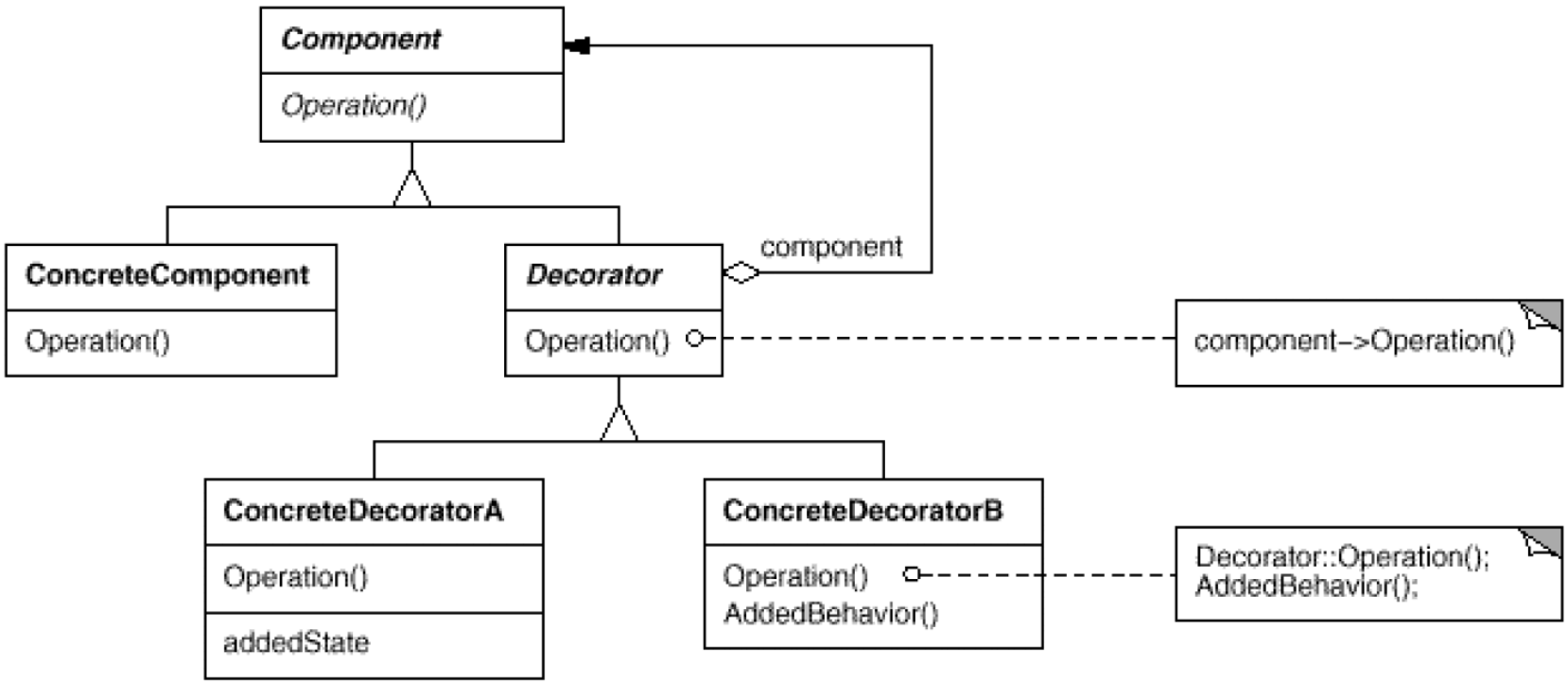
Ա Անհրաժեշտ է անհատական օբյեկտի դինամիկ և թափանցիկ կերպով հավելյալ պատասխանատվություն տալ առանց մնացյալ օբյեկտների վրա ազդելու:

Բ Անհրաժեշտ է օբյեկտի այնպիսի պատասխանատվություն տալ, որը հնարավոր լինի չեղարկել:

Գ Երբ ժառանգման միջոցով ընդլայնումը կիրառելի չէ.

- Երբ մեծ քանակով անկախ ընդլայնումների կարիք կա, բայց բոլոր կոմբինացիաների համար ժառանգ դասերի ստեղծումը կբերի դասերի քանակի չափից ավել աճի:
- Երբ դասի սահմանումը թաքնված է կամ հասանելի չէ ժառանգման համար:

Կառուցվածքը



Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Առավել ճկուն է քան ստատիկ ժառանգումը:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Ա Առավել ճկուն է քան ստատիկ ժառանգումը:
- Բ Խուսափում է հատկություններով հարուստ դասերի հիերարխիայով վեր բարձրացումից:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Առավել ճկուն է քան ստատիկ ժառանգումը:

Բ Խուսափում է հատկություններով հարուստ դասերի հիերարխիայով վեր բարձրացումից:

Գ Դեկորատորը և նրա կոմպոնենտը միևնույն օբյեկտը չեն:

Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Առավել ճկուն է քան ստատիկ ժառանգումը:

Բ Խուսափում է հատկություններով հարուստ դասերի հիերարխիայով վեր բարձրացումից:

Գ Դեկորատորը և նրա կոմպոնենտը միևնույն օբյեկտը չեն:

Դ Բերում է մեծ քանակով փոքր օբյեկտների ստեղծման:

Իրականացումը

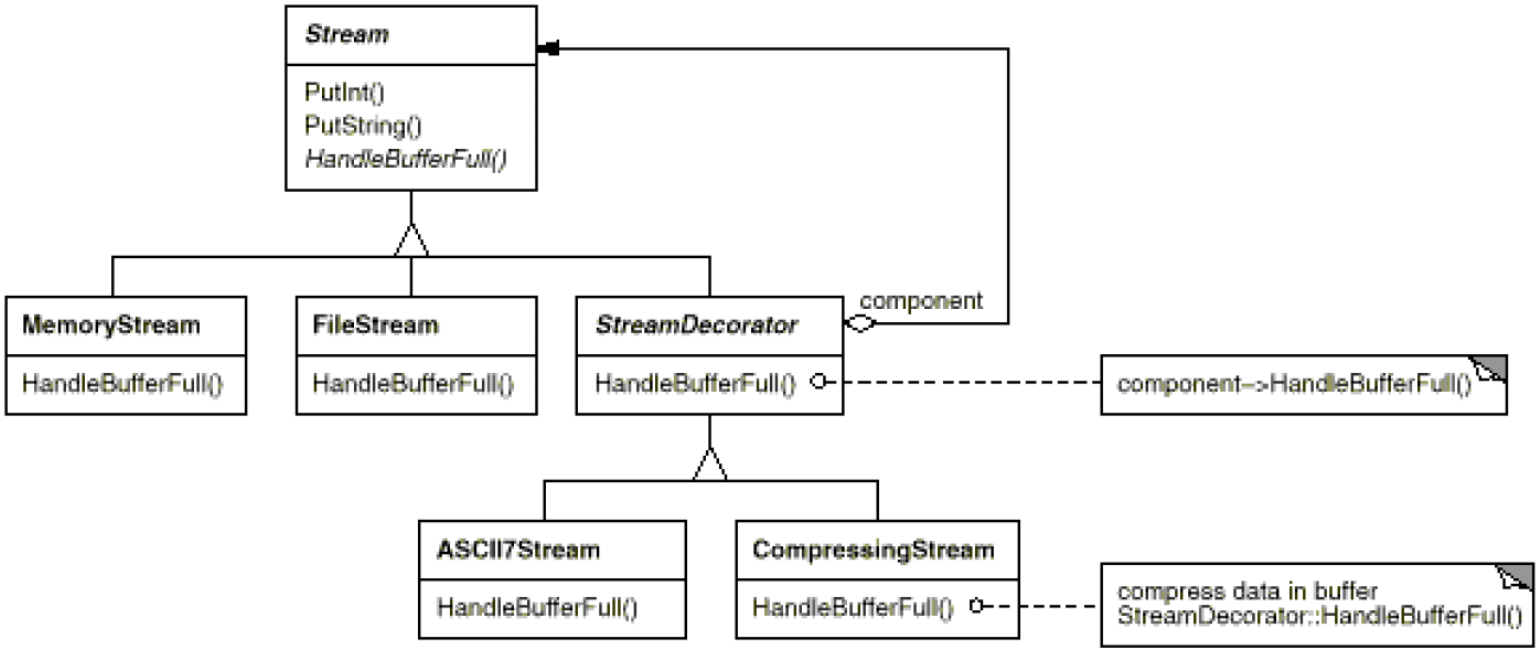
Ա Ինտերֆեյսի պահպանում:

Բ Աբստրակտ դեկորատոր դասի բացակայություն:

Գ Կոմպոնենտ դասերի պարզ (lightweight) իրականացում:

Դ Օբյեկտի արտաքինի փոխարինումն ի տարբերություն օբյեկտի ներքինի փոխարինման:

Օրինակ


Stream
PutInt()
PutString()
HandleBufferFull()
MemoryStream
HandleBufferFull()
FileStream
HandleBufferFull()
StreamDecorator
HandleBufferFull()
component
component->HandleBufferFull()
ASCII7Stream
HandleBufferFull()
CompressingStream
HandleBufferFull()
compress data in buffer
StreamDecorator::HandleBufferFull()

Օրինակ

```
class VisualComponent {

public:
    VisualComponent();
    virtual void Draw();
    virtual void Resize();
};

class Decorator : public VisualComponent {

public:
    Decorator(VisualComponent*);

    virtual void Draw() { component->Draw(); }
    virtual void Resize() { component->Resize(); }

private:
    VisualComponent* component;
};
```

Օրինակ

```
class BorderDecorator : public Decorator {

public:
    BorderDecorator(VisualComponent*, int borderWidth);

    virtual void Draw() {
        Decorator::Draw();
        DrawBorder(width);
    }

private:
    void DrawBorder(int);
    int width;
};
```

Օրինակ

```
Window* window = new Window;

TextView* textView = new TextView;

window->SetContents( new BorderDecorator(
                        new ScrollDecorator( textView ), 1) );
```

Առնչվող Նախագծման Ձևանմուշները

- Adapter
- Composite
- Strategy