Նախագծման Ձևանմուշներ: Flyweight

Հրաչյա Թանդիլյան

2020

Flyweight

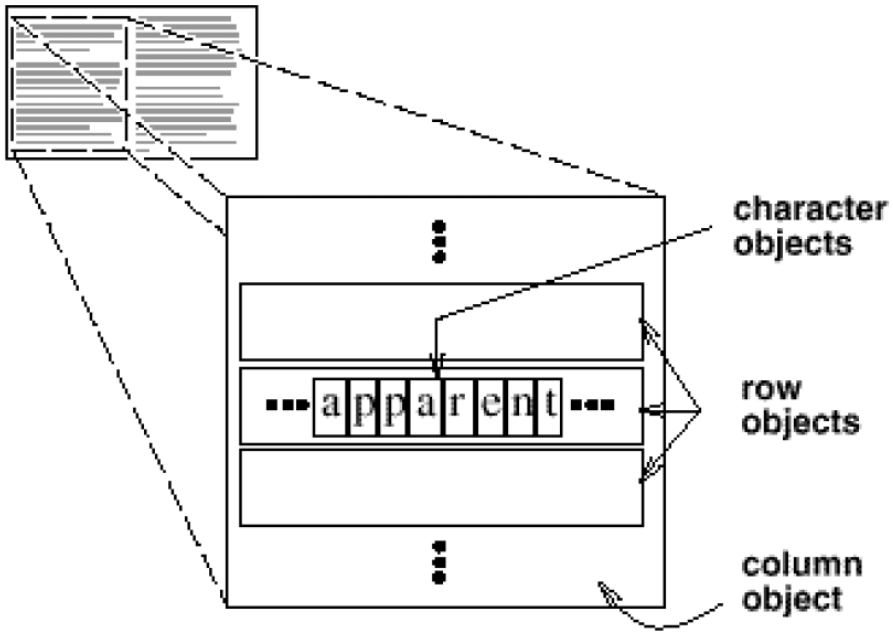
Նպատակը

Մեծ քանակով փոքր օբյեկտներ արդյունավետ կերպով մշակելու համար կիրառել համատեղ օգտագործում (sharing):

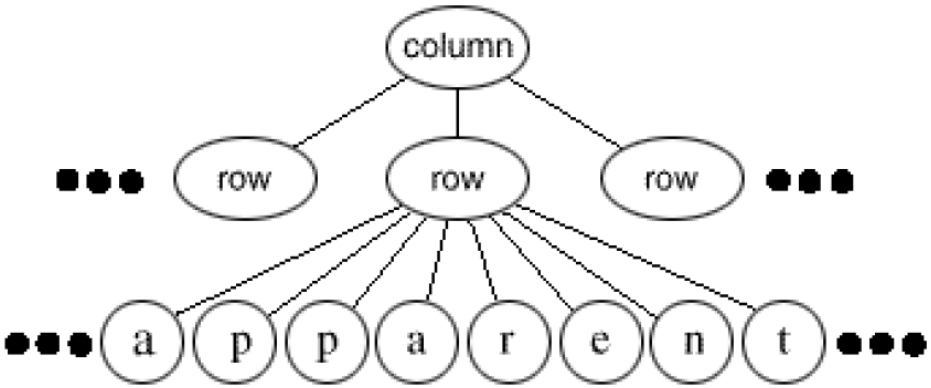
Նաև հայտնի է որպես

- Այլ լայնորեն կիրառվող անուններ չկան:

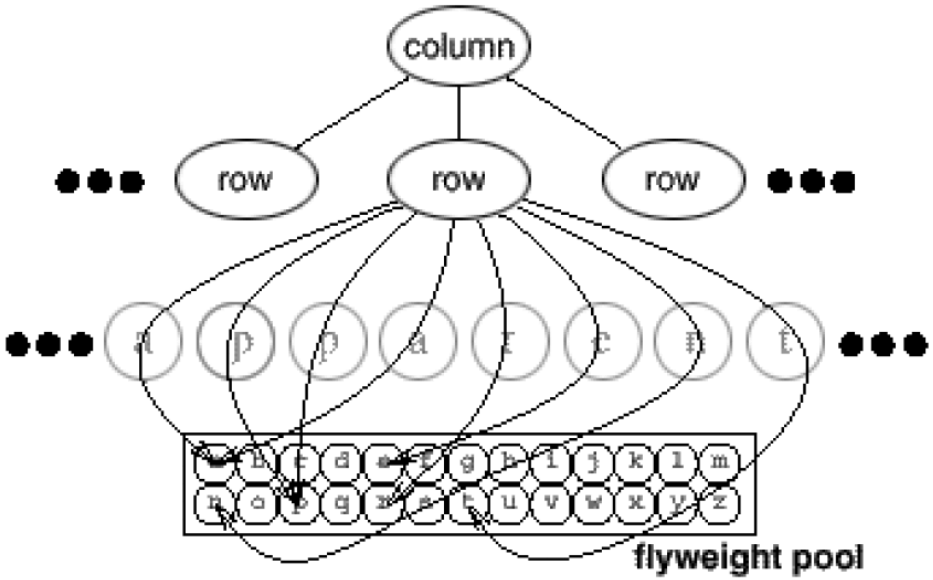
Մոտիվացիան



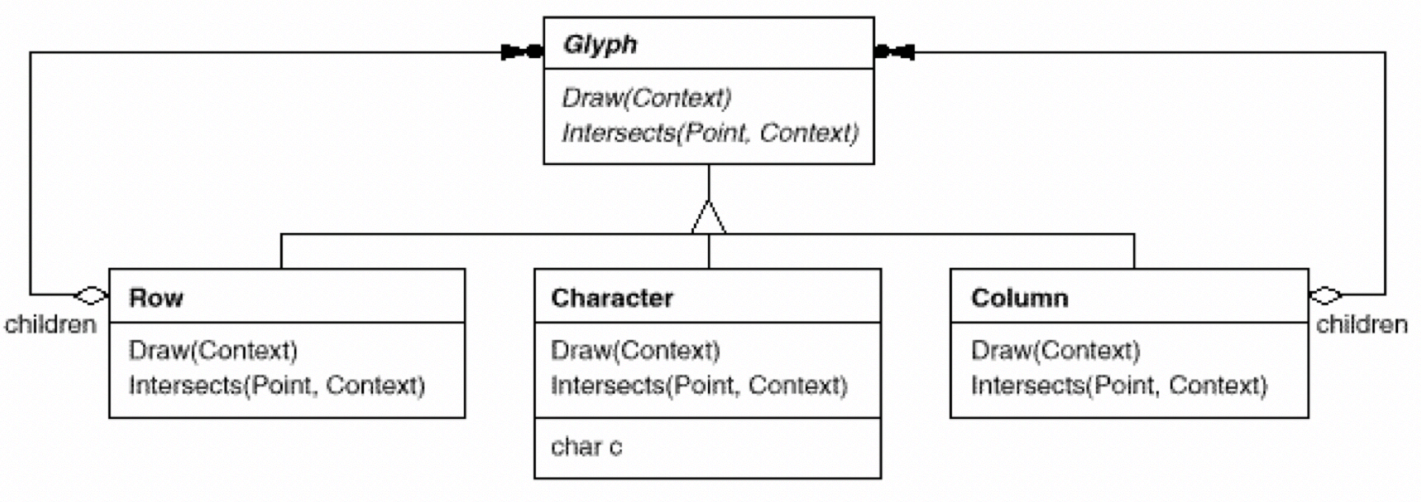
Մոտիվացիան



Մոտիվացիան



Մոտիվացիան

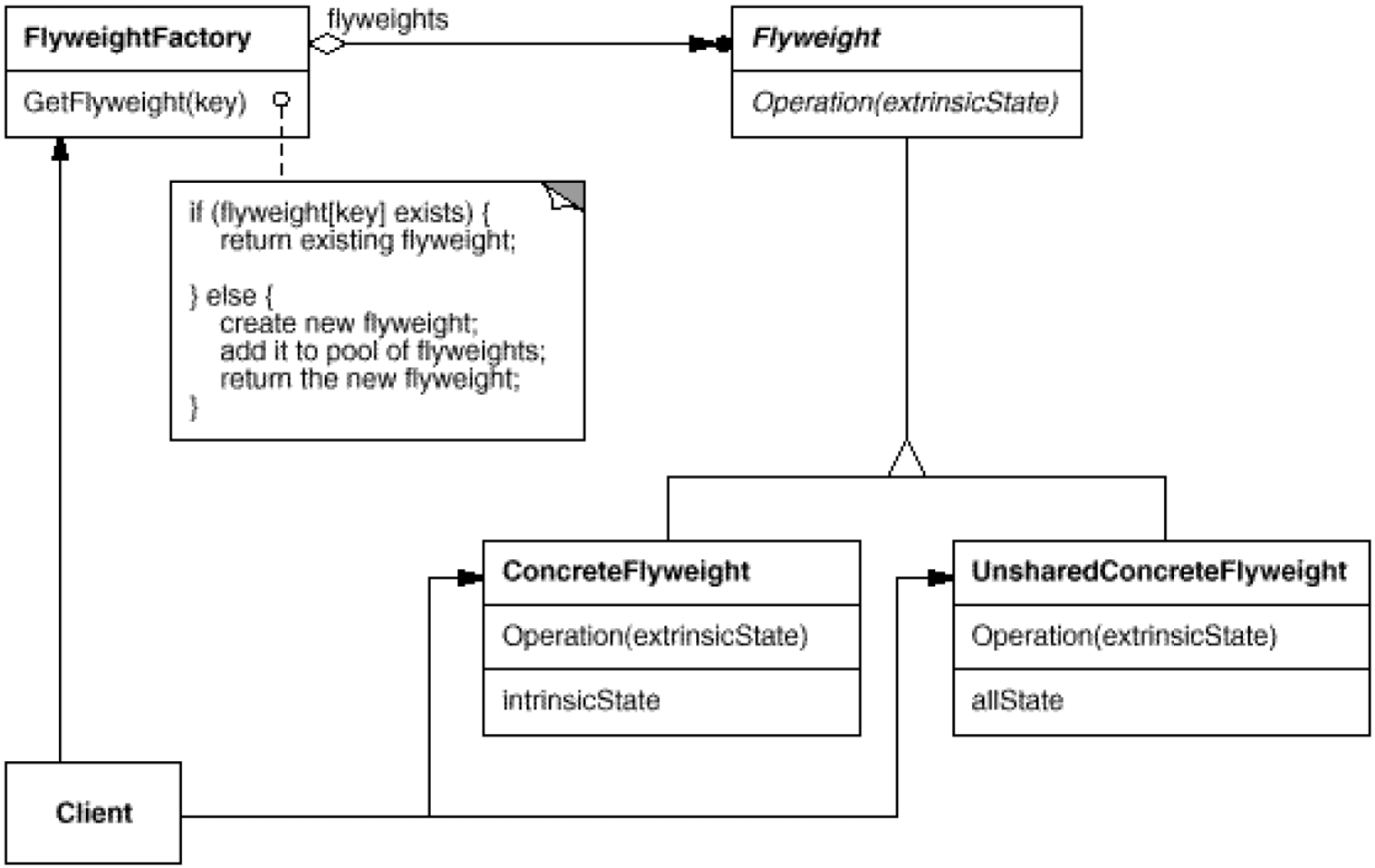

Glyph
Draw(Context)
Intersects(Point, Context)
Row
Draw(Context)
Intersects(Point, Context)
Character
Draw(Context)
Intersects(Point, Context)
char c
Column
Draw(Context)
Intersects(Point, Context)
children
children

Կիրառելիությունը

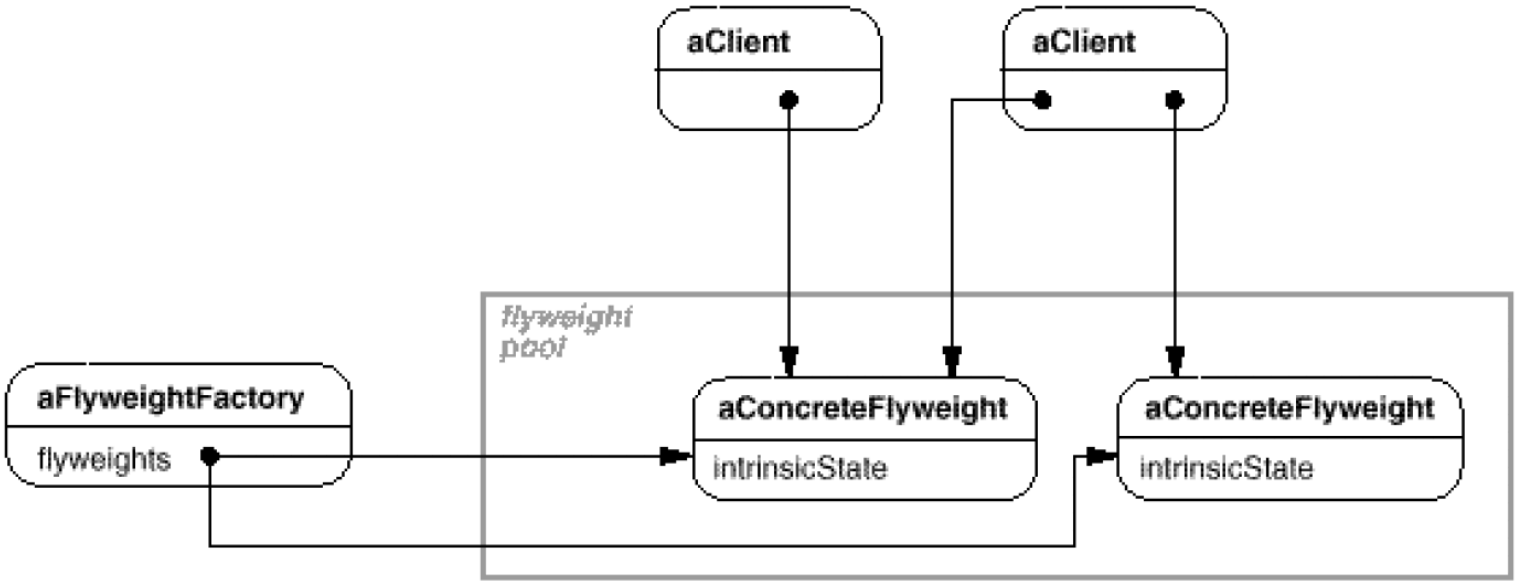
Այս Ն.Ձ. պետք է օգտագործել երբ հետևյալ բոլոր պայմանները տեղի ունեն.

- Ա Կիրառությունը մեծ քանակով օբյեկտներ է օգտագործում:
- Բ Շատ հիշողություն է օգտագործվում, զուտ օբյեկտների քանակի մեծ լինելու պատճառով:
- Գ Օբյեկտների վիճակների մեծ մասը կարելի է արտաքին դարձնել:
- Դ Արտաքին վիճակների հեռացումից հետո օբյեկտների շատ խմբեր կարող են փոխարինվել համեմատաբար քիչ քանակով համատեղ օգտագործմամբ օբյեկտներով:
- Ե Կիրառությունը կախված չէ օբյեկտների ինքնությունից (չի կատարում օբյեկտների հասցեների համեմատում):

Կառուցվածքը



Կառուցվածքը



Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

Ա Այս Ն.Ձ. կարող է առաջացնել արագության վատթարացում , որը լրացվում է ծախսված հիշողության նվազեցմամբ:

Բ Հիշողության օգտագործման նվազեցումը ֆունկիցա է հետևյալ գործոններից.

- Ընդհանուր օգտագործման շնորհիվ կրճատված օբյեկտների թվաքանակից:
- Օբյեկտի ներքին վիճակների քանակից:
- Այն փաստից թե արդյոք օբյեկտի արտաքին վիճակը հաշվարկվում է, թե պահվում:

Գ Բերում է օգտագործված հիշողության երկակի նվազեցում:

Դ Հաճախ այս Ն.Ձ. կիրառվում է Composite Ն.Ձ. հետ համատեղ ընդհանուր տերևային գագաթներով գրաֆներ ներկայացնելու համար:

Իրականացումը

Ա. Արտաքին վիճակների հեռացում:

Բ. Ընդհանուր օգտագործման օբյեկտների ղեկավարում:

Օրինակ

```
class Glyph {

public:
    virtual ~Glyph();
    virtual void Draw(Window*, GlyphContext&);
    virtual void SetFont(Font*, GlyphContext&);
    virtual Font* GetFont(GlyphContext&);
    virtual void First(GlyphContext&);
    virtual void Next(GlyphContext&);
    virtual bool IsDone(GlyphContext&);
    virtual Glyph* Current(GlyphContext&);
    virtual void Insert(Glyph*, GlyphContext&);
    virtual void Remove(GlyphContext&);

protected:
    Glyph();
};
```

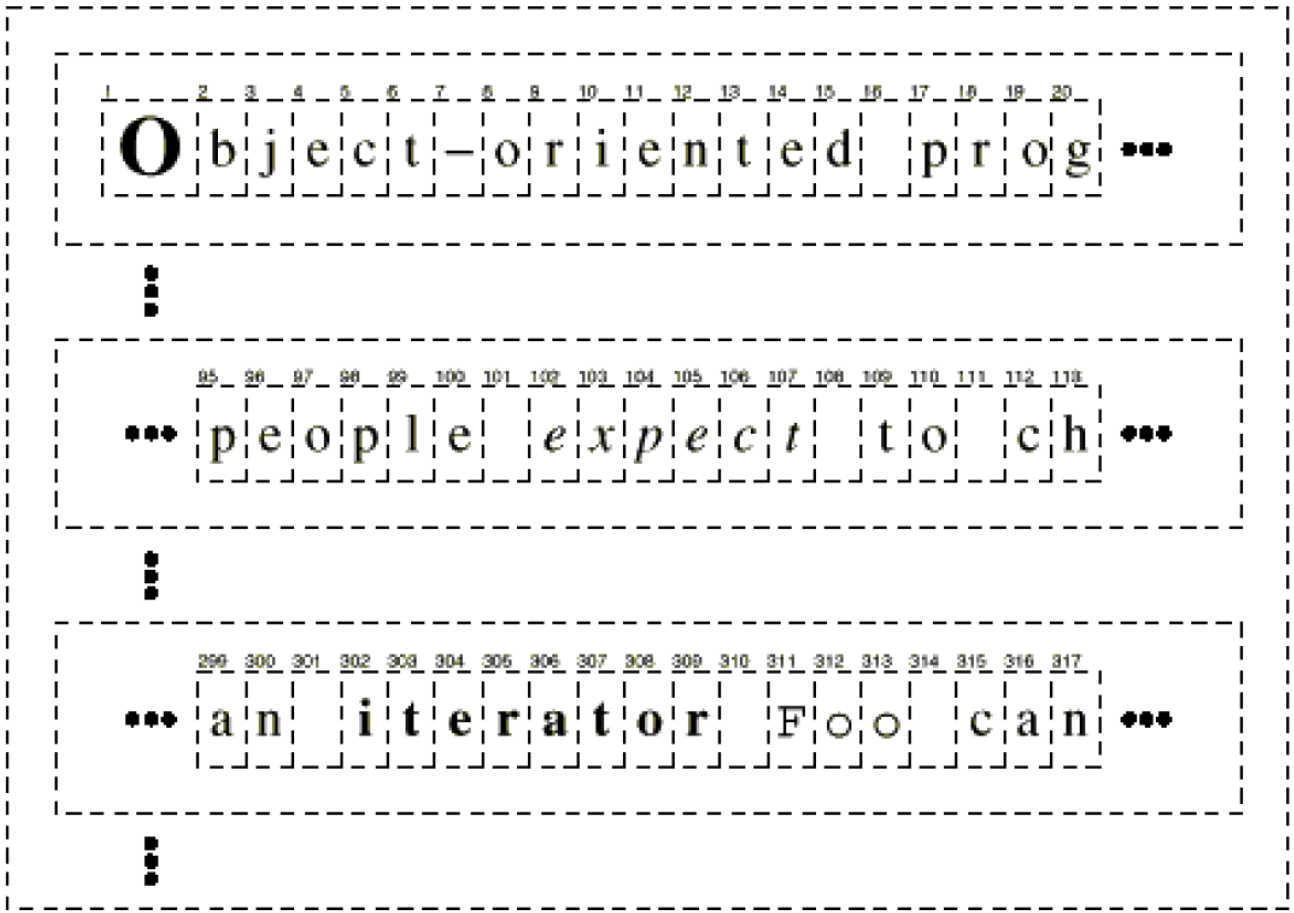
Օրինակ

```
class Character : public Glyph {

public:
    Character(char);
    virtual void Draw(Window*, GlyphContext&);

private:
    char charcode;
};
```

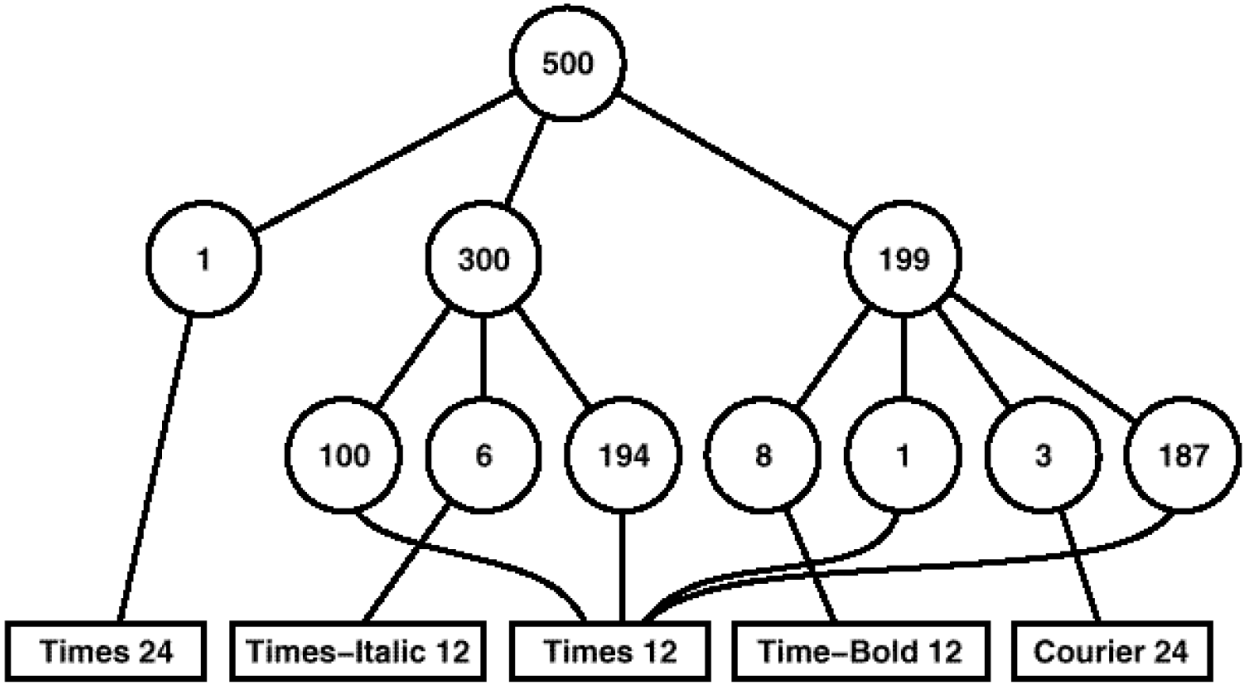
Օրինակ

```
class GlyphContext {

public:
    GlyphContext();
    virtual ~GlyphContext();
    virtual void Next(int step = 1);
    virtual void Insert(int quantity = 1);
    virtual Font* GetFont();
    virtual void SetFont(Font*, int span = 1);

private:
    int index;
    BTree* fonts;
};
```

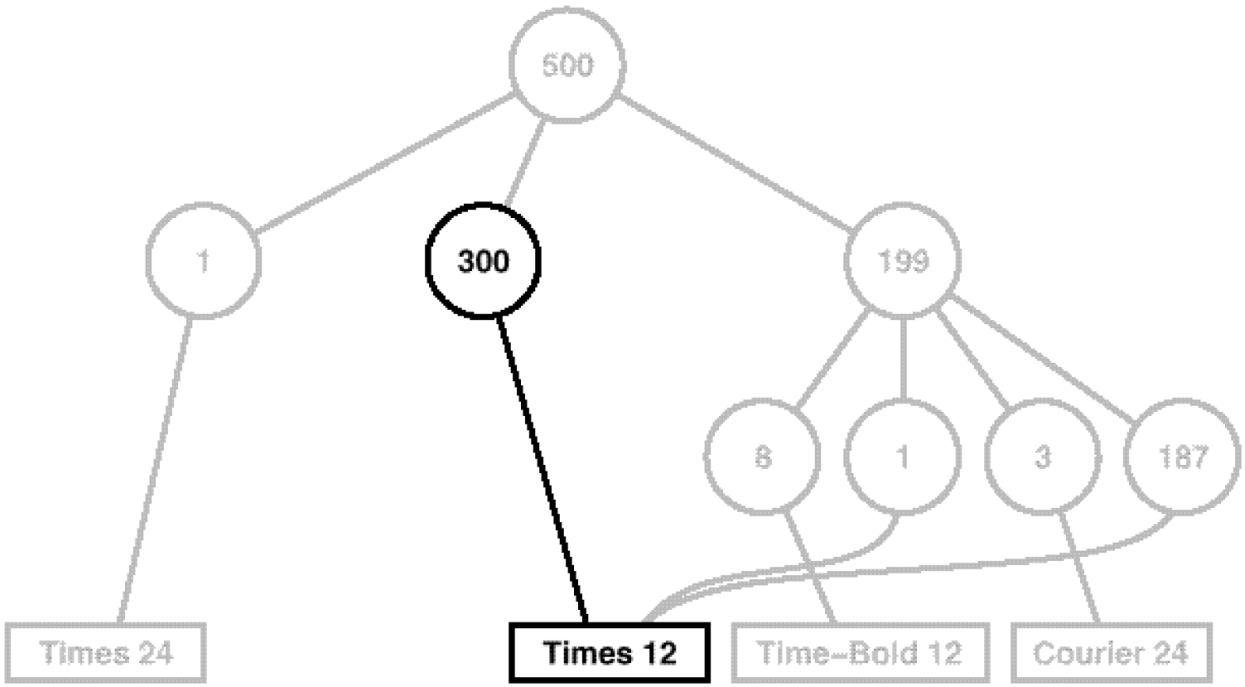
Օրինակ


Object-oriented prog
people expect to ch
an iterator Foo can

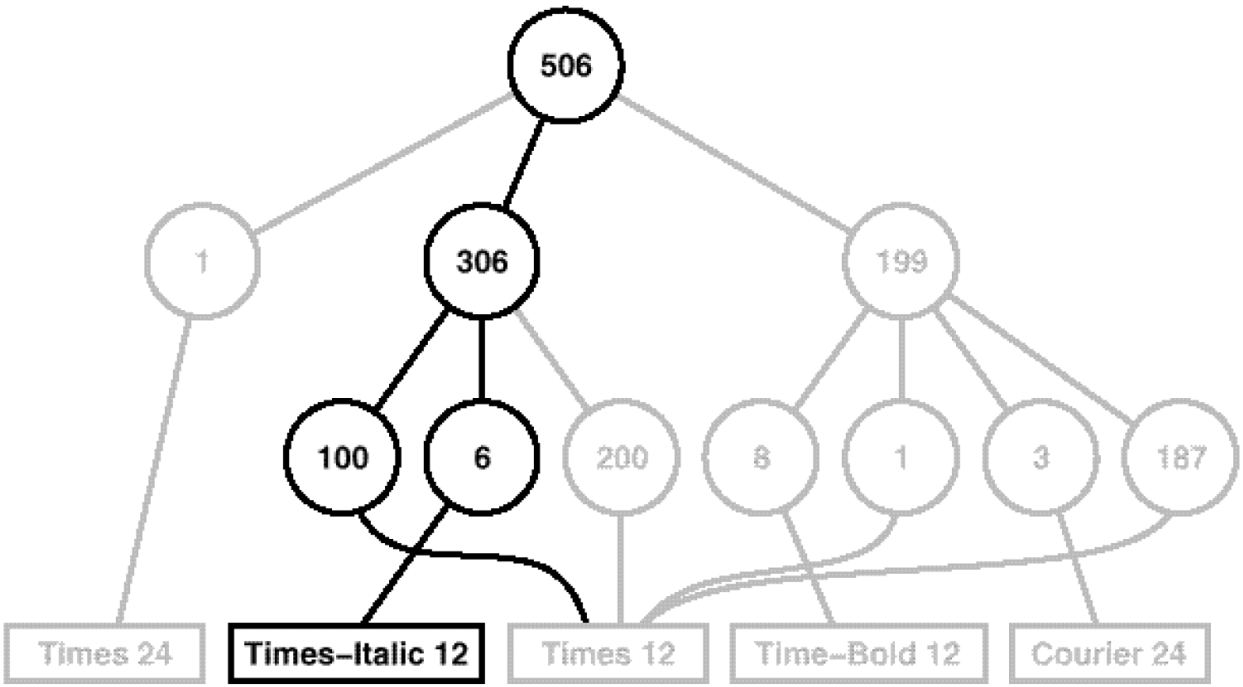
Օրինակ


500
1
300
199
100
6
194
8
1
3
187
Times 24
Times–Italic 12
Times 12
Time–Bold 12
Courier 24

Օրինակ



Օրինակ


506
1
306
199
100
6
200
8
1
3
187
Times 24
Times–Italic 12
Times 12
Time–Bold 12
Courier 24

Օրինակ

```
class GlyphFactory {

    static const int NCHARCODES = 128;

public:
    GlyphFactory();
    virtual ~GlyphFactory();
    virtual Character* CreateCharacter(char);
    virtual Row* CreateRow();
    virtual Column* CreateColumn();
    // other methods

private:
    Character* character[NCHARCODES];
};
```

Օրինակ

```
GlyphFactory::GlyphFactory() {
    for (int i = 0; i < NCHARCODES; ++i) {
        character[i] = 0;
    }
}

Character* GlyphFactory::CreateCharacter(char c) {
    if (!character[c]) {
        character[c] = new Character(c);
    }
    return character[c];
}

Row* GlyphFactory::CreateRow() {
    return new Row;
}

Column* GlyphFactory::CreateColumn() {
    return new Column;
}
```

Առնչվող Նախագծման Ձևանմուշները

- Composite
- State
- Strategy