Flyweight 패턴

타입

Structural Pattern

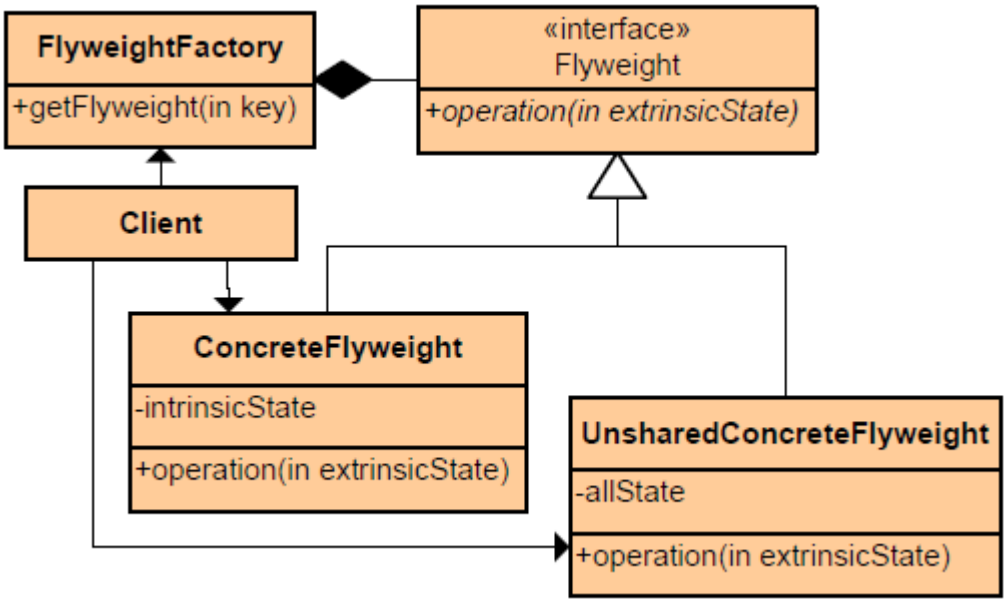
문제

성질이 비슷한 객체가 너무 많아 메모리를 쓸데없이 많이 잡아먹는다.

해결

공유되는 자원(intrinsic state)과 공유되지 않는 자원(extrinsic state)을 나누고 객체를 선언할 때에는 extrinsic state만 사용하도록 한다.

클래스 다이어그램



예제

flyweight1.cpp

```
#include <iostream>
 #include <string>
 #include <vector>
```

```
#define NUMBER_OF_SAME_TYPE_CHARS 3;

/* Actual flyweight objects class (declaration) */
class FlyweightCharacter;

/*
   FlyweightCharacterAbstractBuilder is a class holding the properties
which are shared by
many objects. So instead of keeping these properties in those objects
we keep them externally making
   objects flyweight. See more details in the comments of main
function.
*/
class FlyweightCharacterAbstractBuilder {
   FlyweightCharacterAbstractBuilder() {}
   ~FlyweightCharacterAbstractBuilder() {}
public:
   static std::vector<float> fontSizes; // lets imagine that sizes be
may of floating point type
   static std::vector<std::string> fontNames; // font name may be of
variable length (lets take 6 bytes is average)

static void setFontsAndNames();
   static FlyweightCharacter createFlyweightCharacter(unsigned short
fontSizeIndex,
        unsigned short fontNameIndex,
        unsigned short positionInStream);
};

std::vector<float> FlyweightCharacterAbstractBuilder::fontSizes(3);
std::vector<std::string>
FlyweightCharacterAbstractBuilder::fontNames(3);
void FlyweightCharacterAbstractBuilder::setFontsAndNames() {
   fontSizes[0] = 1.0;
   fontSizes[1] = 1.5;
   fontSizes[2] = 2.0;

   fontNames[0] = "first_font";
   fontNames[1] = "second_font";
   fontNames[2] = "third_font";
}

class FlyweightCharacter {
   unsigned short fontSizeIndex; // index instead of actual font size
   unsigned short fontNameIndex; // index instead of font name

   unsigned positionInStream;

public:
```

```
    FlyweightCharacter(unsigned short fontSizeIndex, unsigned short
fontNameIndex, unsigned short positionInStream):
        fontSizeIndex(fontSizeIndex), fontNameIndex(fontNameIndex),
positionInStream(positionInStream) {}
    void print() {
std::cout << "Font Size: " <<
FlyweightCharacterAbstractBuilder::fontSizes[fontSizeIndex]
            << ", font Name: " <<
FlyweightCharacterAbstractBuilder::fontNames[fontNameIndex]
            << ", character stream position: " << positionInStream <<
std::endl;
    }
    ~FlyweightCharacter() {}
 };

 FlyweightCharacter
FlyweightCharacterAbstractBuilder::createFlyweightCharacter(unsigned
short fontSizeIndex, unsigned short fontNameIndex, unsigned short
positionInStream) {
    FlyweightCharacter fc(fontSizeIndex, fontNameIndex,
positionInStream);

    return fc;
 }

 int main(int argc, char** argv) {
    std::vector<FlyweightCharacter> chars;

    FlyweightCharacterAbstractBuilder::setFontsAndNames();
    unsigned short limit = NUMBER_OF_SAME_TYPE_CHARS;

    for (unsigned short i = 0; i < limit; i++) {
        chars.push_back(FlyweightCharacterAbstractBuilder::createFlywei
ghtCharacter(0, 0, i));
        chars.push_back(FlyweightCharacterAbstractBuilder::createFlywei
ghtCharacter(1, 1, i + 1 * limit));
        chars.push_back(FlyweightCharacterAbstractBuilder::createFlywei
ghtCharacter(2, 2, i + 2 * limit));
    }
    /*
        Each char stores links to it's fontName and fontSize so what we
get is:

        each object instead of allocating 6 bytes (convention above)
for string
        and 4 bytes for float allocates 2 bytes for fontNameIndex and
fontSizeIndex.

        That means for each char we save 6 + 4 - 2 - 2 = 6 bytes.
        Now imagine we have NUMBER_OF_SAME_TYPE_CHARS = 1000 i.e. with
```

```
our code
        we will have 3 groups of chars whith 1000 chars in each group
which will save us

        3 * 1000 * 6 - (3 * 6 + 3 * 4) = 17970 saved bytes.

        3 * 6 + 3 * 4 is a number of bytes allocated by
FlyweightCharacterAbstractBuilder.

        So the idea of the pattern is to move properties shared by many
objects to some
        external container. The objects in that case don't store the
data themselves they
        store only links to the data which saves memory and make the
objects lighter.
        The data size of properties stored externally may be
significant which will save REALLY
        huge ammount of memory and will make each object super light in
comparisson to it's counterpart.
        That's where the name of the pattern comes from: flyweight
(i.e. very light).
    */
    for (unsigned short i = 0; i < chars.size(); i++) {
        chars[i].print();
}

    std::cin.get(); return 0;
 }
```

flyweight2.cpp

```
#include <iostream>
#include <string.h>

using namespace std;

class Icon
{
public:
    Icon(char *fileName)
    {
        strcpy(_name, fileName);
        if (!strcmp(fileName, "go"))
        {
            _width = 20;
            _height = 20;
        }
        if (!strcmp(fileName, "stop"))
        {
```

```
            _width = 40;
            _height = 40;
        }
        if (!strcmp(fileName, "select"))
        {
            _width = 60;
            _height = 60;
        }
        if (!strcmp(fileName, "undo"))
        {
            _width = 30;
            _height = 30;
        }
    }
    const char *getName()
    {
        return _name;
    }
    void draw(int x, int y)
    {
        cout << "   drawing " << _name << ": upper left (" << x << ","
<< y <<
            ") - lower right (" << x + _width << "," << y + _height <<
")" <<
            endl;
    }
private:
    char _name[20];
    int _width;
    int _height;
};

class FlyweightFactory
{
public:
    static Icon *getIcon(char *name)
    {
        for (int i = 0; i < _numIcons; i++)
            if (!strcmp(name, _icons[i]->getName()))
                return _icons[i];
        _icons[_numIcons] = new Icon(name);
        return _icons[_numIcons++];
    }
    static void reportTheIcons()
    {
        cout << "Active Flyweights: ";
        for (int i = 0; i < _numIcons; i++)
            cout << _icons[i]->getName() << " ";
        cout << endl;
    }
private:
```

```
    enum
    {
        MAX_ICONS = 5
    };
    static int _numIcons;
    static Icon *_icons[MAX_ICONS];
};

int FlyweightFactory::_numIcons = 0;
Icon *FlyweightFactory::_icons[];

class DialogBox
{
public:
    DialogBox(int x, int y, int incr): _iconsOriginX(x),
_iconsOriginY(y),
        _iconsXIncrement(incr){}
    virtual void draw() = 0;
protected:
    Icon *_icons[3];
    int _iconsOriginX;
    int _iconsOriginY;
    int _iconsXIncrement;
};

class FileSelection: public DialogBox
{
public:
    FileSelection(Icon *first, Icon *second, Icon *third):
DialogBox(100, 100,
        100)
    {
        _icons[0] = first;
        _icons[1] = second;
        _icons[2] = third;
    }
    void draw()
    {
        cout << "drawing FileSelection:" << endl;
        for (int i = 0; i < 3; i++)
            _icons[i]->draw(_iconsOriginX + (i *_iconsXIncrement),
_iconsOriginY);
    }
};

class CommitTransaction: public DialogBox
{
public:
    CommitTransaction(Icon *first, Icon *second, Icon *third):
DialogBox(150,
```

```
        150, 150)
    {
        _icons[0] = first;
        _icons[1] = second;
        _icons[2] = third;
    }
    void draw()
    {
        cout << "drawing CommitTransaction:" << endl;
        for (int i = 0; i < 3; i++)
            _icons[i]->draw(_iconsOriginX + (i *_iconsXIncrement),
_iconsOriginY);
    }
};

int main()
{
    DialogBox *dialogs[2];
    dialogs[0] = new FileSelection(FlyweightFactory::getIcon("go"),
        FlyweightFactory::getIcon("stop"),
FlyweightFactory::getIcon("select"));
    dialogs[1] = new
CommitTransaction(FlyweightFactory::getIcon("select"),
        FlyweightFactory::getIcon("stop"),
FlyweightFactory::getIcon("undo"));

    for (int i = 0; i < 2; i++)
        dialogs[i]->draw();

    FlyweightFactory::reportTheIcons();
}
```

참고

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Flyweight