Flyweight in C++

Back to **Flyweight** description

Flyweight design pattern demo

Discussion. Flyweight describes how to share objects, so that their use at fine granularity is not cost prohibitive. A key concept is the distinction between "intrinsic" and "extrinsic" state. Intrinsic state consists of information that is independent of the flyweight's context - information that is sharable (i.e. each Icon's name, width, and height).

It is stored in the flyweight (i.e. the `Icon` class). Extrinsic state cannot be shared, it depends on and varies with the Flyweight's context (i.e. the x,y position that each `Icon` instance's upper left corner will be drawn at). Extrinsic state is stored or computed by the client and is passed to the flyweight when an operation is invoked. Clients should not instantiate Flyweights directly, they should obtain them exclusively from a `FlyweightFactory` object to ensure they are shared properly.

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

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;
    }
    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
{
```

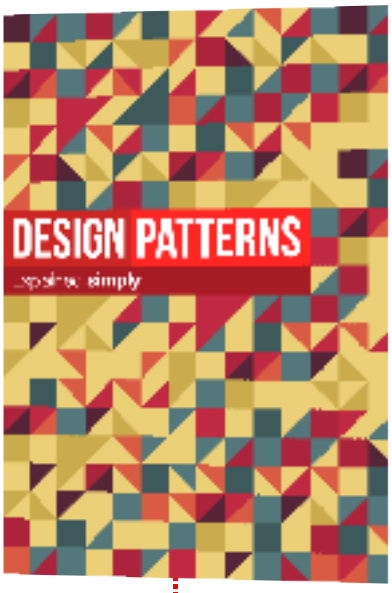
```
  public:
    static Icon *getIcon(char *name)
    {
        for (int i = 0; i < _numIcons; i++)
          if (!strcmp(name, _icons[i]->getName()))
            return _icons[i];
        _icons[_numIcons] = new Icon(name);
        return _icons[_numIcons++];
    }
    static void reportTheIcons()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

Output

```
drawing FileSelection:
   drawing go: upper left (100,100) – lower right (120,120)
   drawing stop: upper left (200,100) – lower right (240,140)
   drawing select: upper left (300,100) – lower right (360,160)
drawing CommitTransaction:
   drawing select: upper left (150,150) – lower right (210,210)
   drawing stop: upper left (300,150) – lower right (340,190)
   drawing undo: upper left (450,150) – lower right (480,180)
Active Flyweights: go stop select undo
```

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	**Flyweight in Java**	**Flyweight in Java**	**Flyweight in Java**	**Flyweight in Java**
C++	**Flyweight in C++: Before and after**	**Flyweight in C++**		
PHP	**Flyweight in PHP**			

Python | Flyweight in

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us