Chương 13

Các mẫu thiết kế phục vụ tổ chức cấu trúc các đối tượng (Structural Patterns)

13.1 Tổng quát về mẫu thiết kế HĐT

- Trong việc phát triển 1 phần mềm, ta thường thực hiện các hoạt động chức năng sau đây :
  1. Nắm bắt yêu cầu phần mềm
  2. Phân tích từng chức năng
  3. Thiết kế
  4. Hiện thực (hay viết code)
  5. Kiểm thử
- Các hoạt động trên có mối quan hệ phụ thuộc nhau, cụ thể kết quả của bước i là dữ liệu đầu vào của bước thứ i+1. Do đó nếu bước thứ i có lỗi, nghĩa là kết quả của nó không đúng thì sẽ kéo theo các bước sau đó sẽ bị lỗi cho dù ta cố gắng thực hiện chúng tốt cách gì đi nữa.

13.1 Tổng quát về mẫu thiết kế HĐT

- Như vậy, lỗi ở bước đầu tiên là nguy hại nhất, kế đó là lỗi ở bước thức 2, thứ 3, ... Tuy nhiên, các bước nắm bắt yêu cầu và phân tích chức năng thường chỉ tạo ra kết quả ít, chưa có độ phức tạp cao, do đó ta vẫn có cách kiểm soát để những kết quả này ít có lỗi nhất. Còn bắt đầu từ bước thiết kế trở đi, kết quả sẽ nhiều và có độ phức tạp cao hơn nên sẽ khó kiểm soát hơn. Và nếu có lỗi ở bước này thì rất nguy hại vì sẽ kéo theo hoạt động hiện thực không có ý nghĩa gì nữa.
- Tóm lại, thiết kế phần mềm là một vấn đề rất khó khăn, nhất là khi phần mềm lớn, mối quan hệ giữa các phần tử sẽ nhiều và phức tạp, bản thiết kế thường không hiệu quả và chứa nhiều lỗi khó biết. Hơn nữa, ta thường phải trả giá cao cho các lỗi thiết kế vì chúng ảnh hưởng nặng nề đến các giai đoạn sau như viết code, kiểm thử….

13.1 Tổng quát về mẫu thiết kế HĐT

- Dùng phương pháp thiết kế hướng đối tượng sẽ giúp ta có thể thiết kế được phần mềm có cấu trúc rõ ràng, mạch lạc, nhờ đó ta dễ phát hiện lỗi nếu có, dễ hiệu chỉnh, dễ nâng cấp từng thành phần (thí dụ nhờ tính bao đóng, bao gộp, thừa kế, đa xạ, tổng quát hóa…).
- Tuy nhiên việc thiết kế phần mềm HĐT còn phụ thuộc nhiều vào khả năng người thiết kế, không phải ai thiết kế đều tạo được những kết quả tích cực như đã nêu trên.

13.1 Tổng quát về mẫu thiết kế HĐT

- Hiện nay, hoạt động thiết kế phần mềm là phải đạt được 3 miêu tiêu chính sau đây (trong nhiều mục tiêu khác) :
  - Mục tiêu 1 : thiết kế được phần mềm giải quyết đúng các chức năng mà user yêu cầu. Đây là mục tiêu chính yếu nhất.
  - Mục tiêu 2 : phải hạn chế được việc tái thiết kế lại trong tương lai, cho dù vì lý do gì.
  - Mục tiêu 3 : bản thiết kế hiện hành phải hỗ trợ tốt nhất việc tái thiết kế lại nếu vì lý do gì đó phải tái thiết kế lại phần mềm.

13.1 Tổng quát về mẫu thiết kế HĐT

Có nhiều nguyên nhân dẫn đến tái thiết kế phần mềm (PM) :

- Do PM phụ thuộc vào phần cứng, hệ điều hành (OS) hay PM khác : PM càng dùng trực tiếp các thông số phần cứng hay PM liên quan sẽ phải thay đổi khi các thông số này thay đổi.
- Do PM phụ thuộc vào giải thuật : khi PM có nhiều giải pháp, nhiều mức độ xử lý cho cùng một vấn đề, việc ràng buộc chặt chẽ PM với giải pháp cụ thể sẽ dẫn đến khó bổ sung, thay đổi PM.
- Do PM chưa có tính tổng quát hóa. Thí dụ tiện ích gỏ phím tiếng Việt lúc đầu chỉ hỗ trợ nhập liệu theo 1 cách gỏ cụ thể, nếu muốn PM này hỗ trợ gỏ nhiều cách khác, ngay cả cách do user tự đặt thì phải thiết kế lại PM gỏ phím.
- Các thành phần của PM liên quan nhau quá chặt chẽ : hiệu chỉnh, nâng cấp 1 thành phần thường phải thay đổi các thành phần phụ thuộc trực tiếp và gián tiếp nó theo kiểu dây chuyền.

13.1 Tổng quát về mẫu thiết kế HĐT

- Một biện pháp được đề xuất để có những bản thiết kế tốt, hạn chế được việc tái thiết kế lại và khi cần thiết kế lại, nó phải hỗ trợ tốt nhất việc tái thiết kế là sử dụng lại những mẫu thiết kế hướng đối tượng (Object Oriented Design Patterns), hay gọi tắt là mẫu thiết kế (Design Patterns).
- Vậy mẫu thiết kế là gì ? Mẫu thiết kế có các đặc điểm sau :
  - Là bản thiết kế của những người chuyên nghiệp và nổi tiếng, đã được sử dụng trong các phần mềm đang được dùng phổ biến và được người dùng đánh giá tốt.
  - Giúp giải quyết 1 trong những vấn đề thiết kế thường gặp trong các phần mềm.
  - Giúp cho bản thiết kế phần mềm có tính uyển chuyển cao, dễ hiệu chỉnh và dễ nâng cấp.

13.1 Tổng quát về mẫu thiết kế HĐT

Vai trò của mẫu thiết kế : mẫu thiết kế đóng 1 số vai trò chính yếu như sau :

- Cung cấp phương pháp giải quyết những vấn đề thực tế thường gặp trong phần mềm mà mọi người đã đánh giá, kiểm nghiệm là rất tốt.
- Là biện pháp tái sử dụng tri thức các chuyên gia phần mềm.
- Hình thành kho tri thức, ngữ vựng trong giao tiếp giữa những người làm phần mềm.
- Giúp ta tìm hiểu để nắm vững hơn đặc điểm ngôn ngữ lập trình hướng đối tượng.

Như vậy, nếu sử dụng triệt để các mẫu thiết kế trong việc thiết kế phần mềm mới, ta sẽ tiết kiệm được chi phí, thời gian và nguồn lực. Hơn nữa PM tạo được sẽ có độ tin cậy, uyển chuyển cao, dễ dàng hiệu chỉnh và nâng cấp sau này khi cần thiết.

13.1 Tổng quát về mẫu thiết kế HĐT

Phân loại các mẫu thiết kế : Dựa vào công dụng, ta có thể phân các mẫu thiết kế thành 3 nhóm chính :

- Structural (nhóm mẫu cấu trúc) : các mẫu này cung cấp cơ chế để quản lý cấu trúc và mối quan hệ giữa các class, thí dụ để dùng lại các class có sẵn (class thư viện, class của bên thứ ba - third party,…), để tạo mối ràng buộc thấp nhất giữa các class (lower coupling) hay cung cấp các cơ chế thừa kế khác.
- Creational (nhóm mẫu phục vụ khởi tạo đối tượng) : giúp khắc phục các vấn đề về khởi tạo đối tượng, nhất là đối tượng lớn và phức tạp, hạn chế sự phụ thuộc của phần mềm vào platform cấp thấp.
- Behavioral (nhóm mẫu giải quyết hành vi của đối tượng) : giúp che dấu sự hiện thực của đối tượng, che dấu giải thuật, hỗ trợ việc thay đổi cấu hình đối tượng một cách linh.

13.1 Tổng quát về mẫu thiết kế HĐT

Phân loại các mẫu thiết kế : Còn nếu dựa vào loại phần tử được dùng trong mẫu, ta có thể phân các mẫu thiết kế thành 2 nhóm chính :

- Class patterns : nhóm mẫu thiết kế chỉ sử dụng các class và mối quan hệ tĩnh giữa các class như thừa kế, hiện thực. Các mối quan hệ này được xác định tại thời điểm dịch, do đó class patterns thích hợp cho thành phần chức năng không cần thay đổi động trong thời gian chạy.
- Object patterns : nhóm mẫu thiết kế có dùng mối quan hệ giữa các đối tượng, mối quan hệ này rất động vì dễ dàng thay đổi bất kỳ lúc nào trong lúc phần mềm chạy

13.1 Tổng quát về mẫu thiết kế HĐT

- Trong phần còn lại của chương này, chúng ta sẽ giới thiệu 1 số mẫu thiết kế có tần suất sử dụng lại trong các ứng dụng cao nhất, mỗi mẫu thiết kế ta sẽ miêu tả các thông tin như :
  - Tên gốc tiếng Anh của mẫu
  - Mục tiêu của mẫu
  - Thí dụ về sử dụng mẫu
  - Lược đồ class miêu tả mẫu : chứa các phần tử (class, đối tượng) trong mẫu và mối quan hệ giữa chúng.

13.2 Mẫu Adapter

Mục tiêu :

- Chuyển đổi interface của một class chức năng có sẵn thành một interface khác theo yêu cầu sử dụng của phần tử sử dụng class đó (ta gọi phần tử sử dụng là Client).
- Thật vậy, trong lập trình đôi khi có những tình huống mà chúng ta cần dùng một class có sẵn nhưng thông qua 1 interface khác chứ không muốn thông qua interface của chính class đó. Mẫu Adapter giúp chúng ta giải quyết vấn đề này

13.2 Mẫu Adapter

Thí dụ về việc dùng mẫu Adapter :

- Ta muốn viết chương trình soạn thảo đồ họa (DrawingEditor) cho phép user vẽ các đối tượng đồ họa như Line, Rectangle, Ellipse, Polygon, Text... Code quản lý các đối tượng đồ họa (ta gọi là Client) xử lý chúng thông qua interface thống nhất là IDrawingShape. Hiện thực các class Line, Rectangle, Ellipse, Polygon từ đầu khá dễ vì chúng là những đối tượng đơn giản nhưng hiện thực class Text thì phức tạp hơn (giả sử class này có khả năng hiển thị nội dung chuỗi trên nhiều dòng và có định dạng phong phú). Giả sử chúng ta đang lập trình trên platform có cung cấp sẵn 1 class có tên là TextView, class này cung cấp chức năng quản lý Text giống như chương trình muốn nhưng interface sử dụng của nó là ITextView không giống với interface IDrawingShap của chương trình.

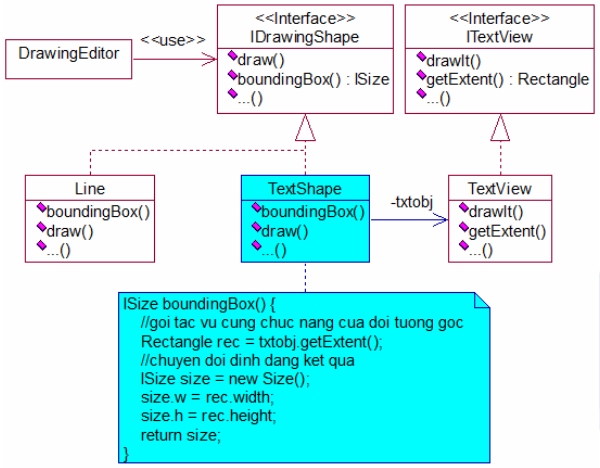
13.2 Mẫu Adapter

Thí dụ về việc dùng mẫu Adapter :

- Việc thay đổi interface IDrawingShape của chương trình thành ITextView là không thể được vì ảnh hưởng đến các class đối tượng đồ họa khác đã viết. Việc hiệu chỉnh lại class TextView để hỗ trợ interface IDrawingShape của chương trình cũng không khả thi vì TextView đã được sử dụng bởi nhiều ứng dụng khác, hệ thống không thể thay đổi tùy tiện interface sử dụng của nó được.
- Cách tốt nhất để chương trình DrawingEditor của chúng ta có thể dùng được class TextView mà không cần thay đổi TextView cũng chẳng cần thay đổi mã nguồn của ứng dụng là dùng mẫu Adapter, cụ thể định nghĩa class TextShape như là phần tử đại diện cho TextView, nó sẽ cung cấp đúng interface sử dụng mà chương trình cần. Chúng ta sẽ thấy chi phí định nghĩa class TextShape là rất thấp và độc lập với độ phức tạp của class TextView.

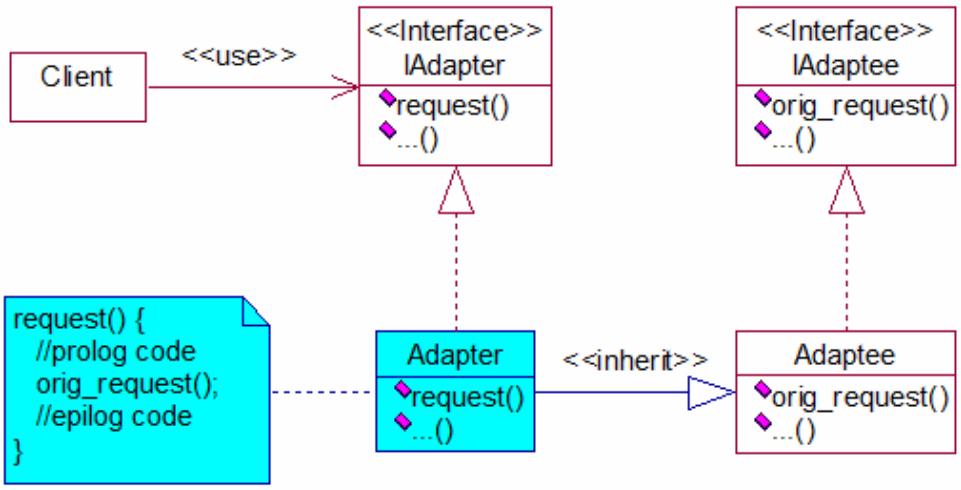
13.2 Mẫu Adapter

Lược đồ class miêu tả mẫu Adapter thể hiện cách thức chương trình DrawingEditor sử dụng class TextView như sau :



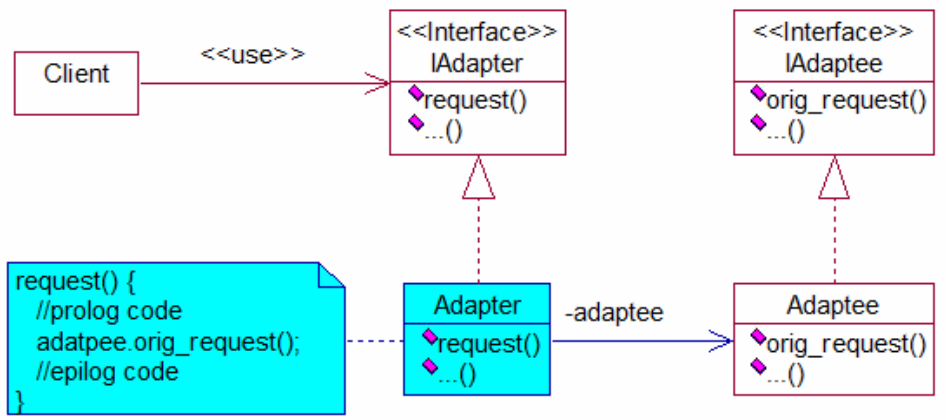
13.2 Mẫu Adapter

Lược đồ class miêu tả mẫu Adapter dạng class pattern :



13.2 Mẫu Adapter

Lược đồ class miêu tả mẫu Adapter dạng object pattern :



13.2 Mẫu Adapter

Các phần tử tham gia :

- IAdapter (IDrawingShape) : định nghĩa interface cho đoạn code client sử dụng.
- IAdaptee (ITextView) : định nghĩa interface của đối tượng có sẵn cần "chuyển" sang interface IAdapter.
- Client (DrawingEditor) : đoạn code của chương trình có sử dụng các đối tượng thông qua interface IAdapter.
- Adaptee (TextView) : class chức năng đã có sẵn cần dùng trong đoạn code client.
- Adapter (TextShape) : class cần định nghĩa với chi phí rất thấp, nhiệm vụ chuyển interface IAdaptee sang interface IAdapter

13.2 Mẫu Adapter

- Ta nói việc định nghĩa class Adaper rất dễ dàng, vì từng tác vụ của class này được viết theo template như sau :

```
Request() {
    //đoạn code prolog (thường rất ngắn thậm chí không có)
    Adaptee.orig_request();     //gọi tác vụ có chức năng tương tự của Adaptee
    //đoạn code epilog (thường rất ngắn thậm chí không có)
}
```


13.2 Mẫu Adapter

- Thí dụ trong class TextShape của chương trình DrawingEditor, tác vụ boundingBox() được viết như sau :

```
ISize boundingBox() {
    //không cần có code prolog
    //gọi tác vụ cùng chức năng của đối tượng gốc
    Rectangle rec = txtobj.getExtent();
    //code epilog sẽ chuyển đổi định dạng kết quả
    ISize size = new Size();
    size.w = rec.width;
    size.h = rec.height;
    return size;
}
```

13.3 Mẫu Composite

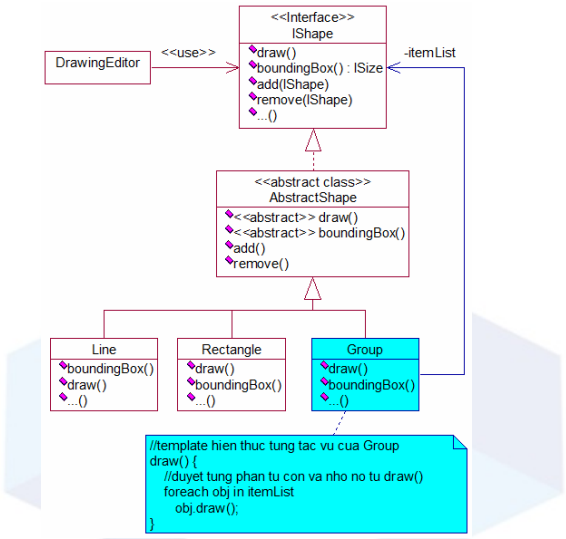
Mục tiêu :

- Tạo quan hệ bao gộp giữa các đối tượng được dùng bởi module client. Client sẽ dùng đối tượng bao gộp và đối tượng bị bao gộp thông qua cùng 1 interface thống nhất, nhờ đó đoạn code Client sẽ có tính tổng quát hóa cao, dễ phát triển, dễ bảo trì, không cần hiệu chỉnh khi các đối tượng được sử dụng bị thay đổi, thêm/bớt.

Thí dụ về việc dùng mẫu Composite :

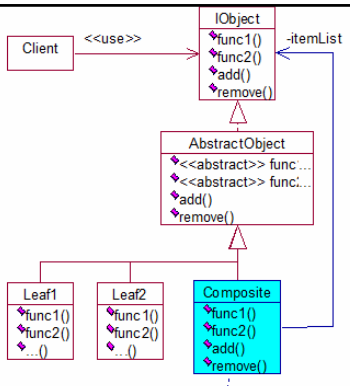
- Chương trình DrawingEditor vừa có các đối tượng đơn (không chứa đối tượng khác) như Line, Rectangle, Ellipse,… vừa có các đối tượng tích hợp như Group chứa nhiều đối tượng khác. Nhưng đoạn code xử lý của chương trình muốn dùng các đối tượng thông qua cùng 1 interface thống nhất để độc lập với số lượng và tính chất của các đối tượng đồ họa. Cách tốt nhất để giải quyết vấn đề này là dùng mẫu Composite với lược đồ class như sau

13.3 Mẫu Composite



13.3 Mẫu Composite

Ta có thể xây dựng mẫu Composite theo loại object pattern để miêu tả mối quan hệ bao gộp giữa các đối tượng. Lược đồ class sẽ như sau :




```
//template hien thuc tung tac vu chuc nang cua Composite
func1() {
    //duyet tung phan tu con va nho no thuc hien func1()
    foreach obj in itemList
        obj.func1();
}
//template hien thuc tung tac vu quan ly nhom cua Composite
add(IObject s) {
    itemList.add(s);
}
remove(IObject s) {
    itemList.remove(s);
}
```


13.3 Mẫu Composite

Các phần tử tham gia :

- IObject (IShape) : interface thống nhất của các đối tượng được sử dụng, nó chứa 2 nhóm tác vụ : nhóm tác vụ chức năng của mọi đối tượng, nhóm tác vụ quản lý các đối tượng thành phần (chỉ cần cho đối tượng bao gộp).
- Client (DrawingEditor) : đoạn code của chương trình có sử dụng các đối tượng thông qua interface thống nhất IObject.
- AbstractObject (AbstractShape) : class trừu tượng, nhiệm vụ là hiện thực các tác vụ dùng chung bởi các đối tượng con, thí dụ như nhóm tác vụ quản lý các đối tượng thành phần để các class đối tượng đơn không cần hiện thực chúng (vì không có liên quan).

13.3 Mẫu Composite

Các phần tử tham gia :

- Leaf (Line) : class đặc tả đối tượng đơn, chi phí đặc tả nó phụ thuộc vào tính chất và chức năng của nó. Chi phí đặc tả này độc lập với việc dùng mẫu Composite.
- Composite (Group) : class đặc tả đối tượng tích hợp, đây là class trọng tâm của mẫu. Ta chỉ tốn chi phí rất thấp để đặc tả class này vì từng tác vụ của class này được viết theo template như sau :

```
//template hiện thực từng tác vụ chức năng của class Composite
func1() {
   //duyệt từng phần tử con và nhờ nó thực hiện func1()
   foreach obj in itemList
     obj.func1();
}
```


13.4 Mẫu Proxy

Mục tiêu :

- Cung cấp đối tượng đại diện cho một đối tượng khác (đối tượng cung cấp dịch vụ gốc) để hỗ trợ hoặc kiểm soát quá trình truy xuất đối tượng gốc đó. Đối tượng thay thế được gọi là Proxy.

Có nhiều lý do để dùng mẫu Proxy, trong đó có 4 lý do phổ biến sau đây :

- Việc khởi tạo những đối tượng lớn và phức tạp sẽ tốn nhiều tài nguyên và thời gian, do đó ta có khuynh hướng trì hoãn việc khởi tạo thực sự các đối tượng này đến khi thật cần thiết. Trong thời gian trì hoãn, đối tượng proxy sẽ đóng vai trò thay thế đối tượng gốc để phục vụ tạm các yêu cầu từ Client.

13.4 Mẫu Proxy

- Chương trình muốn truy xuất một đối tượng ở không gian địa chỉ khác, thí dụ đối tượng nằm ở máy khác. Trong trường hợp này, ta sẽ tạo proxy chạy trên máy chương trình ứng dụng để đại diện cho đối tượng gốc trên máy ở xa.
- Đối tượng gốc cần được che chắn để Client không tương tác trực tiếp được hầu đảm bảo độ bảo mật cao, Client chỉ giao tiếp trực tiếp được với Proxy, sau đó Proxy chuyển yêu cầu của Client tới đối tượng gốc để thực hiện yêu cầu.
- Chương trình muốn kiểm soát, tăng cường, bổ sung một số tính năng của đối tượng gốc. Proxy sẽ đóng vai trò đối tượng thực hiện việc kiểm soát, tăng cường, bổ sung tính năng.

13.4 Mẫu Proxy

- Mỗi yêu cầu proxy trong 4 yêu cầu phổ biến trên được giải quyết bởi 1 loại proxy tương ứng như sau :
  - Virtual proxy : cung cấp đối tượng đại diện cho đối tượng lớn và phức tạp. Mục đích để trì hoãn thời điểm tạo đối tượng lớn (ví dụ đối tượng bitmap trong chương trình soạn thảo tài liệu MSWord).
  - Remote proxy : cung cấp đối tượng đại diện (local) cho một đối tượng từ xa (remote) (ví dụ RMI, JINI).
  - Protection proxy : cung cấp đối tượng đại diện cho một đối tượng khác cần được che chắn, bảo mật từ bên ngoài. Ví dụ các KernelProxies cung cấp dịch vụ truy xuất tới Kernel của hệ điều hành.
  - Smart proxy : cung cấp đối tượng đại diện để bổ sung một số tính năng của đối tượng gốc.

13.4 Mẫu Proxy

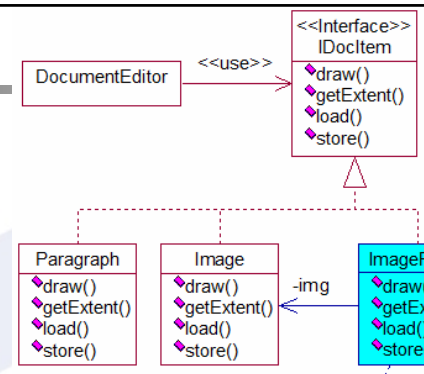
Thí dụ về việc dùng mẫu Proxy :

- Chương trình MSWord quản lý tài liệu đang soạn thảo trong đối tượng Document, đây là đối tượng bao gộp nhiều đối tượng thành phần theo dạng phân cấp. Mỗi lần user mở lại 1 file tài liệu, về nguyên tắc, chương trình sẽ đọc toàn bộ nội dung của file để tạo lại đầy đủ các đối tượng được chứa trong file đó, nối kết chúng theo đúng cấu trúc gốc của đối tượng Document trước khi hiển thị kết quả lên màn hình để người dùng xử lý tiếp. Tuy nhiên việc đọc nội dung các đối tượng lớn và phức tạp như hình bitmap, bảng dữ liệu lớn… vào bộ nhớ chương trình sẽ tốn nhiều thời gian. Hơn nữa mỗi lần chương trình chỉ hiển thị 1 phần rất nhỏ nội dung của tài liệu (1 trang màn hình) và chưa chắc gì user muốn làm việc tiếp với phần còn lại của tài liệu.

13.4 Mẫu Proxy

- Do đó chiến lược đọc toàn bộ tài liệu từ file vào bộ nhớ 1 lần khi có yêu cầu của user là không tốt, vừa làm tăng thời gian đáp ứng với user, vừa phí phạm công sức trong trường hợp user chỉ xử lý 1 phần rất nhỏ của tài liệu. Để khắc phục các nhược điểm trên, ta sẽ dùng mẫu Proxy. Cụ thể mỗi lần user mở 1 file tài liệu, ta chỉ nạp trực tiếp các đối tượng nhỏ của tài liệu vào bộ nhớ, còn các đối tượng lớn và phức tạp như bitmap, bảng dữ liệu, ta sẽ chỉ tạo đối tượng Proxy đại diện cho chúng, đối tượng Proxy sẽ chứa các thông tin cơ bản, thiết yếu về đối tượng gốc. Nhờ đó, thời gian nạp tài liệu (lần đầu) sẽ rất nhanh và đáp ứng kịp thời user. Theo theo thời gian, khi user lật tới trang nào, chương trình sẽ kiểm tra trang đó có chứa đối tượng Proxy không, nếu có thì sẽ nhờ Proxy nạp thật đối tượng gốc trên file vào để user có thể làm việc trực tiếp với đối tượng gốc.

13.4 Mẫu Proxy

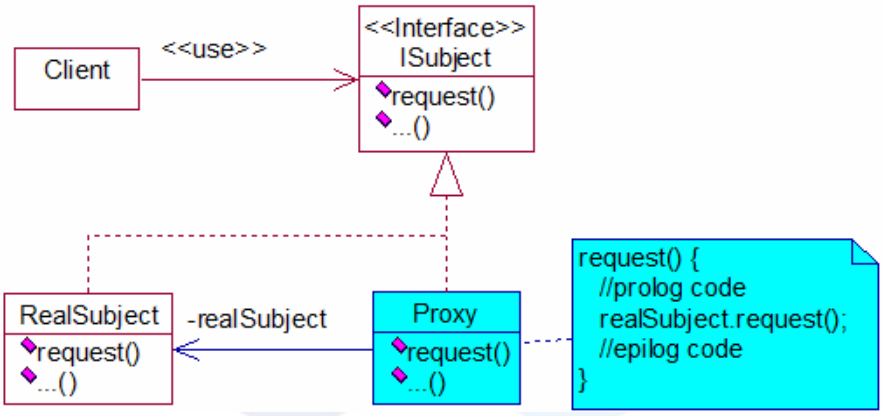



```
//template tac vu can doi tuong goc
draw() {
   if (img == null) img = new Image(filename); //load tu file vao
   img.draw();
}
//template tac vu khong can doi tuong goc
getExtent()
{
  if (img == null) return extent; //tra ve thong tin extent cua phan tu goc
  else return img.getExtent();  //goi phan tu goc tra ve
}
```


13.4 Mẫu Proxy

Ta có thể xây dựng mẫu Proxy theo loại object pattern để miêu tả mối quan hệ giữa proxy và đối tượng gốc mà nó quản lý.

Lược đồ class sẽ như sau :



13.4 Mẫu Proxy

Các phần tử tham gia :

- ISubject (IDocItem) : interface thống nhất của các đối tượng được sử dụng : đối tượng gốc và đối tượng proxy.
- Client (DocumentEditor) : đoạn code của chương trình có sử dụng các đối tượng thông qua interface thống nhất ISubject.
- RealSubject (Image) : class đặc tả đối tượng gốc cần dùng bởi client, chi phí đặc tả nó phụ thuộc vào tính chất và chức năng của nó, nhưng thường là rất lớn. Chi phí đặc tả này độc lập với việc dùng mẫu Proxy.

13.4 Mẫu Proxy

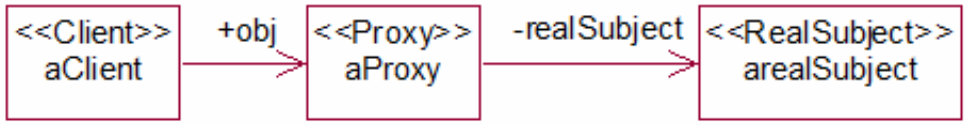
- Proxy (ImageProxy) : class đặc tả đối tượng proxy, đây là class trọng tâm của mẫu. Nó sẽ thay thế đối tượng RealSubject trong khoảng thời gian chưa cần đối tượng RealSubject, nó giữ tham khảo đến đối tượng RealSubject để nhờ vả khi cần, nó kiểm soát quá trình truy xuất đến đối tượng RealSubject, có thể tạo hoặc delete đối tượng RealSubject. Tùy loại proxy mà đối tượng Proxy còn thực hiện 1 số hoạt động khác nữa.

Ta chỉ tốn chi phí rất thấp để đặc tả class Proxy vì từng tác vụ của class này được viết theo template như sau :

```
request() {
  //prolog code, thường rất ngắn, thậm chí là không có
  realSubject.request();  //nếu cần, gọi đối tượng gốc thực hiện
  //epilog code, thường rất ngắn, thậm chí là không có
}
```

13.4 Mẫu Proxy

Quá trình giao tiếp ở thời điểm run-time có thể mô tả bằng lược đồ đối tượng như sau :



13.5 Mẫu Decorator

Mục tiêu :

- Thêm động 1 số trách nhiệm mới cho 1 đối tượng cụ thể mà không ảnh hưởng đến các đối tượng khác cùng chủng loại (cùng class).
- Lưu ý là để thêm trách nhiệm cho toàn bộ các đối tượng của 1 class, ta có thể dùng tính thừa kế hay thậm chí hiệu chỉnh trực tiếp mã nguồn của class đó

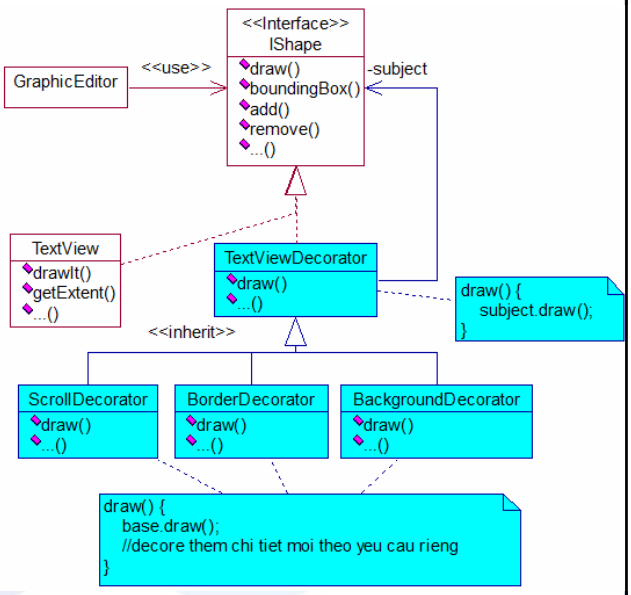
13.5 Mẫu Decorator

Thí dụ về việc dùng mẫu Decorator :

- Chương trình DrawingEditor cho phép tạo và hiển thị nhiều đối tượng đồ họa, trong đó có đối tượng TextView để hiển thị nội dung văn bản. Giả sử TextView là đối tượng hình chữ nhật với kích thước cố định do user qui định, nó chỉ hiển thị nội dung văn bản trong hình chữ nhật do nó quản lý chứ không có đường viền bao quanh.
- Trong trường hợp user tạo 1 TextView mới, có thể họ muốn TextView này có đường viền bao quanh cho rõ ràng. Tương tự họ muốn TextView hiển thị hình nền phía dưới nội dung văn bản và trong trường hợp nội dung văn bản của TextView quá dài không thể được hiển thị hết trong hình chữ nhật giới hạn của TextView thì user muốn hiển thị thêm các scrollbar ngang và dọc để giúp user dời dễ dàng đến vị trí văn bản cần tập trung xử lý.

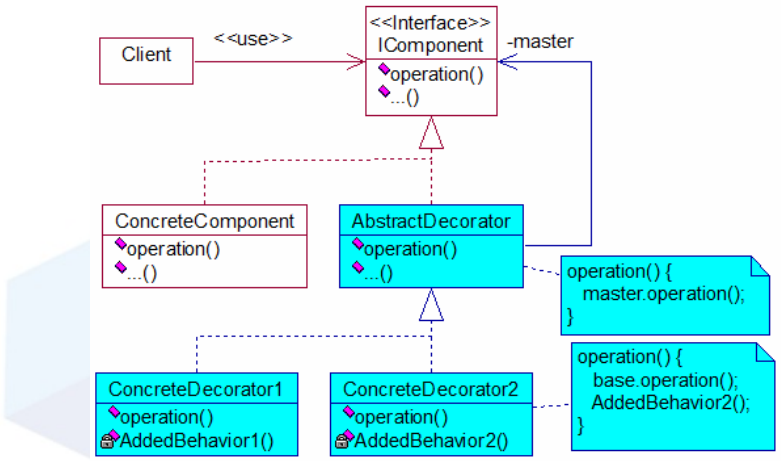
13.5 Mẫu Decorator

Cách tốt nhất để giải quyết các yêu cầu trên là dùng mẫu Decorator với lược đồ class như sau :



13.5 Mẫu Decorator

Ta có thể xây dựng mẫu Decorator theo loại object pattern để miêu tả mối quan hệ giữa đối tượng decore và đối tượng gốc. Lược đồ class sẽ như sau :



13.5 Mẫu Decorator

Các phần tử tham gia :

- IComponent (IShape) : interface thống nhất của các đối tượng được sử dụng : đối tượng gốc và đối tượng decore cho đối tượng gốc.
- Client (DrawingEditor) : đoạn code của chương trình có sử dụng các đối tượng thông qua interface thống nhất IComponent.
- ConcreteComponent (TextView) : class đặc tả đối tượng gốc cần dùng bởi client mà ta muốn decore cho nó, chi phí đặc tả nó phụ thuộc vào tính chất và chức năng của nó. Chi phí đặc tả này độc lập với việc dùng mẫu Decorator.
- AbstractDecorator (TextViewDecorator) : class trừu tượng, nhiệm vụ là đặc tả các thành phần dùng chung bởi các đối tượng con, thí dụ như tham khảo đến đối tượng cần decore, gởi thông điệp gọi tác vụ tương ứng của đối tượng gốc…

13.5 Mẫu Decorator

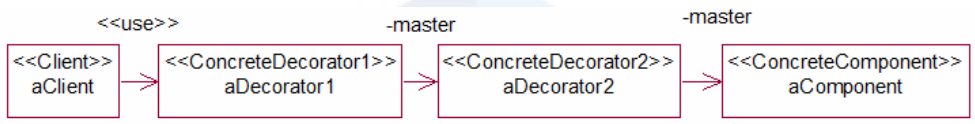
Các phần tử tham gia :

- ConcreteDecorator1… (ScrollDecorator…) : class đặc tả đối tượng decorator cụ thể, đây là class trọng tâm của mẫu. Nó sẽ đại diện đối tượng gốc để giao tiếp với Client, nó giữ tham khảo đến đối tượng gốc để yêu cầu công việc tương ứng. Ta tốn chi phí khá thấp để đặc tả class này vì từng tác vụ của class này được viết theo template như sau :

```
operation() {
   base.operation();  //gọi đối tượng gốc thực hiện công việc trước
   AddedBehavior();    //decore thêm theo yêu cầu
}
```


13.5 Mẫu Decorator

Quá trình giao tiếp ở thời điểm run-time có thể mô tả bằng lược đồ đối tượng như sau :



13.6 Mẫu Facade

Mục tiêu :

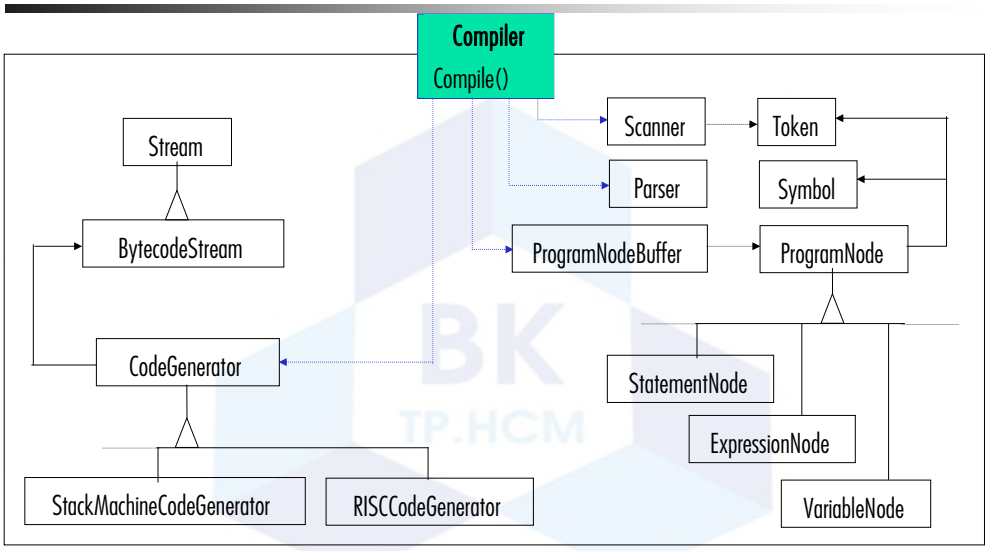
- cung cấp interface hợp nhất cho tập các interface của 1 hệ thống con. Facade định nghĩa 1 interface cấp cao hơn các interface có sẵn để làm cho hệ thống con dễ sử dụng hơn.
- tối thiểu hóa tính "coupling" (nối ghép) giữa các hệ thống con.

13.6 Mẫu Facade

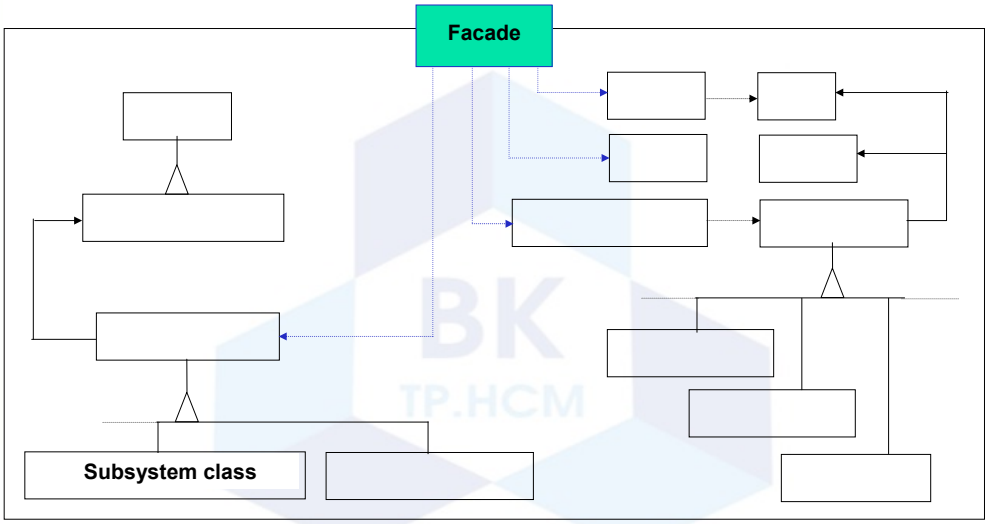
Thí dụ về việc dùng mẫu Facade :

- hệ thống con phục vụ biên dịch có nhiều class phục vụ các bước biên dịch rời rạc như Scanner, Parser, ProgramNode, BytecodeStream, ProgramNodeBuilder. Để dịch source code, ta có thể viết 1 ứng dụng gọi dịch vụ của từng class để duyệt token, parser, xây dựng cây cú pháp, tạo code đối tượng... Tuy nhiên làm như trên sẽ rất khó và dễ gây ra lỗi. Cách khắc phục là định nghĩa 1 class mới với giao tiếp hợp nhất tên là Compiler, nó cung cấp 1 hàm Compile (file), ứng dụng nào cần dịch source code chỉ cần gởi thông điệp Compile đến đối tượng Compiler.

13.6 Mẫu Facade



13.6 Mẫu Facade



13.6 Mẫu Facade

Các phần tử tham gia :

- Facade (Compiler)
  - biết class nào liên quan đến request xác định.
  - nhờ các đối tượng liên quan thực hiện request.
- subsystem classe (Scanner, Parser,..)
  - hiện thực một chức năng cụ thể của hệ thống con.
  - xử lý công việc được đối tượng Façade nhờ.
  - không cần biết Facade, không có tham khảo đến Facade.

13.7 Mẫu Flyweight

Mục tiêu :

- dùng phương tiện dùng chung để quản lý hiệu quả 1 số rất lớn đối tượng có nhiều thành phần giống nhau.

13.7 Mẫu Flyweight

Thí dụ về việc dùng mẫu Flyweight :

- Chương trình xử lý văn bản dùng khái niệm đối tượng để miêu tả bất kỳ phần tử cơ bản nào : ký tự, công thức, hình, ... Ký tự là đối tượng xuất hiện rất nhiều lần trong văn bản, nếu ta miêu tả đối tượng ký tự chứa trực tiếp tất cả các thông tin về nó như mã ký tự, tên font, kích cỡ, màu, biến thể (normal, bold, italic,…), thì mỗi đối tượng ký tự sẽ chiếm nhiều không gian, như vậy việc lưu giữ các đối tượng ký tự sẽ rất không hiệu quả. Mẫu Flyweight rất thích hợp để giải quyết vấn đề miêu tả đối tượng ký tự.

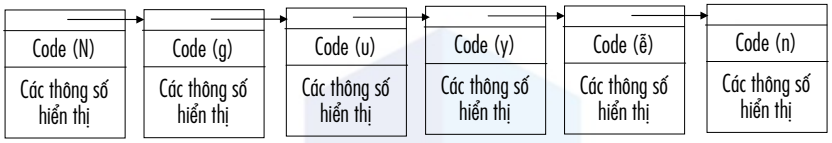
13.7 Mẫu Flyweight

Thí dụ về việc dùng mẫu Flyweight :

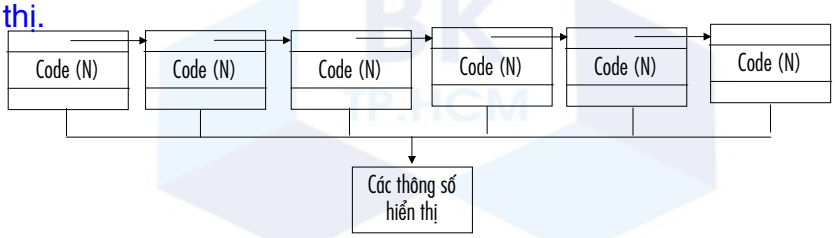
- Flyweight là đối tượng dùng chung dựa vào khái niệm cơ bản là các thuộc tính của nó có thể được chia làm 2 loại :
  - trạng thái trong : các thuộc tính độc lập với ngữ cảnh sử dụng, thí dụ đối tượng ký tự có code ký tự là trạng thái trong.
  - trạng thái ngoài : các thuộc tính phụ thuộc và thay đổi theo ngữ cảnh, thí dụ đối tượng ký tự có thuộc tính tên font, kích cỡ, màu, biến thể,… là trạng thái ngoài. Ta không nên chứa trạng thái ngoài trực tiếp trong đối tượng mà nên gom chúng trong 1 đối tượng khác, flyweight chỉ chứa tham khảo đến đối tượng chứa trạng thái ngoài.
- Đối tượng chứa trạng thái ngoài thường được dùng chung bởi nhiều flyweight khác nhau.

13.7 Mẫu Flyweight

- Miêu tả các đối tượng ký tự 1 cách bình thường

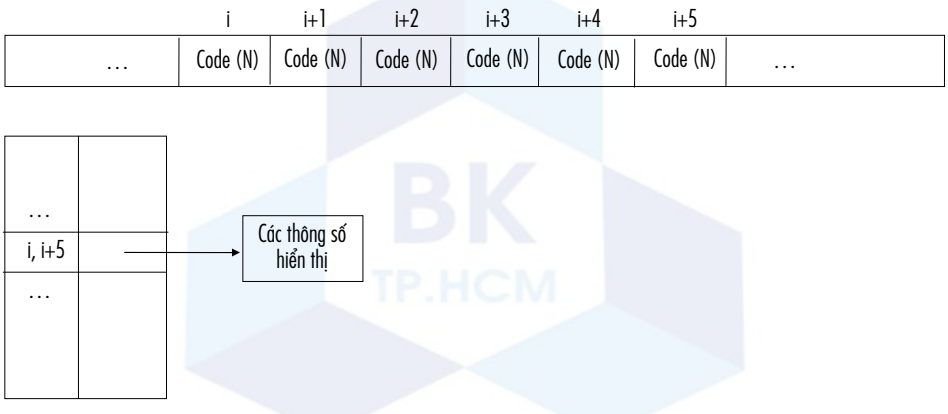


- Miêu tả các đối tượng ký tự dùng mẫu Flyweight giúp dùng chung các thông số hiển thị bởi rất nhiều ký tự có cùng cách thức hiển thị.

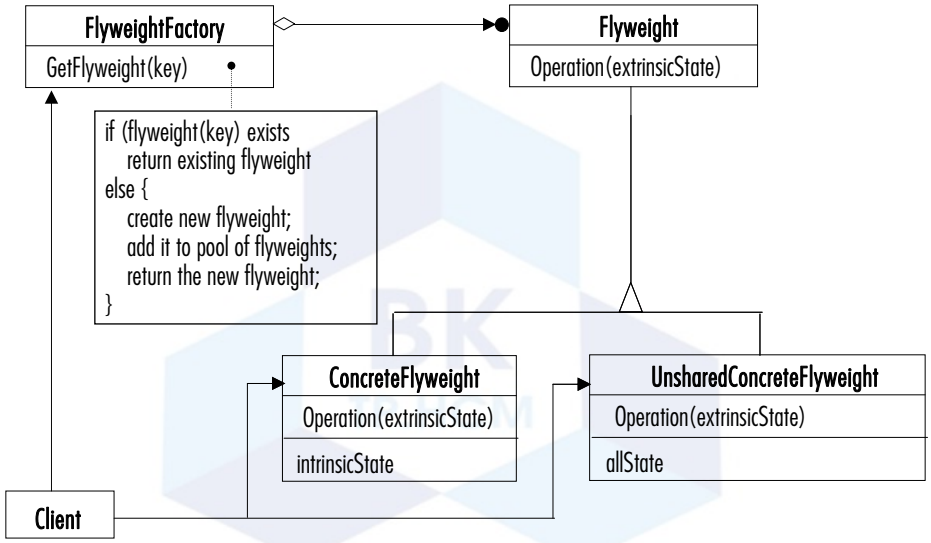


13.7 Mẫu Flyweight

- Miêu tả các đối tượng ký tự trong MS Word (dùng mẫu Flyweight và array)



13.7 Mẫu Flyweight



13.7 Mẫu Facade

Các phần tử tham gia :

- Flyweight (Glyph) : định nghĩa interface cho đối tượng nhận yêu cầu và hoạt động theo trạng thái ngoài.
- ConcreteFlyweight (Character) : hiện thực interface Flyweight thành các đối tượng dùng chung.
- UnsharedConcreteFlyweight (Character) : hiện thực interface Flyweight thành các đối tượng không dùng chung.
- FlyweightFactory : tạo và quản lý các đối tượng Flyweight.
- Client : chứa tham khảo đến flyweight và nhờ khi cần.

13.8 Kết chương

- Chương này đã giới thiệu các thông tin cơ bản về mẫu thiết kế hướng đối tượng và miêu tả mục tiêu, tính chất của các mẫu thiết kế phục vụ cấu trúc các đối tượng như Adapter, Composite, Proxy, Decorator, Facade, Flyweight.