Chương 8

Các mẫu thiết kế che dấu hành vi, thuật giải trong đối tượng (Behavioral Patterns)

8.1 Tổng quát về nhóm mẫu "Behavioral Patterns"
8.2 Mẫu Chain of Responsibility
8.3 Mẫu Template Method
8.4 Mẫu Strategy
8.5 Mẫu State
8.6 Mẫu Command
8.7 Mẫu Observer
8.8 Kết chương

8.1 Tổng quát về nhóm mẫu "Behavioral Patterns"

- Trong đoạn code giải quyết vấn đề của ứng dụng, khi cần phải chọn lựa 1 trong nhiều thuật giải/hành vi khác nhau thì ta thường dùng phát biểu if/switch như sau :

```
switch (acode) {
case ALG1 : //đoạn code miêu tả thuật giải/hành vi 1
case ALG2 : //đoạn code miêu tả thuật giải/hành vi 2
…
case ALGn : //đoạn code miêu tả thuật giải/hành vi n
}
```

- Đoạn code trên có nhiều khuyết điểm như : phu thuộc hoàn toàn vào số lượng thuật giải/hành vi, vào chi tiết cụ thể của từng thuật giải/hành vi, phải hiệu chỉnh khi số lượng/chi tiết của thuật giải/hành vi bị thay đổi.

8.1 Tổng quát về nhóm mẫu “Behavioral Patterns”

- Để khắc phục các nhược điểm của cách lập trình cổ điển trong slide trước, cách tốt nhất là dùng 1 trong các mẫu thuộc nhóm “Behavioral Patterns”.
- Nhiệm vụ của các mẫu thuộc nhóm “” là che dấu các đoạn code miêu tả thuật giải/hành vi vào trong các đối tượng, code của client chỉ giữ tham khảo đến đối tượng và gởi thông điệp nhờ đối tượng thực hiện thuật giải/hành vi cụ thể khi cần thiết.

8.2 Mẫu Chain of Responsibility

Mục tiêu :

- Mẫu dây chuyền trách nhiệm (Chain of Responsibility) giúp tránh được việc gắn kết cứng giữa phần tử gởi request (Client) với phần tử nhận và xử lý request (Server) bằng cách cho phép hơn 1 đối tượng có cơ hội xử lý request đó. Các đối tượng nhận và xử lý request sẽ được liên kết lại thành 1 dây chuyền, Client sẽ tham khảo đến đầu dây chuyền này để gởi request khi có yêu cầu.

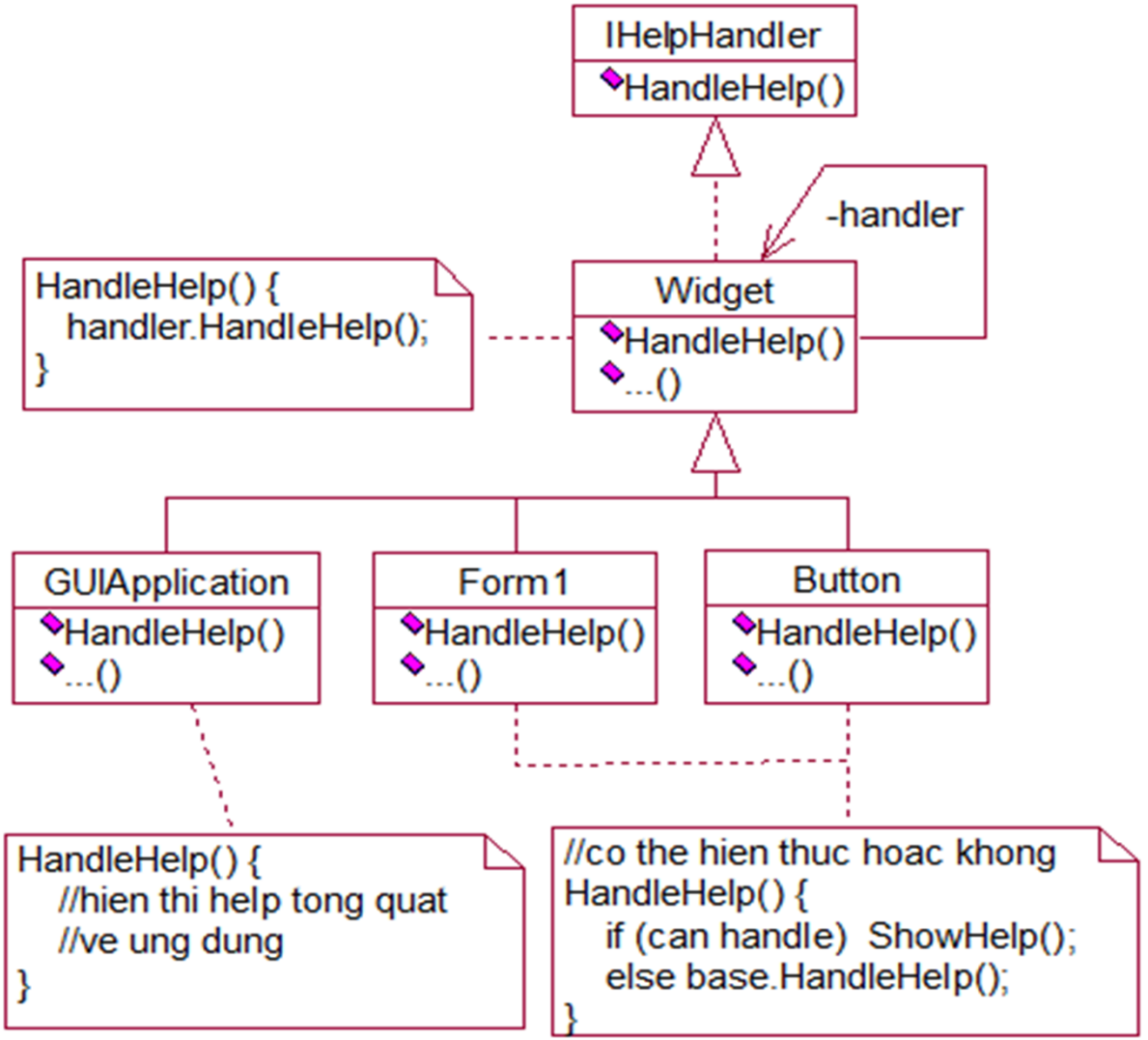
8.2 Mẫu Chain of Responsibility

Thí dụ về việc dùng mẫu Chain of Responsibility :

- Trong ứng dụng có trợ giúp theo ngữ cảnh thì user có thể xem thông tin trợ giúp của 1 phần tử giao diện nào đó trực tiếp từ phần tử đó bằng cách ấn phải chuột vào nó. Lưu ý là các đối tượng giao diện thường được tổ chức theo dạng cây thứ bậc : 1 chương trình có nhiều cửa sổ giao diện, mỗi cửa sổ giao diện chứa nhiều đối tượng giao diện, mỗi đối tượng giao diện có thể là group chứa nhiều đối tượng giao diện con… Tóm lại số lượng các đối tượng giao diện đơn (không chứa đối tượng khác nữa) của chương trình thường rất lớn, chi phí hiện thực tất cả sự trợ giúp cho tất cả các đối tượng đơn này sẽ rất lớn, do đó thường sẽ được hiện thực từ từ thông qua nhiều version mới đạt được sự hoàn chỉnh. Tuy nhiên, dưới góc nhìn user, ngay cả version đầu tiên, chương trình cũng phải đáp ứng tốt mọi yêu cầu trợ giúp theo ngữ cảnh trên mọi đối tượng giao diện.

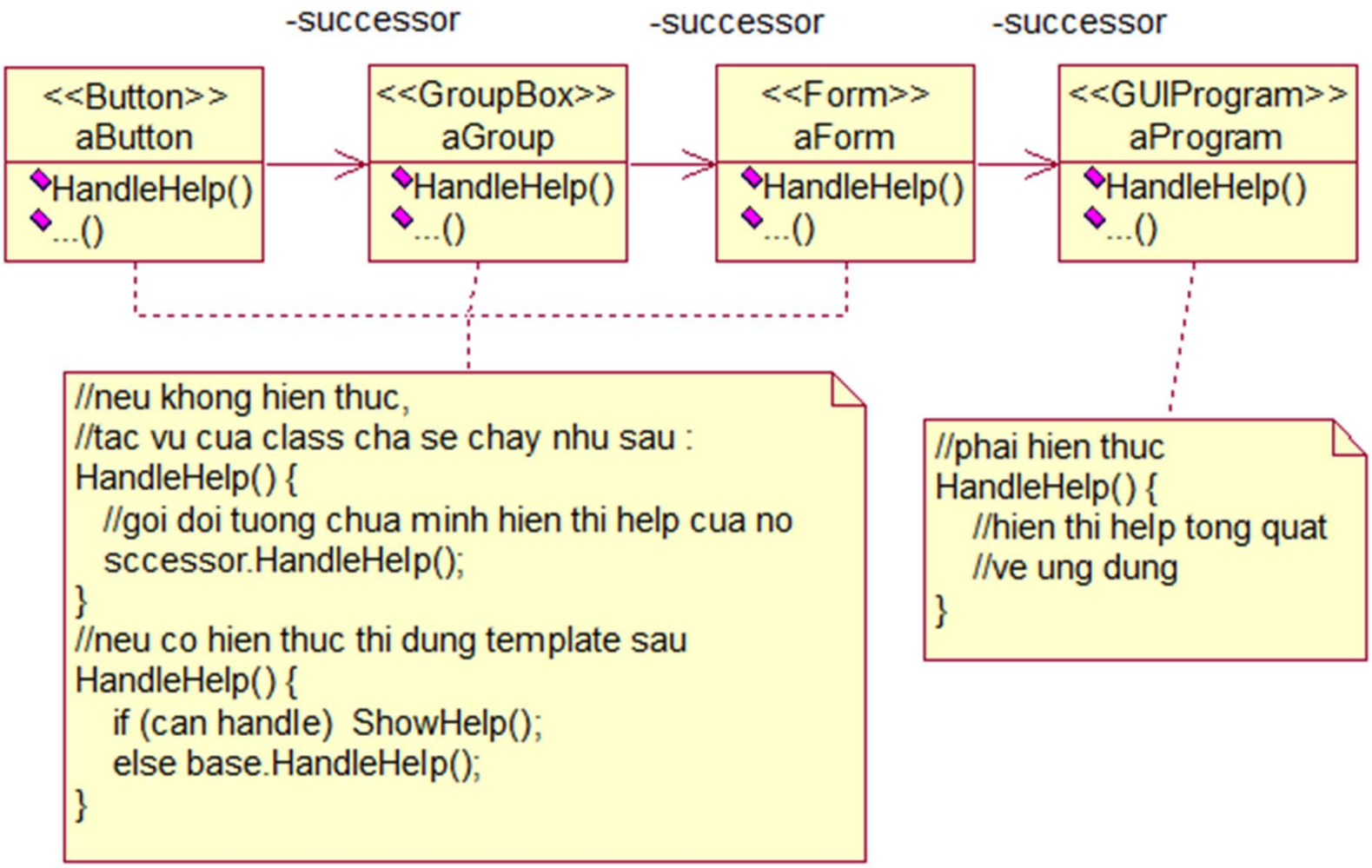
8.2 Mẫu Chain of Responsibility

Cách tốt nhất để giải quyết vấn đề trên là dùng mẫu Chain of Responsibility với lược đồ class như sau :



8.2 Mẫu Chain of Responsibility

Lược đồ đối tượng liên quan đến 1 button nào đó có dạng như sau :



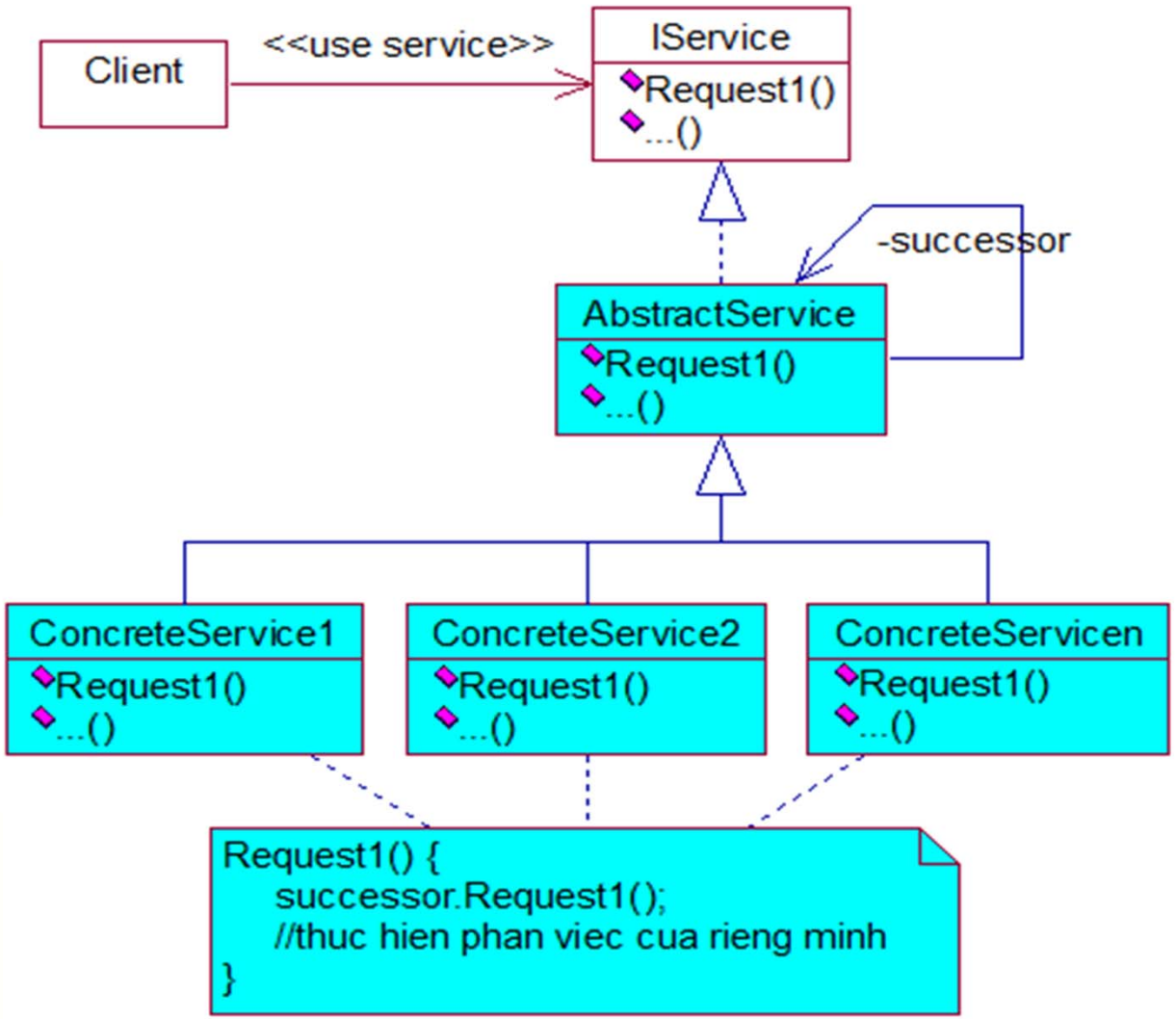
8.2 Mẫu Chain of Responsibility

Theo lược đồ đối tượng của Button như trên thì khi user ấn phải chuột vào button để xem trợ giúp về button đó thì :

- Hoặc là hàm xử lý HandleHelp() của chính button đó chạy (nếu có hiện thực) để hiển thị nội dung trợ giúp chính xác về button đó.
- Hoặc là hàm xử lý HandleHelp() của đối tượng chứa button sẽ chạy (GroupBox - nếu có hiện thực) để hiển thị nội dung trợ giúp về phần tử GroupBox đó, nội dung này thường chứa thông tin trợ giúp của button.
- Tương tự, nếu GroupBox không hiện thực hàm xử lý HandleHelp() thì hàm HandleHelp() của Form sẽ chạy, còn nếu Form cũng không hiện thực hàm xử lý HandleHelp() thì cuối cùng hàm xử lý HandleHelp() của chương trình sẽ chạy. Trong trường hợp này user sẽ xem được nội dung trợ giúp của toàn phần mềm, trong đó có thông tin sử dụng button mà họ cần.

8.2 Mẫu Chain of Responsibility

Ta có thể xây dựng mẫu Chain of Responsibility theo loại object pattern với lược đồ class như sau :



8.2 Mẫu Chain of Responsibility

Các phần tử tham gia :

- IService (IHelpHandler) : định nghĩa interface của tác vụ xử lý request.
- AbstractService (Widget) : đặc tả các thành phần dùng chung cho tất cả đối tượng xử lý request, thí dụ thuộc tính tham khảo đến đối tượng đi sau mình trong dây chuyền xử lý, hiện thực tác vụ request() với nhiệm vụ cơ bản nhất là gọi tác vụ này của đối tượng mà mình tham khảo trực tiếp.
- ConcreteService1… (Button…) : hiện thực tác vụ request() theo yêu cầu riêng của mình theo ý tưởng chung như sau : nếu có thể xử lý được request, nó sẽ xử lý, nếu không thì gởi tiếp request cho đối tượng đi sau giải quyết.
- Client : chứa tham khảo đến đối tượng đầu tiên trong dây chuyền để mỗi lần cần thực hiện request, nó sẽ gởi thông điệp tới đối tượng này.

8.3 Mẫu Template Method

Mục tiêu :

- Định nghĩa giải thuật tổng quát để giải quyết vấn đề nào đó trong một tác vụ, trong giải thuật tổng quát này có gọi 1 số tác vụ chức năng cơ bản nào đó để thực hiện công việc theo yêu cầu của giải thuật tổng quát, tuy nhiên các tác vụ cơ bản được gọi sẽ được hiện thực sau trong các class con, chứ class hiện hành cũng chưa biết chúng sẽ làm gì cụ thể.
- Như chúng ta đã trình bày nhiều lần trong tài liệu này, một trong các mục tiêu chính của việc viết chương trình là phải viết được đoạn code giải quyết đúng chức năng và có tính tổng quát hóa cao để hạn chế tối đa việc hiệu chỉnh lại. Mẫu Template Method là 1 trong những biện pháp hỗ trợ mục tiêu này.

8.3 Mẫu Template Method

Thí dụ về việc dùng mẫu Template Method :

- Giả sử ta muốn viết chương trình quản lý hệ thống file (FileManagerApp) cho phép user thực thực hiện 1 số tác vụ xử lý hệ thống file như xóa file đệ qui từ 1 thư mục xác định; đếm số lượng file con, cháu, chắt…của 1 thư mục; tìm và diệt virus tất cả các file từ thư mục xác định…
- Phân tích các chức năng của chương trình xử lý hệ thống file ta phát hiện 1 số ý tưởng sau :

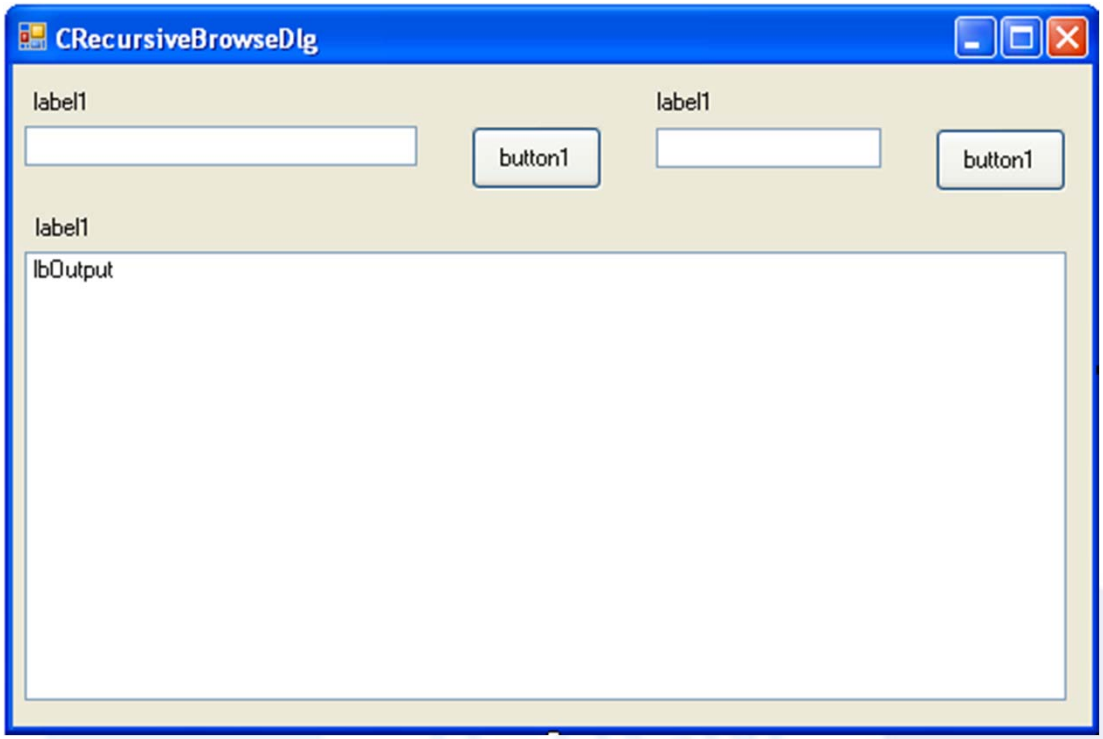
8.3 Mẫu Template Method

Thí dụ về việc dùng mẫu Template Method :

- Mỗi chức năng cần 1 form giao diện với user, nhưng may mắn là các form giao diện phục vụ các chức năng đều khá giống nhau về số lượng và tính chất các phần tử giao diện. Cụ thể mỗi form cần chứa các đối tượng giao diện như : Button để giúp user duyệt chọn thư mục xuất phát, TextBox để hiển thị đường dẫn thư mục xuất phát, TextBox để giúp user đặc tả pattern về các phần tử cần xử lý (*, *.exe,…), Button để user kích hoạt việc thực hiện chức năng, ListBox để hiển thị thông tin về kết quả xử lý…

8.3 Mẫu Template Method

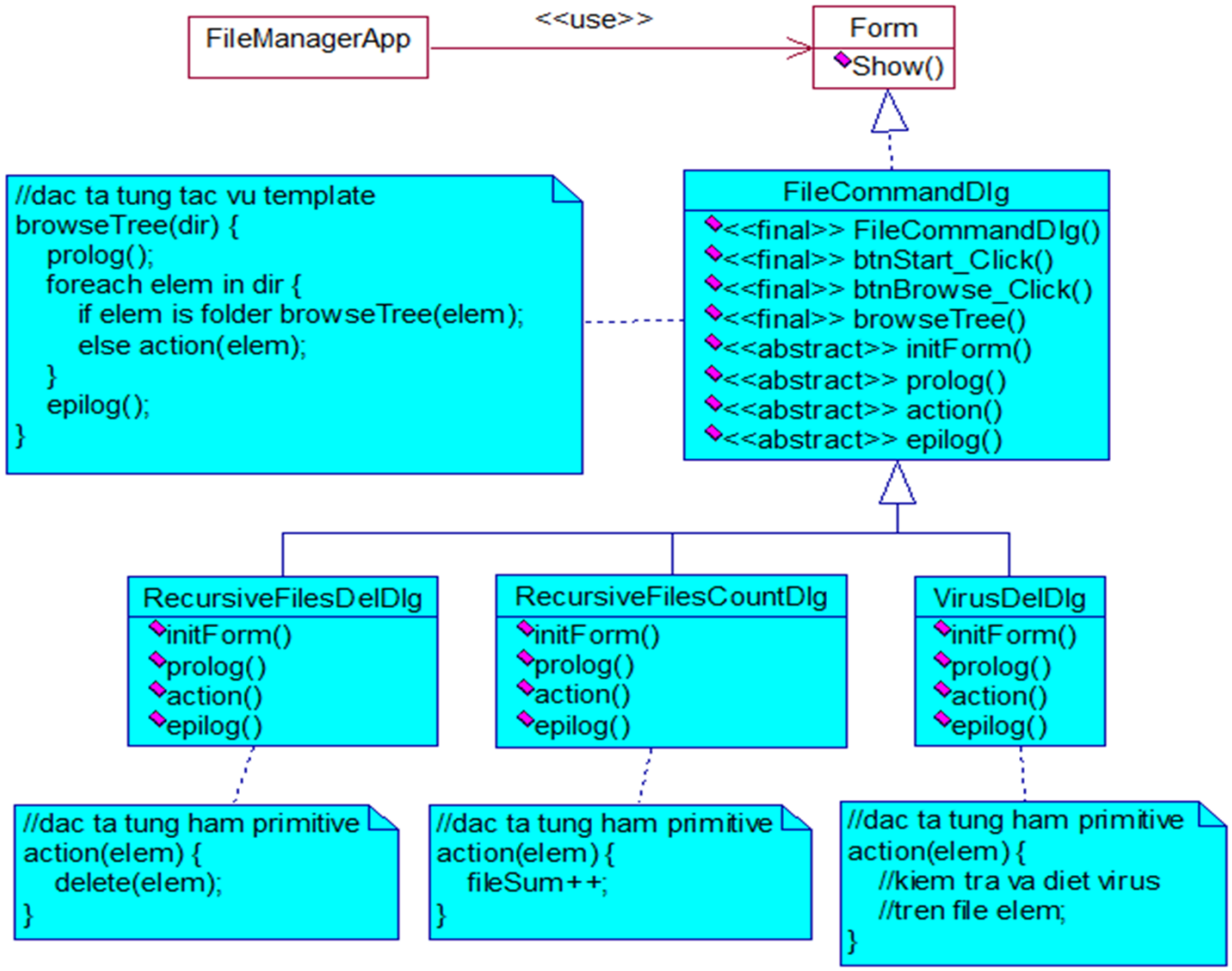
Thí dụ về việc dùng mẫu Template Method :



- Mỗi chức năng cần phải duyệt đệ qui hệ thống file, bắt đầu từ thư mục chỉ định bởi user, để lần lượt gặp từng file rồi thực hiện hoạt động xử lý xác định trên file đó.

8.3 Mẫu Template Method

Cách tốt nhất để xây dựng chương trình trên là dùng mẫu Template Method với lược đồ class như sau :



8.3 Mẫu Template Method

Class FileCommandDlg đặc tả form giao diện tổng quát cho mọi chức năng, nó có 4 tác vụ miêu tả các giải thuật tổng quát được dùng chung cho mọi class con, ta gọi các tác vụ này là “template method” :

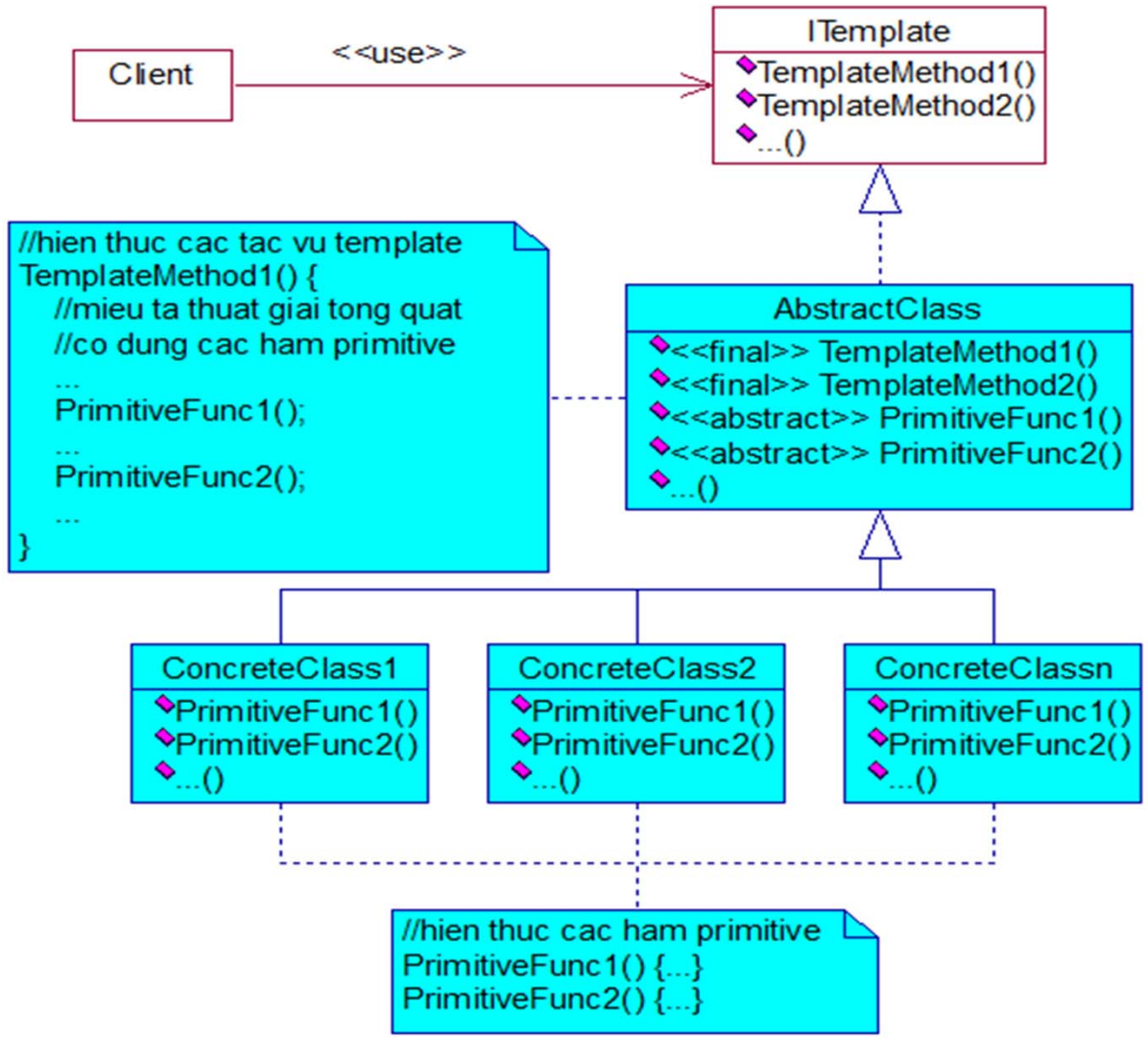
- FileCommandDlg() chứa giải thuật tạo form, tạo các đối tượng con trong form và thêm chúng vào form ở vị trí và kích thước mong muốn. Lệnh cuối cùng của giải thuật tạo form tổng quát này sẽ gọi hàm initForm() để hiệu chỉnh nội dung chuỗi văn bản được hiển thị kèm theo từng phần tử giao diện sao cho phù hợp với chức năng đặc thù.
- Lưu ý là trong các môi trường lập trình trực quan như Visual Studio .Net, người lập trình sẽ dùng tiện ích thiết kế trực quan form giao diện cho dễ dàng, nhanh chóng, chính xác. Máy sẽ tự động sinh mã cho tác vụ tạo form theo đúng yêu cầu thiết kế của user.

8.3 Mẫu Template Method

- btnBrowse_Click() là hàm xử lý sự kiện Click chuột trên button Browse của user, nó có nhiệm vụ chung là hiển thị cửa sổ duyệt chọn thư mục xuất phát.
- btnStart_Click() là hàm xử lý sự kiện Click chuột trên button Start của user, nó có nhiệm vụ chung là thực hiện chức năng trên thư mục xuất phát. Giải thuật của tác vụ này gồm 3 bước công việc : prolog() browseTree() epilog().
- browseTree() chứa giải thuật duyệt từng file 1 cách đệ qui, xuất phát từ thư mục xuất phát do user chỉ định, mỗi lần gặp 1 file thì sẽ gọi tác vụ action() thực hiện hành vi nào đó lên file.
- Các tác vụ initForm(), prolog(), action(), epilog() được dùng trong giải thuật của tác vụ template method nhưng sẽ được từng class con đặc tả cụ thể theo yêu cầu chức năng đặc thù của class con đó. Ta gọi các tác vụ này là “primitive function”.

8.3 Mẫu Template Method

Ta có thể xây dựng mẫu Template Method theo loại class pattern với lược đồ class như sau



8.3 Mẫu Template Method

Các phần tử tham gia :

- ITemplate (Form) : định nghĩa interface thống nhất của các class chức năng cần dùng, interface này thường chứa nhiều tác vụ chức năng có tính chất chung như sau : để thực hiện chức năng ta sẽ dùng giải thuật tổng quát. Như vậy các tác vụ chức năng trong interface thường là các “template method”.
- AbstractClass (FileCommandDlg) : đặc tả class cha dùng chung, class này chứa các tác vụ “template method”, mỗi tác vụ “template method” miêu tả giải thuật tổng quát để thực hiện chức năng tương ứng, trong giải thuật tổng quát có gọi các hàm “primitive function”.

8.3 Mẫu Template Method

Các phần tử tham gia (tt) :

- ConcreteClass1… (RecursiveFilesDelDlg…) : các class con, mỗi class chịu trách nhiệm override các hàm "primitive function" theo yêu cầu xử lý đặc thù của mình. Các class con này không cần và không được phép override các tác vụ "template method" đã được đặc tả 1 lần ở class cha.
- Client (FileManagerApp) : miêu tả đoạn code của client sử dụng các chức năng khác nhau.

8.4 Mẫu Strategy

Mục tiêu :

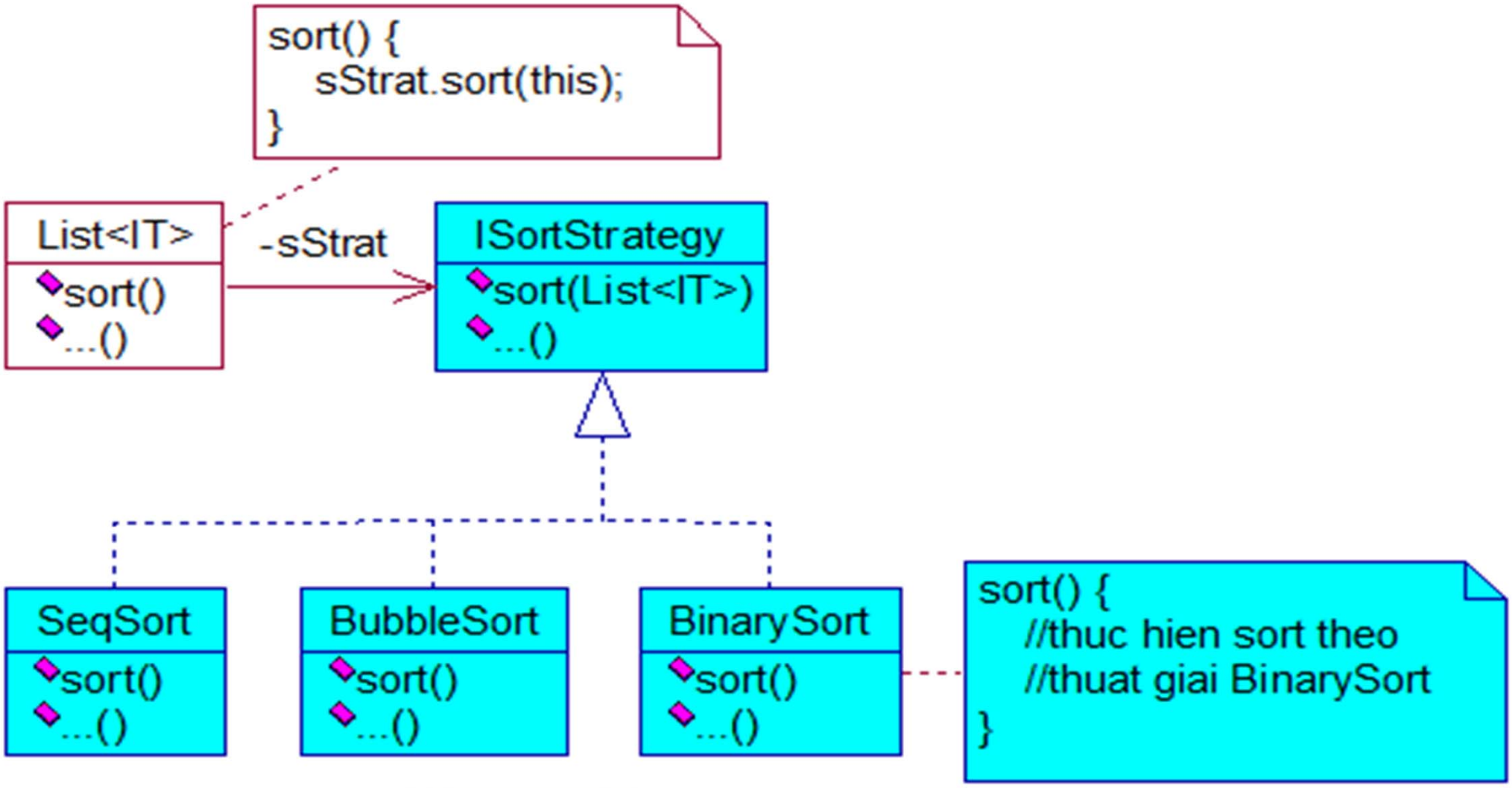
- Cung cấp một họ giải thuật khác nhau để giải quyết cùng 1 vấn đề nào đó và cho phép Client chọn lựa linh động dễ dàng một giải thuật cụ thể theo từng tình huống sử dụng.
- Về nguyên lý chung, thường có nhiều giải thuật khác nhau cùng giải quyết được 1 bài toán. Mỗi giải thuật có những ưu khuyết điểm riêng và sẽ thích hợp hơn trong ngữ cảnh sử dụng nào đó so với các giải thuật còn lại. Cách tốt nhất để giúp Client chọn lựa linh động và dễ dàng 1 giải thuật phù hợp theo từng tình huống là dùng mẫu Strategy.

8.4 Mẫu Strategy

Thí dụ về việc dùng mẫu Strategy :

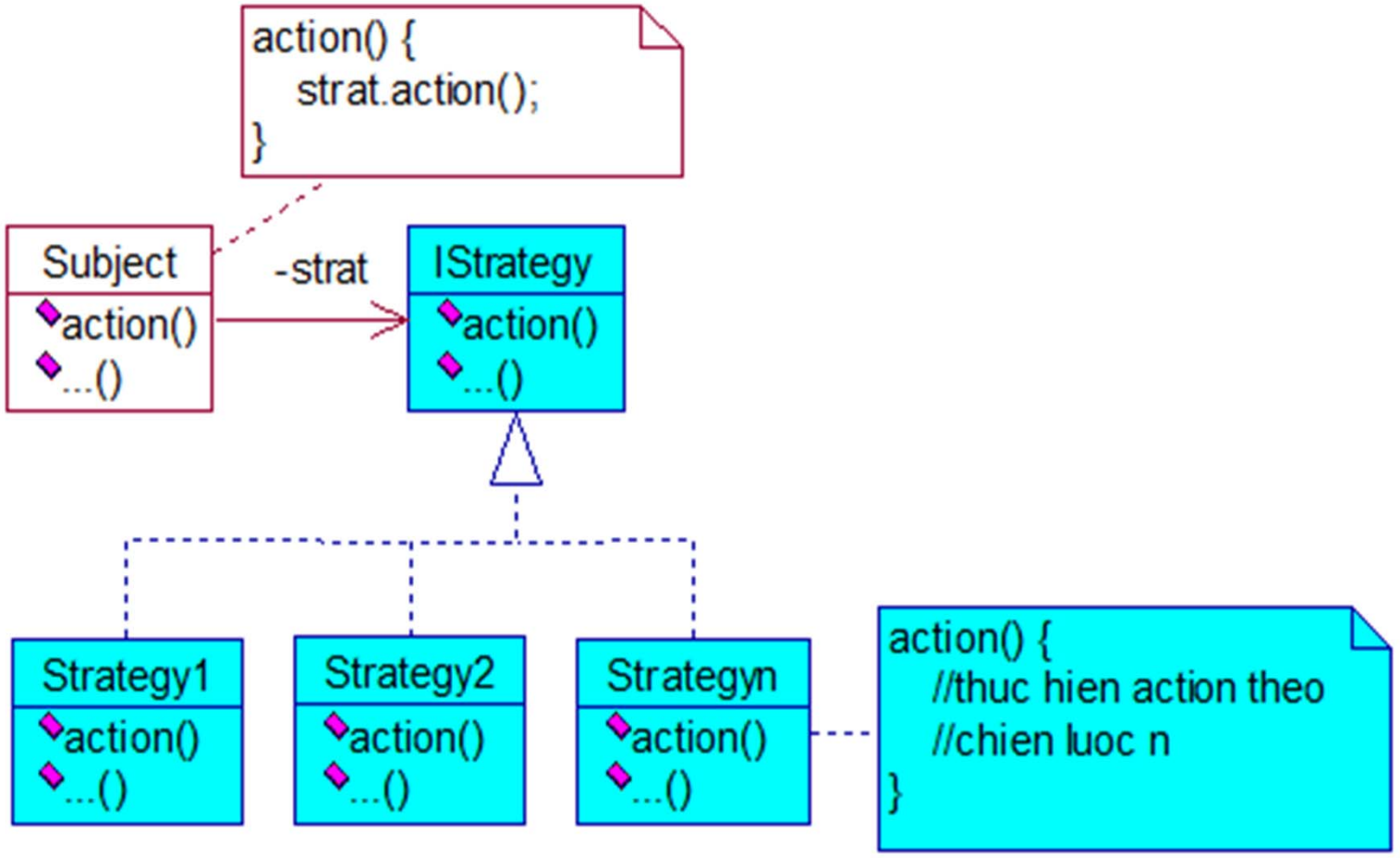
- Thí dụ để sắp xếp thứ tự các phần tử trong 1 danh sách, ta có nhiều giải thuật sắp xếp khác nhau như sắp tuần tự, bubblesort, nhị phân,… Cách tốt nhất để thiết lập linh động giải thuật sắp xếp cho danh sách và giúp code của các tác vụ chức năng trong đối tượng danh sách hoàn toàn độc lập với giải thuật sắp xếp thứ tự là dùng mẫu Strategy với lược đồ class như sau :
- Đối tượng danh sách có 1 tham khảo đến đối tượng thực hiện sắp xếp thứ tự các phần tử, tùy yêu cầu cụ thể, ta tạo đối tượng chứa giải thuật sắp xếp mong muốn và gán tham khảo đến đối tượng này vào thuộc tính tham khảo của đối tượng danh sách. Mỗi lần cần sắp xếp thứ tự các phần tử trong danh sách của mình, nó gởi thông điệp sStart.sort(this) để kích hoạt tác vụ sắp xếp thứ tự chứ nó không biết chính xác giải thuật sắp xếp nào sẽ chạy.

8.4 Mẫu Strategy



8.4 Mẫu Strategy

Ta có thể xây dựng mẫu Strategy theo loại object pattern với lược đồ class như sau :



8.4 Mẫu Strategy

Các phần tử tham gia :

- IStrategy (ISortStrategy) : định nghĩa interface cho tất cả các class thể hiện giải thuật thực hiện 1 chức năng xác định nào đó.
- Strategy1… (BinarySort…) : class miêu tả giải thuật cụ thể để giải quyết chức năng. Nó thường nhận tham khảo đến Client (đối tượng Subject) trong lúc được khởi tạo để thông qua tham khảo này, nó truy xuất dữ liệu của Client hầu phục vụ giải thuật chức năng của mình.
- Subject (List<IT>) : class đặc tả Client có sử dụng giải thuật do các class Strategy hiện thực.

8.5 Mẫu State

Mục tiêu :

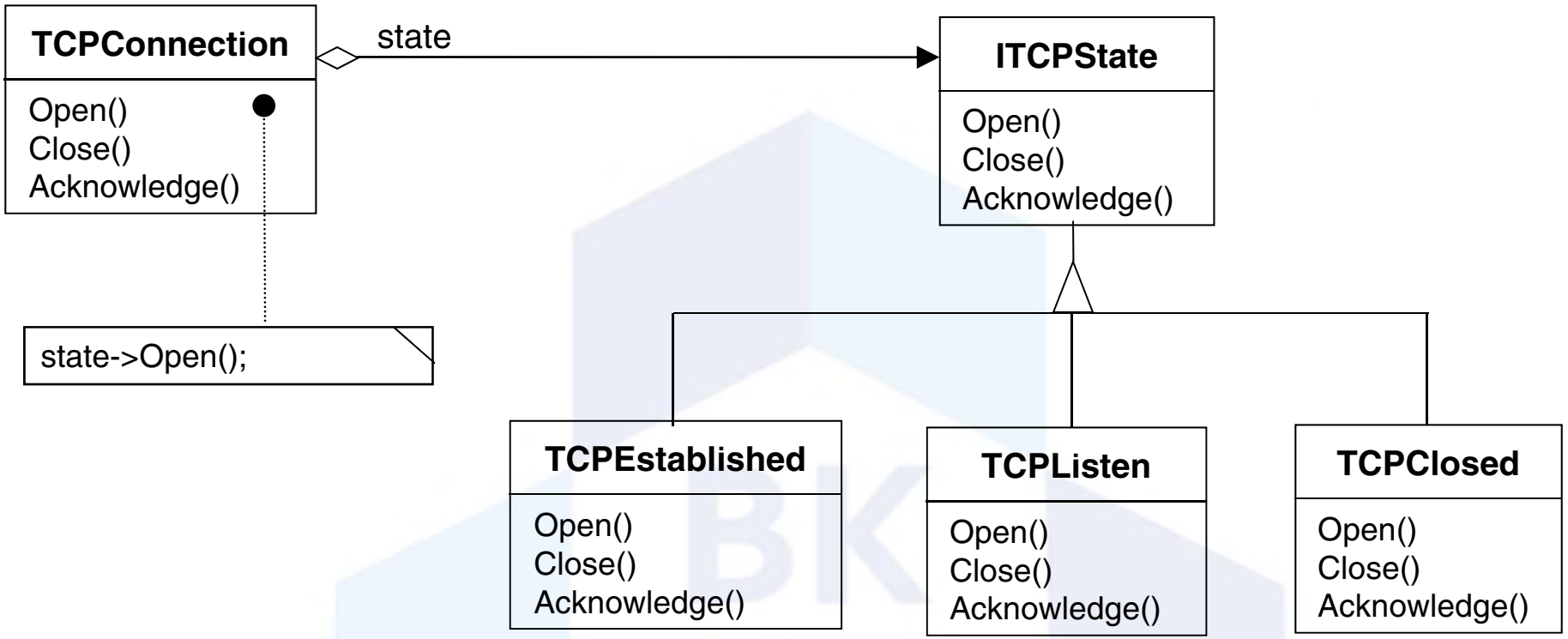
- Về nguyên lý chung, hành vi của đối tượng có thể phụ thuộc vào trạng thái hiện hành của đối tượng đó. Cách tốt nhất để giúp đối tượng thay đổi linh động và dễ dàng 1 hành vi phù hợp theo từng trạng thái là dùng mẫu State.
- Cho phép 1 đối tượng thay đổi hành vi khi trạng thái bên trong của nó thay đổi. Ta có cảm giác như class của đối tượng bị thay đổi.

8.5 Mẫu State

Thí dụ về việc dùng mẫu State :

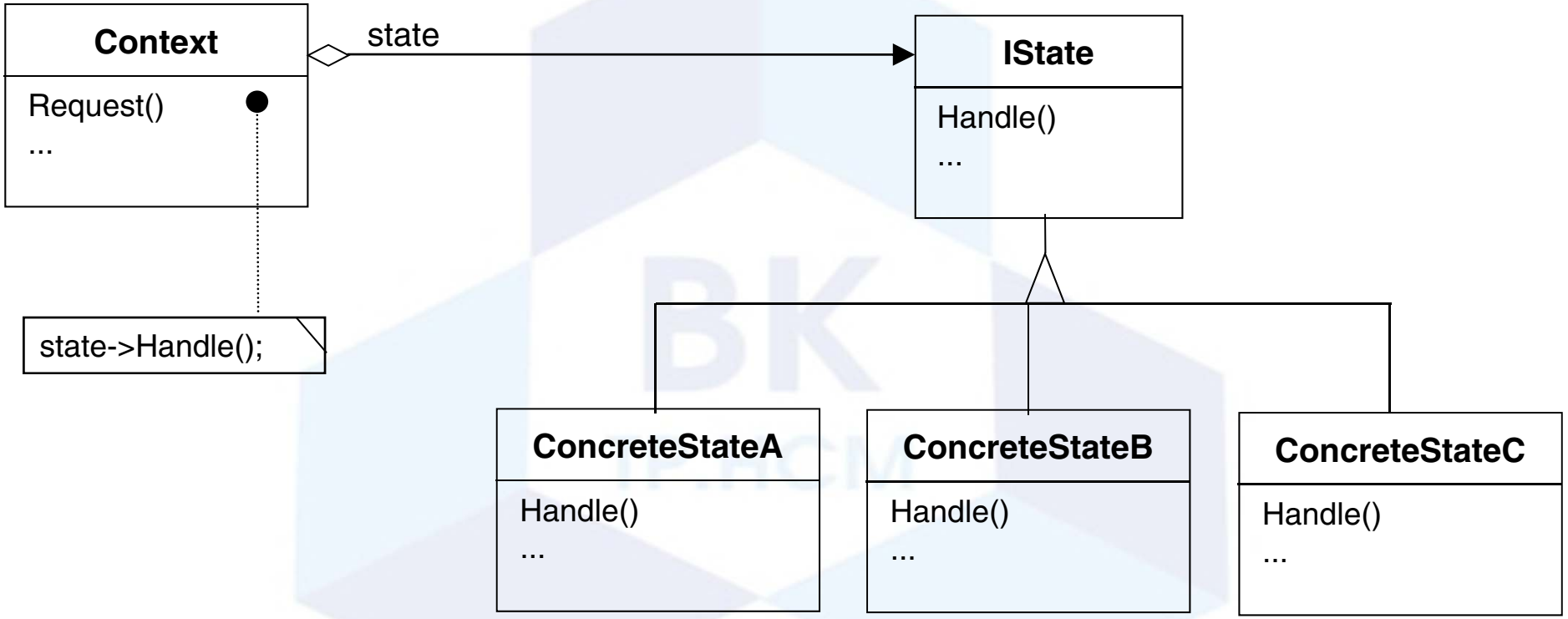
- Thí dụ trong class TCPConnection miêu tả 1 mối nối mạng, đối tượng TCPConnection có thể ở 1 trong nhiều trạng thái : Established, Listening, Closed. Khi đối tượng TCPConnection nhận request nào đó, nó sẽ đáp ứng khác nhau tùy vào trạng thái hiện hành. Cách tốt nhất để giải quyết yêu cầu trên là dùng mẫu State theo lược đồ class sau đây :

8.5 Mẫu State


TCPConnection
Open()
Close()
Acknowledge()
state
ITCPState
Open()
Close()
Acknowledge()
state->Open();
TCPEstablished
Open()
Close()
Acknowledge()
TCPListen
Open()
Close()
Acknowledge()
TCPClosed
Open()
Close()
Acknowledge()

8.5 Mẫu State

Ta có thể xây dựng mẫu State theo loại object pattern với lược đồ class như sau :



8.5 Mẫu State

Các phần tử tham gia :

- Context (TCPConnection) : định nghĩa interface cần dùng cho client. Duy trì 1 tham khảo đến đối tượng của 1 class con ConcreteState mà định nghĩa trạng thái hiện hành.
- IState (ITCPState) : định nghĩa interface nhằm bao đóng hành vi kết hợp với trạng thái cụ thể. Duy trì 1 tham khảo đến đối tượng của 1 class con ConcreteState mà định nghĩa trạng thái hiện hành.
- ConcreteState (TCPEstablished, TCPListen, TCPClose) : định nghĩa và che dấu hành vi cụ thể kết hợp với trạng thái của mình.

8.6 Mẫu Command

Mục tiêu :

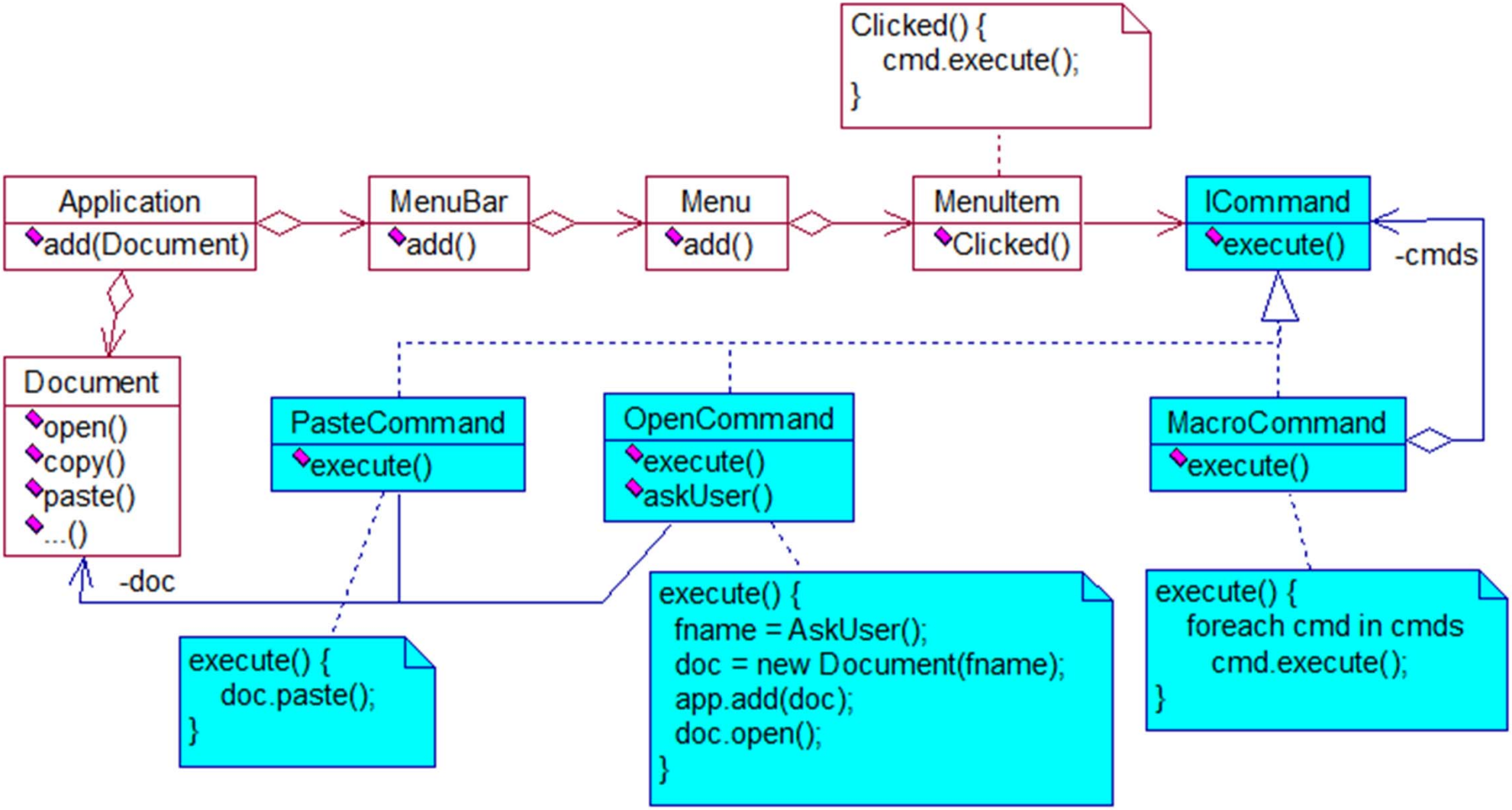
- Đóng gói đoạn code phục vụ 1 yêu cầu xác định của Client trong một đối tượng, nhờ đó có thể thông số hóa đoạn code nhận và thực hiện các thao tác trên request như sắp xếp, ghi logfile, undo,…

8.6 Mẫu Command

Thí dụ về việc dùng mẫu Command :

- Chương trình có giao diện đồ họa trực quan thường dùng nhiều cửa sổ giao diện, mỗi cửa sổ thường có 1 thanh menubar chứa nhiều menu dạng pop-up, mỗi menu pop-up chứa nhiều mục chức năng, mỗi mục chức năng có thể là 1 menu pop-up con… Cuối cùng mỗi mục chức năng cơ bản được dùng để kích hoạt chức năng tương ứng. Yêu cầu phổ biến về thanh menubar của cửa sổ chức năng là nó có thể được hiệu chỉnh động theo thời gian (thêm/bớt/thay đổi từng thành phần trong thanh menu), hoặc thậm chí muốn thay đổi hành vi đáp ứng với từng mục chức năng hiện có trong thanh menubar. Cách tốt nhất để giải quyết vấn đề này là dùng mẫu Command với lược đồ class như sau :

8.6 Mẫu Command

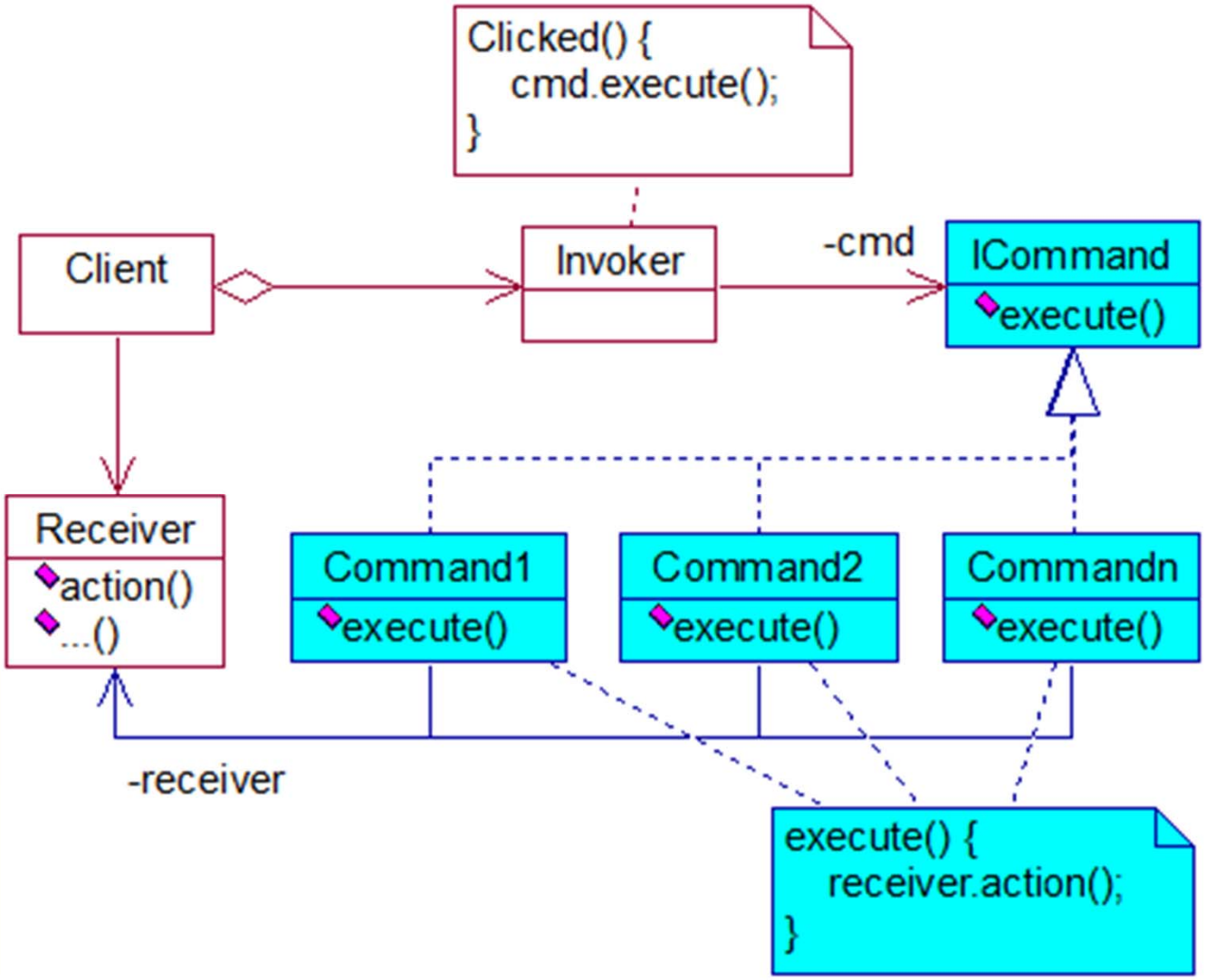


8.6 Mẫu Command

- Mỗi đoạn code thực hiện chức năng nào đó sẽ được đặt trong tác vụ execute() của 1 class tương ứng, các class này đều hỗ trợ cùng interface thống nhất ICommand.
- Mỗi đối tượng giao diện (Button, mục chức năng của menu pop-up,…) chứa 1 tham khảo đến đối tượng ICommand, hàm xử lý sự kiện Click chuột trên nó luôn được viết như sau : cmd.execute(); Lệnh này sẽ kích hoạt tác vụ execute() của đối tượng được tham khảo hiện hành chạy, còn đối tượng được tham khảo là ai là tùy theo hành vi cụ thể nào cần thực hiện.

8.6 Mẫu Command

Ta có thể xây dựng mẫu Command theo loại object pattern với lược đồ class như sau :



8.6 Mẫu Command

Các phần tử tham gia :

- ICommand : interface thống nhất cho mọi đối tượng xử lý request, nó chứa ít nhất 1 tác vụ execute() để thực hiện hành vi được yêu cầu từ client.
- Command1… (PasteCommand, OpenCommand…) : class đặc tả giải thuật thực hiện hành vi cụ thể, nó thường có thuộc tính tham khảo đến đối tượng Receiver chứa dữ liệu và tác vụ chức năng có liên quan.
- Invoker (MenuItem): đối tượng gửi request đến đối tượng ICommand để nhờ thực hiện hành vi tương ứng.
- Client (Application) : module khởi tạo đối tượng Command cụ thể và gởi cho nó tham khảo đến đối tượng Receiver.
- Receiver (Document, Application) : chứa dữ liệu và tác vụ chức năng có liên quan đến hành vi mà đối tượng Command cần thực hiện.

8.7 Mẫu Observer

Mục tiêu :

- Định nghĩa sự phụ thuộc 1-n giữa các đối tượng sao cho khi 1 đối tượng trung tâm bị thay đổi nội dung (trạng thái) thì n đối tượng phụ thuộc nó được cảnh báo hầu hiệu chỉnh tự động theo đồi tượng trung tâm, nhờ đó đảm bảo được tính nhất quán giữa chúng.
- Thường đối tượng trung tâm là đối tượng chứa dữ liệu bên trong ứng dụng, còn n đối tượng phụ thuộc nó là những đối tượng giao diện của ứng dụng. Nội dung của các đối tượng giao diện được chứa và quản lý bởi đối tượng bên trong ứng dụng.

8.7 Mẫu Observer

Thí dụ về việc dùng mẫu Observer :

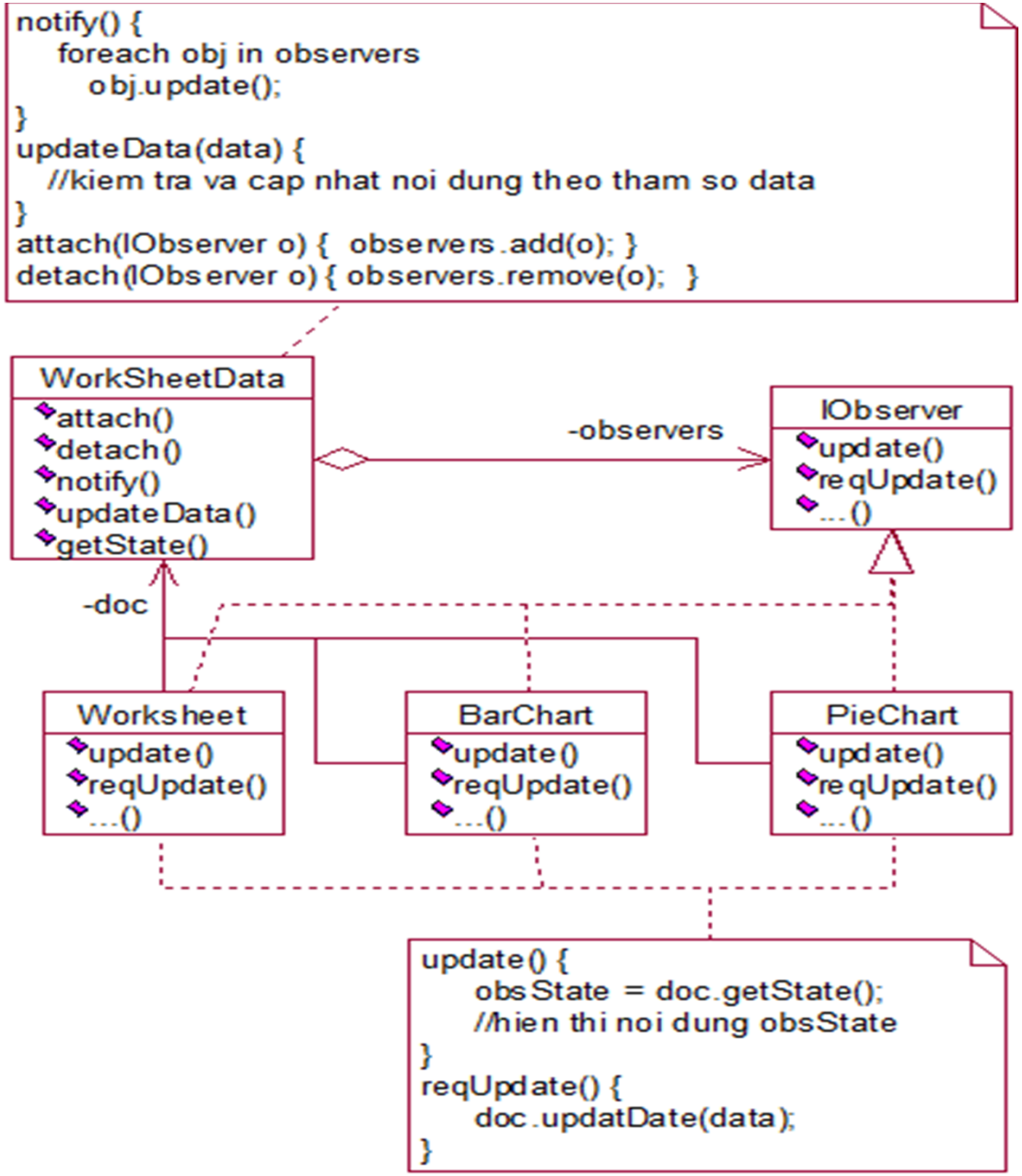
- Trong ứng dụng quản lý bảng tính (MSExcel), mỗi bảng tính là 1 bảng dữ liệu của 1 database tương ứng (ta gọi là workbook hay file *.xls). Ta có thể hiển thị nội dung của bảng dữ liệu trên nhiều đối tượng giao diện khác nhau, thí dụ như spreadsheet (bảng nội dung chi tiết 2 chiều), barchart (biểu đồ vạch), piechart (biểu đồ bánh),...
- Mỗi khi bảng dữ liệu bên trong phần mềm thay đổi nội dung bởi ai đó (có thể do phần mềm khác), nó phải gởi cảnh báo (notify) đến mọi đối tượng giao diện có dùng nội dung của nó để các đối tượng này kịp thời hiển thị lại nội dung mới.

8.7 Mẫu Observer

Thí dụ về việc dùng mẫu Observer :

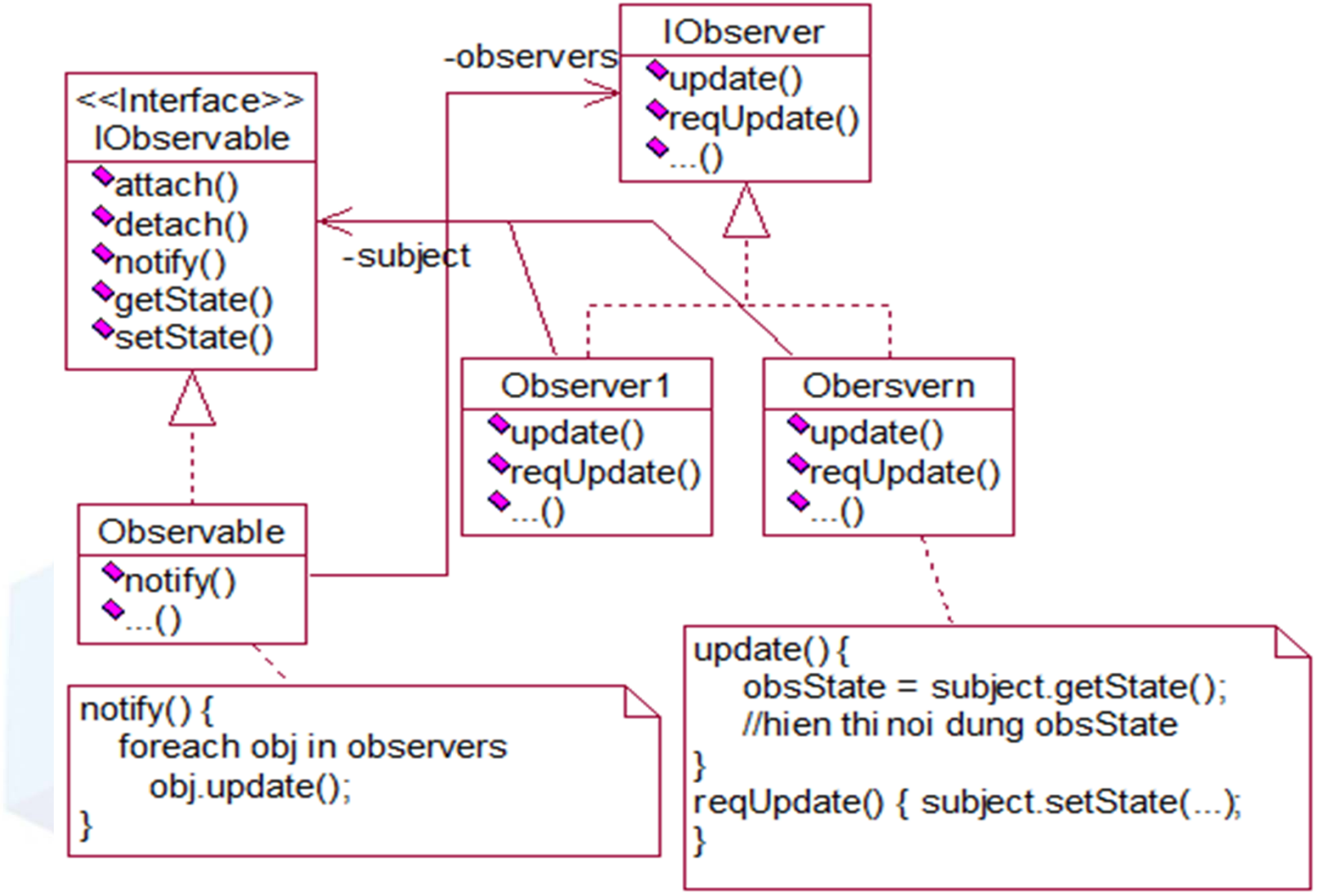
- Tương tự, nếu đối tượng giao diện nào cho phép người dùng cập nhật nội dung (spreadsheet), thì mỗi khi user cập nhật nội dung, nó không được cập nhật cục bộ mà phải gởi yêu cầu cập nhật nội dung về đối tượng trung tâm, chỉ có đối tượng này mới có quyền quyết định cập nhật hay không, nếu nó cập nhật nội dung thì nội dung sẽ bị thay đổi và như thế nó phải gởi cảnh báo cho mọi đối tượng phụ thuộc nó biết.
- Cách tốt nhất để giải quyết vấn đế trên là dùng mẫu thiết kế Observer với lược đồ class như sau :

8.7 Mẫu Observer



8.7 Mẫu Observer

Ta có thể xây dựng mẫu Observer theo loại object pattern với lược đồ class như sau :



8.7 Mẫu Observer

Các phần tử tham gia :

- IObservable : interface của đối tượng trung tâm (đóng vai trò 1 trong mối quan hệ 1-n), nó chứa các tác vụ attach, detach từng đối tượng phụ thuộc nó vào danh sách quản lý; tác vụ notify() gởi cảnh báo cho từng đối tượng phụ thuộc khi có sự thay đổi nội dung; tác vụ updateData() nhận yêu cầu thay đổi nội dung và xử lý yêu cầu.
- Observable (WorksheetData) : class đặc tả đối tượng trung tâm, nó hiện thực interface IObservable.
- IObserver : interface thống nhất của các đối tượng phụ thuộc vào đối tượng trung tâm.
- Observer1… (PieChart…) : class đặc tả đối tượng phụ thuộc cụ thể.

8.8 Kết chương

- Chương này đã giới thiệu các thông tin cơ bản về nhóm mẫu phục vụ che dấu hành vi, thuật giải trong đối tượng (Behavioral Patterns) và thông tin chi tiết cụ thể về các mẫu Chain of Responsibility, Template Method, Strategy, State, Command, Observer.