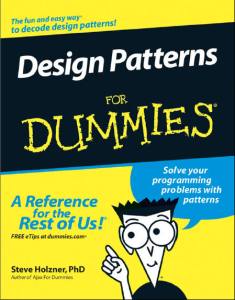
[](https://haihth.files.wordpress.com/2013/02/653b7bd97401.jpg)

## [Design Patterns for Dummies – Giới thiệu](https://haihth.wordpress.com/2013/02/20/series-bai-dich-design-patterns-for-dummies/)

Xin chào các thành viên trong diễn đàn congdongcviet !

Mình yêu máy tính, và lập trình từ nhỏ. Thông thường khi “bí” một vấn đề nào, mình lên mạng tìm kiếm. Hầu hết là tìm ra được lời giải nhờ các bạn khắp nơi từng hỏi và từng trả lời. Thiết nghĩ mình cũng nên có một chút đóng góp gì đó ngược lại cho cộng đồng lập trình viên, dù ít dù nhiều. Qua quá trình tự học và tìm tòi, cứ khi nghe tới một ngôn ngữ mới nào là mình tìm sách đọc và thực hành, nay cũng đã trải qua một số ngôn ngữ và IDE sau: đầu tiên là lập trình file .bat(trong MS-Dos), rồi Foxpro, Pascal,C,C++,Visual Basic 3.0, Java, Visual Basic 6, Visual C++6.0 MFC, Visual J++ 6.0, Visual Basic.Net, HMTL,Asp, Javascript, PHP, và gần đây nhất là C# và ASP.net. Mình nhận thấy tất cả ngôn ngữ chỉ là công cụ, và mục đích của chúng là tạo ra sản phẩm. Công cụ nào cũng có điểm mạnh, điểm yếu, nhưng nếu bạn sử dụng không đúng cách thì không thể tạo ra một sản phẩm tốt.

Vậy làm thế nào để sử dụng công cụ một cách đúng cách. Đó chính là tư duy giải thuật lập trình. Tư duy về giải thuật là cách chúng ta trừu tượng bài toán thành từng bước nhỏ, từng bước, và lắp ráp chúng thành một sản phẩm đúng.

Ví dụ để xây dựng một căn nhà, các bạn cần chuẩn bị gạch, cát, xi măng, nước, gỗ, ngói lợp… Khi có đủ nguyên vật liệu, các bạn bắt tay thực hiện từng bước như xây dựng móng nhà, xây dựng các bức tường, lợp ngói…

Trước đây mọi người lập trình theo phương pháp thủ tục, chia một vấn đề lớn thành nhiều phần nhỏ khác nhau và xử lý từng phần một theo đúng trình tự. Đây là một phương pháp tốt và từng được áp dụng trong một thời gian dài. Tuy nhiên khi chương trình ngày một càng lớn, việc duy trì và phát triển hàng trăm ngàn thủ tục là một công việc khó khăn, tốn chi phí và dễ sai sót. Khi đó phương pháp lập trình hướng đối tượng ra đời. Phương pháp này cũng chia một bài toán lớn thành các phần nhỏ, nhưng các phần nhỏ này được đóng gói vào từng đối tượng, các đối tượng này gần gũi với thực tế hơn rất nhiều, nên việc phát triển phần mềm ngày càng dễ tiếp cận với mọi người.

Ví dụ để lắp ráp một cái ti vi, chúng ta có đối tượng màn hình, đối tượng mainboard, đối tượng loa, đối tượng remote… sau đó lắp lại với nhau. Yahooo! Thật dễ dàng đúng không các bạn.

Trong topic này mình không có ý định giới thiệu về lập trình hướng đối tượng, vì sách vở về chủ đề này có quá nhiều, từ tiếng anh qua tiếng việt, và ít nhiều các bạn vào diễn đàn này đều đã nắm qua.  
Vậy chủ đề thật sự của topic này là gì?

Có ai trong các bạn từng đặt câu hỏi ”tôi có khả năng xây 1 căn nhà, căn nhà 1 tầng, 2 tầng thậm chí 3 tầng. Nhưng không biết xây căn nhà 100 tầng thì sao?” Đối với căn nhà 3 tầng, khi có sai sót, hay thay đổi ở tầng 2, bạn có thể sửa, hoặc thậm chí đập bỏ và làm lại, nhưng đối với căn nhà 100 tầng, bị sai sót ở tầng thứ 3, chẳng lẽ bạn đập bỏ cả 97 tầng còn lại?

Và đó là nguyên nhân ra đời của một thứ gọi là Design Patterns – Các Mẫu Thiết Kế. Các bạn chắc đã từng nghe ai đó nói tới mẫu thiết kế, đúng vậy, tôi nghĩ bạn đã nghe nói đến nó trong lĩnh vực xây dựng kiến trúc. Và giờ đây, chúng ta sẽ nói tới nó trong lĩnh vực xây dựng phần mềm. Vậy mẫu thiết kế là gì: Nói nôm na nó là những giải pháp để giải quyết những vấn đề thường gặp trong phát triển phần mềm theo hướng đối tượng. Nó là những bản mẫu, ta dựa vào đó xây dựng nên những đối tượng. Vậy lợi ích của nó thế nào? Đó là nó giúp ta có những đối tượng mềm dẻo, dễ thay đổi, dễ bảo trì. Và vì sao mẫu thiết kế lại dính dáng tới lập trình hướng đối tượng. Thật ra thì nó bổ sung cho lập trình hướng đối tượng, nó mở rộng khả năng to lớn cho lập trình hướng đối tượng.

Gần đây khi mình có dịp đọc tới Design Patterns, mình thật sự thấy nó cuốn hút, rất có ích cho dân lập trình. Tuy nhiên tài liệu về design patterns cũng không nhiều(mình biết khoảng hơn chục cuốn), ngôn ngữ chủ yếu là tiếng anh, tài liệu tiếng việt thì ít, chắc chỉ có 1 cuốn (do MKPUB phát hành, mình chưa được đọc, vì mình ở tỉnh, không ở Tp.HCM nên không mua được). Cách tiếp cận vấn đề của mỗi sách cũng rất khác nhau, có khi rất chuyên ngành, và trừu tượng, nhiều khi gây khó khăn cho cả người bản xứ chứ không dám nói tới người Việt mình.

Với tham vọng đóng góp cho cộng đồng, mình sẽ dịch thuật một cuốn sách mình cho là tương đối dễ tiếp cận.  
Sau khi đọc qua một số cuốn sách như sau:

* Design Patterns: Elements of Reusable Object-Oriented Software (Do bộ tứ tác giả Erich Gamma, Richard Helm, Ralph Johnson, John M.Vlissides). Cuốn kinh điển và đầy đủ nhất
* C# 3.0 Design Patterns: ByJudith Bishop. ( Nhà xuất bản Oreilly)
* Head First Design Patterns ( Nhà xuất bản Oreilly)
* Design Patterns: by Christopher G. Lasater ( Nhà xuất bản Wordware)
* C# Design Patterns: A Tutorial by James W.Cooper ( Nhà xuất bản Addison-Wesley)
* Design Patterns for Dummies: by Steve Holzner,PhD. ( Nhà xuất bản Wiley)

Mình quyết định chọn cuốn Design Patterns for Dummies, tác giả Steve Holzner,PhD để giới thiệu đến các bạn. Có các nguyên nhân sau:

* Cách dẫn dắt dễ hiểu, ví dụ sinh động, ngôn ngữ thân thiện
* Không sử dụng UML để mô tả biểu đồ, ( sẽ gây khó khăn cho các bạn chưa nắm vững UML)
* Số mẫu tương đối đầy đủ.
* Số trang sách không nhiều lắm. Chỉ khoảng 300trang.

Tuy nhiên các ví dụ trong sách là được viết từ Java. Có thể sẽ gây ra lúng túng một chút cho các bạn sử dụng C#. Nhưng các bạn cũng biết cha đẻ của C# chính là cha đẻ của Borland C, Borland C++, Delphi, Visual J++, nên các bạn sẽ thấy Java và C# không khác biệt lắm.

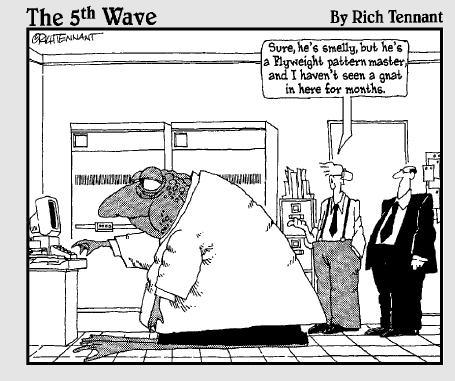
Đối với các bạn chưa thể tự mình chuyển đổi mã nguồn từ Java ->C# theo các ví dụ trong sách, mình cũng viết lại các ví dụ theo C#. Tuy không thể hiện 100% việc chuyển đổi, nhưng vẫn nổi bật được ý đồ của tác giả.

Vì trình độ anh ngữ của mình chưa tốt (chỉ tự học). Nên không đảm bảo dịch sát 100% ý đồ tác giả. Mặt khác việc dịch thuật, và chạy chương trình cũng tốn nhiều thời gian. Mỗi tuần mình sẽ đăng một chương của cuốn sách. Cuốn sách 12 chương. Tổng thời gian trong 3 tháng, các bạn sẽ được làm quen với hầu hết các mẫu thiết kế. Có thể thời gian trên so với mấy cuốn sách của SAM: “Học trong 24giờ”, sẽ làm các bạn thấy nản. Nhưng lập trình là một nghệ thuật, mà học cách để làm chủ nghệ thuật thì 3 tháng chắc chắn còn chưa đủ. Mong các bạn hãy kiên nhẫn.

Mỗi chương mình sẽ lập một topic để các bạn tiện theo dõi và tranh luận.

Thank all!

[**DP4Dummies – Chương 1 – Tổng quan các mẫu DP**](https://haihth.wordpress.com/2013/02/20/dp-chapter1/)

[](https://haihth.files.wordpress.com/2013/02/1.jpg)

**Chương 1: Xin chúc mừng, rắc rối của bạn đã được giải quyết triệt để.**

**Trong chương này, chúng ta sẽ nói đến:**

* Giới thiệu về Mẫu thiết kế Design Patterns là gì?
* Hiểu biết về tác dụng của Design Patterns
* Mở rộng lập trình hướng đối tượng
* Điểm sơ qua một số mẫu Design Pattern

Là một lập trình viên, bạn biết rằng thật khó khăn để nhớ chi tiết những việc bạn đang thực hiện. Và khi bạn không nắm bắt được tổng quát công việc, bạn có thể dễ dàng bỏ lỡ những việc quan trọng. Khi đó, mã nguồn bạn đang viết có thể vẫn còn làm việc tốt đẹp, nhưng trừ khi bạn bao quát được bức tranh lớn hơn, lúc đó mã nguồn bạn viết mới thực sự hoàn hảo.

Những vấn đề nghiêm trọng thực sự thường xuất hiện sau khi bạn đã chỉnh sửa chúng ít nhất một lần. Những nhà phát triển thường tự mình xử lý bằng cách viết lại mã nguồn và sửa các lỗi. Tuy nhiên trong môi trường công việc, những nhà lập trình thường bỏ phần lớn thời gian để bảo trì, chỉnh sửa những công việc cũ hơn là tập trung vào những sản phẩm mới.

Bạn thấy rằng thật vô lý khi cứ phải làm, rồi sửa, làm lại, sửa tiếp… Giải pháp hợp lý nhất là bạn đưa ra được một quy trình tổng quan cho việc thiết kế và bảo trì, có như vậy, bạn mới tránh được các rắc rối phát sinh khi môi trường ứng dụng thay đổi, hoặc ít nhất bạn cũng giúp cho việc bảo trì, chỉnh sữa dễ dàng hơn khi có phát sinh.

Ý tưởng đằng sau cuốn sách này là: Bạn sẽ sử dụng một tập hợp các mẫu thiết kế Design Patterns để làm đơn giản hóa quá trình trên. Kế hoạch này sẽ giúp bạn có một cái nhìn tổng quát. Một mẫu thiết kế Design Pattern là một giải pháp đã được kiểm nghiệm thành công khi đối diện một vấn đề lập trình phát sinh cụ thể. Khi bạn quen thuộc hết các mẫu thiết kế trong sách này, bạn nhìn vào một chương trình. -Bam!- Một giải pháp đúng đắn sẽ xuất hiện trong tâm trí bạn, thay vì bạn phải đập đầu vào tường trong vô vọng, giờ bạn có thể ung dung nói “Ở đây, tôi sẽ sử dụng mẫu Factory, mẫu Observer, hay mẫu Adapter…”

Đó là chưa nói, một số sách về thiết kế khuyên bạn nên dành phần lớn thời gian để phân tích và lên kế hoạch cho một đề án. Vẻ đẹp thật sự ở đây là một người nào đó đã đối mặt với vấn đề bạn đang gặp phải, họ đã có giải pháp đúng đắn cho nó. Và giờ khi bạn nhuần nhuyễn mẫu thiết kế, bạn có thể áp dụng các thiết kế đó một cách dễ dàng.

Làm sao để trở thành chuyên gia thiết kế trong lĩnh vực phần mềm, điều mà ai cũng thèm muốn? Thật dễ dàng, hãy đọc cuốn sách này, nghiền ngẫm những mẫu thiết kế mà tôi dành nhiều tâm huyết để viết. Bạn không cần phải nhớ mọi thứ, bạn chỉ cần biết là có những mẫu thiết kế đó. Và khi bạn đối diện với một vấn đề thực tế, sâu thẳm trong bạn tự nhiên thốt lên “À, có vẻ chỗ này có thể dùng mẫu Iterator…” Sau đó bạn chỉ cần tìm kiếm mẫu thiết kế đó trong cuốn sách này, duyệt qua các ví dụ để biết phải làm gì. Và vì vậy, chương này sẽ là một tour du lịch nho nhỏ, giúp bạn đi qua một số mẫu thiết kế tiện dụng và hữu ích.

**Chỉ cần tìm ra mẫu thiết kế thích hợp**

Điểm kỳ diệu của Design Patterns là nó giúp cho công việc của bạn dễ dàng tái sử dụng, dễ mở rộng và bảo trì. Khi bạn thiết kế không tốt, phần mềm của bạn không có khả năng tái sử dụng và bảo trì, khi gặp vấn đề phát sinh, bạn sẽ phải dành nhiều thời gian, có khi là nhiều hơn cả lúc bạn viết ban đầu, chỉ là để sửa chữa chúng.

Ví dụ: Bạn đang muốn tạo một đối tượng Java, nhiệm vụ là đọc và phân tích một tài liệu XML. Bạn cần phải tạo một lớp Parser (chuyên dùng để phân tích XML) sau đó bạn tạo một đối tượng của lớp này. Bạn thầm nghĩ “Tới giờ mọi việc vẫn ổn”. Nhưng thực tế thì đã có hàng tá lớp Parser do người khác viết, và họ luôn muốn sử dụng lại những tính năng đặc biệt trong lớp của họ. Nếu bạn có thể sử dụng mẫu thiết kế Nhà máy Factory, giờ đây bạn có thể sử dụng bất cứ lớp Parser nào, kể cả của những người khác, thay vì cứ khư khư xài lớp Parser do chính bạn viết ra. Và vì vậy, chương trình của bạn đã trở nên dễ mở rộng, tái sử dụng được và bảo trì dễ dàng.

Nói cách khác, Mẫu thiết kế Design Patterns là những giải pháp giúp cho ta có một thiết kế tốt khi đối diện những vấn đề phát sinh trong việc lập trình. Nhiều người đã gặp vấn đề này, và đã giải quyết tốt, việc của bạn là áp dụng chúng. Bạn không cần phải ghi nhớ mọi thứ, chỉ cần nhìn ra đâu là mẫu thiết kế phù hợp và đặt nó vào đúng chỗ.

Thật tuyệt đúng không các bạn.

**Đôi nét về cuốn sách tên “Gang of Four” Bộ tứ tác giả.**

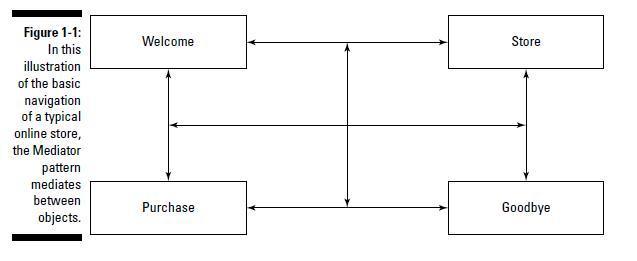
Quyển sách là tập hợp 23 mẫu thiết kế được phát hành bởi Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides, trong một nghiên cứu của họ năm 1995. Với tựa đề gốc là “Design Patterns: Elements of Reusable Object-Oriented Software”. Tạm dịch “Mẫu thiết kế: những thành phần tái sử dụng trong lập trình hướng đối tượng”. Họ được giới lập trình gọi là Bộ tứ Gang of Four, hay GoF. (ND: Bộ tứ ở đây là ẩn dụ với Tên nhóm nhạc nổi tiếng Gang Of Four của Anh hay Bộ tứ quyền lực Mafia trong tác phẩm Bố già hay là bộ tứ quyền lực chính trị của Trung quốc, ở VN cũng có bộ tứ quyền lực của Vietnam Next Top Model…).

Đã có rất nhiều sự thay đổi kể từ khi xuất hiện, một số trong 23 mẫu được sử dụng nhiều, số khác ít khi được sử dụng. Tôi sẽ nói đến đầy đủ 23 mẫu trong cuốn sách này, nhưng tôi sẽ nhấn mạnh ở những mẫu thường được sử dụng hơn. Và kể cả mẫu mới không có trong sách của GoF, trong chương 11.

Có một sự thật là, bạn không chỉ phải nhớ kỹ từng mẫu thiết kế, bạn phải hiểu sâu sắc về nó, để có thể áp dụng đúng đắn trong thực tiễn. Tôi cũng sẽ lưu ý nhiều về lập trình hướng đối tượng xuyên suốt quyển sách này. Lập trình hướng đối tượng OOP là một bước tiến tuyệt vời trong lĩnh vực lập trình. Nhưng có quá nhiều lập trình viên sử dụng chúng một cách tùy tiện, thiếu chiều sâu, và điều đó gây ra nhiều rắc rối tiềm ẩn.

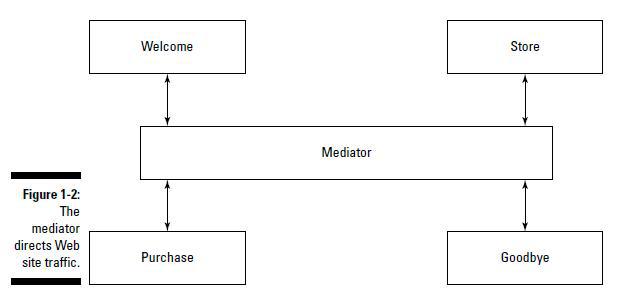
Phần lớn việc tìm hiểu những mẫu thiết kế chính là việc mở rộng khái niệm lập trình hướng đối tượng. Ví dụ: đóng gói những gì thay đổi nhiều nhất (encapsulating what changes most), cách chuyển đổi một quan hệ kế thừa is-a sang quan hệ kết hợp has-a ( xem chương 2 để biết chi tiết) – và tôi sẽ nói chi tiết về chúng.

**Hãy bắt đầu bằng mẫu Mediator Pattern (Người trung gian)**

[](https://haihth.files.wordpress.com/2013/02/f1-1.jpg)

Hình bên trên là một ví dụ về mẫu thiết kế, mẫu Mediator. Hình cho chúng ta thấy chức năng của một mẫu Mediator. Theo hình ta đang có một website với 4 trang. Website cho phép khách hàng duyệt qua kho hàng và đặt mua. Khách hàng có thể đi  từ trang này qua trang khác theo đường vẽ trên hình. Ở đây có một vấn đề phát sinh. Tại từng trang, bạn phải viết mã để nhận biết khi nào khách hàng muốn nhảy qua trang khác và kích hoạt trang đó. Tại một trang bạn có quá nhiều đường để đi tới trang khác, và vì vậy sẽ phát sinh nhiều đoạn code trùng lặp trên nhiều trang khác nhau.

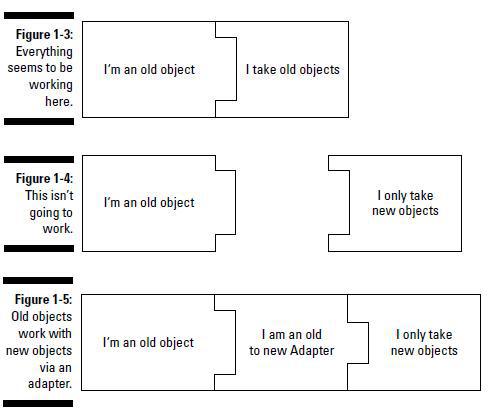
Bạn có thể sử dụng mẫu Mediator để đóng gói tất cả các đường dẫn tới trang vào một module duy nhất, và đặt nó vào trong một đối tượng Mediator. Từ bây giờ, từng trang chỉ cần phải thông báo bất cứ sự thay đổi nào cho Mediator, và Mediator tự biết dẫn trang cần thiết cho khách hàng, như trong hình bên dưới.

[](https://haihth.files.wordpress.com/2013/02/f1-2.jpg)

Bạn có thể tạo ra một Mediator với chức năng dẫn trang. Tại đây bạn có thể chỉnh sửa và thay đổi dễ dàng. Đó chính là chức năng của Mediator (Người trung gian)

**Chuyển đổi với mẫu thiết kế Adaptor (Người chuyển đổi)**

Đây là một mẫu khác, mẫu chuyển đổi Adaptor.

[](https://haihth.files.wordpress.com/2013/02/f1-3-4-5.jpg)

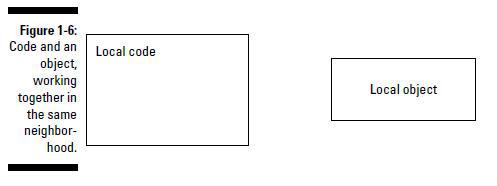
Hãy nhìn vào hình 1-3. Đầu vào là một đối tượng cũ. Hệ thống tiếp nhận đối tượng cũ.  
Với hình 1-4. Khi hệ thống thay đổi, hệ thống không tiếp nhận đối tượng cũ nữa, chỉ tiếp nhận đối tượng mới. (ND: thực ra hình ảnh 1-4 có chút chưa chính xác, phần “I only take new objects, nên vẽ nhỏ lại)  
Hình 1- 5. Đây là nơi xuất hiện mẫu Adaptor (Người chuyển đổi). Mục đích là chuyển đổi đối tượng cũ, thành đối tượng mới, khi đó hệ thống sẽ sẵn sàng tiếp nhận đối tượng này.

“ND: Các bạn hãy tưởng tượng. Các bạn có một chiếc tivi với đầu cắm điện 3 chân. Ổ cắm điện ở nhà bạn là loại ổ 2 chân. Bạn ra ngoài cửa tiệm, mua 1 cục chuyển đổi, từ 3 chân ra 2 chân. Lúc đó bạn có thể sử dụng được ổ điện 2 chân rồi. Cục chuyển đổi từ 3 chân ra 2 chân, chính là Adaptor”

Vấn đề đã được giải quyết. Ai nói học các mẫu thiết kế là khó khăn nhỉ.

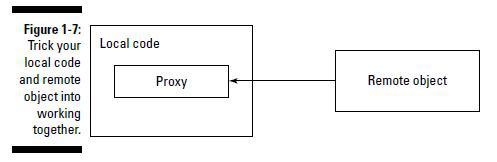
**Đứng trong một đối tượng. Mẫu Proxy. (Người đại diện)**

Đây là một mẫu khác. Mẫu Proxy. Mẫu này nói rằng, mã bạn viết chỉ tương tác với đối tượng cục bộ như hình dưới

[](https://haihth.files.wordpress.com/2013/02/f1-6.jpg)

Thực tế phát sinh, bạn buộc phải tương tác với một đối tượng ở xa, đâu đó trên thế giới. Làm sao bạn có thể làm cho chương trình của bạn tương tác với một đối tượng cục bộ trong khi thực tế là nó đang làm việc với một đối tượng ở xa.

Ở đây mẫu Proxy (Người đại diện) xuất hiện. Nó là một đối tượng nằm bên trong chương trình, làm trách nhiệm tương tác với chương trình, giúp cho chương trình tưởng rằng nó đang tương tác cục bộ, thay vì tương tác với một đối tượng từ xa. Bên trong, Proxy chịu trách nhiệm kết nối với đối tượng từ xa. Như hình dưới

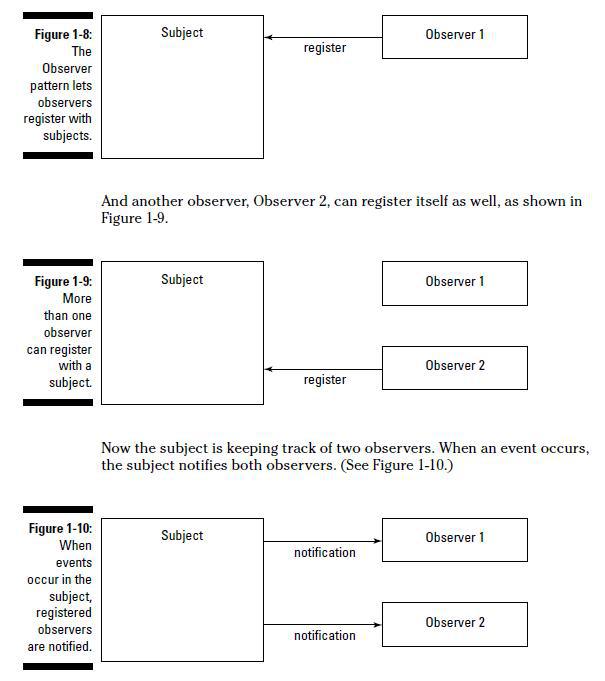
[](https://haihth.files.wordpress.com/2013/02/f1-7.jpg)

Bạn sẽ biết cách mẫu Proxy hoạt động trong chương 9.

**Đôi nét về mẫu Observer (Người quan sát)**

Bạn có thể quen thuộc với một vài mẫu trong sách này, ví dụ như mẫu Observer này chẳng hạn.

Mẫu Observer có thể đăng ký với hệ thống. Khi hệ thống có sự thay đổi, hệ thống sẽ thông báo cho Observer biết. Khi không nữa cần, mẫu Observer sẽ được gỡ khỏi hệ thống.

[](https://haihth.files.wordpress.com/2013/02/f1-8-9-10.jpg)

Hình 8 cho thấy mẫu Observer cho phép 1 observer đăng ký với hệ thống. Hình 9, cho phép observer thứ 2 đăng ký với hệ thống. Hiện tại hệ thống đang liên lạc với 2 observer. Khi hệ thống phát sinh một sự kiện cụ thể nào đó, nó sẽ thông báo với cả 2 observer như hình số 10.

Tôi cố gắng trình bày tất cả các Mẫu thiết kế theo cách dễ hiểu, dễ tiếp cận nhất. Bạn sẽ không phải nhìn vào đống biểu đồ, cùng với các lớp trừu tượng đầy phức tạp nữa. Các chương trong sách nhắm vào các độc giả là lập trình viên, rất hữu ích cho các bạn, cho dù các bạn không đọc hết tất cả các mẫu. Các mẫu thiết kế trong sách này đã trở thành các tiêu chuẩn về lập trình trên thế giới, và chúng hữu dụng cho các bạn, dù bạn đang ở trình độ nào. Hy vọng rằng, trong tương lai, khi bạn đối diện với chương trình của mình, bạn đột nhiên nhận ra: Aha, đây chính là mẫu Façade.