DP4Dummies – Chương 6: Adapter, Facade

Chương 6: ĐƯA MỘT CÁI CHỐT HÌNH TRÒN VÀO MỘT LỖ HÌNH VUÔNG VỚI MẪU CHUYỂN ĐỔI ADAPTER VÀ MẪU MẶT TIỀN FAÇADE

Trong chương này, chúng ta sẽ đi qua các nội dung sau:

- Sử dụng mẫu chuyển đổi Adapter
- Tạo một adapter
- Chuyển đổi một đối tượng Ace thành đối tượng Acme
- Xử lý các rắc rối với việc chuyển đổi
- Sử dụng mẫu façade

Đôi khi một đối tượng không thật sự phù hợp với những gì ta mong muốn. Một lớp có thể thay đổi, hoặc một đối tượng trở nên khó khăn khi sử dụng. Chương này sẽ là giải pháp tốt cho vấn đề trên khi sử dụng hai mẫu thiết kế: mẫu chuyển đổi adapter và mẫu façade. Mẫu adapter cho phép bạn chuyển đổi một đối tượng để cung cấp cho một lớp khác có thể sử dụng chúng. Mẫu façade cũng tương tự vậy, nó thay đổi vẻ ngoài của một đối tượng, nhưng có một chút khác biệt: bạn sử dụng mẫu này để đơn giản hóa các chức năng của nó, làm cho nó dễ làm việc với đối tượng hay lớp khác.

Kịch bản của mẫu Adapter

"Được," trưởng nhóm phát triển MegaGigaCo nói, khi đang bước vào phòng, "Thu dọn mọi thứ, quản lý đã ra lệnh rằng chúng ta phải chuyển đổi cơ sở hạ tầng sang một hệ thống mới, chúng ta vừa mua từ công ty của cháu Giám đốc"

"Hmm", các lập trình viên nói, "Đó là một rắc rối. Giao diện khách hàng trực tuyến của chúng ta cho phép khách hàng sử dụng phần mềm từ công ty Ace và đóng gói chúng trong một lớp tên Ace. Làm sao để chúng ta có thể chuyển đổi chúng cho phù hợp với hệ thống mới?

"Chỉ có thể là một đối tượng Acme mới", trưởng nhỏm nói "không phải là đối tượng Ace"

"Oh, không", mọi người nói "Làm sao có thể như vậy được"

Bạn có thể thấy rắc rối ở đây. Hiện tại, hệ thống phù hợp với một đối tượng Ace như hình vẽ

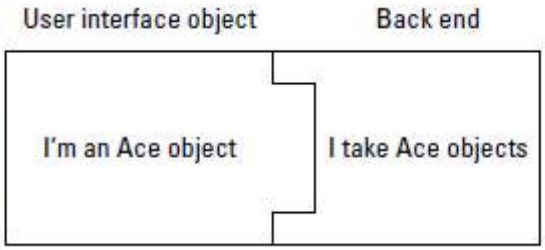

Figure 6-1: These two objects fit together well.

Nhưng khi hệ thống thay đổi, nó yêu cầu một đối tượng Acme (không phải Ace), vì vậy đối tượng Ace không thích hợp nữa. Xem hình sau:

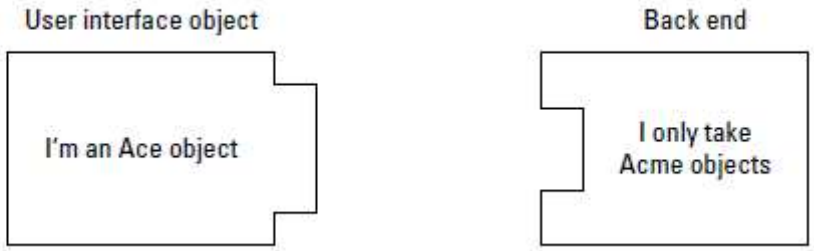

Figure 6-2: The user interface no longer works with the back end.

"Tôi có một giải pháp". Bạn nói. Mọi người quay sang, và bạn nói tiếp "Dĩ nhiên, với tư cách là một tư vấn viên, tôi sẽ tính chi phí cao cho việc này"

"OK", trưởng nhóm nói.

"Bạn cần sử dụng mẫu chuyển đổi Adapter", bạn giải thích, "Mẫu adapter cho phép bạn chuyển đổi một lớp hoặc một đối tượng sang một lớp hoặc đối tượng mới mà bạn mong muốn". Bạn vẽ ra giải pháp lên tấm bảng như hình sau:

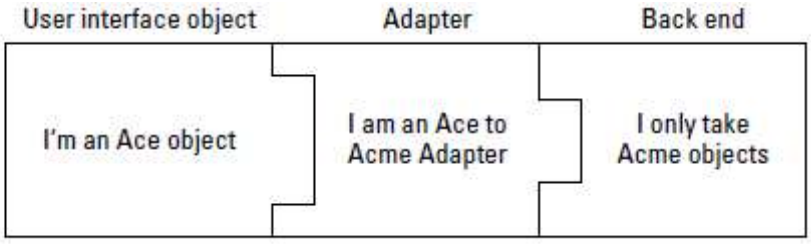

Figure 6-3: The adapter fits the pieces together.

"Ah" cả nhóm phát triển phần mềm nói "Chúng tôi đang hiểu ra vấn đề"

"Tốt" bạn nói "Vậy phải trả phí cho tôi"

Chỉnh sửa rắc rối khi kết nối với mẫu Adapter:

Mẫu thiết kế Adapter cho phép bạn sửa đổi một giao diện giữa đối tượng và một lớp mà không phải sửa đổi trực tiếp lên chúng. Khi bạn làm việc với một ứng dụng mua sẵn, sản phẩm bạn nhận được thường không tương thích với những gì bạn thật sự muốn.

Đây là phần đặc biệt quan trọng trong phát triển trực tuyến. Ngày càng nhiều các công ty tạo ra các sản phẩm cho những công ty lớn, họ đang bỏ qua các phần mềm cho những công ty nhỏ. Và điều đó thật đáng xấu hổ vì việc tương thích của hệ thống, phần mềm từ công ty nhỏ không thể giao tiếp với một hoặc nhiều thành phần khác trong toàn hệ thống. Nhưng việc chuyển sang một giải pháp đắt tiền thì không phải lúc nào cũng cần thiết. Thông thường, giải pháp có thể là một bộ chuyển đổi nhỏ.

Cách tốt nhất để xem mẫu Adapter làm việc là thông qua ví dụ. Hiện tại giao tiếp người dùng công ty MegaGigaCo mà tôi đã nhắc tới trong phần trước của chương này, dữ liệu người dùng được đóng gói trong lớp Ace. Lớp này quản lý tên của khách hàng, với hai hàm sau:

- setName
- getName

Nhưng theo bạn biết, công ty MegaGigaCo đang chuyển sang sử dụng phần mềm Acme, mà cách thức quản lý khách hàng có khác một chút. Vấn đề ở đây là phần mềm Acme cần một đối tượng Acme. Đối tượng này có tới bốn hàm, chứ không phải hai, dùng để quản lý tên khách hàng. Chúng là:

- setFirstName
- setLastName
- getFirstName
- getLastName

Vì vậy bạn cần một bộ chuyển đổi để chắc chắn rằng hệ thống mới Acme có thể xử lý được đối tượng Ace. Bộ chuyển đổi này gọi hai hàm của đối tượng Ace và mở rộng chúng thành bốn hàm mà đối tượng Acme yêu cầu, như hình sau:

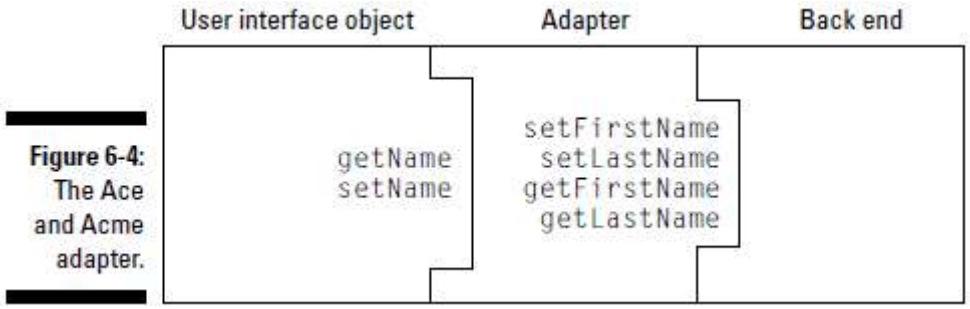

Figure 6-4: The Ace and Acme adapter.

Đây chính là cách thức làm việc của mẫu Adapter

Ghi nhớ: Sách của GoF định nghĩa mẫu Adapter như sau: "Chuyển đổi giao tiếp của một lớp sang một kiểu giao tiếp khác mà khách hàng mong muốn. Mẫu adapter cho phép các lớp có thể làm việc với nhau cho dù giao tiếp của chúng không tương thích nhau"

Thông qua định nghĩa chính thức của mẫu Adapter nói về các lớp, mẫu này bao gồm hai phần chính như sau: một cho đối tượng, một cho lớp. Chúng ta sẽ xem xét cả hai trong chương này.

Bạn sử dụng mẫu Adapter khi bạn cố gắng đưa một cái chốt hình vuông vào cái lỗ hình tròn. Nếu một lớp có giao tiếp không tương thích, bạn có thểm thêm vào một bộ chuyển đổi – giống như bộ chuyển đổi điện trong những chuyến du lịch toàn cầu – để có thể đạt được yêu cầu mong muốn

Mẫu này đặc biệt tốt trong trường hợp bạn đang làm việc với mã nguồn cũ mà yêu cầu là không được thay đổi mã cũ, trong khi phần mềm giao tiếp với mã nguồn cũ này lại thay đổi.

Giờ đã đến lúc để mẫu Adapter làm việc.

Tạo một đối tượng Ace:

Trước khi công ty con của MegaGigaCo phá hỏng mọi thứ trong phòng bạn, đối tượng Ace đang quản lý khách hàng với hai hàm như sau: setName và getName – đây là một giao diện với hai hàm sau:

```
public interface AceInterface
{
  public void setName(String n);
  public String getName();
}
```

Đối tượng Ace sẽ được tạo ra từ lớp AceClass, lớp này hiện thực giao diện trên

```
public class AceClass implements AceInterface
{
    .
    .
    .
}
```

Với hai hàm setName và getName đơn giản như sau:

```
public class AceClass implements AceInterface
{
  String name;

  public void setName(String n)
  {
    name = n;
  }

  public String getName()
  {
    return name;
  }
}
```

Đó là tất cả những gì bạn cần trong hệ thống cũ. Một đối tượng Ace được trả về. Tuy nhiên hiện tại, công ty bạn chuyển sang hệ thống Acme, hệ thống mới cần giao tiếp với đối tượng Acme. (Một lần nữa xin cảm ơn công ty con MegaGigaCo)

Tạo đối tượng Acme

Đối tượng Acme cần quản lý tên khách hàng với bốn hàm sau: setFirstName, setLastName, getFirstName và getLastName. Đây là giao diện của nó, AcmeInterface. Mã như sau:

```
public interface AcmeInterface
{
  public void setFirstName(String f);
  public void setLastName(String l);
  public String getFirstName();
  public String getLastName();
}
```

Đối tượng Acme được tạo từ lớp Acme, hiện thực giao diện AcmeInterface như sau:

```
public class AcmeClass implements AcmeInterface
{
    .
    .
    .
}
```

Và đây là bốn hàm của lớp Acme

```
public class AcmeClass implements AcmeInterface
{
  String firstName;
  String lastName;

  public void setFirstName(String f)
  {
    firstName = f;
  }

  public void setLastName(String l)
  {
    lastName = l;
  }

  public String getFirstName()
  {
    return firstName;
  }

  public String getLastName()
  {
    return lastName;
  }
}
```

Tới lúc này, bạn đã có đối tượng Ace, và đối tượng Acme. Bây giờ bạn cần một bộ chuyển đổi để gắn đối tượng Ace và hệ thống mới Acme.

Tạo đối tượng chuyển đổi Ace-to-Acme

Bạn muốn tạo một bô chuyển đổi để giúp cho ứng dụng có thể làm việc với đối tượng Ace (cho dù nó mong muốn một đối tượng Acme), vì vậy bạn phải tạo một đối tượng chuyển đổi. Đối tượng chuyển đổi này làm việc với sự "kết hợp" composition (xem chương hai để biết thêm về "kết hợp" composition) – một bộ chuyển đổi lưu trữ chính đối tượng mà nó muốn chuyển đổi.

Tiếp tục với ví dụ trong chương này, tôi đặt tên cho bộ chuyển đổi là AceToAcmeAdapter và bởi vì nó trông giống một đối tượng Acme, nó sẽ hiện thực giao diện AcmeInterface:

```
public class AceToAcmeAdapter implements AcmeInterface
{
    .
    .
    .
}
```

Bộ chuyển đổi này sử dụng đối tượng kết hợp "composition" để lưu giữ đối tượng chuyển đổi, đó là một đối tượng AceClass. Bạn có thể chuyển đối tượng này tới lớp thông qua hàm khởi dựng. Mã như sau:

```
public class AceToAcmeAdapter implements AcmeInterface
{
  AceClass aceObject;

  public AceToAcmeAdapter(AceClass a)
  {
    aceObject = a;
  }
  .
  .
  .
}
```

Điểm khác biệt giữa đối tượng Ace và Acme là đối tượng Ace chứa tên khách hàng như là một chuỗi duy nhất, trong khi đối tượng Acme lưu trữ tên và họ khách hàng riêng biệt. Để chuyển đổi giữa đối tượng Ace và Acme, tôi tách phần tên trong đối tượng Ace ra thành tên và họ. Bạn vẫn có thể nhận được tên khách hàng lưu trữ trong đối tượng Ace khi sử dụng hàm getName. Mã như sau:

```
public class AceToAcmeAdapter implements AcmeInterface
{
  AceClass aceObject;
  String firstName;
  String lastName;

  public AceToAcmeAdapter(AceClass a)
  {
    aceObject = a;
    firstName = aceObject.getName().split(" ")[0];
    lastName = aceObject.getName().split(" ")[1];
  }
}
```

Bây giờ bạn đã có tên và họ của khách hàng. Để tương thích với đối tượng Acme, bạn phải hiện thực các hàm của Acme như setFirstName, setLastName, getFirstName và getLastName. Mã như sau:

```
public class AceToAcmeAdapter implements AcmeInterface
{
  AceClass aceObject;
  String firstName;
  String lastName;

  public AceToAcmeAdapter(AceClass a)
  {
    aceObject = a;
    firstName = aceObject.getName().split(" ")[0];
    lastName = aceObject.getName().split(" ")[1];
  }

  public void setFirstName(String f)
  {
    firstName = f;
  }

  public void setLastName(String l)
  {
    lastName = l;
  }

  public String getFirstName()
  {
    return firstName;
  }

  public String getLastName()
  {
    return lastName;
  }
}
```

Tuyệt vời. Bạn đã có một bộ chuyển đổi. Nó làm việc ra sao?

Cho chạy thử bộ chuyển đổi:

Thông qua phần trên, bạn đã chuyển đổi một đối tượng Ace để chúng trông giống như một đối tượng Acme. Giờ là lúc để thấy cách mẫu Adapter làm việc. Bắt đầu bằng việc tạo một đối tượng Ace chứa thông tin khách hàng tên Cary Grant.

```
public class TestAdapter
{
  public static void main(String args[])
  {
    AceClass aceObject = new AceClass();

    aceObject.setName("Cary Grant");
      .
      .
      .
}
```

Sau đó bạn chuyển đối tượng Ace sang cho đối tượng AceToAcmeAdapter

```
public class TestAdapter
{
  public static void main(String args[])
  {
    AceClass aceObject = new AceClass();

    aceObject.setName("Cary Grant");

    AceToAcmeAdapter adapter = new AceToAcmeAdapter(aceObject);
      .
      .
      .
}
```

Và tiếp tục, bạn có thể sử dụng các hàm của Acme như getFirstName và getLastName một cách dễ dàng, không rắc rối gì:

```
public class TestAdapter
{
  public static void main(String args[])
  {
    AceClass aceObject = new AceClass();

    aceObject.setName("Cary Grant");

    AceToAcmeAdapter adapter = new AceToAcmeAdapter(aceObject);

    System.out.println("Customer's first name: " +
      adapter.getFirstName());
    System.out.println("Customer's last name: " +
      adapter.getLastName());
  }
}
```

Kết quả nhận được:

```
Customer's first name: Cary
Customer's last name: Grant
```

Đó là những gì bạn mong muốn khi bạn sử dụng đối tượng Acme thật sự. Bạn đã sử dụng một đối tượng Acme từ một đối tượng Ace nhờ bộ chuyển đổi. Đó là cách thức làm việc của mẫu Adapter. Một Adapter sử dụng một "composition" để lưu trữ một đối tượng mà nó muốn chuyển đổi, và khi các hàm của bộ chuyển đổi được gọi, nó sẽ thay đổi các hàm trong đối tượng gốc sang các hàm của đối tượng mới. Đoạn mã gọi một bộ chuyển đổi sẽ không cần quan tâm tới cách thức làm việc của bộ chuyển đổi, vì bộ chuyển đổi đã thực hiện bên trong và trả về giá cần thiết cho hệ thống.

Sử dụng một đối tượng kết hợp "composition" để bao bọc đối tượng chuyển đổi là một thiết kế hướng đối tượng tốt, như những gì chúng ta đã nói tới trong chương hai. Và chú ý

rằng nếu bạn kế thừa một đối tượng adapter, bộ bao bọc adapter sẽ có thể quản lý các đối tượng kế thừa với sự thay đổi ít nhất.

ĐƠN GIẢN HÓA CUỘC SỐNG VỚI MẪU FACADE

Một mẫu thiết kế tương tự với mẫu Adapter là mẫu Façade. Hai mẫu này làm việc theo cùng một cách, nhưng mục đích sử dụng của chúng khác nhau. Mẫu adapter chuyển đổi mã nguồn để làm việc được với mã nguồn khác. Nhưng mẫu Façade cho phép bạn bao bọc mã nguồn gốc để nó có thể giao tiếp với mã nguồn khác dễ dàng hơn.

Ví dụ như, có một người thiết kế ra một cái máy in, và đưa cho bạn một cách tự hào. "Làm sao để máy có thể in?" bạn hỏi

"Đầu tiên," anh ta nói với bạn, "gọi hàm khởi động."

"Được" bạn nói. "Bây giờ nó in chưa?"

"Chưa, bạn phải gọi turnFanOn"

"OK, giờ nó sẽ in chứ", bạn hỏi

"Chưa, hãy gọi hàm làm nóng máy warmUp"

"Được rồi. Giờ nó sẽ in, phải không?"

"Vẫn chưa. Bạn phải gọi hàm getData để đưa dữ liệu từ máy vi tính tới máy in"

"OK, hàm getData. Còn gì nữa không?"

"Hàm định dạng dữ liệu formatData"

"Và gì nữa?"

"Hàm kiểm tra đầu mực checkToner, hàm kiểm tra giấy checkPaperSupply, hàm kiểm tra hệ thống runInternalDiagnostic, hàm ..."

"Khoan đã", bạn nói, vậy viết cho tôi một hàm đại diện façade cho tất cả cái đống lộn xộn này. Hàm façade này sẽ gọi tất cả các hàm khác bên trong nó, và đơn giản hóa một cách

đáng kể việc giao tiếp. “Chỉ vậy thôi”

“Đó là cái gì?”, người thiết kế máy in hỏi

“Hàm in ấn print”, bạn nói. “Chỉ cần gọi hàm in ấn print, và máy in hoạt động. Không cần làm gì khác.”

“Hey”, anh ta nói “đó có thể là một ý tưởng tuyệt vời. Bây giờ bạn có thể thêm một hàm prepareToCallThePrintMethod , một hàm callThePrintMethod, một hàm cleanupAfterPrinting, một hàm…”

“Anh đúng là hết thuốc chữa” bạn nói.

Mẫu thiết kế Façade tạo ra một giao diện OOP dễ dàng để sử dụng. Nó là một vấn đề thiết kế cơ bản – nếu một đối tượng hay một lớp quá khó để giao tiếp, mẫu Façade tạo ra cho bạn một giao diện để giao tiếp dễ dàng hơn. Đây là định nghĩa chính thức của GoF về mẫu Façade: “Cung cấp một giao tiếp duy nhất cho tập hợp các giao tiếp của hệ thống. Façade định nghĩa một giao tiếp cao hơn để giúp các hệ thống con dễ dàng sử dụng”

Thông thường, bạn sử dụng mẫu Façade khi bạn làm việc với những mã nguồn đã được đóng gói một cách cẩu thả. Không phải ai cũng là một chuyên gia OOP, và bạn cũng nhanh chóng thấy được rằng, khi bạn làm việc trong một môi trường phát triển phần mềm thương mại rộng lớn. Khi bạn trở nên mệt mỏi để giao tiếp với những giao diện được thiết kế rắc rối và nhận thấy rằng, bạn nhận được x,y thay vì một cái z đơn giản hơn. Đó là lúc bạn nên sử dụng một giao diện mới.

(ND: từ “giao diện” trong các đoạn văn trên có nguyên văn tiếng anh là interface – tôi vẫn chưa biết dùng từ tiếng việt nào phù hợp, có thể hiểu nó là một nền tảng, cách thức giao tiếp giữa các đối tượng)

Ý tưởng rất đơn giản; một façade làm đơn giản hóa một giao tiếp interface (sử dụng nghĩa chung của từ “interface”, không phải giao diện interface trong Java) giữa một lớp hay một đối tượng.

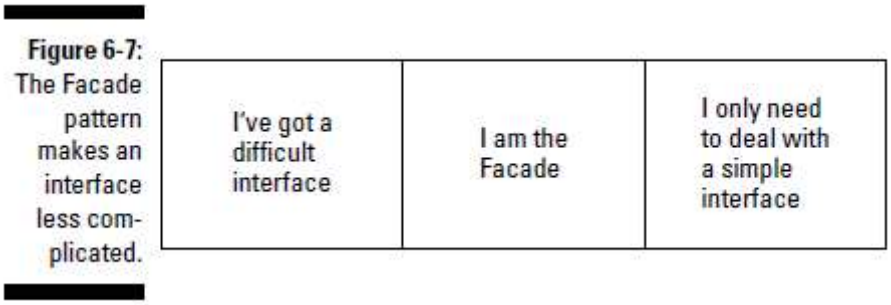

Figure 6-7: The Facade pattern makes an interface less complicated.

Bạn thường sử dụng mẫu thiết kế Façade khi bạn muốn mã nguồn đơn giản hơn nhưng lại không thể chỉnh sửa mã nguồn cũ. Thông qua việc sử dụng mẫu Façade bạn có thể giải quyết vấn đề, nó thêm vào một lớp khác bên trên, và nếu mã nguồn của lớp bên dưới thay đổi, bạn cũng phải thay đổi luôn mã nguồn của mẫu Façade.

Có một định nghĩa OOP làm việc ở đây, đôi khi còn được gọi là "Nguyên tắc về sự hiểu biết ít nhất" , đôi lúc được gọi là "Luật của Demeter" , đôi khi được gọi là "sự đóng gói hiệu quả". Đây là ý tưởng nâng cao sự hiệu quả của OOP, bạn không muốn những thực thể riêng biệt (lớp hay đối tượng) phải biết quá nhiều về nhau. Càng ít càng tốt, bạn có thể che dấu chi tiết của từng lớp hay đối tượng và làm cho sự liên kết của chúng lỏng lẻo càng nhiều càng tốt. (Xem chương bốn để biết thêm về "tháo lỏng các mối liên kết" loose coupling) . Nếu một đối tượng cấn phải biết quá nhiều về đối tượng khác, đó chính là lúc cần sử dụng mẫu Façade.

Ghi chú: Hãy tháo lỏng các mối liên kết càng nhiều càng tốt.

Làm việc với một đối tượng khó khăn

Đây là một ví dụ minh họa cách thức mẫu Façade làm việc. Công ty bạn vừa mua một công ty đối thủ, người quản lý đang rất hân hoan.

"Hmm" bạn nói " Chúng ta đang gặp rắc rối với vấn đề tương thích?. Sản phẩm của họ quá khác biệt với chúng ta"

"Phi lý", Ông chủ lớn nói "Mọi việc không thể đơn giản hơn"

"Được," bạn nói "Làm sao ông có thể đặt tên cho nó"

"Không thể đơn giản hơn. Cứ gọi hàm setFirstNameCharacter. Nó sẽ đặt kí tự đầu cho tên"

"OK, vậy kí tự thứ hai của tên thì sao?"

“Chỉ cần gọi hàm setSecondNameCharacter. Không thể đơn giản hơn”

“OK, để tôi tiếp tục.” bạn nói “Vậy để đặt tên cho một sản phẩm, ông gọi hàm setFirstNameCharacter để thiết lập kí tự đầu tiên cho tên, sau đó gọi hàm setSecondNameCharacter để thiết lập kí tự thứ hai, và bằng cách đó ông tiếp tục gọi hàm setFiveMillionNameCharacter để thiết lập kí thứ thứ năm triệu cho tên?”

“Không”, Ông chủ lớn nói “Bạn chỉ có thể đặt tên với bảy kí tự”

“À,” bạn nói “Không thể đơn giản hơn”

“Đúng”, ông chủ nói

Đây là đoạn mã, sau khi bạn hợp nhất với sản phẩm của công ty mới, lớp DifficultProduct

```
public class DifficultProduct
{
  public DifficultProduct()
  {
  }
  .
  .
  .
}
```

Bạn đặt tên cho sản phẩm này từng kí tự một, sử dụng hàm setFirstNameCharacter, setSecondNameCharacter, setThirdNameCharacter, và hàm cho kí tự thứ tư, thứ năm… như sau:

```
public class DifficultProduct
{
  char nameChars[] = new char[7];

  public DifficultProduct()
  {
  }

  public void setFirstNameCharacter(char c)
  {
    nameChars[0] = c;
  }

  public void setSecondNameCharacter(char c)
  {
    nameChars[1] = c;
  }

  public void setThirdNameCharacter(char c)
  {
    nameChars[2] = c;
  }

  public void setFourthNameCharacter(char c)
  {
    nameChars[3] = c;
  }

  public void setFifthNameCharacter(char c)
  {
    nameChars[4] = c;
  }

  public void setSixthNameCharacter(char c)
  {
    nameChars[5] = c;
  }

  public void setSeventhNameCharacter(char c)
  {
    nameChars[6] = c;
  }
  .
  .
  .
}
```

Và để lấy được tên sản phẩm, bạn gọi hàm getName, trả về một chuỗi.

```
public class DifficultProduct
{
  char nameChars[] = new char[7];

  public DifficultProduct()
  {
  }

  public void setFirstNameCharacter(char c)
  {
    nameChars[0] = c;
  }

  public void setSecondNameCharacter(char c)
  {
    nameChars[1] = c;
  }

  public void setThirdNameCharacter(char c)
  {
    nameChars[2] = c;
  }

  public void setFourthNameCharacter(char c)
  {
    nameChars[3] = c;
  }

  public void setFifthNameCharacter(char c)
  {
    nameChars[4] = c;
  }

  public void setSixthNameCharacter(char c)
  {
    nameChars[5] = c;
  }

  public void setSeventhNameCharacter(char c)
  {
    nameChars[6] = c;
  }

  public String getName()
  {
    return new String(nameChars);
  }
}
```

Đây là cách tạo một máy in, đặt tên theo từng kí tự một:

```
DifficultProduct difficultProduct = new DifficultProduct();

difficultProduct.setFirstNameCharacter('p');
difficultProduct.setSecondNameCharacter('r');
difficultProduct.setThirdNameCharacter('i');
difficultProduct.setFourthNameCharacter('n');
difficultProduct.setFifthNameCharacter('t');
difficultProduct.setSixthNameCharacter('e');
difficultProduct.setSeventhNameCharacter('r');
```

“Thấy không?” ông chủ nói “không thể dễ dàng hơn”

“Quá sức phi lý.” bạn nói ”Tôi sẽ viết một mẫu Façade”

Tạo một mẫu Façade đơn giản:

Ông chủ đồng ý cách của bạn và bạn bắt đầu viết một façade đơn giản như sau:

```
public class SimpleProductFacade
{
  public SimpleProductFacade()
  {
  }
  .
  .
  .
}
```

Façade này phải bao bọc đối tượng (đối tượng DifficultProduct trong ví dụ). Thông thường, bạn viết một façade mà cho phép façade chỉnh sửa giao diện bên ngoài của đối tượng. Bạn cũng có thể chuyển tham số cấu hình vào hàm khởi dựng của façade, nhưng điều đó không cần thiết trong ví dụ này, ta chỉ cần tạo mới đối tượng DifficultProduct như sau:

```
public class SimpleProductFacade
{
  DifficultProduct difficultProduct;

  public SimpleProductFacade()
  {
    difficultProduct = new DifficultProduct();
  }
  .
  .
  .
}
```

Rắc rối với việc sử dụng đối tượng DifficultProduct nguyên thủy là cách mà ta thiết lập tên cho nó, sử dụng vụng về một loạt các hàm setFirtNameCharacter, setSecondNameCharacter, setThirdNameCharacter và vân vân. Để sửa chữa , bạn quyết định cung cấp một façade với một hàm đơn giản là setName để đặt tên cho đối tượng. Mã như sau:

```
public class SimpleProductFacade
{
  DifficultProduct difficultProduct;

  public SimpleProductFacade()
  {
    difficultProduct = new DifficultProduct();
  }

  public void setName(String n)
  {
    char chars[] = n.toCharArray();

    if(chars.length > 0){
      difficultProduct.setFirstNameCharacter(chars[0]);
    }

    if(chars.length > 1){
      difficultProduct.setSecondNameCharacter(chars[1]);
    }

    if(chars.length > 2){
      difficultProduct.setThirdNameCharacter(chars[2]);
    }

    if(chars.length > 3){
      difficultProduct.setFourthNameCharacter(chars[3]);
    }

    if(chars.length > 4){
      difficultProduct.setFifthNameCharacter(chars[4]);
    }

    if(chars.length > 5){
      difficultProduct.setSixthNameCharacter(chars[5]);
    }

    if(chars.length > 6){
      difficultProduct.setSeventhNameCharacter(chars[6]);
    }
  }
  .
  .
  .
}
```

Có một hàm không cần dùng Façade, đó là hàm getName, mã như sau:

```
public class SimpleProductFacade
{
  DifficultProduct difficultProduct;

  public SimpleProductFacade()
  {
    difficultProduct = new DifficultProduct();
  }

  public void setName(String n)
  {
    char chars[] = n.toCharArray();

    if(chars.length > 0){
      difficultProduct.setFirstNameCharacter(chars[0]);
    }
    .
    .
    .
    if(chars.length > 6){
      difficultProduct.setSeventhNameCharacter(chars[6]);
    }
  }

  public String getName()
  {
    return difficultProduct.getName();
  }
}
```

Bây giờ bạn đã bao bọc một đối tượng khó giao tiếp và hợp nhất các hàm rắc rối thành hàm dễ sử dụng. Hãy xem cách thức hoạt động của mẫu Façade

Chạy thử mẫu Façade

Đầu tiên tạo một đối tượng SimpleProductFacade, sau đó thiết lập tên đối tượng “printer” với hàm setName và nhận lại với hàm getName.

```
public class TestFacade
{
  public static void main(String args[])
  {
    TestFacade t = new TestFacade();
  }

  public TestFacade()
  {
    SimpleProductFacade simpleProductFacade =
      new SimpleProductFacade();

    simpleProductFacade.setName("printer");

    System.out.println("This product is a " +
      simpleProductFacade.getName());
  }
}
```

Và đây là kết quả:

```
This product is a printer
```

Bạn đã chinh phục được đối tượng khó khăn với giao diện khó sử dụng bằng một façade.

Download source code C# tại đây

Design Patterns For Dummies
Design Patterns
< DP4Dummies – Chương 5: Singleton, Flyweight
> DP4Dummies – Chương 7: Template, Builder