DP4Dummies – Chương 8: Iterator, Composite

CHƯƠNG VIII:

XỬ LÝ TẬP HỢP VỚI MẪU ITERATOR VÀ MẪU COMPOSITE

Trong chương này:

- Sử dụng mẫu Iterator (đối tượng lặp lại)
- Tạo một đối tượng Iterator
- Duyệt qua danh sách các phó giám đốc bằng một Iterator nội tại
- Hiểu được mẫu Composite (tập hợp)
- Sử dụng một Iterator bên trong một Composite

Giám đốc điều hành của công ty GianDataPool, một công ty mà bạn mới chuyển đến với vị trí tư vấn, vừa đi khẽ vào phòng làm việc của bạn và nói lầm bầm gì đó.

“Gì vậy?” bạn hỏi.

Vị giám đốc nhìn quanh với vẻ mặt bí mật, và nói “Tôi có một dự án tuyệt mật dành cho bạn”

“Tuyệt mật?” bạn nói “Nó nói về cái gì?”

“Đừng to tiếng!” vị giám đốc nói khẽ. “Chúng ta cần một người khách quan cho chuyện này, Vì vậy tôi mới gặp anh. Chúng ta dường như đang gặp phải một số vấn đề với việc quản trị và chúng ta cần phải theo dõi các phó giám đốc – Không ai được biết việc này. Bây giờ, có thể có hai hay vài vị phó giám đốc làm việc như một lập trình viên vậy”

“Thừa thầy thiếu thợ”, bạn thở dài “Chuyện dài tập của các công ty”

“Chúng ta bắt đầu với khu vực bán hàng,” vị giám đốc nói khẽ “Anh có thể viết một chương trình duyệt qua hết hồ sơ và in chúng ra chứ?”

“Còn hơn thế nữa”, bạn nói. “Tôi sẽ sử dụng mẫu Iterator”

Chương này nói về hai mẫu có quan hệ mật thiết với nhau: mẫu Iterator và mẫu Composite. Mẫu Iterator cung cấp cho bạn cách thức truy cập một bộ phận bên trong một đối tượng mà không cần phải hiểu rõ cấu trúc nội tại của đối tượng đó. Ví dụ, hãng Sun đã giới thiệu một kiểu tập hợp trong việc biểu diễn các mối quan hệ trong ngôn ngữ Java, những tập hợp này cho phép bạn tạo iterator – một đối tượng đặc biệt được thiết kế cho phép bạn truy cập một phần tử của tập hợp – để cung cấp một cách thức truy cập dễ dàng.

Mẫu Composite cũng nói về tập hợp. Với mẫu Composite, ý tưởng là bạn có thể một cấu trúc hình cây nơi mà từng đối tượng sẽ thuộc về một cái cây -là một nút lá không có nút con, hoặc là một nhánh cây với nhiều nút lá con – để có thể xử lý trong cùng một cách. Mẫu Composite được thiết kế cho phép bạn xử lý nhiều đối tượng khác chủng loại trong cùng một tập hợp theo cùng một cách, và một đối tượng lặp iterator lại vô tình phù hợp tại đây – dùng để xử lý từng phần tử của một nhánh cây – ví dụ, bạn có thể duyệt qua hết cây. Chúng ta sẽ thảo luận về hai mẫu trong chương này.

Truy cập đối tượng với mẫu Iterator

Khi bạn làm việc với một tập hợp nhiều đối tượng, mẫu Iterator là một giải pháp tốt. Hàng ngày, bạn phải làm việc với nhiều loại tập hợp như cấu trúc cây, cây nhị phân, mảng, vòng đệm, bảng băm, danh sách mảng và vân vân… Cách thức mà tập hợp này lưu trữ đối tượng của nó rất khác nhau, và nếu bạn muốn truy cập dữ liệu của những đối tượng này, bạn phải học những kỹ thuật khác nhau cho từng loại tập hợp.

Và đó là nơi mẫu Iterator xuất hiện. Bạn có thể sử dụng một giao diện interface được xác định rõ ràng để truy cập tới từng phần tử của tập hợp. Trong những năm qua, các phương pháp cơ bản đã dần trở nên thích hợp hơn, và chúng cũng xuất hiện xuyên suốt chương này. Sử dụng những phương pháp này, bạn có thể truy xuất tới các phần tử trong tập hợp theo cách cơ bản nhất.

Ghi nhớ: Theo sách của Gang of Four (Gof), bạn có thể sử dụng mẫu thiết kế Iterator để "Cung cấp một cách thức truy cập tuần tự tới các phần tử của một đối tượng tổng hợp, mà không cần phải tạo dựng riêng các phương pháp truy cập cho đối tượng tổng hợp này"

Nói cách khác, một Iterator được thiết kế cho phép bạn xử lý nhiều loại tập hợp khác nhau bằng cách truy cập những phần tử của tập hợp với cùng một phương pháp, cùng một cách thức định sẵn, mà không cần phải hiểu rõ về những chi tiết bên trong của những tập hợp này.

Gợi ý: Mẫu thiết kế Iterator đặc biệt quan trọng khi tập hợp bạn đang xây dựng được tạo thành từ những tập hợp con riêng rẽ, ví dụ khi bạn chỉnh sửa bảng băm với danh sách mảng, chẳng hạn.

Thông tin: Iterator thường được viết trong Java như là những lớp độc lập. Tại sao những Iterator có thể làm việc được trong các tập hợp khác nhau? Chúng có thể, nhưng trong Java, còn ngôn ngữ khác, chúng không thể. Ý tưởng thiết kế này là một trong những kỹ thuật được gọi là "đơn trách nhiệm" – một lớp chỉ có duy nhất một công việc để làm. Hãy suy nghĩ rằng tập hợp duy trì các phần tử, một iterator cung cấp cách thức làm việc với các phần tử đó. Tách biệt trách nhiệm giữa các lớp rất hữu dụng khi một lớp bị thay đổi – Nếu có quá nhiều thứ bên trong một lớp đơn lẻ, sẽ rất khó khăn để viết lại mã nguồn. Khi diễn ra sự thay đổi, một lớp "đơn trách nhiệm" sẽ chỉ có một lý do duy nhất để thay đổi.

Truy cập đối tượng của bạn với một Iterator

Bạn bắt đầu làm việc với rắc rối giám đốc, đó là phải theo dõi các phó giám đốc. Trong trường hợp này, bạn quyết định lưu các phó giám đốc vào trong một tập hợp, với một tập hợp các chức năng cho phép truy xuất các vị này. Trong phiên bản đầu tiên này, các chức năng cơ bản mà một Iterator phải có như sau:

- was
- first
- next
- isDone
- currentItem

Ngày này Java đã hỗ trợ một giao diện iterator trong java.util.Iterator, được định nghĩa với ba phương pháp sau:

- next
- hasNext
- remove

Hàm next trả về phần tử kế tiếp trong tập hợp, hàm hasNext trả về giá trị True nếu vẫn còn phần tử trong tập hợp và trả về false trong trường hợp ngược lại, hàm remove cho phép bạn gỡ bỏ một phần tử trong tập hợp.

Đó là cách Iterator làm việc – Nó cung cấp một giao diện đơn giản, nhất quán để làm việc với các tập hợp khác nhau. Giả sử rằng khách hàng phải làm việc với một tập hợp phức tạp và rắc rối (như hình sau) và không biết cách thức làm việc với nó như thế nào.

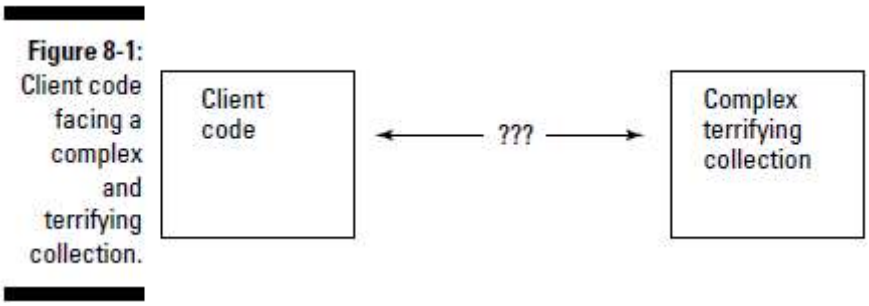

Figure 8-1: Client code facing a complex and terrifying collection.

Khách hàng có thể sử dụng iterator để làm cầu nối với tập hợp, và khách hàng có thể sử dụng các phương thức cơ bản của Iterator để giao tiếp với tập hợp. Như hình sau:

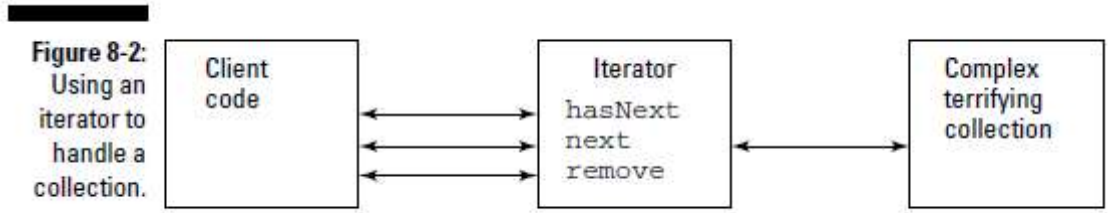

Figure 8-2: Using an iterator to handle a collection.

Công việc đầu tiên khi lưu trữ dữ liệu các phó giám đốc cũng giống cách trên. Bạn quyết định, đầu tiên là tạo một lớp lưu trữ thông tin cho từng phó giám đốc, với tên lớp VP (Vice President – phó giám đốc, phó chủ tịch..)

Bạn phải tạo bốn thành phần quan trọng trong lớp này, bao gồm:

– Một hàm khởi dựng cho phép truyền giá trị tên của vị phó này
– Tên khu vựa làm việc của vị phó
– Hàm getName trả về tên của người này
– Hàm print cho phép in ra thông tin của vị phó này, bao gồm tên và khu vực làm việc

```
public class VP
{
  private String name;
  private String division;

  public VP(String n, String d)
  {
    name = n;
    division = d;
  }

  public String getName()
  {
    return name;
  }

  public void print()
  {
    System.out.println("Name: " + name + " Division: " + division);
  }

}
```

Lớp này đã đóng gói thông tin một phó giám đốc. Bây giờ ta phải lưu trữ tất cả giám đốc trong một lớp.

Thu thập các phó giám đốc vào một tập hợp:

Trong ví dụ này, bạn tạo tập hợp các phó giám đốc dựa trên mảng căn bản của Java. Lí do dùng kiểu căn bản này, thay vì dùng các chức năng có sẵn trong Java như vector, danh sách mảng, bản đồ băm ... với phần tử Iterator có sẵn, đó là việc tạo Iterator từ đầu để làm việc với tập hợp thì hơi ngớ ngẩn, nhưng rất tốt để hiểu về mẫu này.

Bạn quyết định lưu thông tin các phó giám đốc trong từng khu vực, ví dụ khu vực bán hàng Sales, trong lớp tên là Division

```
public class Division
{
    .
    .
    .
}
```

Hàm khởi dựng của lớp Division sẽ lưu trữ tên của khu vực này, ví dụ Sales, và hàm getNames sẽ trả về tên đó

```
public class Division
{
  private String name;

  public Division(String n)
  {
    name = n;
  }

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

Các phó giám đốc sẽ được lưu trong một mảng, tên là vPs, và bạn có thể thêm một phó giám đốc bằng hàm add như sau:

```
public class Division
{
  private VP[] VPs = new VP[100];
  private int number = 0;
  private String name;

  public Division(String n)
  {
    name = n;
  }

  public String getName()
  {
    return name;
  }

  public void add(String n)
  {
    VP vp = new VP(n, name);
    VPs[number++] = vp;
  }
    .
    .
    .
}
```

Nói cách khác, đối tượng Division là một tập hợp, và đối tượng phó giám đốc VP là một phần tử của tập hợp này. Để thêm một iterator, tập hợp cần phải có một hàm – tên bạn có thể đặt tùy ý – ví dụ như iterator chẳng hạn (có thể tên bao gồm việc tạo createIterator và việc nhận getIterator). Hàm này sẽ chuyển mảng các phó giám đốc vào hàm khởi dựng của lớp iterator, ta gọi tên lớp này là lớp DivisionIterator. Mã như sau:

```
public class Division
{
  private VP[] VPs = new VP[100];
  private int number = 0;
  private String name;

  public Division(String n)
  {
    name = n;
  }

  public String getName()
  {
    return name;
  }

  public void add(String n)
  {
    VP vp = new VP(n, name);
    VPs[number++] = vp;
  }

  public DivisionIterator iterator()
  {
    return new DivisionIterator(VPs);
  }

}
```

Bước tiếp theo là tạo iterator, lớp DivisionIterator, cho phép bạn lặp xuyên qua tập hợp các phó giám đốc trong tập hợp.

Tạo lớp Iterator

Lớp iterator, DivisionIterator, hiện thực ba hàm trong giao diện java.util.Iterator : hàm nex, hàm hasNext, và hàm remove. Mã như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
    .
    .
    .
}
```

Hàm khởi dựng chấp nhận một mảng các phần tử VP và lưu trữ lại như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
  private VP[] VPs;

  public DivisionIterator(VP[] v)
  {
    VPs = v;
  }
    .
    .
    .
}
```

Bây giờ bạn phải hiện thực các giao diện của Iterator. Hàm next trả về phần tử kế tiếp trong mảng. Mã như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
  private VP[] VPs;
  private int location = 0;

  public DivisionIterator(VP[] v)
  {
    VPs = v;
  }

  public VP next()
  {
    return VPs[location++];
  }
    .
    .
    .
}
```

Hàm hasNext trả về true nếu có phần tử kế tiếp trong tập hợp, ngược lại trả về false. Trong trường hợp này, bạn phải kiểm trả đã ở cuối của dãy chưa? Bởi vì bạn đang làm việc với một mảng cố định, bạn cũng phải kiểm tra nếu phần tử kế tiếp là phần tử trống (null) – và bạn cũng phải kiểm tra xem mảng có phải là rỗng hay không. Hàm hasNext như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
  private VP[] VPs;
  private int location = 0;

  public DivisionIterator(VP[] v)
  {
    VPs = v;
  }

  public VP next()
  {
    return VPs[location++];
  }

  public boolean hasNext()
  {
    if(location < VPs.length && VPs[location] != null){
      return true;
    } else {
      return false;
    }
  }
    .
    .
    .
}
```

Hiện tại bạn muốn mảng phó giám đốc này chỉ độc, bạn tiếp tục hiện thực hàm remove với nội dung rỗng như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
  private VP[] VPs;
  private int location = 0;

  public DivisionIterator(VP[] v)
  {
    VPs = v;
  }

  public VP next()
  {
    return VPs[location++];
  }

  public boolean hasNext()
  {
    if(location < VPs.length && VPs[location] != null){
      return true;
    } else {
      return false;
    }
  }

  public void remove()
  {
  }
}
```

Tuyệt vời. Bạn đã có đối tượng phó giám đốc, một khu vực thể hiện như một tập hợp các phó giám đốc, và một đối tượng lặp Iterator. Việc cuối cùng là đưa tất cả chúng vào một chương trình và bắt đầu lặp qua các phó giám dốc

Lặp qua các phó giám đốc

Xem mã sau:

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

  public TestDivision()
  {
    .
    .
    .
}
```

Mã nguồn bắt đầu từ việc tạo khu vực bán hàng Sales và thêm vào một vài vị giám đốc:

```
public class TestDivision
{
  Division division;

  public static void main(String args[])
  {
    TestDivision d = new TestDivision();
  }

  public TestDivision()
  {
    division = new Division("Sales");

    division.add("Ted");
    division.add("Bob");
    division.add("Carol");
    division.add("Alice");
      .
      .
      .
  }
}
```

Sau đó ta tạo một iterator bằng cách gọi hàm iterator và sử dụng các hàm hasNext, next để duyệt qua từng phó giám đốc trong tập hợp và hiển thị thông tin từng người một.

```
public class TestDivision
{
  Division division;
  DivisionIterator iterator;

  public static void main(String args[])
  {
    TestDivision d = new TestDivision();
  }

  public TestDivision()
  {
    division = new Division("Sales");

    division.add("Ted");
    division.add("Bob");
    division.add("Carol");
    division.add("Alice");
    iterator = division.iterator();

    while (iterator.hasNext()){
      VP vp = iterator.next();
      vp.print();
    }
  }
}
```

Kết quả là, chương trình in ra toàn bộ thông tin các phó giám đốc:

```
Name: Ted Division: Sales
Name: Bob Division: Sales
Name: Carol Division: Sales
Name: Alice Division: Sales
```

Đặt mọi thứ vào trong tập hợp composites

Giám đốc của GianDataPool Inc, chạy ào vào văn phòng bạn với vẻ đắc thắng và nói lớn: "Tôi muốn sa thải một vài phó giám đốc!"

"Tốt," bạn nói.

"Tôi muốn làm thêm nữa. Bây giờ tôi cần in ra tất cả thông tin phó giám đốc của toàn bộ công ty – không chỉ khu vực bán hàng, mà là toàn bộ các khu vực."

"Tất cả các khu vực?" bạn hỏi.

"Vâng. Và cả các phó giám đốc hoạt động độc lập, không trực thuộc vào một khu vực nào ".

"Hmm", bạn nói ,"Đã đến lúc sử dụng một mẫu thiết kế mới".

"Đợi đã", giám đốc nói "Nhớ kỹ rằng đây là một vụ cắt giảm chi phí đó"

"Tôi sẽ sử dụng mẫu tổng hợp composites", bạn nói.

"Có tốn nhiều chi phí không?"

"Không" bạn nói. "nhưng tôi phải làm nhiều thôi"

Bạn đã hiểu rõ rắc rối. bây giờ bạn phải xử lý toàn bộ công ty, không chỉ là một phân khu. Toàn bộ công ty có nhiều khu vực với các phó giám đốc, và khu vực này có thể bao gồm cả khu vực khác – và bao gồm cả các phó giám đốc tự do nữa. Hình sau chỉ ra mô hình công ty:

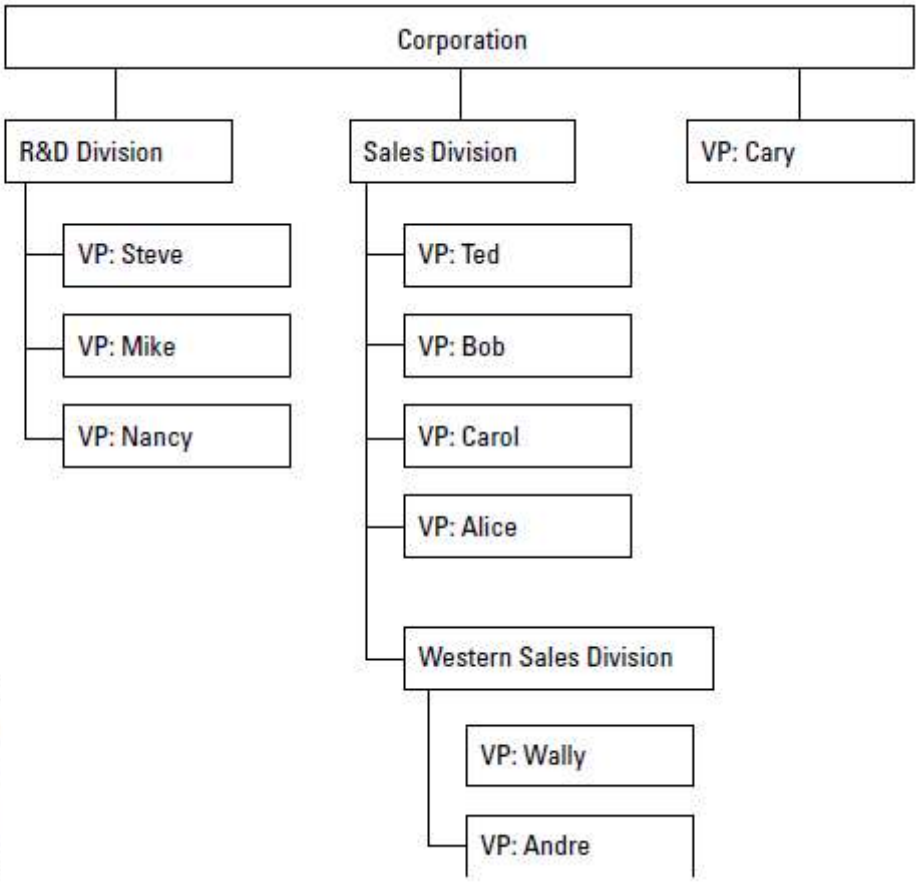

Figure 8-3: All the parts of the corporation.

Vì vậy, giờ đây bạn đang làm việc với một tổ chức phức tạp, không chỉ là một khu vực bán hàng Sales nữa. Và giám đốc điều hành muốn bạn in ra toàn bộ công ty, vì vậy bạn không chỉ cần hàm print của đối tượng VP, mà từng khu vực phải có một hàm print riêng. OK, đã đến lúc sử dụng mẫu tổng hợp Composite.

Bạn muốn có một hàm print, mà khi được gọi, nó sẽ in ra thông tin của một phó giám đốc, một phòng ban, hoặc cả tổ chức. Mẫu Composites là mẫu nói về việc tạo ra một cấu trúc hình cây nơi mà từng lá trong cây, có thể được sử dụng trong cùng một cách với nhánh của nó (nhánh là cấu trúc chứa nhiều lá, và giống các nhánh khác). Ý tưởng chính ở đây là, để làm mọi chuyện dễ dàng, bạn có thể xử lý các nút lá và tập hợp các nút lá trong một cái cây theo cùng một cách.

Ghi nhớ: Sách của GoF nói rằng, bạn sử dụng mẫu Composites để "Tạo ra các đối tượng trong một cấu trúc hình cây để biểu diễn cho một cấu trúc phân cấp. Mẫu Composites cho phép khách hàng xử lý một đối tượng riêng hoặc toàn bộ đối tượng theo cùng một cách"

Đó là những gì bạn cần – một mẫu thiết kế cho phép bạn xử lý các nút lá hoặc các nhánh của cấu trúc cây theo cách giống nhau bởi vì bạn muốn có thể in ra thông tất cả các phó giám đốc riêng lẻ, trong một khu vực, hoặc cả công ty, chỉ bằng cách gọi hàm print.

Mẫu thiết kế Composites rất phù hợp với mẫu Iterator bởi vì khi bạn gọi từng khu vực để in chính nó, nó có thể dễ dàng duyệt qua từng phó giám đốc một. Đó là đặc điểm điển hình của mẫu Composite – khi bạn yêu cầu một nhánh thực hiện một hành động gì đó , nó sẽ lặp qua tất cả các lá con và nhánh con của nó.

Ý tưởng đằng sau của mẫu Composite là việc xử lý các nút lá và nhánh trong một cấu trúc hình cây sẽ giống nhau. Điều này giúp cho việc xử lý các cấu trúc phức tạp theo dạng hình cây sẽ dễ dàng hơn bởi vì bạn không cần phải thiết lập các hàm khác nhau cho từng phần của cấu trúc.

Để thực hiện mẫu Composite, sách của GoF khuyên rằng bạn nên sử một lớp trừu tượng như là một lớp cơ sở cho cả nút lá và các nhánh trong cấu trúc cây. Việc làm này giúp cho các nút lá và các nhánh sẽ có chung một tập hợp các hàm, đó là tất cả những gì mẫu Composite muốn nói tới. Sách của GoF đề nghị bạn sử dụng một lớp trừu tượng, tuy nhiên bạn cũng có thể sử dụng một giao diện interface để làm việc này trong Java.

Tất cả bắt đầu với một lớp trừu tượng

Tôi sẽ theo chỉ dẫn của sách GoF và tạo một lớp trừu tượng cho cả phó giám đốc cũng như khu vực, lớp này tên Corporate. Bên dưới là mã nguồn của lớp này. Chú ý nó cũng có hàm add, và hàm iterator để trả về một iterator, và một hàm print:

```
import java.util.*;

public abstract class Corporate
{
  public String getName()
  {
    return "";
  }

  public void add(Corporate c)
  {
  }
  public Iterator iterator()
  {
    return null;
  }

  public void print()
  {
  }
}
```

Đây là lớp dùng để kế thừa cho cả các nút lá phó giám đốc và các nhánh cây khu vực.

Tạo nút lá phó giám đốc

Lớp VP bạn tạo trước đây phải chỉnh sửa một chút, để bạn có thể thống nhất cách làm việc với cả phó giám đốc và khu vực trong cùng một cây tổ chức, theo cách mẫu Composite đã nói. Đặc biệt, bạn phải kế thừa lớp VP từ lớp trừu tượng Corporate mà bạn đã tạo trong phần trên

```
import java.util.*;

public class VP extends Corporate
{
     .
     .
     .
}
```

Lớp VP trước đây chỉ chứa tên và khu vực làm việc của phó giám đốc và hàm print để in ra thông tin này. Nhưng để khách hàng có thể xử lý lớp VP cùng cách với các khu vực division, bạn cần thêm một hàm tạo iterator cho nó. Bởi vì một phó giám đốc không chứa bất cứ phó giám đốc nào, nên iterator được tạo ra chỉ tạo trả về một đối tượng phó giám đốc duy nhất khi bạn gọi hàm next và hàm hasNext luôn trả về giá trị sai false. Mã như sau:

```
import java.util.*;

public class VP extends Corporate
{
  private String name;
  private String division;

  public VP(String n, String d)
  {
    name = n;
    division = d;
  }

  public String getName()
  {
    return name;
  }

  public void print()
  {
    System.out.println("Name: " + name + " Division: " + division);
  }

  public Iterator iterator()
  {
    return new VPIterator(this);
  }

}
```

Lớp VPIterator sẽ như thế nào? Rất dễ dàng, bạn chỉ cần hiện thực giao diện Iterator của Java, đưa vào đối tượng VP thông qua hàm khởi dựng, tạo hàm next trả về đối tượng đó và hàm hasNext trả về giá trị false, như mã sau:

```
import java.util.Iterator;

public class VPIterator implements Iterator
{
  private VP vp;

  public VPIterator(VP v)
  {
    vp = v;
  }

  public VP next()
  {
    return vp;
  }

  public boolean hasNext()
  {
    return false;
  }

  public void remove()
  {
  }
}
```

Bây giờ khách hàng có thể xử lý nút lá phó giám đốc giống như một nhánh cây khu vực. Thực tế là iterator nút lá phó giám đốc chỉ trả về duy nhất một phó giám đốc, nhưng bây giờ bạn đã có một iterator cho từng nút lá, bạn không phải chỉnh sửa mã nguồn để có thể vừa làm việc với nút lá vừa làm việc với các phân khu.

Tạo một nhánh cây các khu vực

Từng nhánh cây trong cấu trúc cây công ty là một khu vực trong công ty, mà có thể bao gồm nhiều phó giám đốc hoặc khu vực con. Để xử lý khu vực, bạn quyết định chỉnh sửa lớp Division (đã tạo trước đây) theo cách mở rộng từ lớp Corporate, như cách đã làm với lớp VP, mã nguồn như sau:

```
import java.util.*;

public class Division extends Corporate
{
    .
    .
    .
}
```

Phần còn lại của lớp Division sẽ giống như trước, ngoài trừ bạn phải chuyển đổi một chút để phù với đối tượng Corporate. Trước đây, lớp Division lưu trữ một mảng VPs các phó giám đốc bởi vì bạn chỉ làm việc với một khu vực của công ty. Bây giờ bạn phải làm việc với cả công ty, một khu vực có thể chứa một khu vực con như là những mảng các phó giám đốc VPs. Từ khi cả hai lớp divison và mảng VPs kế thừa từ lớp Corparate, bạn có thể dễ dàng hóan chuyển để lưu trữ và làm việc với đối tượng Corparate trong lớp Division – chú ý rằng hàm print sẽ duyệt qua tất cả các đối tượng trong một khu vực, cho dù chúng là mảng VPs hay khu vực.

```
import java.util.*;

public class Division extends Corporate
{
  private Corporate[] corporate = new Corporate[100];
  private int number = 0;
  private String name;

  public Division(String n)
  {
    name = n;
  }
  public String getName()
  {
    return name;
  }

  public void add(Corporate c)
  {
    corporate[number++] = c;
  }

  public Iterator iterator()
  {
    return new DivisionIterator(corporate);
  }

  public void print()
  {
    Iterator iterator = iterator();

    while (iterator.hasNext()){
      Corporate c = (Corporate) iterator.next();
      c.print();
    }
  }
}
```

Bằng cách chuyển đổi từ việc xử lý lớp VP bên trong một khu vực division sang việc xử lý một lớp Corporate, giờ đây bạn có thể lưu trữ mảng VPs và khác khu vực khác – và vì vậy bạn đã hiện thực được mẫu Composite, cho phép bạn có thể xử lý các nút lá hay nhánh cây theo cùng một cách.

Iterator của lớp division, được hiện thực từ lớp DivisionIterator, mã như sau:

```
import java.util.Iterator;

public class DivisionIterator implements Iterator
{
  private Corporate[] corporate;
  private int location = 0;

  public DivisionIterator(Corporate[] c)
  {
    corporate = c;
  }
  public Corporate next()
  {
    return corporate[location++];
  }

  public boolean hasNext()
  {
    if(location < corporate.length && corporate[location] != null){
      return true;
    } else {
      return false;
    }
  }

  public void remove()
  {
  }
}
```

Bạn ngả người ra trong ghế với nụ cười hài lòng trên môi, thầm cảm ơn mẫu thiết kế Composite. Để làm một sự chuyển đổi từ ví dụ một khu vực trong phần đầu của chương này, sang một một cấu trúc hình cây cho toàn bộ công ty, tất cả những gì bạn phải làm là bảo đảm rằng tất cả các đối tượng trong cùng một cây phải dựa trên cùng một lớp, và hiện thực cùng một tập hợp các hàm, cho phép chúng được sử dụng theo cùng một cách.

Xây dựng công ty của bạn

Bạn đã có phó giám đốc; bạn đã có các khu vực. Bây giờ là lúc bạn xây dựng một công ty để chứa chúng. Để giữa cho mọi việc đơn giản, bạn có thể sử dụng một ArrayList để lưu trữ các khu vực divisions và các phó giám đốc trong công ty. Tất cả các đối tượng trong công ty đều là đối tượng Corporate, vì vậy ArrayList sẽ lưu trữ các đối tượng Corporate.

```
import java.util.*;

public class Corporation extends Corporate
{
  private ArrayList<Corporate> corporate = new ArrayList<Corporate>();

  public Corporation()
  {
  }
    .
    .
    .
}
```

Khi bạn muốn thêm một đối tượng Corporate vào cây, chỉ cần sử dụng hàm add của corporate, hàm sẽ thêm một đối tượng mới vào ArrayList.

```
import java.util.*;

public class Corporation extends Corporate
{
  private ArrayList<Corporate> corporate = new ArrayList<Corporate>();

  public Corporation()
  {
  }

  public void add(Corporate c)
  {
    corporate.add(c);
  }
    .
    .
    .
}
```

Muốn in ra thông tin tất cả các đối tượng trong công ty? Chỉ cần gọi hàm print của đối tượng Corporate, khi đó các iterator trong ArrayList sẽ in ra thông tin trong các khu vực và các phó giám đốc của toàn công ty – chú ý rằng khi bạn gọi hàm print của khu vực division, nó sẽ duyệt qua toàn bộ các đối tượng bên trong, và gọi từng hàm print của chúng. Vì vậy khi gọi hàm print từ cấp cao nhất của Corporate, nó sẽ in ra toàn bộ thông tin của công ty.

```
import java.util.*;

public class Corporation extends Corporate
{
  private ArrayList<Corporate> corporate = new ArrayList<Corporate>();

  public Corporation()
  {
  }

  public void add(Corporate c)
  {
    corporate.add(c);
  }
  public void print()
  {
    Iterator iterator = corporate.iterator();

    while (iterator.hasNext()){
      Corporate c = (Corporate) iterator.next();
      c.print();
    }
  }
}
```

OK. Đã đến lúc cho chương trình chạy thử. Đầu tiên bạn tạo một đối tượng Corporation.

```
import java.util.*;

public class TestCorporation
{
  Corporation corporation;

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

  public TestCorporation()
  {
    corporation = new Corporation();
    .
    .
    .
}
```

Sau đó bạn tạo khu vực R&D và tạo ra một vài phó giám đốc

```
import java.util.*;

public class TestCorporation
{
    .
    .
    .
  public TestCorporation()
  {
    corporation = new Corporation();

    Division rnd = new Division("R&D");
    rnd.add(new VP("Steve", "R&D"));
    rnd.add(new VP("Mike", "R&D"));
    rnd.add(new VP("Nancy", "R&D"));
        .
        .
        .
  }
}
```

Tiếp theo, bạn tạo khu vực Sales. Bạn sử dụng hàm add để thêm không chỉ các phó giám đốc mà còn có thể thêm cả các khu vực con, ví dụ khu vực Western Sales, với một số phó giám đốc

```
import java.util.*;

public class TestCorporation
{
    .
    .
    .
  public TestCorporation()
  {
    corporation = new Corporation();

    Division rnd = new Division("R&D");
    rnd.add(new VP("Steve", "R&D"));
    rnd.add(new VP("Mike", "R&D"));
    rnd.add(new VP("Nancy", "R&D"));

    Division sales = new Division("Sales");

    sales.add(new VP("Ted", "Sales"));
    sales.add(new VP("Bob", "Sales"));
    sales.add(new VP("Carol", "Sales"));
    sales.add(new VP("Alice", "Sales"));

    Division western = new Division("Western Sales");
    western.add(new VP("Wally", "Western Sales"));
    western.add(new VP("Andre", "Western Sales"));

    sales.add(western);
        .
        .
        .
  }
}
```

Và bạn có thể thêm các phó giám đốc vào công ty một cách trực tiếp, cũng giống như cách thêm vào một khu vực, bởi vì bạn có thể xử lý các nút lá và các nhánh con theo cùng một cách. Sau khi tạo một phó giám đốc, bạn có thêm phó giám đốc – và khu vực bạn đã tạo trước – vào công ty và in tất cả thông tin chúng ra với một hàm duy nhất là hàm print của đối tượng corporation, hàm này sẽ gọi hàm print của từng phần tử bên trong nó.

```
import java.util.*;

public class TestCorporation
{
    .
    .
    .
  public TestCorporation()
  {
    corporation = new Corporation();

    Division rnd = new Division("R&D");
    rnd.add(new VP("Steve", "R&D"));
    rnd.add(new VP("Mike", "R&D"));
    rnd.add(new VP("Nancy", "R&D"));

    Division sales = new Division("Sales");

    sales.add(new VP("Ted", "Sales"));
    sales.add(new VP("Bob", "Sales"));
    sales.add(new VP("Carol", "Sales"));
    sales.add(new VP("Alice", "Sales"));

    Division western = new Division("Western Sales");
    western.add(new VP("Wally", "Western Sales"));
    western.add(new VP("Andre", "Western Sales"));

    sales.add(western);

    VP vp = new VP("Cary", "At Large");

    corporation.add(rnd);
    corporation.add(sales);
    corporation.add(vp);

    corporation.print();
  }
}
```

Chạy chương trình, và bạn nhận được kết quả.

```
Name: Steve Division: R&D
Name: Mike Division: R&D
Name: Nancy Division: R&D
Name: Ted Division: Sales
Name: Bob Division: Sales
Name: Carol Division: Sales
Name: Alice Division: Sales
Name: Wally Division: Western Sales
Name: Andre Division: Western Sales
Name: Cary Division: At Large
```

Bạn đưa danh sách thắng lợi này tới cho vị Giám đốc điều hành. “Đã tới lúc cắt tỉa bớt cái cây này”, bạn nói.

“Hả?” Giám đốc hỏi

“Loại bỏ những cành cây đã chết”, bạn nói. Giám đốc cười hạnh phúc.

Download source here

Design Patterns For Dummies
Design Patterns
< DP4Dummies – Chương 7: Template, Builder
> DP4Dummies – Chương 9: State, Proxy