DP4Dummies – Chương 9: State, Proxy

CHƯƠNG 9: KIỂM SOÁT ĐỐI TƯỢNG CỦA BẠN VỚI MẪU TRẠNG THÁI STATE VÀ MẪU ĐẠI DIỆN PROXY

Trong chương này:

- Sử dụng mẫu State
- Cho phép trạng thái xác định kết quả
- Hiểu về mẫu Proxy
- Sử dụng proxy để đại diện cho đối tượng của bạn
- Kết nối với các proxy thông qua internet

Giám đốc điều hành của công ty Apartments-N-Stuff Inc đã đăng kí dịch vụ tư vấn của bạn và nói "Chúng tôi đang hoạt động cho thuê căn hộ phức hợp trải rộng khắp đất nước". Vấn đề lớn nhất của chúng tôi là quản lý tài sản, chi phí cho việc này rất lớn. Vì thế chúng tôi đang chuyển đổi tất cả căn hộ phức hợp của chúng tôi sang việc sử dụng các robot phân phối tự động, nó sẽ chấp nhận các đơn thuê nhà và phân phối chìa khóa cho khách hàng. Chúng tôi muốn những người thuê nhà mới sẽ có thể nộp đơn cho hệ thống tự động này, và khi hệ thống chấp nhận, họ sẽ nhận được chìa khóa nhà từ hệ thống.

"Nghe thật tuyệt", bạn nói

"Ý tưởng chính là," vị giám đốc nói tiếp "Bình thường, hệ thống tự động này được đặt xung quanh các tòa nhà và chờ đợi các khách hàng tiềm năng. Khi một người thuê nhà nộp đơn, máy tự động này sẽ kiểm tra đơn. Nếu đơn được chấp thuận, máy tự động sẽ phân phối chìa khóa cho người thuê nhà, ngược lại hệ thống sẽ thông báo từ chối người thuê nhà và trở về trạng thái chờ đợi. Và nếu một hệ thống tự động cho thuê một căn hộ, nó cũng phải kiểm tra để biết chắc rằng còn trống căn hộ nào trong dãy nhà phức hợp không, và sẽ không cho thuê nữa nếu hết căn hộ.

Vị giám đốc điều hành nhìn bạn đang vẽ nguệch ngoạc và cuối cùng hỏi: "Anh đang làm gì đó?"

"Tôi đang vẽ một biểu đồ trạng thái," bạn nói.

“Tôi có thể xem nó không?”, Vị giám đốc hỏi

“Không,” bạn nói “Ông cần biết thêm nhiều kiến thức để hiểu biểu đồ này”

“Oh”, Vị giám đốc nói

Chương này nói về hai mẫu thiết kế: mẫu trạng thái State, nơi mà một đối tượng lưu trữ thông tin về các trạng thái nội tại của nó, và có thể thay đổi hành vi cho thích hợp với trạng thái đó, và mẫu đại diện Proxy, nơi mà một đối tượng có thể hành động như một sự thay thế cho đối tượng khác. Bạn sẽ thấy chi tiết cách hai mẫu làm việc trong chương này.

Ghi nhận tình trạng của bạn với mẫu State

Bạn hiểu những gì vị giám đốc đã nói. Hệ thống tự động cho thuê sẽ có 4 trạng thái sau:

> - Waiting for a new tenant
> - Receiving an application
> - Renting an apartment
> - Fully rented

Đó là: chờ một người thuê nhà mới, nhận một đơn thuê nhà, cho thuê một căn hộ, đã hết căn hộ cho thuê.

Hình sau cho bạn thấy biểu đồ trạng thái, nơi mà từng hộp chủ nhật đại diện cho một trạng thái, và bạn có thể kết nối các trạng thái lại với nhau để biết cách làm việc của một máy tự động cho thuê nhà

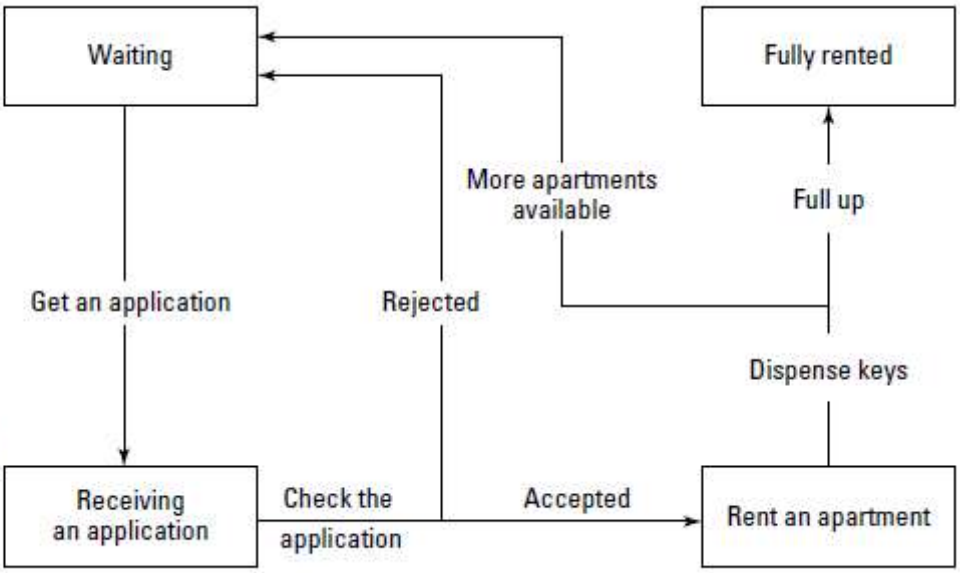

Figure 9-1: The rental automat's states.

Bạn tự nghĩ trong lòng, đây đúng là chỗ để áp dụng mẫu trạng thái State.

Thỉnh thoảng, khi bạn tham gia vào một dự án lớn, mã nguồn thường bắt đầu rất tối tăm. Có quá nhiều điều kiện có thể xảy ra mà bạn phải nắm bắt và rất khó để biết đâu là các ranh giới cũng như làm sao để chia nhỏ mã nguồn ra.

Gợi ý: Khi bạn gặp phải một ứng dụng lớn và mã nguồn không kiểm soát nổi, nó thương giúp cho bạn bắt đầu suy nghĩ tới khái niệm về các trạng thái khác nhau. Đây là công cụ giúp bạn chia nhỏ mã nguồn ra thành từng đơn vị nhỏ hơn, lý tưởng nhất là từng trạng thái này phải độc lập với nhau, và tự động chia nhỏ mã nguồn bạn ra thành từng phần rời rạc.

Ghi nhớ: Sách của Gang o Four (GoF), nói rằng mẫu thiết kế State sẽ "Cho phép một đối tượng thay đổi hành vi khi trạng thái của chính nó thay đổi. Đối tượng sẽ xuất hiện để thay đổi lớp của nó."

Nói cách khác, mã nguồn của bạn theo dõi các trạng thái nội tại, ví dụ như trạng thái chờ một người thuê nhà mới "Waiting for a new tenant". Phần khác của mã nguồn sẽ kiểm tra trạng thái hiện tại là gì và sẽ phản ứng phù hợp. Ví dụ, nếu trạng thái của hệ thống là hết căn hộ cho thuê Fully rented, và một khách hàng tiềm năng mới tới thuê nhà, khi đưa vào đơn thuê nhà, lập tức hệ thống sẽ từ chối đơn đó.

Khi bạn sử dụng mẫu thiết kế State, một phần mã nguồn có thể kiểm tra trạng thái hiện tại là gì. Điều này giúp bạn làm việc với mã nguồn lớn một cách rõ ràng và tập trung hơn bởi vì bạn phải kiểm soát hoạt động của từng đoạn mã rời rạc, chỉ bằng các cách thay đổi trạng thái hiện tại.

Nói chung, mẫu thiết kế State rất hữu ích khi bạn có nhiều mã nguồn mà ngày càng phức tạp và rối rắm. Nếu bạn có thể chia nhỏ công việc bằng cách đưa chúng vào các trạng thái độc lập với nhau, bạn đã làm công việc của mình dễ dàng hơn rất nhiều.

Sử dụng hàm để lưu trữ trạng thái

Ví dụ đầu tiên, bạn cố gắng tạo một máy tự động đơn giản, sử dụng trạng thái. Bạn quyết định rằng bạn có thể viết ứng dụng tập trung vào một tập hợp các hàm, các hàm này có thể hoạt động dựa trên trạng thái hiện tại của hệ thống. Bạn bắt đầu bằng cách tạo từng hằng số trạng thái cho bốn trạng thái của hệ thống. Mã nguồn như sau:

```
import java.util.*;
import java.lang.Math;

public class RentalMethods
{
  final static int FULLY_RENTED = 0;
  final static int WAITING = 1;
  final static int GOT_APPLICATION = 2;
  final static int APARTMENT_RENTED = 3;
  int state = WAITING;
        .
        .
        .
```

Bây giờ bất cứ mã nguồn nào trong ứng dụng cũng có thể kiểm tra tình trạng hiện tại và đáp ứng thích hợp. Ví dụ mã nguồn sau chỉ ra cách thức hàm getApplication làm việc, hàm được gọi khi hệ thống nhận một đơn thuê nhà của khách hàng – chú ý rằng việc hệ thống xử lý sẽ phụ thuộc vào trạng thái nội tại:

```
import java.util.*;
import java.lang.Math;

public class RentalMethods
{
  final static int FULLY_RENTED = 0;
  final static int WAITING = 1;
  final static int GOT_APPLICATION = 2;
  final static int APARTMENT_RENTED = 3;
  Random random;
  int numberApartments;
  int state = WAITING;

  public RentalMethods(int n)
  {
    numberApartments = n;
    random = new Random(System.currentTimeMillis());
  }

  public void getApplication()
  {
    switch (state)
    {
      case FULLY_RENTED:
        System.out.println("Sorry, we're fully rented.");
        break;
      case WAITING:
        state = GOT_APPLICATION;
        System.out.println("Thanks for the application.");
        break;
      case GOT_APPLICATION:
        System.out.println("We already got your application.");
        break;
      case APARTMENT_RENTED:
        System.out.println("Hang on, we're renting you an apartment.");
        break;
    }
  }
  .
  .
  .
```

Vậy nếu hệ thống nhận được đơn thuê nhà và nó đang ở trạng thái FULLY_RENTED, hệ thống sẽ in ra thông báo xin lỗi, chúng tôi đã cho thuê hết căn hộ. “Sorry, we’re fully rented”. Nếu hệ thống nhận một đơn thuê nhà và trạng thái của nó là WAITTING, nó sẽ thông báo “Thanks for the application”. Và thay đổi hệ thống tự thay đổi trạng thái sang GOT_APPLICATION.

Tương tự vậy, nếu bạn gọi hàm checkApplication của hệ thống, việc đáp trả sẽ tùy thuộc vào tình trạng hiện tại: Ví dụ, nếu hệ thống được yêu cầu kiểm tra một đơn thuê nhà, và nó đang ở tình trạng WAITING, nó sẽ báo với khách hàng là họ cần phải nộp đơn thuê nhà trước. Nếu hệ thống đang ở tình trạng GOT_APPLICATION khi bạn gọi hàm

checkApplication, nó sẽ kiểm tra đơn thuê nhà vạ quyết định chấp nhận hay từ chối khách hàng này. Quy trình đó được mô phỏng trong ví dụ này với một số ngẫu nhiên – nếu đơn được chấp nhận, chương trình sẽ đưa hệ thống về trạng thái căn hộ đã cho thuê APARMENT_RENTED và gọi hàm tên rentApartment; nếu đơn bị từ chối, chương trình sẽ đưa hệ thống về trạng thái chờ WAITING.

```
public void checkApplication()
{
  int yesno = random.nextInt() % 10;

  switch (state)
  {
    case FULLY_RENTED:
      System.out.println("Sorry, we're fully rented.");
      break;
    case WAITING:
      System.out.println("You have to submit an application.");
      break;
    case GOT_APPLICATION:
      if (yesno > 4 && numberApartments > 0) {
        System.out.println("Congratulations, you were approved.");
        state = APARTMENT_RENTED;
        rentApartment();
      } else {
        System.out.println("Sorry, you were not approved.");
        state = WAITING;
      }
      break;
    case APARTMENT_RENTED:
      System.out.println("Hang on, we're renting you an apartment.");
      break;
  }
}
```

Hàm rentApartment cũng kiểm tra trạng thái nội tại của hệ thống – nếu trạng thái là APARTMENT_RENTED, nó sẽ giảm số lượng căn hộ xuống 1 đơn vị, và gọi hàm phân phối chìa khóa nhà, hàm dispenseKeys.

```
public void rentApartment()
{
  switch (state)
  {
    case FULLY_RENTED:
      System.out.println("Sorry, we're fully rented.");
      break;
    case WAITING:
      System.out.println("You have to submit an application.");
      break;
    case GOT_APPLICATION:
      System.out.println("You must have your application checked.");
      break;
    case APARTMENT_RENTED:
      System.out.println("Renting you an apartment....");
      numberApartments--;
      dispenseKeys();
      break;
  }
}
```

Cuối cùng, hàm dispenseKeys cũng kiểm tra trạng thái hiện tại và nếu trạng thái là APARTMENT_RENTED, hàm sẽ gởi chìa khóa nhà tới người thuê nhà và đưa hệ thống về trạng thái chờ WAITING để tiếp tục chờ người thuê nhà kế tiếp.

```
public void dispenseKeys()
{
  switch (state)
  {
    case FULLY_RENTED:
      System.out.println("Sorry, we're fully rented.");
      break;
    case WAITING:
      System.out.println("You have to submit an application.");
      break;
    case GOT_APPLICATION:
      System.out.println("You must have your application checked.");
      break;
    case APARTMENT_RENTED:
      System.out.println("Here are your keys!");
      state = WAITING;
      break;
  }
}
```

Ý tưởng chính là như vậy – bạn lưu trữ trạng thái nội tại và mỗi khi mã nguồn bên ngoài gọi các hàm trong chương trình, bạn có thể kiểm tra trạng thái hiện tại để thực hiện công việc thích hợp. Đây là chương trình thử nghiệm. Trong chương trình này bạn tạo mới một đối tượng RentalMethods, đưa vào hàm khởi tạo một số 9, để nói rằng có 9 căn hộ có thể

cho thuê. Sau đó chương trình gọi hàm getApplication và checkApplication để mô phỏng có một người thuê nhà mới.

```
public class TestRentalMethods
{
  RentalMethods rentalMethods;

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

  public TestRentalMethods()
  {
    rentalMethods = new RentalMethods(9);

    rentalMethods.getApplication();
    rentalMethods.checkApplication();

  }
}
```

Và đây là kết quả

```
Thanks for the application.
Congratulations, you were approved.
Renting you an apartment....
Here are your keys!
```

Không tệ, bạn đã tạo ra một hệ thống tự động tiếp nhận và giải quyết các yêu cầu thuê nhà cho khách hàng.

Nhưng có một rắc rối với phương pháp dựa trên các hàm. Đó là khi bạn thêm nhiều trạng thái nữa, từng hàm sẽ trở nên dài và dài hơn, và mỗi hàm phải viết lại cho tất cả trạng thái mới. Bạn sẽ làm gì? Hãy đóng gói chúng.

Sử dụng đối tượng để đóng gói trạng thái

Thay vì lưu trữ từng trạng thái trong các hằng số, có một ý tưởng tốt hơn là đưa chúng vào trong các lớp. Với cách này, bạn có thể gọi hàm dispensKeys hoặc checkApplication trên đối tượng trạng thái state bất cứ đâu trong mã nguồn. Tất cả những gì bạn phải làm là nạp đúng đối tượng trạng thái state vào một biến, và gọi các hàm khác nhau trên biến

đó. Ví dụ nếu đối tượng state hiện tại tương ứng với trạng thái chờ đợi, bạn sẽ nhận được một trả lời khác khi bạn gọi hàm gotApplication hơn là nếu đối tượng hiện tại tương ứng với trạng thái đã cho thuê hết, khi mà không còn căn hộ nào nữa.

Bạn quyết định lưu giữ trạng thái hiện tại trong một đối tượng để làm mã nguồn rõ ràng hơn. Làm sao để thiết kế một hệ thống tự động sử dụng một đối tượng trạng thái state? Hệ thống này lưu trữ trạng thái hiện tại trong một đối tượng state, đối tượng này có thể lưu trữ một trong bốn đối tượng trạng thái có thể sau : WaitingState, GotApplicationState, ApartmentRentedState và FullyRentedState. Xem hình sau:

Design Patterns For Dummies
Design Patterns
< DP4Dummies – Chương 8: Iterator, Composite
> Lần đầu làm cha