View cập nhật dữ liệu (1): phương thức tĩnh

Trong các bài trước chúng ta đã xây dựng được các lớp giao diện để xuất và nhập thông tin. Theo phân tích ở bài đầu tiên, chúng ta phải cung cấp cho người dùng khả năng cập nhật thông tin của một cuốn sách đã có sẵn.

Trong bài học này chúng ta sẽ sử dụng kỹ thuật xây dựng phương thức tĩnh (static method) trong C# để xây dựng một class view mới giúp người dùng cập nhật thông tin của sách.

NỘI DUNG CỦA BÀI [Ẩn]

Thực hành 1: xây dựng lớp BookUpdateView

Trong bài này, chúng ta sẽ xây dựng một lớp giao diện nữa giúp cập nhật thông tin.

Bước 1. Tạo file mã nguồn mới BookUpdateView.cs

Tạo lớp `BookUpdateView` trong file BookUpdateView.cs trong thư mục Views.

Viết code cho lớp `BookUpdateView` như sau

```
1.  using System;
2.
3.  namespace BookMan.ConsoleApp.Views
4.  {
5.      using Models;
6.
7.      class BookUpdateView
8.      {
9.          protected Book Model;
10.
11.         public BookUpdateView(Book model)
12.         {
13.             Model = model;
14.         }
15.
16.         public void Render()
17.         {
18.             Console.ForegroundColor = ConsoleColor.Green;
19.             Console.WriteLine("UPDATE BOOK INFORMATION");
20.             Console.ResetColor();
21.
22.             // hiển thị giá trị cũ
23.             Console.ForegroundColor = ConsoleColor.Magenta;
```

```
            Console.Write("Authors: ");
            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine(Model.Authors);
            // yêu cầu nhập giá trị mới
            Console.ForegroundColor = ConsoleColor.Magenta;
            Console.Write("New value: ");
            Console.ResetColor();
            // đọc giá trị mới
            var str = Console.ReadLine();
            /* nếu người dùng ấn enter luôn (bỏ qua nhập dữ liệu) thì lấy lại giá trị cũ
             * của trường Authors gán cho biến cục bộ authors.
             * Nếu người dùng nhập giá trị mới thì biến cục bộ authors nhận giá trị này.
             * Giá trị của biến authors về sau sẽ chuyển về controller để xử lý.
             */
            var authors = string.IsNullOrEmpty(str.Trim()) ? Model.Authors : str;

            // TẠM DỪNG .... QUÁ NHIỀU CODE LẶP
        }
    }
}
```

Ở bước này chúng ta thử nghiệm việc cập nhật giá trị trường Authors theo logic:

1. Hiển thị giá trị gốc của trường Authors, sau đó yêu cầu người dùng nhập giá trị mới cho từng thuộc tính.
2. Nếu người dùng ấn enter luôn (bỏ qua nhập dữ liệu) thì lấy lại giá trị cũ của trường Authors (trong object Model) để gán cho biến cục bộ authors.
3. Giá trị của authors về sau sẽ truyền về controller (object của BookController) để thực sự thực hiện cập nhật.

Tuy nhiên, code đang bị lặp để hiển thị giao diện nhiều màu sắc. Chúng ta tạm dừng code cho class này để thực hiện một kỹ thuật mới.

Bước 2. Bổ sung thêm phương thức Update vào lớp BookController

```
/// <summary>
/// kích hoạt chức năng cập nhật
/// </summary>
/// <param name="id"></param>
public void Update(int id)
{
    var model = new Book();
    var view = new BookUpdateView(model);
    view.Render();
}
```

Bước 3. Bổ sung thêm một “case” vào phương thức Main

```
while (true)
{
    Console.Write("Request> ");
    string request = Console.ReadLine();

    switch (request.ToLower())
    {
        case "single":
            controller.Single(1);
            break;

        case "create":
            controller.Create();
            break;

        case "update":
            controller.Update(1);
            break;

        default:
```

```
            Console.WriteLine("Unknown command");
            break;
        }
    }
```

Nhắc lại, ở bài trước để tránh lặp code viết chữ ra console với màu sắc chúng ta đã xây dựng một phương thức riêng trong lớp `BookSingleView` như sau:

```
protected void WriteLine(string message, ConsoleColor color)
{
    Console.ForegroundColor = color;
    Console.WriteLine(message);
    Console.ResetColor();
}
```

Ở bài này, chúng ta cũng viết hai phương thức tương tự trong lớp `BookCreateView`

```
private void WriteLine(string message, ConsoleColor color = ConsoleColor.White, bool res
{
    Console.ForegroundColor = color;
    Console.WriteLine(message);
    if (resetColor)
        Console.ResetColor();
}

private void Write(string message, ConsoleColor color = ConsoleColor.White, bool resetCo
{
    Console.ForegroundColor = color;
    Console.Write(message);
    if (resetColor)
        Console.ResetColor();
}
```

Ở đây chúng ta đã viết lặp lại một phương thức trong 2 class.

Nếu để ý nữa chúng ta thấy các phương thức này không hề sử dụng biến thành viên của class chứa chúng. Chỉ cần cung cấp tham số đầu vào phù hợp là sử dụng được hai phương thức này.

Đây là các phương thức phù hợp để chuyển đổi thành phương thức tĩnh trong một class riêng, sao cho tất cả các lớp view đều có thể sử dụng.

Thực hành 2: cải tiến lớp BookUpdateView sử dụng phương thức tĩnh

Bước 1. Tạo class ViewHelp

Trong thư mục Framework tạo mới file mã nguồn ViewHelp.cs cho class `ViewHelp`.

Viết code cho class này như sau (có thể cut & paste hai phương thức này từ lớp `BookCreateView` cho nhanh)

```
using System;

namespace Framework
{
    public static class ViewHelp
    {
        /// <summary>
        /// xuất thông tin ra console với màu sắc (WriteLine có màu)
```

```
        /// </summary>
        /// <param name="message">thông tin cần xuất</param>
        /// <param name="color">màu chữ</param>
        /// <param name="resetColor">trả lại màu mặc định hay không</param>
        public static void WriteLine(string message, ConsoleColor color = ConsoleColor.W
        {
            Console.ForegroundColor = color;
            Console.WriteLine(message);
            if (resetColor)
                Console.ResetColor();
        }

        /// <summary>
        /// xuất thông tin ra console với màu sắc (Write có màu)
        /// </summary>
        /// <param name="message">thông tin cần xuất</param>
        /// <param name="color">màu chữ</param>
        /// <param name="resetColor">trả lại màu mặc định hay không</param>
        public static void Write(string message, ConsoleColor color = ConsoleColor.White
        {
            Console.ForegroundColor = color;
            Console.Write(message);
            if (resetColor)
                Console.ResetColor();
        }
    }
}
```

Lưu ý từ khóa static được đặt trước kiểu trả về. ViewHelp cũng đồng thời là một static class.

Bước 2. Điều chỉnh code của class `BookUpdateView`

```
using Framework;
using System;

namespace BookMan.ConsoleApp.Views
{
    using Models;

    internal class BookUpdateView
    {
        protected Book Model;

        public BookUpdateView(Book model)
        {
            Model = model;
        }

        public void Render()
        {
            ViewHelp.WriteLine("UPDATE BOOK INFORMATION", ConsoleColor.Green); //sử dụng
            ConsoleColor labelColor = ConsoleColor.Magenta, valueColor = ConsoleColor.Wh

            // hiển thị giá trị cũ
            ViewHelp.Write("Authors: ", labelColor); //sử dụng phương thức static
            ViewHelp.WriteLine(Model.Authors, valueColor); //sử dụng phương thức static
            // yêu cầu nhập giá trị mới
            ViewHelp.Write("New value: ", labelColor); //sử dụng phương thức static
            // đọc giá trị mới
            var str = Console.ReadLine();
            /* nếu người dùng ấn enter luôn (bỏ qua nhập dữ liệu) thì lấy lại giá trị cũ
             * của trường Authors gán cho biến cục bộ authors.
             * Nếu người dùng nhập giá trị mới thì biến cục bộ authors nhận giá trị này.
             * Giá trị của biến authors về sau sẽ chuyển về controller để xử lý.
             */
            var authors = string.IsNullOrEmpty(str.Trim()) ? Model.Authors : str;

            // TẠM DỪNG .... VẪN CÒN NHIỀU CODE LẶP
        }
    }
}
```

Lưu ý cách sử dụng phương thức static.

Thực hành 3: cải tiến lớp BookSingleView và BookCreateView sử dung ViewHelp

Bước 1. Cải tiến lớp BookSingleView

```
using System;
using Framework;

namespace BookMan.ConsoleApp.Views // chú ý cách Visual Studio đặt tên namespace
{
    using Models; // chú ý cách dùng using bên trong namespace

    /// <summary>
    /// class để hiển thị một cuốn sách
    /// </summary>
    internal class BookSingleView
    {
        protected Book Model; // biến này để lưu trữ thông tin cuốn sách đang cần hiển t

        /// <summary>
        /// đây là hàm tạo, sẽ được gọi đầu tiên khi tạo object
        /// </summary>
        /// <param name="model">cuốn sách cụ thể sẽ được hiển thị</param>
        public BookSingleView(Book model)
        {
            Model = model; // chuyển dữ liệu từ tham số sang biến thành viên để sử dụng
        }

        /// <summary>
        /// thực hiện in thông tin ra màn hình console
        /// </summary>
        public void Render()
        {
            if (Model == null) // kiểm tra xem có dữ liệu không
            {
                // sử dụng phương thức tĩnh WriteLine của lớp ViewHelp
                ViewHelp.WriteLine("NO BOOK FOUND. SORRY!", ConsoleColor.Red);
                return; // kết thúc thực hiện phương thức (bỏ qua phần còn lại)
            }

            // sử dụng phương thức tĩnh WriteLine của lớp ViewHelp
            ViewHelp.WriteLine("BOOK DETAIL INFORMATION", ConsoleColor.Green);

            /* các dòng dưới đây viết ra thông tin cụ thể theo từng dòng
             * sử dụng cách tạo xâu kiểu "interpolation"
             * và dùng dấu cách để căn chỉnh tạo thẩm mỹ
             */
            Console.WriteLine($"Authors:     {Model.Authors}");
            Console.WriteLine($"Title:       {Model.Title}");
            Console.WriteLine($"Publisher:   {Model.Publisher}");
            Console.WriteLine($"Year:        {Model.Year}");
            Console.WriteLine($"Edition:     {Model.Edition}");
            Console.WriteLine($"Isbn:        {Model.Isbn}");
            Console.WriteLine($"Tags:        {Model.Tags}");
            Console.WriteLine($"Description: {Model.Description}");
            Console.WriteLine($"Rating:      {Model.Rating}");
            Console.WriteLine($"Reading:     {Model.Reading}");
            Console.WriteLine($"File:        {Model.File}");
            Console.WriteLine($"File Name:   {Model.FileName}");
        }
    }
}
```

Ở bước này chúng ta sử dụng phương thức tĩnh ViewHelp.WriteLine thay cho phương thức cục bộ WriteLine xây dựng trong bài trước. Phương thức cục bộ WriteLine có thể xóa bỏ cho gọn code vì giờ không cần dùng đến nữa.

Bước 2. Xây dựng tiếp một số phương thức static cho giao diện

Cut/paste các phương thức `InputString`, `InputInt`, `InputBool` từ lớp `BookCreateView` sang lớp `ViewHelp` và chuyển thành phương thức tĩnh public

```
using System;

namespace Framework
{
    public static class ViewHelp
    {
        /// <summary>
        /// xuất thông tin ra console với màu sắc (WriteLine có màu)
        /// </summary>
        /// <param name="message">thông tin cần xuất</param>
        /// <param name="color">màu chữ</param>
        /// <param name="resetColor">trả lại màu mặc định hay không</param>
        public static void WriteLine(string message, ConsoleColor color = ConsoleColor.W
        {
            Console.ForegroundColor = color;
            Console.WriteLine(message);
            if (resetColor)
                Console.ResetColor();
        }

        /// <summary>
        /// xuất thông tin ra console với màu sắc (Write có màu)
        /// </summary>
        /// <param name="message">thông tin cần xuất</param>
        /// <param name="color">màu chữ</param>
        /// <param name="resetColor">trả lại màu mặc định hay không</param>
        public static void Write(string message, ConsoleColor color = ConsoleColor.White
        {
            Console.ForegroundColor = color;
            Console.Write(message);
            if (resetColor)
                Console.ResetColor();
        }

        /// <summary>
        /// in ra thông báo, chờ người dùng bấm phím bất kỳ.
        /// Nếu bấm 'y' sẽ trả về true, bấm phím khác sẽ trả về false
        /// </summary>
        /// <param name="label"></param>
        /// <param name="labelColor"></param>
        /// <param name="valueColor"></param>
        /// <returns></returns>
        public static bool InputBool(string label, ConsoleColor labelColor = ConsoleColo
        {
            Write($"{label} [y/n]: ", labelColor); //phương thức tĩnh gọi phương thức tĩ
            ConsoleKeyInfo key = Console.ReadKey(); //đọc 1 ký tự vào biến key
            Console.WriteLine();
            bool @char = key.KeyChar == 'y' || key.KeyChar == 'Y' ?
                true : false; //chuyển sang kiểu bool dùng biểu thức điều kiện
            return @char; // lưu ý cách viết tên biến @char
        }

        /// <summary>
        /// in ra thông báo và tiếp nhận chuỗi ký tự người dùng nhập
        /// rồi chuyển sang số nguyên
        /// </summary>
        /// <param name="label">dòng thông báo</param>
        /// <param name="labelColor">màu chữ thông báo</param>
        /// <param name="valueColor">màu chữ người dùng nhập</param>
        /// <returns></returns>
        public static int InputInt(string label, ConsoleColor labelColor = ConsoleColor.
        {
            while (true)
            {
                var str = InputString(label, labelColor, valueColor); //phương thức tĩnh
                var result = int.TryParse(str, out int i);
                if (result == true)
                {
                    return i;
                }
            }
        }

        /// <summary>
        /// in ra thông báo và tiếp nhận chuỗi ký tự người dùng nhập
        /// </summary>
```

```
        /// <param name="label">dòng thông báo</param>
        /// <param name="labelColor">màu chữ thông báo</param>
        /// <param name="valueColor">màu chữ người dùng nhập</param>
        /// <returns></returns>
        public static string InputString(string label, ConsoleColor labelColor = Console
        {
            Write($"{label}: ", labelColor, false); //phương thức tĩnh gọi phương thức t
            Console.ForegroundColor = valueColor;
            string value = Console.ReadLine();
            Console.ResetColor();
            return value;
        }
    }
}
```

Lý do chúng ta chuyển hàng loạt phương thức xuất nhập về lớp `ViewHelp` là vì các phương thức này có thể hoạt động độc lập (không cần biết về trạng thái của object nào), không phụ thuộc nghiệp vụ của bài toán (có thể tái sử dụng trong các dự án khác), và cần thiết cho nhiều class sau này sẽ xây dựng.

Việc dồn các phương thức hỗ trợ giao diện này vào một class chung giúp giảm số lượng code đáng kể ở các lớp view.

Bước 3. Cải tiến lớp BookCreateView

```
using Framework;
using System;

namespace BookMan.ConsoleApp.Views
{
    /// <summary>
    /// class để thêm một cuốn sách mới
    /// </summary>
    internal class BookCreateView
    {
        public BookCreateView() { }

        /// <summary>
        /// yêu cầu người dùng nhập từng thông tin và lưu lại thông tin đó
        /// </summary>
        public void Render()
        {
            ViewHelp.WriteLine("CREATE A NEW BOOK", ConsoleColor.Green);

            var title = ViewHelp.InputString("Title"); //đọc vào biến title
            var authors = ViewHelp.InputString("Authors"); //đọc vào biến authors
            var publisher = ViewHelp.InputString("Publisher"); //đọc vào biến publisher
            var year = ViewHelp.InputInt("Year"); // nhập giá trị cho biến year
            var edition = ViewHelp.InputInt("Edition"); // nhập giá trị cho biến edition
            var tags = ViewHelp.InputString("Tags");
            var description = ViewHelp.InputString("Description");
            var rate = ViewHelp.InputInt("Rate");
            var reading = ViewHelp.InputBool("Reading");
            var file = ViewHelp.InputString("File");
        }
    }
}
```

Ở bước này chúng ta thay các phương thức cục bộ `InputString`, `InputBool`, `InputInt` bằng phương thức tĩnh tương ứng của lớp `ViewHelp`. Bạn để ý thấy tình trạng lặp code đã giảm đáng kể. Bản thân các class view giờ đã rất gọn gàng.

Kết luận

Trong bài học này chúng ta đã vận dụng phương thức tĩnh để tạo ra một lớp hỗ trợ giao diện. Chúng ta đã sử dụng lớp hỗ trợ này để cải tiến hai lớp view cũ và để xây dựng lớp view mới giúp cập nhật thông tin sách.

Việc sử dụng phương thức tĩnh ở đây giúp chúng ta tránh lặp code và có thể tái sử dụng qua các lớp view sau này.

Cũng lưu ý rằng, hai class BookCreateView và BookUpdateView hiện thời chưa thực hiện được trọn vẹn nhiệm vụ của mình. Bạn sẽ quay lại với hai class này trong bài học về Router.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!