Thiết kế Wizard trong ứng dụng Windows Forms.

Nguyễn Huy Dũng | vannessars@yahoo.com

Section I. Wizards Overview

Wizard.

Là một phương pháp thiết kế giao diện người dùng tương tác cho một sản phẩm phần mềm. Là tập hợp các form, theo đó, người dùng sẽ đi từng bước (step by step) để lần lượt giải quyết từng phần việc nhỏ của 1 công việc phức tạp. Trong Windows, người dùng khá hay gặp các Wizard, như Wizard để cài đặt một phần mềm mới, để cấu hình mạng LAN, để add một Hardware mới, để tạo các điểm System Restore …

Bất cứ một hướng tiếp cận nào cũng có cái hay cái dở. Tôi sẽ trình bày **một vài** điểm hay và dở đáng kể nhất theo quan điểm cá nhân của mình.

Pros.

Wizard biến công việc phức tạp, khó khăn trở nên rõ ràng, thích thú hơn đối với người dùng. Họ chỉ cần nhập những nhóm thông tin tương đối nhỏ, liên quan chặt chẽ đến nhau và nhấn Next, Next, Next. Xét trên quan điểm khoa học, việc chia nhỏ công việc khiến các yếu tố thể hiện rõ mối liên hệ logic với nhau, dễ theo dõi và thực hiện hơn. Trên quan điểm tâm lý, chính việc được thấy rõ lộ trình mà mình cần phải theo, biết mình đang ở đâu, làm người dùng cảm thấy an tâm, tin tưởng và bền chí.

Tôi thích nhìn nhận khía cạnh tâm lý của mọi vấn đề. Khi tới một địa điểm mới, con đường lúc đi luôn luôn có cảm giác xa hơn, lâu hơn con đường lúc về. Bởi lần đầu đi, ta mơ hồ về đích đến, về hành trình, vừa đi vừa băn khoăn đi thế này đúng chưa, đường này sao dài thế nhỉ, vân vân. Khi về, ta biết 1 cách chắc chắn rằng đây-là-đường-cần-phải-đi, và-sẽ-về-được, do vậy, tự tin, thoải mái và để tâm trí mà ngắm cảnh bên đường chứ không băn khoăn nữa. Cứ như vậy, bảo sao chả cảm thấy đường ngắn. Khi được thấy rõ rằng: Công việc này chúng tôi (lập trình viên) đã tính toán sẵn đường đi nước bước cho bạn rồi này, cứ lần lượt, từ từ, rồi bạn sẽ xong, người dùng chẳng có gì để e ngại nữa. Thậm chí, thích thú với việc khám phá.

Việc khó đến đâu, có một bản kế hoạch thật tốt vạch sẵn từng bước đặt trước mắt, bạn sẽ có tinh thần sẵn sàng để làm hơn nhiều.

Cons.

Bạn PHẢI đi theo con đường định sẵn. Nếu lập trình viên muốn việc A phải được làm trước B, B phải làm trước C, thì bạn không có cách nào để làm B rồi A rồi C được. Do vậy, nếu không được thiết kế tốt và hợp lý, đường đi vòng vèo, thiếu logic, quá dài, hoặc quá ngắn, đều gây bực mình cho người sử dụng. Và một điểm rõ ràng nữa là: Thiết kế 1 wizard thì hấp dẫn hơn với người dùng nhưng khó khăn hơn với lập trình viên, so với việc đưa tất cả thông tin lên một form dài dằng dặc.

The example problem.

Tôi lấy ví dụ với một bài toán thực tế tôi đang giải quyết trong Project kì 2 (source code để download ở cuối bài)

Chúng ta cần thu thập thông tin cho ngân hàng câu hỏi. Có 2 loại câu hỏi: Đúng-Sai và Nhiều lựa chọn. Mỗi kiểu câu hỏi đều có một loạt thuộc tính tương ứng.

Ta sẽ thiết kế wizard như sau: Form 1 để lựa chọn kiểu câu hỏi, Form 2, tùy theo lựa chọn của Form 1 mà hiển thị bảng nhập liệu cho các câu trả lời. Form 3 thu thập các thông tin mà kiểu câu hỏi nào cũng phải có, Form 4 hiển thị câu hỏi để người dùng review và chỉnh sửa nếu cần.

The Approachs.

Hãy ngẫm kỹ điểm lợi điểm hại. Nếu bạn đã quyết định rằng sẽ thiết kế 1 wizard, giờ là lúc bắt đầu suy nghĩ về cách triển khai. Loạt bài viết này được minh họa bằng C#, thực hiện với Visual Studio 2005.

Một phương pháp "giả wizard", tôi gọi bừa là pseudo_wizard nghe cho oách, là sử dụng TabControl. Khi đến bước nào, hiển thị TabPage tương ứng với bước đó, giấu toàn bộ các Tab còn lại đi.

Phương pháp này:

1. Dễ làm. Tất cả trong 1. Bạn vừa tiết kiệm được công sức tạo các nút nhấn, vừa quản lý các Tab rất dễ dàng, vừa quản lý các thông tin thu thập được trong từng bước một cách đơn giản, bởi tất cả thông tin đều nằm trong phạm vi của **MỘT** form.

2. Khó mở rộng. Với project 2 đang làm, như đã nói, một mặt tôi muốn tạo wizard cho việc lập câu hỏi, một mặt muốn tạo câu hỏi từ submenu. Cách thiết kế này không tiện để nhảy chen vào.

3. Quá nhiều logic trộn lẫn trong 1 form.

4. Giao diện bị hạn chế với việc sử dụng Tab Control.

5. Không có "không khí" Wizard cho lắm!

   Hơi bị nhiều điểm hạn chế, nhể!

Phương pháp thứ hai (mà tôi tập trung vào) thiết kế từng form riêng biệt cho từng bước. Khi đó, cần có 1 Controller đứng ra quản lý việc các form sẽ được hiển thị hay không, vào lúc nào, như thế nào. Và hơn nữa, các thông tin thu thập được cũng phải được đặt ở một Model độc lập, bao trùm tất cả các form trong quá trình, nếu đó là 1 wizard thu thập thông tin (đa số). Nếu là các form thực hiện các chức năng nào đó, tương đối độc lập với nhau, bạn có thể bỏ qua phần này.

Các form lại cũng rất cần có một layout giống nhau, để người dùng có thể cảm nhận được chúng nằm trong 1 mạch chung của wizard.

Do vậy, các bước để tiến hành theo phương pháp này như sau:

- Tạo 1 Class với đầy đủ các thuộc tính và phương thức cần thiết để lưu trữ thông tin trong suốt quá trình thực hiện Wizard. Chỉ khi Wizard đã hoàn thành chúng ta mới lưu data vào CSDL.

- Thiết kế WizardLines và WizardManager phục vụ cho việc control Wizard.

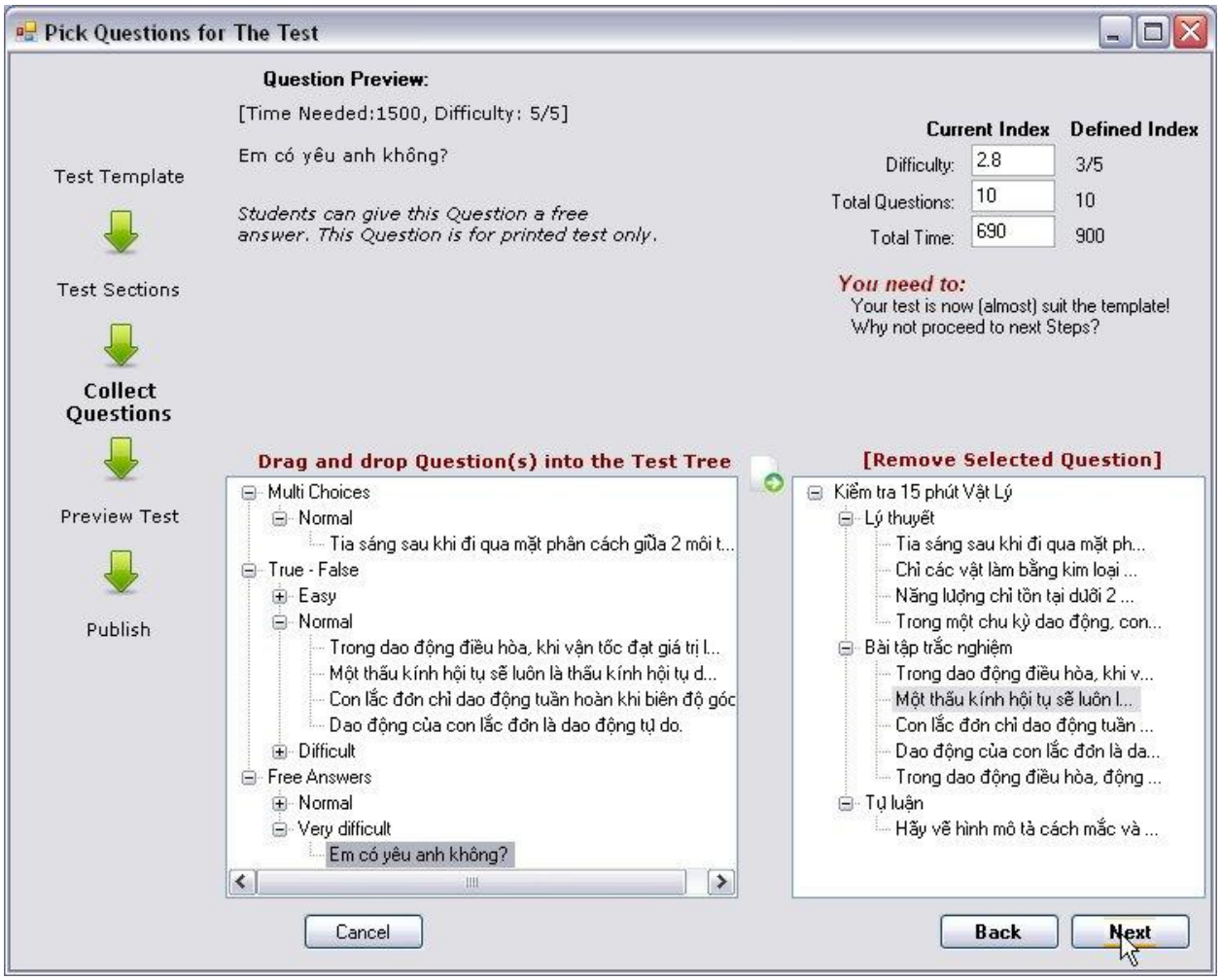
- Tạo 1 form WizardTemplate và hàng loạt form khác kế thừa Template đó, phục vụ cho việc hiển thị và thu lượm thông tin.

Chúng ta sẽ lần lượt đi qua từng phần một cách kỹ càng.

Section II. The Controller

Wizard Lines

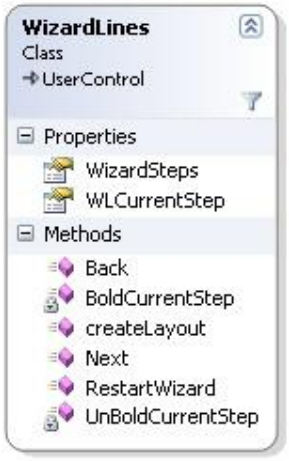
Wizard Lines là một custom control tôi viết để phục vụ nhu cầu thiết kế wizard của mình, và tôi nghĩ rất cần thiết cho bất kỳ 1 wizard nào. Bạn có thể quan sát Wizard Lines trong screenshot dưới đây, phần bên trái của form:



Các yêu cầu **cần thiết** của WL là:

- Hiển thị toàn bộ steps của wizard.
- Làm rõ hiện tại wizard đang thực hiện đến step nào. (trong ví dụ này, tôi đơn giản là sử dụng font chữ Bold.)

Và vì đằng nào WL cũng cần lưu thông tin về các steps, hiện tại đang ở steps nào… nên nó cũng sẽ hỗ trợ luôn cho Wizard Manager trong việc lựa chọn Form cần thiết để hiển thị ra.



Đây là class WizardLines:

BoldCurrentStep và UnBoldCurrentStep là 2 method cực kỳ đơn giản thay đổi hiển thị của step hiện tại. Trước khi chuyển sang step tiếp theo, WL sẽ làm step hiện tại trở về trạng thái bình thường bằng method UnBoldCurrentStep, sau khi chuyển sang step thích hợp rồi, nó sẽ sử dụng tiếp method BoldCurrentStep để tô đậm step hiện tại.

Một đặc điểm quan trọng của WL là các Steps và hình mũi tên cần phải được tạo ra trong quá trình chương trình chạy chứ không có sẵn trên giao diện designer. Hàm createLayout làm công việc đó: Nhận vào 1 Collection của các steps và render các control cần thiết.

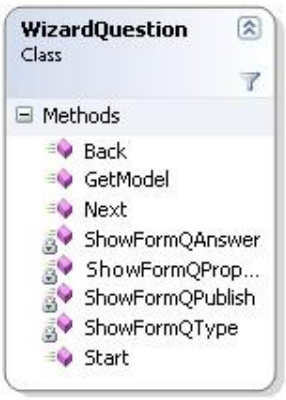
Class WizardStep chỉ có 1 constructor nhận vào string StepName.Các thuộc tính IsFinal, IsStart và StepIndex là các thuộc tính hỗ trợ và sẽ được tự động gán giá trị sau khi các WS đã được đặt vào trong 1 Collection nào đó.

Triển khai cụ thể của WizardLines các bạn có thể xem trong file demo cuối bài viết, trong thư mục Sources. Ở đây tôi giới thiệu phần code quan trọng nhất: tạo và hiển thị các control:

```
1|  public void createLayout() {
2|          pnlWizard.Dock = DockStyle.Fill;
3|          pnlWizard.FlowDirection = FlowDirection.TopDown;
4|          pnlWizard.Controls.Clear();
5|          foreach (WizardStep x in Steps) {
6|            //Create the label
7|            Label lbl = new Label();
8|            lbl.Name = ("lblSteps" + x.StepIndex);
9|            lbl.AutoSize = false;
10|           lbl.Size = new Size(this.Width, 40);
11|           lbl.Text = x.StepName;
12|           lbl.TextAlign = ContentAlignment.MiddleCenter;
13|           lbl.Font = new Font("Verdana", (float)8.25, FontStyle.Regular);
14|
15|           pnlWizard.Controls.Add(lbl);
16|
17|           if (!x.IsFinal) {
18|             //Create the Down Arrow Indicator
19|             DownArrow da = new DownArrow();
20|             Panel pnl = new Panel();
21|             pnl.Size = new Size(this.Width, 34);
22|             da.Name = ("DownArrow" + x.StepIndex);
23|             da.Location = new Point((int)((this.Width - da.Width) / 2), 1);
24|
25|             pnl.Controls.Add(da);
```

```
26|              this.Size = new Size(this.Width, Steps.Count * 85);
27|              pnlWizard.Controls.Add(pnl);
28|
29|            }
30|          }
31|          this.Controls.Add(pnlWizard);
32|      }
```

Wizard Manager



Đây là 1 class đóng vai trò controller để điều khiển việc hiển thị các form phù hợp với hoàn cảnh hiện tại. Mỗi wizard đương nhiên sẽ có một WM tương ứng cho mình. Bên cạnh đây là WM cho wizard tạo Question.

Các phương thức quan trọng với WM:

Khởi tạo các steps tương ứng với Wizard và đưa chúng vào trong 1 Wizard Lines.

```
33|      private WizardStep StepQType = new WizardStep("Question Type");
34|      private WizardStep StepQAnswer = new WizardStep("Question and Answers");
35|      private WizardStep StepQProperty = new WizardStep("Question Properties");
36|      private WizardStep StepQPublish = new WizardStep("Preview and Publish");
37|
38|     //The data model which holds all the data get from each WSs.
39|      Question TheQues;
40|
41|      //The shared WizardLines which controls the wizard flow and is displayed in every forms.
42|      public WizardLines wl = new WizardLines();
43|
44|      public WizardQuestion() {
45|          Collection<WizardStep> cws = new Collection<WizardStep>();
46|          cws.Add(StepQType);
47|          cws.Add(StepQAnswer);
48|          cws.Add(StepQProperty);
49|          cws.Add(StepQPublish);
50|          wl.WizardSteps = cws;
51|      }
52|      public Question GetModel() {
53|          return this.TheQues;
54|      }
```

Các hàm Next và Back kiểm tra làm nhiệm vụ hiển thị các form tương ứng, ở đây chỉ ví dụ hàm Next. Cần chú ý, mỗi form thuộc Wizard đều có 1 hàm GetInfo(), nhận vào tham chiếu

tới DataModel và lưu các thông tin cần thiết, trả về giá trị bool TRUE hoặc FALSE. Chỉ khi quá trình lưu thông tin đã thành công (trả về TRUE), chúng ta mới đóng form lại và hiển thị form mới. Trong triển khai thực tế, các bạn nên viết tất cả các form thuộc Wizard kế thừa từ 1 Interface nào đó với hàm GetInfo(), khi đó sẽ không cần phải ép kiểu cho các Forms như ở đây tôi đã làm.

```
1|        public bool Next(Form f) {
2|            if (wl.WLCurrentStep == this.StepQType) {
3|               if (((WQType)f).GetInfo(out this.TheQues)) {
4|                  wl.Next();
5|                  ShowFormQAnswer();
6|               }
7|               else { return false; }
8|            }
9|            else if (wl.WLCurrentStep == this.StepQAnswer) {
10|              if (((WQAnswer)f).GetInfo(TheQues)) {
11|                 wl.Next();
12|                 ShowFormQProperty();
13|              }
14|              else { return false; }
15|           }
16|           else if (wl.WLCurrentStep == this.StepQProperty) {
17|              if (((WQProperty)f).GetInfo(TheQues)) {
18|                 wl.Next();
19|                 ShowFormQPublish();
20|              }
21|              else { return false; }
22|           }
23|           else if (wl.WLCurrentStep == this.StepQPublish) {
24|              MessageBox.Show("The question will be save now [Actually, no data will be saved.
      This is a demo, don't you remember?]");
25|              return true;
26|           }
27|           else {
28|              return false;
29|           }
30|           return true;
31|       }
```

Hàm Back hoàn toàn tương tự. Ở mỗi hàm Back và Next đều có những lời gọi ShowFormABCDEF(). Đó là các hàm hiển thị Form. Chú ý 1 chút các bạn sẽ thấy khi tạo form thuộc wizard, tôi đều gửi vào WM và WL, 1 cái cần thiết cho việc "đặt form vào khuôn khổ", và 1 cái cần thiết cho việc hiển thị "đường đi" bên cạnh mỗi form:

```
1|        private void ShowFormQType() {
2|           WQType FormQType = new WQType(this, this.wl);
3|           FormQType.Show();
4|        }
5|
6|        private void ShowFormQAnswer() {
7|           WQAnswer FormQAnswer = new WQAnswer(this, this.wl);
8|           FormQAnswer.Show();
```

```
9|        }
10|
11|       private void ShowFormQProperty() {
12|          WQProperty FormQProperty = new WQProperty(this, this.wl);
13|          FormQProperty.Show();
14|       }
15|       private void ShowFormQPublish() {
16|          WQPublish FormQPublish = new WQPublish(this, this.wl);
17|          FormQPublish.Show();
18|       }
```

And, That's it! Bạn đã hoàn thành phần Controller.

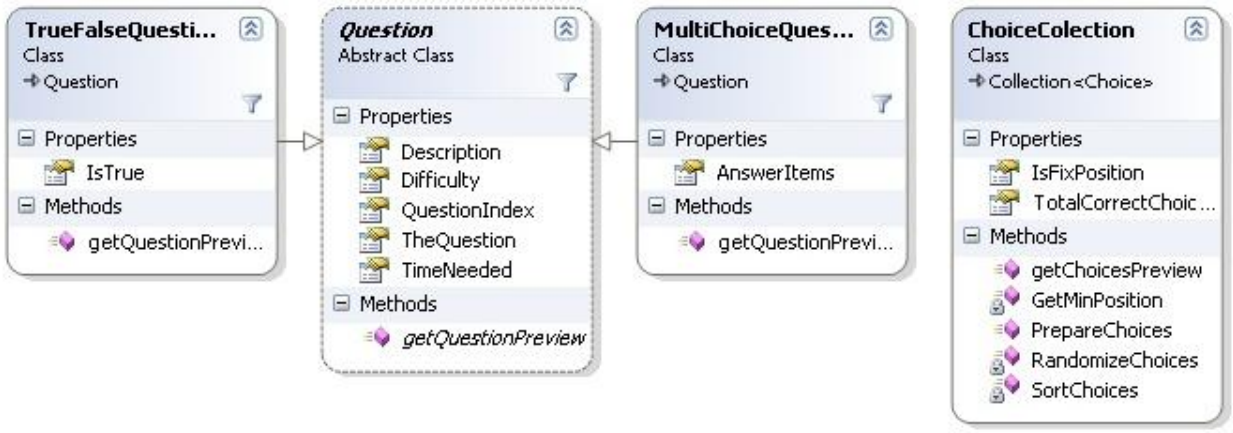
Section III: The Model

Forewords

Mỗi bài toán thực tế đều sẽ khác nhau và do vậy phần Model sẽ khác nhau. Trong demo này, tôi hoàn toàn có thể đặt những Form vô nghĩa chỉ với vài Label "Đây là Form 1" "Đây là form 2", "Đây là form 3" vào. Nhưng là 1 người hay search và đọc tutorials, tôi không thích những bài tut như vậy. Tôi muốn một bài toán thực tế, ít nhất cũng là ở 1 mức độ nào đó.

Và sẽ học được nhiều hơn, yeah?

Dưới đây là Class Diagram của phần Model.



The Question Class

Ở đây, mục đích của Wizard là thu thập thông tin về 1 câu hỏi. Và Class Questions là cái nền của tất cả các kiểu câu hỏi sẽ được hệ thống hỗ trợ. Phía dưới là 1 bản tóm gọn lại của Class Question, đủ cho demo này.

```
1|  public abstract class Question {
2|        //The QuestionIndex property hold the qID of the Question in the database.
3|        //For a new Instance of Question, its value is set to "".
4|        private string index = "";
```

```
5|        public string QuestionIndex {
6|           get { return index; }
7|           set { index = value; }
8|        }
9|
10|       private string tq;
11|       public string TheQuestion {
12|          get { return tq; }
13|          set { tq = value; }
14|       }
15|
16|       //The Descripion property hold some explaination from the Teacher for a question.
17|       //After finish one test, Student can see the Description for each Question which
18|       //help them have a clearly understand on each question, whether they did it
19|       //right or wrong, and if wrong, WHY?
20|       private string des = "";
21|       public string Description{
22|          get { return des; }
23|          set { des = value; }
24|       }
25|
26|       //The Difficulty of a Questions must be an integer in the range of 0 and 5
27|       private int diff = 3;
28|       public int Difficulty {
29|          get { return diff; }
30|          set {
31|             if (value <= 5 || value >= 0) {
32|                diff = value;
33|             }
34|             else {
35|                throw new Exception("The Diffculty value must be greater than or equal 0 and less
      than or equal 5.");
36|             }
37|          }
38|       }
39|
40|       //The TimeNeeded of a Question is an approximate time the Question take
41|       //a student at medium level to solve it,in seconds. It must be a multiple of 30.
42|       private int timeNeeded = 60;
43|       public int TimeNeeded {
44|          get { return timeNeeded; }
45|          set { timeNeeded = value; }
46|       }
47|
48|       public abstract string getQuestionPreview();
49|
50|    }
```

Bạn hãy chú ý method getQuestionPreview() là 1 abstract method. Tất cả các kiểu câu hỏi khác nhau đều cần override method này, đưa ra được Preview tương ứng cho câu hỏi. Ví dụ với 1 câu hỏi TrueFalse, nó chỉ đơn giản là thêm "The statement above is TRUE/FALSE" vào trong phần preview:

```
1|  public class TrueFalseQuestions : Question{
2|      private bool istrue;
3|      public bool IsTrue {
4|        get { return istrue; }
5|        set { istrue = value; }
6|      }
7|      public override string getQuestionPreview() {
8|        return this.TheQuestion + "\n\nThe statement above is " + (this.IsTrue ? "TRUE" :
    "FALSE");
9|      }
10|   }
```

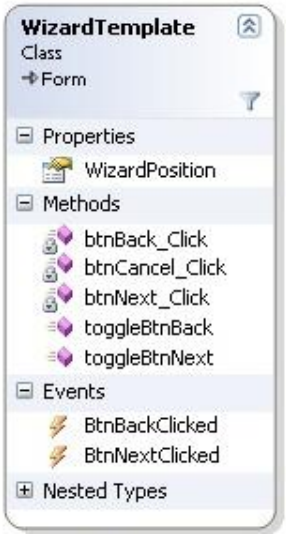
Còn với MultiChoiceQuestion thì phần việc sẽ phức tạp hơn, hẳn nhiên. Trước tiên nó cần check xem các câu trả lời sẽ được hiển thị theo thứ tự ngẫu nhiên hay thứ tự quy định. Có những lúc thứ tự câu trả lời phải được fixed. Đã bao giờ bạn thi trắc nghiệm và gặp câu trả lời "2 ý đầu tiên" nhưng bản thân câu trả lời ấy lại đang nằm ở vị trí đầu tiên? Hoặc "Tất cả các ý phía trên" nhưng phía trên chả có ý nào cả, chỉ có ở dưới thôi? Chuối, rite? Sau đó, mỗi câu trả lời sẽ được thêm [x] vào trước nếu là câu đúng và [ ] vào trước nếu là câu sai. Chi tiết hơn các bạn có thể xem thêm trong source. ('this' trong snippet dưới là ChoicesCollection, kế thừa Collection<Choice>)

```
1|      private void RandomizeChoices(ref ChoiceColection newCC) {
2|        Random r = new Random();
3|        while (this.Count > 0) {
4|          int randomIndex = (int)Math.Floor(r.NextDouble() * this.Count);
5|          newCC.Add(this[randomIndex]);
6|          this.RemoveAt(randomIndex);
7|        }
8|      }
9|
10|     private int GetMinPosition() {
11|       int MinPosition = this[0].FixPosition;
12|       foreach (Choice c in this) {
13|         if (c.FixPosition < MinPosition) {
14|           MinPosition = c.FixPosition;
15|         }
16|       }
17|       return MinPosition;
18|     }
19|
20|     public void PrepareChoices() {
21|       ChoiceColection cc = new ChoiceColection();
22|       if (this.IsFixPosition) {
23|         this.SortChoices(ref cc);
24|       }
25|       else {
26|         this.RandomizeChoices(ref cc);
27|       }
28|       this.Clear();
29|       foreach (Choice c in cc) {
30|         this.Add(c);
```

```
31|         }
32|       }
33|
34|       public string getChoicesPreview() {
35|         string res = "\nThe Choices: \n\n";
36|         this.PrepareChoices();
37|         foreach (Choice c in this) {
38|           res += ((c.FixPosition != -1) ? "   " + c.FixPosition : "    ");
39|           res += c.IsSolution ? "[x] " : "[ ] ";
40|       res += ((c.TheAnswer.Length > 50) ? (c.TheAnswer.Substring(0, 50) + "...") : c.TheAnswer);
41|           res += "\n";
42|         }
43|         return res;
44|       }
```

Section IV: The Views

Để wizard của chúng ta hiển thị được tốt nhất, "tạo không khí" tốt nhất, bạn nên có 1 template chuẩn cho tất cả các form của Wizard, điều này là khá dễ dàng với tính năng Inherited Form của VS2005. Một form template cần tối thiểu là các nút Back, Next và Cancel, tất nhiên, bạn có thể tùy chỉnh thêm tùy theo nhu cầu. Một kinh nghiệm tốt là đặt ngay cho form template hiển thị ở chính giữa màn hình để đảm bảo tất cả các form con của wizard đều sẽ hiển thị ở đúng vị trí của form trước nó.



WizardPosition là một enum với chỉ 3 giá trị: Start – OnTheWay – Finish, để căn cứ vào đó mà enable hay disable các buttons cho phù hợp. Bạn cũng hoàn toàn có thể kiểm tra WizardLines.WLCurrentStep.IsStart và WizardLines.WLCurrentStep.IsFinal rồi sau đó sử dụng các method toggleBtnBack và toggleBtnNext để làm được điều tương tự.

Mỗi form trong wizard có cấu trúc khá thống nhất: 1 constructor nhận vào WM và WL, 2 event handler cho BtnBackClicked và BtnNextClicked, 1 method GetInfo() nhận vào tham chiếu tới Model, sử dụng tham chiếu đó để cập nhật dữ liệu, và một số phương thức chức năng khác tùy theo yêu cầu.

Một yêu cầu của Wizard là khi nhấn Back, các form phải hiển thị đúng những thông tin mà trước đó người dùng đã nhập vào, do vậy trong mỗi hàm constructor bạn đều nên kiểm tra model hiện tại xem các thông tin mà form có nhiệm vụ thu thập đã tồn tại sẵn hay chưa. Nếu đã tồn tại rồi, bạn cần hiển thị nó ra để người dùng có thể chỉnh sửa.

Đây là triển khai của form WQProperty, 1 form khá điển hình trong demo này:

```
1|  public partial class WQProperty : GUI.WizardForms.WizardTemplate {
2|        WizardQuestion wm;
3|        public WQProperty(WizardQuestion wm, WizardLines wl) {
```

```
4|          InitializeComponent();
5|          this.wm = wm;
6|          Question qc = wm.GetModel();
7|          this.WizardPosition = WizardPos.OnTheWay;
8|          wl.Location = new Point(15, 60);
9|          this.Controls.Add(wl);
10|
11|          //get question info if already exist
12|          if(qc.Difficulty >0){
13|             cmbDiff.SelectedIndex = qc.Difficulty - 1;
14|          }
15|          if (qc.TimeNeeded > 0) {
16|             numTime.Value = qc.TimeNeeded / 30;
17|          }
18|
19|       }
20|
21|       private void WQProper_BtnNextClicked() {
22|          if (wm.Next(this)) {
23|             this.Close();
24|          }
25|       }
26|
27|       private void WQProper_BtnBackClicked() {
28|          if (wm.Back(this)) {
29|             this.Close();
30|          };
31|       }
32|
33|       public bool GetInfo(Question qc) {
34|          qc.Difficulty = cmbDiff.SelectedIndex + 1;
35|          qc.TimeNeeded = (int)numTime.Value * 30;
36|          return true;
37|       }
38|
39|       private void numTime_ValueChanged(object sender, EventArgs e) {
40|          decimal time = numTime.Value * (decimal)0.5;
41|          lblResult.Text = time.ToString() + " minute(s)";
42|       }
43|    }
```

Section V: Conclusion

Tổng kết lại, phương pháp của tôi là thiết kế "có hơi hướm" MVC 1 chút. Trong đó các form giống như phần view. Một vài class và UserControl đóng vai trò Controller và 1 vài class đóng vai trò Model. Khả năng tạo ra các UserControl 1 cách dễ dàng có lẽ là chức năng mà tôi ưa thích nhất của nền tảng .NET, nó thực sự rất hữu dụng và mạnh mẽ.

Bây giờ là lúc để bạn download file demo và browse source để tự tìm hiểu thêm! Chắc chắn sẽ có nhiều cách khác để tiếp cận bài toán thiết kế wizard, cũng như có nhiều đoạn code

viết chưa được tối ưu, tránh sao được khi tôi vẫn còn là 1 developer chưa có nhiều kinh nghiệm. Nhưng hy vọng bài viết này đã đem lại cho bạn 1 gợi ý nào đó.

Mọi thắc mắc, trao đổi xin hãy mail về vannessars@yahoo.com. Tôi là Nguyễn Huy Dũng. Chưa bao giờ nghe đến? Hy vọng, rồi sẽ!