

CHƯƠNG 2: GIỚI THIỆU VỀ WINDOWS FORMS

2.1. Giới thiệu

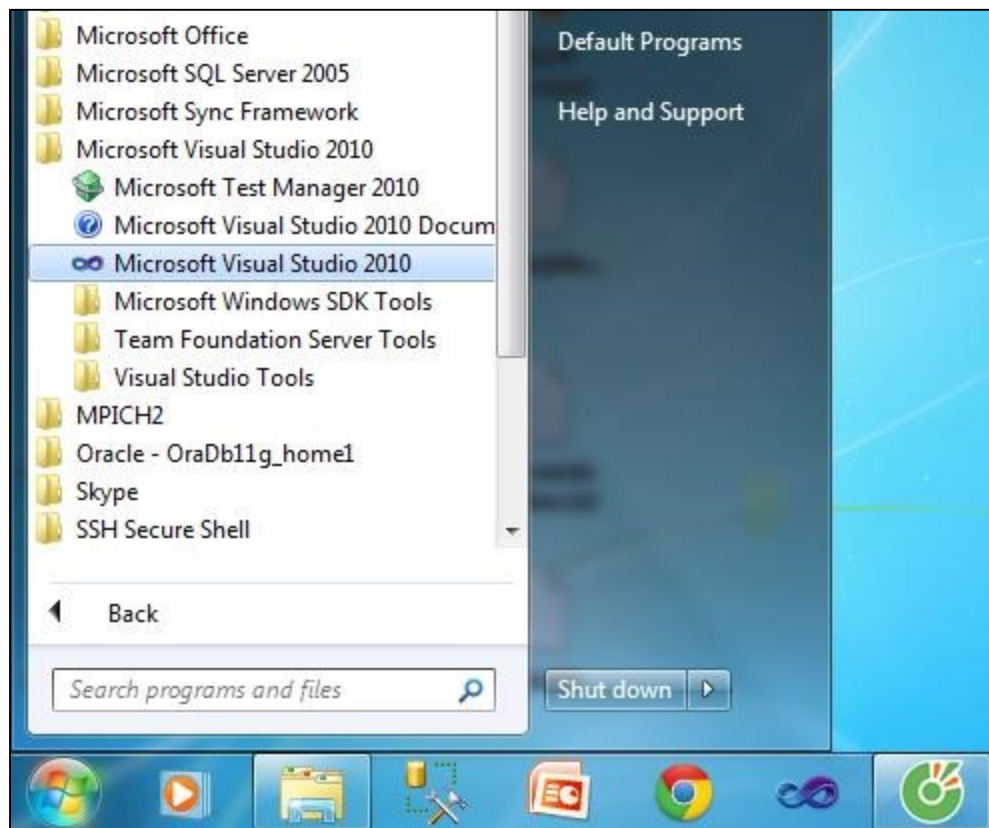
2.1.1. Ứng dụng Windows Forms

Windows Forms là ứng dụng cơ bản của Microsoft; Windows Forms cho phép lập trình viên tạo các form một cách dễ dàng, đồng thời hỗ trợ nhiều điều khiển (Controls) hoặc thành phần (Components) giúp lập trình viên có thể xây dựng, tùy chỉnh các giao diện form một cách nhanh chóng. Tạo ra các form có kích thước và hình dạng khác nhau, bố trí các điều khiển và thành phần trên form phù hợp với nhu cầu, mục đích của người sử dụng.

Ngoài ra, trên mỗi điều khiển thì Windows Forms cung cấp nhiều sự kiện giúp tương tác với thiết bị. Lập trình viên có thể sử dụng một cách đa dạng các sự kiện này trên các thiết bị như: sự kiện ấn nút bàn phím, sự kiện nhấp chuột, hoặc sự kiện ấn tổ hợp phím.

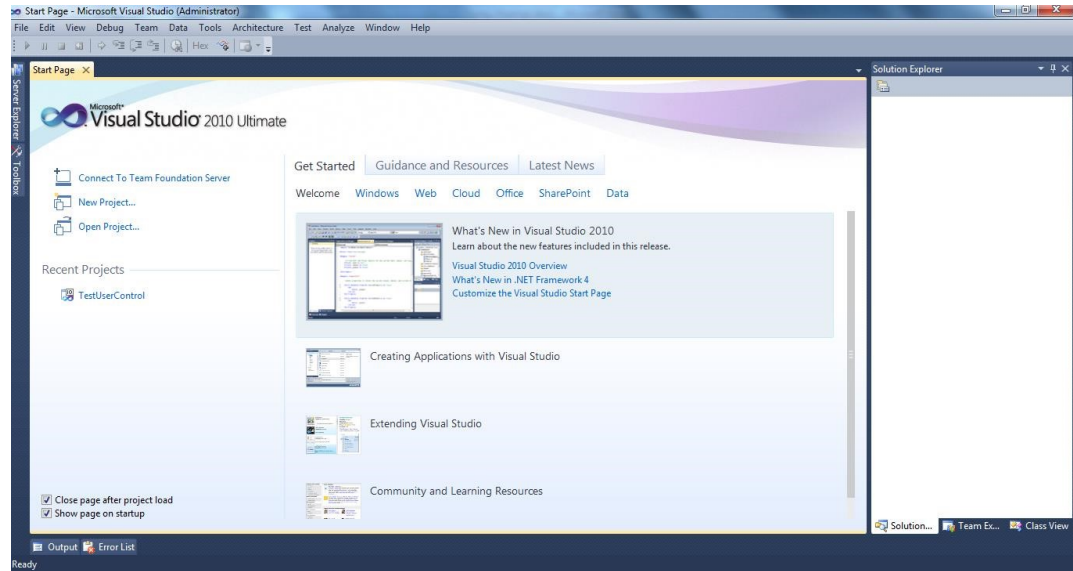
- Tạo một dự án Windows Forms: Để tạo một dự án Windows Forms với C#, cần thực hiện một số bước sau :

Bước 1: Sau khi cài xong Microsoft Visual Studio 2010, tiến hành bật ứng dụng Visual Studio 2010 lên bằng cách nhấp vào biểu tượng như hình 2.1:



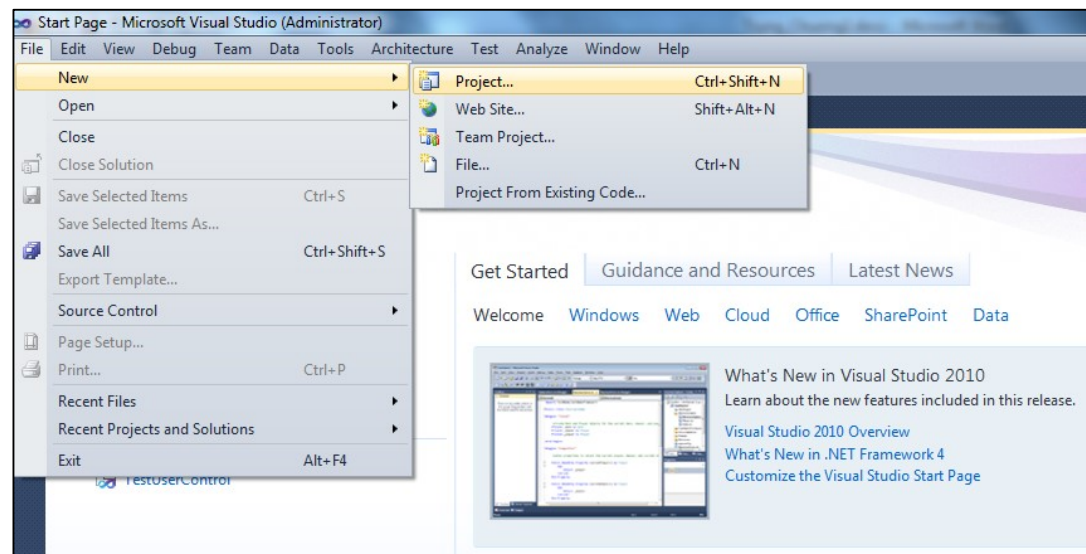
Hình 2.1: Mở ứng dụng Visual Studio 2010

Bước 2: Sau khi đã chọn biểu tượng Visual Studio 2010 giao diện như hình 2.2 sẽ được hiển thị:



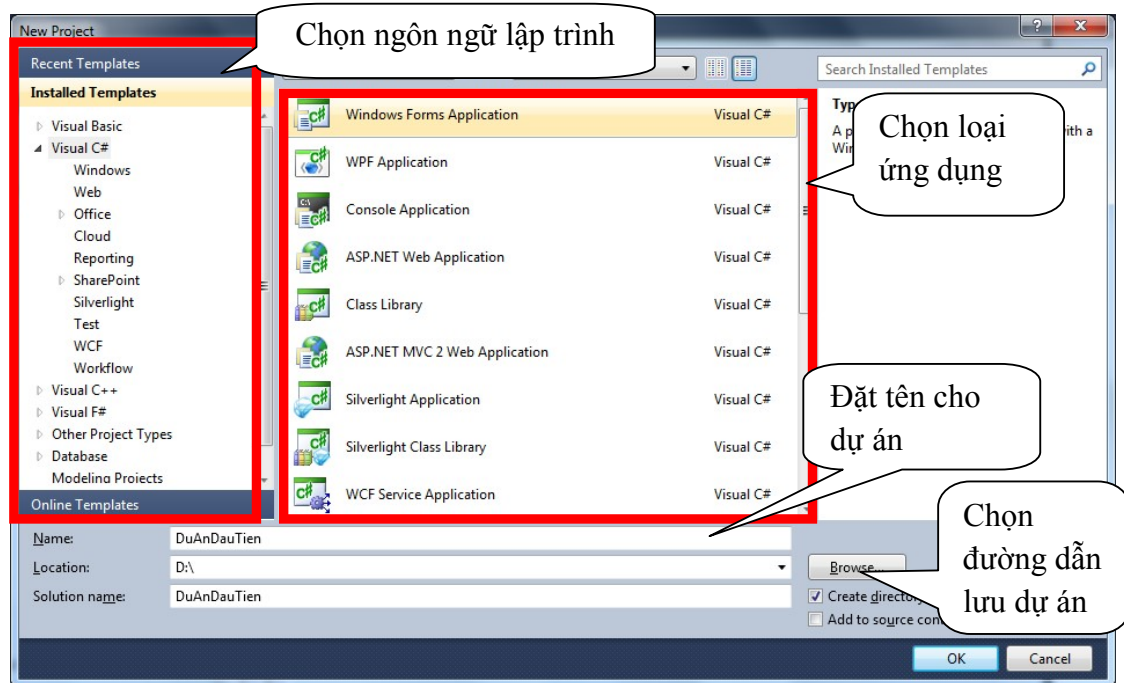
Hình 2.2: Giao diện ứng dụng Visual Studio 2010

Bước 3: Chọn Menu File > New > Project



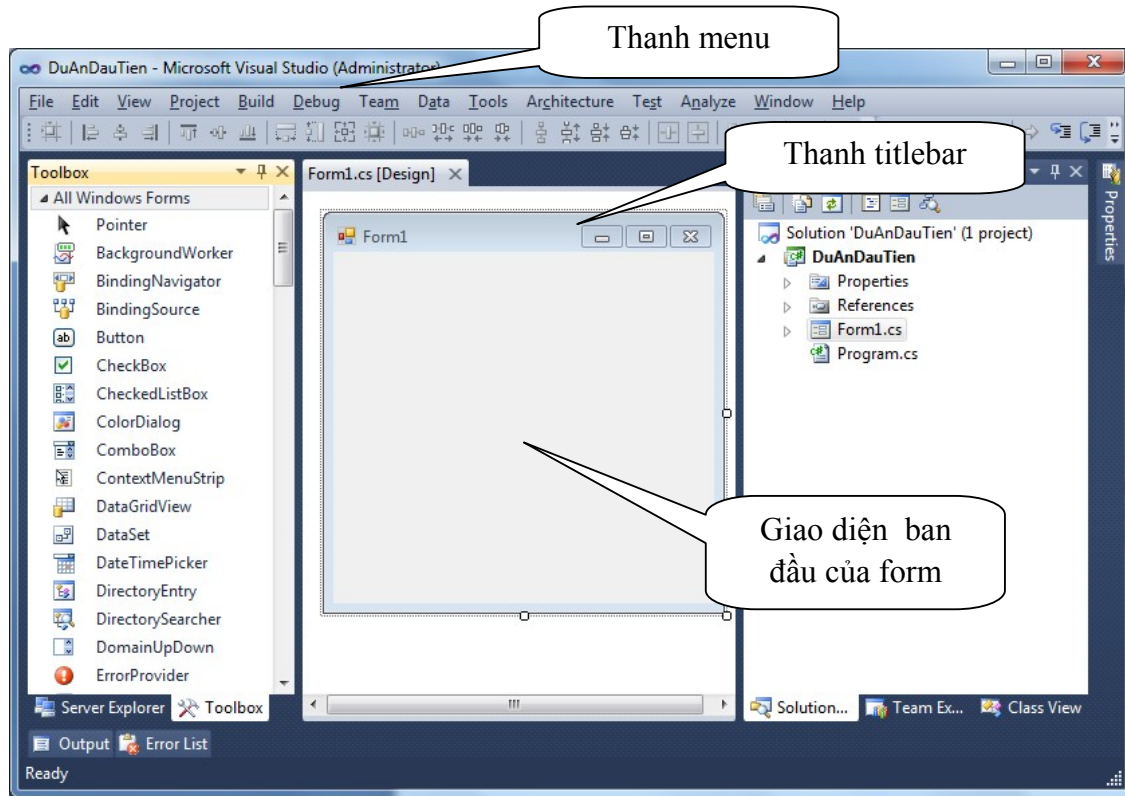
Hình 2.3: Giao diện tạo dự án mới

Bước 4: Sau khi thực hiện xong bước 3, cửa sổ New Project sẽ xuất hiện, tại cửa sổ này lập trình viên có thể chọn ngôn ngữ lập trình sử dụng và chọn loại ứng dụng muốn viết. Cụ thể ở đây chọn ngôn ngữ Visual C#, và loại ứng dụng là Windows Forms. Sau khi đã chọn xong loại ứng dụng đã viết, lập trình viên cần đặt tên cho dự án (nếu không muốn sử dụng tên mặc định của Visual Studio) và đường dẫn sẽ lưu dự án như hình 2.4:



Hình 2.4: Cửa sổ New Project

Sau khi đã tạo xong dự án, một form mới có tên Form1 mặc định sẽ được thêm vào dự án vừa tạo như hình 2.5. Lúc này lập trình viên có thể thiết kế giao diện của Form1 bằng cách thêm các thành phần cần thiết phù hợp với nhu cầu của phần mềm.

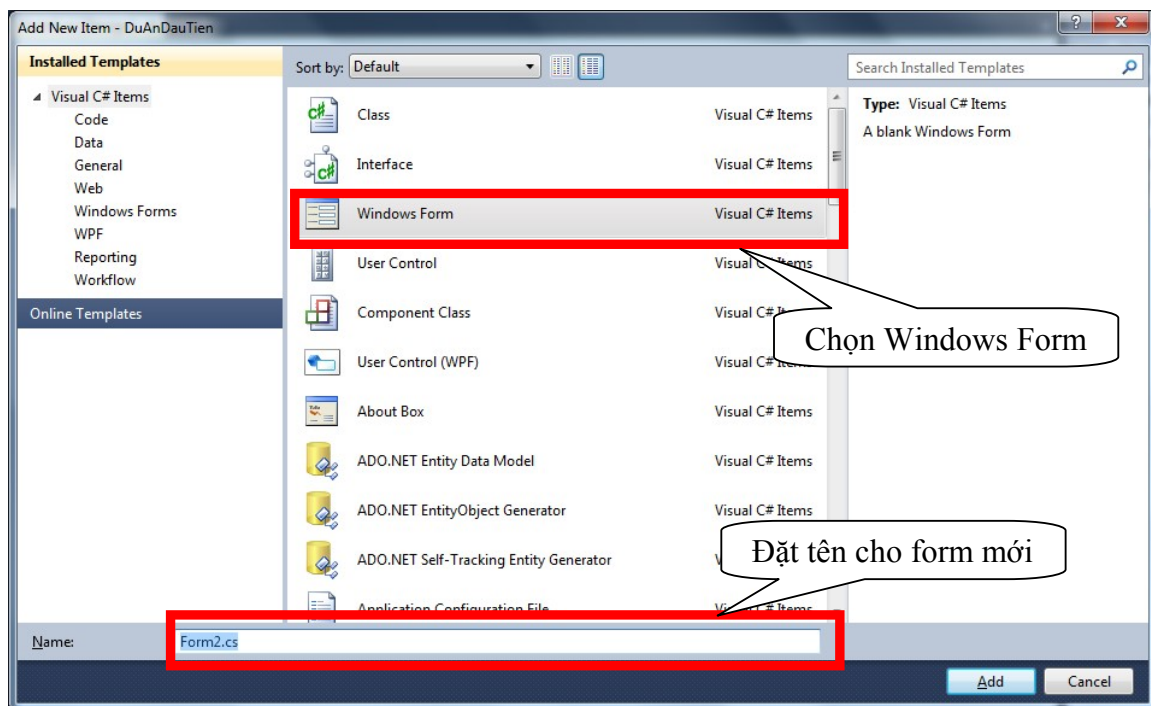


Hình 2.5: Giao diện chính của dự án Windows Forms

➤ Thêm form mới vào dự án:

Phần lớn các dự án phần mềm đều chứa rất nhiều form . Do đó lập trình viên có thể thêm một form mới vào dự án và thiết kế form đó tại thời điểm thiết kế hoặc lập trình viên có thể thêm form mới bằng cách viết các đoạn mã tạo form và form mới sẽ được tạo ra tại thời điểm thực thi của chương trình.

Để thêm một form mới vào dự án tại thời điểm thiết kế, trên menu Project, chọn Add Windows Form. Một hộp thoại Add New Item được mở ra. Sau đó chọn mục Windows Form và đặt tên cho form mới như hình 2.6. Nhấp Add để hoàn tất việc thêm form mới vào dự án phần mềm.



Hình 2.6: Cửa sổ Add New Item

Để thêm một Form mới sau khi đã hoàn thành việc thiết kế giao diện, lập trình viên sẽ phải viết các đoạn mã chương trình tạo form và sau đó form mới sẽ được tạo và hiển thị trong quá trình chương trình được thực thi.

Khái báo đối tượng thuộc lớp Form như sau:

```
Form form1;  
form1 = new Form();
```

Hiển thị Form khi tại thời điểm chương trình thực thi cần sử dụng đến phương thức Show() hoặc ShowDialog():

```
form1.Show();
```

2.1.2. Không gian tên (namespace)

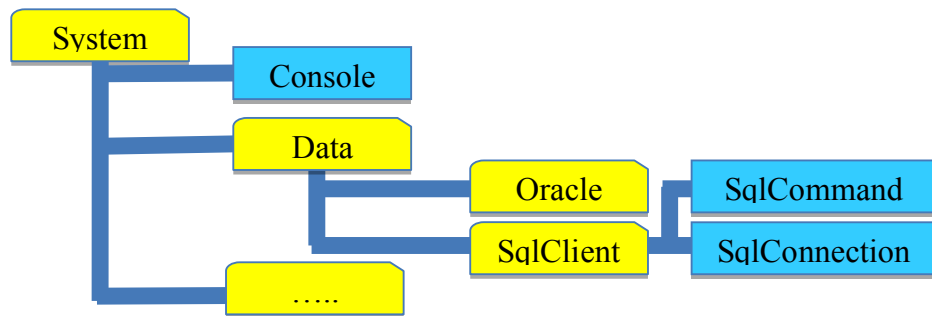
Namespace được gọi là không gian tên. Ý nghĩa lớn của việc sử dụng không gian tên là tránh sự xung đột tên lớp (class) trong một dự án (project). Ngoài ra, không gian tên còn giúp phân nhóm những lớp có cùng chức năng nhằm giúp dễ dàng quản lý mã nguồn. Cụ thể, để làm việc với màn hình Console, C# đã tạo ra một lớp Console, lớp này chứa những thuộc tính và phương thức chỉ dành riêng làm việc với môi trường Console, lớp Console này đặt trong không gian tên System;

Không gian tên tổ chức dạng phân cấp, y như việc phân cấp của cây thư mục. Trong một không gian tên có thể chứa nhiều lớp, các lớp trong cùng một không gian tên không được trùng tên. Hình 2.7 cho thấy sự phân cấp của không gian tên:

Không gian tên System chứa không gian tên Data;

Không gian tên Data chứa không gian tên Oracle và SqlClient;

Trong không gian tên SqlClient chứa nhiều lớp khác tên nhau như: lớp SqlCommand, lớp SqlConnection, ...



Hình 2.7: Tổ chức phân cấp của không gian tên trong C#

➤ Sử dụng không gian tên:

C# xây dựng sẵn nhiều lớp đối tượng đặt trong các không gian tên khác nhau phục vụ cho việc lập trình. Để sử dụng không gian tên trong C#, cần sử dụng từ khóa using.

Ví dụ 2.1:

```
using System;  
using System.Windows.Forms;
```

Khi tạo một dự án kiểu Windows Application, mặc định dự án sẽ sử dụng 8 không gian tên sau:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;
```



```
using System.Linq;  
using System.Text;  
using System.Windows.Forms;
```

Ý nghĩa của các không gian tên như sau:

- Không gian tên System: Chứa các lớp và các kiểu dữ liệu cơ sở (int, double, byte, short, ...).
- Không gian tên System.Data: Chứa các lớp của ADO.NET, các lớp giúp kết nối hoặc thao tác với cơ sở dữ liệu (SQL Server, Oracle, Microsoft Access).
- Không gian tên System.Drawing: Chứa các lớp làm việc với đồ họa.
- Không gian tên System.ComponentModel: Chứa các lớp cơ sở và giao diện giúp thực thi chuyển đổi kiểu, thuộc tính, liên kết với các dữ liệu nguồn và các thành phần cấp phép
- Không gian tên System.Collections.Generic: Chứa các lớp hỗ trợ cho việc thiết lập các tập hợp dạng chung mẫu (template) như:
 - ArrayList List<T>: Tập hợp chung mẫu có thể thay đổi kích thước khi vận hành.
 - Queue Queue<T>: Tập hợp chung mẫu theo mô hình vào trước ra trước(FIFO).
 - Stack Stack<T>: Tập hợp chung mẫu theo mô hình vào sau ra trước (LIFO)
 - ...
- Không gian tên System.Text: Cung cấp các lớp giúp mã hóa các ký tự Ascii hoặc Unicode.
- Không gian tên System.Linq: Cung cấp các lớp hỗ trợ cho việc sử dụng ngôn ngữ truy vấn tích hợp (Language-Integrated Query – LinQ).
- Không gian tên System.Windows.Forms: Là không gian tên chính cung cấp các lớp để xây dựng ứng dụng kiểu Windows Application. Không gian tên System.Windows.Forms chia thành các nhóm sau:
 - Control, UserControl, Form
 - Menu, toolbar: ToolStrip, MenuStrip, ContextMenuStrip, StatusStrip
 - Controls: TextBox, ComboBox, ListBox, ListView, WebBrowser, Label, ...
 - Layout: FlowLayoutPanel, TableLayoutPanel, ... Giúp trình bày các điều khiển trên form.

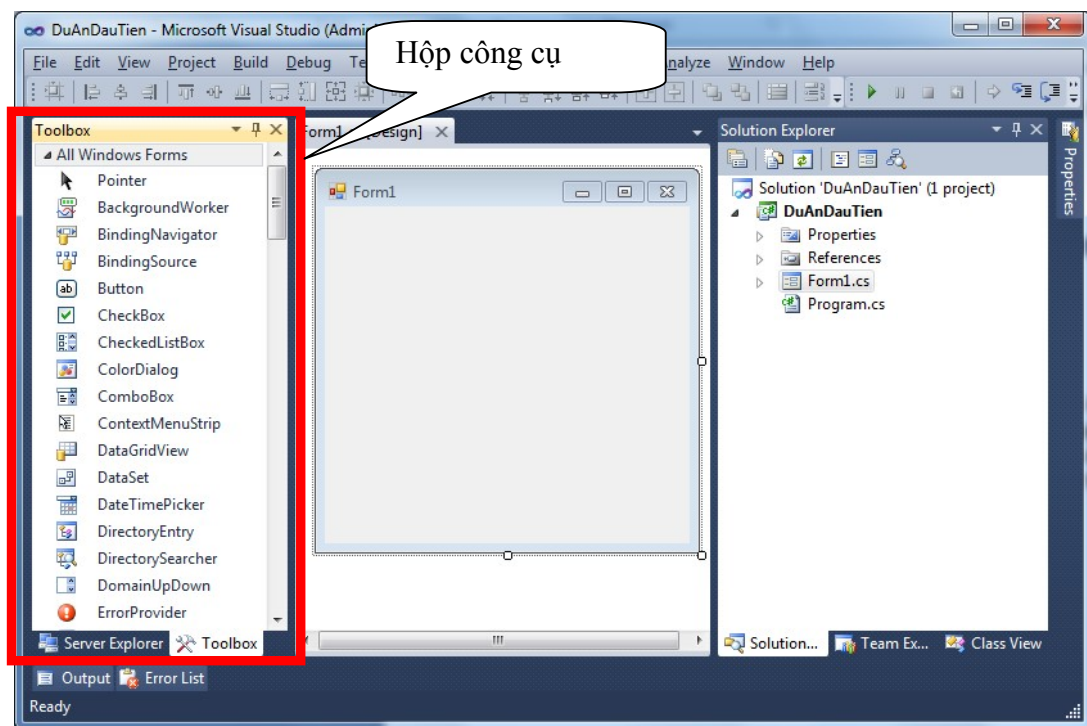
- Data và DataBinding: Gồm các công cụ BindingNavigator, BindingSource, ... giúp liên kết dữ liệu từ cơ sở dữ liệu đến các điều khiển DataSet hoặc DataGridView.
- Command Dialog Boxes: Gồm các điều khiển OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog, PrintDialog làm việc với tập tin, màu sắc, phông chữ, in ấn.

2.1.3. Thanh công cụ (Toolbox)

Cung cấp danh sách các Component/ Control được liệt kê theo nhóm, cho phép lập trình viên sử dụng thao tác kéo thả vào form để thiết kế giao diện chương trình.

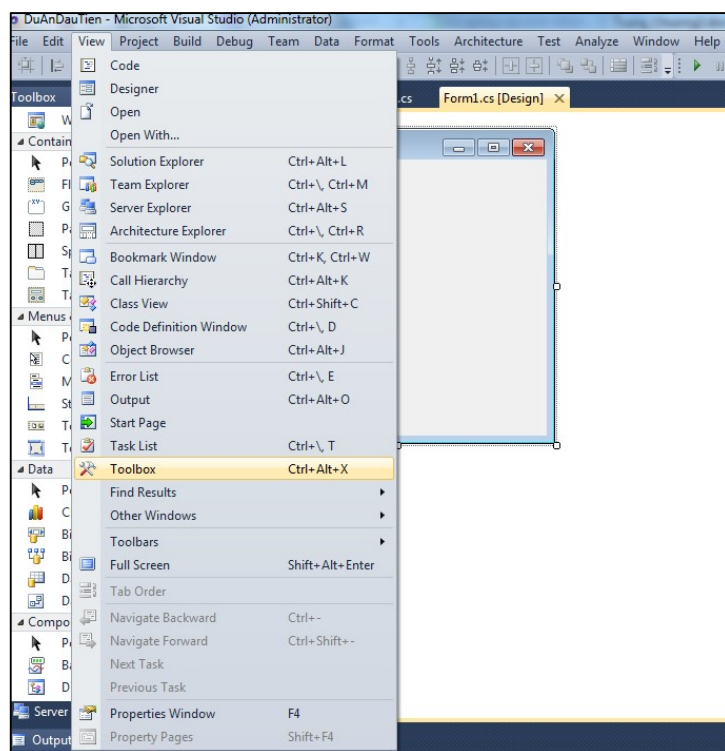
Khi tạo xong dự án, thì giao diện chính của dự án được hiển thị như hình 2.8. Phía bên tay trái hình 2.8 là thanh công cụ.

Ngoài những điều khiển hiển thị mặc định trong thanh công cụ, lập trình viên cũng có thể thêm các thành phần mới vào bằng cách chọn *Project/AddReference*.



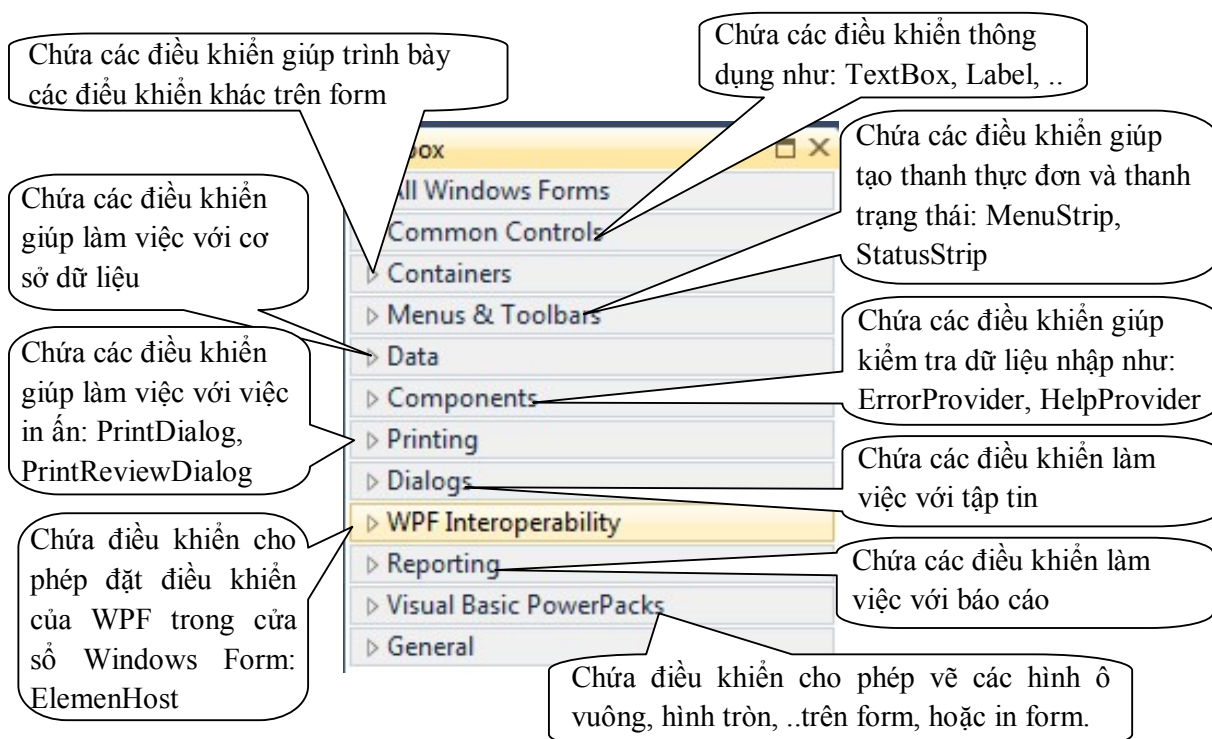
Hình 2.8: Giao diện chính với thanh công cụ bên trái

Nếu giao diện chính không hiển thị thanh công cụ, lập trình viên có thể hiển thị bằng cách chọn *View/ Toolbox* như hình 2.9.



Hình 2.9: Hiện thị thanh công cụ

Thanh công cụ bố trí các điều khiển thành những nhóm riêng biệt như hình 2.10:

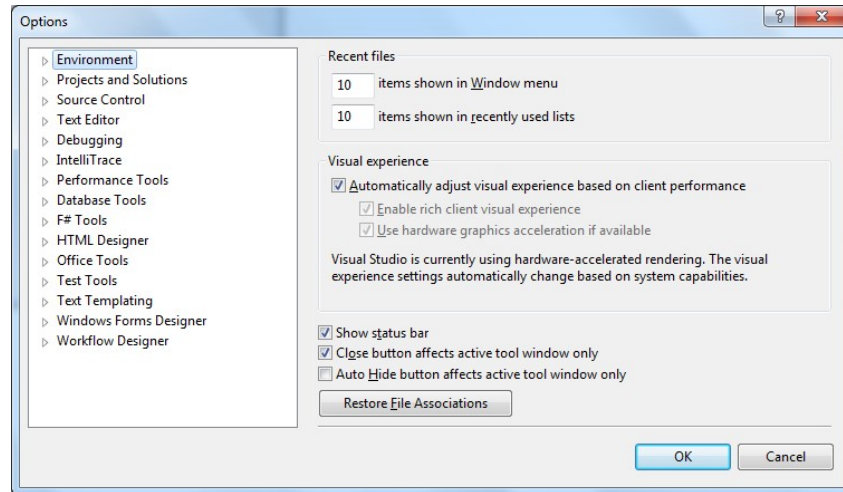


Hình 2.10: Thanh công cụ

Với việc phân loại các điều khiển theo nhóm giúp cho lập trình viên dễ dàng tìm kiếm điều khiển cần sử dụng hơn.

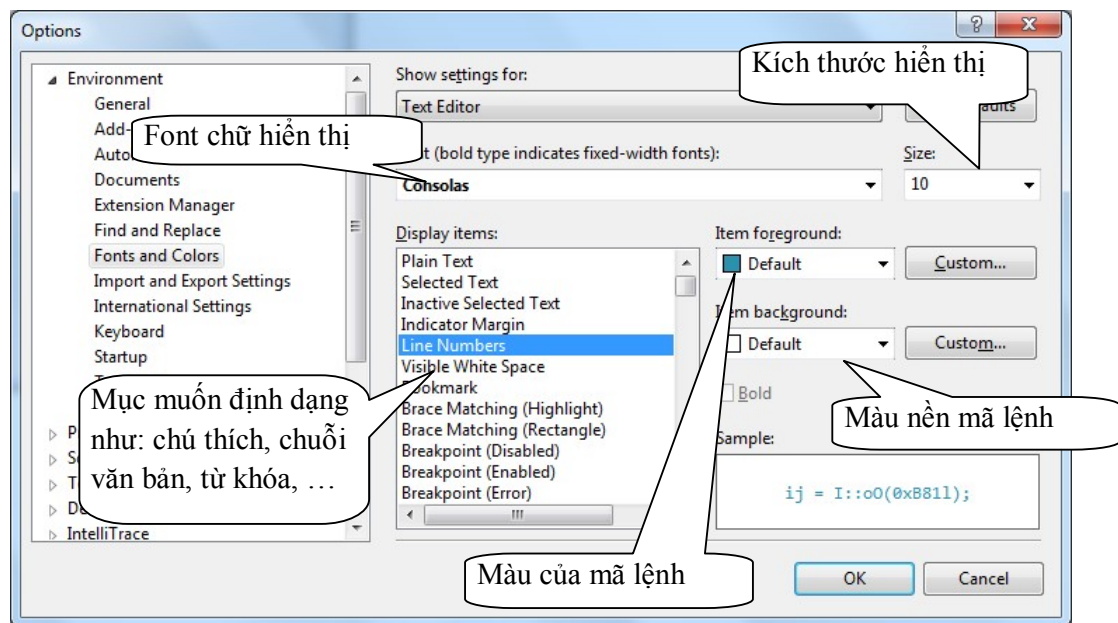
2.1.4. Định dạng mã C#

Để định dạng mã lệnh khi lập trình. Lập trình viên của thẻ tùy chỉnh lại định dạng mã trong cửa sổ Option. Đầu tiên bật cửa sổ Option bằng cách chọn *Tools/Options* như hình 2.11:



Hình 2.11: Giao diện cửa sổ Option

Sau khi đã mở cửa sổ Option, chọn mục Environment/ Fonts and Colors như hình 2.12 để định dạng lại mã lệnh:



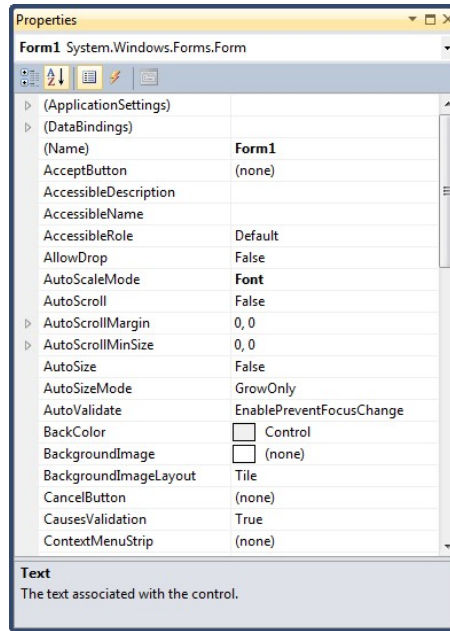
Hình 2.12: Giao diện định dạng mã lệnh

2.2. Các loại Form

2.2.1. Thuộc tính Form

- Thiết lập giá trị thuộc tính form trong cửa sổ Properties:

Windows Forms hỗ trợ nhiều thuộc tính cho việc tùy chỉnh form. Lập trình viên có thể thay đổi giá trị các thuộc tính này trong cửa sổ Properties như hình 2.13 để điều chỉnh kích thước, màu sắc, font chữ, ... của form cho phù hợp.



Hình 2.13: Giao diện cửa sổ Properties

Bảng 2.1 mô tả một số thuộc tính thường sử dụng trong việc thiết kế form

Bảng 2.1: Bảng mô tả thuộc tính Form

Thuộc tính	Mô tả
<i>Name</i>	Đặt tên form
<i>AcceptButton</i>	Giá trị thuộc tính này nhận là tên của một button trên form. Khi đó thay vì nhấp chuột vào button để thực thi thì người dùng có thể ấn phím Enter trên bàn phím
<i>BackColor</i>	Thiết lập màu nền của form
<i>BackgroundImage</i>	Thiết lập hình nền cho form
<i>BackgroundImageLayout</i>	Thiết lập việc hiển thị hình vừa thêm trong thuộc tính BackgroundImage sẽ hiển thị trên form ở dạng: bình thường (None), giữa (Center), ...
<i>CancelButton</i>	Giá trị thuộc tính này nhận là tên của một button trên form. Khi đó thay vì nhấp chuột vào button để thực thi thì người dùng có thể ấn phím Escape trên bàn phím
<i>ControlBox</i>	Mang giá trị true hoặc false. Nếu thiết lập

	thuộc tính là false thì sẽ loại bỏ các nút minimize và nút maximize trên form
<i>Cusor</i>	Thiết lập hình dạng con trỏ khi di chuyển con trỏ vào form
<i>Enable</i>	Mang giá trị true hoặc false; Nếu thiết lập thuộc tính là false thì điều khiển trong form sẽ không cho phép người dùng thao tác.
<i>Font</i>	Thiết lập văn bản hiển thị trên điều khiển
<i>ForeColor</i>	Thiết lập màu mặc định cho chuỗi của các điều khiển trên form
<i>FormBorderStyle</i>	Thiết lập đường viền của form và hành vi của form khi chạy chương trình
<i>HelpButton</i>	Mang giá trị true hoặc false; Nếu thiết lập thuộc tính là true thì trên thanh titlebar sẽ hiện 1 nút có dấu ? (nút này chỉ hiện khi hai thuộc tính MinimizeBox và MaximizeBox được thiết lập là false)
<i>Icon</i>	Biểu tượng hiển thị bên trái trên thanh titlebar của form
<i>KeyReview</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true cho phép các sự kiện bàn phím của form có hiệu lực
<i>Location</i>	Khi thuộc tính StartPosition được thiết lập là Manual, thì thuộc tính Location có tác dụng thiết lập vị trí hiển thị của form trên màn hình
<i>MaximizeBox</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là false thì nút maximize form trên thanh titlebar sẽ mất đi
<i>MaximumSize</i>	Thiết lập kích thước lớn nhất của form (chiều rộng x chiều cao)
<i>MinimizeBox</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là false thì nút minimize form trên thanh titlebar sẽ mất đi
<i>MinimumSize</i>	Thiết lập kích thước nhỏ nhất của form (chiều rộng x chiều cao)

<i>Opacity</i>	Thiết lập độ trong suốt cho form
<i>Size</i>	Kích thước form
<i>StartPosition</i>	Vị trí hiển thị của form trên màn hình
<i>Text</i>	Chuỗi văn bản hiển thị trên titlebar của form
<i>TopMost</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true thì form sẽ luôn hiển thị trên các cửa sổ khác
<i>Visible</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true thì form sẽ được hiển thị trên màn hình, nếu là false sẽ không hiển thị trên màn hình
<i>WindowState</i>	Có 3 giá trị: Normal: hiển thị form bình thường; Minimized: khi chạy chương trình form sẽ bị thu nhỏ dưới thanh taskbar; Maximized: form hiển thị có kích thước đầy màn hình
<i>IsMdiContainer</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> - Nếu là True: Form ở dạng MDI Form - Nếu là False: Form ở dạng bình thường
<i>MdiParent</i>	mang giá trị là đối tượng MDI Form. Khi thiết lập giá trị cho thuộc tính MdiParent thì form sẽ trở thành Child Form

➤ Thiết lập giá trị thuộc tính form bằng mã lệnh:

Ngoài việc thiết kế form bằng viết thiết lập các giá trị trong cửa sổ properties.

Lập trình viên cũng có thể thiết lập giá trị thuộc tính của form bằng mã lệnh như sau:

- Thay đổi chuỗi văn bản hiển thị trên titlebar bằng cách thiết lập giá trị cho thuộc tính *Text*:

```
Form1.Text = "Đây là Form1";
```

- Thiết lập kích thước form: Có thể thiết lập giá trị cho thuộc tính *Width* và thuộc tính *Height* để quy định chiều rộng và chiều cao cho form.

```
Form1.Width = 300;  
Form1.Height = 400;
```

hoặc:

```
Form1.Size = new Size(300, 400);
```

Lưu ý: Nếu thuộc tính *StartPosition* được thiết lập là *WindowsDefaultBounds*, thì form sẽ được thiết lập kích thước mặc định. *StartPosition* được thiết lập giá trị khác thì kích thước form sẽ được thiết lập như quy định ở thuộc tính *Size*.

- Thiết lập độ trong suốt cho form:

```
Form1.Opacity = 0.5;
```

- Thiết lập vị trí hiển thị của form: Sử dụng thuộc tính *StartPosition* để thiết lập vị trí hiển thị ban đầu của form trên màn hình. Giá trị thiết lập của thuộc tính *StartPosition* là một trong các giá trị quy định sẵn trong biến kiểu liệt kê *FormStartPosition*:

Bảng 2.2: Mô tả các giá trị của *FormStartPosition*

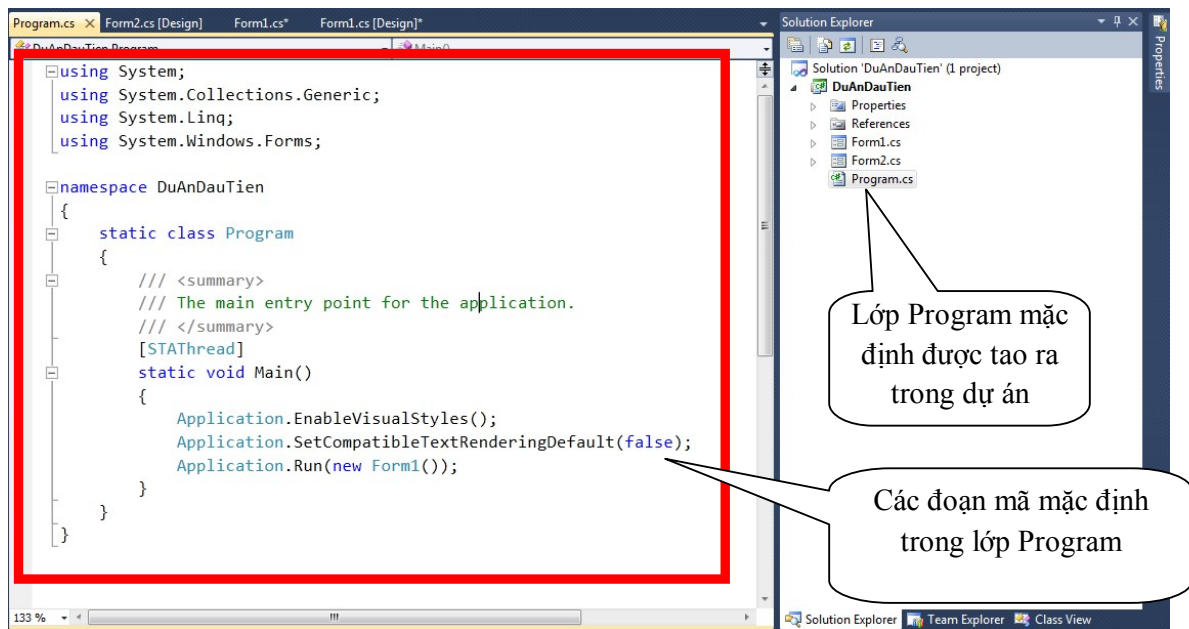
Giá trị	Mô tả
Manual	Nếu <i>StartPosition</i> mang giá trị Manual thì vị trí form hiển thị sẽ là vị trí thiết lập tại thuộc tính <i>Size</i> .
CenterScreen	Form hiển thị giữa màn hình
WindowsDefaultLocation	Form hiển thị tại vị trí mặc định với kích thước form sẽ là kích thước form được thiết lập tại thuộc tính <i>Size</i>
WindowsDefaultBounds	Form hiển thị tại vị trí mặc định với kích thước form sẽ là kích thước mặc định
CenterParent	Form hiển thị ở vị trí giữa form cha

Ví dụ 2.2: Hiển thị Form1 ở vị trí giữa màn hình

```
Form1.StartPosition = FormStartPosition.WindowsDefaultLocation;
```

- Thiết lập form hiển thị:

Trong C#, lập trình viên có thể thiết lập form hiển thị đầu tiên khi dự án được thực thi bằng cách sửa lại mã lệnh trong hàm main của lớp Program như hình 2.14:



Hình 2.14: Lớp Program

Lớp Program chứa một hàm bên trong là hàm Main. Khi dự án được thực thi, thì mã lệnh trong hàm Main sẽ thực thi trước. Cụ thể tại dòng:

```
Application.Run(new Form1());
```

Dòng lệnh trên chỉ ra rằng form đầu tiên sẽ hiển thị là Form1. Do đó, lập trình viên chỉ cần sửa lại mã lệnh chỉ định một form khác muốn hiển thị bằng cách thay tên form đó cho Form1.

Ví dụ 2.3: Hiển thị form có tên Form2 trong dự án

```
Application.Run(new Form2());
```

2.2.2. Các loại Form

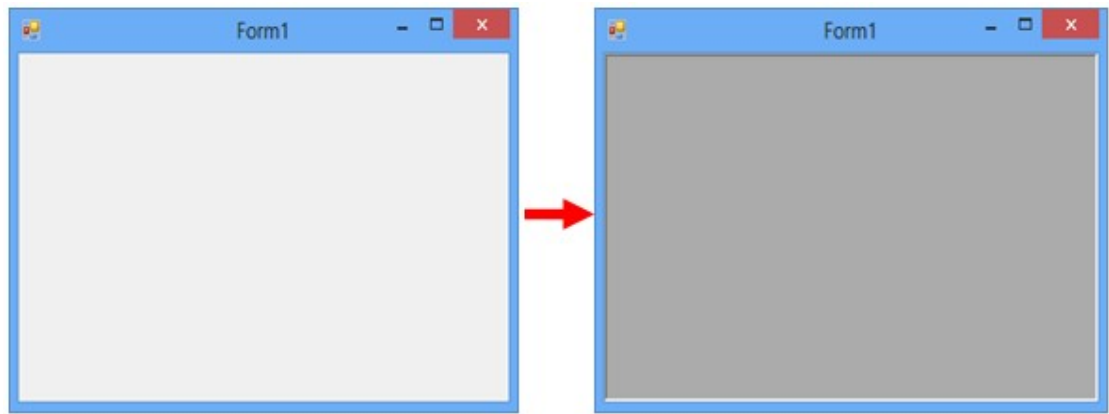
Trong ứng dụng Windows Forms, có 3 loại form: form bình thường (Normal Form), form cha (Mdi Form), form con (Child Form).

➤ Mdi Form:

Mdi Form là form có thể chứa các Child Form bên trong. Để làm được công việc đó thì form phải được thiết lập thuộc tính:

```
IsMdiContainer = True;
```

Lập trình viên có thể thiết lập thuộc tính *IsMdiContainer* trong cửa sổ Properties trên màn hình thiết kế form, hoặc bằng mã lệnh. Khi đã thiết lập thuộc tính *IsMdiContainer* là True thì hình dạng form sẽ thay đổi như hình 2.15.



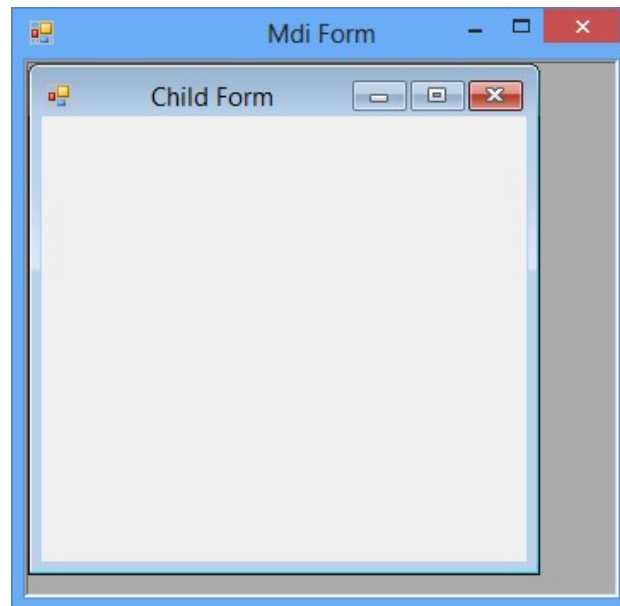
Hình 2.15: Hình dạng form khi chuyển từ Normal Form sang Mdi Form

➤ Child Form:

Child Form là dạng form nằm bên trong vùng làm việc của Mdi Form hay nói cách khác là Child Form được chứa bên trong Mdi Form. Một form muốn trở thành Child Form cần phải khai báo thuộc tính MdiParent có giá trị là đối tượng Mdi Form.

Thuộc tính MdiParent không được biểu diễn trên cửa sổ Properties, do đó lập trình viên bắt buộc phải thiết lập giá trị MdiParent bằng mã lệnh.

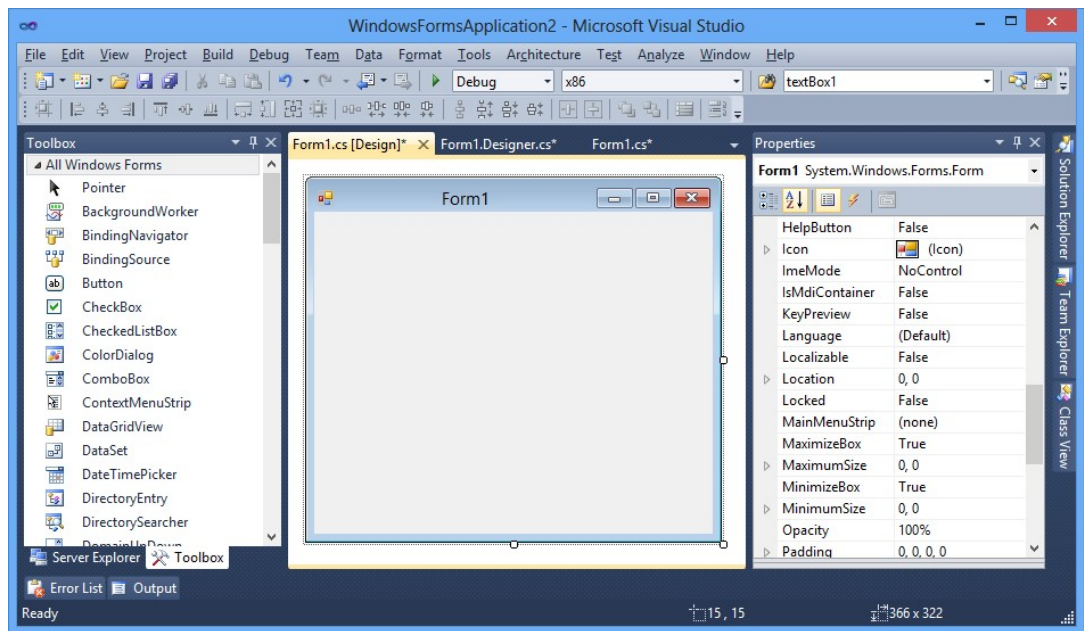
Ví dụ 2.4: Viết chương trình biểu diễn Mdi Form và Child Form như hình 2.16.



Hình 2.16: Mdi Form và Child Form

Hướng dẫn:

- Bước 1: Tạo một dự án mới bằng C#, loại dự án là Windows Forms Application.
 Giao diện đầu tiên của chương trình hiển thị Form1 như hình 2.17.



Hình 2.17: Giao diện Form1 khi tạo dự án

Bước 2: Viết mã lệnh cho chương trình

Tại sự kiện *Load* của Form1 viết các dòng lệnh sau:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Mdi Form";
    this.IsMdiContainer = true;
    Form frm = new Form();
    frm.Text = "Child Form";
    frm.MdiParent = this;
    frm.Show();
}
```

Bước 3: Nhấn Ctrl + Shift + B để biên dịch mã nguồn và nhấn F5 để thực thi chương trình sẽ được Mdi Form và Child Form như hình 2.16.

➤ Normal Form:

Normal Form là form hoạt động độc lập với tất cả các form khác, nghĩa là form sẽ không chứa hoặc được chứa bởi một form nào khác. Thông thường khi tạo mới dự án Windows Forms Application thì form mặc định được tạo và hiển thị đầu tiên là Normal Form. Giao diện của Normal Form như hình 2.17.

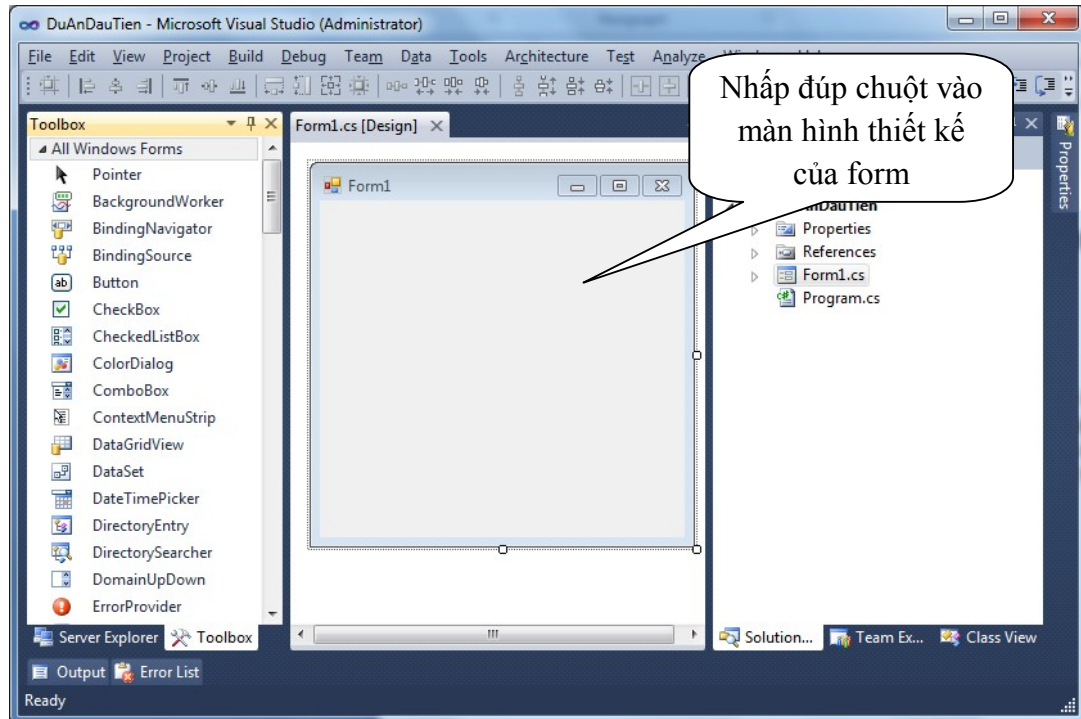
2.2.3. Các hình dạng của Form

Windows Forms mặc định các form được tạo ra đều ở dạng hình chữ nhật. Nếu lập trình viên muốn tạo form với hình dạng khác như hình tròn hoặc hình đa giác thì cần thiết lập lại thuộc tính *Region* của form trong sự kiện *Form_Load*. Tuy nhiên, lập trình viên không thể thấy được hình dạng mới của form trong lúc thiết kế mà chỉ có thể

thấy được hình dạng khác của form khi chương trình được thực thi. Do đó sẽ cần phải thực thi chương trình nhiều lần để kích thước form phù hợp với nhu cầu thiết kế.

Ví dụ 2.5: Thiết kế form có hình ellipse có chiều rộng là 300 và chiều cao là 200

Bước 1: Tạo sự kiện *Form_Load* bằng cách nhấp đúp chuột vào màn hình thiết kế form như hình 2.18:



Hình 2.18: Màn hình thiết kế form

Bước 2: Sau khi tạo ra sự kiện *Form_Load*, tiến hành viết các dòng lệnh trong sự kiện *Form_Load* như sau:

```
private void Form1_Load(object sender, EventArgs e)
{
    System.Drawing.Drawing2D.GraphicsPath path = new
        System.Drawing.Drawing2D.GraphicsPath();
    path.AddEllipse(0, 0, 300, 200);
    Region myRegion = new Region(myPath);
    this.Region = myRegion;
}
```

Bước 3: Ấn F5 để thực thi chương trình, khi chương trình được thực thi sẽ được form như hình 2.19:




Hình 2.19: Form hình Ellipse

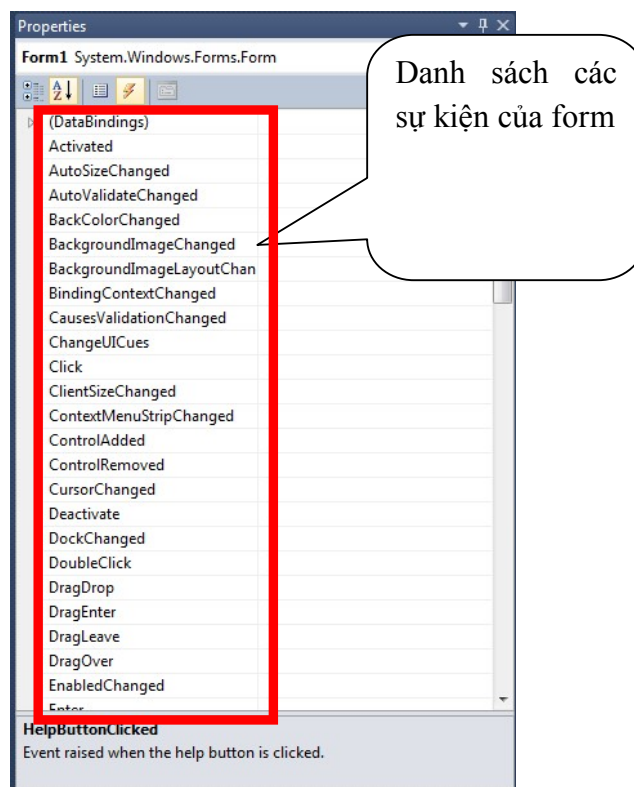
2.2.4. Biến cố của Form

Biến cố của form hay có thể gọi là các hành động hoặc sự kiện liên quan đến form. Ứng dụng Windows Forms được hỗ trợ nhiều sự kiện và các đoạn mã tạo ra các sự kiện đều được sinh ra một cách tự động. Để sử dụng các sự kiện của form lập trình viên cần thực hiện các thao tác sau:

Bước 1: Nhấp chuột trái vào form

Bước 2: Trên cửa sổ Properties tương ứng, nhấp chuột trái vào biểu tượng  để mở hộp thoại các sự kiện như hình 2.20.

Bước 3: Hộp thoại hình 2.20 chứa nhiều sự kiện liên quan đến form như: đóng form, mở form, di chuyển chuột vào form, ... Lập trình viên muốn sử dụng sự kiện nào chỉ cần nhấp đôi chuột trái vào sự kiện đó.



Hình 2.20: Danh sách sự kiện form

Ví dụ 2.6: Muốn sử dụng sự kiện nhấp chuột của form chỉ cần nhấp đôi chuột vào sự kiện *Click* thì một phương thức nhấp chuột của form sẽ được tự động phát sinh mã lệnh:

```
private void Form1_Click(object sender, EventArgs e)
{
    //Mã lệnh khi nhấp chuột vào form
}
```

➤ Bảng mô tả các sự kiện thường sử dụng của form:

Bảng 2.3: Bảng mô tả các sự kiện của form

Sự kiện	Mô tả
AutoSizeChanged	Xảy ra khi thuộc tính Autosize của Form chuyển từ True sang False hay ngược lại là False sang True
BackColorChanged	Xảy ra khi thuộc tính BackColor của Form thay đổi
Click	Xảy ra khi người dùng Click chuột vào vùng làm việc thuộc Form
ControlAdded	Xảy ra khi một điều khiển được Add vào Form
ControlRemoved	Xảy ra khi một điều khiển bị xóa khỏi Form
CursorChanged	Xảy ra khi thuộc tính Cursor của Form thay đổi
DoubleClick	Xảy ra khi người dùng DoubleClick vào vùng làm việc của Form
FontChanged	Xảy ra khi thuộc tính Font của Form có sự thay đổi
ForeColorChanged	Xảy ra khi thuộc tính ForeColor của Form có sự thay đổi
FormClosed	Xảy ra khi Form đã đóng (Nhấn vào nút X màu đỏ trên titlebar)
FormClosing	Xảy ra khi Form đang đóng (2 sự kiện FormClosed và FormClosing thường dùng trong lập trình CSDL: khi xảy ra sự kiện này thì đóng kết nối CSDL)
KeyDown	Xảy ra khi người dùng nhấn một phím hay một tổ hợp phím
KeyPress	Xảy ra khi người dùng nhấn một phím
KeyUp	Xảy ra khi người dùng nhả một phím
MouseClick	Xảy ra khi người dùng nhấn chuột (một trong 3 lựa chọn: Trái, giữa, phải)
MouseDoubleClick	Xảy ra khi người dùng nhấp đúp chuột vào một vùng làm việc của Form (một trong 3 lựa chọn: Trái,

	giữa, phải)
MouseDown	Xảy ra khi người dùng nhấn chuột
MouseHover	Xảy ra khi người dùng di chuyển vào các vùng làm việc Form
MouseLeave	Xảy ra khi di chuyển chuột ra khỏi vùng làm việc của Form
MouseMove	Xảy ra khi di chuyển chuột trên một vùng làm việc thuộc Form (nếu Form có chứa một điều khiển nào đó, khi di chuyển chuột trên điều khiển này thì không xảy ra sự kiện MouseMove của Form)
MouseUp	Xảy ra khi người dùng nhả nhấn chuột (có thể là chuột trái, chuột phải, chuột giữa - chuột cuộn)
Move	Xảy ra khi di chuyển Form (có sự thay đổi vị trí của Form)
StyleChanged	Xảy ra khi thuộc tính FormBorderStyle của Form thay đổi
TextChanged	Xảy ra khi thuộc tính Text của Form thay đổi.

Sự kiện FormClosed: Sự kiện này được gọi khi Form đã đóng

```
private void frmForm_FormClosed(object sender,
                                FormClosedEventArgs e)
{
    MessageBox.Show("Sự kiện FormClosed được gọi");
}
```

Sự kiện FormClosing: Xảy ra khi Form đang đóng

```
private void frmForm_FormClosing(object sender,
                                FormClosingEventArgs e)
{
    DialogResult kq = MessageBox.Show("Bạn muốn đóng
                                     Form lại không?", "FormClosing",
                                     , MessageBoxButtons.YesNo,
                                     MessageBoxIcon.Information);
    if ( kq == DialogResult.Yes)
        e.Cancel = false;    // Đóng Form lại
    else
        e.Cancel = true;     //Không đóng Form nữa
}
```

Sự kiện *KeyPress*: Xảy ra khi ấn phím, nếu không chỉ rõ phím nào được nhấn thì khi nhấn bất cứ phím nào của sự kiện *KeyPress* của form đều xảy ra.

```
private void frmForm_KeyPress(object sender,
                               KeyPressEventArgs e)
{
    if (e.KeyChar == 'a') //nhấn phím a trên bàn phím
        MessageBox.Show("Sự kiện KeyPress được gọi");
}
```

Sự kiện *KeyDown*:

```
private void frmForm_KeyDown(object sender,
                              KeyEventArgs e)
{
    // Nhấn Ctrl+F gọi sự kiện KeyDown
    if (e.KeyCode == Keys.F &&
        e.Modifiers == Keys.Control)
        MessageBox.Show("Sự kiện KeyDown được gọi");
}
```

Sự kiện *MouseClick*: Xảy ra khi nhấn một trong 3 nút: chuột trái, giữa hoặc chuột phải.

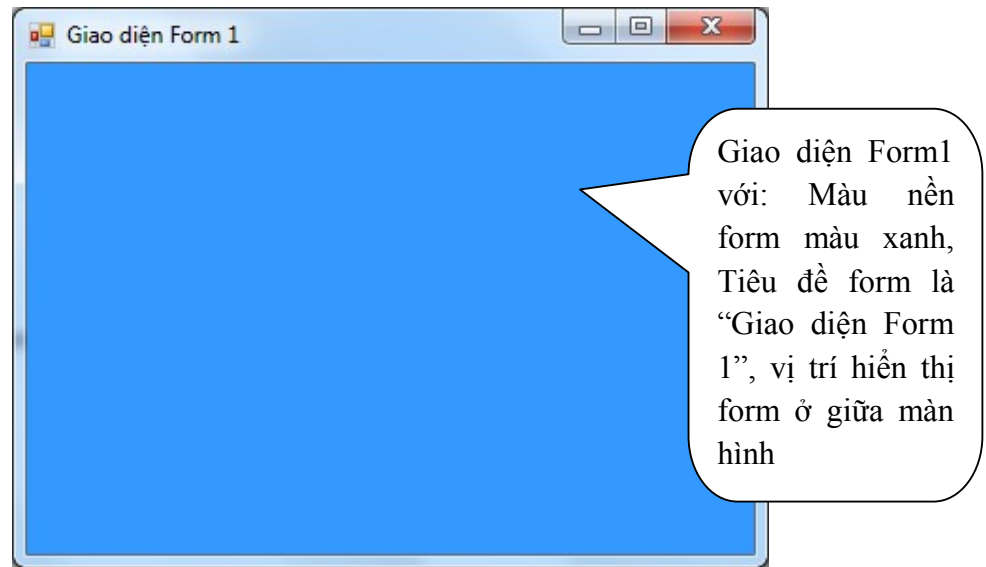
```
private void frmForm_MouseClick(object sender,
                                 MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) //nhấn chuột trái
        MessageBox.Show("Sự kiện MouseClick được gọi");
    else
        if (e.Button == MouseButtons.Middle) //nhấn chuột giữa
            MessageBox.Show("Sự kiện MouseClick được gọi");
        else
            if (e.Button == MouseButtons.Right) //nhấn chuột phải
                MessageBox.Show("Sự kiện MouseClick được gọi");
}
```

2.2.5. Phương thức

Một số phương thức thường sử dụng của form: *Show()*, *ShowDialog()*, *Hide()*, *Close()*.

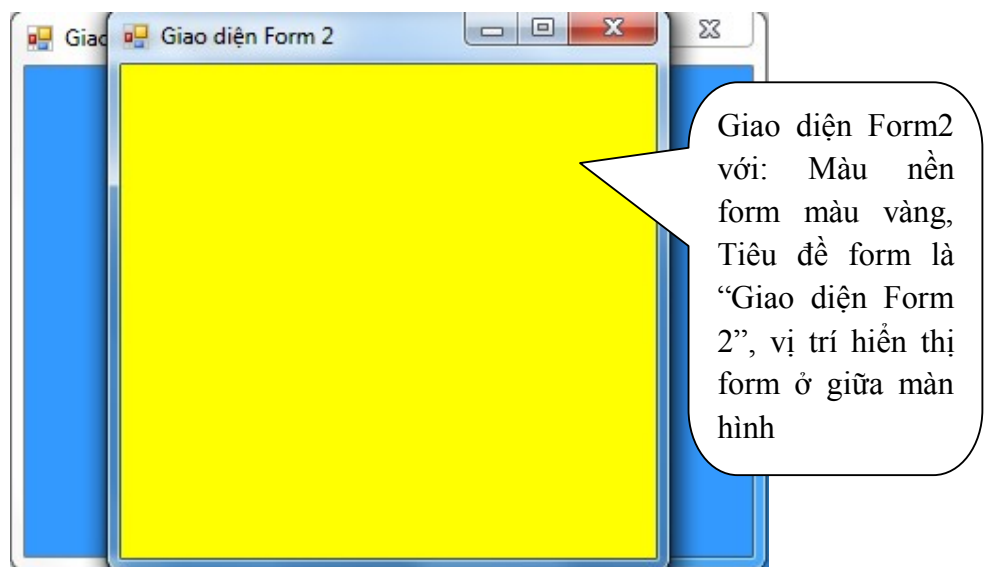
- Phương thức *Show()*: Phương thức *Show()* sử dụng để hiển thị form trên màn hình.

Ví dụ 2.7: Thiết kế chương trình như hình 2.21:



Hình 2.21: Giao diện Form1

Yêu cầu khi nhấp chuột vào Form1 như hình 2.21 thì sẽ hiển thị Form2 ở giữa màn hình như hình 2.22:



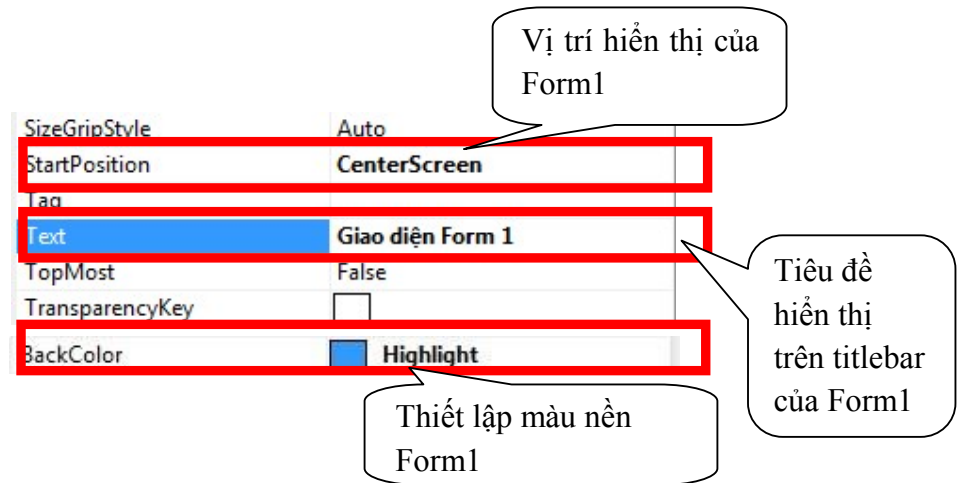
Hình 2.22: Giao diện Form2

Hướng dẫn:

Bước 1: Tạo Dự án Windows Forms mới với form mặc định ban đầu tên Form1.

Bước 2: Thiết lập các thuộc tính cho Form1 như sau:

- Thiết lập tiêu đề, màu nền và vị trí hiển thị cho Form1: Trên cửa sổ Properties thiết lập thuộc tính *Text*, thuộc tính *BackColor* và thuộc tính *StartPosition* như hình 2.23:



Hình 2.23: Thiết lập thuộc tính Form1

Sau khi thiết lập xong 3 thuộc tính như hình 2.23 sẽ được Form1 có giao diện như hình 2.21.

Bước 3: Tạo sự kiện *Click* của Form1 và thêm các mã lệnh sau cho sự kiện Click:

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Phương thức Show() giúp hiển thị Form2
    Form2.Show();
}
```

Khi bước 3 thực hiện xong và nhấn F5 để thực thi chương trình và *Click* chuột vào Form1 thì sẽ được Form2 như hình 2.22.

- Phương thức *ShowDialog()*: Phương thức *ShowDialog()* cũng có tác dụng hiển thị form như phương thức *Show()*. Nhưng khác biệt cơ bản là phương thức *Show()* giúp hiển thị form mới lên nhưng người dùng vẫn có thể quay lại thao tác trên các form đang hiển thị trước đó; Còn phương thức *ShowDialog()* sẽ hiển thị form mới lên và người dùng sẽ chỉ có thể thao tác trên form vừa hiển thị mà không thao tác được trên các form trước đó, do vậy người dùng muốn thao tác lên form hiển thị trước thì phải đóng form hiện hành bằng phương thức

ShowDialog(). Một đặc điểm khác nữa là phương thức *ShowDialog()* là phương thức trả về hai giá trị là *DialogResult.OK* và *DialogResult.Cancel*.

- Phương thức *ShowDialog()* trả về *DialogResult.OK* khi form đang hiển thị.
- Phương thức *ShowDialog()* trả về *DialogResult.Cancel* khi form đóng.

Ví dụ 2.8: Trong sự kiện *Click* của *Form1* như bước 3 trên, tiến hành thay phương thức *Show()* thành phương thức *ShowDialog()*. Khi chương trình được thực thi thì người dùng chỉ có thể thao tác với *Form2* mà không thể quay trở về *Form1* được.

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Phương thức ShowDialog() giúp hiển thị Form2
    Form2.ShowDialog();
}
```

- Phương thức *Hide()*: Phương thức giúp ẩn một form để form đó không hiển thị trên màn hình. Việc ẩn form của phương thức *Hide()* thực chất là thiết lập thuộc tính *Visible = false*.

Ví dụ 2.9: Trong sự kiện *Click* của *Form1* như bước 3, thực hiện yêu cầu là khi nhấp chuột vào *Form1* thì *Form1* sẽ ẩn đi và *Form2* sẽ hiện lên.

```
private void Form1_Click(object sender, EventArgs e)
{
    //this là con trỏ đại diện cho Form hiện hành, Form
    //hiện hành ở đây là Form1
    this.Hide();
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Phương thức ShowDialog() giúp hiển thị Form2
    Form2.ShowDialog();
}
```

➤ Phương thức *Close()*: Sử dụng để đóng form.

Ví dụ 2.10: Trong sự kiện *Click* của Form1 như bước 3, thực hiện yêu cầu là khi đóng Form2 thì Form1 cũng đóng.

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Kiểm tra giá trị trả về của phương thức ShowDialog()
    //nếu giá trị trả về là DialogResult.Cancel thì Form2
    //đã đóng, tiến hành đóng Form1 bằng phương thức
    //Close()
    if (Form2.ShowDialog() == DialogResult.Cancel)
    {
        this.Close();
    }
}
```

2.3. Bài tập cuối chương

Câu 1: Trong các dòng mã lệnh sau đây, mã lệnh nào cho phép tạo và hiển thị một đối tượng Windows Form mới có tên là Form1

- a) Form1 frm = new Form1;
 frm.Show();
- b) Form Form1 = new Form();
 Form1.Show();
- c) Form1 frm ;
 frm.Show();
- d) Form frm;
 frm.Show();
- e) Form Form1 = new Form();
 Form1.ShowDialog();

Câu 2: Trong các thuộc tính sau, thuộc tính nào dùng để thiết lập nội dung hiển thị trên thanh title bar và thuộc tính nào dùng để thiết lập màu nền của form.

- a) Thuộc tính Text và ForeColor
- b) Thuộc tính Display và BackColor
- c) Thuộc tính Text và BackColor

d) Thuộc tính Display và ForeColor

Câu 3: Các thuộc tính sau, thuộc tính nào cho phép thiết lập form trở thành Mdi Form

- a) IsMdiContainer
- b) MdiParent
- c) MdiContainer
- d) ParentForm

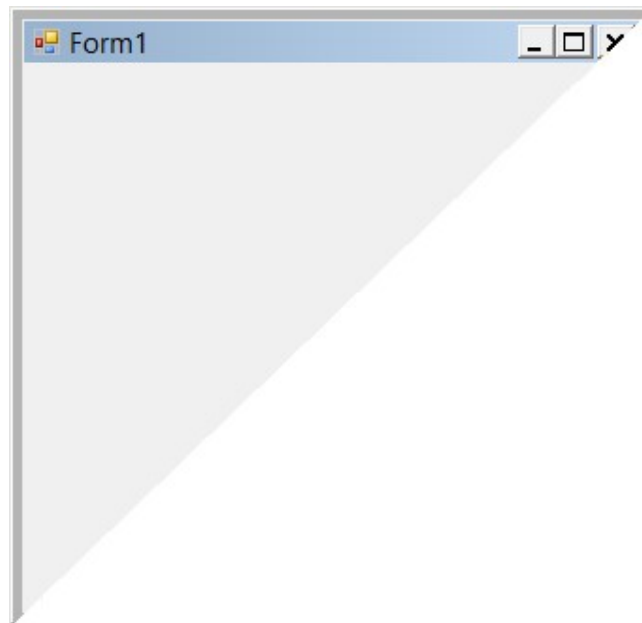
Câu 4: Các thuộc tính sau, thuộc tính nào cho phép thiết lập form trở thành Child Form

- a) IsMdiContainer
- b) MdiParent
- c) MdiContainer
- d) ParentForm

Câu 5: Trong các sự kiện sau, sự kiện nào sẽ phát sinh khi form đã đóng

- a) FormClosed
- b) FormClosing
- c) ClosedForm
- d) ClosingForm
- e) Load
- f) Click

Câu 6: Xây dựng form có dạng hình tam giác như hình 2.21:



Hình 2.21: Form hình dạng tam giác