بنام خدا

طراحی قالب نرم افزاربا#C

نویسنده کتاب:

مهندس احمدی

کلمات کلیدی:

Component (قطعه): شرکت های نرم افزاری در ابتدا با مشکلات زیادی از جمله پیچیده بودن کد، خوانا نبودن سورس برنامه (جهت تغییرات بعدی) ، حجم بالای کد، هزینه بالای تولید و ... مواجه بودند. این مشکلات با آوردن مفهوم جدید قطعه بندی در تحلیل نرم افزار برطرف گردید. امروزه یک نرم افزار پیچیده از قطعات

مختلفی که قاعدتا نیاز به تحلیل ساده تری دارند تشکیل شده است. این مفاهیم در شی گرایی به صورت بارزتری استفاده می شود.

Abstract Class (کلاس انتزاعی): کلاسهایی که به عنوان کلاس پایه در نظر گرفته میشوند و ناکامل هستند. اجزای این کلاس در کلاسهایی که از آن مشتق میشوند کامل میشوند.

Partial Class (کلاس ناتمام): هر فایل سورس شامل قسمتهایی از تعریف کلاس است که در زمان کامپایل تمام این قسمتها (در سورس های مختلف) با هم ترکیب میشوند.

Private Methods (متد اختصاصی): متدهایی که به صورت اختصاصی تعریف میشوند و از کلاسهای دیگر قابل دسترسی نیستند.

Name Space (فضای نام): به مسیر کلاسهای مورد استفاده در پروژه .Net فضای نام گفته میشود که با مفهوم Header ها در زبان C قابل مقایسه است.

GDI+ (Graphics Device Interface): مجموعه ای از ابزارهای گرافیکی که در قالب کلاس در .NET وجود دارد و این ابزارها از فضای نام System.Drawing و System.Drawing.Drawing2D قابل دسترسی هستند.

ARGB Color Standard: در این استاندارد از سه کانال رنگی شامل قرمز، سبز، آبی جهت مشخص نمودن رنگ و یک کانال رنگی آلفا جهت مشخص نمودن میزان شفافیت رنگ در ترکیب با پس زمینه می باشد. میزان تفکیک هر کانال 8 بیت می باشد. تعداد رنگ پشتیبانی شده توسط این استاندارد 256*256*256 رنگ که تا 16 میلیون رنگ را شامل می شود.

Gradient: به طیف رنگی که از رنگ خاصی شروع و به تدریج به رنگ دیگری میل می کند.

مقدمه

معمولا یک برنامه برای ارتباط با کاربر خود از محیطی استفاده می کند که به آن GUI (Graphic User Interface) گفته می شود.

به این قسمت برنامه، با توجه به اینکه کاربر با آن در ارتباط است توجه زیادی می شود. طراحی پوسته (Skin) در نرم افزار های مختلف نیز در همین راستا صورت می گیرد. روشهای مختلفی برای تولید یک پوسته مورد استفاده قرار می گیرد که در ادامه، نحوه طراحی پوسته در یک برنامه مانند فتوشاپ و چگونگی پیاده سازی آن در C# نشان خواهد داده شد.

بخش اول (نحوه طراحی پوسته در فتوشاپ):

احتمالا واضح است که چرا از این برنامه جهت طراحی استفاده شده است، در واقع مهمترین دلیل انتخاب آن قدرت این نرم افزار در طراحی و تولید تصاویر گرافیکی برداری و پیکسلی است. در این آموزش از گرافیک برداری در این نرم افزار استفاده خواهد شد.

چرا قبل از کد نویسی از یک برنامه گرافیکی استفاده شده است؟ خوب، مشخصا اگر دید کلی و یک پیش طراحی از طرح مورد نظر وجود داشته باشد، در زمان کد نویسی چرخه تغییر کد-کامپایل، کاهش می یابد. یعنی در زمان کد نویسی بعد از هر تغییر باید کد را کامپایل کرد تا خروجی را مشاهده نمود و برای قسمتهای مختلف، دوباره کد را تغییر و کامپایل کرد. و این روش تا به نتیجه مورد نظر و یک طرح قابل قبول برسد، قاعدتا نیاز به صرف زمان زیادی دارد.

در این بخش به آشنایی اولیه با برخی از ابزار ها که کاربرد بیشتری در طراحی یک پوسته دارند پرداخته می شود. برای توضیحات بیشتر باید به راهنماهای این برنامه مراجعه نمود. واضح است که برای یک طراحی خوب، داشتن خلاقیت و مهارت کامل در این برنامه، ضروری است.

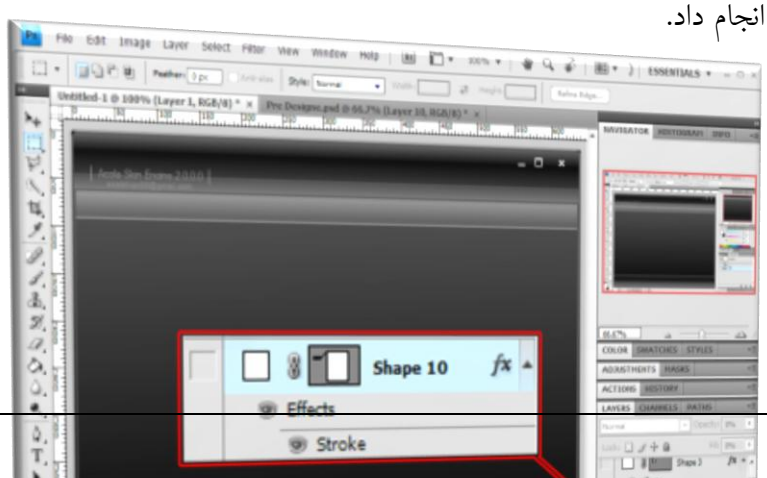
آشنایی با فتوشاپ:

Adobe Photoshop: یک برنامه ویرایش گرافیکی که توسط شرکت Adobe Systems توسعه داده شد.

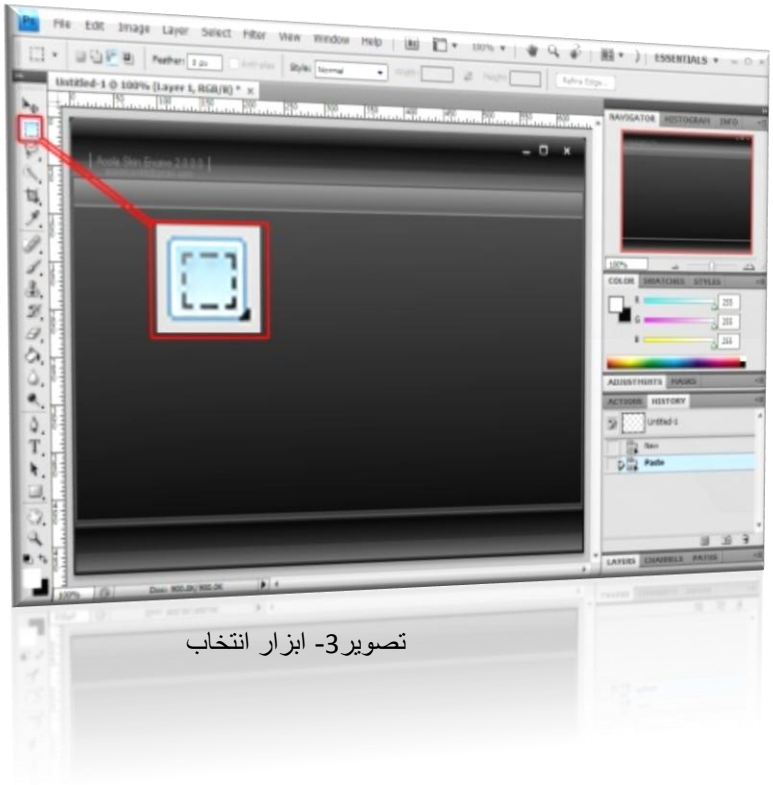
تصویر1- نمایی از فتوشاپ

Plug-in: این قابلیت به برنامه اجازه می دهد که کارکرد خود را توسعه ببخشد. هر Plug-in در واقع یک ویرایشگر کوچک است که به برنامه اضافه شده و قابلیت یا فیلتر خاصی را ارائه می دهد.

Layers: یکی از ویژگی های مهم این برنامه توانایی کار با لایه هاست. ومی توان بر روی هر لایه به صورت مجزا عمل ویرایش یا پردازش تصویر را انجام داد.

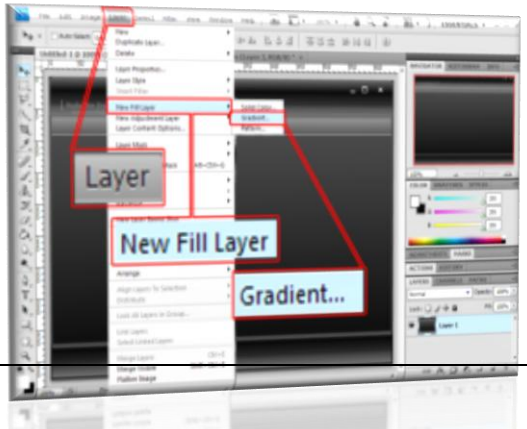


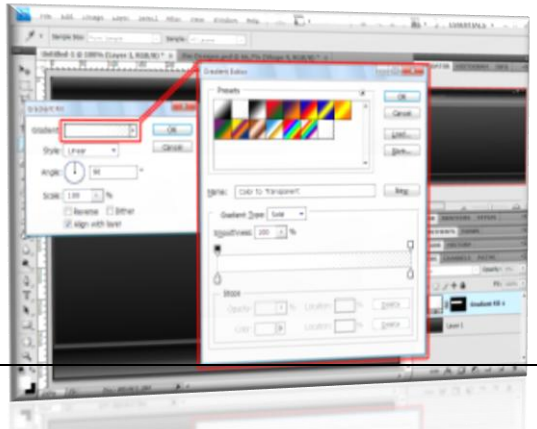
Rectangular marquee tools: جهت انتخاب قسمتی از صفحه، جهت ایجاد یا تغییر گرافیکی در قسمت خاصی از صفحه یا لایه مورد نظر.



تصویر3- ابزار انتخاب

Gradient: برای طراحی طیف رنگی از این ابزار استفاده می شود. چنانچه ناحیه خاصی انتخاب شده باشد، در آن ناحیه ایجاد می شود.





تصویر5- تنظیمات طیف رنگی

تصویر4- ابزار طیف رنگی

Blending Options: در این قسمت می توان افکتها و ویژگیهای مختلفی به لایه مورد نظر افزود. به دلیل اینکه پیاده سازی ویژگی های این قسمت در کد نویسی مشکل (و نه غیر ممکن) است، حد الامکان کمتر از این ابزار باید استفاده شود. زیرا پیاده سازی این قسمت نیاز به دانش و تجربه بیشتری در گرافیک دارد.

تصویر6- ابزار آمیختگی

پس از این آشنایی مختصر با فتوشاپ، طراحی فرم برنامه شروع می شود. همانطور که قبلا نیز اشاره شد برای آشنایی بیشتر به راهنماهای این برنامه مراجع کنید.

ابتدا یک صفحه کاری جدید باز کنید (مثلا در اندازه 800 در 600). در این قسمت با توجه به برداری بودن روند طراحی اندازه صفحه کاری مهم نیست بلکه اندازه لایه های مختلف و اندازه آنها نسبت به هم مهم است که باید بر اساس طرح مورد انتخاب شوند.

با استفاده از ابزار انتخاب، قسمتی از صفحه انتخاب شده و طیف مورد نظر افزورده می شود. این روند تا تشکیل طرح کلی پوسته ادامه می یابد. پس از افزودن لایه می توان میزان شفافیت هر لایه را در ابزار آمیختگی تغییر

داد. پس از اتمام طرح اولیه، باید فایل طراحی را ذخیره نمود تا در مراحل بعدی از این طراحی استفاده شود. حتما توجه شود که هر گونه تغییرات در طرح در مراحل بعدی تاثیر زیادی دارد پس تمام تغییرات باید (یا بهتر است که) در همین مرحله صورت گیرد.

بخش دوم (نحوه پیاده سازی طرح در C#):

لازمه این بخش آشنایی نسبتا کامل با GDI+ می باشد که در ضمیمه آورده شده است. آشنایی اولیه با زبان C# نیز در سندآورده شده که پس از فرا گرفتن آن می توان به ادامه مطلب پرداخت.

بعد از آوردن پروژه جدید Windows Application در ویژوال استادیو، توضیحاتی از پیاده سازی داده خواهد شد. خوب در C# برای افزودن یک رویداد گرافیکی به شی خاص از روش زیر استفاده می شود:

```
this.Paint += new PaintEventHandler(MyForm_Paint);
```

این قطعه کد در قسمت متد سازنده فرم مورد نظر اضافه می شود. با اضافه کردن این رویداد زمانی که نیاز به بازسازی فرم مثلا در Maximize کردن Refresh کردن و... شد، تابع Form_Paint فراخوانی می شود.

نحوه نوشتن تابع به صورت زیر است(در کد قبلی با نوشتن += و دو بار زدن کلید Tab نیز این کد تولید می شود):

```
void MyForm_Paint(object sender, PaintEventArgs e)
{
    throw new NotImplementedException();
}
```

که در بدنه آن باید کدهای گرافیکی پیاده سازی شوند. به عنوان مثال برای افزودن یک مستطیل به فرم کدهای زیر به تابع اضافه می شود.

```
void MyForm_Paint(object sender, PaintEventArgs e)
{
    Point p = new Point(0, 0);       // نقطه شروع
    Size s = new Size(100, 50);      // عرض و ارتفاع
   Rectangle Rect = new Rectangle(p, s); // تعریف مستطیل
   Pen pen = new Pen(Brushes.Aqua);        // تعریف قلم برای خط دور مستطیل
    e.Graphics.DrawRectangle(pen, Rect);    // افزودن این اشیا به فرم
```

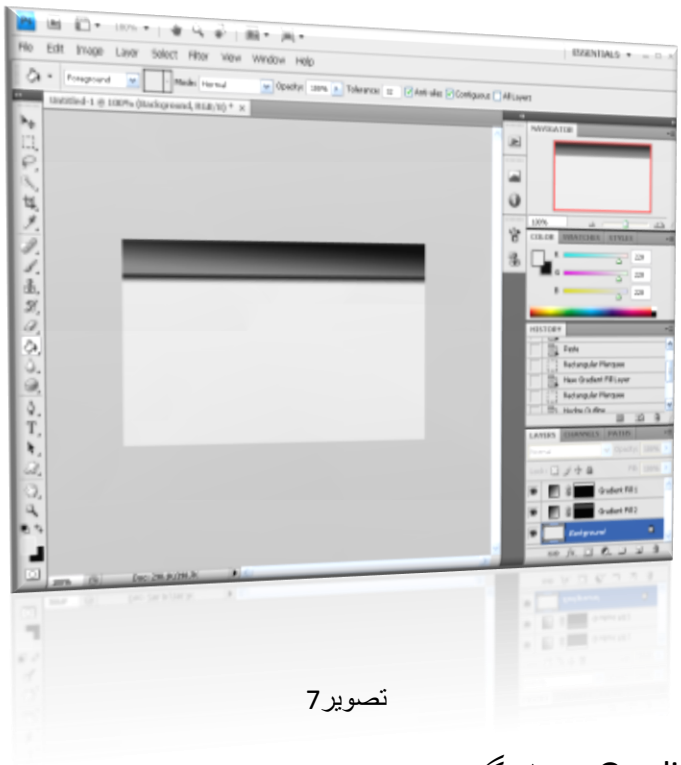
```
}
```

برای اینکه فرم از حالت استاندارد ویندوزی خارج شود (دکمه بستن و ماکسیمایز و مینیمایز و Titlebar نمایش داده نشوند) کد زیر به متد سازنده کلاس اضافه می شود.

```
this.FormBorderStyle = FormBorderStyle.None;
```

بعد از این توضیحات، یکی از لایه ها، مثلا Title bar طرح مورد نظر، برای نمونه پیاده سازی می شود. بدیهی است سایر لایه ها نیز به همین ترتیب پیاده سازی می شوند.

طرح زیر را در فتو شاپ در نظر بگیرید:



تصویر7

برای کد نویسی باید چند متغییر برای هر لایه Gradient در نظر گرفت:

1 -نقطه شروع که بالا ترین و چپ ترین نقطه می باشد شامل دو متغیر X,Y که در C# از نوع Point تعریف می شود.

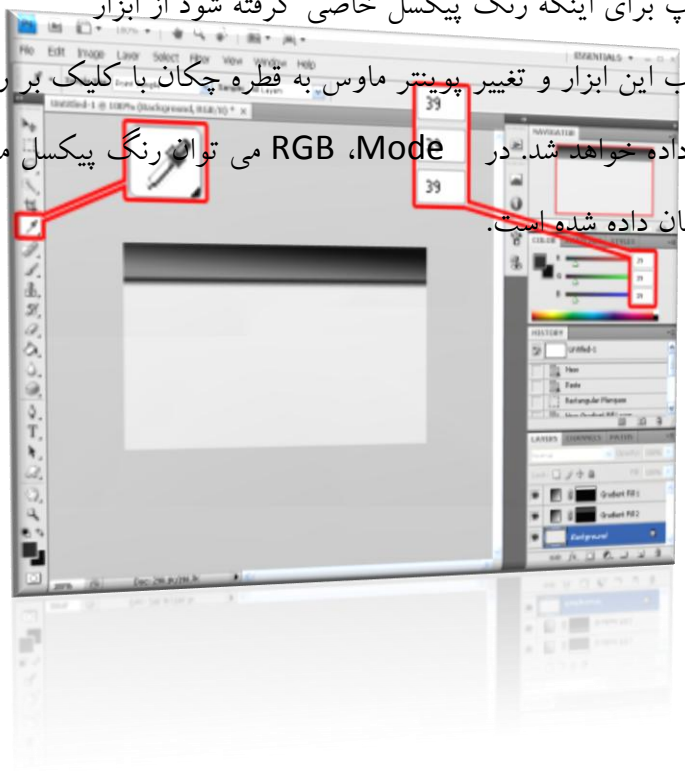
2 -ابعاد لایه شامل عرض و ارتفاع که در C# از نوع Size تعریف می شود.

3 -رنگ شروع و رنگ پایان لایه که در C# از نوع Color تعریف می شود.

4 -نوع Gradient (LinearGradientMode) که از نوع Enum در چهار نوع موجود می باشد. در واقع این پارامتر مشخص کننده جهت طیف می باشد مثلا بالا به پایین، چپ به راست و... .

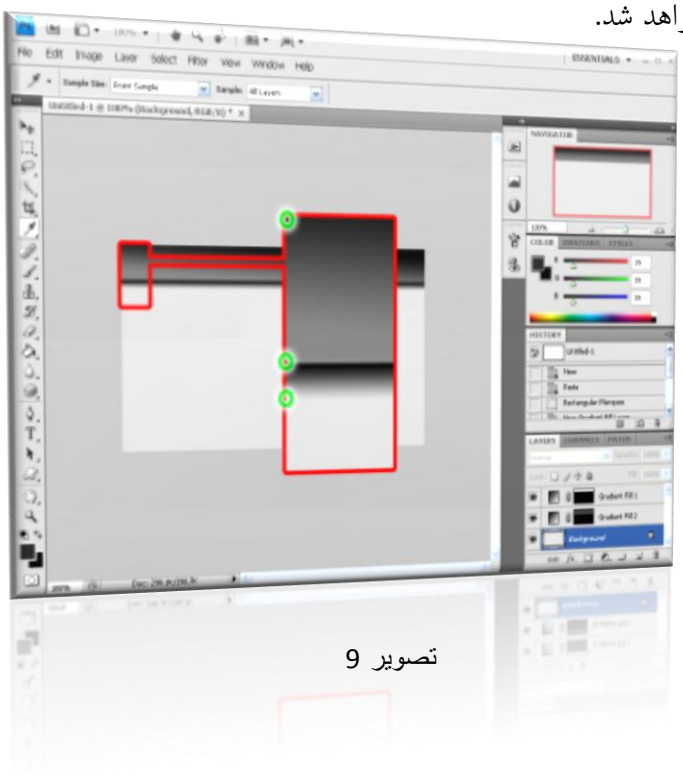
برای موارد 1و2 ابتدا باید خط کش فعال شود برای این منظور باید تیک سربرگ View\Rulers زده شود(یا کلید ترکیبی Ctrl+R). نوع خط کش باید پیکسلی باشد که در صورت نبودن، در تنظیمات برنامه قابل تنظیم است.

این تنظیمات از مسیر Edit\Preferences\Units & Rulers\Rulers قابل دسترسی است. برای موارد 3و4 ابتدا توضیحاتی باید داده شود. در فتو شاپ برای اینکه رنگ پیکسل خاصی گرفته شود از ابزار Eyedropper استفاده می شود. پس از انتخاب این ابزار و تغییر پوینتر ماوس به قطره چکان با کلیک بر روی هر قسمت صفحه کاری رنگ پیکسل نمایش داده خواهد شد. در Mode، RGB می توان رنگ پیکسل مورد نظر را دریافت کرد. در شکل زیر این ابزار نشان داده شده است.



تصویر8- ابزار Eyedropper

بعد از این مرحله باید اطلاعات مختلف، لایه به لایه بر اساس موارد گفته شده در بالا جمع آوری شود. مثلا در مورد نمونه ای که قرار است پیاده سازی شود، رنگ نقاط سبز رنگ شکل زیر، که مشخص شده است باید گرفته شود. رنگ این نقاط در پیاده سازی استفاده خواهد شد.

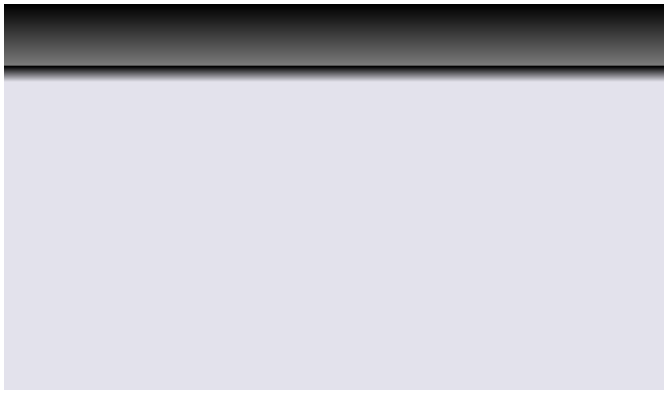


تصویر 9

با توجه به تصویر قبل دو لایه Gradient وجود دارد که باید پیاده سازی شوند. رنگ اولین دایره سبز رنگ را Color1 و دومی را Color2 در نظر بگیرید. رنگ سومین دایره سبز رنگ در حقیقت برابر رنگ پس زمینه است. حال باید تابع به صورت زیر نوشته شود:

```
void FRMMain_Paint(object sender, PaintEventArgs e)
{
       // لایه اول (لایه بالایی)
       Point p = new Point(0, 0); // نقطه شروع لایه
       Size s = new Size(this.Width, 40);
       // 40 عرض لایه به اندازه عرض فرم و ارتفاع لایه
       Rectangle Rect = new Rectangle(p, s);

       Color Color1 = Color.FromArgb(0, 0, 0)
, Color2 = Color.FromArgb(125, 125, 125);
      // رنگهای مشخص شده بالا

       LinearGradientBrush Br = new LinearGradientBrush(Rect, Color1, Color2,
LinearGradientMode.Vertical);
       // تولید طیف رنگی و مشخص کردن نوع آن

     e.Graphics.FillRectangle(Br, Rect);
//-------------------------------------------------------
       // لایه دوم (لایه پایینی که مانند سایه است)
       p.Y += s.Height;
       // نقطه شروع بعد از لایه بالایی

       s.Height = 10;
       Rectangle Rect2 = new Rectangle(p, s);
       LinearGradientBrush Br2 = new LinearGradientBrush(Rect2, Color1,
this.BackColor, LinearGradientMode.Vertical);

       e.Graphics.FillRectangle(Br2, Rect2);

}
```

با کامپایل کد، فرم زیر نمایش داده خواهد شد.



تصویر 10 – فرم برنامه

یک Border یا مرز نیز به طرح فرم اضافه میکنیم تا فرم بهتری داشته باشیم:

```
Rectangle RectBorder = new Rectangle(1,1, this.Width-2,this.Height-2);

e.Graphics.DrawRectangle(new Pen(Brushes.DimGray,2), RectBorder);
```

با کامپایل مجدد کد، فرم زیر نمایش داده خواهد شد:

تصویر 11 – فرم برنامه

همانطور که میبینید نمیتوان از فرم خارج شد! برای این کار باید در Visual studio دکمه Stop Debugging زده شود.

بخش سوم (اضافه کردن کدهایی برای تبدیل طراحی انجام شده به ویژگی های فرمهای عادی):

در فرم باید بتوان آنرا کوچک یا بزرگ کرد و باید یک سری دکمه نیز جهت این کار روی آن داشت. خوب ابتدا از بستن فرم شروع میکنیم:

یک دکمه به فرم اضافه کرده و در رویداد mouse click آن به صورت زیر عمل میکنیم:

- اگر بخواهیم کل فرمهایی که از skin استفاده میکنند بسته شوند:

```
Applicaion.Exit();
```

- اگر بخواهیم فقط فرم جاری بسته شود:

```
This.Close();
```

میتوان رویدادهایی به دکمه اضافه نمود تا مثلا با قرار گرفتن ماوس برروی آن تغییر رنگ دهد، یا عکس خاصی رو کنترل قرار گیرد(که در این مورد می توان از کنترل Picture Box نیز استفاده نمود).

برای نمونه کدهای زیر:

```
private void BTNExit_Click(object sender, EventArgs e)
{
   this.Close();
}

private void BTNExit_MouseEnter(object sender, EventArgs e)
{
   BTNExit.BackColor = Color.Red;
}

private void BTNExit_MouseLeave(object sender, EventArgs e)
{
   BTNExit.BackColor = Color.Gray;
}
```

فرم باید قابلیت جابجایی داشته باشد. برای پیاده سازی باید موارد زیر را به ترتیب در نظر داشت:

1 -آیا دکمه چپ ماوس فشرده شده

2 -آیا این عمل کلیک(یا فشردن دکمه ماوس) در محدوده title bar بوده

3 -آیا ماوس در حالت فشرده بودن حرکت کرده

4 -موقعیت ماوس در فرم و صفحه نمایش چیست

5 -فرم به اندازه جاجایی ماوس (تا زمانی که کلید فشرده شده) جابجا شود

برای جواب به موارد بالا نیاز به پیاده سازی کد در سه event، Mouse Down و Mouse Move و Mouse Up داریم:

```
this.MouseMove += new MouseEventHandler(FRMMain_MouseMove);
this.MouseDown += new MouseEventHandler(FRMMain_MouseDown);
this.MouseUp += new MouseEventHandler(FRMMain_MouseUp);
```

در ادامه کدهای مورد نیاز در توابع آورده شده:

```
bool mousemove = false;
int mousex, mousey, titlebar0 = 50;

void Form1_MouseUp(object sender, MouseEventArgs e)
{
      mousemove = false;
}

void Form1_MouseDown(object sender, MouseEventArgs e)
{
      if (e.Y <= titlebar0)
      {
       mousemove = true;
       mousex = e.X;
       mousey = e.Y;
      }
}

void Form1_MouseMove(object sender, MouseEventArgs e)
{
      if (mousemove == true)
      {
      this.Left += e.X - mousex;
      this.Top += e.Y - mousey;
      }
}
```

titlebar0 مشخص کننده ارتفاع title bar است. که قاعدتا باید به اندازه ارتفاع طرح title bar انتخاب شود.

نحوه جاجایی فرم در event، MouseMove، کد زیر را در نظر بگیرید:

```
this.Left += e.X - mousex;
```

e.X مقعیت ماوس نسبت به فرم را در هر لحظه جاجایی می دهد. Mousex که موقعیت اولیه ماوس در فرم بود. چنانچه ماوس یکی به سمت راست برود e.X یکی از Mousex بزرگتر می شود و حاصل تفریق آن دو مثبت 1 شده و this.Left +=1 می شود. و برعکس اگر به سمت چپ برود e.X یکی از Mousex کوچکتر شده و فرم به چپ می رود. بالا و پایین رفتن فرم نیز به همین صورت عمل می کند.

در نهایت با اضافه کردن یک دکمه کوچک جهت خروج از فرم طرح اینگونه میشود:

تصویر 12

تا اینجا کار طراحی فرم به پایان رسید و در صورتیکه بخواهیم هر گونه تغییری بدهیم بهتر است در همین مرحله باشد(البته در مرحله بعد نیز می شود ولی به دلیل ویژوال نبودن مرحله بعد، این تغییرات کمی سختر می باشد).

کد های نوشته شده تا این مرحله:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
```

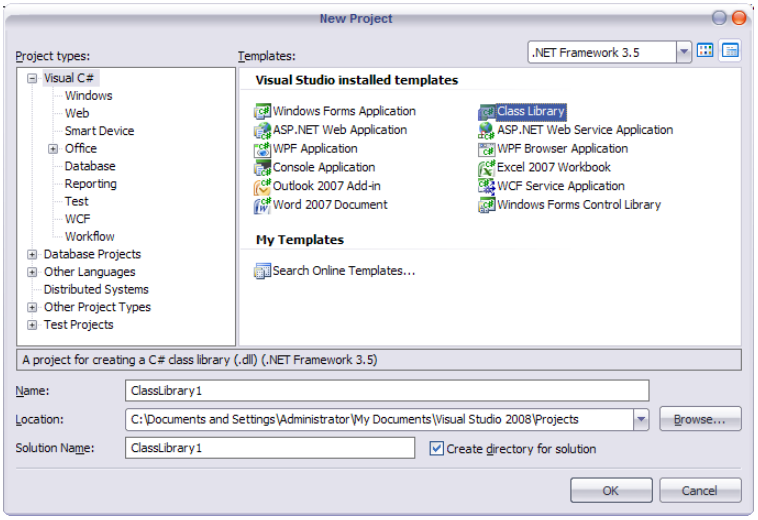
```
namespace SkinTest
{
    public partial class Form1 : Form
    {
        bool CanMove = false, CanResize = false;
        int mousex, mousey, titlebar0 = 50;
        public Form1()
        {
            InitializeComponent();
            this.Paint += new PaintEventHandler(FRMMain_Paint);
            this.MouseMove += new MouseEventHandler(Form1_MouseMove);
            this.MouseDown += new MouseEventHandler(Form1_MouseDown);
            this.MouseUp += new MouseEventHandler(Form1_MouseUp);
            BTNExit.Location = new Point(this.Width - 20, 20);
        }

        void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            CanMove = false;
            CanResize = false;
        }

        void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Y <= titlebar0)
            {
                CanMove = true;
            }
            if (e.X > this.Width - 10 && e.Y > this.Height - 10)
            {
                CanResize = true;
            }
            mousex = e.X;
            mousey = e.Y;
        }

        void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            if (CanResize)
            {
                this.Width += e.X - mousex;
                this.Height += e.Y - mousey;
            }

            if (CanMove == true)
            {
                this.Left += e.X - mousex;
                this.Top += e.Y - mousey;
            }
        }

        void FRMMain_Paint(object sender, PaintEventArgs e)
        {
            Point p = new Point(0, 0);
            Size s = new Size(this.Width, 40);
            Rectangle Rect = new Rectangle(p, s);
```

```
            Color Color1 = Color.FromArgb(0, 0, 0), Color2 =
Color.FromArgb(125, 125, 125);
            LinearGradientBrush Br = new LinearGradientBrush(Rect, Color1,
Color2, LinearGradientMode.Vertical);
            e.Graphics.FillRectangle(Br, Rect);

            p.Y += s.Height;
            s.Height = 10;
            Rectangle Rect2 = new Rectangle(p, s);
            LinearGradientBrush Br2 = new LinearGradientBrush(Rect2, Color1,
this.BackColor, LinearGradientMode.Vertical);
            e.Graphics.FillRectangle(Br2, Rect2);

            Rectangle RectBorder = new Rectangle(1,1, this.Width-
2,this.Height-2);
            e.Graphics.DrawRectangle(new Pen(Brushes.DimGray,2), RectBorder);

            BTNExit.Location = new Point(this.Width - 20, 10);
        }

        private void BTNExit_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void BTNExit_MouseEnter(object sender, EventArgs e)
        {
            BTNExit.BackColor = Color.Red;
        }

        private void BTNExit_MouseLeave(object sender, EventArgs e)
        {
            BTNExit.BackColor = Color.Gray;
        }

    }
}
```

بخش چهارم (تبدیل پروژه به کتابخانه کلاس DLL):

به دلایل مختلف تبدیل چنین پروژه ای به کتابخانه می تواند مفید باشد، برای نمونه راحت تر بودن کار، تحلیل راحت تر و.. و دلایل زیادی که می توان عنوان کرد، مهمترین دلیلی که وجود دارد استفاده از کتابخانه در پروژه های مختلف است.

در ویژوال استادیو برای ایجاد کتابخانه کلاس ابتدا Create Project را زده و ClassLibrary را انتخاب میکنیم. یک پروژه ای ایجاد می شود که خروجی آن یک فایل کتابخانه با فرمت DLL می باشد.



تصویر 13

ابتدا باید بدانیم که در ویژوال استادیو با ایجاد یک پروژه WindowsFormsApplication ویژوال استادیو دو کلاس برای فرم در نظر میگیرد که یکی کلاسی است که ما در آن کد مینویسیم و با زدن View Code به آن دسترسی داریم و دیگری FormDesigner فرم است که کدهای اتوماتیکی که توسط ویژوال استادیو تولید میکند (مثلا زمانی که یک TextBox از ToolsBarدر فرم می اندازیم یک سری کد به این کلاس اضافه میشود) به این کلاس اضافه میشود.

نکته دیگری که وجود دارد این است که این دو کلاس هم نام هستند و کلاس FormDesigner از نوع Partial است. به این معنی که در زمان اجرا این دو کلاس با هم ترکیب شده و Debug میشوند.

خوب جمله های بالا در کتابخانه کردن چه کاربردی دارد؟

جواب این سوال ساده است، یعنی ما کدهای این دو کلاس ، را در کتابخانه کلاسی، که ایجاد کرده ایم کپی میکنیم.

بعد از کپی کردن این دو کلاس به مشکلی بر میخوریم، Debuger ویژوال استادیو به Form و... چند کلمه دیگر Error میدهد دلیل این مشکل کپی نکردن NameSpace های استفاده شده در پروژه قبلی است که الان باید اضافه شوند.

بعد از اضافه کردن این NameSpace ها دوباره با Error مواجه می شویم که این فضای نامها وجود ندارند. برای حل این مشکل به قسمت Solution Explorer رفته و روی شاخه Refrence، کلیک راست و Add Refrence را انتخاب میکنیم. در پنجره باز شده و در سربرگ .NET. باید DLL های زیر اضافه شوند:

- System.Drawing
- System.windows.Form

بعد از این کار دکمه F6 یا Build را جهت تولید DLL میزنیم. ابتدا بدنه کتابخانه کلاس Skin که از دو کلاس تشکیل شده(برای استاندارد تر شدن کتابخانه نام کلاسهای فرم و متد سازنده آن تغییر داده شده و نام پروژه کتابخانه نیز BasicSkinDLL در نظر گرفته شده است):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace BasicSkinDLL
{
    public partial class BasicSkin : Form
    {
        bool CanMove = false, CanResize = false;
        int mousex, mousey, titlebar0 = 50;
        public BasicSkin()
        {
            InitializeComponent();
            this.Paint += new PaintEventHandler(FRMMain_Paint);
            this.MouseMove += new MouseEventHandler(Form1_MouseMove);
            this.MouseDown += new MouseEventHandler(Form1_MouseDown);
            this.MouseUp += new MouseEventHandler(Form1_MouseUp);
            BTNExit.Location = new Point(this.Width - 20, 20);
        }
```

```
        void Form1_MouseUp(object sender, MouseEventArgs e)
        {
            CanMove = false;
            CanResize = false;
        }

        void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Y <= titlebar0)
            {
                CanMove = true;
            }
            if (e.X > this.Width - 10 && e.Y > this.Height - 10)
            {
                CanResize = true;
            }
            mousex = e.X;
            mousey = e.Y;
        }

        void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            if (CanResize)
            {
                this.Width += e.X - mousex;
                this.Height += e.Y - mousey;
            }

            if (CanMove == true)
            {
                this.Left += e.X - mousex;
                this.Top += e.Y - mousey;
            }
        }

        void FRMMain_Paint(object sender, PaintEventArgs e)
        {
            Point p = new Point(0, 0);
            Size s = new Size(this.Width, 40);
            Rectangle Rect = new Rectangle(p, s);
            Color Color1 = Color.FromArgb(0, 0, 0), Color2 =
Color.FromArgb(125, 125, 125);
            LinearGradientBrush Br = new LinearGradientBrush(Rect, Color1,
Color2, LinearGradientMode.Vertical);
            e.Graphics.FillRectangle(Br, Rect);

            p.Y += s.Height;
            s.Height = 10;
            Rectangle Rect2 = new Rectangle(p, s);
            LinearGradientBrush Br2 = new LinearGradientBrush(Rect2, Color1,
this.BackColor, LinearGradientMode.Vertical);
            e.Graphics.FillRectangle(Br2, Rect2);

            Rectangle RectBorder = new Rectangle(1, 1, this.Width - 2,
this.Height - 2);
            e.Graphics.DrawRectangle(new Pen(Brushes.DimGray, 2),
RectBorder);
```

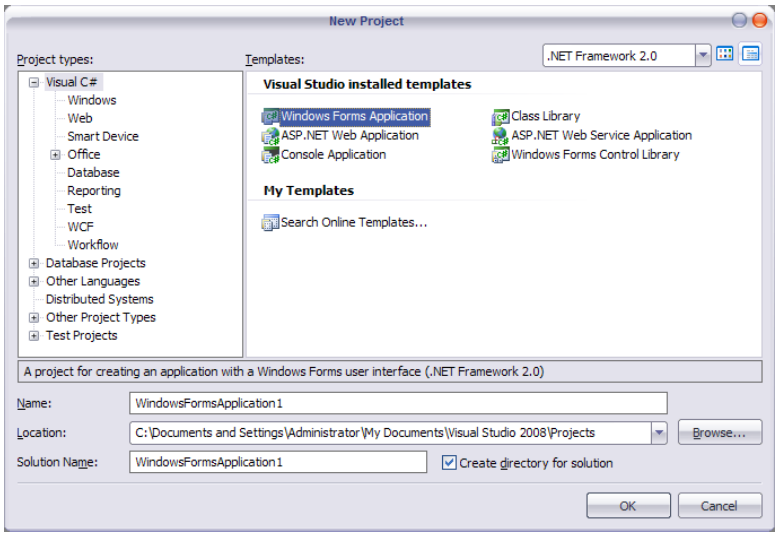
```
            BTNExit.Location = new Point(this.Width - 20, 10);
        }

        private void BTNExit_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void BTNExit_MouseEnter(object sender, EventArgs e)
        {
            BTNExit.BackColor = Color.Red;
        }

        private void BTNExit_MouseLeave(object sender, EventArgs e)
        {
            BTNExit.BackColor = Color.Gray;
        }

    }

    partial class BasicSkin
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.BTNExit = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // BTNExit
            //
```

```
            this.BTNExit.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.BTNExit.BackColor = System.Drawing.Color.Gray;
            this.BTNExit.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
            this.BTNExit.Location = new System.Drawing.Point(407, 12);
            this.BTNExit.Name = "BTNExit";
            this.BTNExit.Size = new System.Drawing.Size(10, 11);
            this.BTNExit.TabIndex = 0;
            this.BTNExit.UseVisualStyleBackColor = false;
            this.BTNExit.MouseLeave += new
System.EventHandler(this.BTNExit_MouseLeave);
            this.BTNExit.Click += new
System.EventHandler(this.BTNExit_Click);
            this.BTNExit.MouseEnter += new
System.EventHandler(this.BTNExit_MouseEnter);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(429, 250);
            this.Controls.Add(this.BTNExit);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.None;
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Button BTNExit;
    }
}
```

در شاخه Bin\Debug پروژه، کتابخانه DLL ایجاد شده، که در مرحله بعد به آن نیاز داریم. البته بنا به دلایل امنیتی و ... بهتر است حالت StartDebuging بر روی Release تنظیم شده باشد، در این حالت کتابخانه DLL ایجاد شده، در مسیر Bin\Release قابل دسترسی است.

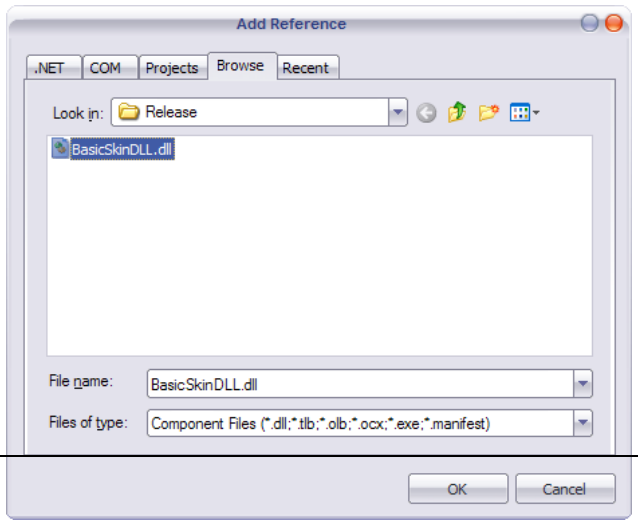
بخش پنجم (نحوه استفاده از کتابخانه کلاس DLL (که در مرحله قبل ایجاد شده است)):

بعد از ایجاد DLL باید بتوان از آن استفاده نمود. برای استفاده، یک پروژه، از نوع



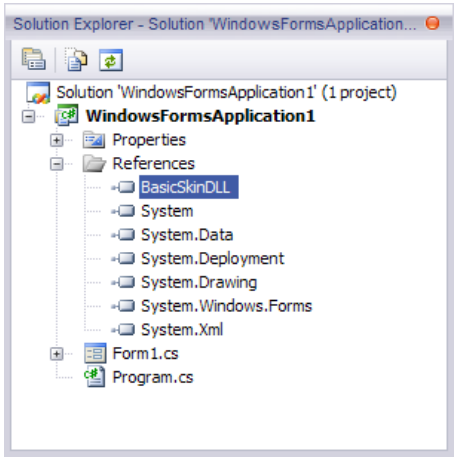
WindowsFormsApplication ایجاد می کنیم.

پس از ایجاد پروژه به قسمت Solution Explorer رفته و روی شاخه Refrence، کلیک راست و Add Refrence را انتخاب میکنیم، از سر برگ Browse به مسیر پروژه کتابخانه رفته و از مسیر bin\Realise،



DLLی را که خودمان ایجاد کردیم (در اینجا با نام BasicSkinDLL.dll) انتخاب میکنیم.

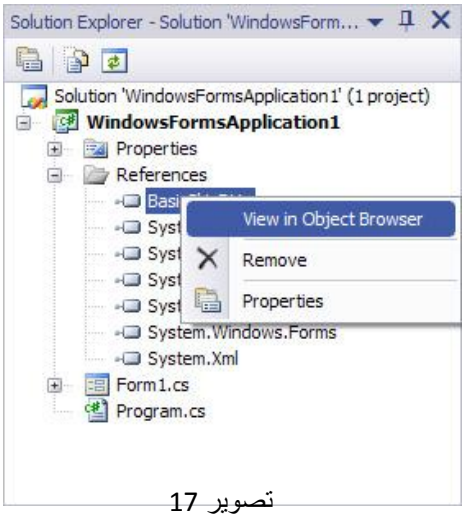
پس از این عمل اگر به شاخه Reference بنگرید، می بینید این DLL نیز اضافه شده است:



تصویر 16

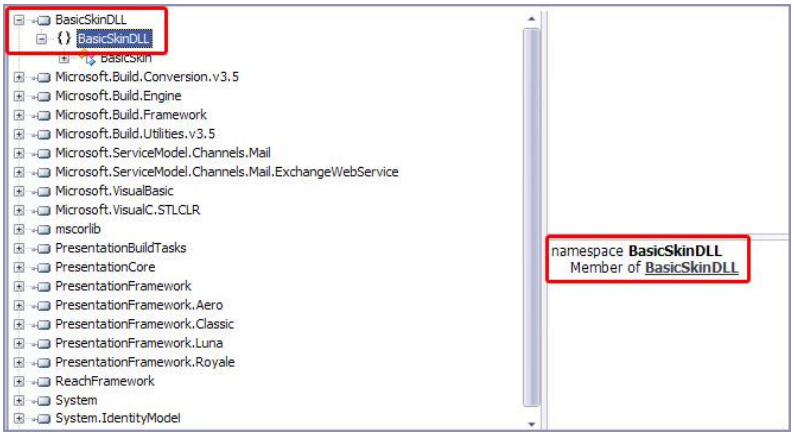
خوب به NameSpace این DLL نیاز داریم، اگر به کدهای کتابخانه نگاهی بیاندازید می توانید فضای نام آن را که BasicSkinDLL مشاهده کنید.

راه بهتر و عمومی تری نیز وجود دارد، به قسمت Solution Explorer رفته و در شاخه Refrence نام BasicSkinDLL را پیدا کنید روی آن کلیک راست کرده و View in Object Browser را انتخاب کنید:



تصویر 17

پنجره Object Browser باز میشود. در این پنجره می توان اطلاعاتی که در سطح Public در کلاس تعریف شده اند را مشاهده نمود مثلا متدها، پروپرتی هاو فضای نام DLL نیز قابل دسترسی است:

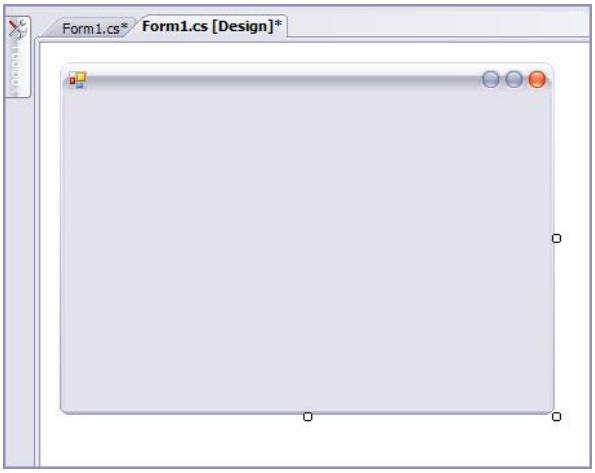


تصویر 18

این NameSpace را به NameSpaceهای پروژه ای که ایجاد کرده ایم اضافه میکنیم:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using BasicSkinDLL;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

حال یک تغییر کوچک مانده تا از Skin استفاده کنیم، قبل از آن به فرم برنامه نگاهی می اندازیم تا بعد از انجام تغییرات مقایسه را ببینیم:



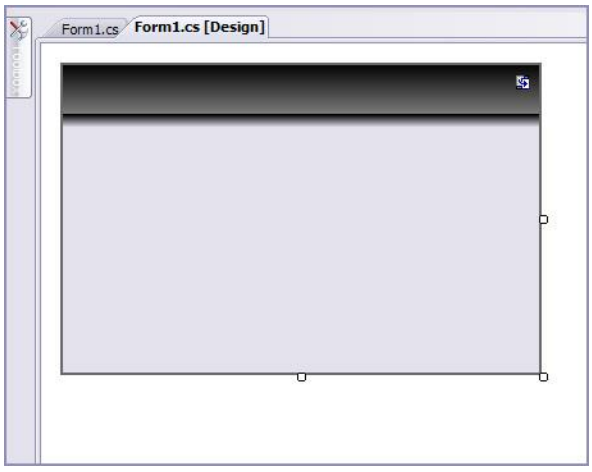
تصویر 19- فرم برنامه قبل از تغییر

به کد بر میگردیم و

`public partial class Form1 : Form` را به

```
public partial class Form1 : BasicSkin
```

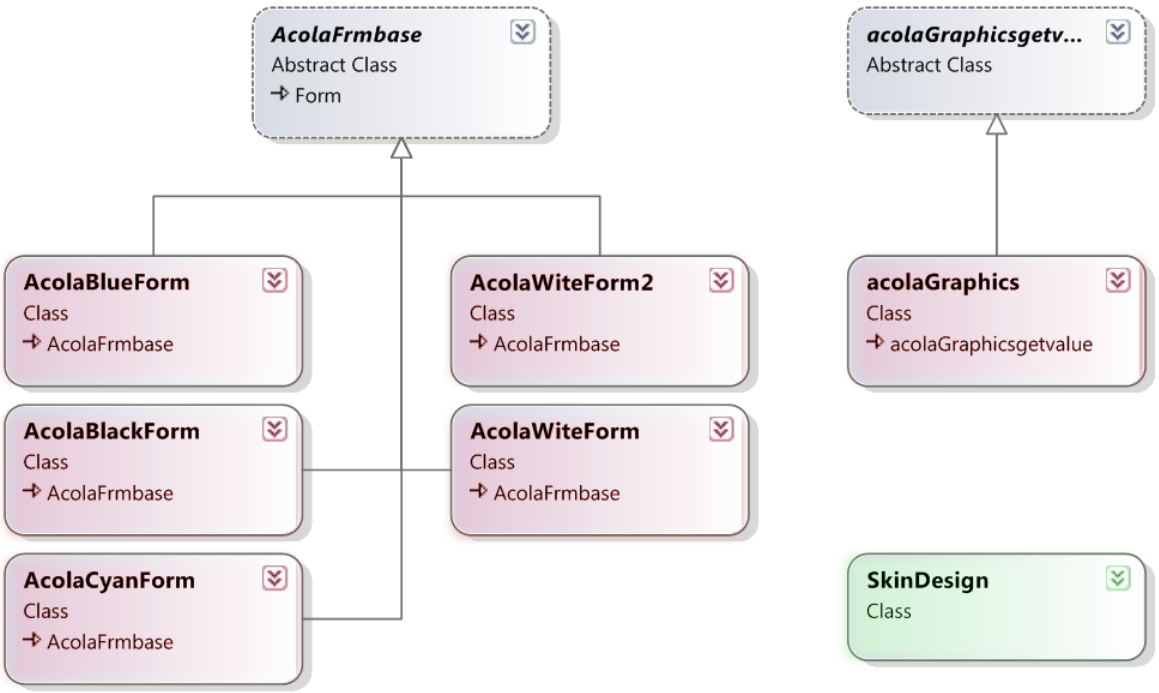
تغییر میدهیم. در حقیقت با این تغییر Form1 از کلاس BasicSkin موجود در DLL ارث میبرد که اگر به کدهای مراحل قبل نگاهی بیاندازید کلاس BasicSkin نیز از کلاس پایه Form ارث برده است . با برگشت به پنجره Form1.cs[Design] نمای فرم به صورت زیر تغییر میکند:



تصویر 20- فرم برنامه بعد از تغییر

از این روند می توان در تمام پروژه ها استفاده نمود، همانطور که گفته شد کافیست تنها DLL به Reference پروژه اضافه شده، Namespace افزوده شود و از کلاس Skin ارث برده شود.

در Skin هایی که قرار است در پروژه های صنعتی استفاده شوند قاعدتا تحلیل بیشتر و کدهای پیچیده تری خواهند داشت که از حوصله این سند خارج است، البته Skinای که در این پروژه استفاده شده است از این قاعده مستثنا نیست. در اینجا تنها به یک دیاگرام کلاس از Skin استفاده شده در پروژه اکتفا میشود:



شماره های تماس:

09186599633-09186599634

آدرس الکترونیکی:

Ir_Ebook@Yahoo.Com