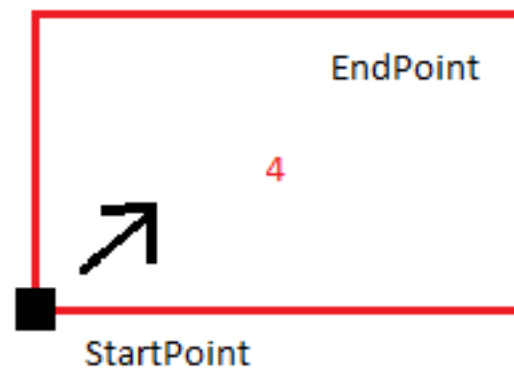
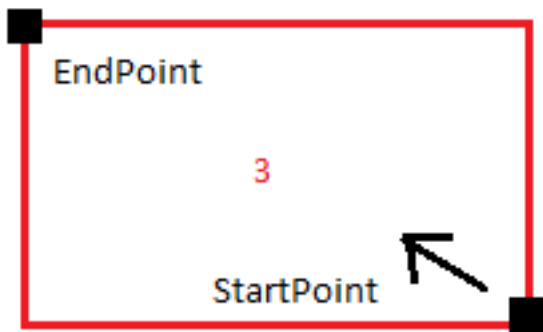
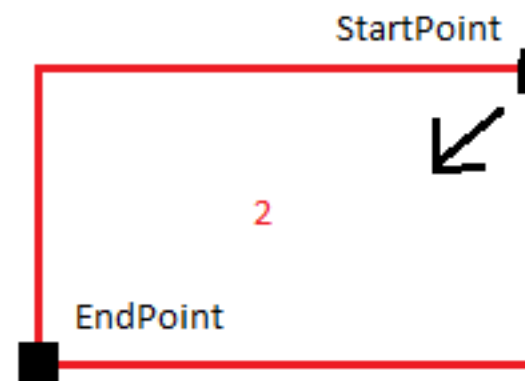
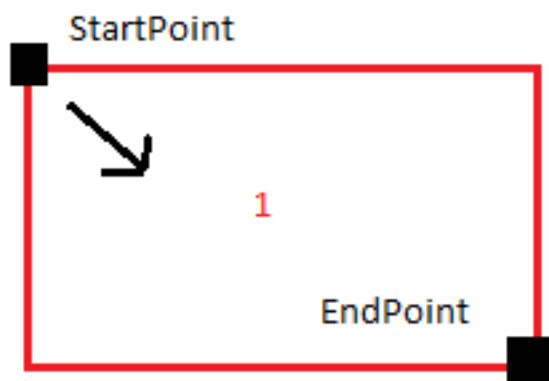


در ادامه مطالب قبل

[پایاده سازی پروژه نقاشی \(Paint\) به صورت شی گرا #1](#)

[پایاده سازی پروژه نقاشی \(Paint\) به صورت شی گرا #2](#)

قبل از شروع توضیحات متدهای کلاس Shape در ادامه پست‌های قبل در [^](#) و [^](#) ابتدا به تشریح یک تصویر می‌پردازیم.



خوب همانگونه که در تصویر بالا مشاهده می‌نمایید، برای رسم یک شی چهار حالت متفاوت ممکن است پیش بیاید. (دقت کنید که ربع اول محور مختصات روی بوم گرافیکی قرار گرفته است، در واقع گوشه بالا و سمت چپ بوم گرافیکی نقطه (0 و 0) محور مختصات است و عرض بوم گرافیکی محور Xها و ارتفاع بوم گرافیکی محور Yها را نشان می‌دهد)

در این حالت $StartPoint.X < EndPoint.X$ و $StartPoint.Y < EndPoint.Y$ خواهد بود. (StartPoint نقطه ای است که ابتدا ماوس شروع به ترسیم می‌کند، و EndPoint زمانی است که ماوس رها شده و پایان ترسیم را مشخص می‌کند).
در این حالت $StartPoint.X > EndPoint.X$ و $StartPoint.Y > EndPoint.Y$ خواهد بود.
در این حالت $StartPoint.X > EndPoint.X$ و $StartPoint.Y < EndPoint.Y$ خواهد بود.
در این حالت $StartPoint.X < EndPoint.X$ و $StartPoint.Y > EndPoint.Y$ خواهد بود.

ابتدا یک کلاس کمکی به صورت استاتیک تعریف می‌کنیم که متدی جهت پیش نمایش رسم شی در حالت جابجایی، رسم، و تغییر اندازه دارد.

```
using System;
using System.Drawing;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Helpers
    /// </summary>
    public static class Helpers
    {
        /// <summary>
        /// Draws the preview.
        /// </summary>
        /// <param name="g">The g.</param>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>
        /// <param name="isFill">if set to <c>true</c> [is fill].</param>
        /// <param name="backgroundBrush">The background brush.</param>
        /// <param name="shapeType">Type of the shape.</param>
        public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor,
byte thickness, bool isFill, Brush backgroundBrush, ShapeType shapeType)
        {
            float x = 0, y = 0;
            float width = Math.Abs(endPoint.X - startPoint.X);
            float height = Math.Abs(endPoint.Y - startPoint.Y);
            if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = startPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = endPoint.X;
                y = endPoint.Y;
            }
            else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
            {
                x = endPoint.X;
                y = startPoint.Y;
            }
            else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
            {
                x = startPoint.X;
                y = endPoint.Y;
            }

            switch (shapeType)
            {
                case ShapeType.Ellipse:
                    if (isFill)
                        g.FillEllipse(backgroundBrush, x, y, width, height);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
                    break;
                case ShapeType.Rectangle:
                    if (isFill)
                        g.FillRectangle(backgroundBrush, x, y, width, height);
                    //else
                    g.DrawRectangle(new Pen(foreColor, thickness), x, y, width, height);
                    break;
                case ShapeType.Circle:
                    float raduis = Math.Max(width, height);

                    if (isFill)
                        g.FillEllipse(backgroundBrush, x, y, raduis, raduis);
                    //else
                    g.DrawEllipse(new Pen(foreColor, thickness), x, y, raduis, raduis);
                    break;
                case ShapeType.Square:
                    float side = Math.Max(width, height);

                    if (isFill)
                        g.FillRectangle(backgroundBrush, x, y, side, side);
                    //else
                    g.DrawRectangle(new Pen(foreColor, thickness), x, y, side, side);
            }
        }
    }
}
```

```

        break;
    case ShapeType.Line:
        g.DrawLine(new Pen(foreColor, thickness), startPoint, endPoint);
        break;
    case ShapeType.Diamond:
        var points = new PointF[4];
        points[0] = new PointF(x + width / 2, y);
        points[1] = new PointF(x + width, y + height / 2);
        points[2] = new PointF(x + width / 2, y + height);
        points[3] = new PointF(x, y + height / 2);
        if (isFill)
            g.FillPolygon(backgroundBrush, points);
        //else
        g.DrawPolygon(new Pen(foreColor, thickness), points);
        break;
    case ShapeType.Triangle:
        var tPoints = new PointF[3];
        tPoints[0] = new PointF(x + width / 2, y);
        tPoints[1] = new PointF(x + width, y + height);
        tPoints[2] = new PointF(x, y + height);
        if (isFill)
            g.FillPolygon(backgroundBrush, tPoints);
        //else
        g.DrawPolygon(new Pen(foreColor, thickness), tPoints);
        break;
    }
    if (shapeType != ShapeType.Line)
    {
        g.DrawString(String.Format("{0},{1}", x, y), new Font(new FontFamily("Tahoma"), 10),
            new SolidBrush(foreColor), x - 20, y - 25);
        g.DrawString(String.Format("{0},{1}", x + width, y + height), new Font(new
            FontFamily("Tahoma"), 10), new SolidBrush(foreColor), x + width - 20, y + height + 5);
    }
    else
    {
        g.DrawString(String.Format("{0},{1}", startPoint.X, startPoint.Y), new Font(new
            FontFamily("Tahoma"), 10), new SolidBrush(foreColor), startPoint.X - 20, startPoint.Y - 25);
        g.DrawString(String.Format("{0},{1}", endPoint.X, endPoint.Y), new Font(new
            FontFamily("Tahoma"), 10), new SolidBrush(foreColor), endPoint.X - 20, endPoint.Y + 5);
    }
}
}
}
}

```

متد های این کلاس:

DrawPreview : این متد پیش نمایشی برای شی در زمان ترسیم، جابجایی و تغییر اندازه آماده می کند، پارامترهای آن عبارتند از :
 بوم گرافیکی ، نقطه شروع ، نقطه پایان و رنگ قلم ترسیم پیش نمایش شی، ضخامت خط ، آیا شی توپر باشد ؟، الگوی پر کردن
 پس زمینه شی ، و نوع شی ترسیمی می باشد.

در ادامه پست های قبل ادامه کد کلاس Shape را تشریح می کنیم.

```

using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Net;

namespace PWS.ObjectOrientedPaint.Models
{
    /// <summary>
    /// Shape (Base Class)
    /// </summary>
    public abstract partial class Shape
    {
        #region Constructors (2)

        /// <summary>
        /// Initializes a new instance of the <see cref="Shape" /> class.
        /// </summary>
        /// <param name="startPoint">The start point.</param>
        /// <param name="endPoint">The end point.</param>
        /// <param name="zIndex">Index of the z.</param>
        /// <param name="foreColor">Color of the fore.</param>
        /// <param name="thickness">The thickness.</param>

```

```

    /// <param name="isFill">if set to <c>true</c> [is fill].</param>
    /// <param name="backgroundColor">Color of the background.</param>
    protected Shape(PointF startPoint, PointF endPoint, int zIndex, Color foreColor, byte
thickness, bool isFill, Color backgroundColor)
    {
        CalulateLocationAndSize(startPoint, endPoint);
        Zindex = zIndex;
        ForeColor = foreColor;
        Thickness = thickness;
        IsFill = isFill;
        BackgroundColor = backgroundColor;
    }

    /// <summary>
    /// Initializes a new instance of the <see cref="Shape" /> class.
    /// </summary>
    protected Shape() { }

#endregion Constructors

#region Methods (10)

// Public Methods (9)

    /// <summary>
    /// Draws the specified g.
    /// </summary>
    /// <param name="g">The g.</param>
    public virtual void Draw(Graphics g)
    {
        if (!IsSelected) return;
        float diff = Thickness + 4;
        Color myColor = Color.DarkSeaGreen;
        g.DrawString(String.Format("{0},{1}", StartPoint.X, StartPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(myColor), StartPoint.X - 20, StartPoint.Y - 25);
        g.DrawString(String.Format("{0},{1}", EndPoint.X, EndPoint.Y), new Font(new
FontFamily("Tahoma"), 10), new SolidBrush(myColor), EndPoint.X - 20, EndPoint.Y + 5);
        if (ShapeType != ShapeType.Line)
        {
            g.DrawRectangle(new Pen(myColor), X, Y, Width, Height);

            // 1 2 3
            // 8 4
            // 7 6 5
            var point1 = new PointF(StartPoint.X - diff / 2, StartPoint.Y - diff / 2);
            var point2 = new PointF((StartPoint.X - diff / 2 + EndPoint.X) / 2, StartPoint.Y - diff
/ 2);
            var point3 = new PointF(EndPoint.X - diff / 2, StartPoint.Y - diff / 2);
            var point4 = new PointF(EndPoint.X - diff / 2, (EndPoint.Y + StartPoint.Y) / 2 - diff /
2);
            var point5 = new PointF(EndPoint.X - diff / 2, EndPoint.Y - diff / 2);
            var point6 = new PointF((StartPoint.X - diff / 2 + EndPoint.X) / 2, EndPoint.Y - diff /
2);
            var point7 = new PointF(StartPoint.X - diff / 2, EndPoint.Y - diff / 2);
            var point8 = new PointF(StartPoint.X - diff / 2, (EndPoint.Y + StartPoint.Y) / 2 - diff
/ 2);

            g.FillRectangle(new SolidBrush(myColor), point1.X, point1.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point2.X, point2.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point3.X, point3.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point4.X, point4.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point5.X, point5.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point6.X, point6.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point7.X, point7.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point8.X, point8.Y, diff, diff);
        }
        else
        {
            var point1 = new PointF(StartPoint.X - diff / 2, StartPoint.Y - diff / 2);
            var point2 = new PointF(EndPoint.X - diff / 2, EndPoint.Y - diff / 2);
            g.FillRectangle(new SolidBrush(myColor), point1.X, point1.Y, diff, diff);
            g.FillRectangle(new SolidBrush(myColor), point2.X, point2.Y, diff, diff);
        }
    }

    /// <summary>
    /// Points the in sahpe.
    /// </summary>
    /// <param name="point">The point.</param>
    /// <param name="tolerance">The tolerance.</param>

```

```

/// <returns>
/// <c>true</c> if [has point in sahpe] [the specified point]; otherwise, <c>false</c>.
/// </returns>
public virtual bool HasPointInSahpe(PointF point, byte tolerance = 5)
{
    return point.X > (StartPoint.X - tolerance) && point.X < (EndPoint.X + tolerance) &&
point.Y > (StartPoint.Y - tolerance) && point.Y < (EndPoint.Y + tolerance);
}

/// <summary>
/// Moves the specified location.
/// </summary>
/// <param name="location">The location.</param>
/// <returns></returns>
public virtual PointF Move(Point location)
{
    StartPoint = new PointF(location.X, location.Y);
    EndPoint = new PointF(location.X + Width, location.Y + Height);
    return StartPoint;
}

/// <summary>
/// Moves the specified dx.
/// </summary>
/// <param name="dx">The dx.</param>
/// <param name="dy">The dy.</param>
/// <returns></returns>
public virtual PointF Move(int dx, int dy)
{
    StartPoint = new PointF(StartPoint.X + dx, StartPoint.Y + dy);
    EndPoint = new PointF(EndPoint.X + dx, EndPoint.Y + dy);
    return StartPoint;
}

/// <summary>
/// Resizes the specified dx.
/// </summary>
/// <param name="dx">The dx.</param>
/// <param name="dy">The dy.</param>
/// <returns></returns>
public virtual SizeF Resize(int dx, int dy)
{
    EndPoint = new PointF(EndPoint.X + dx, EndPoint.Y + dy);
    return new SizeF(Width, Height);
}

/// <summary>
/// Resizes the specified start point.
/// </summary>
/// <param name="startPoint">The start point.</param>
/// <param name="currentPoint">The current point.</param>
public virtual void Resize(PointF startPoint, PointF currentPoint)
{
    var dx = (int)(currentPoint.X - startPoint.X);
    var dy = (int)(currentPoint.Y - startPoint.Y);
    if (startPoint.X >= X - 5 && startPoint.X <= X + 5)
    {
        StartPoint = new PointF(currentPoint.X, StartPoint.Y);
        if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
        {
            Height = Width;
        }
    }
    else if (startPoint.X >= EndPoint.X - 5 && startPoint.X <= EndPoint.X + 5)
    {
        Width += dx;
        if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
        {
            Height = Width;
        }
    }
    else if (startPoint.Y >= Y - 5 && startPoint.Y <= Y + 5)
    {
        Y = currentPoint.Y;
        if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
        {
            Width = Height;
        }
    }
    else if (startPoint.Y >= EndPoint.Y - 5 && startPoint.Y <= EndPoint.Y + 5)
    {

```

```

        Height += dy;
        if (ShapeType == ShapeType.Circle || ShapeType == ShapeType.Square)
        {
            Width = Height;
        }
    }
}

/// <summary>
/// Sets the background brush as hatch.
/// </summary>
/// <param name="hatchStyle">The hatch style.</param>
public virtual void SetBackgroundBrushAsHatch(HatchStyle hatchStyle)
{
    var brush = new HatchBrush(hatchStyle, BackgroundColor);
    BackgroundBrush = brush;
}

/// <summary>
/// Sets the background brush as linear gradient.
/// </summary>
public virtual void SetBackgroundBrushAsLinearGradient()
{
    var brush = new LinearGradientBrush(StartPoint, EndPoint, ForeColor, BackgroundColor);
    BackgroundBrush = brush;
}

/// <summary>
/// Sets the background brush as solid.
/// </summary>
public virtual void SetBackgroundBrushAsSolid()
{
    var brush = new SolidBrush(BackgroundColor);
    BackgroundBrush = brush;
}

// Private Methods (1)

/// <summary>
/// Calculates the size of the location and.
/// </summary>
/// <param name="startPoint">The start point.</param>
/// <param name="endPoint">The end point.</param>
private void CalculateLocationAndSize(PointF startPoint, PointF endPoint)
{
    float x = 0, y = 0;
    float width = Math.Abs(endPoint.X - startPoint.X);
    float height = Math.Abs(endPoint.Y - startPoint.Y);
    if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
    {
        x = startPoint.X;
        y = startPoint.Y;
    }
    else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
    {
        x = endPoint.X;
        y = endPoint.Y;
    }
    else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
    {
        x = endPoint.X;
        y = startPoint.Y;
    }
    else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
    {
        x = startPoint.X;
        y = endPoint.Y;
    }
    StartPoint = new PointF(x, y);
    EndPoint = new PointF(X + width, Y + height);
}

#endregion Methods
}
}

```

Shape : پارامترهای این سازنده به ترتیب عبارتند از **نقطه شروع** ، **نقطه پایان** ، **عمق شی** ، **رنگ قلم** ، **ضخامت خط** ، **آیا شی توپر باشد ؟** ، و **رنگ پر کردن شی** ، در این سازنده ابتدا توسط متدی به نام `CalulateLocationAndSize(startPoint, endPoint);` نقاط ابتدا و انتهای شی مورد نظر تنظیم می شود، در متد مذکور بررسی می شود در صورتی که نقاط شروع و پایان یکی از حالت های 1 ، 2 ، 3 ، 4 از تصویر ابتدا پست باشد همگی تبدیل به حالت 1 خواهد شد.

سپس به تشریح **متدهای** کلاس **Shape** می پردازیم:

Draw : این متد دارای یک پارامتر ورودی است که **بوم گرافیکی** مورد نظر می باشد، در واقع شی مورد نظر خود را بروی این بوم گرافیکی ترسیم می کند. در کلاس پایه کار این متد زیاد پیچیده نیست، در صورتی که شی در حالت انتخاب باشد (`IsSelected = true`) بروی شی مورد نظر 8 مربع کوچک ترسیم می شود و اگر شی مورد نظر خط باشد دو مربع کوچک در طرفین خط رسم می شود که نشان دهنده انتخاب شدن شی مورد نظر است. این متد به صورت `virtual` تعریف شده است یعنی کلاس هایی که از **Shape** ارث می برند می توانند این متد را برای خود از نو بازنویسی کرده (`override` کنند) و تغییر رفتار دهند.

HasPointInShape : این متد نیز به صورت `virtual` تعریف شده است دارای خروجی بولین می باشد. پارامترهای این متد عبارتند از **یک نقطه** و **یک عدد** که نشان دهنده تیرانش نقطه بر حسب پیکسل می باشد. کار این متد این است که یک نقطه را گرفته و بررسی می کند که آیا نقطه مورد نظر با تیرانس وارد شده آیا در داخل شی واقع شده است یا خیر (مثلا وجود نقطه در مستطیل یا وجود نقطه در دایره فرمول های متفاوتی دارند که در اینجا پیش فرض برای تمامی اشیا حالت مستطیل در نظر گرفته شده که می توانید آنها را بازنویسی (`override` کنید).

Move : این متد به عنوان پارامتر **یک نقطه** را گرفته و شی مورد نظر را به آن نقطه منتقل می کند در واقع نقطه شروع و پایان ترسیم شی را تغییر می دهد.

Move : این متد نیز برای جابجایی شی به کار می رود، این متد دارای پارامترهای **جابجایی در راستای محور Xها** ، **جابجایی در راستای محور Yها** ؛ و شی مورد نظر را به آن نقطه منتقل می کند در واقع نقطه شروع و پایان ترسیم شی را با توجه به پارامترهای ورودی تغییر می دهد.

Resize : این متد نیز برای تغییر اندازه شی به کار می رود، این متد دارای پارامترهای **تغییر اندازه در راستای محور Xها** ، **تغییر اندازه در راستای محور Yها** می باشد و نقطه پایان شی مورد نظر را تغییر می دهد اما نقطه شروع تغییری نمی کند.

Resize : این متد نیز برای تغییر اندازه شی به کار می رود، در زمان تغییر اندازه شی با ماوس ابتدا یک نقطه شروع وجود دارد که ماوس در آن نقطه کلیک شده و شروع به درگ کردن شی جهت تغییر اندازه می کند (پارامتر اول این متد نقطه شروع درگ کردن جهت تغییر اندازه را مشخص می کند `startPoint`)، سپس در یک نقطه ای درگ کردن تمام می شود در این نقطه باید شی تغییر اندازه پیدا کرده و ترسیم شود (پارامتر دوم این متد نقطه مذکور می باشد `currentLocation`). سپس با توجه با این دو نقطه بررسی می شود که تغییر اندازه در کدام جهت صورت گرفته است و اعداد جهت تغییرات نقاط شروع و پایان شی مورد نظر محاسبه می شوند. (مثلا تغییر اندازه در مستطیل از ضلع بالا به طرفین، یا از ضلع سمت راست به طرفین و). البته برای مربع و دایره باید کاری کنیم که طول و عرض تغییر اندازه یکسان باشد.

CalulateLocationAndSize : این متد که در سازنده کلاس استفاده شده در واقع دو نقطه شروع و پایان را گرفته و با توجه به تصویر ابتدای پست حالت های 1 و 2 و 3 و 4 را به حالت 1 تبدیل کرده و `StartPoint` و `EndPoint` را اصلاح می کند.

SetBackgroundBrushAsHatch : این متد یک الگوی `Brush` گرفته و با توجه به رنگ پس زمینه شی خصوصیت `BackgroundBrush` را مقداردهی می کند.

SetBackgroundBrushAsLinearGradient : این متد با توجه به خصوصیت `ForeColor` و `BackgroundColor` یک `Gradient Brush` ساخته و آن را به خصوصیت `BackgroundBrush` نسبت می کند.

SetBackgroundBrushAsSolid : یک الگوی پر کردن توپر برای شی مورد نظر با توجه به خصوصیت `BackgroundColor` شی ایجاد کرده و آن را به خصوصیت `BackgroundBrush` شی نسبت می دهد.

تذکر: متدهای Move, Resize و HasPointInShape به صورت virtual تعریف شده تا کلاسهای مشتق شده در صورت نیاز خود کد رفتار مورد نظر خود را override کرده یا از همین رفتار استفاده نمایند.

خوشحال می‌شوم در صورتی که در Refactoring کد نوشته شده با من همکاری کنید.

در پست‌های آینده به بررسی و پیاده سازی دیگر کلاس‌ها خواهیم پرداخت.

نظرات خوانندگان

نویسنده: بتیسا

تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۴۱

با سلام

از مطلب مفیدتون ممنونم
در متد **DrawPreview** اصلی که نوشته شده در بخش هایی که اشیاء توپر رسم می شوند مانند خط 143، 149 و... بجای استفاده از خصوصیت Brush که در بخش [قبل](#) برای پس زمینه در نظر گرفته شده بود هر بار یک براش ایجاد شده که می توانیم به صورت زیر اصلاح کنیم.

```
case ShapeType.Ellipse:
    if (isFill)
        g.FillEllipse(new SolidBrush(backgroundColor), x, y, width, height);
    //else
    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
    break;

//-----[Change to]----->

case ShapeType.Ellipse:
    if (isFill)
        g.FillEllipse(Brush, x, y, width, height);
    //else
    g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
    break;
```

نویسنده: صابر فتح الهی

تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۴۶

بله دوست گلم میشد اینکارو انجام داد
اما با توجه به اینکه متد **DrawPreview** به صورت static تعریف شده نمی توان از خصوصیات غیر استاتیک کلاس در آن استفاده کرد.
درسته؟

نویسنده: بتیسا

تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۰:۵۸

بله به static بودن متد توجه نکرده بودم

نویسنده: بتیسا

تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۱:۱۹

برای برطرف کردن این مسئله هم می توانیم همانطور که در ورودی foreColor را دریافت کردیم brush را نیز دریافت کنیم.

```
public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor, byte
thickness, bool isFill, Color backgroundColor, ShapeType shapeType)
//-----[Change to]----->
public static void DrawPreview(Graphics g, PointF startPoint, PointF endPoint, Color foreColor, byte
thickness, bool isFill, Brush backgroundBrush, ShapeType shapeType)

//-----
case ShapeType.Ellipse:
    if (isFill)
        g.FillEllipse(new SolidBrush(backgroundColor), x, y, width, height);
    //else
```

```
g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
break;

//-----[Change to]----->

case ShapeType.Ellipse:
    if (isFill)
        g.FillEllipse(backgroundBrush, x, y, width, height);
    //else
        g.DrawEllipse(new Pen(foreColor, thickness), x, y, width, height);
    break;
```

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۲:۷

درسته اما در قسمت اینترفیس کاربر باید Brush های مورد نظر ساخته شده و به این متد پاس داده شود، در مراحل پایانی فکر می‌کنم بهتر منظورم بتونم برسونم، به دلایلی (که در پست‌های آینده گفته میشه) از این روش‌ها استفاده نکردم.

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۴:۱۷

چرا drawpreview به صورت استاتیک تعریف شده؟ چرا دوبار تعریف شده؟ و چرا این کلاس پایه اطلاعات زیادی در مورد رسم زیر مجموعه‌های خودش داره؟ آیا بهتر نیست جرئیات ترسیم هر شیء با override شدن به زیر کلاس‌های مشتق شده واگذار بشن؟ چرا این متدها از خاصیت‌های کلاس تعریف شده استفاده نمی‌کنن و دوباره این خاصیت‌ها رو به صورت پارامتر دریافت کردن؟ (همون بحث اعلام استقلال متد تعریف شده به صورت استاتیک و اینکه چرا؟) و اگر این کلاس پایه تا این اندازه لازم هست در مورد رسم دایره و سایر اشکال اطلاعات داشته باشد، چه ضرورتی به abstract بودن آن هست؟ اصلا چه ضرورتی به تعریف اشیاء مشتق شده از آن هست؟

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۷:۱۱

جز متد DrawPreview هیچکدام از متدها پارامتری دریافت نمی‌کنند، اون هم به دلیل اینکه می‌خوام پیش نمایشی از یک شی ترسیم کنم البته می‌تونستیم اون توی یک کلاس جدا بنویسیم که این کلاس زیاد شلوغ نشه فکر می‌کنم با نوشتن کلاس‌های مشتق شده بهتر بشه توی این زمینه‌ها بحث کرد.

پ.ن: متد DrawPreview به یک کلاس ثالث منتقل شد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۱/۱۱/۱۸ ۱۷:۵۹

نظر شما اعمال شد

نویسنده: مسعود بهرامی
تاریخ: ۱۳۹۲/۰۹/۰۱ ۱۶:۵۸

با اجازه دوست عزیزم مهندس فتح الهی
من به نظرم Helpers رو اگه به شکل زیر Re factor کنیم بهتر باشه (:
اول یه کلاس تعریف می‌کنیم و اطلاعات لازم برای ترسیم پیش نمایش رو تو اون کلاس می‌ذاریم

```
public class ShapeSpecification
{
```

```
public PointF StartPoint{get;set;}
public PointF EndPoint{get;set;}
public Color ForeColor{get;set;}
public byte Thickness{get;set;}
public bool IsFill{get;set;}
public Brush BackgroundBrush{get;set;}
}
```

یه کلاس دیگه هم نقاط ابتدا و انتها و طول و عرض رو تو خودش داره

```
public class StartPoints
{
    public float XPoint { get; set; }
    public float YPoint { get; set; }
    public float Width { get; set; }
    public float Height { get; set; }
}
```

حالا یه اینترفیس تعریف می‌کنیم که فقط یه متد داره به نام Draw

```
public interface IPeiview
{
    void Draw(ShapeSpecification shapeScepification);
}
```

حالا می‌رسیم به کلاس Helpers اصلیمون که میتونه هم استاتیک باشه و هم معمولی به دو شکل زیر

```
public class Helpers
{
    private readonly IPeiview peiview;

    public Helpers(IPeiview peiview)
    {
        this.peiview = peiview;
    }

    public void Draw(ShapeSpecification shapeSpecification)
    {
        peiview.Draw(shapeSpecification);
    }
}
```

```
public static class Helpers
{
    public static void Draw(ShapeSpecification shapeSpecification, IPeiview peiview)
    {
        peiview.Draw(shapeSpecification);
    }
}
```

که فقط یه متد ساده Draw داره و اونم تابع Draw اینترفیسی که بش دادیم رو صدا می‌زینه
یه کلاس دیگه هم تعریف میکنیم که مسئولیتش تشخیص بوم‌های چهارگانه است برای شروع نقطه‌ی ترسیم

```
public static class AreaParser
{
    public static StartPoints Parse(PointF startPoint, PointF endPoint)
    {
        var startPoints = new StartPoints();

        startPoints.Width = Math.Abs(endPoint.X - startPoint.X);
        startPoints.Height = Math.Abs(endPoint.Y - startPoint.Y);

        if (startPoint.X <= endPoint.X && startPoint.Y <= endPoint.Y)
        {
            startPoints.XPoint = startPoint.X;
            startPoints.YPoint = startPoint.Y;
        }
    }
}
```

```

        else if (startPoint.X >= endPoint.X && startPoint.Y >= endPoint.Y)
        {
            startPoints.XPoint = endPoint.X;
            startPoints.YPoint = endPoint.Y;
        }

        else if (startPoint.X >= endPoint.X && startPoint.Y <= endPoint.Y)
        {
            startPoints.XPoint = endPoint.X;
            startPoints.YPoint = startPoint.Y;
        }

        else if (startPoint.X <= endPoint.X && startPoint.Y >= endPoint.Y)
        {
            startPoints.XPoint = startPoint.X;
            startPoints.YPoint = endPoint.Y;
        }
        return startPoints;
    }
}

```

نکته: این کلاس رو اگه با Func ایجاد کنیم خیلی بهتر و تمیزتر و قشنگتر هم میشد که من می‌گزریم فعلا ازش حالا ما هر شکل جدید که اضافه کنیم به پروژه Paint خودمون و قصد پیش نمایش اونو داشته باشیم فقط کافیه یه کلاس برا پیش نمایشش ایجاد کنیم که کلاس Ipreview رو Implement کنه و متد Draw مخصوص به خودش را داشته باشه و از شر Swith های طولانی خلاص میشیم مثلا من برای دایره اینکارو کردم

```

public class CirclePreview:IPeiview
{
    private readonly Graphics graphics;

    public CirclePreview(Graphics graphics)
    {
        this.graphics = graphics;
    }

    public void Draw(ShapeSpecification shapeScepfication)
    {
        var startPoints = AreaParser.Parse(shapeScepfication.StartPoint,
        shapeScepfication.EndPoint);

        float raduis = Math.Max(startPoints.Width, startPoints.Height);

        if (shapeScepfication.IsFill)
            this.graphics.FillEllipse(shapeScepfication.BackgroundBrush, startPoints.XPoint,
            startPoints.YPoint, raduis, raduis);
        else
            this.graphics.DrawEllipse(new Pen(shapeScepfication.ForeColor,
            shapeScepfication.Thickness), startPoints.XPoint, startPoints.YPoint, raduis, raduis);
    }
}

```

نویسنده: صابر فتح الهی

تاریخ: ۱۳۹۲/۰۹/۰۱ ۲۳:۲۷

ظاهرا همه چیز مرتبه و درست هست.
من بررسی می‌کنم و نتایجش به شما اطلاع میدم
در هر صورت از وقتی که گذاشتین متشکرم