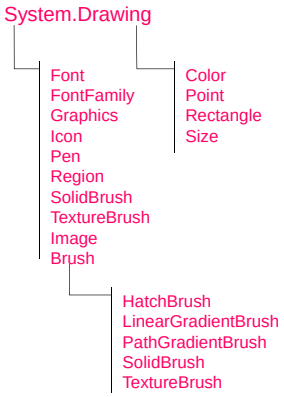
Graphics

GDI+ (Graphics Design Interface) is an application programming interface (API) for creating and manipulating :

- two-dimensional vector graphics
- fonts
- images

Disadvantage

- Some operation systems (Linux, Mac OSX) don't have GDI+



Graphical Context and Graphical Objects

1. Graphical context – represents a drawing surface for drawing on the screen
2. Graphical object – class Graphics
   - Manages a graphical context by controlling how information is drawn
   - Contains methods for
     - drawing
     - font manipulation
     - color manipulation
     - and other graphics-related actions

Creating Graphics in Windows Form Application

Every Windows application (derives from class Form) inherits an virtual OnPaint event handler of the Paint event where most graphics-related operations are performed.

When a control is painting it calls the OnPaint event handler of the Paint event.

The Control.Invalidate method refreshes a control's client area and implicitly repaints all graphical components.

1. Define an event handler
   - Automatically – overriding the OnPaint method
     protected override void OnPaint (PaintEventArgs e)
   - Writing a code
     - Add an event handler for the Paint event in the constructor
       this.Paint += new PaintEventHandler (<handler>);
     - Define an event handler
       public void <handler>(object source, PaintEventArgs e)
2. Implement the event handler
   - Extract the Graphics object from the PaintEventArgs argument
     Graphics g = e.Graphics;
   - Draw shapes and strings on the form
     <drawing methods>

Structure Color

Defines methods and constants: Orange, Pink, Cyan, Magenta, Yellow, Black, White, Gray, DarkGray, Red, Green, Blue, used to manipulate colors.

Class Font

Defines the font name, the size and the style of the font.

```
Font myFont = new Font ("Arial", 24, FontStyle.Bold);
```

Class Pen

Defines a pen.

```
Pen myPen = new Pen (Color.Red);
```

Class Brush

Defines a brush for filling the closed shapes.

```
SolidBrush myBrush = new SolidBrush (Color.Yellow);
```

Graphics methods for drawing and filling shapes and text

public void DrawString (string s, Font font, Brush brush, float x, float y);

Draws the specified text string s at the specified location (x, y) with the specified brush and font.

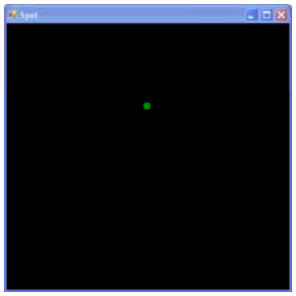
public void DrawLine (Pen pen, int x1, int y1, int x2, int y2);

Draws a line connecting the two points (x1, y1) and (x2, y2) with the specified pen.

public void FillEllipse (Brush brush, int x, int y, int width, int height);

Fills with a specified brush the interior of an ellipse defined by a bounding rectangle specified by a pair of coordinates of the upper left corner (x, y), a width, and a height.

Example: Displays a green spot where the user presses the mouse button (MouseDown form event).



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

namespace Spot
{
   public partial class Form1 : Form
   {
      private Point clickPoint = new Point (0, 0);

      private const int RADIUS = 6;

      public Form1()
      {
         InitializeComponent();
         this.Width = 450;
         this.Height = 450;
      }
```

```
      protected override void OnPaint (PaintEventArgs e)
      {
         // Extract of Graphics object
         Graphics g = e.Graphics;
         // Draw a green filled oval
         SolidBrush brush = new SolidBrush (Color.Green);
         if (clickPoint.X != 0 && clickPoint.Y != 0)
            g.FillEllipse (brush, clickPoint.X - RADIUS, clickPoint.Y - RADIUS,
                           RADIUS * 2, RADIUS * 2);
      }


      private void MousePressed(object sender, MouseEventArgs e)
      {  // The new point is determined by the location at which the mouse is
         // clicked
         clickPoint = new Point (e.X, e.Y);
         // Refresh a control's client area and implicitly repaint all graphical
         // components (call the OnPaint method)
         this.Invalidate ();
      }
   }
}
```

```
namespace Spot
{
   partial class Form1
   {   private System.ComponentModel.IContainer components = null;
       protected override void Dispose(bool disposing) {  …  }
      #region Windows Form Designer generated code
      private void InitializeComponent()
      {   this.SuspendLayout();
         // Form1
         this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
         this.BackColor = System.Drawing.Color.Black;
         this.ClientSize = new System.Drawing.Size(292, 266);
         this.Name = "Form1";
         this.Text = "Spot";
         this.MouseDown +=
          new System.Windows.Forms.MouseEventHandler(this.MousePressed);
         this.ResumeLayout(false);
      }
      #endregion
   }
}
```

Example: Displays a green stretching line using the "rubberbanding" effect (MouseDown, MouseMove and MouseUp form events). The starting point is determined by the location at which the mouse is first pressed down (MouseDown). Then as the mouse is dragged (MouseMove) the end point is continually being reset and the line is "stretched". When the button is released (MouseUp) the line is finished.

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

namespace StretchingLine
{
   public partial class Form1 : Form
   {
      private Point point1 = new Point(0, 0);          // Start point
      private Point point2 = new Point(0, 0);          // End point

      private Line line = null;

      public Form1()
      {
         InitializeComponent();
      }
```

```
      private void MousePressed(object sender, MouseEventArgs e)
      {
         point1 = new Point (e.X, e.Y);                // Start point
      }

      private void MouseDragged(object sender, MouseEventArgs e)
      {
         point2 = new Point (e.X, e.Y);                // End point
         this.Invalidate();                            // Repaint
      }

      private void MouseReleased(object sender, MouseEventArgs e)
      {
         line = new Line (point1, point2);             // Finished line
         point1.X = 0;
         point1.Y = 0;
         point2.X = 0;
         point2.Y = 0;
         this.Invalidate();                            // Repaint
      }
```

```
      private void PaintLine(object sender, PaintEventArgs e)
      {
         Graphics g = e.Graphics;
         Pen pen = new Pen (Color.Green);

         // Draw a line when the button is released
         if (line != null)
            g.DrawLine (pen, line.Point1, line.Point2);

         // Draw a line when the mouse is dragged
         if (point1.X != 0 && point1.Y != 0 && point2.X != 0 && point2.Y != 0)
            g.DrawLine (pen, point1, point2);
      }
   }
}
```

```
namespace StretchingLine
{
   partial class Form1
   {
      private System.ComponentModel.IContainer components = null;

      protected override void Dispose(bool disposing) {  …  }

      #region Windows Form Designer generated code

      private void InitializeComponent()
      {
         this.SuspendLayout();

         // Form1
         this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
         this.BackColor = System.Drawing.Color.Black;
         this.ClientSize = new System.Drawing.Size(292, 266);
         this.Name = "Form1";
         this.Text = "Stretching Line";
```

```
         this.Paint += new
             System.Windows.Forms.PaintEventHandler(this.PaintLine);
         this.MouseUp += new
            System.Windows.Forms.MouseEventHandler(this.MouseReleased);
         this.MouseMove += new
            System.Windows.Forms.MouseEventHandler(this.MouseDragged);
         this.MouseDown += new
             System.Windows.Forms.MouseEventHandler(this.MousePressed);
         this.ResumeLayout(false);
      }

      #endregion
   }
}
```

```
// Project ⇒ Add Class
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace StretchingLine
{
   class Line
   {
      private Point p1;
      public Point Point1  {  get  {  return p1;  }  }
      private Point p2;
      public Point Point2  {  get  {  return p2;  }  }
      public Line (Point p1, Point p2)
      {
         this.p1 = p1;
         this.p2 = p2;
      }
   }
}
```

Save Drawing State

Class GraphicsPath (System.Drawing.Drawing2D)

Represents a series of connected lines and curves.

public void AddLine (Point pt1, Point pt2);

Appends a line segment to this GraphicsPath.

public void StartFigure();

Starts a new figure without closing the current figure.

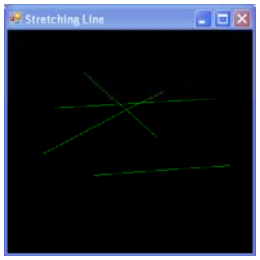
public void CloseFigure();

Closes the current figure and starts a new figure.

public void DrawPath (Pen pen, GraphicsPath path);

Draws a GraphicsPath (method of Graphics class).

Example: Save the state of drawing lines.




```
...
using System.Drawing.Drawing2D;

public class Form1 : System.Windows.Forms.Form
{  // ...
   GraphicsPath myPath = null;

   public Form1()
   {  InitializeComponent();
      myPath = new GraphicsPath();
   }

   private void MousePressed(object sender,
                             System.Windows.Forms.MouseEventArgs e)
   {  myPath.StartFigure ();
      point1 = new Point (e.X,e.Y);
   }
   private void MouseDragged(object sender,
                             System.Windows.Forms.MouseEventArgs e)
   {   point2 = new Point(e.X,e.Y);
       this.Invalidate();
   }
```

```
 private void PaintLine(object sender,
                        System.Windows.Forms.PaintEventArgs e)
{
   Graphics g = e.Graphics;
   Pen pen = new Pen (Color.Green);
   e.Graphics.DrawPath (pen, myPath);
   if (point1.X!=0 && point1.Y != 0 && point2.X != 0 && point2.Y !=0 )
         g.DrawLine (pen, point1, point2);
}
private void MouseReleased(object sender,
                           System.Windows.Forms.MouseEventArgs e)
{
   line = new Line (point1, point2);
   myPath.AddLine (line.Point1, line.Point2);
   myPath.CloseFigure ();
   point1.Y = 0;
   point2.X = 0;
   point2.Y = 0;
   this.Invalidate();
}
```