Course 2DCis: 2D-Computer Graphics with C#
Chapter C2: The Draw Project

Project draw1 with an empty window

Guidance for **Visual C# Express 2008:**
`0)` Main Menu after start of VS 2008: `Tools → Options →` check lower left checkbox: `Show all Settings → Projects and Solutions → Visual Studio projects location: → C:\temp`

`1)` Main Menu after start of VS 2008: `File -> New Project... ->`
`Visual Studio installed templates: Windows Forms Application`
`Name: draw1 -> Location: C:\temp -> Create directory for solution:` switch off `-> OK`
`Form1.cs[Design]` appears.

`2)` Two superfluous files must be deleted: `Form1.Designer.cs` and `Program.cs`.
You reach these files via the `Solution Explorer - draw1`-window:
Click the plus-sign in front of branch `draw1` and the plus-sign in front of branch `Form1.cs`.
Right-click the branch `Program.cs`. A context menu opens. Click `Delete`. A message box appears:
`'Program.cs' will be deleted permanently.` Quit with `OK`.
Right-click the branch `Form1.Designer.cs` and delete this file too.

`3)` Right-click the gray window `Form1`. A small context menu opens. Click `View Code`.
You see now the preprogrammed code of `Form1.cs`. Erase this code completely.

`4)` Write the following three lines into the empty `Form1.cs`:

```
public class Form1 : System.Windows.Forms.Form
{ static void Main() { System.Windows.Forms.Application.Run( new Form1() ); }
}
```

`5)` Click `Debug` in the main menu of VS 2008.
A submenu opens. Click `Start Without Debugging Ctrl F5`.

The rudimentary program now automatically compiles, links and starts.
Please observe the `Error List`-window of Visual Studio below our program.
Our program starts automatically as stand-alone window containing three parts:
1. main window = MainFrame with a blue title row
2. three buttons `Minimize, Maximize, Close`
3. a narrow frame with 4 movable borders and 4 movable edges.
Please enlarge the window by dragging its borders and edges.
This functionality has been inherited from the .NET-class `System.Windows.Forms.Form`.

Important: Always finish all instances of `draw1` before writing new code and starting it !
Start the Task Manager with `Ctrl+Alt+Del` and check if any `draw1.exe`-process is still running. If yes, kill it.

Minimal scribble program

If You don't see Your code anymore, click the tab **Form1.cs** of Visual Studio.
Please delete everything (including the last brace). No code remains.
Write the following lines into the empty window of Form1.cs:

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

public class Form1 : Form
{ public static void Main() { Application.Run( new Form1() ); }
  Graphics g;
  Point p0, p1;
  Brush redbrush   = new SolidBrush( Color.Red );
  Brush graybrush  = SystemBrushes.Control;
  Brush blackbrush = SystemBrushes.ControlText;
  Pen   blackpen   = SystemPens.ControlText;
  Pen   redpen     = new Pen( Color.Red );
  Pen   greenpen   = new Pen( Color.Green );

  public Form1()
  { Text = "Draw1: A Scribble Program";
    Width = 800;
    Height = 600;
    g = this.CreateGraphics();
  }
  protected override void OnMouseDown( MouseEventArgs e )
  { p0 = e.Location;
    Invalidate();
  }
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    p1 = e.Location;
    g.DrawLine( blackpen, p0, p1 );
    p0 = p1;
  }
  protected override void OnMouseUp( MouseEventArgs e )
  {
  }
  protected override void OnPaint( PaintEventArgs e )
  { g.DrawString( "Press the left mouse button and move!", Font, redbrush, 0, 0 );
  }
}
```

Click the Debug-tab of the main menu above the main window.
A sub-menu opens. Click Start Without Debugging Ctrl F5.
The program automatically compiles, links and starts again.

Draw whatever You like, change the Pen in ...OnMouseMove(...) and the Brush in ...OnPaint(...).

Important Tip 1: In case of mistype the compiler presents a Message Box: There were build errors. Continue ?.
You quit with No. A window with warnings and errors will appear in Visual Studio below Your program. In this error list scroll up to the first error (ignore the warnings !). Double click the line with the first error. The cursor jumps automatically into Your code into the line where the error was detected. Look for mistypes in this line and remove them.
(Sometimes You will not find the error in this line but above, where You forgot a comma or a semicolon.)
Ignore all errors below the first error (in most cases they are just followers of the first one) and compile.
Repeat this procedure until further error message boxes disappear and Your program compiles,
links and starts as expected.
Important Tip 2: Switch off the automatic format- and indent- mechanism of the code editor:
1. Main menu of Microsoft Visual Studio: Click "Tools".
2. A drop-down-menu appears. Click "Options...".
3. A dialog box appears. Click "Text Editor", then "C#".
4. A sub-tree appears "General, Tabs, Formatting". Click "Tabs".
5. Set Indenting to "None", Tab size and Indent size to "1" and switch on the option "Insert spaces".
6. In the sub-tree "General, Tabs, Formatting". Click "Formatting".
7. Switch on: "Leave open braces on same line as construct" but switch off all other check boxes.
8. Leave the dialog box with button "OK".

Display of coordinates

Version2: Finish `draw1`.
Insert an additional declaration into the head of `public class Form1 : Form` below the line `Pen greenpen = new Pen(Color.Green);`:

```
  String myline;
```

and insert three additional lines into the function `protected override void OnMouseMove(MouseEventArgs e)` directly below the line `p0 = p1;`, but in front of the brace, which closes method `OnMouseMove`:

```
    g.DrawString( myline, Font, graybrush , 0, Font.Height );
    myline = String.Format( "{0}, {1}", p1.X, p1.Y );
    g.DrawString( myline, Font, blackbrush, 0, Font.Height );
```

Click `Debug` in the main menu above the main window and `Start Without Debugging Ctrl F5`.

Display of vertices

Vertices is the plural of the latin word vertex = edge = computer graphics term for a point of a polygon.
Version3: Finish `draw1`.
Write an additional line in the function `protected override void OnMouseMove(MouseEventArgs e)` below the existing lines:

```
    g.DrawRectangle( blackpen, p1.X-3, p1.Y-3, 7, 7 );
```

Click `Debug` and `Start Without Debugging Ctrl F5`.
Draw speedily and slowly with the mouse and notice how the density of vertices depends on the drawing speed.
Slow computers produce less vertices than fast ones, because their operating system produces less Windows messages `WM_MouseMove` to be sent to `class Form1` activating the event handler `protected override void OnMouseMove(MouseEventArgs e)`.
Important: The program is not able to deliver a constant amount of vertices. The vertex density depends on the drawing velocity and on the hardware. When You draw slowly, a fast computer produces too much vertices with minimal distances. You can obtain funny effects when You increase the size of Your vertex squares:
e.g. `p1.X-19, p1.Y-19, 40, 40`.

Minimal distance of vertices

Version4: Finish `draw1`.
Insert three more lines into the function `protected override void OnMouseMove(MouseEventArgs e)` directly below the statement: `p1.Y = e.Y;`

```
    Int32 dx = p1.X - p0.X;
    Int32 dy = p1.Y - p0.Y;
    if ( dx*dx + dy*dy < 100 ) return;
```

Click `Debug` and `Start Without Debugging Ctrl F5`.
We derive benefit from the Pythagorean theorem: In a right triangle, the square of the length of the hypotenuse is equal to the sum of the squares of the lengths of the legs. In our case let's ignore any vertex,
if the hypotenuse length is less than 10 pixels. Draw speedy and slowly and observe the resulting vertex density. Replace the number `100` by `0, 4, 16, 64, 400, 900, 1600` etc.
see: **../../Lectures/L02_2DVector/2DVector_english.htm**

Vertex array

Version5: Our polygon gets lost, after `Form1` was resized or when `Form1` was covered by another window and it is an open polygon = polyline.
These shortfalls were recovered by 1) storing the vertices in an array of fixed length pus automatic redraw and 2) by closing the polygon automatically.
Finish `draw1`. Write the following lines into the head of `public class Form1 : Form` below the line `Graphics g;`:

```
  Int16 i, n;
  const Int16 nMax = 100;
  Point[] polygon = new Point[nMax];
```

Change function `protected override void OnMouseDown(MouseEventArgs e)` until it looks like that:

```
  protected override void OnMouseDown( MouseEventArgs e )
  { p0 = e.Location;
    polygon[0] = p0;
    n = 1;
    Invalidate();
  }
```

Change function `protected override void OnMouseMove(MouseEventArgs e)` until it looks like that:

```
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    p1 = e.Location;
    Int32 dx = p1.X - p0.X;
    Int32 dy = p1.Y - p0.Y;
    if ( dx*dx + dy*dy < 100 ) return;
    if ( n >= nMax-1 ) return;
    g.DrawLine( blackpen, p0, p1 );
    polygon[n] = p0 = p1;
    n++;
    g.DrawString( myline, Font, graybrush , 0, Font.Height );
    myline = String.Format( "{0}, {1}", p1.X, p1.Y );
    g.DrawString( myline, Font, blackbrush, 0, Font.Height );
    g.DrawRectangle( blackpen, p1.X-3, p1.Y-3, 7, 7 );
  }
```

Change `protected override void OnMouseUp(MouseEventArgs e)` until it looks like that:

```
  protected override void OnMouseUp( MouseEventArgs e )
  { if ( n < 2 ) return;
    polygon[n++] = polygon[0];
    Invalidate();
  }
```

Change `protected override void OnPaint( PaintEventArgs e )` until it looks like that:

```
  protected override void OnPaint( PaintEventArgs e )
  { g.DrawString( "Press the left mouse button and move!", Font, redbrush, 0, 0 );
    for ( int i=0; i < n-1; i++ ) g.DrawLine( blackpen, polygon[i], polygon[i+1] );
  }
```

Click `Debug` and `Start Without Debugging Ctrl F5`.
Try out the program by 1) dragging one of the window borders: make `Form1` extremely small and zoom it back and 2.) cover and uncover `Form1` by an arbitrary Windows application.
Shorten and enlarge the array by changing `nMax` to `10` and `200` and try out what happens during drawing.

Perimeter, area, center of gravity, bounding box

Version6: Finish `draw1`.
Write additional declarations into the head of `public class Form1 : Form` below the line `Point p1 = new Point();`:

```
  Point mid_of_p   = new Point();
  Point mid_of_r   = new Point();
  Rectangle minmax = new Rectangle();
  Double perimeter, area;
```

Change `protected override void OnMouseUp(MouseEventArgs e)` until it looks like that:

```
  protected override void OnMouseUp( MouseEventArgs e )
  { if ( n < 2 ) return;
    p0 = polygon[n++] = polygon[0];
    perimeter = area = 0;
    mid_of_p.X = mid_of_p.Y = 0;
    Int32 xmin, xmax, ymin, ymax;
    xmin = xmax = p0.X;
    ymin = ymax = p0.Y;
    for ( i=1; i < n; i++ )
    { p1 = polygon[i];
      Double dx = p1.X - p0.X;
      Double dy = p1.Y - p0.Y;
      Double my = (p0.Y + p1.Y) / 2.0;
      perimeter  += Math.Sqrt( dx*dx + dy*dy );
      area       += dx * my;
      mid_of_p.X += p1.X;
      mid_of_p.Y += p1.Y;
      if ( p1.X < xmin ) xmin = p1.X;
      if ( p1.X > xmax ) xmax = p1.X;
      if ( p1.Y < ymin ) ymin = p1.Y;
      if ( p1.Y > ymax ) ymax = p1.Y;
      p0 = p1;
    }
    mid_of_r.X = ( xmax + xmin ) / 2;
    mid_of_r.Y = ( ymax + ymin ) / 2;
    mid_of_p.X /= n-1;
    mid_of_p.Y /= n-1;
    minmax.X = xmin-1; minmax.Width  = xmax - xmin + 2;
    minmax.Y = ymin-1; minmax.Height = ymax - ymin + 2;
    Invalidate();
  }
```

Change `protected override void OnPaint( PaintEventArgs e )` until it looks like that:

```
  protected override void OnPaint( PaintEventArgs e )
  { g.DrawString( "Press the left mouse button and move!", Font, redbrush, 0, 0 );
    if ( n < 2 ) return;
    myline = String.Format( "Perimeter= {0}, Area= {1}", (Int32)perimeter, (Int32)area );
    g.DrawString( myline, Font, blackbrush, 0, 2*Font.Height );
    for ( i=0; i < n-1; i++ ) g.DrawLine( blackpen, polygon[i], polygon[i+1] );
    for ( i=0; i < n-3; i+=3 )
      g.DrawBezier( redpen, polygon[i], polygon[i+1], polygon[i+2], polygon[i+3] );
    g.DrawRectangle( greenpen, minmax );
    g.DrawLine( greenpen, mid_of_r.X - 4, mid_of_r.Y    , mid_of_r.X + 4, mid_of_r.Y     );
    g.DrawLine( greenpen, mid_of_r.X    , mid_of_r.Y - 4, mid_of_r.X    , mid_of_r.Y + 4 );
    g.FillEllipse( blackbrush, mid_of_p.X-5, mid_of_p.Y-5, 11, 11 );
  }
```

Click `Debug` and `Start Without Debugging Ctrl F5`.
Try out the program and investigate the coherence between the code of `OnMouseUp / OnPaint` and the behaviour of the program. Notice the sign of `Area` when you draw clockwise and counter clockwise.

Exercises

Click `Help` in the main menu of Visual Studio. Click the sub-menu `Index`.
Go to `Filtered by:` and choose: `.NET Framework SDK`. Then enter into `Look for:` one of the following key words: `Rectangle, Point, Array, MouseEventArgs, PaintEventArgs, Math` etc.
You obtain a selection of key words beginning with those characters. Read the help texts.
The help window covers Your code. When you finished reading, get rid of it with the X-button in the upper right window corner. Finish Visual Studio, start the Explorer, delete the complete directory `C:\temp\draw1`.
Start Visual Studio again and try to write this program by heart.
Writing the code use Drag&Drop = move and copy using the mouse with and without the `Strg`-key.
Invent and try out new variants (in form of new projects `draw2, draw3` etc.)

Mini programs for training

1. MiniDraw without storing and redraw after destruction

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ public static void Main() { Application.Run( new Form1() ); }
  Graphics g;
  Point p0, p1;
  Pen redpen = new Pen( Color.Red, 5 );
  protected override void OnMouseDown( MouseEventArgs e )
  { g = this.CreateGraphics();
    p0 = e.Location;
  }
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    p1 = e. Location;
    g.DrawLine( redpen                , p0, p1 );
    g.DrawLine( SystemPens.ControlText, p0, p1 );
    p0 = p1;
  }
}
```

2. MiniDrawArray with fixed array and `OnPaint`-Event Handler for redraw after destruction

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class Form1 : Form
{ public static void Main() { Application.Run( new Form1() ); }
  Graphics g;
  const Int16 N = 100;
  Point[] p = new Point[N];
  Int16 n;
  protected override void OnMouseDown( MouseEventArgs e )
  { g = this.CreateGraphics();
    p[0] = e. Location;
    n = 1;
    Invalidate();
  }
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    if ( n >= N ) return;
    p[n].X = e. Location;
    g.DrawLine( SystemPens.ControlText, p[n-1], p[n] );
    n++;
  }
  protected override void OnPaint( PaintEventArgs e )
  { for ( Int16 i = 1; i < n; i++ )
      g.DrawLine( SystemPens.ControlText, p[i-1], p[i] );
  }
}
```

3. MiniDrawDynArray with dynamic array

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections;
public class Form1 : Form
{ public static void Main() { Application.Run( new Form1() ); }
  Graphics g;
  Point p0;
  ArrayList p = new ArrayList();
  protected override void OnMouseDown( MouseEventArgs e )
  { g = this.CreateGraphics();
    p0 = e. Location;
    p.Clear(); p.Add( p0 );
    Invalidate();
  }
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    p0 = e. Location;
    g.DrawLine( SystemPens.ControlText, (Point)p[p.Count-1], p0 );
    p.Add( p0 );
    g.FillRectangle( SystemBrushes.Control, 0, 0, 200, 15 );
    String s = "length = " + p.Count.ToString();
    g.DrawString( s, new Font( "Arial", 12 ), SystemBrushes.ControlText, 0, 0 );
  }
  protected override void OnPaint( PaintEventArgs e )
  { for ( Int16 i = 1; i < p.Count; i++ )
      g.DrawLine( SystemPens.ControlText, (Point)p[i-1], (Point)p[i] );
  }
}
```