

# Course 2DCis: 2D-Computer Graphics with C#

## Chapter C1: Comments to the Intro Project

Copyright © by V. Miszalok, last update: 04-01-2006

### using namespaces

//The .NET Framework Class Library FCL contains thousands of classes. For better orientation it is subdivided into "namespaces" each containing a subset of related classes. Any class and its members have to be called by writing the full tree of its namespace which forces the programmer to write spaghetti-long identifiers. With the "using" directive You can shorten such long identifiers and the compiler will complete the missing namespaces for You.

---

```
using System; //Home of the base class of all classes "System.Object" and of all primitive data types such as Int32, Int16, double, string.
```

---

```
using System.Drawing; //Home of the "Graphics" class and its drawing methods such as DrawString, DrawLine, DrawRectangle, FillClosedCurve etc.
```

---

```
using System.Windows.Forms; //Home of the "Form" class (base class of our main window Form1) and its method Application.Run.
```

---

### Entry to start our .NET Windows program: public class Form1 : Form

//We derive our window Form1 from the class Form, which the compiler automatically finds in the System.Windows.Forms namespace.

---

```
static void Main() { Application.Run( new Form1() ); } //Create an instance of Form1 and ask the operating system to start it as main window of our program.
```

---

```
Font arial18 = new Font( "Arial", 18 ); //Create a Font object once that can be used at any call of ...OnPaint(...).
```

---

```
Font arial16 = new Font( "Arial", 16 ); //Create a Font object once that can be used at any call of ...OnPaint(...).
```

---

```
Font courier14 = new Font( "Courier New", 14 ); //Create a Font object once that can be used at any call of ...OnPaint(...).
```

---

```
Brush blackbrush = SystemBrushes.ControlText; //Create a pointer to an already existing Brush object (just a programming shortcut).
```

---

```
Brush redbrush = new SolidBrush( Color.Red ); //Create a new Brush object once that can be used at any call of ...OnPaint(...).
```

---

```
Brush whitebrush = new SolidBrush( Color.White ); //Create a new Brush object once that can be used at any call of ...OnPaint(...).
```

---

```
Pen blackpen = new Pen( Color.Black, 4 ); //Create a Pen object (thick black pen).
```

---

```
Pen randompen = new Pen( Color.Black, 20 ); //Create a Pen object (thick pen with provisional black color. Version 5 will give it a final random color).
```

---

**Constructor** `public Form1()` inside `public class Form1`


---

`BackColor = Color.White;` //Background color of `Form1`. This statement is not obligatory but the default gray is rather ugly.

---

`Text = "Intro1";` //String in the top blue line of `Form1`. This statement is not obligatory.

---

`SetStyle(ControlStyles.ResizeRedraw, true);` //Urges `Form1` to redraw itself at run time by calling protected override `void OnPaint(PaintEventArgs e)` on any change of size (i.e. when the user drags a window border or edge).

---

`StartPosition = FormStartPosition.Manual;` //Allows to set the initial position of the left upper corner of `Form1`. It allows to overwrite `WindowsDefaultLocation` by setting the `Top` and `Left` properties at run time.

---

`Top = 50;` //y-coordinate of the initial upper left corner of `Form1`. This statement does not work without `FormStartPosition.Manual`.

---

`Left = 50;` //x-coordinate of the initial upper left corner of `Form1`. This statement does not work without `FormStartPosition.Manual`.

---

`Width = 50;` //Horizontal extent of `Form1`. This statement always works.

---

`Height = 600;` //Vertical extent of `Form1`. This statement always works.

---

**Overridden event handler** `OnPaint(PaintEventArgs e)` inside `public class Form1`


---

`//Version 2 ***** //Comment`

---

`Graphics g = e.Graphics;` //The current Device Context is loaded (i.e. whenever the user drags a window border or edge).

---

`Rectangle cr = ClientRectangle;` //Ask for the current dimensions of `Form1`. Store the answer into a **structure** `Rectangle` named `cr`. See: [Help->Index->Look for: Rectangle structure](#), all members->Filtered by: .NET Framework SDK.

---

`String s0 = "Hello world, here is Intro1 !";` //Prepare a text. See: [Help->Index->Look for: String class](#), all members->Filtered by: .NET Framework SDK.

---

`String s1 = "Change the size of your window by dragging a corner !";` //Prepare a text. See: [Help->Index->Look for: String class](#), all members->Filtered by: .NET Framework SDK.

---

`String s2w = "Form : Width = " + Width.ToString();` //Prepare a text by concatenating two sub-strings (concatenation operator is "+"). The first sub-string is a literal and the second is a string-converted integer value. See: [Help->Index->Look for: String class](#), all members->Filtered by: .NET Framework SDK.

---

`String s3w = "Client: Width = " + cr.Width.ToString();` //Prepare a text by concatenating two sub-strings (concatenation operator is "+"). The first sub-string is a literal and the second is a string-converted integer value.

---

`String s2h = " Height= " + Height.ToString();` //Prepare a text by concatenating two sub-strings (concatenation operator is "+"). The first sub-string is a literal and the second is a string-converted integer value.

---

`String s3h = " Height= " + cr.Height.ToString();` //Prepare a text by concatenating two sub-strings (concatenation operator is "+"). The first sub-string is a literal and the second is a string-converted integer value.

---

`g.DrawString( s0 , arial18 , blackbrush, 0, 0 );` //Display "Hello world, here is Intro1 !" in the upper left corner.

---

---

```
g.DrawString( s1 , arial16 , redbrush , 0, 20 ); //Display "Change the size of your
window by dragging a corner !" 20 pixel below the upper left corner.
```

---

```
g.DrawString( s2w + s2h, courier14, blackbrush, 0, 40 ); //Display the outer dimensions of
Form1 40 pixel below the upper left corner.
```

---

```
g.DrawString( s3w + s3h, courier14, blackbrush, 0, 60 ); //Display the inner dimensions of
Form1 (called the "client area" of Form1) 60 pixel below the upper left corner.
```

---



---

```
//Version 3 ***** //Comment
```

---

```
Point mid = new Point( cr.Width/2, cr.Height/2 ); //A point in the middle of Form1. See: Help-
>Index->Look for: Point structure, all members->Filtered by: .NET Framework SDK.
```

---

```
g.DrawString( "left" , arial16, blackbrush, 0 , mid.Y ); //Displays the word "left" at the
mid of the left border of Form1.
```

---

```
g.DrawString( "right" , arial16, blackbrush, cr.Width-50, mid.Y ); //Displays the word
"right" 50 pixel to the left of the mid of the right border of Form1.
```

---

```
g.DrawString( "top" , arial16, blackbrush, mid.X , 0 ); //Displays the word "top" at the mid
of the upper border of Form1.
```

---

```
g.DrawString( "bottom", arial16, blackbrush, mid.X , cr.Height-30 ); //Displays the word
"bottom" 30 pixel above the mid of the lower border of Form1.
```

---



---

```
//Version 4 ***** //Comment
```

---

```
g.DrawLine( blackpen, 0, 0, cr.Width, cr.Height ); //A line from the left upper corner to the right
lower corner.
```

---

```
g.DrawLine( blackpen, cr.Width, 0, 0, cr.Height ); //A line from the right upper corner to the
left lower corner.
```

---

```
Int32 w5 = cr.Width / 5; //Width of the empty space at the left and the right side of a central rectangle
(see below). We write Int32 (FCL object representing an integer) instead of int (C# integer) for didactic
reasons. It trains You for thinking in terms of the .NET Framework Class Library FCL (which are identical in all
.NET languages) rather than in terms of one special language like C#.
```

---

```
Int32 h5 = cr.Height / 5; //Height of the empty space above and below of a central rectangle.
```

---

```
g.FillRectangle( whitebrush, w5, h5, 3 * w5, 3 * h5 ); //Clear everything inside the central
rectangle.
```

---

```
g.DrawRectangle( blackpen , w5, h5, 3 * w5, 3 * h5 ); //Draw the central rectangle.
```

---

```
g.DrawEllipse ( blackpen , w5, h5, 3 * w5, 3 * h5 ); //Draw an ellipse inside the central
rectangle..
```

---

```
//Version 5 ***** //Comment
```

---

```
Int16 i, nn = 120; //The no. of radial lines pointing from the center of the ellipse to the periphery (see below). We write Int16 (FCL object representing a short integer) instead of short (C# short integer) for didactic reasons. It trains You for thinking in terms of the .NET Framework Class Library FCL (which are identical in all .NET languages) rather than in terms of one special language like C#.
```

---

```
Int32 red, green, blue; //Three integers carrying RGB-color values between 0 and 255.
```

---

```
mypen.EndCap = System.Drawing.Drawing2D.LineCap.DiamondAnchor; //Attach a diamond form at the outer end of the line. See: Help->Index->Look for: LineCap enumeration->Filtered by: .NET Framework SDK. Try out other line caps instead of DiamondAnchor such as Flat, Round, Triangle.
```

---

```
Point[] splash = new Point[nn]; //Definition of an array of nn vertices.
```

---

```
Double arcus_1 = 2.0 * Math.PI / nn; //Arc of 360 degrees divided by nn = arc of 3 degrees.
```

---

```
Double arcus_i, factor, sinus, cosinus; //4 local variables
```

---

```
Double radius_x = 1.35 * w5; //Maximal horizontal length of a radial line. (The lines should not cross over the border of the ellipse.)
```

---

```
Double radius_y = 1.35 * h5; //Maximal vertical length of a radial line. (The lines should not cross over the border of the ellipse.)
```

---

```
Random random = new Random(); //Instantiate a class that generates random numbers. See: Help->Index->Look for: Random class, about Random class->Filtered by: .NET Framework SDK.
```

---

```
for ( i = 0; i < nn; i++ ) //Run through all radial lines.
```

---

```
red = random.Next( Byte.MaxValue ); //Fill the red color integer with a random value between 0 and 255. See: Help->Index->Look for: Random class, all members->Filtered by: .NET Framework SDK.
```

---

```
green = random.Next( Byte.MaxValue ); //Fill the green color integer with a random value between 0 and 255
```

---

```
blue = random.Next( Byte.MaxValue ); //Fill the blue color integer with a random value between 0 and 255
```

---

```
randompen.Color = Color.FromArgb( red, green, blue ); //Change the color of the pen.
```

---

```
factor = Math.Max( 0.25, random.NextDouble() ); //random.NextDouble() creates a random no between 0.0 and 1.0. When it is less than 0.25, the function Math.Max will take 0.25 instead of the random number (otherwise the line will be too short and its next neighbour will probably cover it completely). Math.Max guarantees a factor of 0.25 at least.
```

---

```
arcus_i = arcus_1 * i; //Compute the next angle as multiple of arcus_1 (= multiple of the arc of 3 degrees)
```

---

```
cosinus = radius_x * factor * Math.Cos( arcus_i ); //arcus_i is the angle of a radial line from the center of ellipse. The length of the line is random but its end point is always inside the ellipse. cosinus is the x-coordinate of this end point.
```

---

```
sinus = radius_y * factor * Math.Sin( arcus_i ); //arcus_i is the angle of a radial line from the center of ellipse. The length of the line is random but its end point is always inside the ellipse. sinus is the y-coordinate of this end point.
```

---

```
g.DrawLine( mypen, mid.X, mid.Y, mid.X + (Int32)cosinus, mid.Y + (Int32)sinus ); //Draw the line starting at the center of Form1 = the center of the ellipse.
```

---

```
splash[i].X = mid.X + (Int32)( cosinus * 0.8 ); //Shorten the length of the line to 80% and  
store the x-coordinate of its new (nearer to the center) end point to the array splash.
```

---

```
splash[i].Y = mid.Y + (Int32)( sinus * 0.8 ); //Shorten the length of the line to 80% and store  
the y-coordinate of its new (nearer to the center) end point to the array splash.
```

---

```
//Version 6 ***** //Comment
```

---

```
g.FillClosedCurve( redbrush, splash ); //Fill the interior of the circularly jagged polygon with red  
color. See: Help->Index->Look for: Graphics class, methods->Filtered by: .NET  
Framework SDK.
```

---

```
g.DrawString( "splash !", arial18, whitebrush, mid.X - 40, mid.Y - 9 ); //Center the  
word "splash !" in big white letters in the middle of the jagged polygon.
```

---

```
//Version 7 ***** //Comment
```

---

```
System.Threading.Thread.Sleep( 100 ); //Stop the execution of the program for 100 milliseconds.  
(Otherwise it will make You nervous by heavy flickering.)
```

---

```
Invalidate(); //Ask the operating system to raise the Paint event again. When there are no other high  
priority tasks the operating system will send a Paint-Message to our Form1 which calls the protected  
override void OnPaint( PaintEventArgs e )-event-handler-function again. Calling Invalidate()  
from inside the OnPaint-event handler results in a sort of recursive loop which infinitely drives the animation.  
That's a primitive method to produce (flickering) movements. In further chapters better methods with a Timer-  
object and with double buffering will be proposed.
```

---