Course 2DCis: 2D-Computer Graphics with C#
Chapter C3: Comments to the XML Project

`using` namespaces

`using System;` //Namespace of the base class of all classes "`System.Object`" and of all primitive data types such as `Int32, Int16, Double, String`.

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

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

`using System.Collections;` //Namespace of the "`ArrayList`" class
(see below the dynamic array `polygon`).

`using System.IO;` //Namespace for the `StreamReader`- and `StreamWriter`- classes (see below inside the `MenuFileSaveAsTXT`, `MenuFileSaveAsXAML`, `MenuFileSaveAsSVG` and `MenuFileRead` functions).

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

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

`Graphics g;` //Class to store and access the current Device Context.

`Point p0 = new Point();` //Coordinates of the starting point of a line.
`Point p1 = new Point();` //Coordinates of the end point of a line
(which serves as starting point of the next line).

`ArrayList polygon = new ArrayList();` //This dynamic array of variable length is aimed to contain all vertices (vertices are `Point`-Objects: see below in the functions `OnMouseDown` and `OnMouseMove`).

`System.Text.StringBuilder polygon_string = new System.Text.StringBuilder();`
//`polygon_string` will carry all x/y-coordinates separated by commas.
The `StringBuilder` class is much more effective concatenating strings than the `String` class.
Example: If you declare `String mystring = "hello";` and concatenate a literal like this:
`mystring += " world";` then concatenating is a time consuming process:
1) A new instance of `mystring` is created.
2) `"hello"` of the old instance is copied into the new instance.
3) `" world"` is attached.
Declaring `System.Text.StringBuilder mystring = new System.Text.StringBuilder("hello");` avoids such an overhead.
The method `mystring.Append(" world");` will execute much faster because it does not create any new instance of `mystring`. Consequence: Whenever You have to build strings by lining up substrings it is better to use the `System.Text.StringBuilder` class than the `String` class.

`Brush redbrush = new SolidBrush(Color.Red);` //Create a red `Brush`.

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

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

`Pen blackpen = SystemPens.ControlText;`
//Create a pointer to an already existing `Pen` object (just a programming shortcut).

`Pen redpen = new Pen(Color.Red, 4);` //Create a thick red `Pen`.

Constructor `public Form1()`

`Text = "XML1: Store & Read Polygones in XML-Format";` //Title in the blue header line of `Form1`.

`MenuItem miSaveTXT = new MenuItem("SaveAsTXT", new EventHandler(MenuFileSaveAsTXT) );` //First submenu line connects to the function `MenuFileSaveAsTXT` (see below).

`MenuItem miSaveXAML = new MenuItem("SaveAsXAML", new EventHandler(MenuFileSaveAsXAML) );`
//Second submenu line connects to the function `MenuFileSaveAsXAML` (see below).

`MenuItem miSaveSVG = new MenuItem("SaveAsSVG", new EventHandler(MenuFileSaveAsSVG) );`
//Third submenu line connects to the function `MenuFileSaveAsSVG` (see below).

`MenuItem miRead = new MenuItem("Read.....", new EventHandler(MenuFileRead) );`
//Fourth submenu line connects to the function `MenuFileRead` (see below).

`MenuItem miExit = new MenuItem("Exit.....", new EventHandler(MenuFileExit) );`
//Fifth submenu line connects to the function `MenuFileExit` (see below).

`MenuItem miFile = new MenuItem("File.....", new MenuItem[] {miSaveTXT, miSaveXAML, miSaveSVG, miRead, miExit} );` //Menu `File` is defined as array containing 4 submenu items.

`Menu = new System.Windows.Forms.MainMenu( new MenuItem[] {miFile} );`
//An instance of menu `File` is attached to `Form1` as its Main menu.

`Width = 800;` //Starting width of `Form1`.
This statement is not obligatory but the default width is rather narrow for drawing.

`Height = 600;` //Starting height of `Form1`.
This statement is not obligatory but the default height is rather narrow for drawing.

`g = this.CreateGraphics();` //The current Device Context is loaded once and will be maintained for the rest of the life time of `Form1`. This is the simplest method to get a Device Context once and forever,
but this method will ignore any change of the window size during run time. Professional code must reload the Device Context much more often i.e. at any `OnMouseMove` and `OnPaint` events in order to keep in pace with possible changes of `Form1` during run time.

Overridden event handler `OnMouseDown(MouseEventArgs e)`

`polygon.Clear();` //Delete all former vertices from the dynamic array `polygon`. Its length is now 0.

`Invalidate();` //Ask the operating system to send a `Paint`-Message to `Form1`. After receiving this message `Form1` will execute our method (described below): `protected override void OnPaint( PaintEventArgs e )` which redraws the complete polygon.
The current effect is just to erase the client area of `Form1` because the currently empty array `polygon` is displayed. This action removes the old polygon from the screen.

`p0.X = e.X;` //Remember the current mouse position.

`p0.Y = e.Y;` //Remember the current mouse position.

`polygon.Add( p0 );` //Write the first vertex at position 0 into the dynamic array `polygon`.

Overridden event handler `OnMouseMove(MouseEventArgs e)`

`if ( e.Button == MouseButtons.None ) return;`
//Do nothing if no mouse button has been pressed during moving.

`p1.X = e.X;` //Take the current mouse position.

`p1.Y = e.Y;` //Take the current mouse position.

`Int32 dx = p1.X - p0.X;` //`dx` is the horizontal distance between the new point and the last vertex.

`Int32 dy = p1.Y - p0.Y;` //`dy` is the vertical distance between the new point and the last vertex.

`if ( dx*dx + dy*dy < 100 ) return;`
//Pythagoras' theorem. If the distance is less then `10`, ignore the new point.

`g.DrawLine( redpen, p0, p1 );` //Draw a line from the old vertex to the current vertex..

`polygon.Add( p1 );` //Elongate the dynamic array `polygon` by adding space for an object of type `Point` and store the current vertex into this empty space.

`p0 = p1;` //Replace the old point by the new one .

Overridden event handler `OnMouseUp(MouseEventArgs e)`

`if ( polygon.Count < 2 ) return;` //Do nothing if there is one vertex only inside the polygon.

`polygon_string.Length = 0;`
//Remove any former content (=clear all data inside the `StringBuilder` object).

`polygon_string.Append ("\"");` //The string of all coordinates must start with a quotation mark.
The notation of a quotation mark string in C, C++ and C# is rather confusing:
The quotation mark must be preceded by an ESCAPE character (= back slash) and enclosed by
2 quotation marks adding up to 4 characters: `"\""` .

`for ( Int32 i=0; i < polygon.Count; i++ )`
//Go through all vertices which a currently stored in the dynamic array `polygon`.

`p0 = (Point)polygon[i];`
//Take object no. `i` from the dynamic array `polygon`, cast this object to type `Point` and copy it to `p`.

`polygon_string.Append( p0.X );` //Extract the X member from `p`, convert this integer to a string and put it at the end of the existing `polygon_string`.

`polygon_string.Append( "," );` //Put a comma separator between any x and its corresponding y.

`polygon_string.Append( p0.Y );` //Extract the Y member from `p`, convert this integer to a string and put it at the end of the existing `polygon_string`..

`if ( i < polygon.Count-1 ) polygon_string.Append( ", " );`
//Put a comma and a blank behind any y but not behind the last one.

`polygon_string.Append( "\"" );` //Put a quotation mark behind the last y that closes the leading quotation mark in front of the first x (see above the second line of the function).

`Invalidate();` //Ask the operating system to send a `Paint`-Message to `Form1`.
After receiving this message `Form1` will execute our method (described below):
`protected override void OnPaint( PaintEventArgs e )` which redraws the complete polygon.
(This statement is not obligatory here, but sometimes the polygon looks better when it is completely redrawn.)

Overridden event handler `OnPaint(PaintEventArgs e)`

`g.FillRectangle( graybrush, 0, 0, Width, 2*Font.Height );` //Draw an invisible rectangle in order to erase any former text `"Press the left mouse button and move!"` from the first line of `Form1`. This is necessary if somebody resizes the width of Form1 and the text must be moved to start at another horizontal position.

`g.DrawString("Press the left mouse button and move!", Font, redbrush, Width/2-50, 0 );`
//Display the text somewhere in the middle of the first line of `Form1`.

`g.DrawString(polygon_string.ToString(), Font, blackbrush, 0, Font.Height );`
//Display `polygon_string` (=all x/y converted to strings, separated by commas, enclosed by quotation marks) at the second line of `Form1`. The string is truncated if there are too much x/y to fit into one line (fixed line length = 800 pixel, defined by `Width = 800;` see above constructor `Form1();`)).

`for ( Int32 i=0; i < polygon.Count-1; i++ ) g.DrawLine( redpen, (Point)polygon[i], (Point)polygon[i+1] );` //Take all objects out of the dynamic array `polygon`, cast these objects to type `Point` and draw all lines between vertex no. 0 and the last vertex.
Please take notice: Index i runs from `0` to the last but one because we draw from index `i` to index `i+1`.

Menu driven function `void MenuFileSaveAsTXT( object obj, EventArgs ea )`

`SaveFileDialog dlg = new SaveFileDialog();`
//Open the standard user dialog to choose a directory and a file name.

`dlg.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*" ;` //The user can choose between 2 file types : 1) `*.txt`-files (=default file extension), and 2) all files `*.*` (=second choice).

`if ( dlg.ShowDialog() != DialogResult.OK ) return;` //Forget everything, if the user does not quit the standard `OpenFileDialog` box by clicking its Open-Button or if something else goes wrong.

`Int32 n0 = dlg.FileName.IndexOf( ".txt" );`
//Seek the extension "`.txt`" in the user's file name. If `n0` returns as `-1`, the user forgot the extension.

`if ( n0 < 0 ) dlg.FileName += ".txt";`
//If the user forgot the extension, append it automatically to the file name.

`StreamWriter sw = new StreamWriter( dlg.FileName );`
//Create (or open if it already exists) the selected file as an ordinary plain text file.

`sw.WriteLine( "This file comes from Prof. Miszalok's XML1.exe" );`
//Write a title line. This line is not important.

`sw.WriteLine( "polyline points " );` //Write the key words `polyline` and `points`.

`sw.WriteLine( polygon_string );` //Write the preformatted string containing all vertices.
(separated by commas and enclosed between 2 quotation marks
see `protected override void OnMouseUp(MouseEventArgs e)`

`sw.Close();` //Close the file.

Menu driven function `void MenuFileSaveAsXAML( object obj, EventArgs ea )`

`SaveFileDialog dlg = new SaveFileDialog();`
//Open the standard user dialog to choose a directory and a file name.

`dlg.Filter = "XAML files (*.xaml)|*.xaml|All files (*.*)|*.*" ;` //The user can choose between 2 file types : 1) `*.xaml`-files (=default file extension), 2) all files `*.*` (=second choice).

`if ( dlg.ShowDialog() != DialogResult.OK ) return;` //Forget everything, if the user does not quit the standard `SaveFileDialog` box by clicking its `Open`-button or if something else goes wrong.

`Int32 n0 = dlg.FileName.IndexOf( ".xaml" );`
//Seek the extension "`.xaml`" in the user's file name. If `n0` returns as `-1`, the user forgot the extension.

`if ( n0 < 0 ) dlg.FileName += ".xaml";`
//If the user forgot the extension, append it automatically to the file name.

`StreamWriter sw = new StreamWriter( dlg.FileName );`
//Create (or open if it already exists) the selected file as an ordinary plain text file.

```
sw.WriteLine( "<Canvas xmlns
=\"http://schemas.microsoft.com/winfx/2006/xaml/presentation\"" );
```
//Write the obligatory first line of a XAML-file.

`sw.WriteLine( "xmlns:x=\"http://schemas.microsoft.com/winfx/2006/xaml\">" );`
//Write the obligatory second line of a XAML-file.

`sw.WriteLine( "<!--This file comes from Prof. Miszalok's XML1.exe.-->" );`
//Write a comment. This line is not important.

`sw.WriteLine( "<Polyline Points = " );`
//Leading information of the XAML-`Polyline` tag indicating that vertices will follow immediately.

`sw.WriteLine( polygon_string + " Stroke=\"Red\" StrokeThickness=\"4\" />" );`
//Write the preformatted string containing all vertices. (separated by commas and enclosed between 2 quotation marks see `protected override void OnMouseUp(MouseEventArgs e)` plus the drawing color and the pen width.

`sw.WriteLine( "</Canvas>" );` //Close the XAML-`Canvas` tag.

`sw.Close();` //The XAML file is complete, close it.

Menu driven function `void MenuFileSaveAsSVG( object obj, EventArgs ea )`

`SaveFileDialog dlg = new SaveFileDialog();`
//Open the standard user dialog to choose a directory and a file name.

`dlg.Filter = "svg files (*.svg)|*.svg|All files (*.*)|*.*" ;` //The user can choose between 2 file types : 1) `*.svg`-files (=default file extension), 2) all files `*.*` (=second choice).

`if ( dlg.ShowDialog() != DialogResult.OK ) return;` //Forget everything, if the user does not quit the standard `SaveFileDialog` box by clicking its `Open`-button or if something else goes wrong.

`Int32 n0 = dlg.FileName.IndexOf( ".svg" );`
//Seek the extension "`.svg`" in the user's file name. If `n0` returns as `-1`, the user forgot the extension.

`if ( n0 < 0 ) dlg.FileName += ".svg";`
//If the user forgot the extension, append it automatically to the file name.

`StreamWriter sw = new StreamWriter( dlg.FileName );`
//Create (or open if it already exists) the selected file as an ordinary plain text file.

`sw.WriteLine( "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>" );`
//Write the obligatory first line of a SVG-file.

`sw.WriteLine( "<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"100%\" height=\"100%\">" );` //Write the obligatory second line of a SVG-file.

`sw.WriteLine( "<title>This file comes from Prof. Miszalok's XML1.exe</title>" );`
//Write a HTML title line. This line is not important.

`sw.WriteLine("<polyline fill = \"none\" stroke = \"red\" stroke-width = \"4\" points = " );`
//Leading information of the XML-`polyline` tag indicating that vertices will follow immediately.

`sw.WriteLine( polygon_string );` //Write the preformattedd string containing all vertices.
(separated by commas and enclosed between 2 quotation marks
see `protected override void OnMouseUp(MouseEventArgs e)`

`sw.WriteLine( " />" );` //Close the XML-`polyline` tag.

`sw.WriteLine( "</svg>" );` //Close the `svg` tag (see the obligatory second line).

`sw.Close();` //The SVG file is complete, close it.

Menu driven function `void MenuFileRead( object obj, EventArgs ea )`

`OpenFileDialog dlg = new OpenFileDialog();`
//Open the standard user dialog to choose a directory and a file name.
`dlg.Filter = "All files (*.*)|*.*" ;`
//The user will see all files of any type which are contained in the directory.
`if ( dlg.ShowDialog() != DialogResult.OK ) return;` //Forget everything, if the user does not quit the standard `OpenFileDialog` box by clicking its Open-Button or if something else goes wrong.

`StreamReader sr = new StreamReader( dlg.FileName );`
//Open the selected file as an ordinary plain text file.
`String file_string = sr.ReadToEnd();`
//Read at once the complete file into a string in the main memory regardless of its length.

`polygon.Clear();` //Remove all former content from `polygon` and set its length (= no. of vertices) to zero.

`Int32 n0 = file_string.IndexOf( "olyline" ); if ( n0 < 1 ) return;`
//Try to find a substring "`polyline`" or "`Polyline`". If it can be found, it starts at position `n0`. If no such substring is found, `n0` returns `-1`. In this case we forget this file, because it contains neither a XAML- nor a SVG-polygon. If there are more than one "`olyline`" in the file, we ignore all but the first one.

`Int32 n1 = file_string.IndexOf( "oints", n0+7 ); if ( n1 < 1 ) return;`
//Starting behind the substring "`olyline`" we look for a substring "`points`" or "`Points`". If no such substring is found, `n1` returns `-1`. In this case we forget this file, because it contains neither XAML- nor SVG-vertices.

`Int32 n2 = file_string.IndexOf( "\"" , n1+5 ); if ( n2 < 1 ) return;`
//Starting behind the substring "`oints`" we look for a quotation mark. If no quotation mark is found, `n2` returns `-1`. In this case we forget this file, because it contains neither XAML- nor SVG-vertices.
The notation of a quotation mark string in C, C++ and C# is rather confusing: The quotation mark must be preceeded by an ESCAPE character (= back slash) and enclosed by 2 quotation marks adding up to 4 characters: `"\""` .

`Int32 n3 = file_string.IndexOf( "\"" , n2+1 ); if ( n3 < 1 ) return;`
//Starting behind the first quotation mark (which preceds the x-coordinate of the first vertex) we look for the next quotation mark (which follows the y-coordinate of the last vertex).

`String all_coordinates_string = file_string.Substring( n2+1, n3-n2-1 );`
//Cut out a substring `all_coordinates_string` from the x-coordinate of the first vertex to the y-coordinate of the last vertex.

`string[] coordinates_string_array = all_coordinates_string.Split(',');`
//Split up `all_coordinates_string` into a dynamic array of even smaller substrings by cutting `all_coordinates_string` in slices wherever a comma is.

`sr.Close();` //Close the file.

`for ( Int32 i=0; i < coordinates_string_array.Length; i+=2 )`
//Run through all slices (each containing a x- or a y-coordinate, the comma separators have been gone). Make jumps of 2 coordinates at once.

`p1.X = Convert.ToInt32( coordinates_string_array[i ] );`
//Take slice no. `i`, convert it from string to integer (leading blanks do not disturb) and store it as member `X` of a `Point` structure.

`p1.Y = Convert.ToInt32( coordinates_string_array[i+1] );`
//Take slice no. `i+1`, convert it from string to integer (leading blanks do not disturb) and store it as member `Y` of a `Point` structure.

`polygon.Add( p1 );` //Enlarge the dynamic array `polygon` by adding one `Point` structure (= one vertex).

`Invalidate();` // //Ask the operating system to send a `Paint`-Message to `Form1`.
After receiving this message `Form1` will execute our method (described above):
`protected override void OnPaint( PaintEventArgs e )` which redraws the complete polygon.