Creating custom controls Graphics

Software techniques

Department of Automation and Applied Informatics

Creating custom controls

Creating custom controls

- If we want to reuse some custom UI functionality
  - Write once, drag from toolbox many times
- Simple
- There are three ways
  - Derive from the Control class
  - Derive from an existing control
  - Create a UserControl

Deriving from the Control base class

- **Use it if you want to create a whole new control that does not resemble existing ones**
- It will just inherit the properties and behavior that is common for all Control objects
- New properties and events can also be added
- We have to write the drawing ourselves
  - Using GDI+
- Example: a label that contains the current time

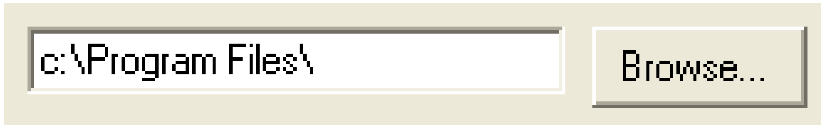
Deriving from an existing control

- **Use it if you just want to tailor an existing control,** e.g. from a TextBox
- Only the custom behavior should be implemented
- New properties and events can also be added
- Example: a special textbox that has got a red background if the entered text is not a valid e-mail address

Demo

Deriving from the UserControl class

- **The control we want to create is a form itself and contains additional controls: we group existing controls in some way.**
- In design time we place controls on the UserControl, just as if it was an ordinary form.
  - What is the difference? It shows up in the Toolbox: thus, it can be placed on forms or on other user controls.
- Example: FilePicker

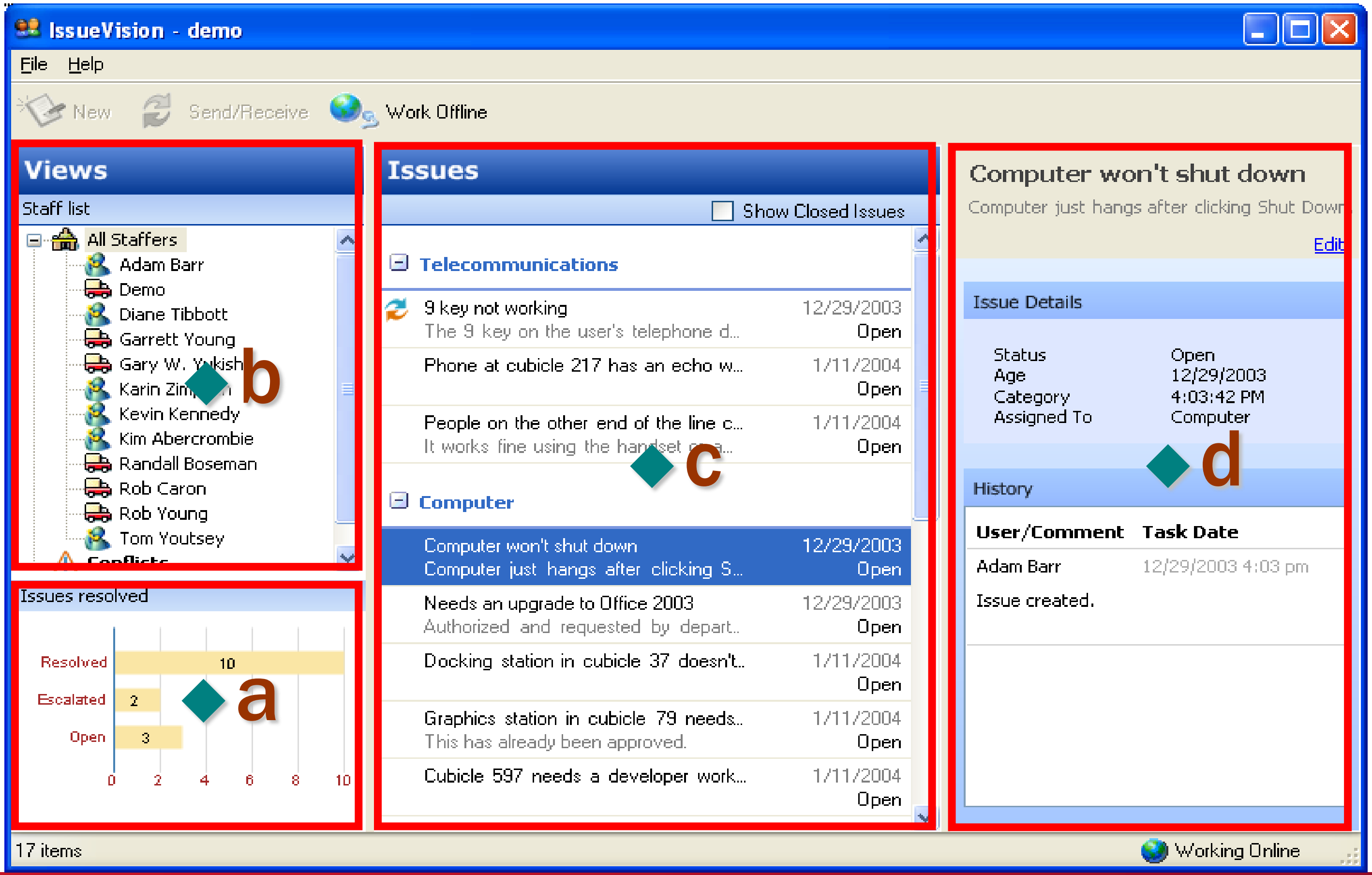


- The contained controls should be private
- How to get started? Visual Studio: Add New Item/User Control

The significance of UserControls

When should we create a custom UserControl?

- Reusability
- Complex user interfaces can be modularized!



9 key not working	The 9 key on the user's telephone d...	12/29/2003 Open
Phone at cubicle 217 has an echo w...		1/11/2004 Open
People on the other end of the line c...	It works fine using the handset or a...	1/11/2004 Open
Computer won't shut down	Computer just hangs after clicking S...	12/29/2003 Open
Needs an upgrade to Office 2003	Authorized and requested by depart...	12/29/2003 Open
Docking station in cubicle 37 doesn't...		1/11/2004 Open
Graphics station in cubicle 79 needs...	This has already been approved.	1/11/2004 Open
Cubicle 597 needs a developer work...		1/11/2004 Open

Status	Open
Age	12/29/2003 4:03:42 PM
Category	Computer
Assigned To	

User/Comment	Task Date
Adam Barr	12/29/2003 4:03 pm
Issue created.	

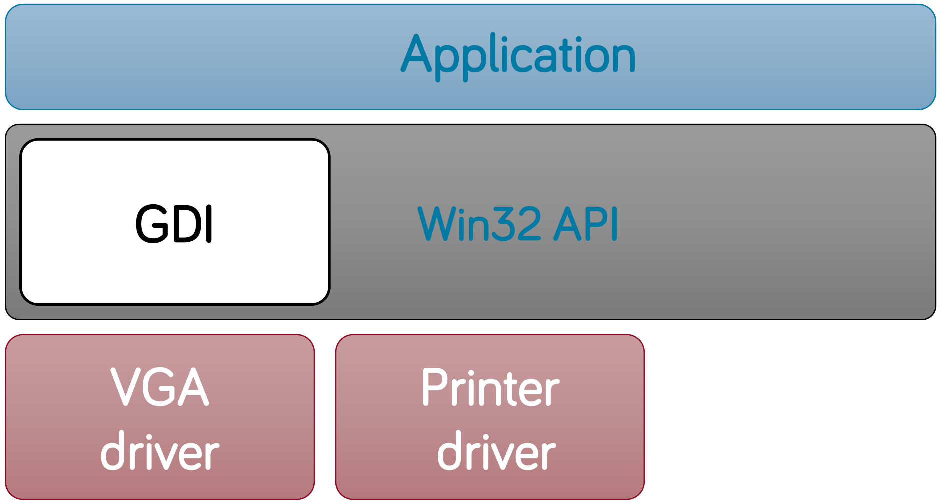
Using graphics in a native environment

GDI - Graphics in native environment

- GDI - Graphical Devices Interface
- Graphical display in native Win32 environment
- Capabilities
  - Vector-graphical operations: drawing point, rectangle, etc.
  - Only for 2D
  - Displaying text
  - Displaying images
  - Supports metafile
  - Coordinate transformations
  - …

GDI architecture

- Supports device-independent display
  - Resolution and color depth independent drawing
  - We “draw” to the printer in the same way as we draw to the screen
    - When drawing to the printer we have to specify the page breaks



- The device-independency is achieved by the help of the Device Context concept

What is the Device Context?

- Represents a graphical device
- Is identified by a HDC
- Steps
  - 1. Create the device context
  - 2. Draw on the device context
  - 3. Close the device context
- It’s the “Drawing surface”: this is the first parameter of every drawing function
- Device context can be created for
  - Client area of a window
  - Full window (to the header, status bar, menu …)
  - Screen ☺
  - Memory (drawing to the memory)
  - Printer
  - Metafile
  - Invalid area of a window

Device context

- Drawing to the client area of a window (don't memorize the code)

```
HDC hdc;
hdc = GetDC(hWnd); // create DC
Rectangle( hdc, 10, 10, 20, 20 );  // Drawing
ReleaseDC(hdc); // release DC
```

- The OS does not save the content of the window
  - If there is an invalid area of a window it has to be redrawn
  - Invalid area:
    - The area of the window that was once hidden by an other window and is now unhidden
  - Why doesn't the OS save the content of a window? There can be many windows -> needs a lot of memory
  - How do we know that there is an invalid region in the window?
    - The window will get a WM_PAINT message

Redrawing the invalid area

- Drawing is done as a response to the WM_PAINT message

```
HRESULT CALLBACK WndProc(HWND hWnd, UNIT message,
       WPARAM wParam, LPARAM lParam)
{
   HDC hdc;
       switch(message)
       {
          case WM_PAINT:
                 hdc = BeginPaint(hWnd, &ps); // create DC
                 Rectangle(hdc, 10, 10, 20, 20); // drawing
                 EndPaint(hWnd, &ps); // release CD
                 break;
       ...
       }
}
```

- Or we create the drawing in the memory and just copy it when the window gets a WM_PAINT message. This is called double buffering.

Graphics in managed environment

Architecture

- .NET supports GDI+ (= GDI + many extra services)
  - Very similar to the native Win32 API model
  - System.Drawing namespace and assembly
  - Instead of the device context (DC) the System.Drawing.Graphics object is used to draw on it. This represents the drawing surface.
- Drawing to the invalid area is just like in the native environment
- How can we handle it?
  - Create an event handler for the Paint event of the Form/Control .
  - Or override the onPaint() method (more efficient)
- When needed call Invalidate() => OnPaint()

Simple example

- In the class that derives from the Form base class

```
protected override void OnPaint(PaintEventArgs e)
{
     e.Graphics.FillRectangle(
          new SolidBrush(Color.Blue), this.ClientRectangle);
}
```

- PaintEventArgs parameter
- PaintEventArgs.Graphics object: we can draw on it
- PaintEventArgs.ClipRectangle: the area that needs refreshing (rarely used)
- Form.ClientRectangle: the client area of the window (rarely used)
- Not only forms but controls can also be redrawn this way!
- Don’t call **OnPaint()** explicitly (OS makes some optimizations) Instead: Call Invalidate() that will invalidate an area that will cause the redraw.

Handling Colors

- Color structure
  - Color.Red - the red color
  - Color.FromArgb
    - 4 component (alpha, red, green, blue), the range is 0-255 for all the components
    - **Color.FromArgb(127, Color.Red)**; // Half opaque red
    - **Color.FromArgb(100, 255, 0, 0);** // Half opaque red
  - Color.Empty – empty color (null color)
  - Color.Transparent - The transparent color

Drawing operations

- The operations of the Graphics class
  - Graphics.DrawRectangle (Pen, Rectangle) - rectangle
  - Graphics.DrawRectangle (Pen, Int32, Int32, Int32, Int32)
  - Graphics.DrawLine(Pen, Point, Point)
  - Graphics.FillRectangle (Brush, Rectangle) - filled rectangle
- There are many other operations:
  - DrawBezier – Bezier curve
  - DrawCurve – Spline curve
  - …

Drawing methods

- Two basic types:
  - Name starts with Draw….(): draw outline with a pen
  - Name starts with Fill…(): draw inside the outline with a brush

Pen

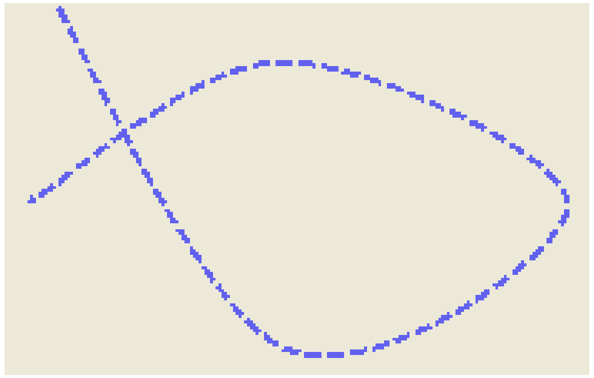
- The pen determines the color, the thickness and the pattern of the lines
- Pen class
  - Example 1 - Unfilled rectangle with a given pen

```
The operation of the Graphics class:
public void DrawRectangle ( Pen pen, int x, int y,
     int width, int height )
```

  - Example 2

```
protected override void OnPaint(PaintEventArgs e) {
     e.Graphics.DrawLine( new Pen(Color.Red), 10, 20, 220, 20);
}
```

  - Example 3



Pen

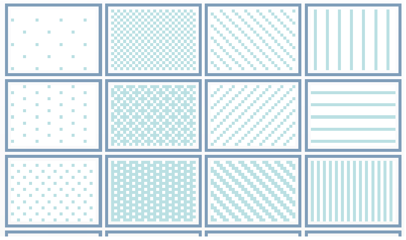
```
protected override void OnPaint(PaintEventArgs e)
{
     // Constructor parameter: the color and the thickness of the line
     using (Pen pen = new Pen(Color.FromArgb(150, Color.Blue), 2))
    {
           // Line pattern (solid if not specified)
        pen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
           // End of line (nothing if not specified)
        pen.EndCap = LineCap.ArrowAnchor;

         // spline
        g.DrawCurve(pen, new Point[] {
                    new Point(10, 110), new Point(100,250), new Point(200, 200),
                    new Point(100, 150), new Point(10, 200)});
    }
}
```

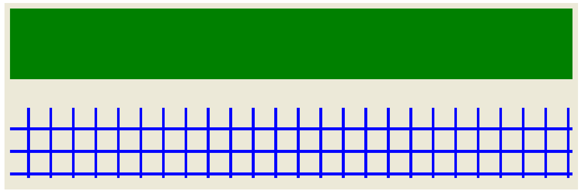
- What you need to know: Using a Pen with DrawLine() and DrawRect()
- There are predefined Pens, e.g.: Pen.Blue (blue pen with a thickness of 1px)
  - Simple: no need to create and release!

Brush

- The brush determines the fill color and pattern.
- Brush is the base class, the derived classes:
  - SolidBrush - with a given color
  - HatchBrush - with a line-based pattern
  - TextureBrush - with a bitmap pattern
  - LinearGradientBrush - linear gradient for two colors
  - …
- Example 1



```
protected override void OnPaint(PaintEventArgs e) {
       e.Graphics.FillRectangle(new SolidBrush(Color.Green),
         10, 30, 200, 25);
       e.Graphics.FillRectangle(new HatchBrush(HatchStyle.Cross,
          Color.Blue, Color.Transparent), 10, 65, 200, 25);
}
```



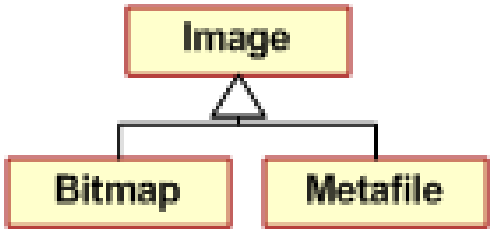
Brush

- Example 2

```
protected override void OnPaint(PaintEventArgs e)
{
        Rectangle r = new Rectangle(10, 100, 200, 25);
        g.FillRectangle( new LinearGradientBrush(
                  r, // The rectangle for the brush
                  Color.Red, // Start color
                  Color.Yellow, // Target color
                  LinearGradientMode.BackwardDiagonal), // Style
          r);
}
```

- There are predefined ones:
  - Brushes.Blue - blue brush
  - Simple to use: no need to create, no need to release!

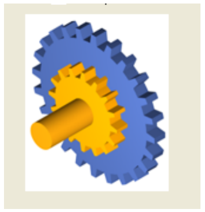
Displaying an image



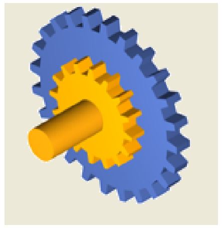
- **Bitmap - Bitmap class**

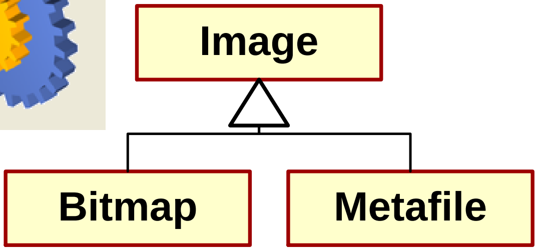
```
protected override void OnPaint(PaintEventArgs e)
{
        using (Bitmap bmp = new Bitmap("wheel.png")) // Path for the file
     {
             // (0,0) point defines the transparent color
           if (checkBoxTransparentImage.Checked)
                bmp.MakeTransparent(bmp.GetPixel(0, 0));
           e.Graphics.DrawImage(bmp, 250, 70, bmp.Width / 2, bmp.Height / 2);
     }
}
```

◆ Not transparent display:



◆ Transparent display:





- **Metafile - Metafile class**
  - Pre-saved vector graphical operations

Displaying text

- Simple, uses the font style of the form

```
protected override void OnPaint(PaintEventArgs e)
{
      e.Graphics.DrawString("Hello World", this.Font,
          new SolidBrush(Color.Black), 10, 10);
}
```

- In the left, up corner of the rectangle

```
protected override void OnPaint(PaintEventArgs e)
{
     e.Graphics.DrawString("Hello World",
       new Font("Lucida Sans Unicode", 8),
       new SolidBrush(Color.Blue),
        new RectangleF(100, 100, 250, 350));
}
```

Displaying text

- StringFormat can be used for “everything”

```
string text = "Hello World";

StringFormat format = new StringFormat();
// Justifying (Near - left, Central, Far - right)
format.Alignment = StringAlignment.Center;
// Horizontal alignment (Near – up, Center, Far – down)
format.LineAlignment = StringAlignment.Center;
//No line break
format.FormatFlags = StringFormatFlags.NoWrap;
// will clip the text if necessary
format.Trimming = StringTrimming.EllipsisCharacter;

g.DrawString( text, new Font("Arial", 16, FontStyle.Bold),
     new SolidBrush(Color.Blue),
     new RectangleF(100, 100, 250, 350),
     format);
```

What else can the GDI+ be used for?

- Anti-aliasing
- Built-in support for: .jpeg, .png, .gif, .bmp
- Image processing: lightness, contrast, etc.
- Line styles
- Full Unicode support
- We can use floats as well

```
public void Graphics.FillRectangle( Brush brush,
        float x, float y, float width, float height )
```

  - RectangleF, PointF types
- Custom shaped forms
- Printing!
- Many other things…

- No 3D (DirectX is needed for 3D)

Windows Forms: summary

- Windows Forms makes creating smart client applications simple and convenient
- Far from perfect:
  - For example it is difficult to create DPI independent applications
- Newer technologies exist
  - Windows Presentation Foundation, WPF (from .NET 3.0)
  - Universal Windows Application, UWP (from Windows 10)
- We've missed a few subjects for now
  - Data validation
  - Data binding
  - Printing
  - …

Examples

- Exercise 1 Count the number of redrawings of a window. Display the number in the middle of the window.
  - Resizing the window: by default it won't cause a redraw. For Permitting it: the windows class has to be changed

```
this.SetStyle(ControlStyles.ResizeRedraw,true);
UpdateStyles();
```

  - In this case it will flicker when it is resized. Allow double buffering for the form!

```
SetStyle(ControlStyles.DoubleBuffer |
        ControlStyles.UserPaint |
        ControlStyles.AllPaintingInWmPaint,
        true);
UpdateStyles();
```

Examples

- Exercise 2 Increase a counter by one every second. Display the counter in the middle of the window!
  - In the event handler of the timer increase the counter by one
  - Don’t call OnPaint(), instead: Invalidate()

Manual double buffering

- If the drawing itself is slow (there are complex figures, many drawings) the display will flicker. Reason: for every OnPaint() call the slow redrawing algorithm will redraw the user interface.
  - Note that: simply enabling the double buffering won't solve the problem
  - Solution: Using virtual windows. Basic idea: we draw on a memory Graphics object and just copy it to the screen in the OnPaint() method.

Steps of manual double buffering

- 1. Preparation: save the bitmap in a member variable

```
private Bitmap m_Bmp = null;

private void InitMemGraphics()
{
     using (Graphics gForm = this.CreateGraphics()) // this means the form i
     {
     if (m_Bmp != null)
         m_Bmp.Dispose();
      m_Bmp = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
          Screen.PrimaryScreen.Bounds.Height, gForm);
       ...
     }
}
```

> Remark:
  - The Form.CreateGraphics will create a Graphics for a given form.
  - The last parameter of the constructor of the Bitmap class is a Graphics object

Steps of manual double buffering

- 2. If the drawing changes: we recreate the drawing in the memory

```
private Bitmap m_Bmp = null;

private void CreateDrawing()
{
     using (Graphics g = Graphics.FromImage(m_Bmp))
     {
        // Draw on it
        g.DrawLine(...);
        ...
        // Refresh the drawing on the screen
        Invalidate();
     }
}
```

Steps of manual double buffering

- 2. In the OnPaint() method we copy the bitmap to the screen (this is very fast, and will not flicker)

```
protected override void OnPaint(PaintEventArgs e)
{
        if (m_Bmp != null)
          e.Graphics.DrawImage( m_Bmp, new Point(0, 0) );
}
```

Coordinate transformations

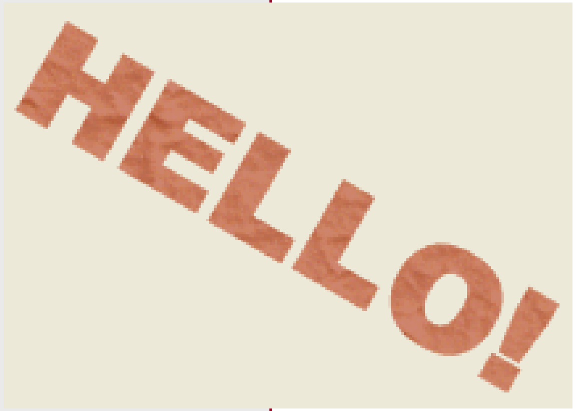
- Linear transformations can be used for a drawings

```
protected override void OnPaint(PaintEventArgs e)
   {
      Graphics g = e.Graphics;
       // Matrix for the transformation
      Matrix m = new Matrix();
      // Rotate around the (80, 270) point

      m.RotateAt(30, new PointF(80, 270));
      // Activate the transformation
      g.Transform = m;

      g.DrawString("HELLO!",
         new Font("Arial Black", 40, FontStyle.Bold),
         new TextureBrush(
            new Bitmap("Santa Fe Stucco.bmp")),
         new PointF(10, 260)
         );

      g.ResetTransform();
}
```



Questions

- Creating custom controls
  - What are the steps of creating custom controls?
  - What can custom controls be used for?
- Graphics
  - GDI architecture What does device independent drawing mean?
  - What is a device context?
  - Introduce the drawing mechanism in native environment! (Invalid area, WM_PAINT, messages, etc.)
  - Introduce the drawing mechanism in managed environment! (with code demonstration) (Invalid area, Paint event, OnPaint method, Graphics class, …)
  - Introduce the basics of using colors
  - Introduce the Pen and the Brush classes, give some simple examples!
  - What is a metafile?

Questions

- What are the GDI+ resources? What do we have to pay attention to?
- How can textual information be displayed? Give an example for displaying a text at a given point!
- Introduce the usage of virtual windows! (When should it be used? What are the steps of the solution?)
- Example: Write a C# application that counts the number of window redraws and displays this counter at the 10,10 position of the window!
- Example: Write a C# application that increases the value of a counter by one in every second. Display the value of the counter at the 10,10 position of the window!
- Write a C# application that increases the value of a counter by one in every second. Display the value of the counter at the 10,10 position of the window!

Questions

- Write a C# application that draws a red rectangle at position 10, 10. The sides of the rectangle are 10 pixels. The user should be able to move the rectangle by pressing the cursor keys. (The codes for the cursor keys: Keys.Up, Key.Left, …)

References

- Graphics and Drawing in Windows Forms
  - http://msdn2.microsoft.com/en-us/library/a36fascx.aspx
- Graphics and Windows Forms Quickstart
  - http://samples.gotdotnet.com/quickstart/winforms/doc/WinFormsGDIPlus.aspx#Introduction

⑩