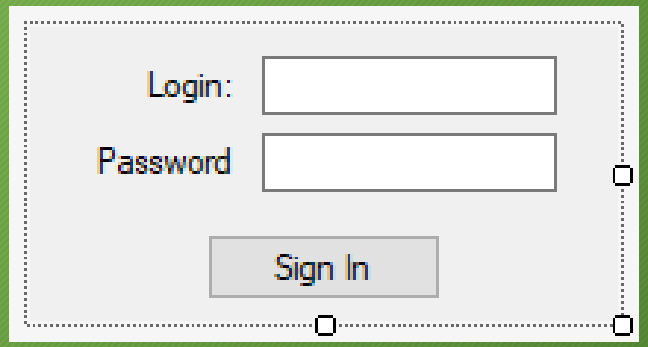
Windows Forms

Szymon Szczepański szymon.szczepanski@gmail.com
Maciej Świechowski m.swiechowski@mini.pw.edu.pl
Paweł Aszklar p.aszklar@mini.pw.edu.pl

UserControl

- Composite controls
- Reusable portions of the user interface
- Similar to forms, but have no border, title bar and cannot be top-level windows
- Allow usage of standard controls with their functionality and appearance
- For more complicated views improve performance of designer
- Do not create control as god controls
- The best practice is have model for UserControl



Custom Controls

- **Composite control (user control)**
  - composition of Windows Forms controls in a common container (**UserControl)**
  - the only choice with design-time support
  - It's most common way to create custom control
- **Extended control (derived control)**
  - inherited from an existing Windows Forms control
  - functionality extended
  - OnPaint() method overriden to create a custom appearance
  - Often use if some custom behaviors/styles are neede but control is one of core controls
- **Custom control**
  - inherited from one of the base control classes
    - Component, Control, ScrollableControl, ContainerControl
  - the most flexible (and the most time-consuming) way to create controls
  - Use only when you need create something from scratch

Visual Studio Support

- Every time a class library is compiled, Visual Studio scans through the classes it contains, and adds each component or control to a special temporary tab at the top of the Toolbox
  - Options -> Windows Forms Designer -> Automatically Populate Toolbox
- The first time a control is added to a project (e.g. by dragging from the Toolbox), Visual Studio:
  - adds a reference to the assembly where the control is defined
  - copies this assembly to the project directory
- Toolbox can be customized
  - toolbox is a user-specific Visual Studio setting, not a project-specific setting
- Every change in control will be available after build

Preprocessing Input Messages

- **PreProcessMessage**
  - virtual method on **Control** class
  - messages pre-processed: WM_KEYDOWN, WM_SYSKEYDOWN, WM_CHAR, WM_SYSCHAR
  - return:
    - true – for processed messages
    - base.PreProcessMessage – otherwise
- More specialized methods:
  - IsInputChar(), IsInputKey()
  - ProcessCmdKey()
  - ProcessDialogChar(), ProcessDialogKey()

Message Handling Hooks - Application.AddMessageFilter()

- Use a message filter to prevent specific events from being raised or to perform special operations for an event before it is passed to an event handler. Message filters are unique to a specific thread.

```
public class TestMessageFilter : IMessageFilter
{
   public bool PreFilterMessage(ref Message m)
   {
      //Blocks all the messages relating
      //to the left mouse button.
      if (m.Msg >= 513 && m.Msg <= 515)
      {
         Console.WriteLine("Processing the messages :”
                                                + m.Msg);
         return true;
      }
      return false;
   }
}
```

```
Application.AddMessageFilter(new TestMessageFilter());
```

Overriding Window Procedure

- Message: {HWnd, LParam, Msg, Result, WParam}
- Form.Handle – HWND

```
private const int HTCAPTION = 0x0002;
private const int WM_NCHITTEST = 0x0084;


protected override void WndProc(ref Message m)
{
  switch (m.Msg)
  {
    case WM_NCHITTEST:
      m.Result = (IntPtr)HTCAPTION;
      break;
    default:
      base.WndProc (ref m);
      break;
  }
}
```

Data Binding Components

- **BindingSource**
  - component encapsulating data source to simplify binding (both simple & complex)
  - currency management
  - change notifications
- **ContainerControl.BindingContext**
  - manages collection of instances of **BindingManagerBase**:
    - **PropertyManagers** for simple binding
    - **CurrencyManagers** for complex binding

```
this.BindingContext[DataItems].Position += 1;
```

Data Binding

- Mechanism for connecting control properties with data (one-way or two-way) e.g.:
  - Setting the graphic of an image control.
  - Setting the background color of one or more controls.
  - Setting the size of controls.
- Notify (INotifyPropertyChanged interface or *PropertyName*Changed events)

- Simple binding: single values
- Complex data binding: collections
  - Typical usage:
    - **DataGridView** and **DataSet**
      - **DataSet**
        - represents relational data (incl. tables and relations)
        - supports multiple data views
        - supports data changes tracking
      - **DataGridView**
        - designed for presentation and modification of tabular data
        - extensive support for binding - especially to DataSets and DataTables

Data Source

- Simple binding
  - Control.DataBindings.Add (*propertyName*, *dataSource*, *dataMember*)
- Complex binding:
  - **DataGridView**
    - DataSource
    - DataMember
  - **ListBox**, **ComboBox**, …
    - DataSource
    - DisplayMember, ValueMember, SelectedValue
- Structures to Bind To
  - BindingSource
  - SimpleObject or IEditableObject
  - BindingList<T>
  - Array or Collection (IList<T> or preferably IBindingList<T>)
  - IEnumerable (through BindingSource)
  - ADO.NET (DataSet, DataTable, DataColumn, DataViewManager)

Resources

- An assembly is a collection of types and optional resources
  - the binary data, text files, audio files, video files, string tables, icons, images, XML files
- Localized applications
  - a problem with multilingual user interface
  - This step involves customizing an application for specific cultures or regions. If the globalization and localizability steps have been performed correctly, localization consists primarily of translating the user interface.
  - for each resource added to an assembly, it is possible to specify the culture information (a language and country, e.g. "pl-PL", "en-US", "de-De", "de-AT")
  - satellite assemblies
  - Winres.exe

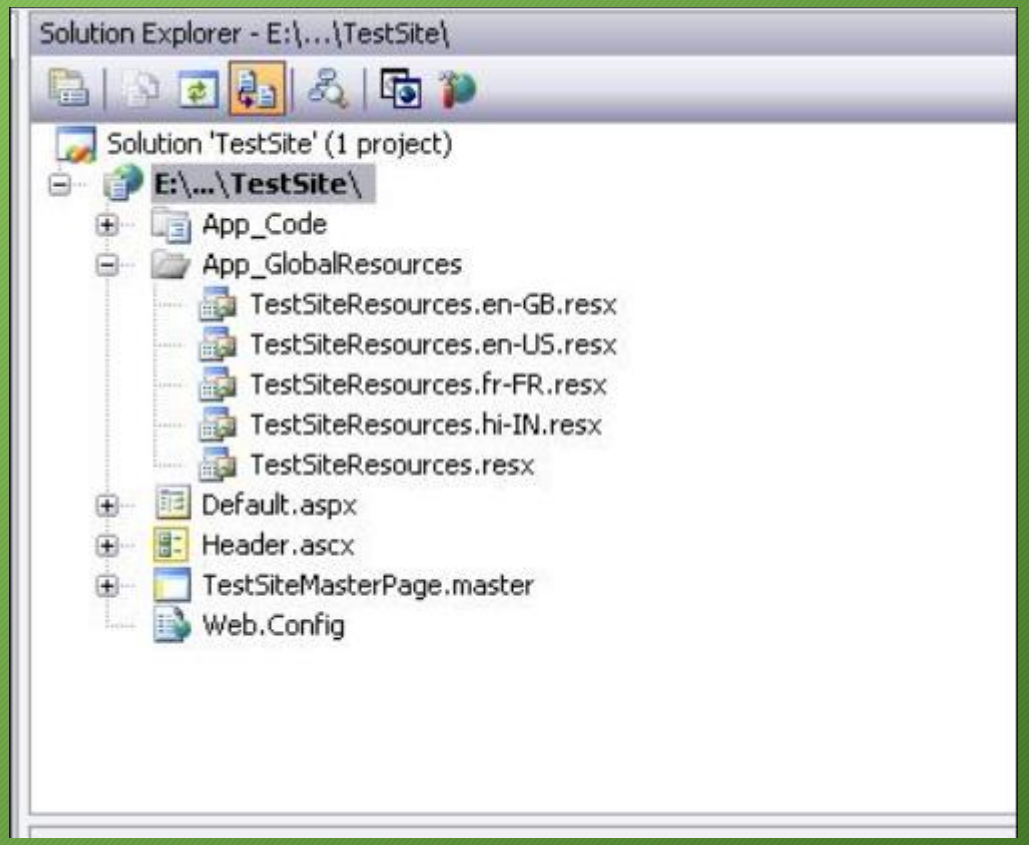
Satellite Assembly

- A single satellite assembly must include all the resources for a particular culture. In other words, you should compile multiple .txt or .resx files into a single binary .resources file.
- There must be a separate subdirectory in the application directory for each localized culture that stores that culture's resources. The subdirectory name must be the same as the culture name. Alternately, you can store your satellite assemblies in the global assembly cache. In this case, the culture information component of the assembly's strong name must indicate its culture.

Satellite assembly directory

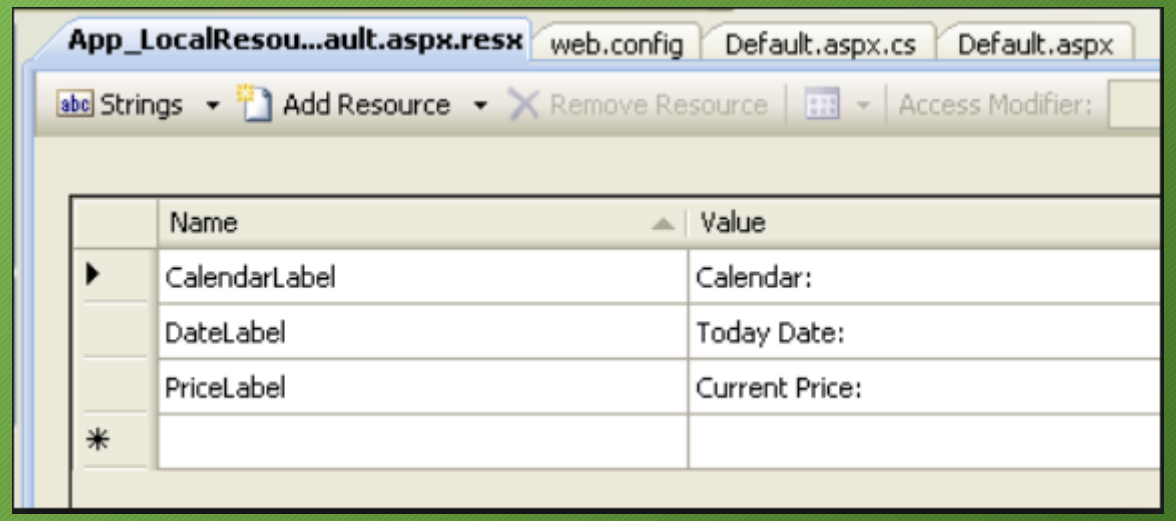
Typed Resource Files

- .txt
  - textual name/value format
  - an easy way to add string resources
- .resx
  - the XML format
  - support for both strings and other objects such as images
- .resources
  - the binary format
  - a binary equivalent of the XML file
  - the only format that can be embedded in an assembly, the other formats must be converted



Resources with Visual Studio

- Adding resources
  - in resx files:
    - Project > Add New Item > Resources File
    - or: Project > Properties > Resources
    - a property in a special class is generated for easier access to resource
- Editing resources
  - Visual Studio built-in editors
    - the binary editor, image editor
  - external editors (e.g. the Paint for image files)
    - other applications can be associated with types of resources
- Compiling resources into assemblies
  - resgen.exe tool is called automatically



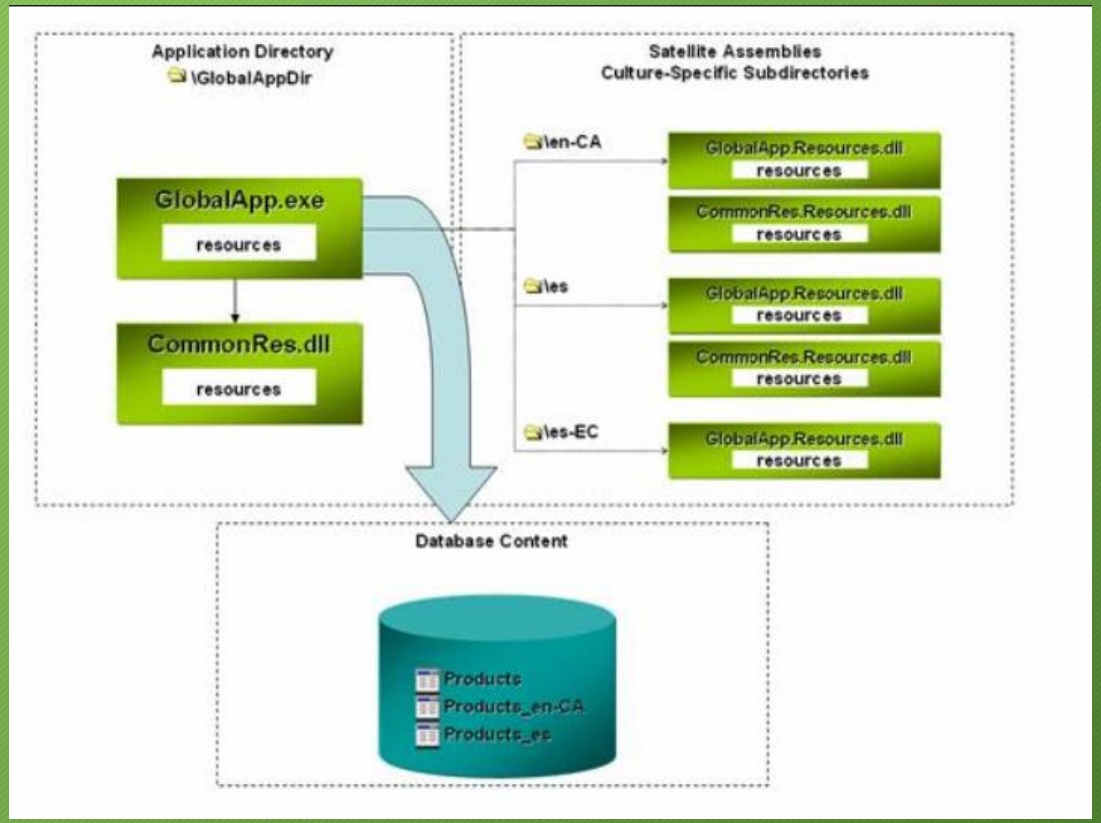
Retrieving Resources

- ResourceManager class to retrieve data from resources

```
ResourceManager rm =
   new ResourceManager("myApp.myRes",
   Assembly.GetExecutingAssembly());

MessageBox.Show(rm.GetString("HelloWorld"));

pictureBox.Image =
    (Bitmap)rm.GetObject("background");

rm.ReleaseAllResources();
```

Localizing Applications

- Retrieved resources language:
  - **ResourcesManager.Get...(*name*, *culture*)**
  - **Thread.CurrentUICulture**
  - VS-generated helper class:
    - **Resources.Culture**
- Default data formatting language:
  - **Thread.CurrentCulture**
- Keyboard input language:
  - **InputLanguage.InstalledInputLanguages, InputLanguage.DefaultInputLanguage, InputLanguage.CurrentInputLanguage**
  - **Application.CurrentInputLanguage**



Application Settings

- The Application Settings feature of Windows Forms makes it easy to create, store, and maintain custom application and user preferences on the client computer.
- Application-scoped settings can be stored in either the machine.config or app.exe.config files. Machine.config is always read-only, while app.exe.config is restricted by security considerations to read-only for most applications
- User-scoped settings can be stored in app.exe.config, in which case they are treated as static defaults.
- on-default user-scoped settings are stored in a new file, user.config, where user is the user name of the person currently executing the application. You can specify a default for a user-scoped setting with DefaultSettingValueAttribute. Because user-scoped settings often change during application execution, user.config is always read/write.
- All three configuration files store settings in XML format

GDI+

- Successor to Windows Graphics Device Interface (GDI)
  - supports GDI for compatibility with existing applications
  - optimizes many of the capabilities of GDI
  - provides additional features
- Class-based application programming interface (API)
  - two independent implementations: .NET (managed code) and unmanaged code
- Availability:
  - included in Windows XP+ and Windows Server 2003+
  - redistributable for NT 4.0 SP6, 2000, 98, Me
    - Gdiplus.dll

Graphics Class

- **Graphics** class:
  - core of GDI+
  - Device Context equivalent
    - associated with a particular window on the screen
    - contains attributes that specify how items are to be drawn
  - improvements (compared to DC):
    - less state:
      - pens, brushes, paths, images, and fonts as method parameters
      - no current position for drawing lines
    - separate methods for draw and fill
- GDI object equivalents:
  - represented as true objects
  - in .NET: implement IDisposable

Using **Graphics**

- Obtaining **Graphics** object:
  - Paint event handler: **PaintEventArgs.Graphics**
  - **Control.CreateGraphics**()
  - Graphics: **FromHdc()**, **FromHwnd()**, **FromImage()**
- Triggering the Paint Event:
  - **Control.Invalidate()**
  - **Control.Update()**, **Control.Refresh()**
- Flicker-free drawing: **this.DoubleBuffered = true** (more efficient since .NET 4.6)
- **High DPI** - in .NET 4.7

Simple Shapes

- Simple figures:
  - DrawLine(), DrawRectangle(), DrawEllipse(), DrawArc(), DrawPolygon()
  - FillEllipse(), FillPie(), FillPolygon(), FillRectangle()
- Cardinal splines:
  - DrawCurve(), DrawClosedCurve()
  - FillClosedCurve()
- Bezier splines:
  - DrawBezier(), DrawBeziers()

C# Copy

```
Rectangle myRectangle = new Rectangle(100, 50, 80, 40);
myGraphics.DrawEllipse(myPen, myRectangle);
```

Paths

- Formed by combining lines, rectangles, and simple curves
- **GraphicsPath** class:
  - adding simple figures:
    AddLine(), AddRectangle(), AddEllipse(), AddArc(), AddPie(), AddBezier(), AddCurve(), AddClosedCurve(), AddPolygon(), AddString()
  - joining two paths: AddPath()
- **Graphics.DrawPath()**
- **Graphics.FillPath()**

C# Copy

```
     GraphicsPath myGraphicsPath = new GraphicsPath();

     Point[] myPointArray = {
new Point(5, 30),
new Point(20, 40),
new Point(50, 30)};

     FontFamily myFontFamily = new FontFamily("Times New Roman");
     PointF myPointF = new PointF(50, 20);
     StringFormat myStringFormat = new StringFormat();

     myGraphicsPath.AddArc(0, 0, 30, 20, -90, 180);
     myGraphicsPath.StartFigure();
     myGraphicsPath.AddCurve(myPointArray);
     myGraphicsPath.AddString("a string in a path", myFontFamily,
        0, 24, myPointF, myStringFormat);
     myGraphicsPath.AddPie(230, 10, 40, 40, 40, 110);
     myGraphics.DrawPath(myPen, myGraphicsPath);
```

Regions

- Describe interiors of graphics shapes composed of rectangles and paths
- **Region** class
- Hit testing
  - Region.IsVisible(point, graphics)
- Clipping
  - Graphics.SetClip(region)

C# Copy

```
Point point = new Point(60, 10);

// Assume that the variable "point" contains the location of the
// most recent mouse click.
// To simulate a hit, assign (60, 10) to point.
// To simulate a miss, assign (0, 0) to point.

SolidBrush solidBrush = new SolidBrush(Color.Black);
Region region1 = new Region(new Rectangle(50, 0, 50, 150));
Region region2 = new Region(new Rectangle(0, 50, 150, 50));

// Create a plus-shaped region by forming the union of region1 and
// region2.
// The union replaces region1.
region1.Union(region2);

if (region1.IsVisible(point, e.Graphics))
{
    // The point is in the region. Use an opaque brush.
    solidBrush.Color = Color.FromArgb(255, 255, 0, 0);
}
else
{
    // The point is not in the region. Use a semitransparent brush.
    solidBrush.Color = Color.FromArgb(64, 255, 0, 0);
}

e.Graphics.FillRegion(solidBrush, region1);
```

Brushes

- **Brush** – abstract base class for all brushes:
  - SolidBrush, HatchBrush, TextureBrush, LinearGradientBrush, PathGradientBrush

C# Copy

```
System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);
System.Drawing.Graphics formGraphics;
formGraphics = this.CreateGraphics();
formGraphics.FillEllipse(myBrush, new Rectangle(0, 0, 200, 300));
myBrush.Dispose();
formGraphics.Dispose();
```

Pens

- Attributes
  - **Width**
  - **Alignment (PenAlignment** enumeration)
  - Line caps:
    - **StartCap**, **EndCap** (**LineCap** enumeration)
    - **CustomStartCap**, **CustomEndCap** (**CustomLineCap** class)
  - **LineJoin** (**LineJoin** enumeration)
  - **DashStyle, DashPattern**, **DashCap**
  - **Brush**

C# Copy

```
System.Drawing.Pen myPen;
myPen = new System.Drawing.Pen(System.Drawing.Color.Tomato);
```

Images

- **Image** – abstract base class
- **Bitmap** – derived from **Image**
  - contains specialized methods for loading, displaying, and manipulating raster images
- **Graphics.DrawImage…()**
  - many overrides available
  - flexible resizing/cropping possibilites
  - influenced by **Graphics.InterpolationMode**
  - influenced by **Graphics.Transform**

C# C++ VB

```
private void ImageExampleForm_Paint(object sender, PaintEventArgs e)
{
    // Create image.
    Image newImage = Image.FromFile("SampImag.jpg");

    // Create Point for upper-left corner of image.
    Point ulCorner = new Point(100, 100);

    // Draw image to screen.
    e.Graphics.DrawImage(newImage, ulCorner);
}
```

Image encoders/decoders

- Listing installed encoders and decoders:
  - **ImageCodecInfo.GetImageEncoders**(), **ImageCodecInfo.GetImageDecoders**()
    - return arrays of **ImageCodecInfo**

```
public void ConvertImage(string srcPath,        string destPath,
ImageFormat destFormat)
{
    Image image = new Bitmap(srcPath);
    image.Save(destPath, destFormat);
}
```

```
try {
   ConvertImage("bird.jpg", "bird.png", ImageFormat.Png);
}
catch (Exception exc) { … }
```

Text

- **Graphics.DrawString()**
  - at specified location
  - in a rectangle
  - **StringFormat** argument:
    - Alignment, LineAlignment
    - SetTabStops()
    - FormatFlags
  - TextRenderingHint – allows switching on antialiasing
- **Graphics.MeasureString()** – find size before drawing

C# C++ VB

```
public void DrawStringPointF(PaintEventArgs e)
{

    // Create string to draw.
    String drawString = "Sample Text";

    // Create font and brush.
    Font drawFont = new Font("Arial", 16);
    SolidBrush drawBrush = new SolidBrush(Color.Black);

    // Create point for upper-left corner of drawing.
    PointF drawPoint = new PointF(150.0F, 150.0F);

    // Draw string to screen.
    e.Graphics.DrawString(drawString, drawFont, drawBrush, drawPoint);
}
```

Text

- **TextRenderer**
  - Native, GDI based, faster
  - Matches text displayed by controls
  - Better support for international text
  - Different wrapping and spacing behavior
  - **TextRenderer.DrawText()**, **TextRenderer.MeasutreText()**
- Alternatively, to make text inside controls match Graphics.DrawString() output
  - For all controls, set in Main:

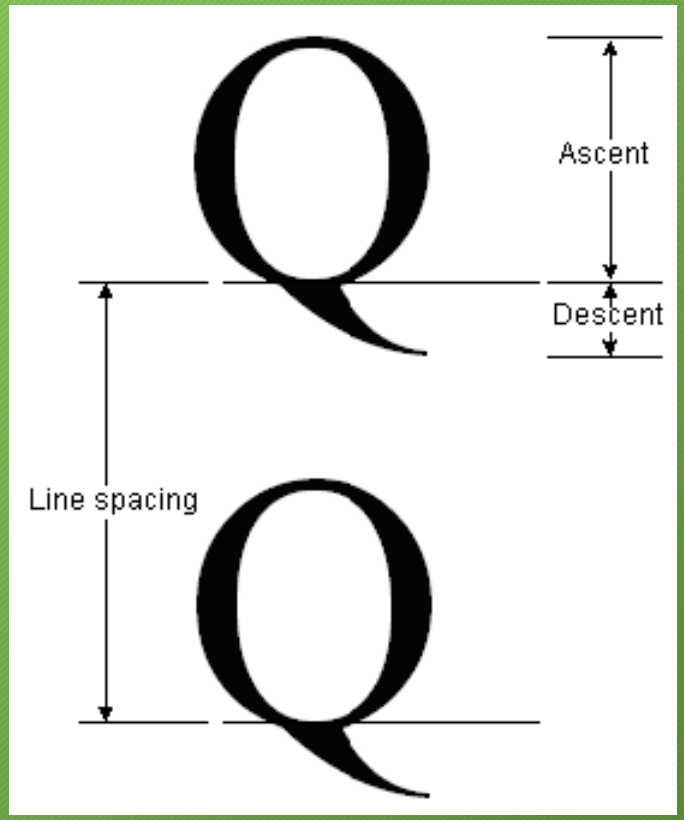
```
Application.SetCompatibleTextRenderingDefault(true);
```

  - For individual Control

```
button1.UseCompatibleTextRendering = true;
```

Fonts

- **Font** class
- Metrics
  - Font.GetSize()
  - FontFamily.GetEmHeight()
  - FontFamily.GetCellAscent()
  - FontFamily.GetCellDescent()
  - FontFamily.GetLineSpacing()




```
Font font = new Font("Arial", 16,
      FontStyle.Regular);
```

Transformations

- **Matrix** class
- **Graphics.Transform**
  - helper methods:
    - **ScaleTransform()**
    - **RotateTransform()**
    - **TranslateTransform()**
- Transformation order is significant

Graphics Containers

- **GraphicsContainer** class
- Stores state of **Graphics**:
  - link to device context
  - quality settings
  - transformations
  - clipping region
- Alternative:
  - Graphics.Save()

```
Pen pen = new Pen(Color.Red);
gr.TranslateTransform(100.0f, 80.0f);

GraphicsContainer
   graphicsContainer = gr.BeginContainer();
gr.RotateTransform(30.0f);
gr.DrawRectangle(pen,-60,-30,120,60);
gr.EndContainer(graphicsContainer);

gr.DrawRectangle(pen,-60,-30,120,60);
```