Course: Introduction to C# .NET
Title: Variables and Calling Methods
Dates: Afternoon, 21st March 2005

NAPIER UNIVERSITY
EDINBURGH SCOTLAND

2 Day 1 Monday 21st March 2005

Day 1 (Afternoon) Instructor: A. Cumming
Scope
Variables and Types Naming Variables Using Built-In Data Types Introduction to the Visual Studio Environment • Design Surface – dynamic help • Toolbox • Code Page • Properties Window • Solution Explorer
Practical Examples
Deep Thought (Demonstration) Flags (Programming exercise)

2.1 Variables and Types

The C# language is strongly typed. This means that variables are constrained by a type mechanism that is checked at compile time.

Some variable declarations and assignments:

```
int    lue = 42;
string awh = "Andrew was here";
Point  p = new Point(100,150);
Button b = new Button();
Label  l = new Label();
l.Text   = awh;
b.Top    = lue;
```

Note that int and string are primitive, scalar types – they are relatively simple.

The class Point represents a pair of numbers (often to specify a point on the screen) – in the example above we use the word new to construct an instance of the class.

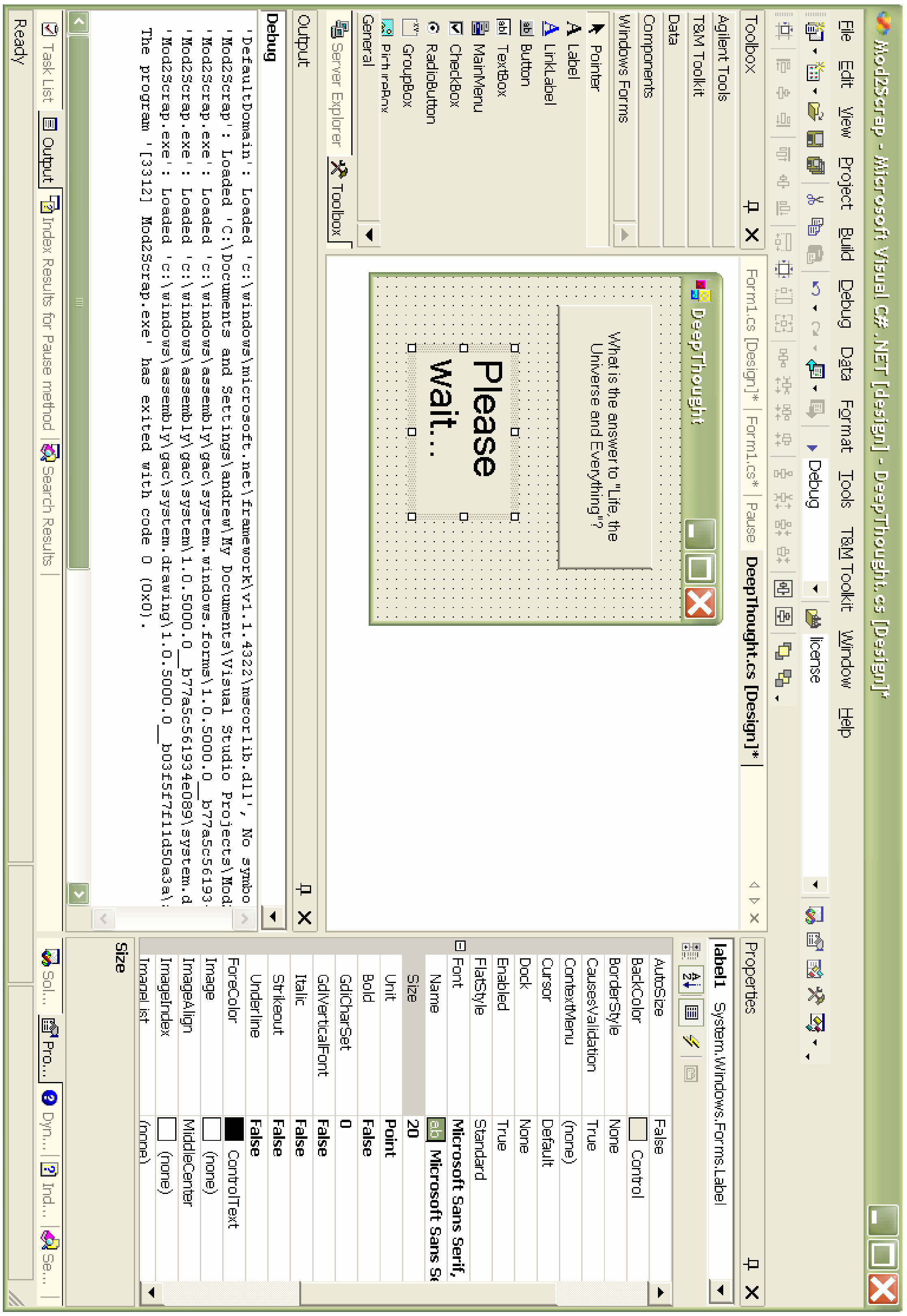
The classes Button and Label are much more complex. They each have many properties that control what they do and how they appear on a form.

2.2 Type Errors

The type checking mechanism prevents us from making a whole range of fundamental errors. It can be annoying – but it is a good thing that we are not allowed to do any of the following:

```
lue   = "Fourty two";
awh   = "Andrew was here" * 2;
p     = new Button();
l.Top = awh;
```

2.3 Visual Studio



label1 System.Windows.Forms.Label

AutoSize	False
BackColor	Control
BorderStyle	None
CausesValidation	True
ContextMenu	(none)
Cursor	Default
Dock	None
Enabled	True
FlatStyle	Standard
Font	Microsoft Sans Serif,
Name	Microsoft Sans Se
Size	20
Unit	Point
Bold	False
GdiCharSet	0
GdiVerticalFont	False
Italic	False
Strikeout	False
Underline	False
ForeColor	ControlText
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageList	(none)

Size

Debug

```
'DefaultDomain': Loaded 'c:\windows\microsoft.net\framework\v1.1.4322\mscorlib.dll', No symbo
'Mod2Scrap': Loaded 'C:\Documents and Settings\andrew\My Documents\Visual Studio Projects\Mod
'Mod2Scrap.exe': Loaded 'c:\windows\assembly\gac\system.windows.forms\1.0.5000.0__b77a5c5619
'Mod2Scrap.exe': Loaded 'c:\windows\assembly\gac\system\1.0.5000.0__b77a5c561934e089\system.d
'Mod2Scrap.exe': Loaded 'c:\windows\assembly\gac\system.drawing\1.0.5000.0__b03f5f7f11d50a3a\
The program '[3312] Mod2Scrap.exe' has exited with code 0 (0x0).
```

2.4 Some important elements of Visual Studio

2.4.1 The design surface

We can drag components from the toolbox onto the design surface.
Components include:

- Button
- Label
- TextBox
-
-
-
-
-

These components may be moved, resized, anchored or docked in a variety of ways.

2.4.2 The toolbox

Items from the toolbox may be included using drag, select and paint, double click.

2.4.3 The properties pane

Components on the design surface may be manipulated directly by dragging them around.
Components may also have their properties changed at design time from the properties panel. Properties of a component such as a button include:

- Left
- Width
- ForeColor
-
-
-

2.4.4 The Events properties

On the properties panel there are events. We can attach our own code to events so that we specify what happens when the user takes an action. Typical events that might be raised by a button:

- Click
-
-
-

2.5 Exercise One – Deep Thought.

The program Deep Thought calculates the answer to "Life, the Universe and Everything." I should display this answer.

- Create a new Windows Application in C#.
- Put a button and a label on the form.
- Change the properties of these components so that the text of the label is in a large friendly font and the button is enticing.
- The text for the label should be "Don't panic"
- Double click on the button; the code window should open:
    - Insert the following code

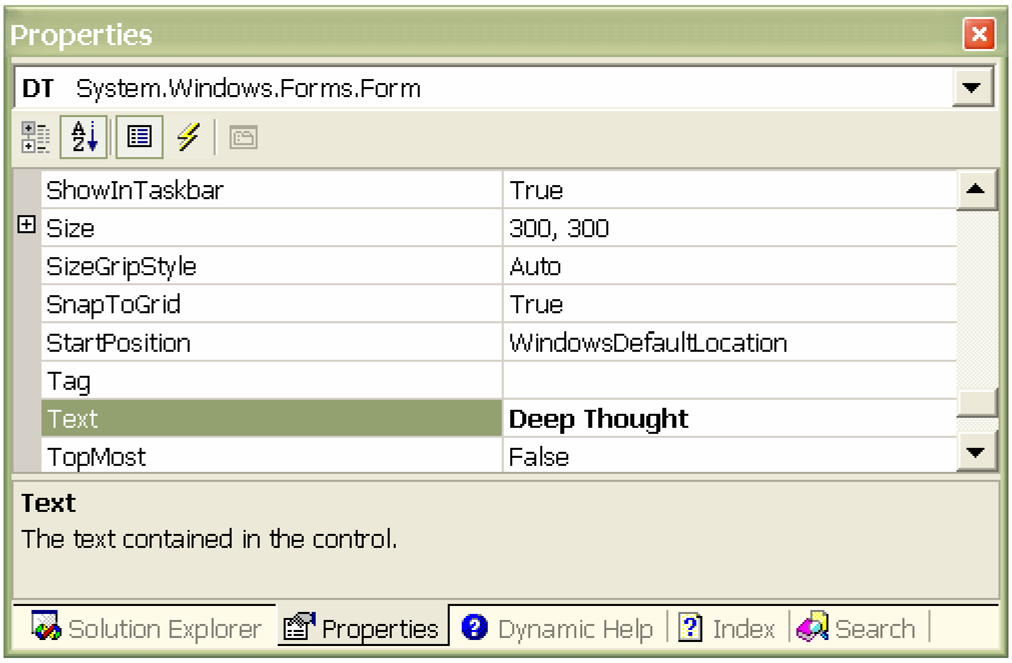
```
label1.Text = "42";
```

Run the application.

Terms used:

Name	Description	Notes
label1	A control	Also: a component, an object
button1	A component	
button1_Click	An event (more or less)	
label1.Text	A property of label1	
"42"	A string literal	
form1		
form1.Text		
form1.Width		
button1.Left		
button1.Location.X		

2.5.1 The Properties pane


Properties

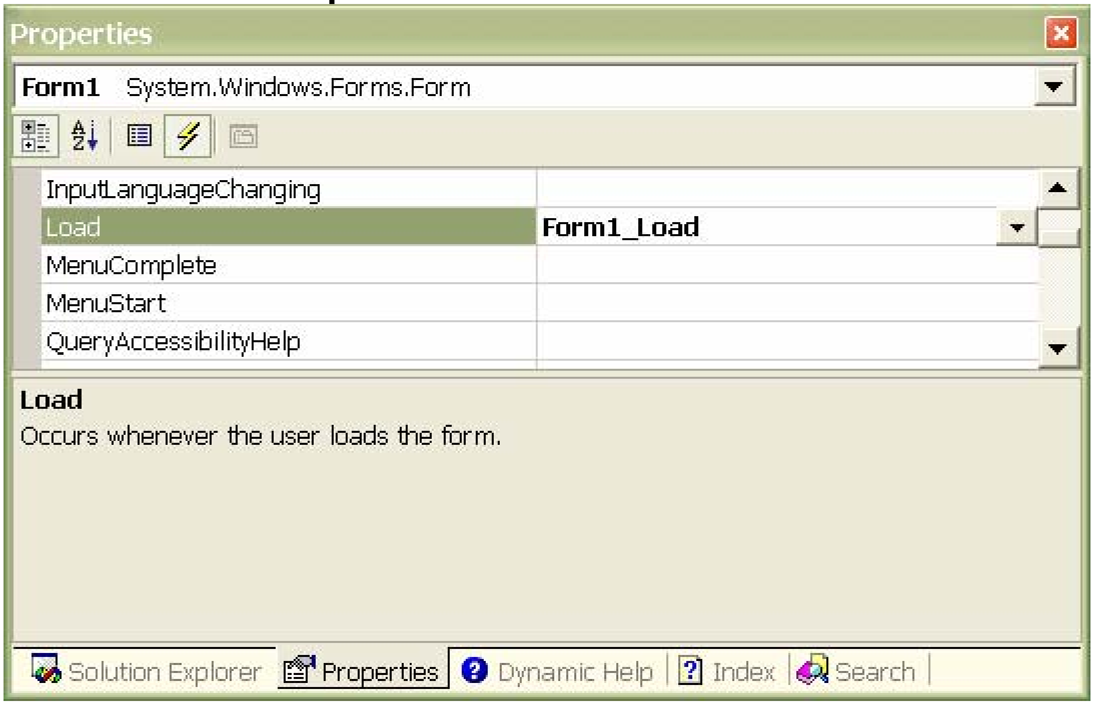
DT System.Windows.Forms.Form

ShowInTaskbar	True
⊞ Size	300, 300
SizeGripStyle	Auto
SnapToGrid	True
StartPosition	WindowsDefaultLocation
Tag	
Text	**Deep Thought**
TopMost	False

Text
The text contained in the control.

From the properties pane we can set values at design time.
Most of the properties that are visible may also be set a run time.

2.5.2 The Events pane


Properties

Form1 System.Windows.Forms.Form

InputLanguageChanging	
Load	Form1_Load
MenuComplete	
MenuStart	
QueryAccessibilityHelp	

Load
Occurs whenever the user loads the form.

From the events pane we can attach code to events that the user triggers.
The environment creates and "wires-up" procedures – we can fill in the procedure body.

```
private void button1_Click(object sender,
System.EventArgs e)
{
    label1.Text = "42";
}
```

2.6 Working with Graphics

We will be using the following methods of the Graphics class:

DrawLine	Overloaded. Draws a line connecting the two points specified by coordinate pairs.
FillPie	Overloaded. Fills the interior of a pie section defined by an ellipse specified by a pair of coordinates, a width, and a height and two radial lines.
FillPolygon	Overloaded. Fills the interior of a polygon defined by an array of points specified by Point structures.
FillRectangle	Overloaded. Fills the interior of a rectangle specified by a pair of coordinates, a width, and a height.
RotateTransform	Overloaded. Applies the specified rotation to the transformation matrix of this **Graphics** object.
ScaleTransform	Overloaded. Applies the specified scaling operation to the transformation matrix of this **Graphics** object by prepending it to the object's transformation matrix.
SetClip	Overloaded. Sets the clipping region of this **Graphics** object to the **Clip** property of the specified **Graphics** object.
TranslateTransform	Overloaded. Prepends the specified translation to the transformation matrix of this **Graphics** object.

A typical method is public void FillRectangle(Brush, int, int, int, int); - the brush dictates the colour and the integers specify the left, top, width and height of the rectangle to be drawn.

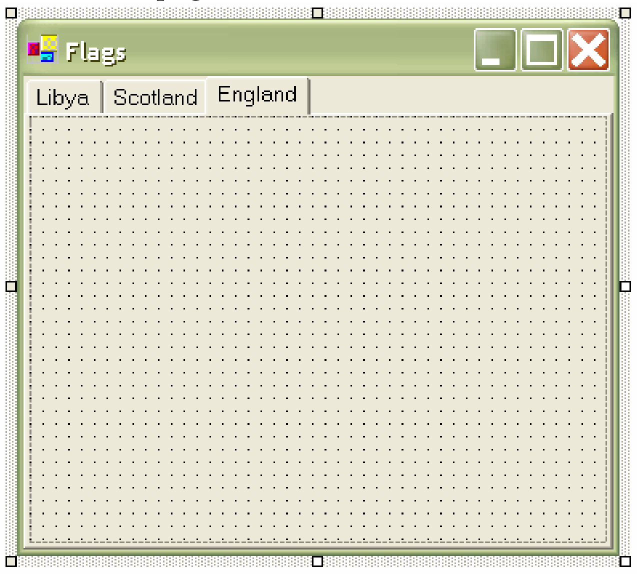
```
g.FillRectangle(Brushes.Green,0,0,500,300);
```

The system includes some standard brushes that we can access using the Brushes collection: Brushes.Green or Brushes.Blue for example. If none of these are suitable we can construct our own brush.
In order to draw a line we must specify a pen. There is a collection Pens that we may use: Pens.White for example.

2.7 Flags Tutorial

Create a new Windows C# project. Name the project Flags.

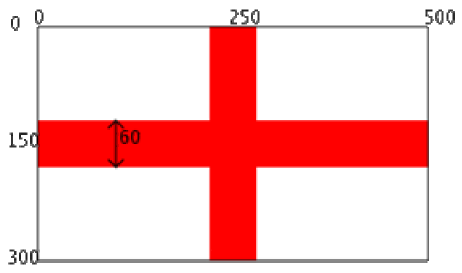
- Note that the form includes a tab control with two pages.
- Add a third page to the tab control – make the text of the new page read Libya.

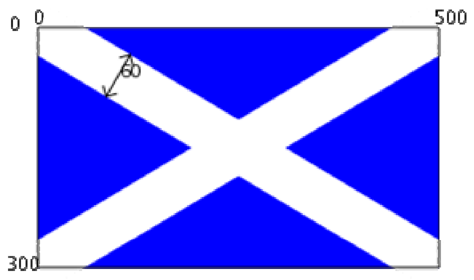


-
- Take care when setting this up – notice that you can select tabControl1 or tabPage1 or tabPage2...
- Set up the Paint event for tabPage1 – Libya. The flag for Libya is a green rectangle – we can draw that using the following code:

```
private void tabPage1_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.Green,0,0,500,300);
}
```

- Draw the flags for England (red on white) and Scotland (white on blue) in a similar manner





2.7.1 Drawing the Cross of St. George: Step-By-Step

We need a white, rectangular background, we can use one of the pre-built brushes for this:

```
private void tabPage2_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White,0,0,500,300);
}
```

To draw the cross we could use red rectangles – but we prefer to draw thick lines.

```
Graphics g = e.Graphics;
g.FillRectangle(Brushes.White,0,0,500,300);
Pen p = new Pen(Brushes.Red,60);
g.DrawLine(p,0,150,500,150);
g.DrawLine(p,250,0,250,300);
```

In the above code we create our own custom pen and call it p. We then use the pen twice and "throw it away".

Notice the lifetime of the objects we are using:

- Graphics g – this is a large, expensive object.
    - We do not create this object explicitly, it was created by the Paint operation, the PaintEventArgs e gives us convenient access to it.
- Brushes.White – this is a pre-built system object.
    - There are 140 odd system defined brushes – from AliceBlue to YellowGreen.
- Pen p – this object is created by our code.
    - When p goes out of scope it will be reclaimed by creating this object for every paint event we are being inefficient – but a pen is cheap so we don't care.

2.7.2 The Cross of St. Andrew

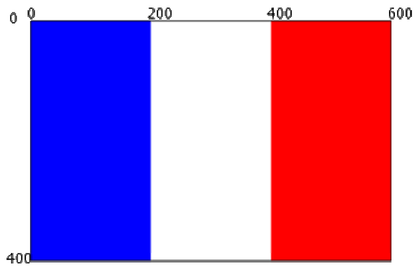
Drawing the diagonal cross should be easy however you may wish to:

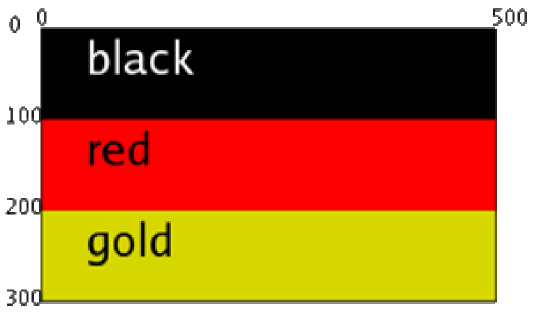
- Set a clip rectangle before drawing the lines – this should prevent the ends of the lines extending outside the flag boundary.
- Set the Smoothing method to AntiAlias – this gives diagonal lines a better appearance.

```
g.SmoothingMode=System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
```

2.7.3 More Easy Flags

Add additional pages to draw the flags of France (blue, white, red) and Germany (black, red, gold) as shown:





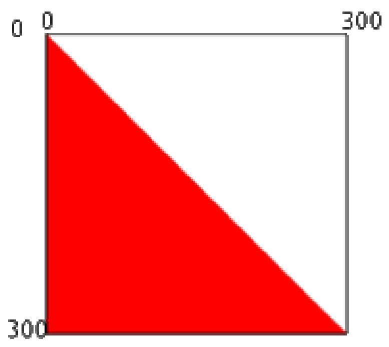
2.7.4 FillPolygon

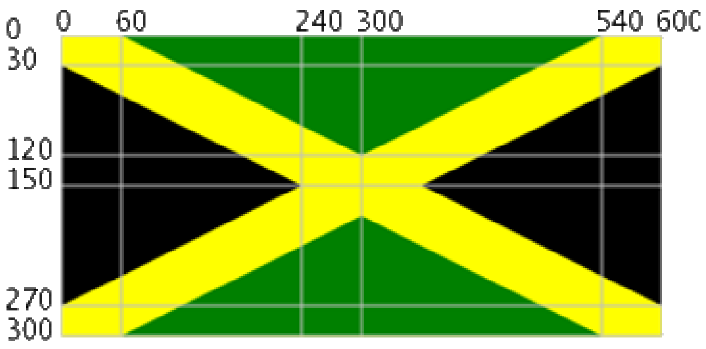
We can define a polygon as an array of Point. The Graphics method FillPolygon may be used.

```
Graphics g = e.Graphics;
g.FillRectangle(Brushes.White,0,0,300,300);
Point [] tri = {new Point(0,0),
                new Point(300,0),
                new Point(0,300)};
g.FillPolygon(Brushes.Red,tri);
```

In the example we define an array of points and call it tri.
These are **not** the right vertices for the Napier triangle.

- Create additional pages to draw the Napier triangle and the Jamaican flag.





Note that each of the four triangles on the Jamaican flag must have a different name. Suitable names might be ltri, ttri, rtri and btri (for left, top, right, bottom).
Napier is red and white. Jamaica is black, green and yellow.

2.7.5 TranslateTransform

We can dictate where things get drawn by performing the TranslateTransform. This has the effect of "shifting the origin" for all following operations.
This can be useful when the same polygon needs to be drawn in different places.

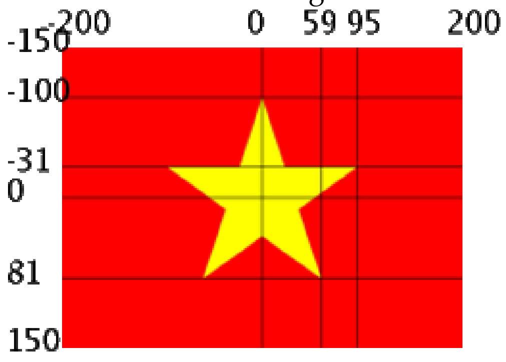
2.7.6 Vietnam:

The five-pointed star is a common motif. A star with radius 100 and center at (0,0) is given below:

```
Point [] star = {
                    new Point(0,-100),
                    new Point(59,81),
                    new Point(-95,-31),
                    new Point(95,-31),
                    new Point(-59,81)
                };
```

The flag of Vietnam requires this star to be drawn at coordinates (200,150) – we can draw the star as defined above – saving us tedious arithmetic.

Yellow star on red background.



```
Graphics g = e.Graphics;
g.TranslateTransform(200,150);
g.FillRectangle(Brushes.Red,-200, -150,400,300);
g.FillPolygon(Brushes.Yellow,star,
  System.Drawing.Drawing2D.FillMode.Winding);
```

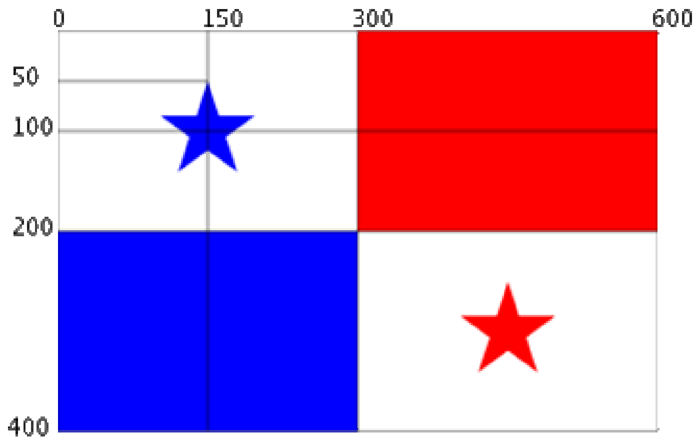
You may be curious to know the purpose the Winding value…

2.7.7 ScaleTransform

We can set a scale transform. Following the scale transform all drawing operations are bigger or smaller depending on the values given.

The Vietnamese star is too large for the flag of Panama – on the Panamanian flag the radius of the star should be half as big.

The rectangle and star on the left is blue, on the right both are red.



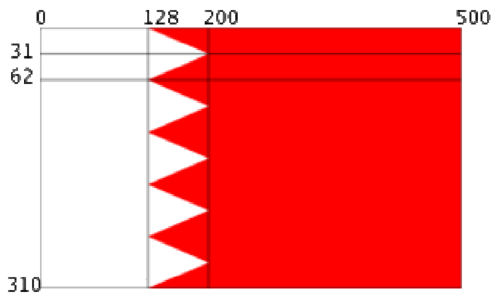
Note that the ScaleTransform method requires floating point values – we include the letter f after each number to indicate that this it is of type float.

```
g.TranslateTransform(150,100);
g.ScaleTransform(.5f,.5f);
g.FillPolygon(Brushes.Blue,star,
    System.Drawing.Drawing2D.FillMode.Winding);
```

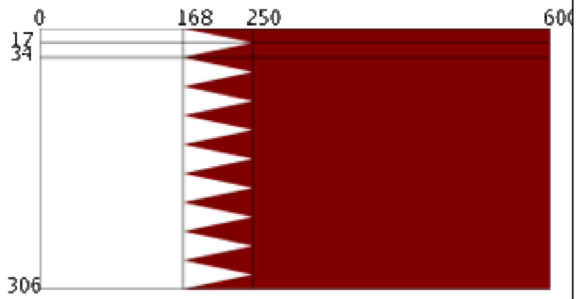
Be warned that to draw the red star we will have to translate twice as far – the ScaleTransform effects **all** subsequent operations.

2.8 Some harder flags

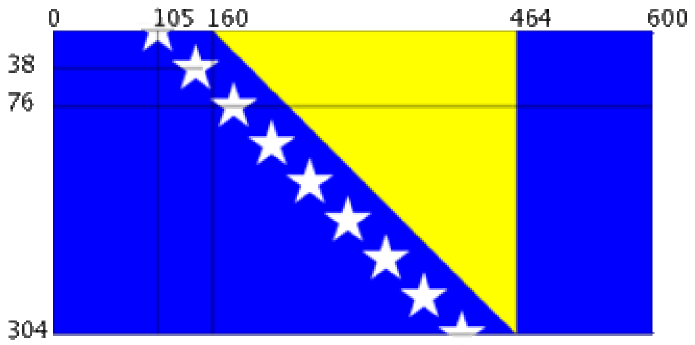
Bahrain, white and red.
5 pillars



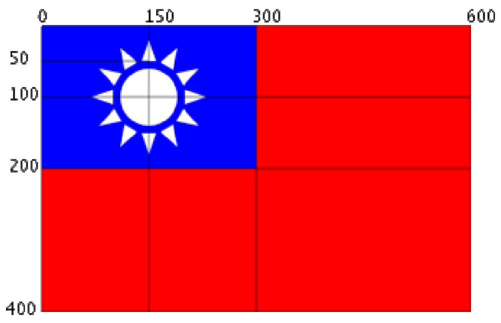
Qatar, white and maroon
9 pillars



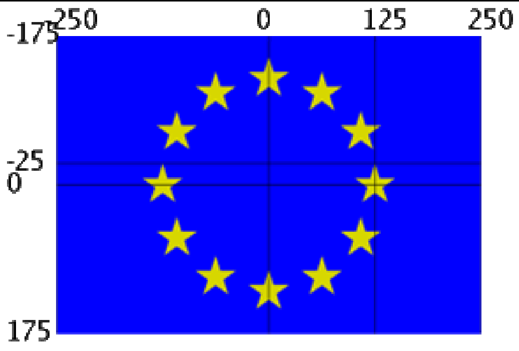
Bosnia Herzigovina, Yellow triangle, white stars, blue background.
9 stars, first and last are clipped. Each star is 38 left and 38 below the previous.



The flag of Taiwan involves 12 triangles arranged in a circle. Each triangle has a base 20 and height 30. Red background, blue top left corner, white circle and triangles.



European Union, gold stars on blue.
12 stars of radius 25 in a circle radius 125



Some other countries with interesting and instructive flags: USA, Namibia

Help with Bahrain

With flags such as the Bahrain we want to repeat the same shape several times.
If we are a little bit careful we can simply copy and paste the same phrase over and over:

```
Graphics g = e.Graphics;
Point [] pillar = {new Point(0,0),
                   new Point(128,0),
                   new Point(200,31)
                   //...
                   };
g.FillRectangle(Brushes.Red,0,0,500,310);

g.FillPolygon(Brushes.White,pillar);
g.TranslateTransform(0,62);
g.FillPolygon(Brushes.White,pillar);
g.TranslateTransform(0,62);
g.FillPolygon(Brushes.White,pillar);
g.TranslateTransform(0,62);
g.FillPolygon(Brushes.White,pillar);
g.TranslateTransform(0,62);
g.FillPolygon(Brushes.White,pillar);
g.TranslateTransform(0,62);
```

Although this takes little effort to produce it will lead to code that is difficult to maintain. Also it can work fine for 5 or 50 repetitions but would be painful for a thousand.
We can make use of a **for** loop to repeat sections of code. The simplest **for** loop takes the form:

```
for (int i=0;i<5;i++)
{
    g.FillPolygon(Brushes.White,pillar);
    g.TranslateTransform(0,62);
}
```

2.8.1 Help with European Union

To draw the flag of the European Union we should draw the background then shift the origin to the center of the flag.

To draw a star at an angle of 30 degrees we can do the following:

- Rotate 30
- Move up 125
- Scale by .25
- Draw a star
- Scale by 4
- Move down 125
- Rotate –30

Notice that each of the transformations gets undone in the opposite order – this is important as at the end of the sequence the Graphic is left in the same position as when it started.

```
Graphics g= e.Graphics;
g.FillRectangle(Brushes.Blue,0,0,500,350);
g.TranslateTransform(250,175);

g.RotateTransform(30);
g.TranslateTransform(0,-125);
g.ScaleTransform(.25f,.25f);
g.FillPolygon(Brushes.Yellow,star);
g.ScaleTransform(4f,4f);
g.TranslateTransform(0,125);
g.RotateTransform(-30);
```

You will notice two problems with this. The stars are rotated and they have the wrong "FillMode".

- Rotate 30
- Move up 125
- Scale by .25
- Rotate -30
- Draw a star
- Rotate 30
- Scale by 4
- Move down 125
- Rotate –30

We can use a **for** loop to do this 12 times.

```
    for (float i=0;i<360;i+=30)
```

2.9 Summary

- The visual studio environment includes a wysiwyg design surface.
- Help is conveniently located, we can search for help and use "dynamic help".
- We can declare variables. Variables have type.
- The .NET framework includes hundreds of classes some of these are complicated.
    - Visual components include Button, Label, TextBox
    - The Graphics class includes methods such as DrawLine and FillRectangle.
    - We can find the names and the types of the arguments of a method from intellisense.