Windows Forms

Windows Forms

- Programming model used to write GUI applications for the .NET Framework
- Look like ordinary Windows applications
  - Windows
  - Incorporate common GUI elements
    - Menus
    - Controls
    - Dialog boxes
- They’re managed applications

Windows Applications

- Event-driven
  - Wait for system to deliver input
  - React to input
  - Return control back to system
- How do the System and Windows application communicate?
  - Windows messages

Windows Programming

- When user moves mouse, system sends messages to application
- A special function called the window procedure processes the message
- What things, other than user interaction, can create a message?
  - Power management

Message Routing

- Two ways
  - Send message to a first-in first-out queue
    - System examines the message to determine which window is supposed to process the message
    - Send the message to the proper queue
    - From queue, message is sent to the window procedure
  - Send message directly to windows procedure
    - Examples include resizing and window activation

Window Applications

- Each application is made up of several forms
- Each form will have a window procedure to process the messages sent to it
- The process that sends the messages to the windows procedure is known as the message pump

Messages

- There are two types of messages
  - System defined
  - Application defined

Messages

- To identify the message, four parameters are sent with it
  - Window handle: long value used to identify a specific window. Also called hWnd.
  - Message id: named constant to identify the message. Example is WM_PAINT that tells the window it needs to repaint itself
  - lParam: could contain anything
  - wParam: could contain anything

MFC

- MFC is an object-oriented library built using C++ that represents basic objects used to build windows applications
- MFC is organized into a class hierarchy

Modal and Modeless Forms

- A modal form needs to be closed to access any other form in an application
  - The modal form takes over the execution of the program and returns only when it is closed
- A **modeless** form in contrast does not block the execution of the program

Project (modeless) on Turing

```
using System.Windows.Forms;
public class MyForm : Form
{
  public MyForm()
  {
    this.Text = "Hello world";
    this.Size = new System.Drawing.Size(500, 500);
    this.FormBorderStyle = FormBorderStyle.Fixed3D;
  }
  static void HandleClosing(object sender,
           System.ComponentModel.CancelEventArgs e)
  {
    MessageBox.Show("The form is closing");
  }
  static void Main()
  {
    MyForm myForm = new MyForm();
    myForm.Closing +=
      new System.ComponentModel.CancelEventHandler(HandleClosing);
    Application.Run(myForm);
  }
}
```

Project (modal) on Turing

```
public class WarningForm : Form
{
  public WarningForm()
  {
    Button yesButton = new Button();
    Button noButton = new Button();
    yesButton.Text = "Yes";
    yesButton.Location = new System.Drawing.Point(10, 10);
    noButton.Text = "No";
    noButton.Location = new System.Drawing.Point(yesButton.Right +
         10, yesButton.Top);
    yesButton.DialogResult = DialogResult.Yes;
    noButton.DialogResult = DialogResult.No;
    this.Text = "Are you sure you want to close?";

    this.Size = new System.Drawing.Size(300, 100);

    this.Controls.Add(yesButton);
    this.Controls.Add(noButton);
  }
}
```

Project (modal) on Turing

```
public class MyForm : Form
{
  public MyForm()
  {
    this.Text = "Hello world";
    this.Size = new System.Drawing.Size(500, 500);
    this.FormBorderStyle = FormBorderStyle.Fixed3D;
  }

  static void HandleClosing(object sender,
    System.ComponentModel.CancelEventArgs e)
  {
    WarningForm modalForm = new WarningForm();
    if(modalForm.ShowDialog() == DialogResult.No)
    {
      e.Cancel = true;
    }
    else
    {
    }
  }
```

Project (modal) on Turing

```
static void Main()
  {
    MyForm myForm = new MyForm();

    myForm.Closing +=
      new
     System.ComponentModel.CancelEventHandler(HandleClosing);

    Application.Run(myForm);
  }
```

Drawing in a Form

- Graphics is the main class used to draw in forms
- The method OnPaint redraws or repaints the form

Project (painting) on Turing

```
protected override void OnPaint( PaintEventArgs e )
   {
     Pen pen = new Pen( Color.Black );
     e.Graphics.DrawRectangle( pen, 10, 10, 390, 90 );
     SolidBrush solid = new SolidBrush( Color.Red );
     e.Graphics.FillRectangle( solid, 10, 110, 390, 90 );
     e.Graphics.DrawRectangle( pen, 10, 110, 390, 90 );
     Rectangle rect = new Rectangle( 10, 210, 390, 90 );
     LinearGradientBrush gradient = new
         LinearGradientBrush( rect, Color.Thistle,
            Color.Tomato, LinearGradientMode.Horizontal );
     e.Graphics.FillRectangle( gradient, rect );
     e.Graphics.DrawRectangle( pen, rect );
     pen.Dispose();
     solid.Dispose();
   }
```

Coordinates and Transformations

- x and y coordinates are used to determine the location of the drawing objects
- 0, 0 is the top, left point
- These are world coordinates
- Translation moves an object to another location
- Transform moves the object by an angle

Project (transformations) on Turing

```
protected override void OnPaint( PaintEventArgs e )
   {
     SolidBrush brush = new SolidBrush( Color.Red );
     e.Graphics.TranslateTransform( 100.0f, 100.0f );
     e.Graphics.RotateTransform( -30.0f );
     e.Graphics.FillRectangle( brush, 0, 0, 200, 100 );
     brush.Dispose();
   }
```

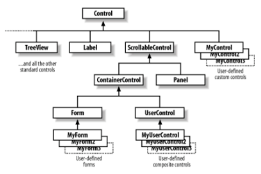
Project (clock) on Turing

- See handout
- What happens if we delete line 10?
- On line 32, why is hour %12?
- On line 26, why is the rectangle drawn at location 85, -5?
- On line 68, what are the points being passed into the FillPolygon function?

Controls

- The System.Windows.Forms namespace defines a class called Control
- Any visual element of an application—whether it is a window, a button, a toolbar, or a custom user-defined control—is represented by an object of some class deriving from Control

Controls



Designer

- When creating forms in the design view, code is generated for us
- Adding a button will generate

```
private System.Windows.Forms.Button button1;
...
private void InitializeComponent()
{
    ...
    this.button1 = new System.Windows.Forms.Button();
    ...
    this.button1.Location = new
                  System.Drawing.Point(8, 8);
    this.button1.Name = "button1";
    this.button1.Size = new
                  System.Drawing.Size(104, 32);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    ...
}
```

Docking and Anchoring

- When adding controls to a form in the design view, the locations are fixed
  - Try resizing the form when it's running
- By anchoring, you can fix the control in any of the four directions (N, S, E, W). The control will not move in that direction when resizing
- You can use the dock to attach the control to any of the four directions (N, S, E, W)

Project (Windows Forms) on Turing

- Let's work on the above project