IAP C# Lecture 4
Misc Syntax, then start Windows Presentation Foundation

Geza Kovacs

- Read number and tells you if it’s odd or even

```
using System;
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a number");
        string entered = Console.ReadLine();
        int number = int.Parse(entered);
        if (number % 2 == 0)
            Console.WriteLine("even number entered");
        else
            Console.WriteLine("odd number entered");
    }
}
```

- What if I enter something that isn't a number?
  - System.FormatException is thrown

```
using System;
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a number");
        string entered = Console.ReadLine();
        int number = int.Parse(entered);
        if (number % 2 == 0)
            Console.WriteLine("even number entered");
        else
            Console.WriteLine("odd number entered");
    }
}
```

Exceptions

- To indicate that you've encountered an error, throw a Exception instance (or some subclass)

```
static int parseDigit(string s) {
    if (s == "0") return 0;
    if (s == "1") return 1;
    if (s == "2") return 2;
    if (s == "3") return 3;
    if (s == "4") return 4;
    if (s == "5") return 5;
    if (s == "6") return 6;
    if (s == "7") return 7;
    if (s == "8") return 8;
    if (s == "9") return 9;
    throw new Exception("not a digit from 0 to 9");
}
```

Exceptions

- To indicate that you've encountered an error, throw a Exception instance (or some subclass)

```
class NotDigitException : Exception {}

    static int parseDigit(string s) {
        if (s == "0") return 0;
        if (s == "1") return 1;
        if (s == "2") return 2;
        if (s == "3") return 3;
        if (s == "4") return 4;
        if (s == "5") return 5;
        if (s == "6") return 6;
        if (s == "7") return 7;
        if (s == "8") return 8;
        if (s == "9") return 9;
        throw new NotDigitException();
    }
```

- Use try-catch blocks to handle exceptions

```
using System;
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a number");
        string entered = Console.ReadLine();
        int number = 0;
        try {
            int.Parse(entered);
        } catch (FormatException e)  {
            Console.WriteLine("you didn't enter a number");
            return;
        }
        if (number % 2 == 0)
            Console.WriteLine("odd number entered");
        else
            Console.WriteLine("even number entered");
    }
}
```

- Can handle all exceptions in a single catch block
  - All exceptions subclass Exception

```
using System;
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a numerator and denominator");
        string num = Console.ReadLine();
        string den = Console.ReadLine();
        int result = 0;
        try {
            result = int.Parse(num) / int.Parse(den);
        } catch (Exception e) {
            Console.WriteLine(“bad input");
            return;
        }
        Console.WriteLine(result);
    }
}
```

- Or, handle each type of exception individually

```
using System;
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a numerator and denominator");
        string num = Console.ReadLine();
        string den = Console.ReadLine();
        int result = 0;
        try {
            result = int.Parse(num) / int.Parse(den);
        } catch (FormatException e) {
            Console.WriteLine("you didn't enter a number");
            return;
        } catch (DivideByZeroException e) {
            Console.WriteLine("can't divide by zero");
            return;
        }
        Console.WriteLine(result);
    }
}
```

- Or, don’t catch the exception at all (no “checked exceptions” like in Java)
  - If an unhandled exception gets thrown, you application will just crash

```
static class MyMainClass
{
    static void Main(string[] args)
    {
        Console.WriteLine("enter a numerator and denominator");
        string num = Console.ReadLine();
        string den = Console.ReadLine();
        int result = int.Parse(num) / int.Parse(den);
        Console.WriteLine(result);
    }
}
```

enum

- Defines a type which can take one of several predefined values

```
enum Directions
{
    North, South, East, West
}
```

- Can declare an instance of an enum type

```
using System;
enum Directions { North, South, East, West }

static class MyMainClass
{
    static void Main(string[] args)
    {
        Directions x = Directions.North;
        Console.WriteLine(IsVertical(x)); // North
    }
}
```

- Can pass them to methods

```
using System;
enum Directions { North, South, East, West }

static class MyMainClass
{
    static bool IsVertical(Directions d) {
        if (d == Directions.North || d == Directions.South)
            return true;
        return false;
    }
    static void Main(string[] args)
    {
        Directions x = Directions.North;
        Console.WriteLine(IsVertical(x)); // True
    }
}
```

- Can add extension methods (but not regular methods)

```
using System;
enum Directions { North, South, East, West }

static class MyMainClass
{
    static bool IsVertical(this Directions d) {
        if (d == Directions.North || d == Directions.South)
            return true;
        return false;
    }
    static void Main(string[] args)
    {
        Directions x = Directions.North;
        Console.WriteLine(x.IsVertical()); // True
    }
}
```

Extension method

- Underlying implementation of enum uses integers

```
using System;
enum Directions {
    North, South, East, West
}

static class MyMainClass
{
    static void Main(string[] args)
    {
        Directions x = Directions.North + 1;
        Console.WriteLine(x); // South
    }
}
```

- Underlying implementation of enum uses integers

Underlying datatype can be changed to
byte, sbyte, short, ushort, int, uint, long, and ulong

```
using System;
enum Directions : byte {
    North, South, East, West
}

static class MyMainClass
{
    static void Main(string[] args)
    {
        Directions x = Directions.North + 1;
        Console.WriteLine(x); // South
    }
}
```

- Underlying implementation uses integers
  - Values start at 0 by default

```
using System;
enum Directions { North, South, East, West }

static class MyMainClass
{
    static void Main(string[] args)
    {
        Directions x = (Directions)1;
        Console.WriteLine(x); // South
    }
}
```

- Underlying implementation uses integers
  - Values start at 0 by default

Can change start value

```
using System;
enum Directions { North = 100, South, East, West }

static class MyMainClass
{
    static void Main(string[] args)
    {
        var x = (Directions)102;
        Console.WriteLine(x); // East
    }
}
```

- Flagged enum: can take on multiple values
  - Use bitwse OR to combine values

```
[Flags]
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
       var x = Directions.North | Directions.East;
       Console.WriteLine(x); // North, East
    }
}
```

- Flagged enum: can take on multiple values
  - Use HasFlag() to check for an individual flag

```
[Flags]
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
        var x = Directions.North | Directions.East;
        Console.WriteLine(x.HasFlag(Directions.North));
        // True
    }
}
```

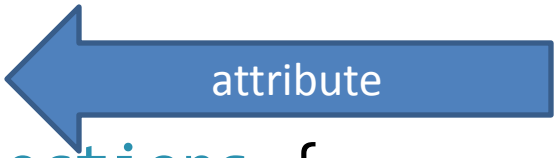
- Flagged enum: can take on multiple values
  - Use HasFlag() to check for an individual flag

```
[Flags]
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
        var x = Directions.North | Directions.East;
        Console.WriteLine(x.HasFlag(Directions.East));
        // True
    }
}
```

- Flagged enum: can take on multiple values
  - Use HasFlag() to check for an individual flag

```
[Flags]
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
        var x = Directions.North | Directions.East;
        Console.WriteLine(x.HasFlag(Directions.South));
        // False
    }
}
```

- When making a flagged enum, need [Flags] attribute
  - Attribute: metadata associated with a type, class, method, etc




```
[Flags]
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
       var x = Directions.North | Directions.East;
       Console.WriteLine(x); // North, East
    }
}
```

- When making a flagged enum, need [Flags] attribute
  - Attribute: metadata associated with a type, class, method, etc
  - Without [Flags] attribute, various operations will fail

```
enum Directions {
    North = 1 << 0,
    South = 1 << 1,
    East = 1 << 2,
    West = 1 << 3
}
static class MyMainClass {
    static void Main(string[] args) {
        var x = Directions.North | Directions.East;
        Console.WriteLine(x); // 5
    }
}
```

Multicasting

- Consider the following problem: I have a Messenger who needs to send some message (string) to a number of listeners
- To represent a listener, use a delegate: Action<string> (has 1 string argument)

```
delegate void Action<T>(T arg1);
```

```
using System;
using System.Collections.Generic;
class Messenger {
    LinkedList<Action<string>> listeners = new LinkedList<Action<string>>();
    public void AddListener(Action<string> newListener) {
        listeners.AddLast(newListener);
    }
    public void SendMessage(string message) {
        foreach (Action<string> x in listeners)
            x(message);
    }
}
```

- Approach 1: a linked list of delegates

```
using System;
using System.Collections.Generic;
class Messenger {
    LinkedList<Action<string>> listeners = new LinkedList<Action<string>>();
    public void AddListener(Action<string> newListener) {
        listeners.AddLast(newListener);
    }
    public void SendMessage(string message) {
        foreach (Action<string> x in listeners)
            x(message);
    }
}
static class MyMainClass {
    static void Main(string[] args) {
        Messenger m = new Messenger();
        m.AddListener((s) => {
            Console.WriteLine("Listener1:"+s);
        });
        m.AddListener((s) => {
            Console.WriteLine("Listener2:" + s);
        });
        m.SendMessage("some message");
    }
}
```

- Approach 1: a linked list of delegates

A Better Approach using Delegates

- In addition to using “=” for delegates:

```
Action<string> messenger = (s) => {
    Console.WriteLine(s);
}
messenger("someMessage");
```

- Delegates also support “+=” (subscribe); multicasts calls to all methods that have subscribed:

```
Action<string> messenger = null;
messenger += (s) => { Console.WriteLine("Listener1:" + s); };
messenger += (s) => { Console.WriteLine("Listener2:“ + s); };
messenger("someMessage");
```

A Better Approach using Delegates

- Use “-=” (unsubscribe) to remove methods from those to which the delegate will multicast to

```
static class MyMainClass {
    static void Main(string[] args) {
        Action<string> messenger = null;
        Action<string> listener1 = (s) => {
            Console.WriteLine("Listener1:" + s); };
        Action<string> listener2 = (s) => {
            Console.WriteLine("Listener2:" + s); };
        messenger += listener1;
        messenger += listener2;
        messenger("first message");
        messenger -= listener2;
        messenger("second message");
    }
}
```

event (used extensively in WPF)

- A modifier for delegates, which makes the following changes:
  - Can only be subscribed (+=) or unsubscribed (-=) from, not assigned to
  - Can be part of a class instance or in an interface, but not declared locally
  - Can be invoked only within the class

- Using events

```
using System;
class Messenger {
    public event Action<string> MessageEvent;
    public void SendMessage(string message) {
        MessageEvent(message);
    }
}

static class MyMainClass {
    static void Main(string[] args) {
        Messenger m = new Messenger();
        m.MessageEvent += (s) => {
            Console.WriteLine("Listener1:" + s);
        };
        m.MessageEvent += (s) => {
            Console.WriteLine("Listener2:" + s);
        };
        m.SendMessage("some message");
    }
}
```

- Using events
  - Events can be in interfaces

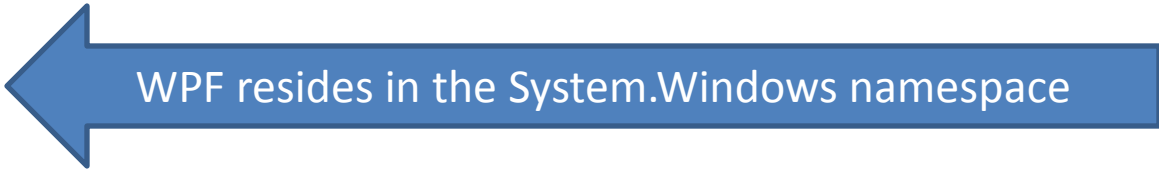
```
using System;
interface IMessenger {
    event Action<string> MessageEvent;
    void SendMessage(string message);
}
class Messenger {
    public event Action<string> MessageEvent;
    public void SendMessage(string message) {
        MessageEvent(message);
    }
}
static class MyMainClass {
    static void Main(string[] args) {
        IMessenger m = new Messenger();
        m.MessageEvent += (s) => {
            Console.WriteLine("Listener1:" + s);
        };
        m.MessageEvent += (s) => {
            Console.WriteLine("Listener2:" + s);
        };
        m.SendMessage("some message");
    }
}
```

What is WPF?

- Is a **library** for building GUIs on Windows, Windows Phone 7, and Silverlight
  - Library: a collection of classes that are available to you in a compiled .dll file (.NET calls this an **assembly**)
  - You allow your application to use libraries by adding a reference to them (Project -> Add Reference)
- Before using WPF, need to add a reference to the following assemblies:
  - PresentationFramework
  - PresentationCore
  - WindowsBase
  - System.Xaml

Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```



Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```

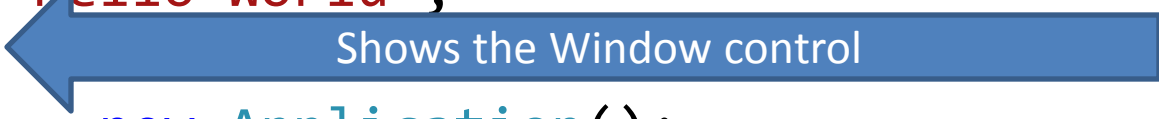
STAThread: attribute having to do with threading model in COM, need to have this attribute in Main method of WPF applications

Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```

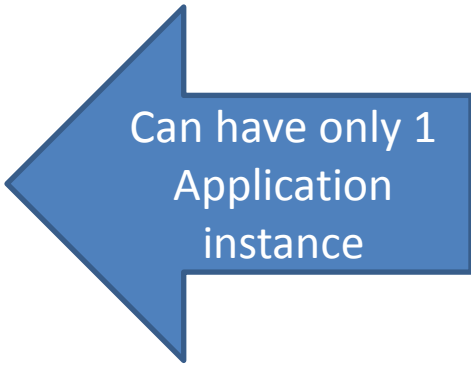
Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```



Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```



Hello World in WPF

```
using System;
using System.Windows;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Application app = new Application();
        app.Run();
    }
}
```

Starts WPF event loop, blocks until all windows are closed

Hello World with a Button

```
using System;
using System.Windows;
using System.Windows.Controls;

static class MyMainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Button button = new Button();
        button.Content = "Click Me";
        button.FontSize = 32.0;
        window.Content = button;
        Application app = new Application();
        app.Run();
    }
}
```

Button is in namespace System.Windows.Controls

Subscribing to the Button's Click event

- class Button has as a member, a Click event:

```
public event RoutedEventHandler Click;
```

- RoutedEventHandler is in turn a delegate type:

```
delegate void RoutedEventHandler(object sender, RoutedEventArgs e);
```

- Where **sender** is the instance which sent the event (in this case, the button), and RoutedEventArgs e stores info about how the event was relayed across the GUI
- Subscribe to events using the "+=" notation

Subscribing to the Button's Click event

```
using System;
using System.Windows;
using System.Windows.Controls;

static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Button button = new Button();
        button.Content = "Click Me";
        button.FontSize = 32.0;
        button.Click += (object o, RoutedEventArgs e) => {
            Console.WriteLine("button was clicked");
        };
        window.Content = button;
        Application app = new Application();
        app.Run();
    }
}
```

Printing Value of Slider while Sliding

```
using System;
using System.Windows;
using System.Windows.Controls;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Slider slider = new Slider();
        slider.Minimum = 0;
        slider.Maximum = 100;
        slider.ValueChanged +=  (o, e) => {
            Console.WriteLine(slider.Value);
        };
        window.Content = slider;
        Application app = new Application();
        app.Run();
    }
}
```

Printing Contents of TextBox when changed

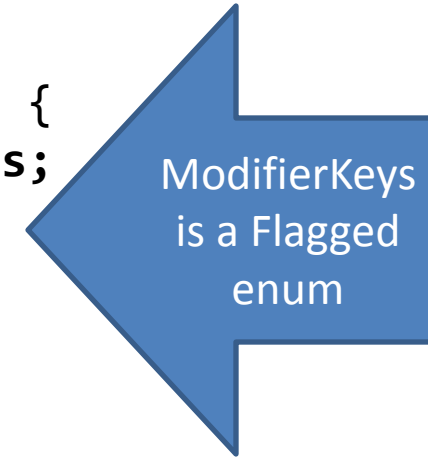
```
using System;
using System.Windows;
using System.Windows.Controls;

static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.TextChanged += (object o, TextChangedEventArgs e) => {
            Console.WriteLine(textBox.Text);
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```

Printing Contents of TextBox when Return pressed

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.KeyDown += (object o, KeyEventArgs e) => {
            if (e.Key == Key.Return) {
                    Console.WriteLine(textBox.Text);
            }
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```

Printing Contents of TextBox when Ctrl-Shift-Return pressed

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.KeyDown += (object o, KeyEventArgs e) => {
            ModifierKeys mod = e.KeyboardDevice.Modifiers;
            if (mod.HasFlag(ModifierKeys.Control) &&
                mod.HasFlag(ModifierKeys.Shift) &&
                e.Key == Key.Return) {
                    Console.WriteLine(textBox.Text);
            }
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```



- Suppose we instead want our shortcut to be Ctrl-Z: problem, already handled by undo

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.KeyDown += (object o, KeyEventArgs e) => {
            ModifierKeys mod = e.KeyboardDevice.Modifiers;
            if (mod.HasFlag(ModifierKeys.Control) &&
                e.Key == Key.Z) {
                    Console.WriteLine(textBox.Text);
            }
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.PreviewKeyDown += (object o, KeyEventArgs e) => {
            ModifierKeys mod = e.KeyboardDevice.Modifiers;
            if (mod.HasFlag(ModifierKeys.Control) &&
                e.Key == Key.Z) {
                    Console.WriteLine(textBox.Text);
            }
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```

- Suppose we instead want our shortcut to be Ctrl-Z: problem, already handled by undo
  - Use PreviewKeyDown event to get to it before undo can

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        TextBox textBox = new TextBox();
        textBox.PreviewKeyDown += (object o, KeyEventArgs e) => {
            ModifierKeys mod = e.KeyboardDevice.Modifiers;
            if (mod.HasFlag(ModifierKeys.Control) &&
                e.Key == Key.Z) {
                    Console.WriteLine(textBox.Text);
                    e.Handled = true;
            }
        };
        window.Content = textBox;
        Application app = new Application();
        app.Run();
    }
}
```

- Suppose we instead want our shortcut to be Ctrl-Z: problem, already handled by undo
  - Use PreviewKeyDown event to get to it before undo can
  - Set Handled property to true to ensure undo ignores the event

```
using System;
using System.Windows;
using System.Windows.Controls;

static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Button button1 = new Button();
        button1.FontSize = 36.0;
        button1.Content = "Button 1";
        Button button2 = new Button();
        button2.FontSize = 36.0;
        button2.Content = "Button 2";
        StackPanel panel = new StackPanel();
        panel.Children.Add(button1);
        panel.Children.Add(button2);
        window.Content = panel;
        Application app = new Application();
        app.Run();
    }
}
```

Use a Layout for showing multiple items (ex: StackPanel for displaying 2 buttons)

```
using System;
using System.Windows;
using System.Windows.Controls;

static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Slider slider = new Slider();
        slider.Minimum = 0;
        slider.Maximum = 100;
        TextBox textBox = new TextBox();
        StackPanel panel = new StackPanel();
        panel.Children.Add(slider);
        panel.Children.Add(textBox);
        window.Content = panel;
        Application app = new Application();
        app.Run();
    }
}
```

Use a Layout for showing multiple items (ex: StackPanel for displaying a slider and a TextBox)

```
using System;
using System.Windows;
using System.Windows.Controls;
static class MyMainClass {
    [STAThread]
    static void Main(string[] args) {
        Window window = new Window();
        window.Title = "Hello World";
        window.Show();
        Slider slider = new Slider();
        slider.Minimum = 0;
        slider.Maximum = 100;
        TextBox textBox = new TextBox();
        slider.ValueChanged += (o, e) => {
            textBox.Text = slider.Value.ToString();
        };
        textBox.TextChanged += (o, e) => {
            slider.Value = double.Parse(textBox.Text);
        };
        StackPanel panel = new StackPanel();
        panel.Children.Add(slider);
        panel.Children.Add(textBox);
        window.Content = panel;
        Application app = new Application();
        app.Run();
    }
}
```

A Slider and TextBox which display each others' values