Today’s Lecture
- C# Primer (Continued)
  - STL Equivalents
  - Properties
  - Delegates
- Example Code
  - Student List
  - Teaser
  - Balance Board / WPF

Reminders
- Blog Post (Week)
- C# Programming

Interfaces are not just software
**Interfaces - Remote**

![Remote Control Images]

**Small Group Exercise**

- Think of non-GUI interfaces
- Best remote?
- Worst remote?

Split into groups of 2-4 students

**Fun Link – Brief History Programming**


1970 - Guy Steele and Gerald Sussman create Scheme. Their work leads to a series of “Lambda the Ultimate” papers culminating in “Lambda the Ultimate Kitchen Utensil.” This paper becomes the basis for a long-running, but ultimately unsuccessful run of late night infomercials. Lambdas are relegated to relative obscurity until Java makes them popular by not having them.
1983 - Bjarne Stroustrup bolts everything he's ever heard of onto C to create C++. The resulting language is so complex that programs must be sent to the future to be compiled by the Skynet artificial intelligence. Build times suffer. Skynet's motives for performing the service remain unclear but spokespeople from the future say "there is nothing to be concerned about, baby," in an Austrian accented monotone. There is some speculation that Skynet is nothing more than a pretentious buffer overrun.

1996 - James Gosling invents Java. Java is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Sun loudly heralds Java's novelty.

2001 - Anders Hejlsberg invents C#. C# is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Microsoft loudly heralds C#'s novelty.

Simple student list

☐ Revisit the classic list of stuff
  ■ Student
    ☐ First name
    ☐ Last name
    ☐ GPA
    ☐ Major
  ■ Make a list of them and display it
☐ Homework 2
  ■ List and manipulate files

Pick up where we left off
Add a new class....

```c++
class Student
{
    private:
        String m_sFirstName;
        // ...

    public:
        Student ()
        {  ...  }

        void getFirstName ();
}
```

Class Design – C#

- Several changes
  - No header file
  - Per item ACL
    - Designate each item / function
    - Default is private
  - Similar modes
    - public, protected, private
  - Can also make “global”
    - static – do not need to instantiate
Big Change - Properties

- Normally do get / set functions
  - `getFirstName()`
  - `setFirstName()`

```csharp
class Student
{
    String m_sFirstName;

    public String FirstName
    {
        get { return m_sFirstName; }
        set { m_sFirstName = value; }
    }
}
```

Let's see that again....

- `public Type PropertyName`,
  - `get` // Stuff to get,
  - `set` // Stuff to set

- May leave off `set or set`,
- Return type must match property type
- `value` is the same as the property type

Using Properties

```csharp
Student theStudent;
theStudent = new Student();

// Old School
theStudent.FirstName = "Aaron";
Console.WriteLine(theStudent.FirstName);

// New school
theStudent.FirstName = "Aaron";
Console.WriteLine(theStudent.FirstName);
```

* just hidden
Automatic Properties

```csharp
public String FirstName
{
    get;
    set;
}
```

New feature in .NET 3.0
Automatically creates the variable underneath

Does it range check?

Let's see that again....

```csharp
public Type PropertyName
{
    get { // Stuff to get }
    set { // Stuff to set }
}
```

Instance.PropertyName

Where do we use this?

Real power is when we tie it to controls

Student.FirstName  TextBox.DataBindings

Any changes in the control automatically update our variable

WHOOT!
Populate the class

- See example code on wiki

Swapping over to Visual Studio

Override a function

```java
public override string ToString()
{
    string sTemp = FirstName + " " + LastName + " -> GPA: " + GPA + " -> Major: " + Major;
    return sTemp;
}
```

override -> virtual
Place in child

**toString exists in every class**
Part of base object

Code Example

```csharp
Student theStudent;
theStudent = new Student();
theStudent.FirstName = "John";
theStudent.LastName = "Doe";
theStudent.GPA = (float)3.67;
theStudent.Major = "Comp Sci";
```

**Properties:** FirstName, LastName, GPA, Major
Collections

- What is similar to STL Vector?
  - List - Collection

```csharp
List<Type> varName;
varName = new List<Type>();

List<Student> listStudents;
listStudents = new List<Student>();
```

Using the List type

```csharp
List<Student> listStudents;
listStudents = new List<Student>();

theStudent = new Student();
theStudent.firstName = "John";
theStudent.lastName = "Doe";
theStudent.sGA = (float)3.49;
theStudent.major = "CS 8";
theList.Add(theStudent);
theStudent = new Student();
theStudent.firstName = "Mary";
theStudent.lastName = "Smith";
theStudent.sGA = (float)3.70;
theStudent.major = "EE";
theList.Add(theStudent);
Console.WriteLine(theList[0]);
Console.WriteLine(theList[1]);
Console.WriteLine(theList);
```

Example output

- Add -> push_back
- Access like an array

```
John Doe -> GPA: 3.49 -> Major: CS 8
Mary Smith -> GPA: 3.70 -> Major: EE
Option.Collections.Generic.List<T>(StudentListing.Student)
```

If we want to have it iterate inside the list,
We have to make our own holder class that inherits or has an internal List class
Neat operator - foreach

```csharp
foreach (Type VarName in CollectionName)
{
    // Do whatever
}
```

```csharp
foreach (Student theSt in m_StudentList)
{
    Console.WriteLine(theSt);
}
```

Code commentary

```csharp
/// -> Triple slash
/* Similar to doxygen / javadoc */
/* Visual Studio will help auto-populate */

/// <summary>
/// A string representing the major for the
/// the particular student
/// </summary>
public String Major
{
    get { return m_Major; }
    set { m_Major = value; }
}
```

Writing out the student list

- Sketch out on the board

See example code on Wiki
Preview - WPF

Data binding in action

<Window Margin="0,1,1,1" Name="Window1"
    Grid.RowSpan="4" Margin="0,1,1,1" Name="listBox1"
    ItemsSource="{Binding Source={StaticResource StudentList}}"/>

<TextBox Name="textBox1" Margin="10,16,12,0"
    Text="{Binding ElementName=listBox1, Path=SelectedItem.FirstName}"/>

<TextBox Name="textBox2" Margin="10,16,12,0"
    Text="{Binding ElementName=listBox1, Path=SelectedItem.LastName}"/>

<TextBox Name="textBox3" Margin="10,0,12,3"
    Text="{Binding ElementName=listBox1, Path=SelectedItem.GPA}"/>

<Button Margin="13,12,18,10"
    Name="button1" Click="button1_Click">Load Info</Button>

How does it work?

Yikes!

WPF = WTF?

Key Part:

Text="{Binding ElementName=listBox1, Path=SelectedItem.FirstName}"
Questions?

- Homework 2
- Week 2 - Blog